МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ МІСЬКОГО ГОСПОДАРСТВА імені О. М. БЕКЕТОВА
Навчально-науковий Інститут енергетичної, інформаційної та транспортної інфраструктури
Кафедра автоматизації та комп'ютерно-інтегрованих технологій

РОЗРАХУНКОВО-ПОЯСНЮВАЛЬНА ЗАПИСКА ДО КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

на тему Проєктування та реалізація інформаційної системи обліку складських запасів з аналітичними функціями

Виконав: здобувач вищої освіти
4 курсу, групи Сінж-2022-1
напряму підготовки (спеціальності)
151 «Автоматизація та комп'ютерно-Інтегровані технології»
Сідак Денис Вячеславович
(прізвище та ініціали)

Керівник: Цегельник Є.В.
(прізвище та ініціали, наук. ступ., вч. звання)

Рецензент: Тимофєєв Олексій Володимирович. Директор.
(прізвище та ініціали, наук. ступ., вч. звання)

Харків – 2026

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ МІСЬКОГО ГОСПОДАРСТВА ІМЕНІ О. М. БЕКЕТОВА

Навчально-науковий інститут енергетичної, інформаційної та транспортної інфраструктури

Кафедра: автоматизації та комп'ютерно-інтегрованих технологій

Освітньо-кваліфікаційний рівень – бакалавр

Галузь знань: 15 – Автоматизація та приладобудування

Спеціальність: 151 – Автоматизація та комп'ютерно-інтегровані технології

ЗАТВЕРДЖУЮ

Завідувач кафедри АКІТ

_ __Баранов О.О.

« 19 » Червень 2026 року

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА

Сідак Денис Вячеславович

1.Тема роботи: Інформаційна система управління складськими запасами

Затверджена наказом університету від « 22 » травня 2026 року No 440-03

Керівник роботи Доцент кафедри АКІТ Цегельник Євген Володимирович

2. Строк подання роботи здобувачем вищої освіти: « 20 » Червня 2026 року

3. Вихідні дані до проекту: Вебсистема для управління складськими запасами, прогнозування попиту та оптимізації логістичних процесів.

4. Зміст розрахунково пояснювальної записки (перелік питань, які потрібно розробити): Вступ. Теоретичні основи управління запасами. Проєктування інформаційної системи управління запасами. Програмна реалізація системи. Вибір інструментів реалізації. Охорона праці. Заходи безпеки та зниження ризиків.

5. Перелік графічного матеріалу. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень); Презентація.

6. Консультанти роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
Аналіз проблеми	Цегельник Є.В.	11.05.2026	21.05.2026
Основна Частина	Цегельник Є.В.	22.05.2026	31.05.2026
Спеціальний розділ	Цегельник Є.В.	01.06.2026	17.06.2026
Охорона праці	Малишева В.В.	08.06.2026	15.06.2026

7. Дата видачі завдання: « 11 » травня 2026 року

Керівник _____Цегельник Є.В.

Завдання прийняв до виконання Сідак Д.В.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1.	Розробка 1го розділу бакалаврської роботи	11.05.2026 - 21.05.2026	
2.	Розробка 2го розділу бакалаврської роботи	22.05.2026 - 31.05.2026	
3.	Розробка 3го розділу бакалаврської роботи	01.06.2026 - 17.06.2026	
4.	Розробка розділу з охорони праці	08.06.2026 - 15.06.2026	
6.	Рецензування бакалаврської роботи	22.06.2026	

7.	Захист на ДЕК	23.06.2026	

Здобувача вищої освіти  Сідак Д.В.

Керівник _____ Цегельник Є.В.

РЕФЕРАТ

Розробка інформаційної системи управління складськими запасами – Сідак Денис Вячеславович, дипломна робота бакалавра, Харків, Харківський національний університет міського господарства імені О.М. Бекетова, кількість сторінок 95, кількість таблиць 11, кількість рисунків 36, кількість формул 4, кількість джерел літератури 29, кількість додатків 2.

Дипломна робота присвячена розробці інформаційної системи управління складськими запасами, призначеної для обліку товарів, контролю їх руху, аналізу залишків та прогнозування попиту. Основною метою роботи є створення програмного забезпечення, яке надає користувачу зручні та ефективні засоби ведення складського обліку, контролю відвантажень та аналізу стану запасів із використанням сучасних вебтехнологій React та Firebase.

ЦІЛЬ РОБОТИ – створення ефективної інформаційної системи для обліку складських запасів, контролю руху товарів, аналізу залишків і підтримки прийняття рішень щодо поповнення запасів.

ОБЄКТ ДОСЛІДЖЕННЯ: методи, технології, алгоритми та програмні засоби управління складськими запасами, обліку товарів, контролю руху продукції та прогнозування попиту.

ПРЕДМЕТ ДОСЛІДЖЕННЯ: Інформаційна система управління складськими запасами.

ЗАДАЧІ: проаналізувати особливості управління складськими запасами та існуючі програмні рішення у цій сфері. Розробити структуру інформаційної системи для обліку товарів, категорій, постачальників, місць зберігання, операцій руху товарів та їх відвантаження. Реалізувати інтуїтивний і зручний інтерфейс користувача для роботи із системою. Реалізувати функціональні модулі для додавання, редагування та видалення товарів, категорій і постачальників, а також для ведення обліку руху товарів на складі. Розробити засоби аналізу залишків, контролю мінімального рівня запасів і прогнозування

попиту на основі історії продажу. Провести тестування розробленої системи для перевірки її працездатності, надійності та ефективності.

МЕТОДИ ДОСЛІДЖЕННЯ: аналіз предметної області, порівняння існуючих програмних рішень, методи проєктування інформаційних систем, сучасні методи веброзробки, експериментальні методи дослідження, емпіричний аналіз результатів тестування.

В роботі проводиться проєктування, програмна реалізація та тестування інформаційної системи управління складськими запасами. Оцінка ефективності передбачає аналіз зручності ведення складського обліку, контролю руху товарів, виявлення дефіциту запасів і можливості прогнозування попиту на основі накопичених даних. Отримані результати підтверджують, що розроблена інформаційна система є ефективним засобом для автоматизації обліку складських запасів і підтримки основних логістичних процесів підприємства.

КЛЮЧОВІ СЛОВА: складські запаси, управління запасами, інформаційна система, складський облік, рух товарів, прогнозування попиту, React, Firebase.

ABSTRACT

Development of an Information System for Warehouse Inventory Management – Last Name, First Name, Middle Name, Bachelor's thesis, Kharkiv, O.M. Beketov National University of Urban Economy, number of pages 95, number of tables 11, number of figures 36, number of formulas 4, number of literature sources 29, number of appendices 2.

This thesis is devoted to the development of an inventory management information system designed to track goods, monitor their movement, analyze inventory levels, and forecast demand. The main objective of this work is to create software that provides users with convenient and effective tools for warehouse accounting, shipment control, and inventory analysis using modern web technologies such as React and Firebase.

PURPOSE – to create an effective information system for tracking inventory, monitoring the movement of goods, analyzing stock levels, and supporting decision-making regarding inventory replenishment.

OBJECT OF RESEARCH: methods, technologies, algorithms, and software tools for inventory management, inventory tracking, product flow control, and demand forecasting.

SUBJECT OF RESEARCH: Information system for warehouse inventory management.

OBJECTIVES: Analyze the specifics of warehouse inventory management and existing software solutions in this field. Develop the structure of an information system for tracking products, categories, suppliers, storage locations, inventory movements, and shipments. Implement an intuitive and user-friendly interface for working with the system. Implement functional modules for adding, editing, and deleting goods, categories, and suppliers, as well as for tracking goods movement in the warehouse. Develop tools for analyzing inventory levels, monitoring minimum stock levels, and forecasting demand based on sales history. Test the developed system to verify its functionality, reliability, and efficiency.

RESEARCH METHODS: domain analysis, comparison of existing software solutions, information system design methods, modern web development methods, experimental research methods, empirical analysis of test results.

The work presents the design, software implementation, and testing of an information system for warehouse inventory management. The evaluation of its effectiveness involves analyzing the ease of maintaining inventory records, monitoring the movement of goods, identifying inventory shortages, and the ability to forecast demand based on accumulated data. The results confirm that the developed information system is an effective tool for automating inventory accounting and supporting the enterprise’s core logistics processes.

KEYWORDS: inventory, inventory management, information system, inventory accounting, goods movement, demand forecasting, React, Firebase.

ЗМІСТ

ВСТУП

У сучасних умовах розвитку цифрових технологій ефективне управління складськими запасами набуває особливого значення для підприємств різних сфер діяльності. Склад є важливою частиною логістичної системи, оскільки саме тут зосереджуються процеси приймання, зберігання, переміщення та відвантаження товарів. Від правильності ведення обліку запасів, своєчасного контролю їх залишків і швидкості обробки складських операцій значною мірою залежить безперервність роботи підприємства, рівень обслуговування клієнтів та загальна економічна ефективність діяльності. Саме тому тема розробки інформаційної системи управління складськими запасами є актуальною та практично значущою.

У процесі діяльності підприємства постійно виникає потреба в точному контролі наявності товарів, їх руху, обліку постачальників, місць зберігання та операцій відвантаження. Використання застарілих або неавтоматизованих підходів до ведення складського обліку часто призводить до помилок, дублювання даних, втрати часу на пошук інформації та ускладнює прийняття управлінських рішень. Крім того, відсутність інструментів для аналізу залишків і прогнозування попиту може спричинити дефіцит окремих товарних позицій або, навпаки, надлишкове накопичення продукції на складі. Усе це негативно впливає як на внутрішні бізнес-процеси, так і на логістичну діяльність підприємства в цілому.

У зв’язку з цим виникає необхідність у створенні сучасної інформаційної системи, яка б забезпечувала централізований облік товарів, категорій, постачальників і місць зберігання, дозволяла фіксувати рух товарів на складі, оформлювати відвантаження, контролювати залишки та надавати користувачу засоби для аналізу складських процесів. Важливим аспектом такої системи є не лише автоматизація рутинних операцій, а й підтримка прийняття рішень шляхом виявлення критичних залишків, оцінки руху товарів та прогнозування попиту на основі накопичених даних. Саме це дозволяє

підвищити точність складського обліку, знизити ризик дефіциту продукції та покращити організацію внутрішніх логістичних процесів.

Розроблювана інформаційна система управління складськими запасами має на меті надати зручний програмний інструмент для роботи з основними складськими операціями. Система повинна забезпечувати можливість ведення довідників товарів, категорій і постачальників, фіксації операцій надходження, відвантаження, списання та переміщення товарів, а також формування аналітичних даних щодо стану запасів. Для реалізації системи доцільно використовувати сучасні вебтехнології, зокрема React для побудови зручного користувацького інтерфейсу та Firebase для організації зберігання даних і підтримки серверної логіки. Використання таких технологій дозволяє створити сучасний, доступний і функціональний програмний продукт, орієнтований на практичне застосування.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ УПРАВЛІННЯ ЗАПАСАМИ

1.1 Складські запаси та особливості їх обліку

Складські запаси є однією з ключових складових матеріальних ресурсів підприємства, які забезпечують безперервність виробничих, торговельних і логістичних процесів. Вони охоплюють товари, сировину, матеріали, комплектуючі, готову продукцію та інші цінності, що перебувають на зберіганні й використовуються або реалізуються в процесі господарської діяльності. Саме наявність достатнього обсягу запасів дозволяє підприємству своєчасно виконувати замовлення, підтримувати стабільність постачання, уникати перерв у роботі та реагувати на зміни попиту [1, с. 174] Водночас надмірне накопичення запасів також є небажаним, оскільки воно призводить до збільшення витрат на зберігання, ризику псування продукції, заморожування оборотних коштів і зниження загальної ефективності діяльності. У зв’язку з цим питання правильного обліку складських запасів має важливе значення як з економічної, так і з організаційної точки зору.

В широкому розумінні складські запаси можна розглядати як сукупність товарно-матеріальних цінностей, що перебувають у системі постачання, зберігання та реалізації. Вони є необхідним резервом, який забезпечує узгодження між надходженням продукції та її подальшим використанням або відвантаженням. На практиці складські запаси виконують декілька важливих функцій. Передусім вони гарантують безперервність діяльності підприємства в умовах нерівномірності постачання або змін попиту. Крім того, вони дозволяють формувати певний страховий резерв на випадок затримок поставок, коливань ринку чи інших непередбачених обставин. Запаси виступають об’єктом постійного контролю, оскільки будь-які помилки в їх обліку можуть призвести до фінансових втрат, порушення логістичних процесів та спотворення управлінської інформації [2].

Особливість складських запасів полягає в тому, що вони одночасно виступають як економічний ресурс і як об’єкт інформаційного обліку. З одного боку, це реальні матеріальні цінності, які мають фізичні характеристики, місце зберігання, кількісні та якісні параметри. З іншого боку, для ефективного управління ними необхідно вести точний облік надходження, переміщення, списання та залишків. Саме облік забезпечує формування достовірної інформації про фактичний стан запасів, їх рух у часі, зв’язок із постачальниками, категоріями товарів, місцями зберігання та документами відвантаження [3]. Без такої інформаційної підтримки управління складом стає фрагментарним і не дозволяє приймати обґрунтовані рішення.

У практиці діяльності підприємств складські запаси можуть бути різними за своїм призначенням, характером використання та способом обліку. Доцільно виділити декілька основних груп запасів, які найчастіше трапляються у сфері складського господарства [4]. Узагальнену характеристику цих запасів подано в таблиці 1.1.

Таблиця 1.1 – Основні види складських запасів та особливості їх обліку

Вид запасів	**Характеристика**	**Особливості обліку**
Сировина і матеріали	Використовуються у виробничому процесі для виготовлення продукції	Облік ведеться за кількістю, вартістю, датою надходження та постачальником
Товари	Призначені для подальшого продажу або відвантаження	Необхідний постійний контроль залишків, руху та цінових параметрів
Готова продукція	Завершений результат виробництва, підготовлений до реалізації	Облік пов’язаний із зберіганням, відвантаженням та контролем придатності
Комплектуючі	Окремі елементи, що входять до складу кінцевого продукту	Важливо контролювати залишки для недопущення зупинки виробництва
Страхові запаси	Резерв, який створюється на випадок затримок постачання або стрибка попиту	Потребують окремого контролю мінімального рівня та своєчасного поповнення

Як видно з таблиці 1.1, різні види запасів мають спільну облікову природу, однак можуть відрізнятися за роллю в діяльності підприємства та підходами до контролю. Для торговельних підприємств основною категорією є товари, що зберігаються на складі з метою подальшої реалізації або відвантаження. У виробничій сфері, навпаки, більшу увагу приділяють сировині, матеріалам і комплектуючим. Незалежно від галузі, спільною вимогою є забезпечення точного обліку кількості запасів, фіксації їх руху, контролю місця зберігання та своєчасного виявлення критичних залишків.

У контексті сучасних інформаційних систем облік складських запасів дедалі частіше поєднується з аналітичними функціями, з цього виходить, що система не лише зберігає інформацію про товари, а й дозволяє оцінювати динаміку залишків, визначати товари з найвищим обігом, аналізувати періоди підвищеного попиту та будувати прості прогнози на основі попередніх операцій. Такий підхід значно підвищує цінність складського обліку для управління підприємством. Якщо раніше дані про залишки використовувалися переважно для контролю наявності товарів, то сьогодні вони стають основою для прийняття рішень щодо закупівель, організації поставок і оптимізації роботи складу.

Для інформаційної системи управління складськими запасами особливе значення має правильне структурування даних обліку. Усі товари повинні бути описані за уніфікованими характеристиками, такими як назва, артикул, категорія, постачальник, одиниця виміру, закупівельна ціна, поточна кількість, мінімальний рівень запасу та місце зберігання. Операції руху товарів мають містити відомості про тип операції, кількість, дату, документ-основу та зв’язок із конкретним товаром. Окремо повинні фіксуватися документи відвантаження, які відображають передачу товару отримувачу та можуть бути використані як джерело даних для подальшого аналізу попиту. Така структура дозволяє не лише підтримувати точність обліку, а й створює основу для побудови функціональних модулів системи.

Складські запаси є важливим об’єктом управління та обліку, від стану якого залежить ефективність діяльності підприємства, стабільність логістичних процесів і якість обслуговування споживачів. Особливості їх обліку полягають у необхідності постійного контролю руху товарів, фіксації залишків, відображення місць зберігання, аналізу критичних рівнів запасів і забезпечення достовірності облікових даних. У сучасних умовах ці завдання доцільно реалізовувати за допомогою автоматизованих інформаційних систем, які дозволяють поєднати функції обліку, контролю та аналітики. Саме тому дослідження складських запасів і особливостей їх обліку є необхідною теоретичною основою для подальшого проєктування інформаційної системи управління запасами.

1.2 Методи контролю запасів і прогнозування попиту

Ефективне управління складськими запасами неможливе без застосування методів їх контролю та аналізу попиту. Якщо облік запасів дозволяє отримати інформацію про поточний стан товарів на складі, то контроль запасів і прогнозування попиту спрямовані на забезпечення оптимального рівня цих запасів у майбутньому. Саме завдяки таким методам підприємство може своєчасно поповнювати товарні позиції, уникати дефіциту, зменшувати надлишкове накопичення продукції та підтримувати безперервність логістичних процесів. У сучасних умовах ці завдання мають особливе значення, оскільки нестабільність попиту, зміни обсягів продажів і часові затримки поставок безпосередньо впливають на результативність роботи складу.

Контроль запасів являє собою сукупність організаційних і аналітичних дій, спрямованих на спостереження за станом товарів, виявлення відхилень від установлених норм та своєчасне прийняття рішень щодо поповнення або зменшення запасів. Основна мета такого контролю полягає в тому, щоб забезпечити наявність достатньої кількості товарів для задоволення попиту,

але водночас не допустити надмірного зберігання, яке призводить до додаткових витрат. Інакше кажучи, система контролю запасів повинна знаходити баланс між двома протилежними ризиками: нестачею товарів і перевантаженням складу.

Одним із базових методів контролю є встановлення мінімального та максимального рівнів запасів. Мінімальний рівень визначає критичну межу, нижче якої залишок товару не повинен опускатися, а максимальний рекомендований верхній обсяг, перевищення якого є небажаним. Це відношення є досить простим, однак на практиці воно виявляється ефективним, особливо в автоматизованих системах, де можна швидко порівнювати фактичний залишок із нормативними значеннями. Якщо поточний запас знижується до мінімального рівня, система може формувати сигнал про необхідність дозамовлення. Якщо кількість перевищує максимальний рівень, це може свідчити про низьку оборотність товару або помилку в плануванні закупівель.

Для оцінки потреби в поповненні запасів часто використовується розрахунок рекомендованої кількості замовлення. У найпростішому вигляді така величина може бути визначена за формулою:

$$Q_{rec} = Q_{max} - Q_{cur} \tag{1.1}$$

де:

Q_{rec} – рекомендована кількість замовлення;

Q_{max} – бажаний або максимальний рівень запасу;

Q_{cur} – поточний залишок товару на складі.

Ця формула показує, скільки одиниць товару необхідно замовити для досягнення бажаного рівня запасу. Її доцільно використовувати в тих випадках, коли підприємство вже визначило нормативний обсяг зберігання для конкретної позиції. Перевага такого підходу полягає в його простоті, однак він не враховує зміну інтенсивності попиту в часі. Саме тому в більш гнучких

системах управління запасами контроль залишків доповнюється аналізом середнього споживання товару.

Ще одним поширеним методом контролю є розрахунок середнього попиту за певний період, він дає змогу оцінити, скільки товару в середньому вибуває зі складу протягом дня, тижня або місяця. Його можна визначити за такою формулою [5]:

$$D_{avg} = \frac{\sum_{i=1}^{n} D_i}{n} \quad (1.2)$$

де:

D_{avg} – середній попит за період;

D_i – попит або обсяг вибуття товару в i-му періоді;

n – кількість періодів спостереження.

Практичне значення цієї формули полягає в тому, що вона дозволяє отримати усереднений показник споживання товару, на основі якого можна приймати рішення щодо частоти та обсягу поповнення запасів. Наприклад, якщо в середньому товар вибуває зі складу по 5 одиниць на день, а на складі залишається 20 одиниць, можна припустити, що за відсутності поповнення запасу вистачить приблизно на чотири дні, ці розрахунки допомагають для оперативного контролю й планування закупок.

Ще одним важливим методом контролю є аналіз оборотності запасів. Він дає змогу визначити, наскільки швидко товар проходить через склад, тобто як часто його запаси оновлюються протягом певного періоду. Висока оборотність свідчить про активний попит і правильну організацію запасів, тоді як низька може означати перевантаження складу малорухомими позиціями. У спрощеному вигляді коефіцієнт оборотності можна обчислити за формулою [6]:

$$K_{turn} = \frac{Q_{out}}{Q_{avg}} \quad (1.3)$$

де:

K_{turn} – коефіцієнт оборотності запасів;

Q_{out} – загальний обсяг вибуття товару за період;

Q_{avg} – середній залишок товару за той самий період.

Цей показник дозволяє порівнювати різні товарні позиції між собою та виділяти ті з них, які найбільше впливають на логістику підприємства. Товари з високою оборотністю потребують частішого контролю та точнішого прогнозування, тоді як для товарів із низькою оборотністю важливо не допустити накопичення надлишкових запасів.

Питання контролю запасів тісно пов'язане з прогнозуванням попиту. Якщо контроль дозволяє оцінити поточний стан запасу, то прогнозування дає змогу передбачити його зміну в майбутньому. Попит на товар може змінюватися під впливом багатьох чинників: сезонності, змін ринку, поведінки споживачів, цінової політики, маркетингових кампаній чи навіть зовнішніх економічних умов. У цьому зв'язку точне прогнозування попиту є складним завданням, однак навіть прості методи можуть дати корисний результат для системи управління запасами.

Одним із найпростіших і найпоширеніших методів прогнозування є метод середнього значення, за якого прогноз майбутнього попиту базується на середньому обсязі вибуття за попередні періоди. Точнішим за цей, є методом ковзне середнє, яке враховує лише останні кілька періодів і тим самим краще реагує на зміни. Формула ковзного середнього може бути подана так [7]:

$$F_{t+1} = \frac{D_t + D_{t-1} + \cdots + D_{t-m+1}}{m} \quad (1.4)$$

де:

F_{t+1} – прогноз на наступний період;

$D_t, D_{t-1}, \ldots$ – фактичні значення попиту в останніх періодах;

m – кількість останніх періодів, які враховуються.

Перевага цього методу в тому що він краще відображає актуальну динаміку попиту, особливо якщо споживання товару не є повністю стабільним. В контексті інформаційної системи управління складськими запасами метод ковзного середнього є досить зручним, оскільки його легко реалізувати на основі історії продажу або операцій вибуття товару.

В межах розроблюваної інформаційної системи доцільно використовувати не надто складні, але практично корисні методи контролю запасів і прогнозування попиту. До них належать контроль мінімального та максимального рівня запасів, розрахунок точки замовлення, визначення середнього попиту, оцінка тривалості наявного запасу та побудова простого прогнозу на основі історії вибуття товарів. Такі методи не потребують складних математичних моделей, але водночас забезпечують достатній рівень аналітичної підтримки для користувача системи.

1.3 Постановка задачі

Метою роботи є створення веборієнтованої інформаційної системи управління складськими запасами, яка забезпечуватиме зручний доступ до даних, автоматизацію облікових операцій, контроль руху товарів, аналіз залишків і підтримку прийняття рішень щодо поповнення запасів.

Для досягнення поставленої мети в межах роботи необхідно розв’язати такі основні завдання:

– Проаналізувати предметну область управління складськими запасами та визначити ключові процеси, які повинні бути підтримані в інформаційній системі;

– Проаналізувати існуючі програмні рішення у сфері складського обліку та визначити їх основні переваги й обмеження;

– Розробити структуру інформаційної системи управління складськими запасами з урахуванням особливостей одного складу та одного основного користувача системи;

– Спроєктувати структуру бази даних для зберігання інформації про товари, категорії, постачальників, місця зберігання, операції руху товарів, документи відвантаження та позиції відвантаження;

– Реалізувати авторизацію користувача для забезпечення доступу до функціоналу системи лише після входу в обліковий запис;

– Реалізувати модуль роботи з товарами, який має забезпечувати додавання, редагування, перегляд і видалення товарних позицій, а також збереження основних характеристик товару, зокрема назви, артикулу, категорії, постачальника, одиниці виміру, ціни, кількості, мінімального та максимального рівнів запасу і місця зберігання;

– Реалізувати модуль роботи з категоріями товарів, який забезпечуватиме їх створення, редагування, видалення та використання для групування продукції;

– Реалізувати модуль роботи з постачальниками, у межах якого повинна зберігатися інформація про назву постачальника, контактну особу, телефон, електронну пошту, адресу та додаткові примітки;

– Реалізувати модуль обліку руху товарів, що забезпечуватиме фіксацію таких операцій, як надходження, відвантаження, списання, повернення та переміщення, із зазначенням дати, кількості, типу операції, коментаря та документа-основи;

– Реалізувати модуль відвантаження товарів, у якому користувач зможе створювати документ відвантаження, вказувати отримувача, дату, статус документа, коментар, а також перелік товарів, що входять до відвантаження;

– Забезпечити автоматичне оновлення залишків товарів після підтвердження відповідних операцій руху або відвантаження;

– Реалізувати контроль мінімального рівня запасів і формування списку товарів, що потребують поповнення;

– Реалізувати засоби прогнозування попиту на основі історії вибуття або відвантаження товарів, зокрема розрахунок середнього попиту, оцінку тривалості наявного запасу та визначення рекомендованої кількості дозамовлення;

– Забезпечити наочне відображення аналітичної інформації, пов’язаної із залишками товарів, критичними позиціями та динамікою вибуття продукції;

– Розробити інтуїтивно зрозумілий користувацький інтерфейс, який забезпечуватиме просту взаємодію із системою та швидкий доступ до основних функцій;

– Забезпечити адаптивність інтерфейсу для коректної роботи системи як на персональних комп’ютерах, так і на мобільних пристроях;

– Провести тестування розробленої системи для перевірки правильності роботи її основних модулів, достовірності обліку даних, зручності використання та відповідності поставленим вимогам.

Реалізація такого програмного продукту дозволить підвищити точність складського обліку, спростити виконання основних операцій та забезпечити користувача інструментами для ефективнішого управління запасами.

1.4 Висновки до розділу

В першому розділі було розглянуто теоретичні основи управління складськими запасами та визначено їх значення в діяльності підприємства.

Проаналізовано основні методи контролю запасів і прогнозування попиту. Розглянуто підходи до визначення мінімального і максимального рівнів запасів, точки замовлення, середнього попиту, тривалості наявного запасу, а також прості методи прогнозування на основі історичних даних.

Було описано постановку задачі, яка полягає у розробці інформаційної системи управління складськими запасами з можливістю авторизації користувача, ведення обліку товарів, категорій, постачальників, місць

зберігання, операцій руху товарів і відвантажень, а також реалізації засобів контролю залишків, аналізу даних і прогнозування попиту.

Отримані результати першого розділу створюють теоретичну основу для подальшого проєктування структури системи, бази даних і програмної реалізації основних функціональних модулів.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Аналіз аналогів

Перед проєктуванням системи варто проаналізувати існуючі програмні рішення, які вже використовуються для автоматизації складського обліку, контролю запасів та оформлення складських операцій. Це дозволяє визначити найбільш поширені підходи та методи до реалізації подібних систем, оцінити їх функціональні можливості, виявити сильні та слабкі сторони, а також сформувати вимоги до власної системи.

Для аналізу було обрано чотири відомі програмні продукти, що належать до класу систем управління запасами та складськими процесами: Zoho Inventory, Odoo Inventory, inFlow Inventory та Sortly. Обрані рішення різняться за рівнем складності, глибиною функціоналу та цільовою аудиторією, що дозволяє провести більш об’єктивне порівняння.

Zoho Inventory є хмарною системою управління запасами, яка надає можливості обліку товарів, відстеження партій і серійних номерів, контролю залишків, отримання сповіщень про низький рівень запасів, а також формування звітності. Серед функцій сервісу також заявлено роботу зі штрихкодами, підтримку декількох складів і міжскладських переміщень [8].

Інтерфейс Zoho Inventory орієнтований на роботу через вебсередовище та побудований за класичною схемою бізнес-застосунків: меню навігації, панель керування, окремі модулі для товарів, замовлень, закупівель і звітів (рис. 2.1). Це рішення виглядає функціонально насиченим і придатним для підприємств, яким потрібен не лише складський облік, а й ширша інтеграція з бізнес-процесами.

Переваги Zoho Inventory [9]:

- широкий набір функцій для обліку товарів і запасів;
- підтримка штрихкодів, серій і партій;
- наявність сповіщень про низькі залишки;

– розвинена звітність;

– підтримка багатоскладської логіки.

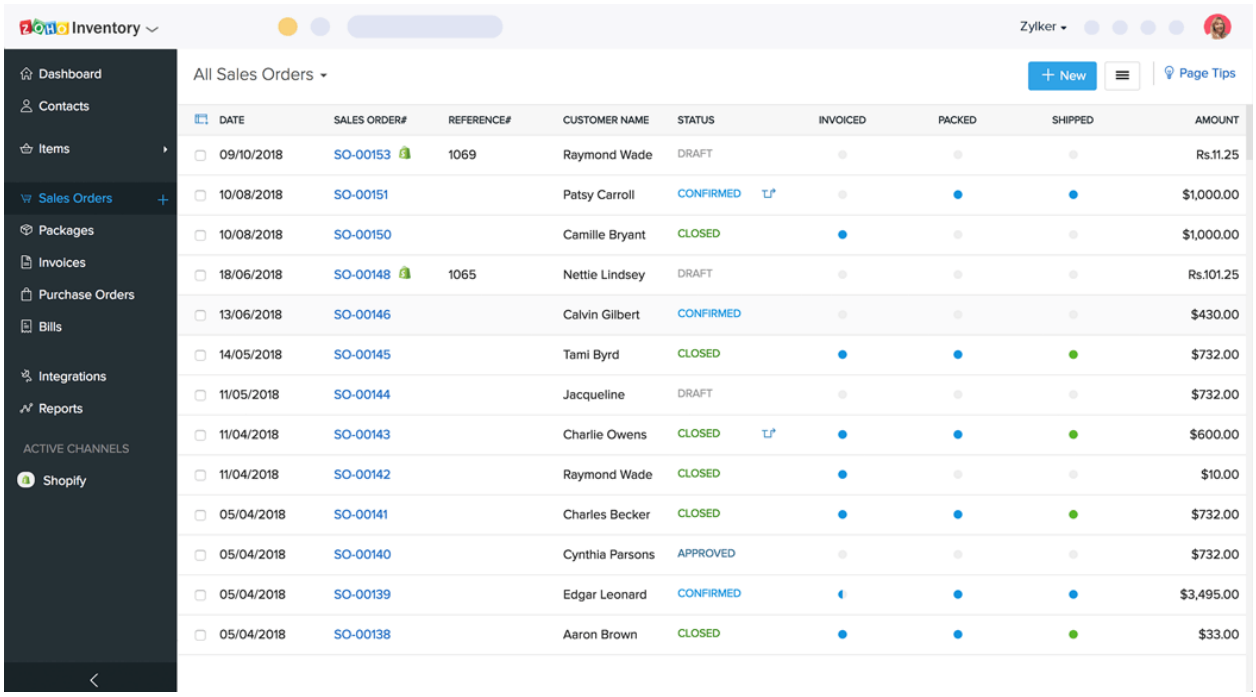


Рис. 2.1 – Інтерфейс Zoho Inventory

Недоліки Zoho Inventory [9]:

– значна частина функціоналу орієнтована на складніші бізнес-сценарії, ніж потрібні для спрощеної однокористувацької системи;

– інтерфейс містить багато модулів, що може ускладнювати швидке освоєння системи;

– для невеликого навчального проєкту частина можливостей є надлишковою. Це є висновком на основі аналізу заявленого функціоналу.

Odoo Inventory позиціонується як система управління запасами та складом, що підтримує керування строками постачання, автоматичне поповнення запасів, розширені маршрути, операції приймання та відвантаження, а також різні конфігурації складських процесів, зокрема одно- та двокрокові схеми приймання і доставки. Odoo також підтримує роботу зі штрихкодами та інструменти внутрішньоскладського переміщення [10].

Інтерфейс Odoo є модульним і побудований у стилі ERP-системи. Користувач працює з окремими застосунками, серед яких склад, закупівлі, продажі, виробництво та інші підсистеми (рис. 2.2).

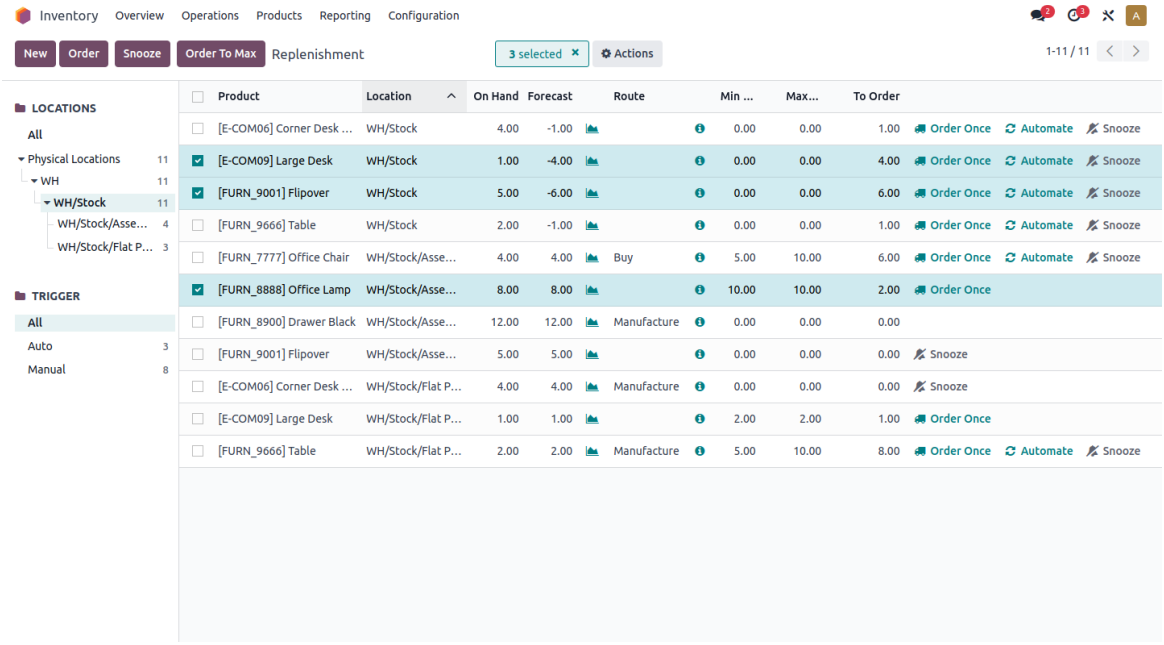


Рис. 2.2 – Інтерфейс Odoo Inventory

Переваги Odoo Inventory [11]:

– розвинений функціонал для управління складськими процесами;

– підтримка автоматичного поповнення запасів;

– можливість налаштування сценаріїв приймання та відвантаження;

– підтримка штрихкодів;

– придатність для комплексної автоматизації підприємства.

Недоліки Odoo Inventory [11]:

– система є складнішою за структурою та потребує більше часу на налаштування;

– ERP-підхід може бути надмірним для невеликої складської системи з одним складом;

– інтерфейс та логіка роботи більше підходять для багатомодульного корпоративного середовища. Це є узагальненим висновком на основі функціональної моделі сервісу.

inFlow Inventory є системою управління запасами, що орієнтується на спрощення контролю товарів, відстеження замовлень і оновлення залишків у реальному часі. Сервіс підтримує роботу із закупівлями, прийманням товару до запасів, звітністю, а також інструменти для обліку товарів по локаціях. На офіційному сайті також зазначено наявність понад 32 звітів, включно зі звітами про залишки, кількість і вартість товарів за локаціями [12].

Інтерфейс inFlow є більш прикладним і орієнтованим на щоденну операційну роботу (рис. 2.3).

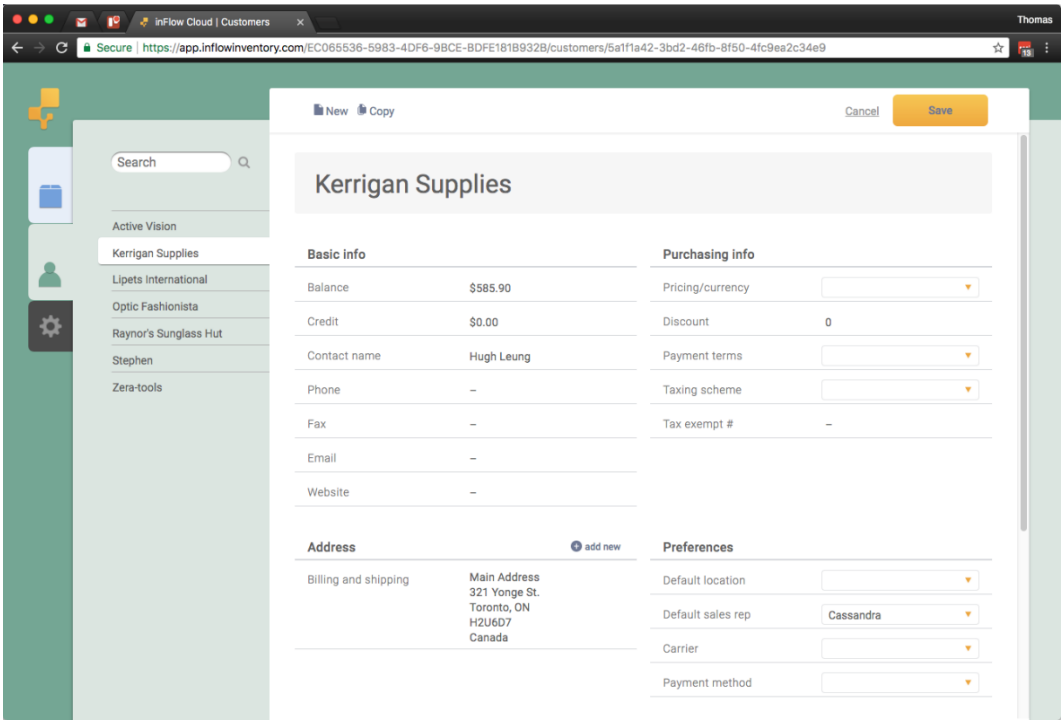


Рис. 2.3 – Інтерфейс inFlow

Значна увага приділяється товарним карткам, замовленням на закупівлю та продаж, а також звітам. Це робить систему ближчою до практичного складського використання без надмірної ERP-структури.

Переваги inFlow Inventory [13]:

– зручна орієнтація на товарний облік і замовлення;

– підтримка закупівель і приймання товару до запасів;

– наявність детальної звітності;

– актуалізація даних у реальному часі.

Недоліки inFlow Inventory [13]:

– частина можливостей орієнтована на повні бізнес-процеси закупівлі та продажу, а не лише на спрощений складський облік;

– для навчального вебпроєкту деякі функції можуть бути зайвими;

– система більше пристосована до комерційного використання, ніж до мінімалістичної однокористувацької реалізації. Це є аналітичним висновком на підставі заявлених функцій.

Sortly є системою інвентаризації, яка робить акцент на простоті використання, мобільному доступі, штрихкодах і QR-кодах, фотофіксації позицій, сповіщеннях про низькі залишки та звітності. На офіційних сторінках сервісу підкреслюється робота через смартфон, підтримка офлайн-доступу та зручне швидке оновлення інформації про товар [14].

Інтерфейс Sortly є простішим і більш візуальним порівняно з іншими розглянутими системами (рис. 2.4).

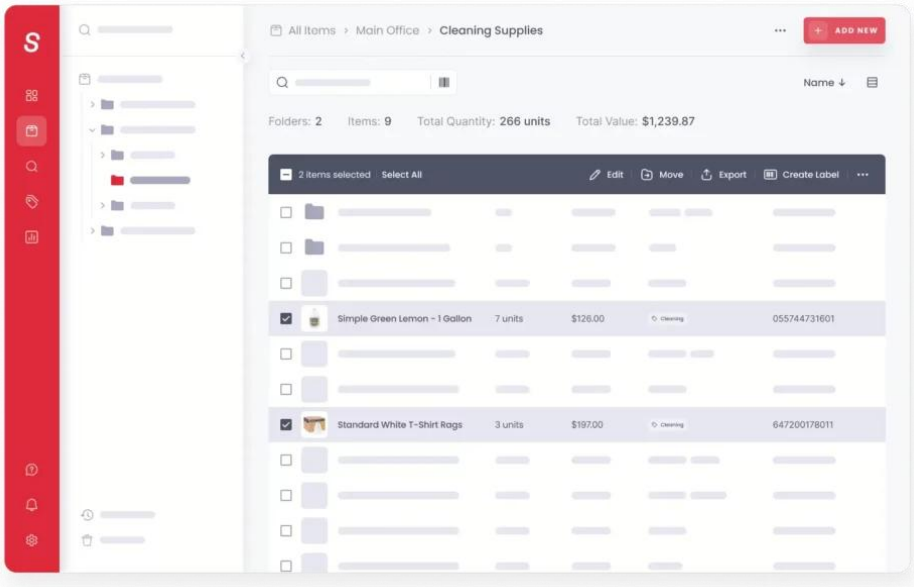


Рис. 2.4 – Інтерфейс Sortly

Великий акцент зроблено на мобільному використанні, швидкому скануванні, фотографіях і легкому доступі до основних операцій. Це робить систему зручною для невеликих складів або бізнесів, де потрібен швидкий облік без складної логіки документообігу.

Переваги Sortly [15]:

– простий і зрозумілий інтерфейс;
– сильна орієнтація на мобільні пристрої;
– підтримка штрихкодів і QR-кодів;
– наявність сповіщень про низькі залишки;
– зручність для швидкої інвентаризації.

Недоліки Sortly [15]:

– менш виражена документна логіка відвантажень і складських процесів порівняно з повноцінними складськими системами;
– більший акцент на простому інвентарному обліку, ніж на глибокому управлінні складом;
– для задачі аналізу руху товарів і побудови більш формалізованого журналу операцій можливостей може бути недостатньо. Останні два пункти є висновком на основі офіційно заявленого позиціонування сервісу.

Узагальнені результати порівняння аналогів наведено в таблиці 2.1.

Таблиця 2.1 – Порівняння аналогів систем управління запасами

Характеристика	**Zoho Inventory**	**Odoo Inventory**	**inFlow Inventory**	**Sortly**
Облік товарів	Так	Так	Так	Так
Відвантаження / документи	Так	Так	Так	Частково
Контроль залишків	Так	Так	Так	Так
Аналітика і звіти	Так	Так	Так	Так
Мобільність	Середня	Середня	Середня	Висока
Складність системи	Висока	Висока	Середня	Низька

Як видно з таблиці 2.1, усі розглянуті системи забезпечують базову підтримку товарного обліку та контролю залишків. Водночас Zoho Inventory та Odoo Inventory мають більш комплексний і багатофункціональний характер, що робить їх придатними для великих або багатоскладських сценаріїв, але надмірними для спрощеної системи з одним складом. inFlow Inventory є ближчим до практичного складського використання, однак також орієнтований на ширший набір бізнес-процесів. Sortly, навпаки, вирізняється простотою й мобільністю, однак поступається глибиною документного та процесного опрацювання.

Проведений аналіз дає підстави сформувати вимоги до власної розробки: система повинна забезпечувати базовий облік товарів, категорій і постачальників, підтримувати рух товарів і відвантаження, контролювати критичні залишки, надавати просту аналітику та при цьому мати інтуїтивний вебінтерфейс без надлишкової складності.

2.2 Проєктування архітектури системи

Під час проєктування інформаційної системи управління складськими запасами важливо визначити її загальну архітектуру, оскільки саме вона задає спосіб взаємодії між інтерфейсом користувача, програмною логікою та засобами зберігання даних. Для розроблюваної системи обрано трирівневу архітектуру вебзастосунку, яка є зрозумілою, зручною в реалізації та добре підходить для побудови сучасних інформаційних систем (рис. 2.5).

Перший рівень архітектури представлений клієнтським середовищем, у межах якого працює вебінтерфейс системи. Він реалізується у форматі React SPA-застосунку та відповідає за взаємодію користувача із системою. На цьому рівні розміщуються UI-компоненти, через які користувач працює з товарами, категоріями, постачальниками, рухом товарів, відвантаженнями й аналітикою. Також тут виконується керування станом інтерфейсу, що забезпечує оновлення даних на сторінках та коректну взаємодію між окремими модулями.

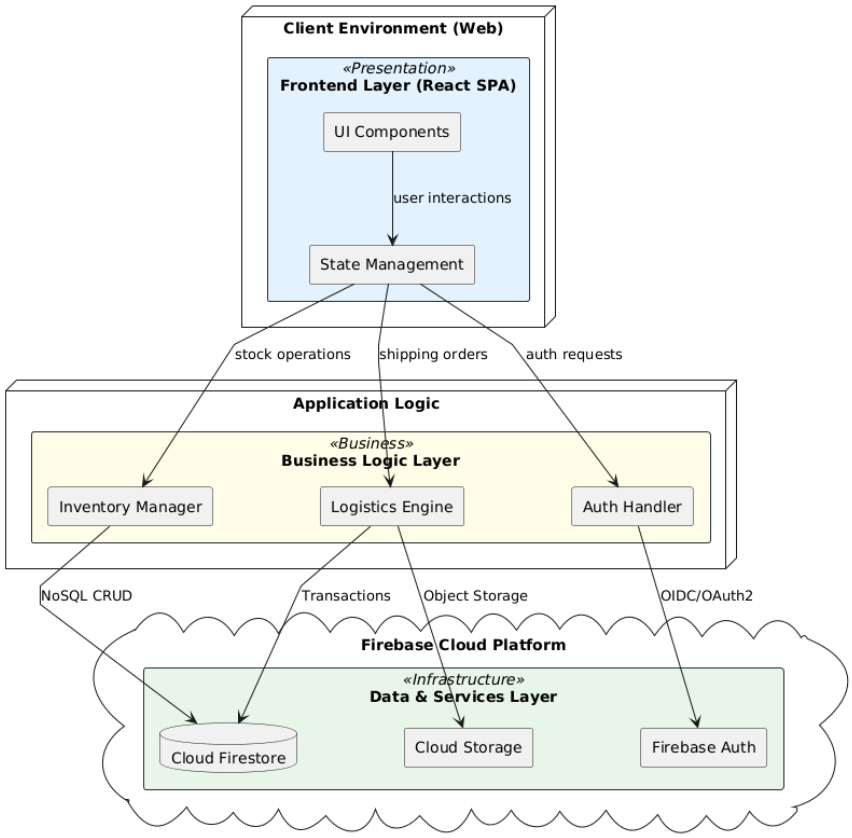


Рис. 2.5 – Діаграма розгортання системи

Другий рівень становить прикладна логіка, яка реалізує основні правила роботи системи. У межах цього рівня можна виділити три головні компоненти: Inventory Manager, Logistics Engine та Auth Handler. Компонент Inventory Manager відповідає за роботу з товарами, категоріями, постачальниками та залишками. Logistics Engine забезпечує обробку складських операцій, зокрема надходження, списання, повернення, переміщення та відвантаження товарів. Auth Handler реалізує логіку авторизації користувача та перевірки доступу до функціоналу системи.

Третій рівень архітектури представлений Firebase Cloud Platform, яка виконує роль середовища зберігання даних і сервісів. Для зберігання структурованої інформації використовується Cloud Firestore, у якому зберігаються дані про товари, категорії, постачальників, місця на складі, рух товарів та відвантаження. Для реалізації авторизації застосовується Firebase

Auth. За потреби для зберігання зображень або супровідних файлів може використовуватися Cloud Storage.

Взаємодія між рівнями здійснюється послідовно. Користувач працює з інтерфейсом, який передає запити до прикладної логіки. Далі відповідний модуль опрацьовує дію користувача та звертається до сервісів Firebase для читання, запису або оновлення даних. Наприклад, під час оформлення відвантаження система створює документ відвантаження, змінює залишок відповідного товару та формує запис у журналі руху товарів.

Обрана архітектура є доцільною для цієї системи, оскільки вона забезпечує чіткий розподіл функцій між інтерфейсом, логікою та даними.

2.3 Розробка бази даних та ключових сутностей

Під час проєктування інформаційної системи управління складськими запасами важливе значення має правильна організація структури бази даних. В межах розроблюваної системи було виділено декілька ключових колекцій, кожна з яких відповідає за окрему частину функціоналу.

Колекція users призначена для зберігання даних про зареєстрованих користувачів системи. Вона використовується для ідентифікації користувача, відображення його профілю та прив’язки інших документів до конкретного облікового запису.

Поля колекції users:

– createdAt, timestamp, дата і час створення облікового запису;

– email, string, електронна пошта користувача;

– fullName, string, повне ім’я користувача.

Колекція categories використовується для зберігання категорій товарів. Вона дозволяє групувати товари за типами, що спрощує навігацію, пошук і фільтрацію продукції в системі.

Поля колекції categories:

– createdAt, timestamp, дата і час створення категорії;

– name, string, назва категорії;

– userId, string, ідентифікатор користувача, якому належить категорія.

Колекція suppliers містить інформацію про постачальників товарів. Вона використовується для зв’язку товарів із джерелом постачання та для збереження контактних даних постачальників.

Поля колекції suppliers:

– address, string, адреса постачальника;

– companyType, string, тип організації, наприклад ТОВ або ФОП;

– contactPerson, string, контактна особа постачальника;

– createdAt, timestamp, дата і час створення запису;

– email, string, електронна пошта постачальника;

– name, string, назва постачальника;

– notes, string, додаткові примітки;

– phone, string, номер телефону;

– userId, string, ідентифікатор користувача, якому належить запис.

Колекція storageLocations призначена для зберігання інформації про місця на складі. У межах системи вона використовується для опису стелажів і комірок, у яких розміщуються товари.

Поля колекції storageLocations:

– cells, array, масив комірок, що належать до конкретного стелажа;

– cellsCount, int64, кількість комірок у стелажі;

– createdAt, timestamp, дата і час створення запису;

– name, string, назва або код стелажа;

– userId, string, ідентифікатор користувача, якому належить запис.

Окремо кожен елемент масиву cells має власну внутрішню структуру:

– code, string, код комірки;

– status, string, стан комірки, наприклад free або occupied.

Колекція products є основною в системі, оскільки саме в ній зберігається інформація про товари, їх характеристики, кількість, постачальника, категорію та місце розміщення на складі. Вона використовується як центральний довідник товарних позицій.

Поля колекції products:

- barcode, string, штрихкод товару;
- categoryId, string, ідентифікатор категорії;
- categoryName, string, назва категорії;
- cellCodes, array, список кодів комірок, у яких розміщений товар;
- createdAt, timestamp, дата і час створення товару;
- maxStock, int64, максимальний або бажаний запас товару;
- minStock, int64, мінімальний допустимий запас;
- name, string, назва товару;
- purchasePrice, int64, закупівельна ціна;
- quantity, int64, поточна кількість товару на складі;
- rackId, string, ідентифікатор стелажа;
- rackName, string, назва стелажа;
- salePrice, int64, відпускна або продажна ціна;
- sku, string, артикул товару;
- status, string, поточний статус товару;
- storageLocation, string, текстове представлення місця зберігання;
- supplierId, string, ідентифікатор постачальника;
- supplierName, string, назва постачальника;
- unit, string, одиниця виміру;
- userId, string, ідентифікатор користувача, якому належить товар.

Колекція stock_movements призначена для ведення журналу руху товарів. У ній фіксуються всі операції, які впливають на зміну залишків, зокрема надходження, списання, повернення, переміщення та інші подібні дії.

Поля колекції stock_movements:

– afterQty, int64, кількість товару після виконання операції;

– beforeQty, int64, кількість товару до виконання операції;

– createdAt, timestamp, дата і час створення запису;

– note, string, примітка до операції;

– productId, string, ідентифікатор товару;

– productName, string, назва товару;

– productSku, string, артикул товару;

– quantity, int64, кількість товару, яка бере участь в операції;

– type, string, технічний тип операції;

– typeLabel, string, текстове позначення типу операції;

– userId, string, ідентифікатор користувача, який виконав операцію.

Колекція shipments використовується для зберігання документів відвантаження. Вона містить інформацію про отримувача, список товарів, статус документа та інші параметри, пов’язані з передачею товарів клієнту або іншому об’єкту.

Поля колекції shipments:

– appliedToStock, boolean, ознака того, чи було вже застосовано зміни до залишків товару;

– createdAt, timestamp, дата і час створення документа;

– customerName, string, ім’я або назва отримувача;

– customerPhone, string, контактний номер телефону отримувача;

– invoiceNumber, string, номер накладної;

– items, array, список товарних позицій у документі;

– note, string, додатковий коментар до відвантаження;

– status, string, технічний статус документа;

– statusLabel, string, текстове позначення статусу;

– userId, string, ідентифікатор користувача, якому належить документ.

Кожен елемент масиву items у колекції shipments має таку структуру:

– currentQuantity, int64, поточний залишок товару на момент оформлення;

– productId, string, ідентифікатор товару;

– productName, string, назва товару;

– productSku, string, артикул товару;

– quantity, int64, кількість товару у відвантаженні;

– unit, string, одиниця виміру товару.

Запропонована структура бази даних охоплює всі основні процеси, які реалізуються в інформаційній системі управління складськими запасами. Для її графічного зображення можна використати логічну діаграму (Додаток А).

Така організація даних є зручною для реалізації в середовищі Cloud Firestore, оскільки дозволяє гнучко працювати з окремими колекціями, зменшує складність запитів і забезпечує достатню масштабованість системи. Крім того, вона добре відповідає функціональним вимогам розроблюваного вебзастосунку та створює надійну основу для подальшої програмної реалізації.

2.4 Висновки до розділу

В другому розділі було виконано проєктування інформаційної системи управління складськими запасами. На початку розділу проведено аналіз існуючих аналогів, що дозволило визначити їх основні функціональні можливості, переваги та недоліки, а також сформувати вимоги до розроблюваної системи. На основі цього було обґрунтовано доцільність створення власного вебзастосунку зі спрощеною, але достатньою для поставленої задачі функціональністю.

У межах розділу також було спроєктовано загальну архітектуру системи, яка базується на трирівневій моделі та включає клієнтський рівень, рівень прикладної логіки та рівень даних і сервісів. Визначено, що для реалізації системи доцільно використовувати React для побудови інтерфейсу

користувача та Firebase для організації автентифікації, зберігання даних і підтримки основних сервісів. Такий підхід забезпечує логічну структуру системи, зручність реалізації та можливість подальшого розширення функціоналу.

Розроблено структуру бази даних та визначено ключові сутності системи, зокрема користувачів, категорії, постачальників, місця на складі, товари, рух товарів і відвантаження. Запропонована структура даних охоплює всі основні процеси складського обліку та створює надійну основу для подальшої програмної реалізації системи в межах наступного розділу.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Вибір інструментів реалізації

Для програмної реалізації інформаційної системи управління складськими запасами було обрано сучасні інструменти веброзробки, які забезпечують зручність створення інтерфейсу, роботу з даними, побудову візуалізацій та розгортання готового застосунку. Вибір саме такого набору технологій зумовлений потребою створити адаптивний, функціональний і зручний у використанні вебзастосунок, який можна швидко реалізувати та підтримувати.

Як основне середовище розробки було використано Visual Studio Code. Це середовище є одним із найпоширеніших у сучасній веброзробці завдяки зручному інтерфейсу, підтримці великої кількості розширень, інтеграції з системами контролю версій і комфортній роботі з проєктами на JavaScript [16]. Використання Visual Studio Code дозволяє ефективно працювати з компонентною структурою React-застосунку, швидко редагувати код, виконувати перевірку синтаксису та організовувати структуру проєкту.

Для створення клієнтської частини системи було обрано React. Ця бібліотека дозволяє будувати інтерфейс у вигляді незалежних компонентів, що значно спрощує реалізацію сторінок, форм, таблиць і повторно використовуваних елементів. React добре підходить для створення односторінкових вебзастосунків, у яких важливими є динамічне оновлення даних, зручна навігація між сторінками та швидка реакція інтерфейсу на дії користувача [17]. У контексті цієї системи React використовується для реалізації сторінок товарів, категорій, постачальників, руху товарів, відвантажень, аналітики та профілю користувача.

Для стилізації інтерфейсу було використано Tailwind CSS. Його застосування дає змогу швидко створювати сучасний адаптивний дизайн без написання великої кількості окремих CSS-файлів. Перевагою цього

інструмента є можливість безпосередньо в коді задавати стилі для компонентів, що спрощує верстку і прискорює розробку [18]. Tailwind CSS особливо зручний для побудови адаптивного інтерфейсу, який коректно працює як на персональних комп’ютерах, так і на мобільних пристроях.

Для відображення графіків та аналітичних даних було обрано бібліотеку Recharts. Вона дозволяє створювати зрозумілі та наочні діаграми на основі даних, отриманих із системи [19]. У межах розроблюваного застосунку Recharts доцільно використовувати для візуалізації залишків товарів, динаміки відвантажень, критичних позицій та інших показників, пов’язаних із аналітикою складських запасів. Ця бібліотека добре інтегрується з React і дозволяє будувати графіки без надмірного ускладнення коду.

Для сповіщень, повідомлень про успішність операцій і попереджень було використано SweetAlert2. Цей інструмент дозволяє реалізувати більш зручні та візуально привабливі діалогові вікна порівняно зі стандартними браузерними повідомленнями [20]. Його використання є доцільним для підтвердження видалення записів, повідомлень про успішне додавання товару, помилки введення даних або попередження про критичний рівень запасу.

Для підключення іконок в інтерфейсі застосунку було використано React Icons [21]. Ця бібліотека містить велику кількість готових іконок, які можуть бути використані в кнопках, меню, картках та інших елементах інтерфейсу.

Для організації серверної частини, автентифікації та зберігання даних було використано Firebase. Ця хмарна платформа є зручною для реалізації навчальних і прикладних вебпроєктів, оскільки надає готові сервіси для роботи з користувачами та базою даних без необхідності розгортання окремого бекенд-сервера [22]. Вибір Firebase дозволяє суттєво спростити програмну реалізацію системи та зосередитися на створенні основного функціоналу.

Для реалізації входу користувача в систему було використано Firebase Authentication, цей сервіс забезпечує створення облікових записів, вхід до системи та контроль доступу до захищених сторінок застосунку [23]. У межах

даної системи Firebase Authentication використовується для авторизації користувача перед початком роботи з товарами, відвантаженнями та іншими модулями.

Для зберігання основних даних системи було використано Cloud Firestore, це документно-орієнтована база даних, яка дозволяє організовувати інформацію у вигляді колекцій і документів [24]. Firestore добре підходить для зберігання даних про користувачів, категорії, постачальників, товари, стелажі, рух товарів та відвантаження. Його перевагою є гнучкість структури, зручність інтеграції з React і можливість швидко реалізовувати основні CRUD-операції.

Для зберігання вихідного коду та контролю змін у проєкті було використано GitHub. Ця платформа дозволяє вести історію розробки, зберігати код у віддаленому репозиторії та за потреби повертатися до попередніх версій проєкту [25]. Використання GitHub є доцільним не лише з точки зору розробки, а й для подальшого розгортання вебзастосунку.

Для публікації та хостингу клієнтської частини системи було обрано Netlify. Цей сервіс дозволяє швидко розгорнути React-застосунок, забезпечити доступ до нього через мережу Інтернет та автоматизувати оновлення після внесення змін у GitHub-репозиторій [26]. Використання Netlify є зручним рішенням для розміщення готового вебінтерфейсу системи без складних налаштувань серверної інфраструктури.

Сукупність цих засобів є достатньою для створення повноцінного вебзастосунку, що відповідає поставленим функціональним вимогам.

3.2 Реалізація основних функціональних модулів системи

Одним із базових функціональних модулів розроблюваної системи є модуль автентифікації користувача. Його призначення полягає в забезпеченні безпечного доступу до функціоналу системи лише після успішного входу або створення нового облікового запису. Реалізація форм входу та реєстрації

виконана у вигляді окремих React-компонентів. Нижче наведено фрагмент коду, який реалізує основну логіку входу та реєстрації користувача:

```
      const handleSubmit = async (e) => {
        e.preventDefault();
        setError("");
        if (!form.fullName.trim() || !form.email.trim() ||
!form.password.trim()) {
          setError("Будь ласка, заповніть усі поля");
          return;
        }
        if (form.password.trim().length < 6) {
          setError("Пароль повинен містити щонайменше 6 символів");
          return;
        }
        const userCredential = await createUserWithEmailAndPassword(
          auth,
          form.email.trim(),
          form.password.trim()
        );
        const user = userCredential.user;
        await updateProfile(user, {
          displayName: form.fullName.trim(),
        });
        await setDoc(doc(db, "users", user.uid), {
          fullName: form.fullName.trim(),
          email: form.email.trim(),
          createdAt: serverTimestamp(),
        });
      };
```

У наведеному фрагменті реалізовано логіку реєстрації нового користувача. Спочатку форма перевіряється на наявність порожніх полів, а також контролюється мінімальна довжина пароля. Після цього за допомогою функції createUserWithEmailAndPassword() створюється обліковий запис у Firebase Authentication. Далі виконується оновлення профілю користувача через updateProfile(), що дозволяє зберегти його ім’я у властивості displayName. Завершальним етапом є запис додаткової інформації до колекції users, де зберігаються повне ім’я, електронна пошта та дата реєстрації. Логіка входу реалізована схожим чином, але є простішою, оскільки не передбачає створення нового документа в базі даних. Перед надсиланням запиту система перевіряє, чи користувач заповнив електронну пошту та пароль. Якщо валідація проходить успішно, викликається функція signInWithEmailAndPassword(), яка виконує перевірку облікових даних у Firebase Authentication. У разі успішного входу користувач автоматично

перенаправляється на головну сторінку системи. Якщо ж дані введено неправильно, у формі відображається повідомлення про помилку.

Важливим моментом реалізації є те, що обидва компоненти використовують локальний стан через useState(). Це дозволяє зручно зберігати введені значення форми та текст помилки, а також оперативно оновлювати інтерфейс у відповідь на дії користувача. Додатково для переходу між сторінками застосовано useNavigate() та компонент Link, що забезпечує зручну навігацію між формами входу і реєстрації.

Після успішного входу користувач потрапляє на домашню сторінку системи, яка виконує роль інформаційної панелі. Її основне призначення полягає в тому, щоб надати швидкий доступ до ключових показників складу, відобразити останні події та забезпечити зручну навігацію до основних розділів. На цій сторінці зібрано узагальнену інформацію про товари, запаси, відвантаження, рух товарів, критичні залишки та стан комірок складу.

Нижче наведено фрагмент коду, який відповідає за отримання основних даних для формування домашньої сторінки:

```
      const fetchData = useCallback(async () => {
        if (!user) return;

        setLoading(true);

        try {
          const [
            productsSnapshot,
            categoriesSnapshot,
            suppliersSnapshot,
            shipmentsSnapshot,
            movementsSnapshot,
            storageSnapshot,
          ] = await Promise.all([
            getDocs(query(collection(db, "products"), where("userId", "==",
user.uid))),
            getDocs(query(collection(db, "categories"), where("userId", "==",
user.uid))),
            getDocs(query(collection(db, "suppliers"), where("userId", "==",
user.uid))),
            getDocs(query(collection(db, "shipments"), where("userId", "==",
user.uid))),
            getDocs(query(collection(db, "stock_movements"), where("userId",
"==", user.uid))),
            getDocs(query(collection(db, "storageLocations"), where("userId",
"==", user.uid))),
          ]);
```

У цьому фрагменті реалізовано одночасне отримання даних із кількох основних розділів системи. Це дозволяє завантажити товари, категорії, постачальників, накладні, журнал руху та місця зберігання в межах одного запиту на оновлення сторінки. Після цього отримані записи перетворюються у зручний формат і зберігаються у стані компонента для подальшого відображення.

Важливою частиною домашньої сторінки є блок статистики. Для цього на основі завантажених даних обчислюються основні показники, які відображаються у вигляді інформаційних карток:

```
const dashboardStats = useMemo(() => {
  const totalProducts = products.length;
  const totalCategories = categories.length;
  const totalSuppliers = suppliers.length;

  const totalQuantity = products.reduce(
    (sum, item) => sum + (Number(item.quantity) || 0),
    0,
  );

  const lowStockCount = products.filter(
    (item) => (Number(item.quantity) || 0) <= (Number(item.minStock) ||
0),
  ).length;

  const activeShipments = shipments.filter(
    (item) => item.status === "new" || item.status === "confirmed",
  ).length;
```

У наведеному коді розраховуються ключові зведені показники: кількість товарів, категорій і постачальників, сумарний обсяг запасів, число товарів із критично низьким залишком, а також кількість активних відвантажень. Саме ці значення використовуються на початку сторінки для швидкого ознайомлення зі станом складу.

Окрему роль на домашній сторінці виконує блок швидких переходів до основних розділів системи:

```
const quickLinks = [
  {
    title: "Товари",
    text: "Перегляд і керування всіма товарами",
    to: "/products",
  },
  {
    title: "Категорії",
```

```
    text: "Структура товарів за групами",
    to: "/categories",
  },
  {
    title: "Постачальники",
    text: "Контакти та дані постачальників",
    to: "/suppliers",
  },
  {
    title: "Рух товарів",
    text: "Операції та журнал змін",
    to: "/inventory-movements",
  },
];
```

Цей фрагмент показує, що домашня сторінка використовується не лише як інформаційна панель, а і як центральна точка навігації. Користувач може швидко перейти до керування товарами, категоріями, постачальниками, рухом товарів, відвантаженнями чи аналітикою. Завдяки цьому скорочується кількість зайвих дій у системі та покращується загальна зручність користування.

Для відображення активності складу реалізовано окремий аналітичний блок, у якому дані журналу руху групуються за датами:

```
const activityChartData = useMemo(() => {
  const grouped = {};

  movements.forEach((item) => {
    const key = formatShortDate(item.createdAt);

    if (!grouped[key]) {
      grouped[key] = {
        date: key,
        incoming: 0,
        outgoing: 0,
        transfer: 0,
      };
    }

    const qty = Number(item.quantity) || 0;

    if (item.type === "incoming") grouped[key].incoming += qty;
    if (item.type === "outgoing") grouped[key].outgoing += qty;
    if (item.type === "transfer") grouped[key].transfer += qty;
  });

  return Object.values(grouped).slice(-10);
}, [movements]);
```

У цьому фрагменті всі записи журналу групуються за короткою датою, після чого для кожного дня окремо підраховуються надходження, відвантаження та переміщення. У результаті формується масив даних, який

надалі використовується для побудови графіка активності складу. Така візуалізація дозволяє оцінити динаміку роботи системи та побачити, як змінювався рух товарів упродовж останніх днів.

Інтерфейс домашньої сторінки побудовано у вигляді окремих змістових блоків: вітального сектора, карток статистики, графіка активності, блоку швидких дій, списку останніх операцій, накладних, критичних товарів і загального стану системи. Таке компонування дозволяє логічно розділити інформацію та зробити сторінку зручною як для роботи на комп'ютері, так і на мобільних пристроях.

Сторінки для роботи з категоріями, постачальниками та складом мають подібну логіку, оскільки всі вони належать до довідкових модулів системи. Їх основне призначення полягає у створенні, редагуванні, пошуку та видаленні довідкових записів, які надалі використовуються в інших частинах системи. Саме тому ці модулі доцільно розглядати разом, зосереджуючи увагу не на повторюваних операціях додавання чи відображення даних, а на важливих програмних рішеннях, що забезпечують цілісність структури системи.

Одним із таких рішень є паралельне завантаження пов'язаних даних. Наприклад, на сторінці категорій одночасно завантажуються як самі категорії, так і товари, що належать користувачу:

```
const [categoriesSnapshot, productsSnapshot] = await Promise.all([
  getDocs(categoriesQuery),
  getDocs(productsQuery),
]);

const categoriesData = categoriesSnapshot.docs.map((item) => ({
  id: item.id,
  ...item.data(),
}));

const productsData = productsSnapshot.docs.map((item) => ({
  id: item.id,
  ...item.data(),
}));

setCategories(categoriesData);
setProducts(productsData);
```

Це дозволяє не лише відобразити список категорій, а й одразу показати, скільки товарів належить до кожної з них. Завдяки цьому сторінка стає

інформативнішою, а користувач отримує додатковий контекст без окремих переходів до сторінки товарів. Аналогічний підхід використано і для сторінки місць зберігання, де разом із переліком стелажів завантажуються товари для визначення зайнятості комірок.

Ще одним важливим рішенням є забезпечення узгодженості пов'язаних даних під час редагування категорій. Якщо назва категорії змінюється, це оновлення повинно бути відображене і в пов'язаних товарах:

```
await updateDoc(doc(db, "categories", category.id), {
  name: newName,
});

const relatedProducts = products.filter(
  (product) => product.categoryId === category.id
);

if (relatedProducts.length > 0) {
  const batch = writeBatch(db);

  relatedProducts.forEach((product) => {
    batch.update(doc(db, "products", product.id), {
      categoryName: newName,
    });
  });

  await batch.commit();
}
```

У цьому фрагменті реалізовано каскадне оновлення даних: після зміни назви категорії автоматично оновлюється поле categoryName у всіх товарах, що пов'язані з нею. Це дозволяє зберегти логічну цілісність системи й уникнути ситуації, коли в довіднику категорія вже змінена, а в товарних позиціях ще відображається стара назва.

Схожий підхід застосовано й при видаленні категорії. У цьому випадку видаляється не лише сам запис категорії, а й усі товари, які до неї належать:

```
const batch = writeBatch(db);

const relatedProducts = products.filter(
  (product) => product.categoryId === category.id
);

relatedProducts.forEach((product) => {
  batch.delete(doc(db, "products", product.id));
});

batch.delete(doc(db, "categories", category.id));
```

```
      await batch.commit();
```

Таке рішення дозволяє уникнути появи ізольованих товарів без категорії. Воно також спрощує подальшу роботу з даними, оскільки після видалення категорії система не зберігає непов’язані записи, що могли б створювати помилки в інтерфейсі або аналітиці.

На сторінці постачальників важливим є рішення щодо редагування записів через діалогове вікно. Замість перенесення користувача на окрему сторінку редагування використовується компактне модальне вікно з попередньо заповненими полями:

```
      const { value: values } = await Swal.fire({
        title: "Редагування постачальника",
        html: `
          <input id="swal-name" class="swal2-input" value="${supplier.name ||
""}">
          <input id="swal-contactPerson" class="swal2-input"
value="${supplier.contactPerson || ""}">
          <input id="swal-phone" class="swal2-input" value="${supplier.phone
|| ""}">
          <input id="swal-email" class="swal2-input" value="${supplier.email
|| ""}">
        `,
        preConfirm: () => {
          const updated = {
            name: document.getElementById("swal-name").value.trim(),
            contactPerson: document.getElementById("swal-
contactPerson").value.trim(),
            phone: document.getElementById("swal-phone").value.trim(),
            email: document.getElementById("swal-email").value.trim(),
          };
          return updated;
        },
      });
```

Перевага такого підходу полягає в тому, що користувач редагує запис без втрати контексту поточної сторінки, це пришвидшує роботу з довідником постачальників і робить керування записами більш зручним. Окремо варто відзначити, що перед збереженням виконується перевірка обов’язкових полів і дублювання назв, що зменшує ризик помилкового введення даних.

Для сторінки місць зберігання було реалізовано окреме програмне рішення, яке автоматизує створення комірок у стелажі. Користувач вводить лише назву стелажа та кількість комірок, після чого система самостійно формує їх коди:

```
const getRackPrefix = (name) => {
  const firstLetter = name.trim().charAt(0).toUpperCase();
  return firstLetter || "A";
};

const buildCells = (rackName, count) => {
  const prefix = getRackPrefix(rackName);

  return Array.from({ length: count }, (_, index) => ({
    code: `${prefix}${index + 1}`,
    status: "free",
  }));
};
```

Завдяки цьому користувачеві не потрібно вручну створювати кожну комірку окремо. Така автоматизація значно спрощує початкове налаштування складу та забезпечує єдиний підхід до формування кодів комірок. Наприклад, для стелажа з назвою G система автоматично створить комірки G1, G2, G3 і так далі.

Ще одним важливим моментом для модуля складу є автоматичне визначення того, чи є комірка вільною або зайнятою. Це відбувається на основі аналізу товарів, які вже прив’язані до певного стелажа та комірок:

```
const applyOccupiedStatus = useCallback((rackList, productList) => {
  return rackList.map((rack) => {
    const updatedCells = (rack.cells || []).map((cell) => {
      const occupied = productList.some(
        (product) =>
          product.rackId === rack.id &&
          Array.isArray(product.cellCodes) &&
          product.cellCodes.includes(cell.code),
      );

      return {
        ...cell,
        status: occupied ? "occupied" : "free",
      };
    });

    return { ...rack, cells: updatedCells };
  });
}, []);
```

У результаті статус комірки не задається вручну, а визначається автоматично відповідно до фактичного розміщення товарів. Це дозволяє підтримувати актуальний стан складу та зменшує кількість ручних дій під час адміністрування стелажів і комірок.

Модуль роботи з товарами є центральною частиною системи, оскільки саме через нього формується основний складський каталог. Оскільки частина базових дій, таких як завантаження записів, додавання, видалення та фільтрація, уже була розглянута в попередніх підрозділах, у цьому модулі доцільно зосередитися на тих рішеннях, які безпосередньо пов’язані зі специфікою товарів. Насамперед це перевірка готовності довідкових даних, підвантаження категорій, постачальників і стелажів для вибору, а також контроль зайнятості комірок під час додавання товару.

Одним із важливих рішень є попередня перевірка, чи створив користувач необхідні записи для повноцінного додавання товару. Для цього в модулі аналізується наявність категорій, постачальників і місць зберігання.

```
      const prerequisitesReady =
        categories.length > 0 && suppliers.length > 0 && racks.length > 0;

      {!prerequisitesReady && (
        <div className="rounded-[1.5rem] border border-amber-200 bg-amber-50
px-5 py-5 mb-6">
          <h3 className="text-xl font-bold text-gray-900 mb-2">
            Недостатньо даних для додавання товару
          </h3>
          <p className="text-gray-700 leading-relaxed">
            Щоб додати товар, необхідно створити по одному запису для:
            Категорій, Постачальників та Місць зберігання.
          </p>
        </div>
      )}
```

У цьому фрагменті реалізовано перевірку готовності довідкових даних. Якщо хоча б один із потрібних довідників ще не заповнений, користувач отримує попередження, а форма додавання товару не використовується повноцінно. Це рішення дозволяє уникнути помилок, коли товар створюється без категорії, постачальника або місця на складі.

Іншим важливим моментом є підвантаження пов’язаних записів, які використовуються у формі як дані для вибору:

```
      const [
        productsSnapshot,
        categoriesSnapshot,
        suppliersSnapshot,
        racksSnapshot,
      ] = await Promise.all([
```

```
        getDocs(query(collection(db, "products"), where("userId", "==",
user.uid))),
        getDocs(query(collection(db, "categories"), where("userId", "==",
user.uid))),
        getDocs(query(collection(db, "suppliers"), where("userId", "==",
user.uid))),
        getDocs(query(collection(db, "storageLocations"), where("userId",
"==", user.uid))),
      ]);
```

У наведеному коді одночасно отримуються товари, категорії, постачальники та стелажі. Це дозволяє не лише сформувати список товарів, а й одразу підготувати значення для випадаючих списків у формі. Завдяки цьому користувач може вибрати категорію, постачальника та місце зберігання зі вже створених записів, не вводячи ці значення вручну. Окремо в модулі реалізовано логіку вибору стелажа й комірок. Після зміни стелажа список обраних комірок очищується, а доступні комірки обчислюються на основі поточного вибору:

```
const selectedRack = useMemo(() => {
  return racks.find((rack) => rack.id === form.rackId) || null;
}, [racks, form.rackId]);

const availableCells = useMemo(() => {
  if (!selectedRack) return [];
  return selectedRack.cells || [];
}, [selectedRack]);

if (name === "rackId") {
  setForm((prev) => ({
    ...prev,
    rackId: value,
    cellCodes: [],
  }));
  return;
}
```

Це потрібно для того, щоб користувач не міг випадково зберегти товар із комірками, які належать іншому стелажу. Крім того, він забезпечує коректне оновлення інтерфейсу: після вибору нового стелажа система автоматично показує лише ті комірки, які пов’язані саме з ним. Перед додаванням товару реалізовано кілька перевірок, серед яких контроль обов’язкових полів, числових значень та унікальності артикулу:

```
if (
  !payload.name ||
  !payload.sku ||
```

```
  !payload.categoryId ||
  !payload.supplierId ||
  !payload.unit ||
  !payload.rackId ||
  payload.cellCodes.length === 0
) {
  toast("error", "Заповніть усі обов’язкові поля");
  return;
}

const skuExists = products.some(
  (item) => item.sku?.toLowerCase() === payload.sku.toLowerCase(),
);

if (skuExists) {
  toast("error", "Товар з таким артикулом уже існує");
  return;
}
```

У цьому фрагменті контролюється, щоб товар не був збережений із порожніми ключовими полями, а також щоб у системі не з’явилися дублікати за артикулом. Такий контроль є важливим саме для товарного модуля, оскільки артикул використовується як один із головних ідентифікаторів товарної позиції.

Ще одним важливим рішенням є перевірка фактичної зайнятості обраних комірок перед записом товару та одночасне оновлення стану стелажа:

```
const busySelected = payload.cellCodes.some((code) => {
  const cell = latestCells.find((item) => item.code === code);
  return !cell || cell.status === "occupied";
});

if (busySelected) {
  toast("error", "Одна або кілька обраних комірок уже зайняті");
  fetchData();
  return;
}

const updatedCells = latestCells.map((cell) =>
  payload.cellCodes.includes(cell.code)
    ? { ...cell, status: "occupied" }
    : cell,
);
```

Цей фрагмент забезпечує захист від подвійного використання тих самих комірок. Якщо одна з обраних комірок уже зайнята, система не дозволяє додати товар і оновлює дані сторінки. Якщо ж перевірка проходить успішно, для вибраних комірок відразу встановлюється стан occupied. Це дозволяє підтримувати актуальну інформацію про розміщення товарів на складі.

Під час видалення товару також передбачено важливе пов’язане оновлення, звільнення зайнятих комірок:

```
if (
  product.rackId &&
  Array.isArray(product.cellCodes) &&
  product.cellCodes.length > 0
) {
  const updatedCells = (rackData.cells || []).map((cell) =>
    product.cellCodes.includes(cell.code)
      ? { ...cell, status: "free" }
      : cell,
  );

  batch.update(rackRef, {
    cells: updatedCells,
  });
}
```

Це рішення дає змогу зберегти актуальну структуру складу після видалення товару. Якщо б комірки не звільнялися автоматично, система поступово накопичувала б некоректні дані про зайнятість місць зберігання. Саме тому видалення товару в цьому модулі пов’язане не лише з вилученням самого запису, а й з оновленням стану складу.

Модулі руху товарів і відвантаження є ключовими для всієї системи, оскільки саме вони забезпечують зміну фактичних залишків, фіксацію складських операцій і формування документів передачі товару. Хоча ці сторінки мають різне призначення, вони тісно пов’язані між собою: сторінка руху товарів відповідає за внутрішні складські операції, а сторінка відвантаження за створення накладних і списання товару після підтвердження відправки.

На сторінці руху товарів реалізовано кілька типів операцій: надходження, списання, повернення та переміщення. Для цього використовується окремий набір кнопок, який дозволяє швидко переключатися між типами операцій:

```
const operationLabels = {
  incoming: "Надходження",
  writeoff: "Списання",
  return: "Повернення",
  transfer: "Переміщення",
};
```

```
{Object.entries(operationLabels).map(([key, label]) => (
  <button
    key={key}
    type="button"
    onClick={() => {
      setActiveOperation(key);
      resetForm();
    }}
  >
    {label}
  </button>
))}
```

У цьому фрагменті видно, що сторінка не дублює окремі форми для кожної операції, а працює через один спільний інтерфейс. Після зміни типу операції форма очищується, що дозволяє уникнути перенесення даних між різними сценаріями роботи. Важливою частиною модуля є формування запису в журналі руху товарів. Для цього використано окрему допоміжну функцію, яка створює уніфікований запис для будь-якого типу операції:

```
const createMovementRecord = async ({
  batch,
  product,
  type,
  quantity,
  note,
  beforeQty,
  afterQty,
  extra = {},
}) => {
  const movementRef = doc(collection(db, "stock_movements"));

  batch.set(movementRef, {
    productId: product.id,
    productName: product.name,
    productSku: product.sku,
    type,
    typeLabel: operationLabels[type],
    quantity,
    beforeQty,
    afterQty,
    note: note || "",
    createdAt: serverTimestamp(),
    ...extra,
  });
};
```

Таке рішення є зручним тим, що незалежно від типу операції система формує запис однакової структури. Завдяки цьому журнал руху товарів має єдиний формат, а в подальшому його можна легко використовувати для пошуку, фільтрації або аналітики. Для операцій надходження, списання та повернення реалізовано просту, але важливу логіку перерахунку залишку.

Наприклад, при списанні попередньо перевіряється, чи достатньо товару на складі:

```
if (activeOperation === "writeoff") {
  if (quantity > currentQty) {
    toast("error", "Недостатня кількість товару для списання");
    return;
  }

  const nextQty = currentQty - quantity;

  batch.update(productRef, {
    quantity: nextQty,
  });

  await createMovementRecord({
    batch,
    product,
    type: "writeoff",
    quantity,
    note: form.note,
    beforeQty: currentQty,
    afterQty: nextQty,
  });
}
```

У цьому фрагменті видно, що перед виконанням списання перевіряється допустимість операції. Якщо користувач намагається списати більше товару, ніж є в наявності, система не дозволяє виконати дію. Після успішної перевірки оновлюється кількість товару та створюється відповідний запис у журналі.

Окремої уваги потребує операція переміщення, оскільки вона змінює не кількість товару, а його місце розміщення на складі. Для цього користувач обирає новий стелаж і комірки, після чого система перевіряє, чи не зайняті вони іншим товаром:

```
const busySelected = form.targetCellCodes.some((code) => {
  const occupiedByAnother = products.some(
    (item) =>
      item.id !== product.id &&
      item.rackId === form.targetRackId &&
      Array.isArray(item.cellCodes) &&
      item.cellCodes.includes(code)
  );

  const exists = latestCells.some((item) => item.code === code);
  return !exists || occupiedByAnother;
});

if (busySelected) {
  toast("error", "Одна або кілька обраних комірок зайняті");
  return;
}
```

Це рішення не дозволяє користувачу перемістити товар у вже зайняті комірки. Далі система звільняє попередні комірки товару, оновлює новий стелаж і записує факт переміщення в журнал. Таким чином забезпечується узгодженість між журналом операцій і фактичним станом складу.

Модуль відвантаження реалізовано у вигляді сторінки для створення накладних, формування їх складу та зміни статусу документа. На першому етапі користувач вводить номер накладної, одержувача, телефон і коментар, а потім додає товари до складу документа:

```
if (!invoiceNumber || !customerName) {
  toast("error", "Заповніть номер накладної та одержувача");
  return;
}

if (invoiceItems.length === 0) {
  toast("error", "Додайте хоча б один товар у накладну");
  return;
}

const exists = shipments.some(
  (item) =>
    item.invoiceNumber?.toLowerCase() === invoiceNumber.toLowerCase(),
);

if (exists) {
  toast("error", "Накладна з таким номером уже існує");
  return;
}
```

У наведеному фрагменті виконується базова перевірка накладної перед збереженням: контролюється заповнення ключових полів, наявність хоча б однієї позиції та унікальність номера документа. Це дозволяє уникнути появи неповних або дубльованих відвантажень у системі. Для додавання товарів у накладну реалізовано окремий проміжний список позицій. Під час додавання кожної позиції система перевіряє, чи не перевищує вказана кількість поточний залишок товару:

```
const quantity = Number(itemForm.quantity);

if (quantity > Number(product.quantity || 0)) {
  toast("error", "Не можна додати більше товару, ніж є на складі");
  return;
}
```

```
      const exists = invoiceItems.some((item) => item.productId ===
product.id);

      if (exists) {
        toast("error", "Цей товар вже доданий до накладної");
        return;
      }
```

Тут реалізовано відразу два важливі обмеження: користувач не може додати до накладної більше товару, ніж є на складі, а також не може повторно додати ту саму позицію в межах одного документа. Це спрощує подальшу обробку накладної та зменшує ризик помилок.

Ключовим етапом у модулі відвантаження є зміна статусу накладної на «відвантажено». Саме в цей момент система фактично змінює залишки товарів і створює записи в журналі руху:

```
      const beforeQty = Number(product.quantity) || 0;
      const afterQty = beforeQty - Number(line.quantity);

      batch.update(productRef, {
        quantity: afterQty,
      });

      const movementRef = doc(collection(db, "stock_movements"));

      batch.set(movementRef, {
        productId: product.id,
        productName: product.name,
        productSku: product.sku,
        type: "outgoing",
        typeLabel: "Відвантаження",
        quantity: Number(line.quantity),
        beforeQty,
        afterQty,
        note: `Накладна ${shipment.invoiceNumber}`,
        shipmentId: shipment.id,
        shipmentNumber: shipment.invoiceNumber,
        createdAt: serverTimestamp(),
      });
```

Цей фрагмент є центральним для зв’язку між модулями відвантаження та руху товарів. Після відвантаження кожна позиція накладної зменшує кількість відповідного товару, а в журналі створюється запис типу outgoing. Таким чином відвантаження не існує окремо від загального руху товарів, а автоматично відображається в загальній історії складських операцій.

Перед підтвердженням відвантаження реалізовано також додатковий контроль: якщо після списання деякі товари опустяться нижче мінімального залишку, система показує попередження та просить підтвердити дію:

```
      if (belowMinimum.length > 0) {
        const warningText = belowMinimum
          .map(
            (item) =>
              `${item.productName}: залишок буде ${item.afterQty}, мінімум
${item.minStock}`,
          )
          .join("<br>");

        const confirmMinimum = await Swal.fire({
          title: "Увага до залишків",
          html: `Після відвантаження деякі товари опустяться нижче
мінімального залишку:<br><br>${warningText}<br><br>Продовжити?`,
          icon: "warning",
          showCancelButton: true,
        });

        if (!confirmMinimum.isConfirmed) return;
      }
```

Це робить процес відвантаження більш контрольованим. Система не просто виконує списання, а попереджає користувача про можливі наслідки для складських запасів. Завдяки цьому користувач може вчасно звернути увагу на дефіцитні позиції.

Сторінка аналітики працює на основі даних, що вже накопичені в системі: товарів, категорій, постачальників, накладних, журналу руху та місць зберігання. Спочатку ці дані завантажуються, після чого на їх основі формуються узагальнені показники:

```
      const kpis = useMemo(() => {
        const totalProducts = products.length;
        const totalCategories = categories.length;
        const totalSuppliers = suppliers.length;
        const totalShipments = shipments.length;

        const totalQuantity = products.reduce(
          (sum, item) => sum + (Number(item.quantity) || 0),
          0,
        );

        const lowStockCount = products.filter(
          (item) => (Number(item.quantity) || 0) <= (Number(item.minStock) ||
0),
        ).length;

        return {
          totalProducts,
```

```
      totalCategories,
      totalSuppliers,
      totalShipments,
      totalQuantity,
      lowStockCount,
    };
  }, [products, categories, suppliers, shipments]);
```

У цьому фрагменті обчислюються основні зведені показники: кількість товарів, категорій, постачальників, накладних, сумарний обсяг запасів і число товарів із низьким залишком. Саме ці значення використовуються як базові індикатори стану системи та відображаються у верхній частині сторінки.

Для аналізу руху товарів окремо формується набір даних, згрупований за датами та типами операцій:

```
const movementChartData = useMemo(() => {
  const grouped = {};

  movements.forEach((movement) => {
    const key = formatDate(movement.createdAt) || "—";

    if (!grouped[key]) {
      grouped[key] = {
        date: key,
        incoming: 0,
        writeoff: 0,
        return: 0,
        transfer: 0,
        outgoing: 0,
      };
    }
```

Тут записи журналу групуються за датою, після чого для кожного дня окремо накопичуються надходження, списання, повернення, переміщення та відвантаження. У результаті формується структура, придатна для побудови графіка активності складу. Це дає змогу побачити, які типи операцій переважали в різні періоди роботи системи.

На сторінці також реалізовано механізм оцінки попиту, для цього аналізуються операції відвантаження, а далі на їх основі формується базовий прогноз на найближчі сім днів:

```
    const outgoingMovements = movements.filter((item) => item.type ===
"outgoing");

    outgoingMovements.forEach((item) => {
      if (!perProduct[item.productId]) {
        perProduct[item.productId] = {
```

```
            productId: item.productId,
            name: item.productName,
            totalOutgoing: 0,
            count: 0,
          };
        }

        perProduct[item.productId].totalOutgoing += Number(item.quantity) ||
0;
        perProduct[item.productId].count += 1;
      });
```

У цьому фрагменті враховуються лише операції типу outgoing, тобто фактичні відвантаження. Для кожного товару накопичується загальна кількість вибуття та число відповідних операцій. Надалі на цій основі обчислюється середній попит і прогноз споживання на сім днів. Це є спрощеною моделлю, однак вона дозволяє виявляти товари з підвищеним ризиком дефіциту.

Окрему практичну цінність має блок критичних залишків. У ньому система не лише показує товари, що опустилися нижче мінімального рівня, а й розраховує рекомендований обсяг поповнення:

```
      const lowStockProducts = useMemo(() => {
        return products
          .filter((item) => (Number(item.quantity) || 0) <=
(Number(item.minStock) || 0))
          .map((item) => ({
            id: item.id,
            name: item.name,
            quantity: Number(item.quantity) || 0,
            minStock: Number(item.minStock) || 0,
            maxStock: Number(item.maxStock) || 0,
            recommendedOrder:
              Math.max(
                (Number(item.maxStock) || 0) - (Number(item.quantity) || 0),
                0,
              ) || 0,
          }));
      }, [products]);
```

Аналітичний модуль не обмежується візуалізацією статистики, а й надає користувачу підказки для прийняття рішень. У випадку критичних позицій система відразу показує орієнтовну кількість, яку доцільно замовити для відновлення запасу до бажаного рівня.

Сторінка профілю користувача реалізує три основні функції: зміну імені, зміну пароля та видалення профілю. При цьому форма зміни імені синхронізує оновлення з записом користувача в базі даних:

```
const saveName = async (e) => {
  e.preventDefault();

  const fullName = nameForm.fullName.trim();

  if (!fullName) {
    toast("error", "Введіть нове ім’я");
    return;
  }

  await updateProfile(user, { displayName: fullName });
  await updateDoc(doc(db, "users", user.uid), { fullName });

  toast("success", "Ім’я успішно змінено");
};
```

У наведеному фрагменті показано, що після перевірки введеного значення виконується оновлення імені в профілі користувача та в його записі. Завдяки цьому зміни стають узгодженими для всієї системи, а нове ім’я надалі може відображатися на домашній сторінці та в інших частинах інтерфейсу.

Зміна пароля реалізована окремо й передбачає перевірку поточного пароля перед встановленням нового:

```
const credential = EmailAuthProvider.credential(
  user.email,
  currentPassword,
);

await reauthenticateWithCredential(user, credential);
await updatePassword(user, newPassword);

setPasswordForm({
  currentPassword: "",
  newPassword: "",
});
```

Тут спочатку формується обліковий запис-підтвердження на основі поточного пароля, після чого виконується повторна перевірка користувача. Лише після цього система дозволяє змінити пароль. Такий підхід підвищує безпеку модуля профілю та зменшує ризик несанкціонованої зміни пароля.

Найскладнішою частиною профілю є видалення облікового запису, оскільки ця дія повинна охоплювати не лише сам профіль, а й усі пов’язані записи користувача:

```
      const collectionsToCheck = [
        { name: "products", fields: ["userId", "ownerId", "createdBy"] },
        { name: "categories", fields: ["userId", "ownerId", "createdBy"] },
        { name: "suppliers", fields: ["userId", "ownerId", "createdBy"] },
        { name: "storageLocations", fields: ["userId", "ownerId", "createdBy"]
},
        { name: "shipments", fields: ["userId", "ownerId", "createdBy"] },
      ];

      for (const item of collectionsToCheck) {
        for (const field of item.fields) {
          const q = query(collection(db, item.name), where(field, "==", uid));
          const snapshot = await getDocs(q);

          snapshot.forEach((document) => {
            batch.delete(document.ref);
          });
        }
      }
```

У цьому рішенні система проходить по кількох ключових колекціях і шукає записи, пов’язані з поточним користувачем. Після цього вони видаляються пакетно. У результаті забезпечується цілісність даних і очищення пов’язаних сутностей.

3.3 Тестування системи та демонстрація отриманих результатів

Тестування буде проводитись в ручному режимі безпосередньо в браузері, оскільки вебзастосунок було опубліковано в мережі Інтернет і він доступний для повноцінної практичної перевірки. Такий підхід дозволяє оцінити коректність роботи основних модулів, зручність інтерфейсу та загальну відповідність системи поставленим вимогам.

При переході до системи користувач потрапляє на головну сторінку (рис. 3.1). У цій частині розміщено фонове зображення складу, назву системи StockFlow, короткий опис її призначення та дві основні кнопки для початку роботи: реєстрації та входу до кабінету.

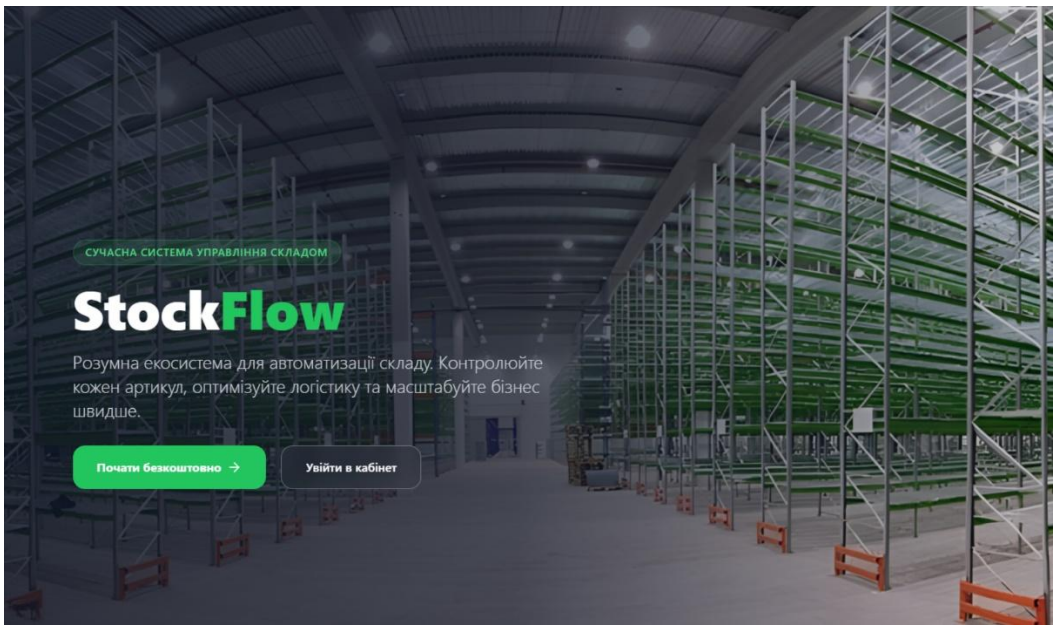


Рис. 3.1 – Частина головної сторінки

Після початкового блоку головної сторінки користувач може ознайомитися з основними перевагами системи, які наведені у вигляді окремих інформаційних карток (рис. 3.2).

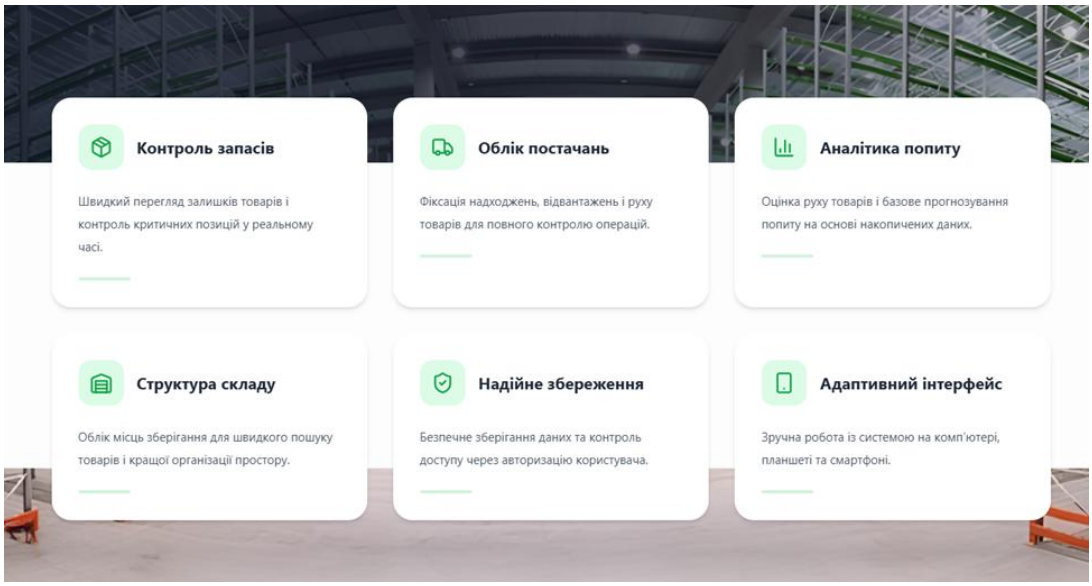


Рис. 3.2 – Друга частина головної сторінки

В цій частині головної сторінки відображено ключові можливості вебзастосунку StockFlow. Кожна картка містить іконку, коротку назву функції та стислий опис її призначення.

Нижче на головній сторінці розміщено додатковий блок із переліком основних модулів системи та закликом до початку роботи (рис. 3.3).

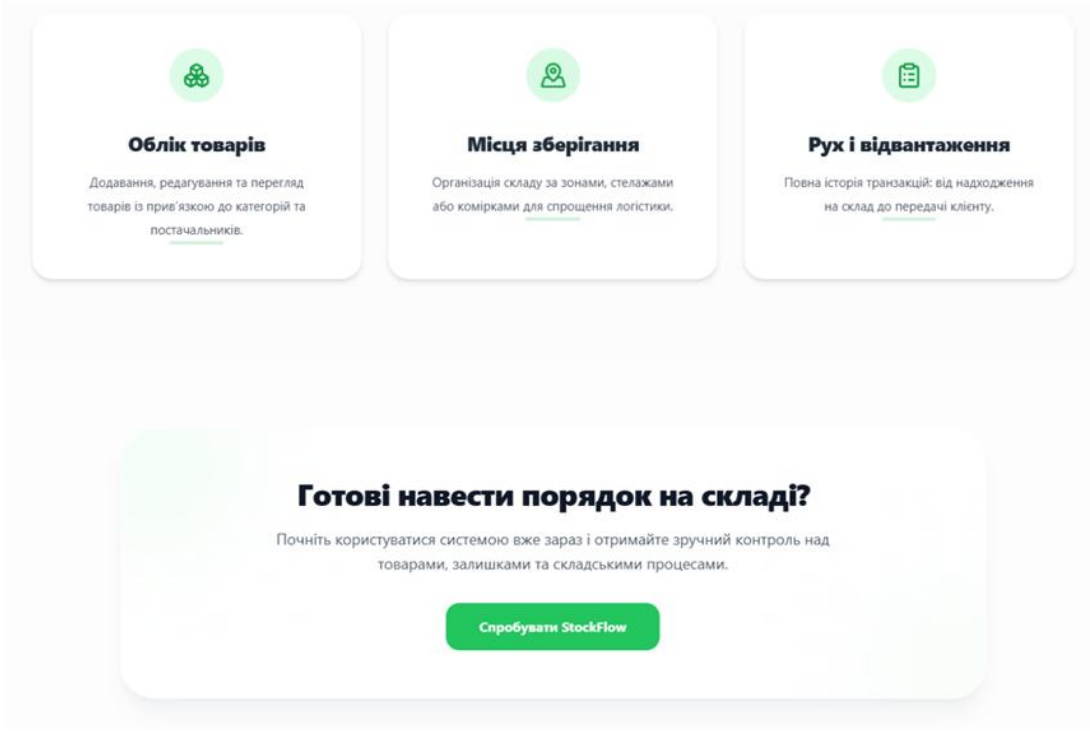


Рис. 3.3 – Кінець головної сторінки

У нижній частині головної сторінки зображено три основні напрями роботи системи: облік товарів, місця зберігання, а також рухи і відвантаження. Після цього розміщено окремий інформаційний блок із текстом «Готові навести порядок на складі?» та кнопкою «Спробувати StockFlow». Цей елемент завершує головну сторінку та мотивує користувача перейти до використання системи.

Для авторизованого доступу до функцій вебзастосунку користувач може перейти до форми входу, яка наведена на рисунку 3.4.

Форма входу виконана у мінімалістичному стилі та містить поля для введення електронної пошти й пароля. У верхній частині розміщено іконку входу та заголовок «Вхід до системи», що чітко вказує на призначення сторінки. Після заповнення полів користувач натискає кнопку «Увійти».

Також передбачено посилання для переходу до реєстрації, якщо користувач ще не має облікового запису.

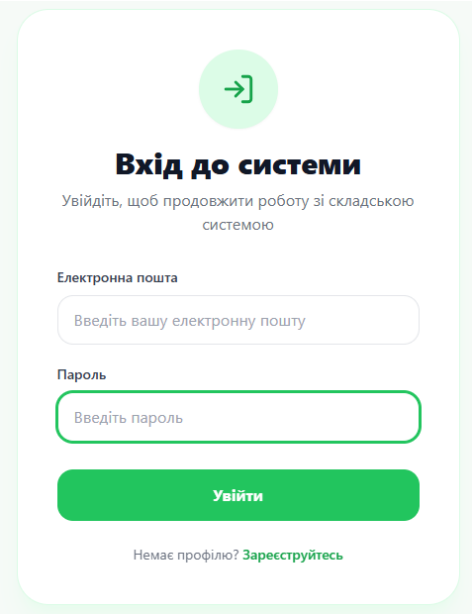


Рис. 3.4 – Форма входу до системи

Якщо користувач не зареєстрований у системі, він може створити новий обліковий запис за допомогою форми реєстрації (рис. 3.5).

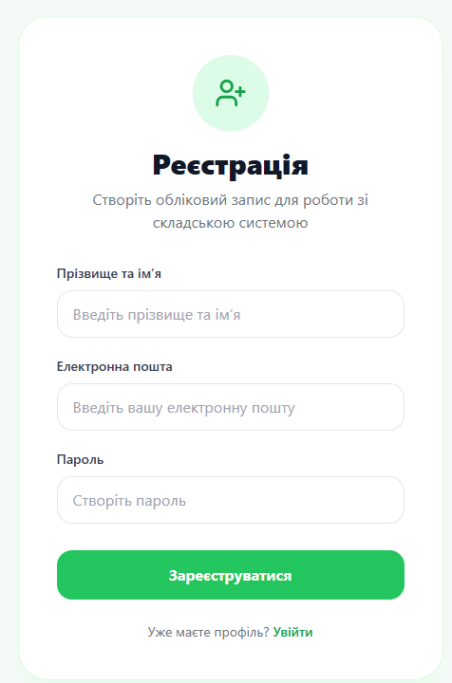


Рис. 3.5 – Форма реєстрації користувача

Форма реєстрації призначена для створення нового профілю користувача в системі, вона містить поля для введення прізвища та імені, електронної пошти й пароля. Після заповнення необхідних даних користувач натискає кнопку «Зареєструватися», після чого може розпочати роботу із системою. У нижній частині форми також розміщено посилання для переходу до сторінки входу, якщо обліковий запис уже створено.

Після успішної авторизації користувач потрапляє до внутрішньої частини системи, де відкривається головна сторінка особистого кабінету (рис. 3.6).

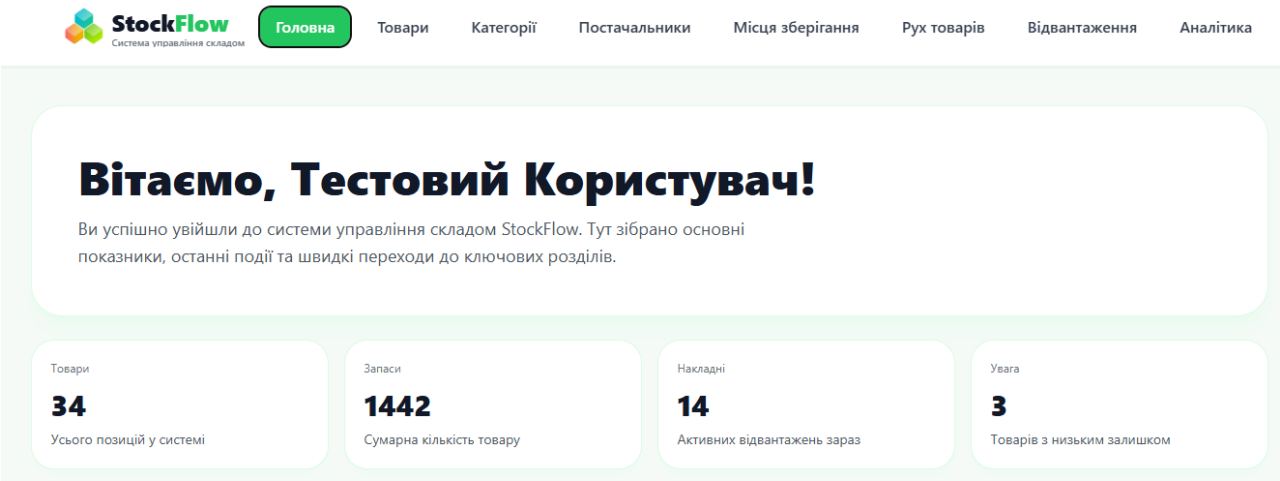


Рис. 3.6 – Головна сторінка системи після авторизації

На головній сторінці системи відображено верхнє навігаційне меню з основними розділами: «Головна», «Товари», «Категорії», «Постачальники», «Місця зберігання», «Рух товарів», «Відвантаження» та «Аналітика». У центральній частині розміщено вітальний блок для користувача, а нижче подано короткі статистичні показники системи: кількість товарів, сумарні запаси, кількість активних накладних та кількість товарів із низьким залишком. Це дозволяє користувачу одразу оцінити загальний стан складської системи після входу.

Для візуального аналізу складської активності на головній сторінці передбачено графік, який наведено на рисунку 3.7.

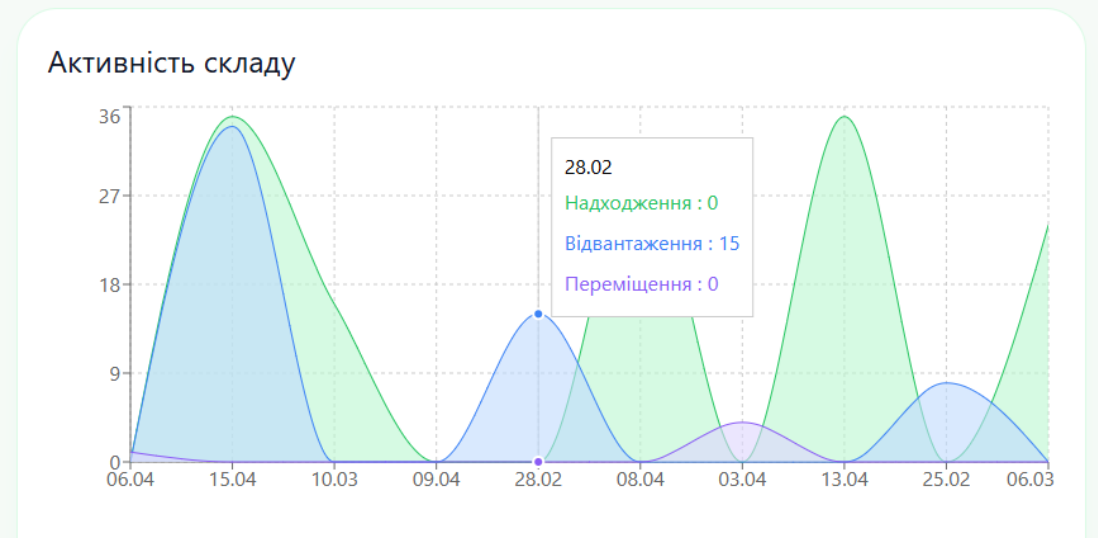


Рис. 3.7 – Графік активності складу

На графіку відображено динаміку основних складських операцій за датами. Окремими кольорами показано надходження, відвантаження та переміщення товарів.

Для зручної навігації між основними модулями системи реалізовано блок швидких дій (рис. 3.8).

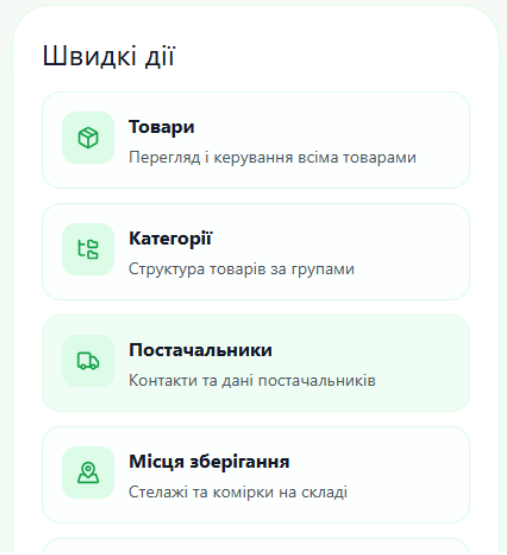


Рис. 3.8 – Блок швидкого переходу до розділів системи

Блок швидких дій містить посилання на ключові сторінки вебзастосунку: товари, категорії, постачальники, місця зберігання, рух товарів, відвантаження та аналітику. Кожен пункт має окрему іконку, назву розділу та короткий опис його призначення. Це спрощує роботу користувача, оскільки він може швидко перейти до потрібного функціонального модуля без пошуку в меню.

Також на головній сторінці відображаються останні складські операції та накладні, що дає змогу контролювати нещодавні зміни в системі (рис. 3.9).

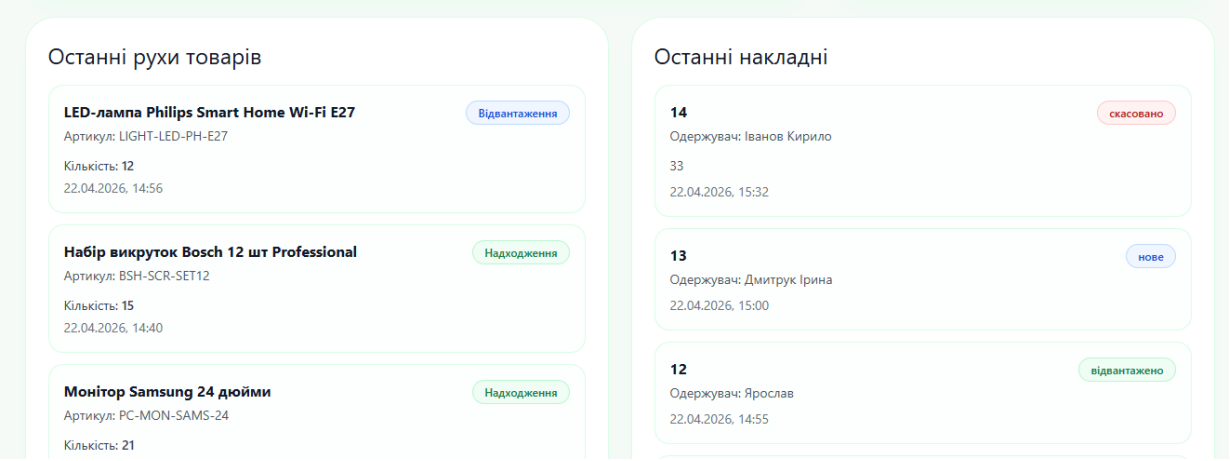


Рис. 3.9 – Останні рухи товарів та накладні

У цьому блоці зліва подано останні рухи товарів із зазначенням назви товару, артикулу, кількості, дати та типу операції. Праворуч відображено останні накладні з номером, одержувачем, датою та поточним статусом. Використання кольорових позначок статусів дозволяє швидко визначити, які операції вже виконані, які є новими, а які були скасовані.

У нижній частині головної сторінки користувач може переглянути критичні залишки товарів і загальний стан системи (рис. 3.10).

В блоці критичних залишків показано товари, кількість яких є меншою за встановлений мінімальний рівень. Для кожного товару вказано поточний

залишок і мінімальне допустиме значення, що допомагає своєчасно виявляти потребу в поповненні запасів.

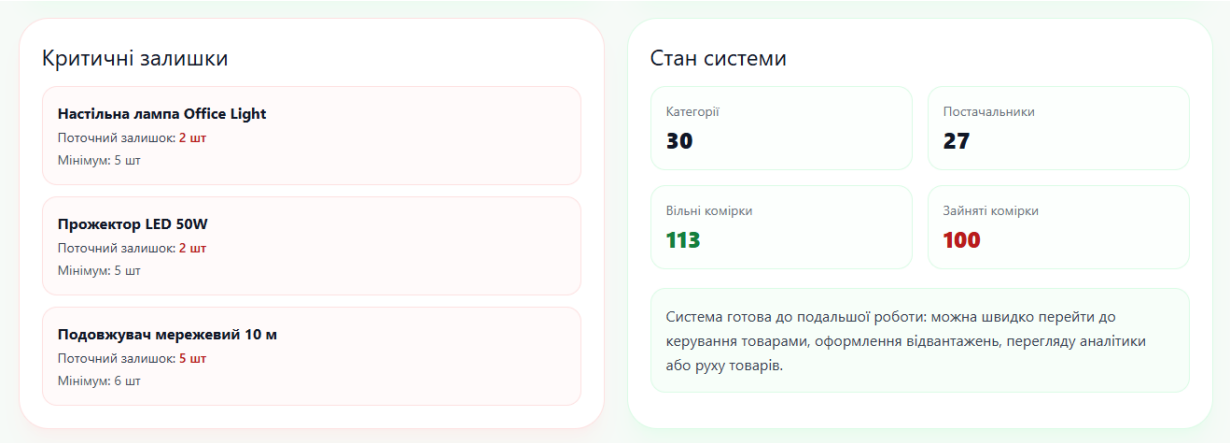


Рис. 3.10 – Критичні залишки та загальний стан системи

Поруч розміщено загальний стан системи, де відображено кількість категорій, постачальників, вільних і зайнятих комірок.

Наступним функціональним розділом системи є сторінка «Товари», яка призначена для додавання товарів та керування вже наявними складі (рис. 3.11).

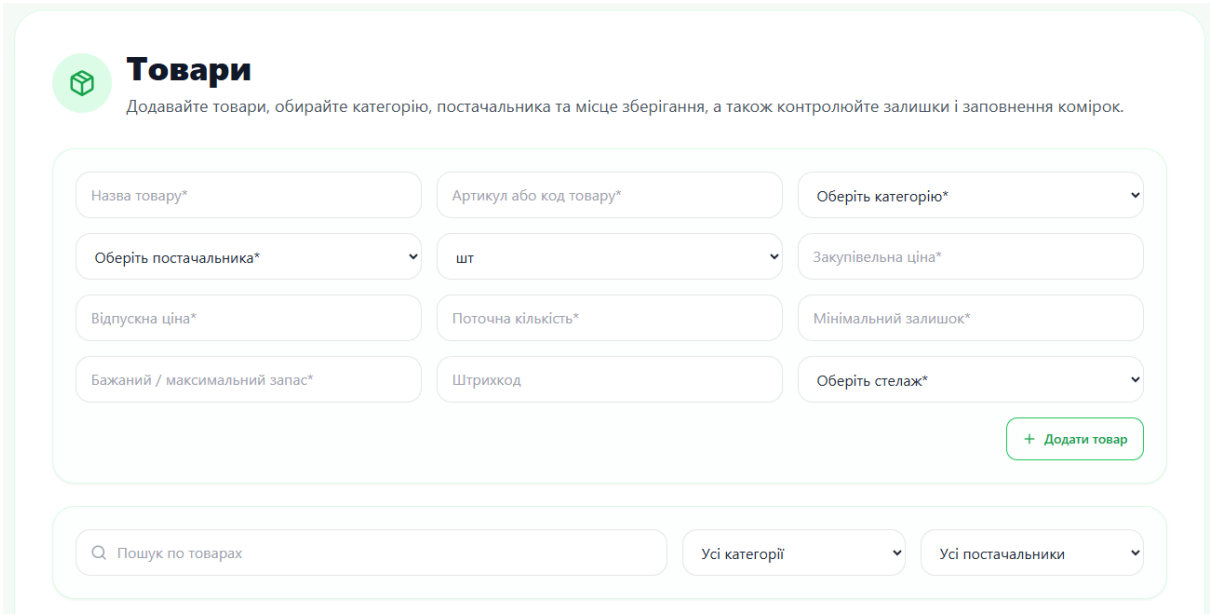
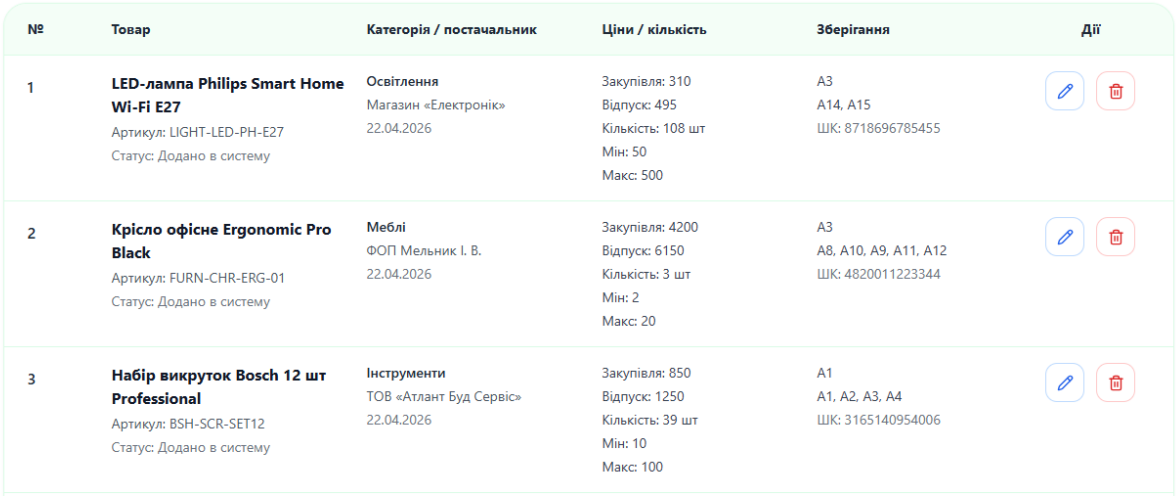


Рис. 3.11 – Форма додавання товару

На сторінці товарів розміщено форму для внесення нової товарної позиції до системи. У формі користувач може вказати назву товару, артикул або код, категорію, постачальника, одиницю вимірювання, закупівельну та відпускну ціну, поточну кількість, мінімальний залишок, бажаний або максимальний запас, штрихкод і стелаж зберігання. Після заповнення необхідних полів товар додається до системи за допомогою кнопки «Додати товар». Нижче також передбачено пошук за товарами та фільтри за категоріями й постачальниками, що спрощує роботу з великою кількістю позицій.

Після додавання товарів вони відображаються у таблиці, де користувач може переглядати основну інформацію про кожну позицію (рис. 3.12).



№	Товар	Категорія / постачальник	Ціни / кількість	Зберігання	Дії
1	**LED-лампа Philips Smart Home Wi-Fi E27** Артикул: LIGHT-LED-PH-E27 Статус: Додано в систему	Освітлення Магазин «Електронік» 22.04.2026	Закупівля: 310 Відпуск: 495 Кількість: 108 шт Мін: 50 Макс: 500	A3 A14, A15 ШК: 8718696785455	
2	**Крісло офісне Ergonomic Pro Black** Артикул: FURN-CHR-ERG-01 Статус: Додано в систему	Меблі ФОП Мельник І. В. 22.04.2026	Закупівля: 4200 Відпуск: 6150 Кількість: 3 шт Мін: 2 Макс: 20	A3 A8, A10, A9, A11, A12 ШК: 4820011223344	
3	**Набір викруток Bosch 12 шт Professional** Артикул: BSH-SCR-SET12 Статус: Додано в систему	Інструменти ТОВ «Атлант Буд Сервіс» 22.04.2026	Закупівля: 850 Відпуск: 1250 Кількість: 39 шт Мін: 10 Макс: 100	A1 A1, A2, A3, A4 ШК: 3165140954006	

Рис. 3.12 – Таблиця товарів

У таблиці товарів наведено список усіх товарних позицій, які збережені в системі. Для кожного товару відображається порядковий номер, назва, артикул, статус, категорія, постачальник, дата додавання, закупівельна та відпускна ціна, поточна кількість, мінімальний і максимальний залишок, місце зберігання, комірки та штрихкод. У правій частині таблиці розміщено кнопки дій, за допомогою яких користувач може редагувати або видаляти обраний товар.

Під час натискання на кнопку редагування відкривається окреме модальне вікно, у якому можна змінити дані вибраного товару (рис. 3.13).

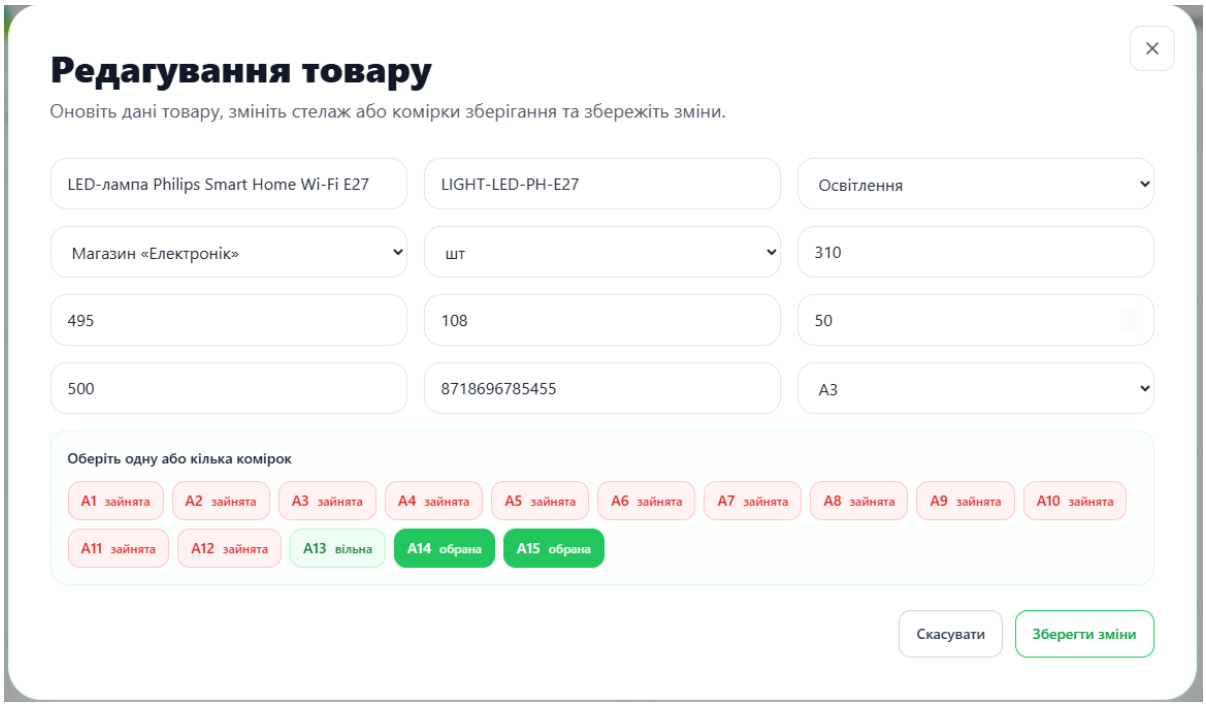


Рис. 3.13 – Форма редагування товару

Форма редагування товару містить уже заповнені поля з поточними даними обраної товарної позиції. Користувач може змінити назву, артикул, категорію, постачальника, одиницю вимірювання, ціни, кількість, мінімальний і максимальний залишок, штрихкод, стелаж та комірки зберігання. Окремо показано доступність комірок: зайняті комірки позначаються червоним кольором, вільні — зеленим, а вибрані — насиченим зеленим кольором. Після внесення змін користувач може зберегти їх або скасувати редагування.

Для запобігання випадковому видаленню товару в системі передбачено додаткове підтвердження дії (рис. 3.14).

Після натискання на кнопку видалення система відкриває попереджувальне вікно з повідомленням про те, що товар буде видалено, а його комірки на складі стануть вільними. Користувач може скасувати дію або підтвердити її за допомогою кнопки «Так, видалити».

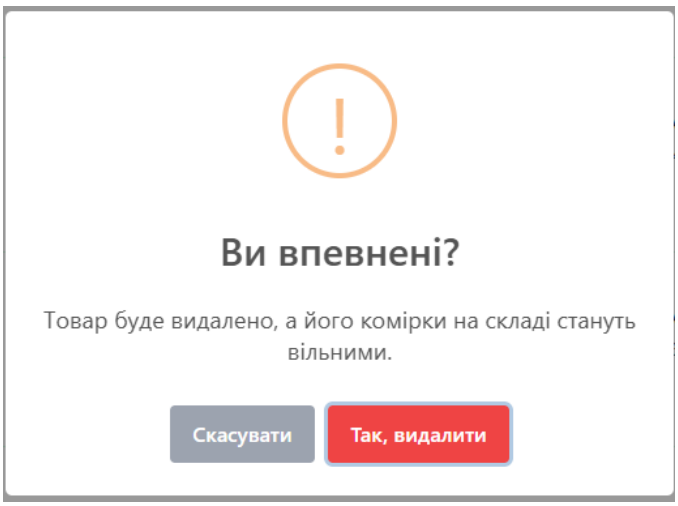


Рис. 3.14 – Форма підтвердження видалення товару

Такий механізм зменшує ризик випадкового видалення важливих даних і забезпечує більш безпечну роботу з товарними позиціями.

Далі в системі реалізовано сторінку керування категоріями товарів, яка має подібну структуру до попередніх розділів і дозволяє підтримувати порядок у каталозі продукції (рис. 3.15).

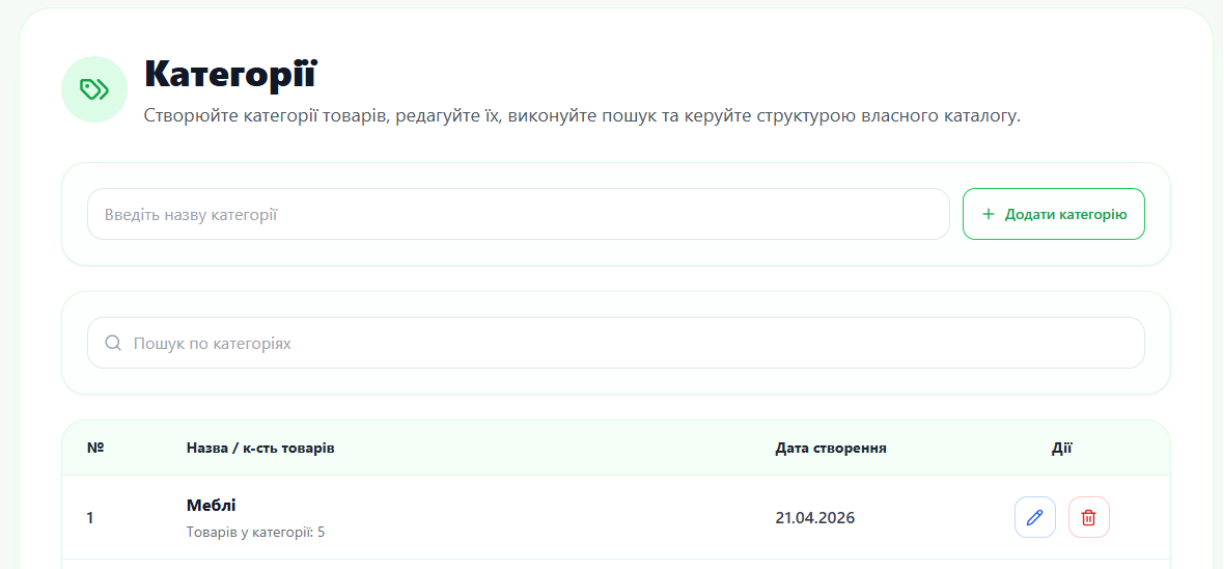


Рис. 3.15 – Сторінка категорій товарів

На сторінці категорій користувач може створювати нові категорії товарів, виконувати пошук за вже доданими категоріями та переглядати їх у табличному форматі. У таблиці відображається номер запису, назва категорії,

кількість товарів у ній, дата створення та доступні дії. Для кожної категорії передбачено можливість редагування або видалення, що дозволяє гнучко керувати структурою товарного каталогу.

Окремий розділ системи призначений для роботи з постачальниками, контактними даними та інформацією про співпрацю з ними (рис. 3.16).

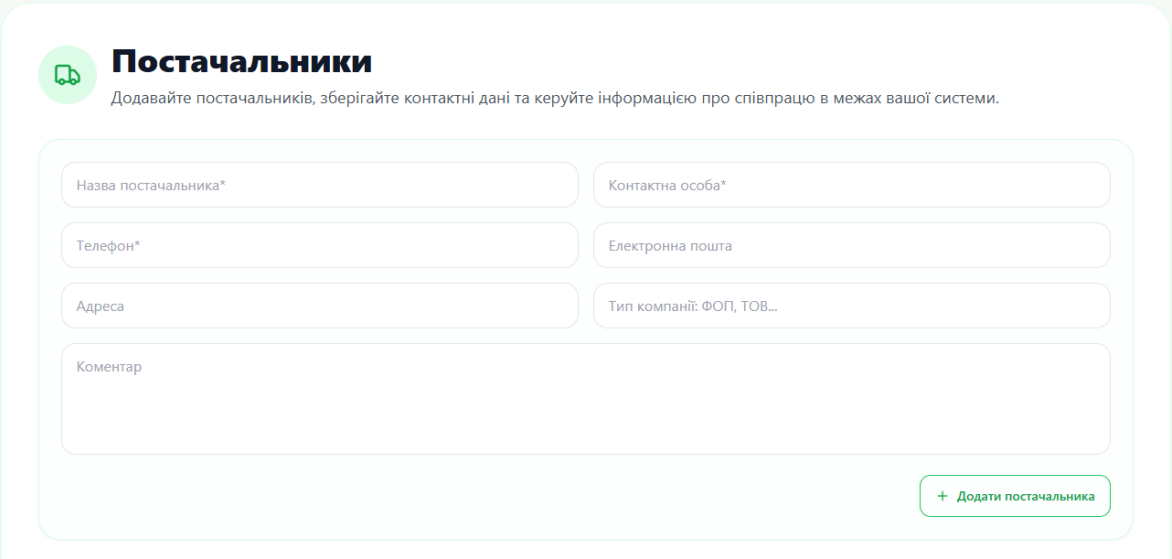


Рис. 3.16 – Сторінка постачальників

На сторінці постачальників розміщено форму додавання нового постачальника, де користувач може вказати назву компанії, контактну особу, телефон, електронну пошту, адресу, тип компанії та додатковий коментар. Така структура дає змогу зберігати всю основну інформацію про постачальників в одному місці. Нижче на сторінці передбачено табличний вивід записів, пошук, а також стандартні дії для подальшого редагування або видалення даних.

Для організації складського простору в системі передбачено сторінку місць зберігання, яка наведена на рисунку 3.17.

Сторінка місць зберігання дозволяє створювати стелажі та задавати кількість комірок для кожного з них. У таблиці відображається назва стелажа, загальна кількість комірок, дата створення, а також стан кожної комірки.

Вільні комірки позначені зеленим кольором, а зайняті червоним, що дає змогу швидко оцінити завантаженість складського простору.

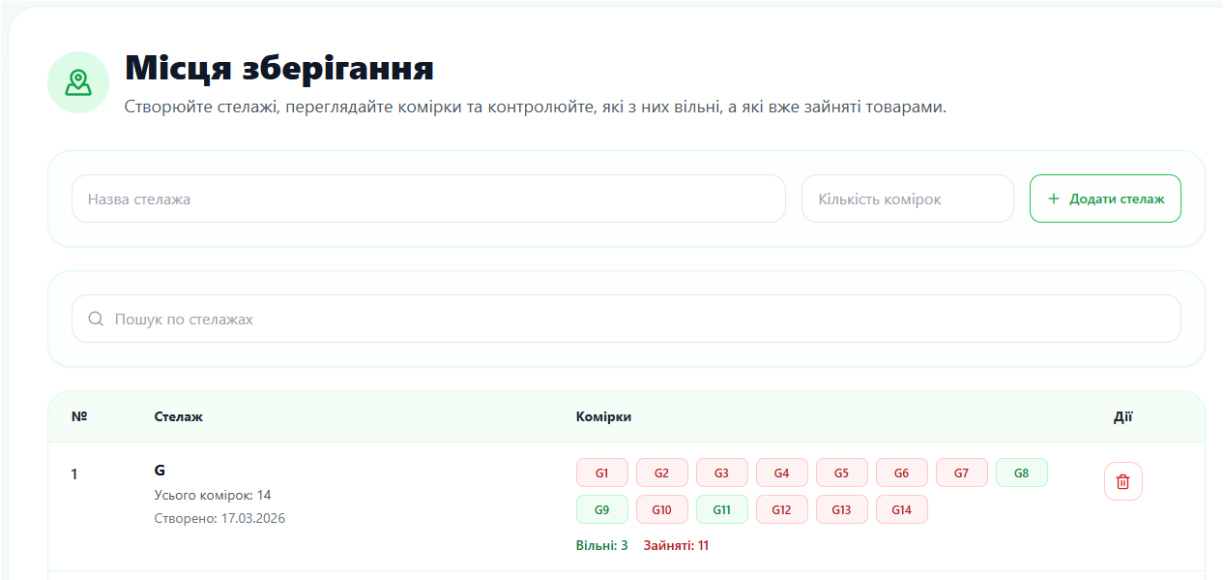


Рис. 3.17 – Сторінка місць зберігання

Також користувач може виконувати пошук за стелажами та видаляти непотрібні записи.

Для фіксації змін кількості товарів у системі реалізовано сторінку руху товарів (рис. 3.18).

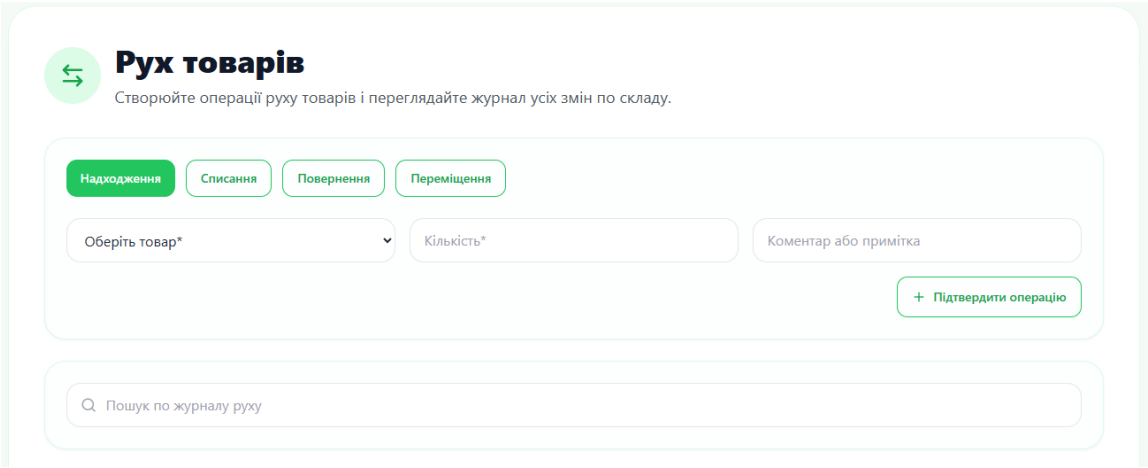


Рис. 3.18 – Сторінка руху товарів

На сторінці руху товарів користувач може створювати операції над складськими запасами. У верхній частині доступні типи операцій: надходження, списання, повернення та переміщення. Після вибору типу операції користувач обирає товар, вказує кількість і за потреби додає коментар або примітку. Після підтвердження операція зберігається в журналі руху, який нижче відображається у табличному форматі. Оскільки сторінки категорій, постачальників, місць зберігання та руху товарів мають схожий принцип побудови, окремо деталізувати кожну таблицю недоцільно, адже всі вони використовують єдиний підхід: форма введення даних, пошук, табличний перегляд і кнопки керування записами.

Наступною сторінкою системи є розділ «Відвантаження», який призначений для створення накладних і контролю процесу передачі товарів одержувачам (рис. 3.19).

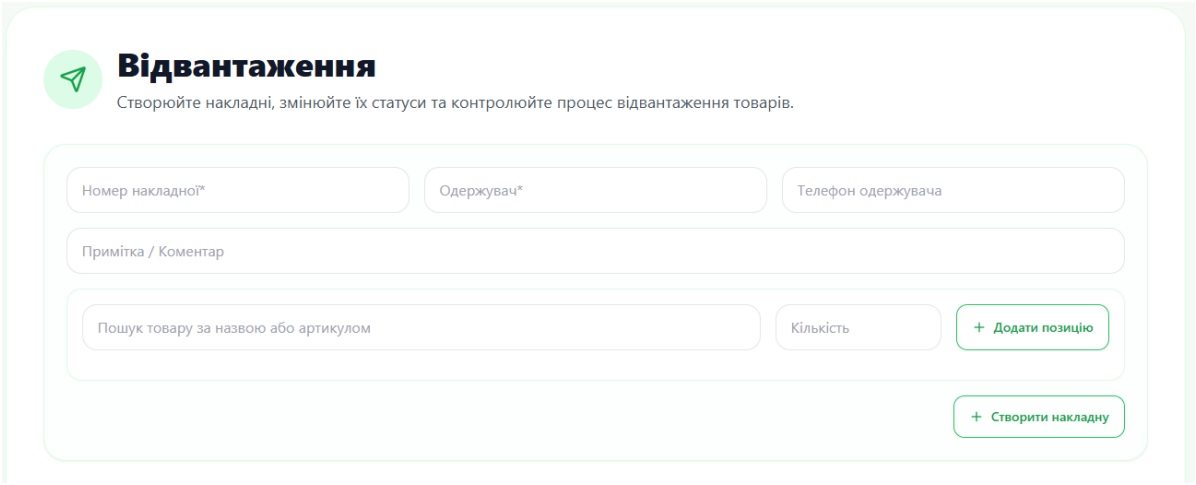


Рис. 3.19 – Сторінка створення відвантаження

На сторінці відвантаження користувач може створити нову накладну, вказавши її номер, одержувача, телефон одержувача та примітку або коментар. Також передбачено додавання товарних позицій до накладної: користувач може знайти товар за назвою або артикулом, вказати потрібну кількість і додати позицію до списку. Після заповнення необхідних даних накладна створюється за допомогою кнопки «Створити накладну».

Нижче на сторінці відображається список створених накладних, причому кожна з них візуально виділяється залежно від поточного статусу (рис. 3.20).

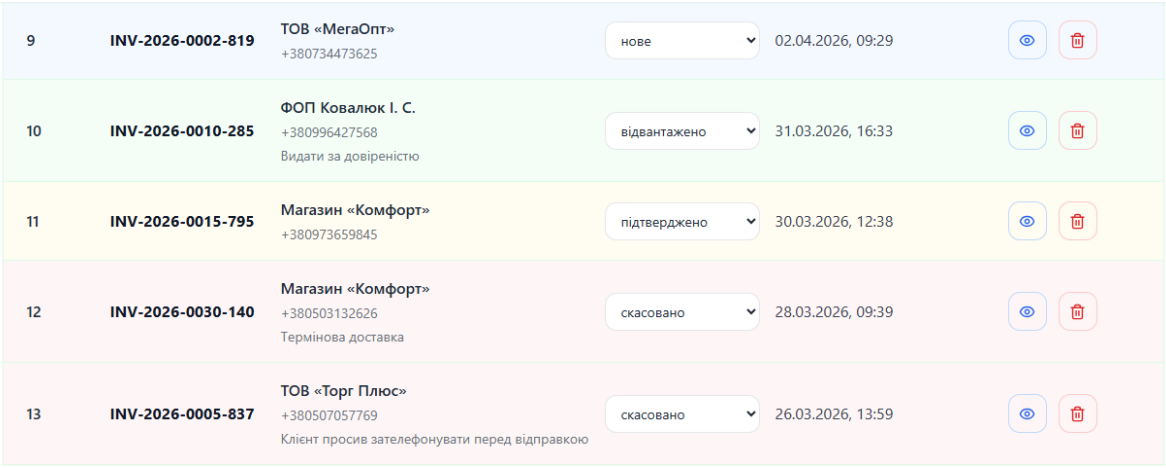


Рис. 3.20 – Список накладних із кольоровим позначенням статусів

У таблиці накладних відображаються номер запису, номер накладної, дані одержувача, статус, дата створення та доступні дії. Для зручності сприйняття записи мають різне кольорове оформлення відповідно до статусу накладної. Для перегляду повної інформації про конкретну накладну користувач може натиснути на іконку ока, після чого відкривається модальне вікно з деталями (рис. 3.21).

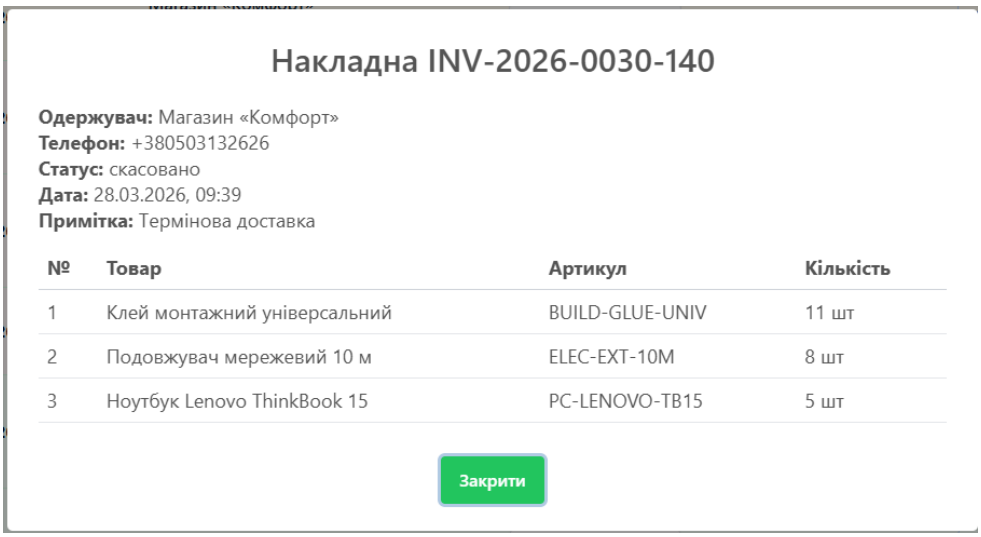


Рис. 3.21 – Детальна інформація про накладну

У вікні детального перегляду відображається номер накладної, одержувач, контактний телефон, статус, дата створення та примітка. Нижче подано перелік товарів, які входять до накладної, із зазначенням їх назви, артикулу та кількості.

Користувач також може переглянути та змінити дані власного профілю (рис. 3.22).

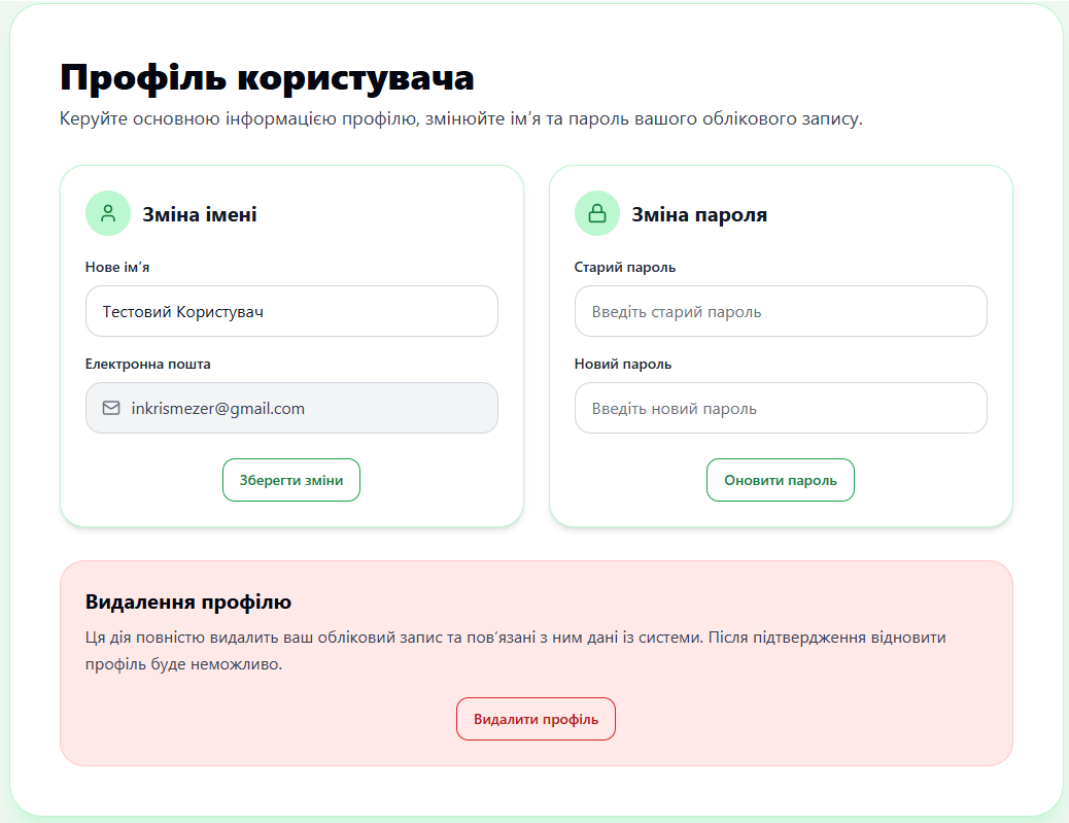


Рис. 3.22 – Сторінка профілю користувача

На сторінці профілю користувач може керувати основною інформацією свого облікового запису. У першому блоці доступна зміна імені користувача, при цьому електронна пошта відображається окремо як дані облікового запису. У другому блоці реалізовано зміну пароля шляхом введення старого та нового пароля. Також на сторінці передбачено окрему зону видалення профілю, яка виділена червоним кольором і попереджає користувача про незворотність цієї дії.

Основним підсумковим розділом системи є сторінка «Аналітика», яка дає змогу переглядати узагальнені показники роботи складу та оцінювати стан товарних запасів (рис. 3.23).

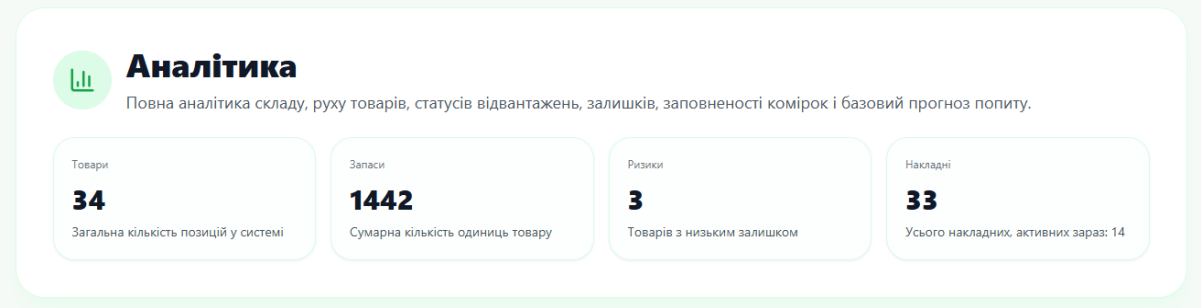


Рис. 3.23 – Загальні показники сторінки аналітики

У верхній частині сторінки аналітики розміщено інформаційні картки з основними показниками системи. Тут відображається загальна кількість товарних позицій, сумарна кількість одиниць товару, кількість ризикових позицій із низьким залишком, а також загальна кількість накладних і число активних накладних. Такий блок дозволяє швидко оцінити поточний стан складу без переходу до окремих розділів.

Для аналізу змін на складі використовується графік активності руху товарів, який наведено на рисунку 3.24.

Рис. 3.24 – Графік активності руху товарів

На графіку показано активність складських операцій за датами. Окремими кольорами відображаються різні типи руху товарів: надходження, відвантаження, списання, повернення та переміщення. Завдяки такому поданню користувач може порівняти інтенсивність операцій у різні періоди та визначити дні з найбільшим навантаженням на склад.

Для контролю стану товарних запасів у системі передбачено кругову діаграму залишків (рис. 3.25).

Рис. 3.25 – Кругова діаграма статусів залишків

Кругова діаграма відображає співвідношення товарів із нормальним залишком та товарів із низьким залишком. Зелений сектор показує позиції, кількість яких відповідає нормі, а червоний сектор позначає товари, що потребують уваги через наближення до мінімального рівня. Це дає змогу швидко виявити потенційні проблеми із запасами.

Окремо на сторінці аналітики розміщено кругові діаграми, що показують стан накладних та заповненість місць зберігання (рис. 3.26).

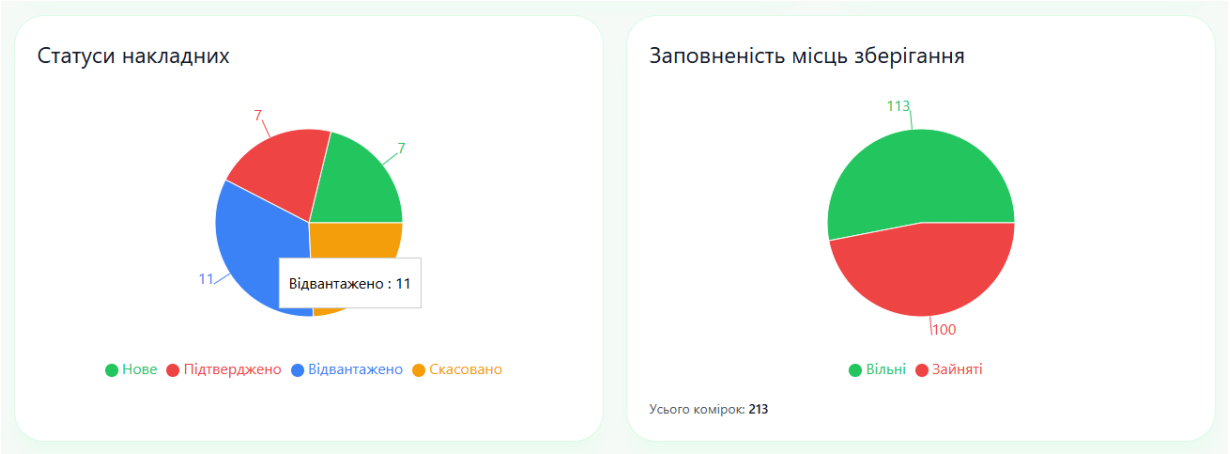


Рис. 3.26 – Кругові діаграми статусів накладних і заповненості місць зберігання

У першій діаграмі відображено розподіл накладних за статусами: нові, підтверджені, відвантажені та скасовані. Друга діаграма показує співвідношення вільних і зайнятих комірок на складі. Таке візуальне подання допомагає одночасно контролювати процес відвантаження та рівень завантаженості складського простору.

Для оцінки вартості товарних груп і популярності окремих позицій використовуються графіки топових показників (рис. 3.27).

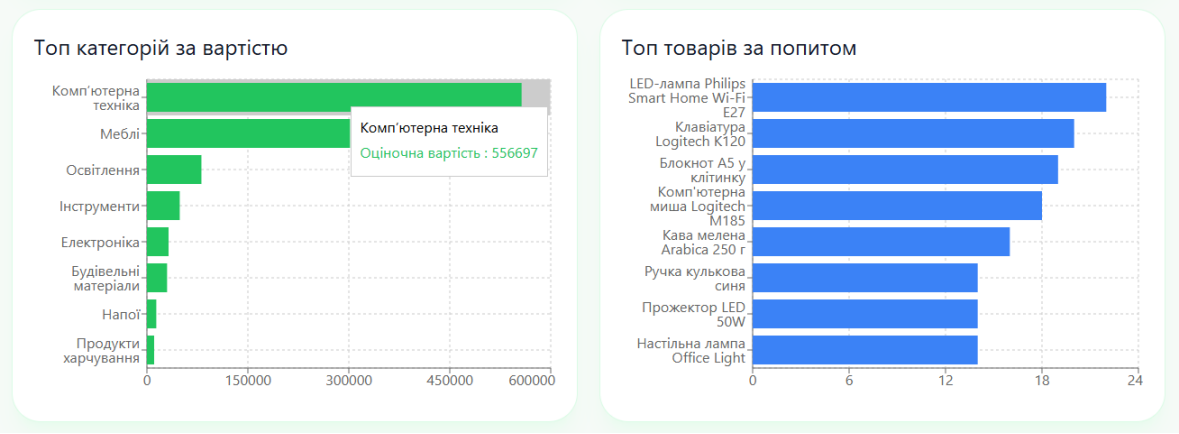


Рис. 3.27 – Графіки топових категорій і товарів

У цьому блоці відображено топ категорій за вартістю та топ товарів за попитом. Перший графік дозволяє визначити, які категорії формують

найбільшу частину вартості складських залишків. Другий графік показує товари, що мають найбільший попит за результатами операцій відвантаження. Це допомагає краще розуміти структуру запасів і визначати найбільш важливі товарні позиції.

Додатково в системі подано стовпчикові діаграми, що узагальнюють рух товарів і дані про постачальників (рис. 3.28).

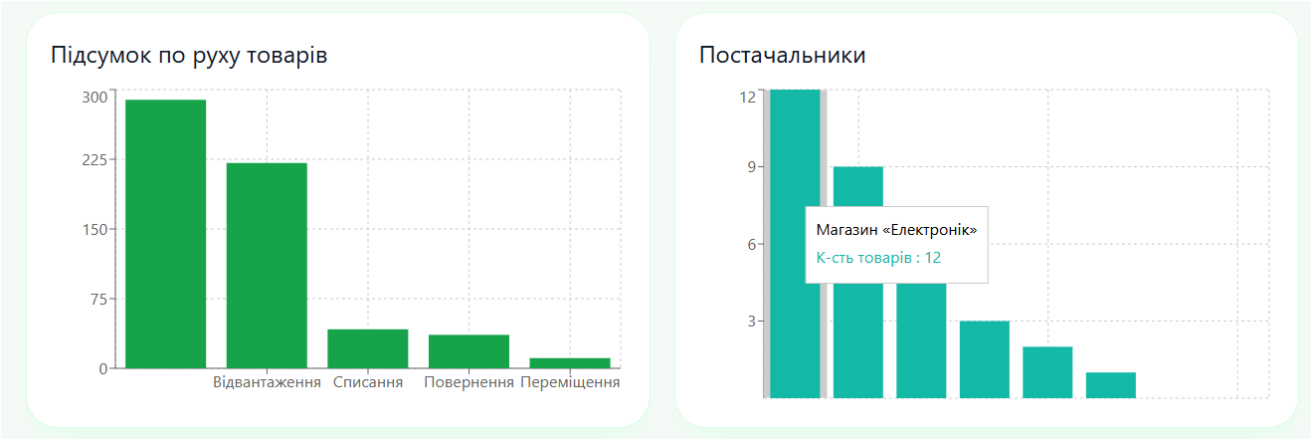


Рис. 3.28 – Стовпчикові діаграми руху товарів і постачальників

У першій стовпчиковій діаграмі показано підсумок за основними типами руху товарів, зокрема відвантаженням, списанням, поверненням і переміщенням. Друга діаграма відображає постачальників за кількістю пов’язаних із ними товарів. Завдяки цим графікам користувач може швидко оцінити, які операції виконуються найчастіше, а також які постачальники мають найбільшу кількість товарних позицій у системі.

Для базового прогнозування потреби в товарах у розділі аналітики передбачено графік прогнозу попиту (рис. 3.29).Графік прогнозу попиту порівнює поточний залишок товарів із прогнозованою потребою на найближчі сім днів.

Окремими лініями показано наявні запаси та очікуваний попит, що дозволяє виявити товари, які можуть швидко вичерпатися. Такий інструмент допомагає користувачу завчасно планувати поповнення складу.

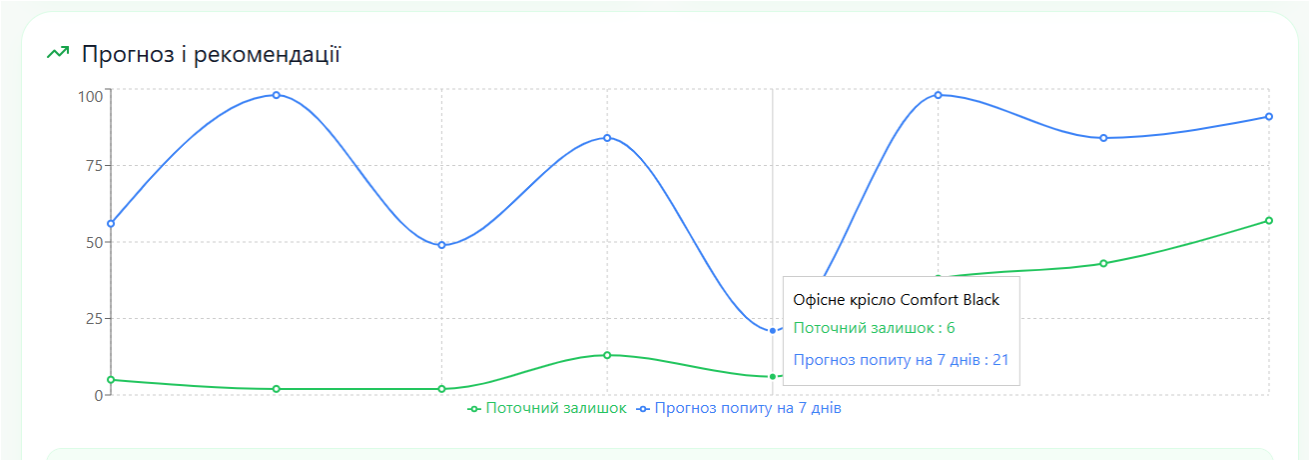


Рис. 3.29 – Графік прогнозу попиту

Нижче прогнозні дані також подаються у табличному вигляді для зручнішого порівняння конкретних товарів (рис. 3.30).

Товар	Залишок	Прогноз 7 днів	Днів до вичерпання
Подовжувач мережевий 10 м	5 шт	56 шт	0
Настільна лампа Office Light	2 шт	98 шт	0
Прожектор LED 50W	2 шт	49 шт	0
LED панель 36W	13 шт	84 шт	1
Офісне крісло Comfort Black	6 шт	21 шт	2
Крупа гречана 1 кг	38 шт	98 шт	2
Чай чорний байховий 100 г	43 шт	84 шт	3
Цукор фасований 1 кг	57 шт	91 шт	4

Рис. 3.30 – Таблиця прогнозу попиту

У таблиці прогнозу попиту наведено перелік товарів, їх поточний залишок, прогнозовану потребу на сім днів та орієнтовну кількість днів до вичерпання запасу. Такий формат дозволяє швидко визначити критичні позиції, для яких потрібно найближчим часом сформувати замовлення або поповнення.

У завершальній частині сторінки аналітики відображаються додаткові підсумкові дані щодо критичних запасів і загальної вартості залишків (рис. 3.31). У цьому блоці подано товари з критичним залишком, де вказано поточну кількість, мінімальний рівень, бажаний запас і рекомендовану кількість для

замовлення. Також нижче відображено закупівельну вартість залишків, оціночну вартість продажу та співвідношення кількості категорій і постачальників.

Рис. 3.31 – Додаткові аналітичні показники системи

Ці дані допомагають користувачу оцінити фінансовий стан запасів і прийняти рішення щодо подальшого поповнення складу.

В ході демонстрації було проведено і тестування основних сторінкок та функціональних можливостей вебзастосунку StockFlow. Продемонстровані модулі працюють коректно: дані правильно додаються, відображаються, редагуються та видаляються, а основні показники системи оновлюються відповідно до виконаних дій. Інтерфейс є послідовним і зрозумілим, усі елементи розміщені логічно, а табличне та графічне подання інформації дозволяє швидко оцінити стан складу.

3.4 Висновки до розділу

В третьому розділі було виконано практичну реалізацію інформаційної системи управління складськими запасами StockFlow. Обґрунтовано вибір інструментів розробки. Також було використано допоміжні бібліотеки для побудови інтерфейсу, виведення повідомлень, роботи з графіками та покращення зручності користування системою.

У межах розділу реалізовано основні функціональні модулі системи. Окрему увагу приділено сторінці аналітики, де реалізовано відображення загальних показників складу, графіків руху товарів, статусів залишків, заповненості комірок, попиту на товари та рекомендацій щодо поповнення запасів.

Було проведено тестування системи та продемонстровано результати її роботи безпосередньо в браузері. Перевірено коректність роботи основних сторінок, форм введення даних, таблиць, кнопок редагування й видалення, модальних вікон, навігації та аналітичних блоків.

РОЗДІЛ 4. ОХОРОНА ПРАЦІ

4.1 Організаційно-правові основи забезпечення безпеки праці

Охорона праці є важливою складовою організації безпечної та ефективної трудової діяльності, оскільки вона спрямована на збереження життя, здоров'я і працездатності працівників у процесі виконання ними професійних обов'язків. У сучасних умовах питання безпеки праці набуває особливого значення, адже будь-яка виробнича або організаційна діяльність має супроводжуватися дотриманням встановлених норм, правил і вимог щодо захисту працівників від можливих ризиків.

Законодавчі основи охорони праці в Україні формуються системою нормативно-правових актів, які визначають права та обов'язки працівників і роботодавців у сфері безпеки праці. Одним із базових документів є Кодекс законів про працю України [27], у якому закріплено основні гарантії працівників, вимоги до умов праці, режиму робочого часу, відпочинку та відповідальності сторін трудових відносин. Цей документ створює загальну правову основу для регулювання трудових відносин і забезпечення належних умов праці.

Важливе місце в системі правового регулювання займає Закон України «Про охорону праці» [28]. Він визначає основні принципи державної політики у сфері охорони праці, встановлює вимоги до організації безпечного виробничого середовища, проведення інструктажів, навчання працівників, оцінювання ризиків і запобігання нещасним випадкам. Відповідно до цього закону, роботодавець зобов'язаний створювати безпечні умови праці, забезпечувати працівників необхідними засобами захисту, контролювати стан робочих місць і своєчасно усувати небезпечні фактори.

Законодавство у сфері охорони праці виконує не лише регулятивну, а й профілактичну функцію. Воно визначає порядок організації безпечної роботи, встановлює вимоги до технічного стану обладнання, умов перебування

працівників на робочих місцях, санітарно-гігієнічних норм і заходів щодо попередження професійних захворювань. Дотримання цих вимог дозволяє зменшити ризик травматизму, підвищити рівень відповідальності роботодавців і працівників, а також забезпечити стабільну роботу підприємств та організацій.

Окрему увагу в сучасній практиці охорони праці приділяють ризик-орієнтованому підходу. Його сутність полягає в тому, що безпека праці забезпечується не лише шляхом виконання загальних правил, а й через постійне виявлення, аналіз і мінімізацію можливих небезпек. Такий підхід дозволяє своєчасно визначати потенційні загрози для працівників, оцінювати ймовірність їх виникнення та впроваджувати заходи для їх попередження. Це особливо важливо для підприємств, де використовуються сучасні інформаційні системи, технічне обладнання або автоматизовані процеси.

Досвід Європейського Союзу також має значення для розвитку системи охорони праці, оскільки в ЄС діє комплекс директив і стандартів, спрямованих на забезпечення безпечних умов праці. Одним із ключових документів є Рамкова директива 89/391/ЄЕС [29], яка визначає загальні принципи профілактики ризиків, захисту працівників і відповідальності роботодавців у сфері безпеки праці. Також важливою є Директива 89/654/ЄЕС, що встановлює мінімальні вимоги до безпеки та гігієни праці на робочому місці. Ці документи підкреслюють необхідність системного управління ризиками, проведення профілактичних заходів і постійного контролю умов праці.

Законодавчі основи охорони праці формують правову базу для створення безпечного робочого середовища, захисту працівників від небезпечних і шкідливих факторів, а також запобігання нещасним випадкам. Для інформаційної системи управління складськими запасами дотримання таких вимог є важливим, оскільки її використання пов’язане з роботою персоналу за комп’ютерним обладнанням, обробкою даних і організацією складських процесів. Тому врахування законодавчих вимог у сфері охорони

праці є необхідною умовою безпечного та ефективного функціонування системи.

4.2 Характеристика об’єкта та виявлення потенційних небезпек

Об’єктом розгляду є інформаційна система управління складськими запасами StockFlow, розроблена у вигляді вебзастосунку. Система призначена для автоматизації основних процесів складського обліку, зокрема ведення інформації про товари, категорії, постачальників, місця зберігання, рух товарів, відвантаження та аналітичні показники. Робота з системою здійснюється через веббраузер із використанням персонального комп’ютера або ноутбука. Основними користувачами системи є адміністратор, менеджер складу, працівник складу або інша відповідальна особа, яка виконує операції з обліку товарів, контролю залишків, оформлення накладних і перегляду аналітичних даних. Система може використовуватися як в офісному приміщенні, так і безпосередньо в робочій зоні складу. Робоче місце користувача зазвичай включає комп’ютер або ноутбук, монітор, клавіатуру, мишу, робочий стіл, крісло, мережеве обладнання та периферійні пристрої. До фізичних факторів належать недостатня освітленість робочої зони, підвищена яскравість або відблиски на екрані, підвищений рівень шуму, незадовільні параметри мікроклімату, а також небезпека ураження електричним струмом під час використання комп’ютерної техніки. До психофізіологічних факторів належать розумове перенапруження, перенапруження зорового аналізатора, монотонність праці, емоційне навантаження та статичне фізичне навантаження через тривале перебування в сидячому положенні.

Хімічні та біологічні небезпечні фактори для роботи з вебзастосунком StockFlow не є основними, оскільки експлуатація системи не передбачає безпосередньої роботи з токсичними речовинами, хімічними реагентами, патогенними мікроорганізмами або біологічними об’єктами. Водночас у складському середовищі можуть бути додаткові умови, що впливають на

комфорт працівника, зокрема запиленість, недостатня вентиляція або шум від складського обладнання.

Загальна характеристика об’єкта проєктування наведена в таблиці 4.1.

Таблиця 4.1 – Характеристика об’єкта проєктування

Ознака	Характеристика
Назва об’єкта	Інформаційна система управління складськими запасами StockFlow
Тип об’єкта	Вебзастосунок для автоматизації складського обліку
Основне призначення	Облік товарів, категорій, постачальників, місць зберігання, руху товарів, відвантажень та аналітики
Середовище використання	Офісне приміщення або робоча зона складу
Основні користувачі	Адміністратор системи, менеджер складу, працівник складу, відповідальна особа за облік
Технічні засоби	Комп’ютер або ноутбук, монітор, клавіатура, миша, мережеве обладнання
Режим роботи	Періодична або тривала робота з даними через браузер
Основні операції користувача	Додавання, редагування, видалення та перегляд даних, створення накладних, аналіз залишків
Основні фактори впливу	Робота з екраном, сидяче положення, електрообладнання, інформаційне навантаження, складське середовище

Як видно з таблиці система StockFlow є програмним об’єктом, однак її використання безпосередньо пов’язане з робочим місцем користувача та технічними засобами. Тому під час аналізу небезпек потрібно враховувати як фізичні фактори, пов’язані з роботою за комп’ютером, так і організаційні та інформаційні ризики.

Для виявлення потенційних небезпек доцільно враховувати не лише сам програмний продукт, а й умови його експлуатації.

Оскільки користувач працює з системою через комп’ютер, основними джерелами небезпеки є монітор, електрообладнання, організація робочого місця, освітлення, мікроклімат приміщення, рівень шуму та характер виконуваних операцій.

Виявлені потенційні небезпеки наведено в таблиці 4.2.

Таблиця 4.2 – Виявлення потенційних небезпек

№	Група факторів	Потенційна небезпека	Джерело небезпеки	Можливі наслідки
1	Фізичні	Недостатня освітленість робочої зони	Неправильне розміщення робочого місця, нестача природного або штучного освітлення	Втома очей, головний біль, зниження концентрації
2	Фізичні	Підвищена яскравість, відблиски на екрані	Неправильне розташування монітора, пряме або відбите світло	Перенапруження зору, дискомфорт під час роботи
3	Фізичні	Підвищений рівень шуму	Робота складського обладнання, офісної техніки, рух працівників або транспорту	Втома, подразнення, зниження уважності
4	Фізичні	Незадовільний мікроклімат	Недостатня вентиляція, підвищена або знижена температура, низька рухливість повітря	Сонливість, погіршення самопочуття, зниження працездатності
5	Фізичні	Небезпека ураження електричним струмом	Комп’ютерна техніка, розетки, подовжувачі, зарядні пристрої, мережеве обладнання	Травмування працівника, пошкодження обладнання, зупинка роботи системи
6	Фізичні	Пожежна небезпека	Несправна електропроводка, перевантаження електромережі, коротке замикання	Пошкодження обладнання, загроза життю і здоров’ю працівників
7	Психофізіологічні	Статичне фізичне навантаження	Тривале сидяче положення під час роботи з системою	Біль у спині, шиї, плечах, порушення постави
8	Психофізіологічні	Перенапруження зорового аналізатора	Тривала робота з монітором, перегляд таблиць, форм і графіків	Втома очей, зниження уважності, головний біль
9	Психофізіологічні	Розумове перенапруження	Робота з великою кількістю товарів, накладних, залишків і аналітичних даних	Перевтома, зниження концентрації, підвищення ймовірності помилок
10	Психофізіологічні	Монотонність праці	Повторювані операції введення, редагування та перевірки даних	Втома, зниження мотивації, неуважність
11	Психофізіологічні	Емоційне навантаження	Відповідальність за правильність складського обліку, накладних і залишків	Стрес, напруження, помилки в роботі

Продовження таблиці 4.2

12	Організаційні / інформаційні	Помилки введення даних	Некоректне заповнення форм, неправильне внесення кількості, ціни, статусу або місця зберігання	Невідповідність складських залишків, помилки у звітності, порушення обліку
13	Організаційні / інформаційні	Несанкціонований доступ до системи	Слабкий пароль, залишена активна сесія, передача доступу іншим особам	Зміна, видалення або викрадення складських даних
14	Організаційні / інформаційні	Втрата або пошкодження даних	Помилки користувача, збої мережі, некоректне видалення записів	Порушення роботи складу, потреба у відновленні інформації

Як видно з таблиці найбільш характерними для об’єкта проєктування є фізичні та психофізіологічні небезпечні й шкідливі фактори. Фізичні фактори пов’язані переважно з умовами робочого середовища та використанням комп’ютерної техніки. До них належать освітлення, шум, мікроклімат, електробезпека та пожежна безпека.

Психофізіологічні фактори пов’язані з тривалою роботою за комп’ютером, інтенсивним сприйняттям інформації, необхідністю уважного введення даних та відповідальністю за правильність складського обліку. Окремо слід враховувати організаційні та інформаційні небезпеки. Помилки введення даних, неправильна зміна статусів накладних, некоректне видалення записів або несанкціонований доступ до системи можуть призвести до порушення складського обліку та втрати достовірності інформації.

Для узагальнення виявлених факторів їх можна згрупувати за характером впливу на користувача та систему, що наведено в таблиці 4.3.

Таблиця 4.3 – Групування потенційних небезпек за характером впливу

Група небезпек	**Приклади небезпек**	**Сфера впливу**
Фізичні	Недостатнє освітлення, відблиски, шум, незадовільний мікроклімат, електричний струм	Здоров’я працівника, безпека робочого місця, справність обладнання

Продовження таблиці 4.3

Психофізіологічні	Статичне навантаження, перенапруження зору, розумове перенапруження, монотонність праці	Працездатність, уважність, фізичний і психологічний стан користувача
Організаційні	Недостатня підготовка користувача, неправильна організація робочого процесу	Якість виконання операцій, ефективність роботи
Інформаційні	Несанкціонований доступ, помилки введення, втрата або зміна даних	Достовірність складського обліку, захист даних, стабільність роботи системи

Для системи StockFlow найбільш суттєвими є фізичні та психофізіологічні фактори, оскільки робота користувача пов’язана з комп’ютерною технікою, тривалим перебуванням у сидячому положенні, зоровим навантаженням і обробкою значного обсягу складських даних.

Вплив більшості визначених небезпечних і шкідливих факторів можна зменшити або повністю усунути шляхом правильної організації робочого місця, забезпечення належного освітлення, вентиляції, дотримання режиму праці та відпочинку, використання справного електрообладнання, проведення інструктажу користувачів і застосування засобів захисту інформації.

4.3 Дослідження ризику реалізації потенційних небезпек та розробка заходів щодо їх попередження

Для оцінювання ризиків, пов’язаних із використанням інформаційної системи StockFlow, доцільно застосувати метод матриці оцінювання ризиків. Цей метод дозволяє визначити рівень небезпеки з урахуванням двох основних показників: серйозності можливих наслідків та ймовірності виникнення небезпечної події.

Під час роботи з інформаційною системою StockFlow найбільш актуальними є ризики, пов’язані з перенапруженням зору, статичним навантаженням, психоемоційним напруженням, помилками введення даних, електробезпекою, пожежною небезпекою та інформаційною безпекою. Для

кожної потенційної небезпеки необхідно визначити категорію серйозності наслідків і рівень імовірності її виникнення.

Категорії серйозності небезпеки наведено в таблиці 4.4.

Таблиця 4.4 – Категорії серйозності небезпеки

Вид	Категорія	Опис можливих наслідків
Катастрофічна	I	Смерть людини, повне руйнування або критична втрата працездатності системи
Критична	II	Серйозна травма, стійке захворювання, значне пошкодження обладнання або суттєве порушення роботи системи
Гранична	III	Незначна травма, короткочасне захворювання, тимчасове погіршення стану працівника або часткове порушення роботи системи
Незначна	IV	Незначний дискомфорт, короткочасна втома, незначні помилки або несуттєві порушення в роботі

Рівні ймовірності виникнення небезпечної події наведено в таблиці 4.5.

Таблиця 4.5 – Рівні ймовірності виникнення небезпеки

Вид	Рівень	Опис імовірності
Часто	A	Подія має високу ймовірність виникнення під час звичайної роботи
Ймовірно	B	Подія може траплятися кілька разів протягом періоду експлуатації системи
Час від часу	C	Подія може іноді виникати за певних умов
Віддалено	D	Подія є малоймовірною, але можливою
Неймовірно	E	Подія є настільки малоймовірною, що її виникнення майже не очікується

Перед використанням матриці кожна виявлена небезпека попередньо оцінюється за двома показниками: можливою тяжкістю наслідків та частотою або ймовірністю її виникнення. Поєднання цих показників дозволяє визначити умовний індекс ризику, який надалі використовується для встановлення пріоритетності заходів безпеки. Чим вищою є серйозність наслідків і частота події, тим більшу увагу необхідно приділити усуненню або мінімізації відповідної небезпеки Матрицю оцінювання ризиків наведено в таблиці 4.6.

Таблиця 4.6 – Матриця оцінювання ризику

Частота виникнення події	**I Катастрофічна**	**II Критична**	**III Гранична**	**IV Незначна**
A Часто	1A	2A	3A	4A
B Ймовірно	1B	2B	3B	4B
C Час від часу	1C	2C	3C	4C
D Віддалено	1D	2D	3D	4D
E Неймовірно	1E	2E	3E	4E

Індекси ризику дозволяють визначити ступінь його припустимості. Класифікацію рівнів ризику наведено в таблиці 4.7.

Таблиця 4.7 – Класифікація індексів ризику

Індекс ризику	**Рівень ризику**
1A, 1B, 1C, 2A, 2B, 3A	Неприпустимий, надмірний ризик
1D, 2C, 2D, 3B, 3C	Небажаний, гранично допустимий ризик
1E, 2E, 3D, 3E, 4A, 4B	Припустимий ризик із перевіркою
4C, 4D, 4E	Припустимий ризик без перевірки

На основі визначених у попередньому підрозділі потенційних небезпек було проведено оцінювання ризиків для інформаційної системи StockFlow, результати наведено в таблиці 4.8.

Таблиця 4.8 – Оцінювання ризиків

№	**Потенційна небезпека**	**Категорія серйозності**	**Рівень імовірності**	**Індекс ризику**	**Оцінка ризику**
1	Перенапруження зору під час роботи з монітором	III	B	3B	Небажаний, гранично допустимий
2	Статичне навантаження через тривале сидяче положення	III	B	3B	Небажаний, гранично допустимий
3	Психоемоційне навантаження під час роботи з великою кількістю даних	III	C	3C	Небажаний, гранично допустимий
4	Помилки введення даних у систему	III	B	3B	Небажаний, гранично допустимий

Продовження таблиці 4.8

5	Ураження електричним струмом	II	D	2D	Небажаний, гранично допустимий
6	Пожежна небезпека через несправність електрообладнання	II	D	2D	Небажаний, гранично допустимий
7	Недостатня вентиляція або незадовільний мікроклімат	IV	C	4C	Припустимий без перевірки
8	Підвищений рівень шуму в складському середовищі	IV	C	4C	Припустимий без перевірки
9	Несанкціонований доступ до системи	III	C	3C	Небажаний, гранично допустимий
10	Втрата або пошкодження даних	II	D	2D	Небажаний, гранично допустимий

За результатами оцінювання видно, що більшість ризиків належить до небажаного, але гранично допустимого рівня. Це означає, що такі ризики не є критичними за умови правильної організації праці, однак потребують обов’язкового контролю та впровадження профілактичних заходів.

Найбільш важливими для користувачів системи StockFlow є ризики, пов’язані з перенапруженням зору, статичним навантаженням, помилками введення даних, електробезпекою та захистом інформації.

Для зниження виявлених ризиків необхідно застосувати комплекс організаційних, технічних, ергономічних та інформаційних заходів. Запропоновані заходи наведено в таблиці 4.9.

Таблиця 4.9 – Заходи щодо зниження ризиків

№	**Небезпека**	**Запропонований захід**	**Очікуваний результат**
1	Перенапруження зору	Забезпечити достатнє освітлення робочої зони, правильно налаштувати яскравість монітора, уникати відблисків, робити короткі перерви під час тривалої роботи	Зменшення втоми очей, головного болю та підвищення концентрації користувача
2	Статичне навантаження	Використовувати зручне крісло, правильно розміщувати монітор, клавіатуру та мишу, періодично змінювати положення тіла	Зменшення навантаження на спину, шию, плечі та руки

Продовження таблиці 4.9

3	Психоемоційне навантаження	Раціонально організовувати робочий час, уникати надмірного навантаження, робити перерви, розподіляти обов’язки між працівниками	Зниження стресу, втоми та підвищення уважності
4	Помилки введення даних	Використовувати обов’язкові поля, перевірку введених значень, підтвердження видалення, пошук, фільтри та зрозумілу структуру форм	Зменшення кількості помилок у складському обліку
5	Ураження електричним струмом	Використовувати справне обладнання, перевіряти кабелі, не перевантажувати подовжувачі та розетки, дотримуватися правил електробезпеки	Зниження ризику травмування користувача та пошкодження техніки
6	Пожежна небезпека	Контролювати стан електромережі, не використовувати пошкоджені кабелі, забезпечити доступ до засобів пожежогасіння	Зменшення ймовірності виникнення пожежі
7	Недостатня вентиляція або незадовільний мікроклімат	Регулярно провітрювати приміщення, підтримувати комфортну температуру та вологість повітря	Покращення самопочуття та працездатності працівника
8	Підвищений рівень шуму	Організувати робоче місце подалі від джерел шуму, за потреби використовувати шумозахисні засоби або виділену робочу зону	Підвищення концентрації та зменшення втоми
9	Несанкціонований доступ до системи	Використовувати надійні паролі, не передавати облікові дані іншим особам, завершувати сеанс після роботи	Захист складських даних від стороннього доступу
10	Втрата або пошкодження даних	Обмежувати доступ до критичних дій, застосовувати підтвердження видалення, контролювати правильність операцій, використовувати хмарне зберігання даних	Зменшення ризику втрати інформації та підвищення надійності обліку

Окрему роль у зниженні ризиків відіграють функціональні можливості самої системи StockFlow. Наявність авторизації користувача дозволяє обмежити доступ до системи та захистити складські дані від стороннього втручання. Використання структурованих форм введення даних зменшує ймовірність помилок під час додавання товарів, постачальників, місць зберігання або накладних. Кнопки підтвердження видалення допомагають

уникнути випадкового вилучення важливої інформації, а таблиці, пошук і фільтри полегшують роботу з великими обсягами даних. Аналітичний модуль також сприяє зниженню організаційних ризиків. Завдяки графікам, діаграмам, показникам критичних залишків і прогнозу попиту користувач може швидше виявляти проблемні позиції та приймати обґрунтовані рішення щодо поповнення запасів. Це знижує ризик помилок у плануванні та підвищує ефективність управління складом.

Після впровадження запропонованих заходів очікується зниження рівня більшості ризиків до припустимого, особливо це стосується ризиків, пов’язаних із роботою за комп’ютером, організацією робочого місця, правильністю введення даних і захистом інформації. Ризики, які мають низьку ймовірність, але можуть мати серйозні наслідки, зокрема ураження електричним струмом або пожежна небезпека, повинні контролюватися шляхом регулярної перевірки обладнання та дотримання правил безпеки.

Застосування матриці оцінювання ризиків дозволило визначити найбільш суттєві небезпеки при роботі з інформаційною системою StockFlow та встановити рівень їх припустимості. Запропоновані заходи спрямовані на зменшення впливу фізичних, психофізіологічних, технічних, організаційних та інформаційних ризиків. Їх виконання забезпечує безпечніші умови праці, підвищує точність складського обліку та сприяє стабільній роботі системи.

4.4 Висновки

В розділі «Охорона праці» було розглянуто питання безпечної роботи користувачів з інформаційною системою StockFlow. Основну увагу приділено умовам праці під час використання комп’ютерної техніки, організації робочого місця, електробезпеці та захисту інформації.

Було охарактеризовано об’єкт проєктування вебзастосунок StockFlow, який використовується для управління складськими запасами, товарами, постачальниками, місцями зберігання, рухом товарів і відвантаженнями.

Визначено основні потенційні небезпеки: перенапруження зору, статичне навантаження, психоемоційне напруження, помилки введення даних, ризик ураження електричним струмом, пожежна небезпека, шум, недостатня вентиляція та інформаційні ризики.

Для оцінювання небезпек було використано матрицю оцінювання ризиків. Встановлено, що більшість ризиків є гранично допустимими, але потребують контролю та впровадження профілактичних заходів.

Для зниження ризиків запропоновано забезпечити правильне освітлення, зручне робоче місце, регулярні перерви, справність електрообладнання, провітрювання приміщення, використання надійних паролів і уважну перевірку даних перед виконанням операцій у системі.

Запропоновані заходи дозволяють зменшити вплив небезпечних і шкідливих факторів, підвищити безпеку праці користувачів та забезпечити стабільну роботу системи StockFlow.

ВИСНОВКИ

В ході виконання роботи було розглянуто теоретичні та практичні аспекти розробки інформаційної системи управління складськими запасами. На початковому етапі було проаналізовано поняття складських запасів, особливості їх обліку, методи контролю залишків і підходи до прогнозування попиту. Це дозволило визначити основні проблеми, які виникають під час ручного або частково автоматизованого ведення складського обліку, а також обґрунтувати потребу у створенні зручної веборієнтованої системи.

У процесі дослідження було сформульовано задачу розробки системи, яка має забезпечувати облік товарів, категорій, постачальників, місць зберігання, руху товарів і відвантажень. Також було визначено необхідність реалізації аналітичного модуля, який дозволяє користувачу отримувати узагальнені показники щодо стану запасів, активності складських операцій, критичних залишків і прогнозованого попиту.

На етапі проєктування було проведено аналіз існуючих аналогів, що дало змогу визначити їх переваги та недоліки, а також сформувати вимоги до власної системи. Було спроєктовано архітектуру вебзастосунку, визначено основні компоненти системи та логіку їх взаємодії. Окрему увагу приділено розробці структури бази даних, у якій передбачено зберігання інформації про користувачів, товари, категорії, постачальників, складські комірки, рухи товарів і накладні.

Практичним результатом роботи стала розроблена інформаційна система StockFlow. Для її реалізації було використано React для створення клієнтської частини та Firebase для автентифікації користувачів, зберігання даних і забезпечення взаємодії з базою даних. У системі реалізовано сторінки реєстрації та входу, головну панель користувача, модулі керування товарами, категоріями, постачальниками, місцями зберігання, рухом товарів, відвантаженнями, профілем користувача та аналітикою.

Проведене тестування показало, що основні функціональні можливості системи працюють коректно. Дані успішно додаються, редагуються, видаляються та відображаються у відповідних таблицях і графіках. Інтерфейс системи є зрозумілим і послідовним, а використання інформаційних карток, таблиць, діаграм і графіків дозволяє користувачу швидко оцінювати поточний стан складу. Отримані результати підтвердили, що розроблена система відповідає поставленим вимогам і може бути використана для автоматизації базових процесів складського обліку.

Також у роботі було розглянуто питання охорони праці під час використання системи. Визначено основні потенційні небезпеки, пов’язані з роботою за комп’ютером, зокрема ергономічні ризики, перенапруження зору, психоемоційне навантаження, електробезпеку та інформаційну безпеку. Запропоновано заходи для зниження цих ризиків, що сприяє безпечній і комфортній роботі користувачів із системою.

Мету роботи було досягнуто, оскільки в результаті виконання дослідження було спроєктовано, реалізовано та протестовано вебзастосунок для управління складськими запасами. Розроблена система дозволяє автоматизувати основні складські процеси, зменшити ризик помилок під час обліку, покращити контроль залишків і надати користувачу зручні інструменти для аналізу стану складу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Марчук В. Є., Григорак М. Ю., Гармаш О. М., Овдієнко О. В. Складська логістика : навчальний посібник. Київ : ОЛДІ-ПЛЮС, 2020. URL: https://www.researchgate.net/profile/Oleg-Garmash/publication/377659147_Skladska_logistika_navcalnij_posibnik/links/65b1646b7fe0d83cb565a40c/Skladska-logistika-navcalnij-posibnik.pdf
2. Логістика : навчальний посібник. URL: https://mk.nmu.org.ua/ua/source/Logisticl9.pdf
3. Логістика : навчальний матеріал. URL: https://studfile.net/preview/8896934/
4. Логістика : електронний підручник. Організація бух. обліку в бюджетних установах 6. ОРГАНІЗАЦІЯ ОБЛІКУ МАТЕРІАЛЬНИХ ЗАПАСІВ. URL: https://buklib.net/books/25047/
5. Логістика : навчальний матеріал. URL: https://studfile.net/preview/5013330/page:4/
6. Попит, пропозиція, їх взаємодія. Оборотність запасів : економічний показник. URL: https://analizua.com/slovnik-ekonomichnikh-terminiv/293-oborotnist-zapasiv
7. Метод ковзного середнього. URL: http://wiki.tntu.edu.ua/Метод_ковзного_середнього
8. Zoho Inventory: Inventory Management Software for Growing Businesses. URL: https://www.zoho.com/inventory/
9. Zoho Inventory Review: Is It the Right Inventory Management Tool? The Retail Exec. URL: https://theretailexec.com/tools/zoho-inventory-review/
10. Odoo Inventory: Modern Online Warehouse Management Software. URL: https://www.odoo.com/uk_UA/app/inventory
11. Odoo Inventory Reviews: Pros & Cons. G2 Business Software Reviews. URL: https://www.g2.com/products/odoo-inventory/reviews?qs=pros-and-cons

12. inFlow Inventory: Inventory Software for Small and Mid-size Business. URL: https://www.inflowinventory.com/

13. inFlow Inventory Software Reviews & Ratings. Software Advice. URL: https://www.softwareadvice.com/scm/inflow-inventory-profile/reviews/

14. Sortly: Simple Inventory Management Software. URL: https://www.sortly.com/

15. Sortly Reviews: Pros & Cons. G2 Business Software Reviews. URL: https://www.g2.com/products/sortly/reviews?qs=pros-and-cons

16. Visual Studio Code: Code Editing. Redefined. URL: https://code.visualstudio.com/

17. React: A JavaScript library for building user interfaces. URL: https://legacy.reactjs.org/

18. Tailwind CSS: Rapidly build modern websites without ever leaving your HTML. URL: https://tailwindcss.com/

19. Recharts: A composable charting library built on React components. URL: https://recharts.github.io/

20. SweetAlert2: A beautiful, responsive, customizable, accessible replacement for JavaScript's popup boxes. URL: https://sweetalert2.github.io/

21. React Icons: Include popular icons in your React projects easily. URL: https://react-icons.github.io/react-icons/

22. Firebase: Google's mobile and web app development platform. URL: https://firebase.google.com/

23. Firebase Authentication: Authenticate and manage users. URL: https://firebase.google.com/docs/auth

24. Cloud Firestore: Flexible, scalable NoSQL cloud database. URL: https://firebase.google.com/docs/firestore

25. GitHub: Where the world builds software. URL: https://github.com/

26. Netlify: Develop, deploy, and scale modern web projects. URL: https://www.netlify.com/

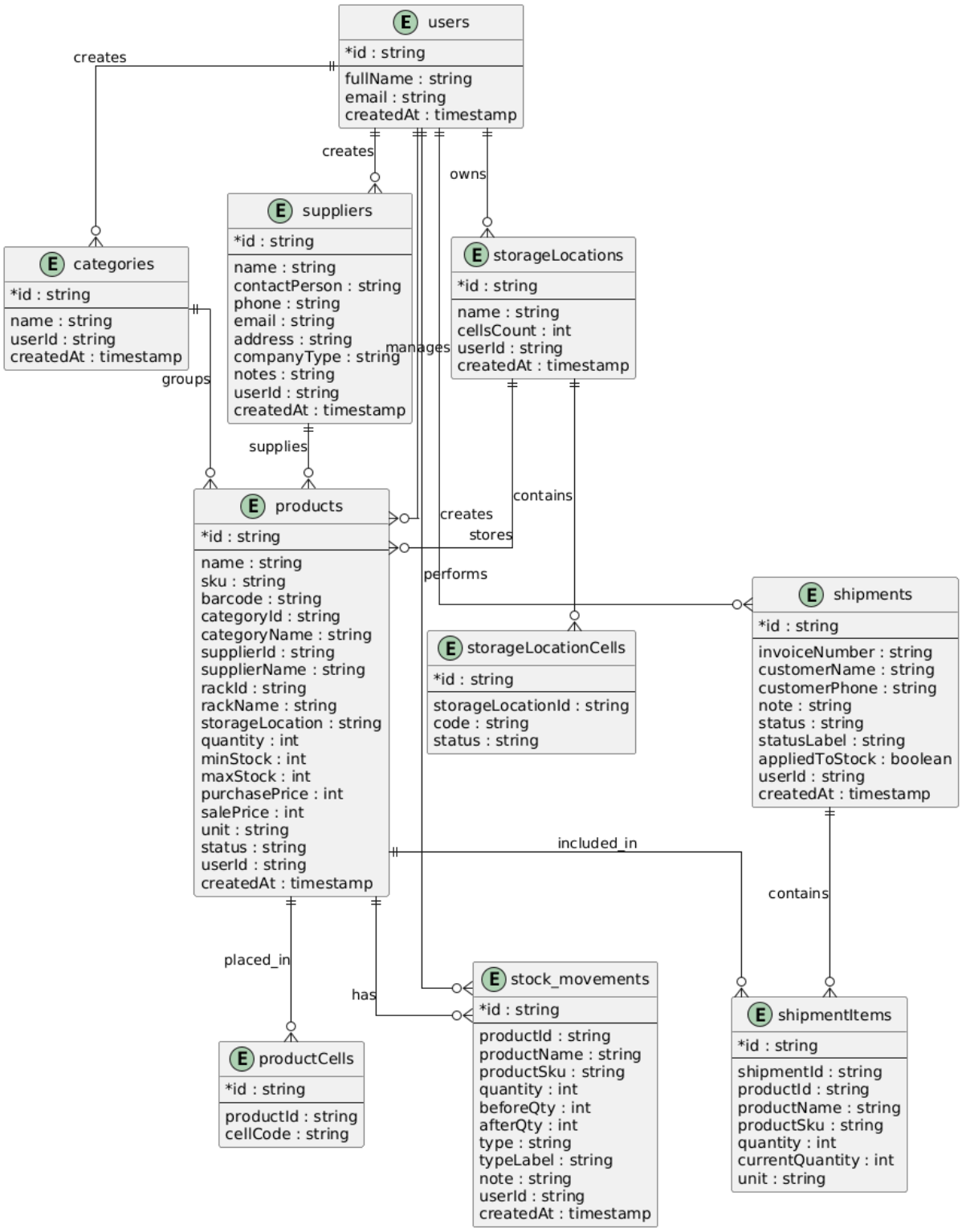
27. Кодекс законів про працю України : Закон України від 10.12.1971 № 322-VIII. URL: https://zakon.rada.gov.ua/laws/show/322-08#Text

28. Про охорону праці : Закон України від 14.10.1992 № 2694-XII. URL: https://zakon.rada.gov.ua/laws/show/2694-12#Text

29. Директива Ради 89/391/ЄЕС про впровадження заходів, що сприяють покращенню безпечних і здорових умов праці працівників від 12.06.1989. URL: http://rac.org.ua/uploads/content/274/files/directive89-391.pdf

ДОДАТОК А

Логічна діаграма бази даних:



ДОДАТОК Б

Лістинг ключових елементів системи.

Головна публічна сторінка, компонент MainPage.jsx:

```
import { useCallback, useEffect, useMemo, useState } from "react";
import { Link } from "react-router-dom";
import {
  Package,
  Truck,
  ArrowLeftRight,
  Send,
  BarChart3,
  FolderTree,
  MapPinned,
} from "lucide-react";
import { useAuth } from "../context/AuthContext";
import { db } from "../firebase";
import { collection, getDocs, query, where } from "firebase/firestore";
import {
  ResponsiveContainer,
  AreaChart,
  Area,
  CartesianGrid,
  XAxis,
  YAxis,
  Tooltip,
} from "recharts";

export default function MainPage() {
  const { user } = useAuth();

  const [loading, setLoading] = useState(true);
  const [products, setProducts] = useState([]);
  const [categories, setCategories] = useState([]);
  const [suppliers, setSuppliers] = useState([]);
  const [shipments, setShipments] = useState([]);
  const [movements, setMovements] = useState([]);
  const [storageLocations, setStorageLocations] = useState([]);

  const fetchData = useCallback(async () => {
    if (!user) return;

    setLoading(true);

    try {
      const [
        productsSnapshot,
        categoriesSnapshot,
        suppliersSnapshot,
        shipmentsSnapshot,
        movementsSnapshot,
        storageSnapshot,
      ] = await Promise.all([
        getDocs(query(collection(db, "products"), where("userId", "==",
user.uid))),
        getDocs(query(collection(db, "categories"), where("userId",
"==", user.uid))),
```

```
            getDocs(query(collection(db, "suppliers"), where("userId", "==",
user.uid))),
            getDocs(query(collection(db, "shipments"), where("userId", "==",
user.uid))),
            getDocs(query(collection(db, "stock_movements"), where("userId",
"==", user.uid))),
            getDocs(query(collection(db, "storageLocations"),
where("userId", "==", user.uid))),
          ]);

          setProducts(productsSnapshot.docs.map((doc) => ({ id: doc.id,
...doc.data() })));
          setCategories(categoriesSnapshot.docs.map((doc) => ({ id: doc.id,
...doc.data() })));
          setSuppliers(suppliersSnapshot.docs.map((doc) => ({ id: doc.id,
...doc.data() })));
          setShipments(shipmentsSnapshot.docs.map((doc) => ({ id: doc.id,
...doc.data() })));
          setMovements(movementsSnapshot.docs.map((doc) => ({ id: doc.id,
...doc.data() })));
          setStorageLocations(storageSnapshot.docs.map((doc) => ({ id:
doc.id, ...doc.data() })));
        } finally {
          setLoading(false);
        }
      }, [user]);

      useEffect(() => {
        fetchData();
      }, [fetchData]);

      const sortByCreatedAtDesc = (items) => {
        return [...items].sort((a, b) => {
          const aSec = a.createdAt?.seconds || 0;
          const bSec = b.createdAt?.seconds || 0;
          return bSec - aSec;
        });
      };

      const formatDateTime = (timestamp) => {
        if (!timestamp?.toDate) return "—";
        return timestamp.toDate().toLocaleString("uk-UA", {
          day: "2-digit",
          month: "2-digit",
          year: "numeric",
          hour: "2-digit",
          minute: "2-digit",
        });
      };

      const formatShortDate = (timestamp) => {
        if (!timestamp?.toDate) return "—";
        return timestamp.toDate().toLocaleDateString("uk-UA", {
          day: "2-digit",
          month: "2-digit",
        });
      };

      const dashboardStats = useMemo(() => {
        const totalProducts = products.length;
        const totalCategories = categories.length;
        const totalSuppliers = suppliers.length;
        const totalShipments = shipments.length;
```

```
          const totalQuantity = products.reduce(
            (sum, item) => sum + (Number(item.quantity) || 0),
            0,
          );

          const lowStockCount = products.filter(
            (item) => (Number(item.quantity) || 0) <= (Number(item.minStock)
|| 0),
          ).length;

          const allCells = storageLocations.flatMap((item) => item.cells ||
[]);
          const occupiedCells = allCells.filter((cell) => cell.status ===
"occupied").length;
          const freeCells = allCells.filter((cell) => cell.status ===
"free").length;

          const activeShipments = shipments.filter(
            (item) => item.status === "new" || item.status === "confirmed",
          ).length;

          return {
            totalProducts,
            totalCategories,
            totalSuppliers,
            totalShipments,
            totalQuantity,
            lowStockCount,
            occupiedCells,
            freeCells,
            activeShipments,
          };
        }, [products, categories, suppliers, shipments, storageLocations]);

        const recentMovements = useMemo(() => {
          return sortByCreatedAtDesc(movements).slice(0, 5);
        }, [movements]);

        const recentShipments = useMemo(() => {
          return sortByCreatedAtDesc(shipments).slice(0, 5);
        }, [shipments]);

        const lowStockProducts = useMemo(() => {
          return products
            .filter((item) => (Number(item.quantity) || 0) <=
(Number(item.minStock) || 0))
            .sort((a, b) => (Number(a.quantity) || 0) - (Number(b.quantity) ||
0))
            .slice(0, 5);
        }, [products]);

        const activityChartData = useMemo(() => {
          const grouped = {};

          movements.forEach((item) => {
            const key = formatShortDate(item.createdAt);

            if (!grouped[key]) {
              grouped[key] = {
                date: key,
                incoming: 0,
                outgoing: 0,
```

```
        transfer: 0,
      };
    }

    const qty = Number(item.quantity) || 0;

    if (item.type === "incoming") grouped[key].incoming += qty;
    if (item.type === "outgoing") grouped[key].outgoing += qty;
    if (item.type === "transfer") grouped[key].transfer += qty;
  });

  return Object.values(grouped).slice(-10);
}, [movements]);

const quickLinks = [
  {
    title: "Товари",
    text: "Перегляд і керування всіма товарами",
    to: "/products",
    icon: <Package size={20} />,
  },
  {
    title: "Категорії",
    text: "Структура товарів за групами",
    to: "/categories",
    icon: <FolderTree size={20} />,
  },
  {
    title: "Постачальники",
    text: "Контакти та дані постачальників",
    to: "/suppliers",
    icon: <Truck size={20} />,
  },
  {
    title: "Місця зберігання",
    text: "Стелажі та комірки на складі",
    to: "/storage-locations",
    icon: <MapPinned size={20} />,
  },
  {
    title: "Рух товарів",
    text: "Операції та журнал змін",
    to: "/inventory-movements",
    icon: <ArrowLeftRight size={20} />,
  },
  {
    title: "Відвантаження",
    text: "Накладні та статуси відправок",
    to: "/shipments",
    icon: <Send size={20} />,
  },
  {
    title: "Аналітика",
    text: "Графіки, статистика та прогноз",
    to: "/analytics",
    icon: <BarChart3 size={20} />,
  },
];

const getShipmentBadgeClass = (status) => {
  switch (status) {
    case "new":
      return "bg-blue-50 text-blue-700 border-blue-200";
```

```
            case "confirmed":
              return "bg-amber-50 text-amber-700 border-amber-200";
            case "shipped":
              return "bg-green-50 text-green-700 border-green-200";
            case "canceled":
              return "bg-red-50 text-red-700 border-red-200";
            default:
              return "bg-gray-50 text-gray-700 border-gray-200";
          }
        };

        const getMovementBadgeClass = (type) => {
          switch (type) {
            case "incoming":
              return "bg-green-50 text-green-700 border-green-200";
            case "outgoing":
              return "bg-blue-50 text-blue-700 border-blue-200";
            case "writeoff":
              return "bg-red-50 text-red-700 border-red-200";
            case "return":
              return "bg-amber-50 text-amber-700 border-amber-200";
            case "transfer":
              return "bg-violet-50 text-violet-700 border-violet-200";
            default:
              return "bg-gray-50 text-gray-700 border-gray-200";
          }
        };

        if (loading) {
          return (
            <main className="min-h-screen bg-[#f6faf7] px-4 py-10">
              <div className="max-w-7xl mx-auto">
                <div className="rounded-[2rem] border border-green-100 bg-
white shadow-xl shadow-green-100/40 p-10 text-center text-gray-500">
                  Завантаження головної сторінки...
                </div>
              </div>
            </main>
          );
        }

        return (
          <main className="min-h-screen bg-[#f6faf7] px-4 py-10">
            <div className="max-w-7xl mx-auto space-y-6">
              <div className="rounded-[2rem] border border-green-100 bg-white
shadow-xl shadow-green-100/40 p-8 sm:p-10 md:p-12">
                <h1 className="text-3xl sm:text-4xl md:text-5xl font-black
text-gray-900 mb-4">
                  Вітаємо{user?.displayName ? `, ${user.displayName}` : ""}!
                </h1>

                <p className="text-gray-600 text-base sm:text-lg leading-
relaxed max-w-3xl">
                  Ви успішно увійшли до системи управління складом StockFlow.
Тут зібрано
                  основні показники, останні події та швидкі переходи до
ключових розділів.
                </p>
              </div>

              <div className="grid grid-cols-1 sm:grid-cols-2 xl:grid-cols-4
gap-4">
```

```
                <div className="rounded-[1.5rem] border border-green-100 bg-
white p-5 shadow-lg shadow-green-100/20">
                  <div className="flex items-center justify-between mb-3">
                    <span className="text-xs text-gray-500">Товари</span>
                  </div>
                  <div className="text-3xl font-black text-gray-900">
                    {dashboardStats.totalProducts}
                  </div>
                  <p className="text-sm text-gray-600 mt-2">
                    Усього позицій у системі
                  </p>
                </div>

                <div className="rounded-[1.5rem] border border-green-100 bg-
white p-5 shadow-lg shadow-green-100/20">
                  <div className="flex items-center justify-between mb-3">
                    <span className="text-xs text-gray-500">Запаси</span>
                  </div>
                  <div className="text-3xl font-black text-gray-900">
                    {dashboardStats.totalQuantity}
                  </div>
                  <p className="text-sm text-gray-600 mt-2">
                    Сумарна кількість товару
                  </p>
                </div>

                <div className="rounded-[1.5rem] border border-green-100 bg-
white p-5 shadow-lg shadow-green-100/20">
                  <div className="flex items-center justify-between mb-3">
                    <span className="text-xs text-gray-500">Накладні</span>
                  </div>
                  <div className="text-3xl font-black text-gray-900">
                    {dashboardStats.activeShipments}
                  </div>
                  <p className="text-sm text-gray-600 mt-2">
                    Активних відвантажень зараз
                  </p>
                </div>

                <div className="rounded-[1.5rem] border border-green-100 bg-
white p-5 shadow-lg shadow-green-100/20">
                  <div className="flex items-center justify-between mb-3">
                    <span className="text-xs text-gray-500">Увага</span>
                  </div>
                  <div className="text-3xl font-black text-gray-900">
                    {dashboardStats.lowStockCount}
                  </div>
                  <p className="text-sm text-gray-600 mt-2">
                    Товарів з низьким залишком
                  </p>
                </div>
              </div>

              <div className="grid grid-cols-1 xl:grid-cols-3 gap-6">
                <div className="xl:col-span-2 rounded-[2rem] border border-
green-100 bg-white shadow-xl shadow-green-100/40 p-6">
                  <h2 className="text-2xl text-gray-900 mb-4">
                    Активність складу
                  </h2>
                  <div className="h-[320px]">
                    <ResponsiveContainer width="100%" height="100%">
                      <AreaChart data={activityChartData}>
                        <CartesianGrid strokeDasharray="3 3" />
```

```
                        <XAxis dataKey="date" />
                        <YAxis />
                        <Tooltip />
                        <Area
                          type="monotone"
                          dataKey="incoming"
                          name="Надходження"
                          stroke="#22c55e"
                          fill="#bbf7d0"
                        />
                        <Area
                          type="monotone"
                          dataKey="outgoing"
                          name="Відвантаження"
                          stroke="#3b82f6"
                          fill="#bfdbfe"
                        />
                        <Area
                          type="monotone"
                          dataKey="transfer"
                          name="Переміщення"
                          stroke="#8b5cf6"
                          fill="#ddd6fe"
                        />
                      </AreaChart>
                    </ResponsiveContainer>
                  </div>
                </div>

                <div className="rounded-[2rem] border border-green-100 bg-
white shadow-xl shadow-green-100/40 p-6">
                  <h2 className="text-2xl  text-gray-900 mb-4">
                    Швидкі дії
                  </h2>

                  <div className="space-y-3">
                    {quickLinks.map((item) => (
                      <Link
                        key={item.to}
                        to={item.to}
                        className="flex items-start gap-3 rounded-2xl border
border-green-100 bg-[#fcfffd] p-4 hover:bg-green-50 transition-all duration-
300"
                      >
                        <div className="w-11 h-11 rounded-xl bg-green-100
text-green-600 flex items-center justify-center shrink-0">
                          {item.icon}
                        </div>
                        <div>
                          <div className="font-bold text-gray-
900">{item.title}</div>
                          <div className="text-sm text-gray-600 mt-
1">{item.text}</div>
                        </div>
                      </Link>
                    ))}
                  </div>
                </div>
              </div>

              <div className="grid grid-cols-1 xl:grid-cols-2 gap-6">
                <div className="rounded-[2rem] border border-green-100 bg-
white shadow-xl shadow-green-100/40 p-6">
```

```
                  <h2 className="text-2xl  text-gray-900 mb-4">
                    Останні рухи товарів
                  </h2>

                  {recentMovements.length === 0 ? (
                    <div className="text-gray-600">
                      Поки що немає жодного запису про рух товарів.
                    </div>
                  ) : (
                    <div className="space-y-3">
                      {recentMovements.map((item) => (
                        <div
                          key={item.id}
                          className="rounded-2xl border border-green-100 bg-
[#fcfffd] p-4"
                        >
                          <div className="flex items-start justify-between
gap-3">
                            <div>
                              <div className="font-bold text-gray-900">
                                {item.productName}
                              </div>
                              <div className="text-sm text-gray-600 mt-1">
                                Артикул: {item.productSku}
                              </div>
                            </div>

                            <div
                              className={`inline-flex rounded-full border px-3
py-1 text-xs font-semibold ${getMovementBadgeClass(item.type)}`}
                            >
                              {item.typeLabel}
                            </div>
                          </div>

                          <div className="mt-3 text-sm text-gray-700">
                            Кількість: <span className="font-
semibold">{item.quantity}</span>
                          </div>

                          <div className="text-sm text-gray-500 mt-1">
                            {formatDateTime(item.createdAt)}
                          </div>
                        </div>
                      ))}
                    </div>
                  )}
                </div>

                <div className="rounded-[2rem] border border-green-100 bg-
white shadow-xl shadow-green-100/40 p-6">
                  <h2 className="text-2xl  text-gray-900 mb-4">
                    Останні накладні
                  </h2>

                  {recentShipments.length === 0 ? (
                    <div className="text-gray-600">
                      Поки що немає жодної накладної.
                    </div>
                  ) : (
                    <div className="space-y-3">
                      {recentShipments.map((item) => (
                        <div
```

```
                          key={item.id}
                          className="rounded-2xl border border-green-100 bg-
[#fcfffd] p-4"
                        >
                          <div className="flex items-start justify-between
gap-3">
                            <div>
                              <div className="font-bold text-gray-900">
                                {item.invoiceNumber}
                              </div>
                              <div className="text-sm text-gray-600 mt-1">
                                Одержувач: {item.customerName}
                              </div>
                            </div>

                            <div
                              className={`inline-flex rounded-full border px-3
py-1 text-xs font-semibold ${getShipmentBadgeClass(item.status)}`}
                            >
                              {item.statusLabel}
                            </div>
                          </div>

                          {item.note && (
                            <div className="text-sm text-gray-600 mt-
3">{item.note}</div>
                          )}

                          <div className="text-sm text-gray-500 mt-2">
                            {formatDateTime(item.createdAt)}
                          </div>
                        </div>
                      ))}
                    </div>
                  )}
                </div>
              </div>

              <div className="grid grid-cols-1 xl:grid-cols-2 gap-6">
                <div className="rounded-[2rem] border border-red-100 bg-white
shadow-xl shadow-red-100/30 p-6">
                  <h2 className="text-2xl  text-gray-900 mb-4">
                    Критичні залишки
                  </h2>

                  {lowStockProducts.length === 0 ? (
                    <div className="text-gray-600">
                      Усі товари мають достатній залишок.
                    </div>
                  ) : (
                    <div className="space-y-3">
                      {lowStockProducts.map((item) => (
                        <div
                          key={item.id}
                          className="rounded-2xl border border-red-100 bg-red-
50/40 p-4"
                        >
                          <div className="font-bold text-gray-
900">{item.name}</div>
                          <div className="text-sm text-gray-700 mt-1">
                            Поточний залишок:{" "}
                            <span className="font-semibold text-red-700">
                              {item.quantity} {item.unit}
```

```
                        </span>
                      </div>
                      <div className="text-sm text-gray-600 mt-1">
                        Мінімум: {item.minStock} {item.unit}
                      </div>
                    </div>
                  ))}
                </div>
              )}
            </div>

            <div className="rounded-[2rem] border border-green-100 bg-
white shadow-xl shadow-green-100/40 p-6">
              <h2 className="text-2xl  text-gray-900 mb-4">
                Стан системи
              </h2>

              <div className="grid grid-cols-1 sm:grid-cols-2 gap-4">
                <div className="rounded-2xl border border-green-100 bg-
[#fcfffd] p-4">
                  <div className="text-sm text-gray-500 mb-
1">Категорії</div>
                  <div className="text-2xl font-black text-gray-900">
                    {dashboardStats.totalCategories}
                  </div>
                </div>

                <div className="rounded-2xl border border-green-100 bg-
[#fcfffd] p-4">
                  <div className="text-sm text-gray-500 mb-
1">Постачальники</div>
                  <div className="text-2xl font-black text-gray-900">
                    {dashboardStats.totalSuppliers}
                  </div>
                </div>

                <div className="rounded-2xl border border-green-100 bg-
[#fcfffd] p-4">
                  <div className="text-sm text-gray-500 mb-1">Вільні
комірки</div>
                  <div className="text-2xl font-black text-green-700">
                    {dashboardStats.freeCells}
                  </div>
                </div>

                <div className="rounded-2xl border border-green-100 bg-
[#fcfffd] p-4">
                  <div className="text-sm text-gray-500 mb-1">Зайняті
комірки</div>
                  <div className="text-2xl font-black text-red-700">
                    {dashboardStats.occupiedCells}
                  </div>
                </div>
              </div>

              <div className="mt-5 rounded-2xl border border-green-100 bg-
green-50/50 p-4 text-gray-700 leading-relaxed">
                Система готова до подальшої роботи: можна швидко перейти
до керування
                товарами, оформлення відвантажень, перегляду аналітики або
руху товарів.
              </div>
            </div>
```

```
            </div>
          </div>
        </main>
      );
    }
```

Сторінка реєстрації, компонент Register.jsx:

```
    import { useState } from "react";
    import { createUserWithEmailAndPassword, updateProfile } from
"firebase/auth";
    import { auth, db } from "../firebase";
    import { doc, setDoc, serverTimestamp } from "firebase/firestore";
    import { useNavigate, Link } from "react-router-dom";
    import { UserPlus } from "lucide-react";

    export default function Register() {
      const [form, setForm] = useState({
        fullName: "",
        email: "",
        password: "",
      });
      const [error, setError] = useState("");
      const navigate = useNavigate();

      const handleChange = (e) => {
        setForm({ ...form, [e.target.name]: e.target.value });
        if (error) setError("");
      };

      const handleSubmit = async (e) => {
        e.preventDefault();
        setError("");

        if (!form.fullName.trim() || !form.email.trim() ||
!form.password.trim()) {
          setError("Будь ласка, заповніть усі поля");
          return;
        }

        if (form.password.trim().length < 6) {
          setError("Пароль повинен містити щонайменше 6 символів");
          return;
        }

        try {
          const userCredential = await createUserWithEmailAndPassword(
            auth,
            form.email.trim(),
            form.password.trim()
          );

          const user = userCredential.user;

          await updateProfile(user, {
            displayName: form.fullName.trim(),
          });

          await setDoc(doc(db, "users", user.uid), {
            fullName: form.fullName.trim(),
            email: form.email.trim(),
            createdAt: serverTimestamp(),
```

```
            });

            navigate("/home");
          } catch (err) {
            setError("Помилка реєстрації. Перевірте дані або спробуйте іншу
пошту.");
          }
        };

        return (
          <div className="min-h-screen flex items-center justify-center bg-
[#f6faf7] px-4 py-10">
            <div className="w-full max-w-md rounded-[2rem] border border-
green-100 bg-white shadow-xl shadow-green-100/40 p-8 sm:p-10">
              <div className="flex flex-col items-center text-center mb-8">
                <div className="w-20 h-20 rounded-full bg-green-100 text-
green-600 flex items-center justify-center mb-4 shadow-sm">
                  <UserPlus size={34} />
                </div>

                <h2 className="text-3xl font-black text-gray-900 mb-2">
                  Реєстрація
                </h2>

                <p className="text-gray-500 text-sm sm:text-base">
                  Створіть обліковий запис для роботи зі складською системою
                </p>
              </div>

              <form onSubmit={handleSubmit} className="flex flex-col gap-5">
                <div>
                  <label className="block text-sm font-semibold text-gray-700
mb-2">
                    Прізвище та ім’я
                  </label>
                  <input
                    type="text"
                    name="fullName"
                    placeholder="Введіть прізвище та ім’я"
                    value={form.fullName}
                    onChange={handleChange}
                    className="w-full rounded-2xl border border-gray-200 bg-
white px-4 py-3 text-gray-800 placeholder:text-gray-400 focus:outline-none
focus:ring-2 focus:ring-green-500 focus:border-green-500 transition-all
duration-300"
                    autoComplete="name"
                  />
                </div>

                <div>
                  <label className="block text-sm font-semibold text-gray-700
mb-2">
                    Електронна пошта
                  </label>
                  <input
                    type="email"
                    name="email"
                    placeholder="Введіть вашу електронну пошту"
                    value={form.email}
                    onChange={handleChange}
                    className="w-full rounded-2xl border border-gray-200 bg-
white px-4 py-3 text-gray-800 placeholder:text-gray-400 focus:outline-none
```

```
focus:ring-2 focus:ring-green-500 focus:border-green-500 transition-all
duration-300"
                    autoComplete="email"
                  />
                </div>

                <div>
                  <label className="block text-sm font-semibold text-gray-700
mb-2">
                    Пароль
                  </label>
                  <input
                    type="password"
                    name="password"
                    placeholder="Створіть пароль"
                    value={form.password}
                    onChange={handleChange}
                    className="w-full rounded-2xl border border-gray-200 bg-
white px-4 py-3 text-gray-800 placeholder:text-gray-400 focus:outline-none
focus:ring-2 focus:ring-green-500 focus:border-green-500 transition-all
duration-300"
                    autoComplete="new-password"
                  />
                </div>

                {error && (
                  <p className="text-red-600 text-sm text-center -mt-
1">{error}</p>
                )}

                <button
                  type="submit"
                  className="mt-2 rounded-2xl bg-green-500 px-6 py-3.5 text-
white font-bold hover:bg-green-600 transition-all duration-300 hover:-
translate-y-0.5"
                >
                  Зареєструватися
                </button>
              </form>

              <p className="text-center text-sm text-gray-500 mt-6">
                Уже маєте профіль?{" "}
                <Link
                  to="/login"
                  className="text-green-600 font-semibold hover:text-green-700
hover:underline transition"
                >
                  Увійти
                </Link>
              </p>
            </div>
          </div>
        );
      }
```

Сторінка входу до системи, компонент Login.jsx:

```
      import { useState } from "react";
      import { signInWithEmailAndPassword } from "firebase/auth";
      import { auth } from "../firebase";
      import { useNavigate, Link } from "react-router-dom";
      import { LogIn } from "lucide-react";
```

```
export default function Login() {
  const [form, setForm] = useState({ email: "", password: "" });
  const [error, setError] = useState("");
  const navigate = useNavigate();

  const handleChange = (e) => {
    setForm({ ...form, [e.target.name]: e.target.value });
    if (error) setError("");
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    setError("");

    if (!form.email.trim() || !form.password.trim()) {
      setError("Будь ласка, заповніть усі поля");
      return;
    }

    try {
      await signInWithEmailAndPassword(
        auth,
        form.email.trim(),
        form.password.trim()
      );
      navigate("/home");
    } catch (err) {
      setError("Невірна електронна пошта або пароль");
    }
  };

  return (
    <div className="min-h-screen flex items-center justify-center bg-
[#f6faf7] px-4 py-10">
      <div className="w-full max-w-md rounded-[2rem] border border-
green-100 bg-white shadow-xl shadow-green-100/40 p-8 sm:p-10">
        <div className="flex flex-col items-center text-center mb-8">
          <div className="w-20 h-20 rounded-full bg-green-100 text-
green-600 flex items-center justify-center mb-4 shadow-sm">
            <LogIn size={34} />
          </div>

          <h2 className="text-3xl font-black text-gray-900 mb-2">
            Вхід до системи
          </h2>

          <p className="text-gray-500 text-sm sm:text-base">
            Увійдіть, щоб продовжити роботу зі складською системою
          </p>
        </div>

        <form onSubmit={handleSubmit} className="flex flex-col gap-5">
          <div>
            <label className="block text-sm font-semibold text-gray-700
mb-2">
              Електронна пошта
            </label>
            <input
              type="email"
              name="email"
              placeholder="Введіть вашу електронну пошту"
              value={form.email}
```

```
                    onChange={handleChange}
                    className="w-full rounded-2xl border border-gray-200 bg-
white px-4 py-3 text-gray-800 placeholder:text-gray-400 focus:outline-none
focus:ring-2 focus:ring-green-500 focus:border-green-500 transition-all
duration-300"
                    autoComplete="email"
                  />
                </div>

                <div>
                  <label className="block text-sm font-semibold text-gray-700
mb-2">
                    Пароль
                  </label>
                  <input
                    type="password"
                    name="password"
                    placeholder="Введіть пароль"
                    value={form.password}
                    onChange={handleChange}
                    className="w-full rounded-2xl border border-gray-200 bg-
white px-4 py-3 text-gray-800 placeholder:text-gray-400 focus:outline-none
focus:ring-2 focus:ring-green-500 focus:border-green-500 transition-all
duration-300"
                    autoComplete="current-password"
                  />
                </div>

                {error && (
                  <p className="text-red-600 text-sm text-center -mt-
1">{error}</p>
                )}

                <button
                  type="submit"
                  className="mt-2 rounded-2xl bg-green-500 px-6 py-3.5 text-
white font-bold hover:bg-green-600 transition-all duration-300 hover:-
translate-y-0.5"
                >
                  Увійти
                </button>
              </form>

              <p className="text-center text-sm text-gray-500 mt-6">
                Немає профілю?{" "}
                <Link
                  to="/register"
                  className="text-green-600 font-semibold hover:text-green-700
hover:underline transition"
                >
                  Зареєструйтесь
                </Link>
              </p>
            </div>
          </div>
        );
      }
```

Головна сторінка після входу, компонент Home.jsx:

```
      import React  from "react";
      import { Link } from "react-router-dom";
```

```
      import {
        Package,
        Truck,
        Warehouse,
        BarChart3,
        ShieldCheck,
        Smartphone,
        Boxes,
        MapPinned,
        ClipboardList,
        ArrowRight,
      } from "lucide-react";

      export default function Home() {



        const advantages = [
          {
            icon: <Package size={28} />,
            title: "Контроль запасів",
            text: "Швидкий перегляд залишків товарів і контроль критичних
позицій у реальному часі.",
          },
          {
            icon: <Truck size={28} />,
            title: "Облік постачань",
            text: "Фіксація надходжень, відвантажень і руху товарів для
повного контролю операцій.",
          },
          {
            icon: <BarChart3 size={28} />,
            title: "Аналітика попиту",
            text: "Оцінка руху товарів і базове прогнозування попиту на основі
накопичених даних.",
          },
          {
            icon: <Warehouse size={28} />,
            title: "Структура складу",
            text: "Облік місць зберігання для швидкого пошуку товарів і кращої
організації простору.",
          },
          {
            icon: <ShieldCheck size={28} />,
            title: "Надійне збереження",
            text: "Безпечне зберігання даних та контроль доступу через
авторизацію користувача.",
          },
          {
            icon: <Smartphone size={28} />,
            title: "Адаптивний інтерфейс",
            text: "Зручна робота із системою на комп’ютері, планшеті та
смартфоні.",
          },
        ];

        const systemCards = [
          {
            icon: <Boxes size={32} />,
            title: "Облік товарів",
            text: "Додавання, редагування та перегляд товарів із прив’язкою до
категорій та постачальників.",
          },
```

```
          {
            icon: <MapPinned size={32} />,
            title: "Місця зберігання",
            text: "Організація складу за зонами, стелажами або комірками для
спрощення логістики.",
          },
          {
            icon: <ClipboardList size={32} />,
            title: "Рух і відвантаження",
            text: "Повна історія транзакцій: від надходження на склад до
передачі клієнту.",
          },
        ];

        return (
          <main className="bg-[#fdfdfd] text-[#1f2937] overflow-hidden">
            <section className="relative min-h-screen flex items-center pt-4
sm:pt-6 lg:pt-10">
        <div className="absolute top-0 -left-20 w-72 h-72 sm:w-96 sm:h-96 bg-
green-100 rounded-full mix-blend-multiply filter blur-3xl opacity-60 animate-
blob"></div>
        <div className="absolute top-0 -right-20 w-72 h-72 sm:w-96 sm:h-96 bg-
blue-100 rounded-full mix-blend-multiply filter blur-3xl opacity-60 animate-
blob animation-delay-2000"></div>

        <div
          className="absolute inset-0 z-0 bg-cover bg-center md:bg-fixed"
          style={{ backgroundImage: "url('/bg.png')" }}
        >
          <div className="absolute inset-0 bg-gradient-to-r from-gray-900/90
via-gray-900/75 to-gray-900/40 md:to-transparent"></div>
        </div>

        <div className="relative z-10 max-w-7xl mx-auto px-4 sm:px-6 lg:px-12
w-full">
          <div className="max-w-3xl">
            <span className="inline-block px-3 py-1.5 sm:px-4 mb-5 text-[11px]
sm:text-sm font-medium tracking-wider text-green-400 uppercase bg-green-
400/10 rounded-full border border-green-400/20">
              Сучасна система управління складом
            </span>

            <h1 className="text-4xl sm:text-6xl lg:text-7xl font-black text-
white leading-tight mb-5">
              Stock<span className="text-green-500">Flow</span>
            </h1>

            <p className="text-base sm:text-xl lg:text-2xl text-gray-300 font-
light leading-relaxed mb-8 max-w-2xl">
              Розумна екосистема для автоматизації складу. Контролюйте кожен
артикул,
              оптимізуйте логістику та масштабуйте бізнес швидше.
            </p>

            <div className="flex flex-col sm:flex-row sm:flex-wrap gap-4
sm:gap-5 w-full sm:w-auto">
              <Link
                to="/register"
                className="group inline-flex items-center justify-center
rounded-2xl bg-green-500 px-6 sm:px-8 py-3.5 sm:py-4 text-white font-bold
hover:bg-green-600 transition-all duration-300 hover:-translate-y-1 w-full
sm:w-auto"
              >
```

```
                Почати безкоштовно
                <ArrowRight className="ml-2 w-5 h-5 group-hover:translate-x-1
transition-transform" />
              </Link>

              <Link
                to="/login"
                className="inline-flex items-center justify-center rounded-2xl
border border-white/30 backdrop-blur-md px-6 sm:px-8 py-3.5 sm:py-4 text-
white font-semibold hover:bg-white/10 transition-all duration-300 w-full
sm:w-auto"
              >
                Увійти в кабінет
              </Link>
            </div>
          </div>
        </div>
      </section>

          <section className="relative -mt-20 z-20 pb-0">
        <div className="max-w-7xl mx-auto px-6 relative z-20">
          <div className="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 gap-
8">
            {advantages.map((item, index) => (
              <div
                key={index}
                className="group rounded-3xl border border-white/60 bg-white
p-8 shadow-md hover:shadow-xl hover:-translate-y-2 transition-all duration-
300"
              >
                <div className="mb-6 flex items-center gap-4">
                  <div className="flex h-14 w-14 items-center justify-center
rounded-2xl bg-green-100 text-green-600 transition-all duration-300 group-
hover:bg-green-500 group-hover:text-white">
                    {item.icon}
                  </div>
                  <h3 className="text-xl font-bold text-gray-800 leading-snug
transition-colors duration-300 group-hover:text-green-600">
                    {item.title}
                  </h3>
                </div>

                <p className="text-gray-600 text-[15px] leading-7">
                  {item.text}
                </p>

                <div className="mt-6 h-1 w-16 rounded-full bg-green-500/20
transition-all duration-300 group-hover:w-24 group-hover:bg-green-500"></div>
              </div>
            ))}
          </div>
        </div>

        <div
          className="relative z-10 -mt-16 h-32 sm:h-40 lg:h-52 bg-cover bg-
center md:bg-fixed"
          style={{ backgroundImage: "url('/bg.png')" }}
        >
          <div className="absolute inset-0 bg-gradient-to-b from-transparent
via-gray-900/20 to-gray-50/40"></div>
        </div>
      </section>
```

```
        <section className="py-24 bg-gray-50/40">
        <div className="max-w-7xl mx-auto px-6 text-center">
          <div className="mb-16">
            <h2 className="text-4xl md:text-5xl font-black text-gray-900 mb-6
tracking-tight">
              Єдина консоль управління
            </h2>
            <p className="max-w-2xl mx-auto text-gray-600 text-lg leading-
relaxed">
              Усі ключові інструменти для ефективного керування складом в
одному зручному середовищі.
            </p>
            <div className="w-24 h-1.5 bg-green-500 mx-auto rounded-full mt-
6"></div>
          </div>

          <div className="relative -mt-12 grid grid-cols-1 lg:grid-cols-3 gap-
8 z-20">
            {systemCards.map((card, index) => (
              <div
                key={index}
                className="group rounded-3xl border border-white/60 bg-white
p-10 shadow-md hover:shadow-xl hover:-translate-y-2 transition-all duration-
300"
              >
                <div className="w-16 h-16 rounded-full bg-green-100 text-
green-600 flex items-center justify-center mb-8 mx-auto transition-all
duration-300 group-hover:bg-green-500 group-hover:text-white group-
hover:scale-110">
                  {card.icon}
                </div>

                <h3 className="text-2xl font-extrabold text-gray-800 mb-4
tracking-tight group-hover:text-green-600 transition-colors duration-300">
                  {card.title}
                </h3>

                <p className="text-gray-600 leading-7 text-[15px]">
                  {card.text}
                </p>

                <div className="mt-0 h-1 w-16 mx-auto rounded-full bg-green-
500/20 transition-all duration-300 group-hover:w-24 group-hover:bg-green-
500"></div>
              </div>
            ))}
          </div>
        </div>
      </section>

           <section className="py-20 px-6">
        <div className="max-w-5xl mx-auto rounded-[2.5rem] bg-white border
border-white/60 shadow-xl shadow-gray-200/50 p-10 md:p-14 text-center
relative overflow-hidden">
          <div className="absolute -top-10 -left-10 w-40 h-40 bg-green-100
rounded-full blur-3xl opacity-60 pointer-events-none"></div>
          <div className="absolute -bottom-10 -right-10 w-40 h-40 bg-green-50
rounded-full blur-3xl opacity-70 pointer-events-none"></div>

          <div className="relative z-10">
            <h2 className="text-3xl md:text-4xl font-black text-gray-900 mb-5
tracking-tight">
              Готові навести порядок на складі?
```

```
            </h2>

            <p className="max-w-2xl mx-auto text-gray-600 text-lg leading-
relaxed mb-8">
              Почніть користуватися системою вже зараз і отримайте зручний
контроль над товарами, залишками та складськими процесами.
            </p>

            <Link
              to="/register"
              className="inline-flex items-center justify-center rounded-2xl
bg-green-500 px-10 py-4 text-white font-bold hover:bg-green-600 transition-
all duration-300 hover:-translate-y-1"
            >
              Спробувати StockFlow
            </Link>
          </div>
        </div>
      </section>
          </main>
        );
      }
```

Сторінка товарів, компонент Products.jsx:

```
import { useCallback, useEffect, useMemo, useState } from "react";
import { Package, Search, Plus, Trash2, Pencil } from "lucide-react";
import { useAuth } from "../context/AuthContext";
import EditProductModal from "../components/EditProductModal";
import { db } from "../firebase";
import {
  collection,
  doc,
  getDoc,
  getDocs,
  query,
  serverTimestamp,
  where,
  writeBatch,
} from "firebase/firestore";
import Swal from "sweetalert2";

export default function Products() {
  const { user } = useAuth();
  const [editingProduct, setEditingProduct] = useState(null);
  const [isEditOpen, setIsEditOpen] = useState(false);
  const [form, setForm] = useState({
    name: "",
    sku: "",
    categoryId: "",
    supplierId: "",
    unit: "шт",
    purchasePrice: "",
    salePrice: "",
    quantity: "",
    minStock: "",
    maxStock: "",
    rackId: "",
    cellCodes: [],
    barcode: "",
  });
```

```
        const [searchTerm, setSearchTerm] = useState("");
        const [categoryFilter, setCategoryFilter] = useState("");
        const [supplierFilter, setSupplierFilter] = useState("");
        const [products, setProducts] = useState([]);
        const [categories, setCategories] = useState([]);
        const [suppliers, setSuppliers] = useState([]);
        const [racks, setRacks] = useState([]);
        const [loading, setLoading] = useState(true);
        const handleOpenEdit = (product) => {
          setEditingProduct(product);
          setIsEditOpen(true);
        };

        const handleCloseEdit = () => {
          setEditingProduct(null);
          setIsEditOpen(false);
        };
        const toast = useCallback((icon, title) => {
          Swal.fire({
            toast: true,
            position: "top-end",
            icon,
            title,
            showConfirmButton: false,
            timer: 3500,
            timerProgressBar: true,
          });
        }, []);

        const formatDate = (timestamp) => {
          if (!timestamp?.toDate) return "—";
          return timestamp.toDate().toLocaleDateString("uk-UA", {
            day: "2-digit",
            month: "2-digit",
            year: "numeric",
          });
        };

        const sortByCreatedAtDesc = (items) => {
          return [...items].sort((a, b) => {
            const aSec = a.createdAt?.seconds || 0;
            const bSec = b.createdAt?.seconds || 0;
            return bSec - aSec;
          });
        };

        const fetchData = useCallback(async () => {
          if (!user) return;

          setLoading(true);

          try {
            const [
              productsSnapshot,
              categoriesSnapshot,
              suppliersSnapshot,
              racksSnapshot,
            ] = await Promise.all([
              getDocs(
                query(collection(db, "products"), where("userId", "==",
user.uid)),
              ),
              getDocs(
```

```
              query(collection(db, "categories"), where("userId", "==",
user.uid)),
            ),
            getDocs(
              query(collection(db, "suppliers"), where("userId", "==",
user.uid)),
            ),
            getDocs(
              query(
                collection(db, "storageLocations"),
                where("userId", "==", user.uid),
              ),
            ),
          ]);

          const productsData = productsSnapshot.docs.map((item) => ({
            id: item.id,
            ...item.data(),
          }));

          const categoriesData = categoriesSnapshot.docs.map((item) => ({
            id: item.id,
            ...item.data(),
          }));

          const suppliersData = suppliersSnapshot.docs.map((item) => ({
            id: item.id,
            ...item.data(),
          }));

          const racksData = racksSnapshot.docs.map((item) => ({
            id: item.id,
            ...item.data(),
          }));

          setProducts(sortByCreatedAtDesc(productsData));
          setCategories(sortByCreatedAtDesc(categoriesData));
          setSuppliers(sortByCreatedAtDesc(suppliersData));
          setRacks(sortByCreatedAtDesc(racksData));
        } catch (error) {
          toast("error", "Не вдалося завантажити дані");
        } finally {
          setLoading(false);
        }
      }, [user, toast]);

      useEffect(() => {
        fetchData();
      }, [fetchData]);

      const prerequisitesReady =
        categories.length > 0 && suppliers.length > 0 && racks.length > 0;

      const selectedRack = useMemo(() => {
        return racks.find((rack) => rack.id === form.rackId) || null;
      }, [racks, form.rackId]);

      const availableCells = useMemo(() => {
        if (!selectedRack) return [];
        return selectedRack.cells || [];
      }, [selectedRack]);

      const handleChange = (e) => {
```

```
  const { name, value } = e.target;

  if (name === "rackId") {
    setForm((prev) => ({
      ...prev,
      rackId: value,
      cellCodes: [],
    }));
    return;
  }

  setForm((prev) => ({
    ...prev,
    [name]: value,
  }));
};

const handleToggleCell = (code, status) => {
  if (status === "occupied") return;

  setForm((prev) => {
    const exists = prev.cellCodes.includes(code);

    return {
      ...prev,
      cellCodes: exists
        ? prev.cellCodes.filter((item) => item !== code)
        : [...prev.cellCodes, code],
    };
  });
};

const resetForm = () => {
  setForm({
    name: "",
    sku: "",
    categoryId: "",
    supplierId: "",
    unit: "шт",
    purchasePrice: "",
    salePrice: "",
    quantity: "",
    minStock: "",
    maxStock: "",
    rackId: "",
    cellCodes: [],
    barcode: "",
  });
};

const handleAddProduct = async () => {
  if (!user) {
    toast("error", "Користувача не знайдено");
    return;
  }

  if (!prerequisitesReady) {
    toast(
      "error",
      "Спочатку створіть категорію, постачальника і місце зберігання",
    );
    return;
  }
```

```
            const payload = {
              name: form.name.trim(),
              sku: form.sku.trim(),
              categoryId: form.categoryId,
              supplierId: form.supplierId,
              unit: form.unit,
              purchasePrice: Number(form.purchasePrice),
              salePrice: Number(form.salePrice),
              quantity: Number(form.quantity),
              minStock: Number(form.minStock),
              maxStock: Number(form.maxStock),
              rackId: form.rackId,
              cellCodes: form.cellCodes,
              barcode: form.barcode.trim(),
            };

            if (
              !payload.name ||
              !payload.sku ||
              !payload.categoryId ||
              !payload.supplierId ||
              !payload.unit ||
              !form.purchasePrice ||
              !form.salePrice ||
              !form.quantity ||
              !form.minStock ||
              !form.maxStock ||
              !payload.rackId ||
              payload.cellCodes.length === 0
            ) {
              toast("error", "Заповніть усі обов’язкові поля");
              return;
            }

            if (
              Number.isNaN(payload.purchasePrice) ||
              Number.isNaN(payload.salePrice) ||
              Number.isNaN(payload.quantity) ||
              Number.isNaN(payload.minStock) ||
              Number.isNaN(payload.maxStock)
            ) {
              toast("error", "Перевірте числові поля");
              return;
            }

            if (payload.minStock <= 0) {
              toast("error", "Мінімальний залишок не може бути 0");
              return;
            }

            if (payload.quantity < 0) {
              toast("error", "Поточна кількість не може бути від’ємною");
              return;
            }

            if (payload.maxStock < payload.minStock) {
              toast("error", "Максимальний запас має бути не меншим за
мінімальний");
              return;
            }

            const skuExists = products.some(
```

```
            (item) => item.sku?.toLowerCase() === payload.sku.toLowerCase(),
          );

          if (skuExists) {
            toast("error", "Товар з таким артикулом уже існує");
            return;
          }

          const category = categories.find((item) => item.id ===
payload.categoryId);
          const supplier = suppliers.find((item) => item.id ===
payload.supplierId);
          const rack = racks.find((item) => item.id === payload.rackId);

          if (!category || !supplier || !rack) {
            toast("error", "Не вдалося знайти пов’язані дані");
            return;
          }

          try {
            const rackRef = doc(db, "storageLocations", rack.id);
            const rackSnap = await getDoc(rackRef);

            if (!rackSnap.exists()) {
              toast("error", "Стелаж не знайдено");
              return;
            }

            const latestRack = rackSnap.data();
            const latestCells = latestRack.cells || [];

            const busySelected = payload.cellCodes.some((code) => {
              const cell = latestCells.find((item) => item.code === code);
              return !cell || cell.status === "occupied";
            });

            if (busySelected) {
              toast("error", "Одна або кілька обраних комірок уже зайняті");
              fetchData();
              return;
            }

            const updatedCells = latestCells.map((cell) =>
              payload.cellCodes.includes(cell.code)
                ? { ...cell, status: "occupied" }
                : cell,
            );

            const productRef = doc(collection(db, "products"));
            const batch = writeBatch(db);

            batch.set(productRef, {
              userId: user.uid,
              name: payload.name,
              sku: payload.sku,
              categoryId: category.id,
              categoryName: category.name,
              supplierId: supplier.id,
              supplierName: supplier.name,
              unit: payload.unit,
              purchasePrice: payload.purchasePrice,
              salePrice: payload.salePrice,
              quantity: payload.quantity,
```

```
              minStock: payload.minStock,
              maxStock: payload.maxStock,
              rackId: rack.id,
              rackName: rack.name,
              cellCodes: payload.cellCodes,
              storageLocation: `${rack.name}: ${payload.cellCodes.join(",
")}`,
              barcode: payload.barcode,
              status: "Додано в систему",
              createdAt: serverTimestamp(),
            });

            batch.update(rackRef, {
              cells: updatedCells,
            });

            await batch.commit();

            resetForm();
            toast("success", "Товар успішно додано");
            fetchData();
          } catch (error) {
            toast("error", "Не вдалося додати товар");
          }
        };

        const handleDeleteProduct = async (product) => {
          const result = await Swal.fire({
            title: "Ви впевнені?",
            text: "Товар буде видалено, а його комірки на складі стануть
вільними.",
            icon: "warning",
            showCancelButton: true,
            confirmButtonText: "Так, видалити",
            cancelButtonText: "Скасувати",
            confirmButtonColor: "#ef4444",
            cancelButtonColor: "#9ca3af",
            reverseButtons: true,
          });

          if (!result.isConfirmed) return;

          try {
            const batch = writeBatch(db);
            const productRef = doc(db, "products", product.id);

            batch.delete(productRef);

            if (
              product.rackId &&
              Array.isArray(product.cellCodes) &&
              product.cellCodes.length > 0
            ) {
              const rackRef = doc(db, "storageLocations", product.rackId);
              const rackSnap = await getDoc(rackRef);

              if (rackSnap.exists()) {
                const rackData = rackSnap.data();
                const updatedCells = (rackData.cells || []).map((cell) =>
                  product.cellCodes.includes(cell.code)
                    ? { ...cell, status: "free" }
                    : cell,
                );
```

```
            batch.update(rackRef, {
              cells: updatedCells,
            });
          }
        }

        await batch.commit();

        toast("success", "Товар видалено");
        fetchData();
      } catch (error) {
        toast("error", "Не вдалося видалити товар");
      }
    };

    const filteredProducts = useMemo(() => {
      const value = searchTerm.toLowerCase().trim();

      return products.filter((item) => {
        const matchesSearch =
          item.name?.toLowerCase().includes(value) ||
          item.sku?.toLowerCase().includes(value) ||
          item.barcode?.toLowerCase().includes(value) ||
          item.categoryName?.toLowerCase().includes(value) ||
          item.supplierName?.toLowerCase().includes(value);

        const matchesCategory = categoryFilter
          ? item.categoryId === categoryFilter
          : true;

        const matchesSupplier = supplierFilter
          ? item.supplierId === supplierFilter
          : true;

        return matchesSearch && matchesCategory && matchesSupplier;
      });
    }, [products, searchTerm, categoryFilter, supplierFilter]);

    return (
      <main className="min-h-screen bg-[#f6faf7] px-4 py-10">
        <div className="max-w-7xl mx-auto">
          <div className="rounded-[2rem] border border-green-100 bg-white
shadow-xl shadow-green-100/40 p-6 sm:p-8 md:p-10">
            <div className="flex items-center gap-4 mb-8">
              <div className="w-16 h-16 rounded-full bg-green-100 text-
green-600 flex items-center justify-center shrink-0">
                <Package size={30} />
              </div>

              <div>
                <h1 className="text-3xl sm:text-4xl font-black text-gray-
900 mb-2">
                  Товари
                </h1>
                <p className="text-gray-600 text-base sm:text-lg leading-
relaxed">
                  Додавайте товари, обирайте категорію, постачальника та
місце
                  зберігання, а також контролюйте залишки і заповнення
комірок.
                </p>
              </div>
```

```
              </div>

              {!prerequisitesReady && (
                <div className="rounded-[1.5rem] border border-amber-200 bg-
amber-50 px-5 py-5 mb-6">
                  <h3 className="text-xl font-bold text-gray-900 mb-2">
                    Недостатньо даних для додавання товару
                  </h3>
                  <p className="text-gray-700 leading-relaxed">
                    Щоб додати товар, необхідно створити по одному запису
для:
                    Категорій, Постачальників та Місць зберігання.
                  </p>
                </div>
              )}

              <div className="rounded-[1.5rem] border border-green-100 bg-
[#fcfffd] p-5 sm:p-6 shadow-sm mb-6">
                <div className="grid grid-cols-1 md:grid-cols-2 xl:grid-
cols-3 gap-4 mb-4">
                  <input
                    type="text"
                    name="name"
                    value={form.name}
                    onChange={handleChange}
                    placeholder="Назва товару*"
                    className="w-full rounded-2xl border border-gray-200 bg-
white px-4 py-3 text-gray-800 placeholder:text-gray-400 focus:outline-none
focus:ring-2 focus:ring-green-500 focus:border-green-500 transition-all
duration-300"
                    disabled={!prerequisitesReady}
                  />

                  <input
                    type="text"
                    name="sku"
                    value={form.sku}
                    onChange={handleChange}
                    placeholder="Артикул або код товару*"
                    className="w-full rounded-2xl border border-gray-200 bg-
white px-4 py-3 text-gray-800 placeholder:text-gray-400 focus:outline-none
focus:ring-2 focus:ring-green-500 focus:border-green-500 transition-all
duration-300"
                    disabled={!prerequisitesReady}
                  />

                  <select
                    name="categoryId"
                    value={form.categoryId}
                    onChange={handleChange}
                    className="w-full rounded-2xl border border-gray-200 bg-
white px-4 py-3 text-gray-800 focus:outline-none focus:ring-2 focus:ring-
green-500 focus:border-green-500 transition-all duration-300"
                    disabled={!prerequisitesReady}
                  >
                    <option value="">Оберіть категорію*</option>
                    {categories.map((item) => (
                      <option key={item.id} value={item.id}>
                        {item.name}
                      </option>
                    ))}
                  </select>
```

```
                    <select
                      name="supplierId"
                      value={form.supplierId}
                      onChange={handleChange}
                      className="w-full rounded-2xl border border-gray-200 bg-
white px-4 py-3 text-gray-800 focus:outline-none focus:ring-2 focus:ring-
green-500 focus:border-green-500 transition-all duration-300"
                      disabled={!prerequisitesReady}
                    >
                      <option value="">Оберіть постачальника*</option>
                      {suppliers.map((item) => (
                        <option key={item.id} value={item.id}>
                          {item.name}
                        </option>
                      ))}
                    </select>

                    <select
                      name="unit"
                      value={form.unit}
                      onChange={handleChange}
                      className="w-full rounded-2xl border border-gray-200 bg-
white px-4 py-3 text-gray-800 focus:outline-none focus:ring-2 focus:ring-
green-500 focus:border-green-500 transition-all duration-300"
                      disabled={!prerequisitesReady}
                    >
                      <option value="шт">шт</option>
                      <option value="кг">кг</option>
                      <option value="л">л</option>
                    </select>

                    <input
                      type="number"
                      min="0"
                      step="0.01"
                      name="purchasePrice"
                      value={form.purchasePrice}
                      onChange={handleChange}
                      placeholder="Закупівельна ціна*"
                      className="w-full rounded-2xl border border-gray-200 bg-
white px-4 py-3 text-gray-800 placeholder:text-gray-400 focus:outline-none
focus:ring-2 focus:ring-green-500 focus:border-green-500 transition-all
duration-300"
                      disabled={!prerequisitesReady}
                    />

                    <input
                      type="number"
                      min="0"
                      step="0.01"
                      name="salePrice"
                      value={form.salePrice}
                      onChange={handleChange}
                      placeholder="Відпускна ціна*"
                      className="w-full rounded-2xl border border-gray-200 bg-
white px-4 py-3 text-gray-800 placeholder:text-gray-400 focus:outline-none
focus:ring-2 focus:ring-green-500 focus:border-green-500 transition-all
duration-300"
                      disabled={!prerequisitesReady}
                    />

                    <input
                      type="number"
```

```
                    min="0"
                    step="1"
                    name="quantity"
                    value={form.quantity}
                    onChange={handleChange}
                    placeholder="Поточна кількість*"
                    className="w-full rounded-2xl border border-gray-200 bg-
white px-4 py-3 text-gray-800 placeholder:text-gray-400 focus:outline-none
focus:ring-2 focus:ring-green-500 focus:border-green-500 transition-all
duration-300"
                    disabled={!prerequisitesReady}
                  />

                  <input
                    type="number"
                    min="1"
                    step="1"
                    name="minStock"
                    value={form.minStock}
                    onChange={handleChange}
                    placeholder="Мінімальний залишок*"
                    className="w-full rounded-2xl border border-gray-200 bg-
white px-4 py-3 text-gray-800 placeholder:text-gray-400 focus:outline-none
focus:ring-2 focus:ring-green-500 focus:border-green-500 transition-all
duration-300"
                    disabled={!prerequisitesReady}
                  />

                  <input
                    type="number"
                    min="1"
                    step="1"
                    name="maxStock"
                    value={form.maxStock}
                    onChange={handleChange}
                    placeholder="Бажаний / максимальний запас*"
                    className="w-full rounded-2xl border border-gray-200 bg-
white px-4 py-3 text-gray-800 placeholder:text-gray-400 focus:outline-none
focus:ring-2 focus:ring-green-500 focus:border-green-500 transition-all
duration-300"
                    disabled={!prerequisitesReady}
                  />

                  <input
                    type="text"
                    name="barcode"
                    value={form.barcode}
                    onChange={handleChange}
                    placeholder="Штрихкод"
                    className="w-full rounded-2xl border border-gray-200 bg-
white px-4 py-3 text-gray-800 placeholder:text-gray-400 focus:outline-none
focus:ring-2 focus:ring-green-500 focus:border-green-500 transition-all
duration-300"
                    disabled={!prerequisitesReady}
                  />

                  <select
                    name="rackId"
                    value={form.rackId}
                    onChange={handleChange}
                    className="w-full rounded-2xl border border-gray-200 bg-
white px-4 py-3 text-gray-800 focus:outline-none focus:ring-2 focus:ring-
green-500 focus:border-green-500 transition-all duration-300"
```

```
                    disabled={!prerequisitesReady}
                  >
                    <option value="">Оберіть стелаж*</option>
                    {racks.map((item) => (
                      <option key={item.id} value={item.id}>
                        {item.name}
                      </option>
                    ))}
                  </select>
                </div>

                {form.rackId && (
                  <div className="rounded-2xl border border-green-100 bg-
white p-4 mb-4">
                    <div className="text-sm font-semibold text-gray-800 mb-
3">
                      Оберіть одну або кілька комірок
                    </div>

                    <div className="flex flex-wrap gap-2">
                      {availableCells.map((cell, index) => {
                        const checked = form.cellCodes.includes(cell.code);
                        const occupied = cell.status === "occupied";

                        return (
                          <label
                            key={index}
                            className={`inline-flex items-center gap-2 px-3
py-2 rounded-xl border text-sm font-medium transition-all duration-300 ${
                              occupied
                                ? "border-red-200 bg-red-50 text-red-500
cursor-not-allowed"
                                : checked
                                  ? "border-green-500 bg-green-500 text-
white cursor-pointer"
                                  : "border-green-200 bg-green-50 text-
green-700 cursor-pointer hover:bg-green-100"
                            }`}
                          >
                            <input
                              type="checkbox"
                              checked={checked}
                              disabled={occupied}
                              onChange={() =>
                                handleToggleCell(cell.code, cell.status)
                              }
                              className="hidden"
                            />
                            {cell.code}
                            <span className="text-xs opacity-90">
                              {occupied ? "зайнята" : checked ? "обрана" :
"вільна"}
                            </span>
                          </label>
                        );
                      })}
                    </div>
                  </div>
                )}

                <div className="flex justify-end">
                  <button
                    type="button"
```

```
                    onClick={handleAddProduct}
                    disabled={!prerequisitesReady}
                    className="inline-flex items-center justify-center gap-2
rounded-xl border border-green-500 px-4 py-3 text-sm font-semibold text-
green-600 transition-all duration-300 hover:bg-green-500 hover:text-white
disabled:opacity-50 disabled:cursor-not-allowed disabled:hover:bg-transparent
disabled:hover:text-green-600"
                  >
                    <Plus size={16} />
                    Додати товар
                  </button>
                </div>
              </div>

              <div className="rounded-[1.5rem] border border-green-100 bg-
[#fcfffd] p-5 sm:p-6 shadow-sm mb-6">
                <div className="grid grid-cols-1 md:grid-cols-
[1fr_240px_240px] gap-4">
                  <div className="relative">
                    <Search
                      className="absolute left-4 top-1/2 -translate-y-1/2
text-gray-400"
                      size={18}
                    />
                    <input
                      type="text"
                      value={searchTerm}
                      onChange={(e) => setSearchTerm(e.target.value)}
                      placeholder="Пошук по товарах"
                      className="w-full rounded-2xl border border-gray-200
bg-white pl-11 pr-4 py-3 text-gray-800 placeholder:text-gray-400
focus:outline-none focus:ring-2 focus:ring-green-500 focus:border-green-500
transition-all duration-300"
                    />
                  </div>

                  <select
                    value={categoryFilter}
                    onChange={(e) => setCategoryFilter(e.target.value)}
                    className="w-full rounded-2xl border border-gray-200 bg-
white px-4 py-3 text-gray-800 focus:outline-none focus:ring-2 focus:ring-
green-500 focus:border-green-500 transition-all duration-300"
                  >
                    <option value="">Усі категорії</option>
                    {categories.map((item) => (
                      <option key={item.id} value={item.id}>
                        {item.name}
                      </option>
                    ))}
                  </select>

                  <select
                    value={supplierFilter}
                    onChange={(e) => setSupplierFilter(e.target.value)}
                    className="w-full rounded-2xl border border-gray-200 bg-
white px-4 py-3 text-gray-800 focus:outline-none focus:ring-2 focus:ring-
green-500 focus:border-green-500 transition-all duration-300"
                  >
                    <option value="">Усі постачальники</option>
                    {suppliers.map((item) => (
                      <option key={item.id} value={item.id}>
                        {item.name}
                      </option>
```

```
                    ))}
                  </select>
                </div>
              </div>

              <div className="rounded-[1.5rem] border border-green-100 bg-
white shadow-sm overflow-hidden">
                {loading ? (
                  <div className="px-6 py-12 text-center text-gray-500">
                    Завантаження товарів...
                  </div>
                ) : filteredProducts.length === 0 ? (
                  <div className="px-6 py-14 text-center">
                    <h3 className="text-2xl font-bold text-gray-900 mb-3">
                      Товарів поки немає
                    </h3>
                    <p className="text-gray-600 max-w-2xl mx-auto leading-
relaxed">
                      Додайте перший товар через форму вище. Після створення
він
                      з’явиться у списку нижче.
                    </p>
                  </div>
                ) : (
                  <>
                    <div className="hidden xl:grid grid-cols-
[70px_1.2fr_1.1fr_1fr_1fr_120px] gap-4 bg-green-50/70 border-b border-green-
100 px-6 py-4 text-sm font-bold text-gray-800">
                      <div>№</div>
                      <div>Товар</div>
                      <div>Категорія / постачальник</div>
                      <div>Ціни / кількість</div>
                      <div>Зберігання</div>
                      <div className="text-center">Дії</div>
                    </div>

                    <div className="divide-y divide-green-100">
                      {filteredProducts.map((product, index) => (
                        <div
                          key={product.id}
                          className="px-4 sm:px-6 py-4 hover:bg-green-50/40
transition-all duration-300"
                        >
                          <div className="hidden xl:grid grid-cols-
[70px_1.2fr_1.1fr_1fr_1fr_120px] gap-4 items-start">
                            <div className="text-gray-800 font-semibold pt-
1">
                              {index + 1}
                            </div>

                            <div>
                              <div className="text-gray-900 font-bold text-
base">
                                {product.name}
                              </div>
                              <div className="text-sm text-gray-600 mt-1">
                                Артикул: {product.sku}
                              </div>
                              <div className="text-sm text-gray-500 mt-1">
                                Статус: {product.status}
                              </div>
                            </div>
```

```
                            <div className="space-y-1 text-sm">
                              <div className="text-gray-800 font-medium">
                                {product.categoryName}
                              </div>
                              <div className="text-gray-600">
                                {product.supplierName}
                              </div>
                              <div className="text-gray-500">
                                {formatDate(product.createdAt)}
                              </div>
                            </div>

                            <div className="space-y-1 text-sm text-gray-
700">
                              <div>Закупівля: {product.purchasePrice}</div>
                              <div>Відпуск: {product.salePrice}</div>
                              <div>
                                Кількість: {product.quantity} {product.unit}
                              </div>
                              <div>Мін: {product.minStock}</div>
                              <div>Макс: {product.maxStock}</div>
                            </div>

                            <div className="space-y-1 text-sm text-gray-
700">
                              <div>{product.rackName || "—"}</div>
                              <div>{product.cellCodes?.join(", ") || "—
"}</div>
                              {product.barcode && (
                                <div className="text-gray-500">
                                  ШК: {product.barcode}
                                </div>
                              )}
                            </div>

                             <div className="flex items-center justify-
center gap-3">
                              <button
                                onClick={() => handleOpenEdit(product)}
                                className="inline-flex items-center justify-
center w-10 h-10 rounded-xl border border-blue-200 text-blue-600 hover:bg-
blue-500 hover:text-white transition-all duration-300"
                                title="Редагувати"
                              >
                                <Pencil size={17} />
                              </button>
                              <button
                                onClick={() => handleDeleteProduct(product)}
                                className="inline-flex items-center justify-
center w-10 h-10 rounded-xl border border-red-200 text-red-600 hover:bg-red-
500 hover:text-white transition-all duration-300"
                                title="Видалити"
                              >
                                <Trash2 size={17} />
                              </button>
                            </div>
                          </div>

                          <div className="xl:hidden rounded-2xl border
border-green-100 bg-[#fcfffd] p-4">
                            <div className="flex items-start justify-between
gap-3 mb-3">
                              <div>
```

```
                                <div className="text-xs text-gray-500 mb-1">
                                  Товар №{index + 1}
                                </div>
                                <div className="text-gray-900 font-bold
text-lg">
                                  {product.name}
                                </div>
                                <div className="text-sm text-gray-600 mt-1">
                                  Артикул: {product.sku}
                                </div>
                              </div>

                              <button
                                onClick={() => handleDeleteProduct(product)}
                                className="inline-flex items-center justify-
center w-10 h-10 rounded-xl border border-red-200 text-red-600 hover:bg-red-
500 hover:text-white transition-all duration-300"
                                title="Видалити"
                              >
                                <Trash2 size={17} />
                              </button>
                            </div>

                            <div className="grid grid-cols-1 sm:grid-cols-2
gap-2 text-sm text-gray-600">
                              <div>
                                <span className="font-semibold text-gray-
800">
                                  Категорія:
                                </span>{" "}
                                {product.categoryName}
                              </div>
                              <div>
                                <span className="font-semibold text-gray-
800">
                                  Постачальник:
                                </span>{" "}
                                {product.supplierName}
                              </div>
                              <div>
                                <span className="font-semibold text-gray-
800">
                                  Закупівля:
                                </span>{" "}
                                {product.purchasePrice}
                              </div>
                              <div>
                                <span className="font-semibold text-gray-
800">
                                  Відпуск:
                                </span>{" "}
                                {product.salePrice}
                              </div>
                              <div>
                                <span className="font-semibold text-gray-
800">
                                  Кількість:
                                </span>{" "}
                                {product.quantity} {product.unit}
                              </div>
                              <div>
                                <span className="font-semibold text-gray-
800">
```

```
                                    Мін / Макс:
                                  </span>{" "}
                                  {product.minStock} / {product.maxStock}
                                </div>
                                <div className="sm:col-span-2">
                                  <span className="font-semibold text-gray-
800">
                                    Зберігання:
                                  </span>{" "}
                                  {product.rackName}{" "}
                                  {product.cellCodes?.length
                                    ? `(${product.cellCodes.join(", ")})`
                                    : ""}
                                </div>
                                {product.barcode && (
                                  <div className="sm:col-span-2">
                                    <span className="font-semibold text-gray-
800">
                                      Штрихкод:
                                    </span>{" "}
                                    {product.barcode}
                                  </div>
                                )}
                                <div className="sm:col-span-2">
                                  <span className="font-semibold text-gray-
800">
                                    Статус:
                                  </span>{" "}
                                  {product.status}
                                </div>
                                <div className="sm:col-span-2">
                                  <span className="font-semibold text-gray-
800">
                                    Дата створення:
                                  </span>{" "}
                                  {formatDate(product.createdAt)}
                                </div>
                              </div>
                            </div>
                          </div>
                        ))}
                      </div>
                    </>
                  )}
                </div>
              </div>
            </div>
            {isEditOpen && editingProduct && (
          <EditProductModal
            product={editingProduct}
            products={products}
            categories={categories}
            suppliers={suppliers}
            racks={racks}
            user={user}
            onClose={handleCloseEdit}
            onSaved={() => {
              handleCloseEdit();
              fetchData();
            }}
          />
        )}
            </main>
```

```
  );
}
```

Сторінка категорій, компонент Categories.jsx:

```
import { useCallback, useEffect, useMemo, useState } from "react";
import { Tags, Pencil, Trash2, Search, Plus } from "lucide-react";
import { useAuth } from "../context/AuthContext";
import { db } from "../firebase";
import {
  addDoc,
  collection,
  doc,
  getDocs,
  query,
  serverTimestamp,
  updateDoc,
  where,
  writeBatch,
} from "firebase/firestore";
import Swal from "sweetalert2";

export default function Categories() {
  const { user } = useAuth();

  const [categoryName, setCategoryName] = useState("");
  const [searchTerm, setSearchTerm] = useState("");
  const [categories, setCategories] = useState([]);
  const [products, setProducts] = useState([]);
  const [loading, setLoading] = useState(true);

  const toast = (icon, title) => {
    Swal.fire({
      toast: true,
      position: "top-end",
      icon,
      title,
      showConfirmButton: false,
      timer: 3500,
      timerProgressBar: true,
    });
  };

  const formatDate = (timestamp) => {
    if (!timestamp?.toDate) return "—";
    return timestamp
      .toDate()
      .toLocaleDateString("uk-UA", {
        day: "2-digit",
        month: "2-digit",
        year: "numeric",
      });
  };

  const fetchData = useCallback(async () => {
  if (!user) return;

  setLoading(true);

  try {
    const categoriesQuery = query(
      collection(db, "categories"),
```

```
      where("userId", "==", user.uid)
    );

    const productsQuery = query(
      collection(db, "products"),
      where("userId", "==", user.uid)
    );

    const [categoriesSnapshot, productsSnapshot] = await Promise.all([
      getDocs(categoriesQuery),
      getDocs(productsQuery),
    ]);

    const categoriesData = categoriesSnapshot.docs.map((item) => ({
      id: item.id,
      ...item.data(),
    }));

    const productsData = productsSnapshot.docs.map((item) => ({
      id: item.id,
      ...item.data(),
    }));

    setCategories(categoriesData);
    setProducts(productsData);
  } catch (error) {
    toast("error", "Не вдалося завантажити категорії");
  } finally {
    setLoading(false);
  }
}, [user]);

useEffect(() => {
  fetchData();
}, [fetchData]);

  const handleAddCategory = async () => {
  if (!user) {
    toast("error", "Користувача не знайдено");
    return;
  }

  const trimmedName = categoryName.trim();

  if (!trimmedName) {
    toast("error", "Введіть назву категорії");
    return;
  }

  const exists = categories.some(
    (item) => item.name?.toLowerCase() === trimmedName.toLowerCase()
  );

  if (exists) {
    toast("error", "Така категорія вже існує");
    return;
  }

  try {
    const docRef = await addDoc(collection(db, "categories"), {
      name: trimmedName,
      userId: user.uid,
      createdAt: serverTimestamp(),
```

```
    });

    setCategories((prev) => [
      {
        id: docRef.id,
        name: trimmedName,
        userId: user.uid,
        createdAt: null,
      },
      ...prev,
    ]);

    setCategoryName("");
    toast("success", "Категорію успішно додано");
    fetchData();
  } catch (error) {
    toast("error", "Не вдалося додати категорію");
  }
};

  const handleEditCategory = async (category) => {
    const result = await Swal.fire({
      title: "Редагування категорії",
      input: "text",
      inputLabel: "Нова назва категорії",
      inputValue: category.name || "",
      inputPlaceholder: "Введіть нову назву",
      showCancelButton: true,
      confirmButtonText: "Зберегти",
      cancelButtonText: "Скасувати",
      confirmButtonColor: "#22c55e",
      cancelButtonColor:"#d1d5db",
      inputValidator: (value) => {
        if (!value.trim()) {
          return "Назва категорії не може бути порожньою";
        }
      },
    });

    if (!result.isConfirmed) return;

    const newName = result.value.trim();

    const exists = categories.some(
      (item) =>
        item.id !== category.id &&
        item.name?.toLowerCase() === newName.toLowerCase()
    );

    if (exists) {
      toast("error", "Категорія з такою назвою вже існує");
      return;
    }

    try {
      await updateDoc(doc(db, "categories", category.id), {
        name: newName,
      });

      const relatedProducts = products.filter(
        (product) => product.categoryId === category.id
      );
```

```
            if (relatedProducts.length > 0) {
              const batch = writeBatch(db);

              relatedProducts.forEach((product) => {
                batch.update(doc(db, "products", product.id), {
                  categoryName: newName,
                });
              });

              await batch.commit();
            }

            toast("success", "Категорію оновлено");
            fetchData();
          } catch (error) {
            toast("error", "Не вдалося оновити категорію");
          }
        };

        const handleDeleteCategory = async (category) => {
          const result = await Swal.fire({
            title: "Ви впевнені?",
            text: "Категорію та всі товари в ній буде видалено без можливості
відновлення.",
            icon: "warning",
            showCancelButton: true,
            confirmButtonText: "Так, видалити",
            cancelButtonText: "Скасувати",
            confirmButtonColor: "#ef4444",
            cancelButtonColor: "#9ca3af",
            reverseButtons: true,
          });

          if (!result.isConfirmed) return;

          try {
            const batch = writeBatch(db);

            const relatedProducts = products.filter(
              (product) => product.categoryId === category.id
            );

            relatedProducts.forEach((product) => {
              batch.delete(doc(db, "products", product.id));
            });

            batch.delete(doc(db, "categories", category.id));

            await batch.commit();

            toast("success", "Категорію та пов’язані товари видалено");
            fetchData();
          } catch (error) {
            toast("error", "Не вдалося видалити категорію");
          }
        };

        const filteredCategories = useMemo(() => {
          return categories.filter((item) =>
            item.name?.toLowerCase().includes(searchTerm.toLowerCase().trim())
          );
        }, [categories, searchTerm]);
```

```
        const getProductsCount = (categoryId) => {
          return products.filter((product) => product.categoryId ===
categoryId).length;
        };

        return (
          <main className="min-h-screen bg-[#f6faf7] px-4 py-10">
            <div className="max-w-6xl mx-auto">
              <div className="rounded-[2rem] border border-green-100 bg-white
shadow-xl shadow-green-100/40 p-6 sm:p-8 md:p-10">
                <div className="flex items-center gap-4 mb-8">
                  <div className="w-16 h-16 rounded-full bg-green-100 text-
green-600 flex items-center justify-center shrink-0">
                    <Tags size={30} />
                  </div>

                  <div>
                    <h1 className="text-3xl sm:text-4xl font-black text-gray-
900 mb-2">
                      Категорії
                    </h1>
                    <p className="text-gray-600 text-base sm:text-lg leading-
relaxed">
                      Створюйте категорії товарів, редагуйте їх, виконуйте
пошук та
                      керуйте структурою власного каталогу.
                    </p>
                  </div>
                </div>

                <div className="rounded-[1.5rem] border border-green-100 bg-
[#fcfffd] p-5 sm:p-6 shadow-sm mb-6">
                  <div className="flex flex-col md:flex-row gap-3">
                    <input
                      type="text"
                      value={categoryName}
                      onChange={(e) => setCategoryName(e.target.value)}
                      placeholder="Введіть назву категорії"
                      className="flex-1 rounded-2xl border border-gray-200 bg-
white px-4 py-3 text-gray-800 placeholder:text-gray-400 focus:outline-none
focus:ring-2 focus:ring-green-500 focus:border-green-500 transition-all
duration-300"
                    />

                    <button
                      type="button"
                      onClick={handleAddCategory}
                      className="inline-flex items-center justify-center gap-2
rounded-xl border border-green-500 px-4 py-3 text-sm font-semibold text-
green-600 transition-all duration-300 hover:bg-green-500 hover:text-white"
                    >
                      <Plus size={16} />
                      Додати категорію
                    </button>
                  </div>
                </div>

                <div className="rounded-[1.5rem] border border-green-100 bg-
[#fcfffd] p-5 sm:p-6 shadow-sm mb-6">
                  <div className="relative">
                    <Search
                      className="absolute left-4 top-1/2 -translate-y-1/2
text-gray-400"
```

```
                    size={18}
                  />
                  <input
                    type="text"
                    value={searchTerm}
                    onChange={(e) => setSearchTerm(e.target.value)}
                    placeholder="Пошук по категоріях"
                    className="w-full rounded-2xl border border-gray-200 bg-
white pl-11 pr-4 py-3 text-gray-800 placeholder:text-gray-400 focus:outline-
none focus:ring-2 focus:ring-green-500 focus:border-green-500 transition-all
duration-300"
                  />
                </div>
              </div>

              <div className="rounded-[1.5rem] border border-green-100 bg-
white shadow-sm overflow-hidden">
                {loading ? (
                  <div className="px-6 py-12 text-center text-gray-500">
                    Завантаження категорій...
                  </div>
                ) : filteredCategories.length === 0 ? (
                  <div className="px-6 py-14 text-center">
                    <h3 className="text-2xl font-bold text-gray-900 mb-3">
                      Категорій поки немає
                    </h3>
                    <p className="text-gray-600 max-w-2xl mx-auto leading-
relaxed">
                      Щоб почати роботу, введіть назву нової категорії у
поле вище та
                      натисніть кнопку додавання. Після створення категорії
вона
                      з’явиться в таблиці нижче.
                    </p>
                  </div>
                ) : (
                  <>
                    <div className="hidden md:grid grid-cols-
[80px_1fr_180px_160px] gap-4 bg-green-50/70 border-b border-green-100 px-6
py-4 text-sm font-bold text-gray-800">
                      <div>№</div>
                      <div>Назва / к-сть товарів</div>
                      <div>Дата створення</div>
                      <div className="text-center">Дії</div>
                    </div>

                    <div className="divide-y divide-green-100">
                      {filteredCategories.map((category, index) => (
                        <div
                          key={category.id}
                          className="px-4 sm:px-6 py-4 hover:bg-green-50/40
transition-all duration-300"
                        >
                          <div className="hidden md:grid grid-cols-
[80px_1fr_180px_160px] gap-4 items-center">
                            <div className="text-gray-800 font-semibold">
                              {index + 1}
                            </div>

                            <div>
                              <div className="text-gray-900 font-bold">
                                {category.name}
                              </div>
```

```
                                <div className="text-sm text-gray-500 mt-1">
                                  Товарів у категорії:
{getProductsCount(category.id)}
                                </div>
                              </div>

                              <div className="text-gray-700 font-medium">
                                {formatDate(category.createdAt)}
                              </div>

                              <div className="flex items-center justify-center
gap-3">
                                <button
                                  onClick={() => handleEditCategory(category)}
                                  className="inline-flex items-center justify-
center w-10 h-10 rounded-xl border border-blue-200 text-blue-600 hover:bg-
blue-500 hover:text-white transition-all duration-300"
                                  title="Редагувати"
                                >
                                  <Pencil size={17} />
                                </button>

                                <button
                                  onClick={() =>
handleDeleteCategory(category)}
                                  className="inline-flex items-center justify-
center w-10 h-10 rounded-xl border border-red-200 text-red-600 hover:bg-red-
500 hover:text-white transition-all duration-300"
                                  title="Видалити"
                                >
                                  <Trash2 size={17} />
                                </button>
                              </div>
                            </div>

                            <div className="md:hidden rounded-2xl border
border-green-100 bg-[#fcfffd] p-4">
                              <div className="flex items-start justify-between
gap-3 mb-3">
                                <div>
                                  <div className="text-xs text-gray-500 mb-1">
                                    Категорія №{index + 1}
                                  </div>
                                  <div className="text-gray-900 font-bold
text-lg">
                                    {category.name}
                                  </div>
                                </div>

                                <div className="flex items-center gap-2">
                                  <button
                                    onClick={() =>
handleEditCategory(category)}
                                    className="inline-flex items-center
justify-center w-10 h-10 rounded-xl border border-blue-200 text-blue-600
hover:bg-blue-500 hover:text-white transition-all duration-300"
                                    title="Редагувати"
                                  >
                                    <Pencil size={17} />
                                  </button>

                                  <button
```

```
                                    onClick={() =>
handleDeleteCategory(category)}
                                    className="inline-flex items-center
justify-center w-10 h-10 rounded-xl border border-red-200 text-red-600
hover:bg-red-500 hover:text-white transition-all duration-300"
                                    title="Видалити"
                                  >
                                    <Trash2 size={17} />
                                  </button>
                                </div>
                              </div>

                              <div className="text-sm text-gray-600 mb-2">
                                Товарів у категорії:
{getProductsCount(category.id)}
                              </div>

                              <div className="text-sm text-gray-600">
                                Дата створення:
{formatDate(category.createdAt)}
                              </div>
                            </div>
                          </div>
                        ))}
                      </div>
                    </>
                  )}
                </div>
              </div>
            </div>
          </main>
        );
      }
```

Сторінка постачальників, компонент Suppliers.jsx:

```
import { useCallback, useEffect, useMemo, useState } from "react";
import { Truck, Pencil, Trash2, Search, Plus } from "lucide-react";
import { useAuth } from "../context/AuthContext";
import { db } from "../firebase";
import {
  addDoc,
  collection,
  deleteDoc,
  doc,
  getDocs,
  query,
  serverTimestamp,
  updateDoc,
  where,
} from "firebase/firestore";
import Swal from "sweetalert2";

export default function Suppliers() {
  const { user } = useAuth();

  const [form, setForm] = useState({
    name: "",
    contactPerson: "",
    phone: "",
    email: "",
    address: "",
```

```
  companyType: "",
  notes: "",
});

const [searchTerm, setSearchTerm] = useState("");
const [suppliers, setSuppliers] = useState([]);
const [loading, setLoading] = useState(true);

const toast = (icon, title) => {
  Swal.fire({
    toast: true,
    position: "top-end",
    icon,
    title,
    showConfirmButton: false,
    timer: 3500,
    timerProgressBar: true,
  });
};

const formatDate = (timestamp) => {
  if (!timestamp?.toDate) return "—";
  return timestamp.toDate().toLocaleDateString("uk-UA", {
    day: "2-digit",
    month: "2-digit",
    year: "numeric",
  });
};

const fetchSuppliers = useCallback(async () => {
  if (!user) return;

  setLoading(true);

  try {
    const suppliersQuery = query(
      collection(db, "suppliers"),
      where("userId", "==", user.uid)
    );

    const snapshot = await getDocs(suppliersQuery);

    const suppliersData = snapshot.docs.map((item) => ({
      id: item.id,
      ...item.data(),
    }));

    setSuppliers(suppliersData);
  } catch (error) {
    toast("error", "Не вдалося завантажити постачальників");
  } finally {
    setLoading(false);
  }
}, [user]);

useEffect(() => {
  fetchSuppliers();
}, [fetchSuppliers]);

const handleChange = (e) => {
  setForm({ ...form, [e.target.name]: e.target.value });
};
```

```
const resetForm = () => {
  setForm({
    name: "",
    contactPerson: "",
    phone: "",
    email: "",
    address: "",
    companyType: "",
    notes: "",
  });
};

const handleAddSupplier = async () => {
  if (!user) {
    toast("error", "Користувача не знайдено");
    return;
  }

  const payload = {
    name: form.name.trim(),
    contactPerson: form.contactPerson.trim(),
    phone: form.phone.trim(),
    email: form.email.trim(),
    address: form.address.trim(),
    companyType: form.companyType.trim(),
    notes: form.notes.trim(),
  };

  if (
    !payload.name ||
    !payload.contactPerson ||
    !payload.phone ||
    !payload.address
  ) {
    toast("error", "Заповніть усі обов’язкові поля");
    return;
  }

  const exists = suppliers.some(
    (item) => item.name?.toLowerCase() === payload.name.toLowerCase()
  );

  if (exists) {
    toast("error", "Постачальник з такою назвою вже існує");
    return;
  }

  try {
    const docRef = await addDoc(collection(db, "suppliers"), {
      ...payload,
      userId: user.uid,
      createdAt: serverTimestamp(),
    });

    setSuppliers((prev) => [
      {
        id: docRef.id,
        ...payload,
        userId: user.uid,
        createdAt: null,
      },
      ...prev,
    ]);
```

```
            resetForm();
            toast("success", "Постачальника успішно додано");
            fetchSuppliers();
          } catch (error) {
            toast("error", "Не вдалося додати постачальника");
          }
        };

        const handleEditSupplier = async (supplier) => {
          const { value: values } = await Swal.fire({
            title: "Редагування постачальника",
            html: `
              <input id="swal-name" class="swal2-input" placeholder="Назва
постачальника" value="${supplier.name || ""}">
              <input id="swal-contactPerson" class="swal2-input"
placeholder="Контактна особа" value="${supplier.contactPerson || ""}">
              <input id="swal-phone" class="swal2-input" placeholder="Телефон"
value="${supplier.phone || ""}">
              <input id="swal-email" class="swal2-input" placeholder="Email"
value="${supplier.email || ""}">
              <input id="swal-address" class="swal2-input"
placeholder="Адреса" value="${supplier.address || ""}">
              <input id="swal-companyType" class="swal2-input"
placeholder="Тип компанії" value="${supplier.companyType || ""}">
              <input id="swal-notes" class="swal2-input"
placeholder="Коментар" value="${supplier.notes || ""}">
            `,
            focusConfirm: false,
            showCancelButton: true,
            confirmButtonText: "Зберегти",
            cancelButtonText: "Скасувати",
            confirmButtonColor: "#22c55e",
            cancelButtonColor:"#d1d5db",
            preConfirm: () => {
              const updated = {
                name: document.getElementById("swal-name").value.trim(),
                contactPerson: document.getElementById("swal-
contactPerson").value.trim(),
                phone: document.getElementById("swal-phone").value.trim(),
                email: document.getElementById("swal-email").value.trim(),
                address: document.getElementById("swal-address").value.trim(),
                companyType: document.getElementById("swal-
companyType").value.trim(),
                notes: document.getElementById("swal-notes").value.trim(),
              };

              if (
                !updated.name ||
                !updated.contactPerson ||
                !updated.phone ||
                !updated.address

              ) {
                Swal.showValidationMessage("Заповніть усі обов’язкові поля");
                return false;
              }

              return updated;
            },
          });

          if (!values) return;
```

```
  const exists = suppliers.some(
    (item) =>
      item.id !== supplier.id &&
      item.name?.toLowerCase() === values.name.toLowerCase()
  );

  if (exists) {
    toast("error", "Постачальник з такою назвою вже існує");
    return;
  }

  try {
    await updateDoc(doc(db, "suppliers", supplier.id), values);
    toast("success", "Постачальника оновлено");
    fetchSuppliers();
  } catch (error) {
    toast("error", "Не вдалося оновити постачальника");
  }
};

const handleDeleteSupplier = async (supplier) => {
  const result = await Swal.fire({
    title: "Ви впевнені?",
    text: "Постачальника буде видалено без можливості відновлення.",
    icon: "warning",
    showCancelButton: true,
    confirmButtonText: "Так, видалити",
    cancelButtonText: "Скасувати",
    confirmButtonColor: "#ef4444",
    cancelButtonColor: "#9ca3af",
    reverseButtons: true,
  });

  if (!result.isConfirmed) return;

  try {
    await deleteDoc(doc(db, "suppliers", supplier.id));
    toast("success", "Постачальника видалено");
    fetchSuppliers();
  } catch (error) {
    toast("error", "Не вдалося видалити постачальника");
  }
};

const filteredSuppliers = useMemo(() => {
  const value = searchTerm.toLowerCase().trim();

  return suppliers.filter(
    (item) =>
      item.name?.toLowerCase().includes(value) ||
      item.contactPerson?.toLowerCase().includes(value) ||
      item.phone?.toLowerCase().includes(value) ||
      item.email?.toLowerCase().includes(value) ||
      item.address?.toLowerCase().includes(value) ||
      item.companyType?.toLowerCase().includes(value)
  );
}, [suppliers, searchTerm]);

return (
  <main className="min-h-screen bg-[#f6faf7] px-4 py-10">
    <div className="max-w-7xl mx-auto">
```

```
            <div className="rounded-[2rem] border border-green-100 bg-white
shadow-xl shadow-green-100/40 p-6 sm:p-8 md:p-10">
              <div className="flex items-center gap-4 mb-8">
                <div className="w-16 h-16 rounded-full bg-green-100 text-
green-600 flex items-center justify-center shrink-0">
                  <Truck size={30} />
                </div>

                <div>
                  <h1 className="text-3xl sm:text-4xl font-black text-gray-
900 mb-2">
                    Постачальники
                  </h1>
                  <p className="text-gray-600 text-base sm:text-lg leading-
relaxed">
                    Додавайте постачальників, зберігайте контактні дані та
керуйте
                    інформацією про співпрацю в межах вашої системи.
                  </p>
                </div>
              </div>

              <div className="rounded-[1.5rem] border border-green-100 bg-
[#fcfffd] p-5 sm:p-6 shadow-sm mb-6">
                <div className="grid grid-cols-1 md:grid-cols-2 gap-4 mb-4">
                  <input
                    type="text"
                    name="name"
                    value={form.name}
                    onChange={handleChange}
                    placeholder="Назва постачальника*"
                    className="w-full rounded-2xl border border-gray-200 bg-
white px-4 py-3 text-gray-800 placeholder:text-gray-400 focus:outline-none
focus:ring-2 focus:ring-green-500 focus:border-green-500 transition-all
duration-300"
                  />

                  <input
                    type="text"
                    name="contactPerson"
                    value={form.contactPerson}
                    onChange={handleChange}
                    placeholder="Контактна особа*"
                    className="w-full rounded-2xl border border-gray-200 bg-
white px-4 py-3 text-gray-800 placeholder:text-gray-400 focus:outline-none
focus:ring-2 focus:ring-green-500 focus:border-green-500 transition-all
duration-300"
                  />

                  <input
                    type="text"
                    name="phone"
                    value={form.phone}
                    onChange={handleChange}
                    placeholder="Телефон*"
                    className="w-full rounded-2xl border border-gray-200 bg-
white px-4 py-3 text-gray-800 placeholder:text-gray-400 focus:outline-none
focus:ring-2 focus:ring-green-500 focus:border-green-500 transition-all
duration-300"
                  />

                  <input
                    type="text"
```

```
                    name="email"
                    value={form.email}
                    onChange={handleChange}
                    placeholder="Електронна пошта"
                    className="w-full rounded-2xl border border-gray-200 bg-
white px-4 py-3 text-gray-800 placeholder:text-gray-400 focus:outline-none
focus:ring-2 focus:ring-green-500 focus:border-green-500 transition-all
duration-300"
                  />

                  <input
                    type="text"
                    name="address"
                    value={form.address}
                    onChange={handleChange}
                    placeholder="Адреса"
                    className="w-full rounded-2xl border border-gray-200 bg-
white px-4 py-3 text-gray-800 placeholder:text-gray-400 focus:outline-none
focus:ring-2 focus:ring-green-500 focus:border-green-500 transition-all
duration-300"
                  />

                  <input
                    type="text"
                    name="companyType"
                    value={form.companyType}
                    onChange={handleChange}
                    placeholder="Тип компанії: ФОП, ТОВ..."
                    className="w-full rounded-2xl border border-gray-200 bg-
white px-4 py-3 text-gray-800 placeholder:text-gray-400 focus:outline-none
focus:ring-2 focus:ring-green-500 focus:border-green-500 transition-all
duration-300"
                  />
                </div>

                <textarea
                  name="notes"
                  value={form.notes}
                  onChange={handleChange}
                  placeholder="Коментар"
                  rows={4}
                  className="w-full rounded-2xl border border-gray-200 bg-
white px-4 py-3 text-gray-800 placeholder:text-gray-400 focus:outline-none
focus:ring-2 focus:ring-green-500 focus:border-green-500 transition-all
duration-300 mb-4 resize-none"
                />

                <div className="flex justify-end">
                  <button
                    type="button"
                    onClick={handleAddSupplier}
                    className="inline-flex items-center justify-center gap-2
rounded-xl border border-green-500 px-4 py-3 text-sm font-semibold text-
green-600 transition-all duration-300 hover:bg-green-500 hover:text-white"
                  >
                    <Plus size={16} />
                    Додати постачальника
                  </button>
                </div>
              </div>

              <div className="rounded-[1.5rem] border border-green-100 bg-
[#fcfffd] p-5 sm:p-6 shadow-sm mb-6">
```

```
                <div className="relative">
                  <Search
                    className="absolute left-4 top-1/2 -translate-y-1/2
text-gray-400"
                    size={18}
                  />
                  <input
                    type="text"
                    value={searchTerm}
                    onChange={(e) => setSearchTerm(e.target.value)}
                    placeholder="Пошук по постачальниках"
                    className="w-full rounded-2xl border border-gray-200 bg-
white pl-11 pr-4 py-3 text-gray-800 placeholder:text-gray-400 focus:outline-
none focus:ring-2 focus:ring-green-500 focus:border-green-500 transition-all
duration-300"
                  />
                </div>
              </div>

              <div className="rounded-[1.5rem] border border-green-100 bg-
white shadow-sm overflow-hidden">
        {loading ? (
          <div className="px-6 py-12 text-center text-gray-500">
            Завантаження постачальників...
          </div>
        ) : filteredSuppliers.length === 0 ? (
          <div className="px-6 py-14 text-center">
            <h3 className="text-2xl font-bold text-gray-900 mb-3">
              Постачальників поки немає
            </h3>
            <p className="text-gray-600 max-w-2xl mx-auto leading-relaxed">
              Заповніть форму вище, щоб додати першого постачальника до
системи.
              Після створення він з’явиться в таблиці нижче.
            </p>
          </div>
        ) : (
          <>
            <div className="hidden lg:grid grid-cols-
[70px_1.2fr_1.1fr_1fr_120px] gap-4 bg-green-50/70 border-b border-green-100
px-6 py-4 text-sm font-bold text-gray-800">
              <div>№</div>
              <div>Постачальник</div>
              <div>Контакти</div>
              <div>Адреса / коментар</div>
              <div className="text-center">Дії</div>
            </div>

            <div className="divide-y divide-green-100">
              {filteredSuppliers.map((supplier, index) => (
                <div
                  key={supplier.id}
                  className="px-4 sm:px-6 py-4 hover:bg-green-50/40
transition-all duration-300"
                >
                  <div className="hidden lg:grid grid-cols-
[70px_1.2fr_1.1fr_1fr_120px] gap-4 items-start">
                    <div className="text-gray-800 font-semibold pt-1">
                      {index + 1}
                    </div>

                    <div>
                      <div className="text-gray-900 font-bold text-base">
```

```
                      {supplier.name}
                    </div>
                    <div className="text-sm text-gray-600 mt-1">
                      {supplier.companyType || "Тип не вказано"}
                    </div>
                    <div className="text-sm text-gray-500 mt-1">
                      Створено: {formatDate(supplier.createdAt)}
                    </div>
                  </div>

                  <div className="space-y-1">
                    <div className="text-gray-800 font-medium">
                      {supplier.contactPerson || "—"}
                    </div>
                    <div className="text-sm text-gray-600">
                      {supplier.phone || "—"}
                    </div>
                    <div className="text-sm text-gray-600 break-all">
                      {supplier.email || "—"}
                    </div>
                  </div>

                  <div className="space-y-2">
                    <div className="text-sm text-gray-700">
                      {supplier.address || "Адресу не вказано"}
                    </div>
                    {supplier.notes && (
                      <div className="text-sm text-gray-500 line-clamp-3">
                        {supplier.notes}
                      </div>
                    )}
                  </div>

                  <div className="flex items-center justify-center gap-3 pt-
1">
                    <button
                      onClick={() => handleEditSupplier(supplier)}
                      className="inline-flex items-center justify-center w-
10 h-10 rounded-xl border border-blue-200 text-blue-600 hover:bg-blue-500
hover:text-white transition-all duration-300"
                      title="Редагувати"
                    >
                      <Pencil size={17} />
                    </button>

                    <button
                      onClick={() => handleDeleteSupplier(supplier)}
                      className="inline-flex items-center justify-center w-
10 h-10 rounded-xl border border-red-200 text-red-600 hover:bg-red-500
hover:text-white transition-all duration-300"
                      title="Видалити"
                    >
                      <Trash2 size={17} />
                    </button>
                  </div>
                </div>

                <div className="lg:hidden rounded-2xl border border-green-
100 bg-[#fcfffd] p-4">
                  <div className="flex items-start justify-between gap-3 mb-
3">
                    <div>
                      <div className="text-xs text-gray-500 mb-1">
```

```
                        Постачальник №{index + 1}
                      </div>
                      <div className="text-gray-900 font-bold text-lg">
                        {supplier.name}
                      </div>
                      <div className="text-sm text-gray-600 mt-1">
                        {supplier.companyType || "Тип не вказано"}
                      </div>
                    </div>

                    <div className="flex items-center gap-2">
                      <button
                        onClick={() => handleEditSupplier(supplier)}
                        className="inline-flex items-center justify-center
w-10 h-10 rounded-xl border border-blue-200 text-blue-600 hover:bg-blue-500
hover:text-white transition-all duration-300"
                        title="Редагувати"
                      >
                        <Pencil size={17} />
                      </button>

                      <button
                        onClick={() => handleDeleteSupplier(supplier)}
                        className="inline-flex items-center justify-center
w-10 h-10 rounded-xl border border-red-200 text-red-600 hover:bg-red-500
hover:text-white transition-all duration-300"
                        title="Видалити"
                      >
                        <Trash2 size={17} />
                      </button>
                    </div>
                  </div>

                  <div className="grid grid-cols-1 sm:grid-cols-2 gap-2
text-sm text-gray-600">
                    <div>
                      <span className="font-semibold text-gray-800">
                        Контактна особа:
                      </span>{" "}
                      {supplier.contactPerson || "—"}
                    </div>
                    <div>
                      <span className="font-semibold text-gray-800">
                        Телефон:
                      </span>{" "}
                      {supplier.phone || "—"}
                    </div>
                    <div>
                      <span className="font-semibold text-gray-800">
                        Email:
                      </span>{" "}
                      {supplier.email || "—"}
                    </div>
                    <div>
                      <span className="font-semibold text-gray-800">
                        Тип:
                      </span>{" "}
                      {supplier.companyType || "—"}
                    </div>
                    <div className="sm:col-span-2">
                      <span className="font-semibold text-gray-800">
                        Адреса:
                      </span>{" "}
```

```
                  {supplier.address || "Не вказано"}
                </div>
                {supplier.notes && (
                  <div className="sm:col-span-2">
                    <span className="font-semibold text-gray-800">
                      Коментар:
                    </span>{" "}
                    {supplier.notes}
                  </div>
                )}
                <div className="sm:col-span-2">
                  <span className="font-semibold text-gray-800">
                    Дата створення:
                  </span>{" "}
                  {formatDate(supplier.createdAt)}
                </div>
              </div>
            </div>
          </div>
        ))}
      </div>
    </>
  )}
</div>
        </div>
      </div>
    </main>
  );
}
```

Сторінка місць зберігання, компонент StorageLocations.jsx:

```
import { useCallback, useEffect, useMemo, useState } from "react";
import { MapPinned, Plus, Trash2, Search } from "lucide-react";
import { useAuth } from "../context/AuthContext";
import { db } from "../firebase";
import {
  addDoc,
  collection,
  doc,
  getDocs,
  query,
  serverTimestamp,
  where,
  writeBatch,
} from "firebase/firestore";
import Swal from "sweetalert2";

export default function StorageLocations() {
  const { user } = useAuth();

  const [form, setForm] = useState({
    rackName: "",
    cellsCount: "",
  });
  const [searchTerm, setSearchTerm] = useState("");
  const [racks, setRacks] = useState([]);
  const [products, setProducts] = useState([]);
  const [loading, setLoading] = useState(true);

  const toast = (icon, title) => {
    Swal.fire({
```

```
    toast: true,
    position: "top-end",
    icon,
    title,
    showConfirmButton: false,
    timer: 3500,
    timerProgressBar: true,
  });
};

const formatDate = (timestamp) => {
  if (!timestamp?.toDate) return "—";
  return timestamp.toDate().toLocaleDateString("uk-UA", {
    day: "2-digit",
    month: "2-digit",
    year: "numeric",
  });
};

const getRackPrefix = (name) => {
  const firstLetter = name.trim().charAt(0).toUpperCase();
  return firstLetter || "A";
};

const buildCells = (rackName, count) => {
  const prefix = getRackPrefix(rackName);

  return Array.from({ length: count }, (_, index) => ({
    code: `${prefix}${index + 1}`,
    status: "free",
  }));
};

const applyOccupiedStatus = useCallback((rackList, productList) => {
  return rackList.map((rack) => {
    const updatedCells = (rack.cells || []).map((cell) => {
      const occupied = productList.some(
        (product) =>
          product.rackId === rack.id &&
          Array.isArray(product.cellCodes) &&
          product.cellCodes.includes(cell.code),
      );

      return {
        ...cell,
        status: occupied ? "occupied" : "free",
      };
    });

    return {
      ...rack,
      cells: updatedCells,
    };
  });
}, []);

const fetchData = useCallback(async () => {
  if (!user) return;

  setLoading(true);

  try {
    const racksQuery = query(
```

```
      collection(db, "storageLocations"),
      where("userId", "==", user.uid),
    );

    const productsQuery = query(
      collection(db, "products"),
      where("userId", "==", user.uid),
    );

    const [racksSnapshot, productsSnapshot] = await Promise.all([
      getDocs(racksQuery),
      getDocs(productsQuery),
    ]);

    const racksData = racksSnapshot.docs.map((item) => ({
      id: item.id,
      ...item.data(),
    }));

    const productsData = productsSnapshot.docs.map((item) => ({
      id: item.id,
      ...item.data(),
    }));

    setProducts(productsData);
    setRacks(applyOccupiedStatus(racksData, productsData));
  } catch (error) {
    toast("error", "Не вдалося завантажити місця зберігання");
  } finally {
    setLoading(false);
  }
}, [user, applyOccupiedStatus]);

useEffect(() => {
  fetchData();
}, [fetchData]);

const handleChange = (e) => {
  setForm((prev) => ({
    ...prev,
    [e.target.name]: e.target.value,
  }));
};

const resetForm = () => {
  setForm({
    rackName: "",
    cellsCount: "",
  });
};

const handleAddRack = async () => {
  if (!user) {
    toast("error", "Користувача не знайдено");
    return;
  }

  const rackName = form.rackName.trim();
  const cellsCount = Number(form.cellsCount);

  if (!rackName || !form.cellsCount) {
    toast("error", "Заповніть усі поля");
    return;
```

```
    }

    if (!Number.isInteger(cellsCount) || cellsCount <= 0) {
      toast("error", "Кількість комірок має бути додатнім числом");
      return;
    }

    if (cellsCount > 200) {
      toast("error", "Надто велика кількість комірок");
      return;
    }

    const exists = racks.some(
      (rack) => rack.name?.toLowerCase() === rackName.toLowerCase(),
    );

    if (exists) {
      toast("error", "Стелаж з такою назвою вже існує");
      return;
    }

    const cells = buildCells(rackName, cellsCount);

    try {
      const docRef = await addDoc(collection(db, "storageLocations"), {
        name: rackName,
        cellsCount,
        cells,
        userId: user.uid,
        createdAt: serverTimestamp(),
      });

      setRacks((prev) => [
        {
          id: docRef.id,
          name: rackName,
          cellsCount,
          cells,
          userId: user.uid,
          createdAt: null,
        },
        ...prev,
      ]);

      resetForm();
      toast("success", "Стелаж успішно додано");
      fetchData();
    } catch (error) {
      toast("error", "Не вдалося додати стелаж");
    }
  };

  const handleDeleteRack = async (rack) => {
    const result = await Swal.fire({
      title: "Ви впевнені?",
      text: "Стелаж буде видалено, а місце зберігання у всіх пов’язаних
товарах буде очищено.",
      icon: "warning",
      showCancelButton: true,
      confirmButtonText: "Так, видалити",
      cancelButtonText: "Скасувати",
      confirmButtonColor: "#ef4444",
      cancelButtonColor: "#9ca3af",
```

```
          reverseButtons: true,
        });

        if (!result.isConfirmed) return;

        try {
          const batch = writeBatch(db);

          const relatedProducts = products.filter(
            (product) => product.rackId === rack.id,
          );

          relatedProducts.forEach((product) => {
            batch.update(doc(db, "products", product.id), {
              rackId: "",
              rackName: "",
              cellCode: "",
              storageLocation: "",
            });
          });

          batch.delete(doc(db, "storageLocations", rack.id));

          await batch.commit();

          toast("success", "Стелаж видалено");
          fetchData();
        } catch (error) {
          toast("error", "Не вдалося видалити стелаж");
        }
      };

      const filteredRacks = useMemo(() => {
        return racks.filter((rack) =>

rack.name?.toLowerCase().includes(searchTerm.toLowerCase().trim()),
        );
      }, [racks, searchTerm]);

      return (
        <main className="min-h-screen bg-[#f6faf7] px-4 py-10">
          <div className="max-w-7xl mx-auto">
            <div className="rounded-[2rem] border border-green-100 bg-white
shadow-xl shadow-green-100/40 p-6 sm:p-8 md:p-10">
              <div className="flex items-center gap-4 mb-8">
                <div className="w-16 h-16 rounded-full bg-green-100 text-
green-600 flex items-center justify-center shrink-0">
                  <MapPinned size={30} />
                </div>

                <div>
                  <h1 className="text-3xl sm:text-4xl font-black text-gray-
900 mb-2">
                    Місця зберігання
                  </h1>
                  <p className="text-gray-600 text-base sm:text-lg leading-
relaxed">
                    Створюйте стелажі, переглядайте комірки та контролюйте,
які з
                    них вільні, а які вже зайняті товарами.
                  </p>
                </div>
              </div>
```

```
                <div className="rounded-[1.5rem] border border-green-100 bg-
[#fcfffd] p-5 sm:p-6 shadow-sm mb-6">
                  <div className="grid grid-cols-1 md:grid-cols-
[1fr_220px_auto] gap-4">
                    <input
                      type="text"
                      name="rackName"
                      value={form.rackName}
                      onChange={handleChange}
                      placeholder="Назва стелажа"
                      className="w-full rounded-2xl border border-gray-200 bg-
white px-4 py-3 text-gray-800 placeholder:text-gray-400 focus:outline-none
focus:ring-2 focus:ring-green-500 focus:border-green-500 transition-all
duration-300"
                    />

                    <input
                      type="number"
                      min="1"
                      name="cellsCount"
                      value={form.cellsCount}
                      onChange={handleChange}
                      placeholder="Кількість комірок"
                      className="w-full rounded-2xl border border-gray-200 bg-
white px-4 py-3 text-gray-800 placeholder:text-gray-400 focus:outline-none
focus:ring-2 focus:ring-green-500 focus:border-green-500 transition-all
duration-300"
                    />

                    <button
                      type="button"
                      onClick={handleAddRack}
                      className="inline-flex items-center justify-center gap-2
rounded-xl border border-green-500 px-4 py-3 text-sm font-semibold text-
green-600 transition-all duration-300 hover:bg-green-500 hover:text-white"
                    >
                      <Plus size={16} />
                      Додати стелаж
                    </button>
                  </div>
                </div>

                <div className="rounded-[1.5rem] border border-green-100 bg-
[#fcfffd] p-5 sm:p-6 shadow-sm mb-6">
                  <div className="relative">
                    <Search
                      className="absolute left-4 top-1/2 -translate-y-1/2
text-gray-400"
                      size={18}
                    />
                    <input
                      type="text"
                      value={searchTerm}
                      onChange={(e) => setSearchTerm(e.target.value)}
                      placeholder="Пошук по стелажах"
                      className="w-full rounded-2xl border border-gray-200 bg-
white pl-11 pr-4 py-3 text-gray-800 placeholder:text-gray-400 focus:outline-
none focus:ring-2 focus:ring-green-500 focus:border-green-500 transition-all
duration-300"
                    />
                  </div>
                </div>
```

```
                <div className="rounded-[1.5rem] border border-green-100 bg-
white shadow-sm overflow-hidden">
                  {loading ? (
                    <div className="px-6 py-12 text-center text-gray-500">
                      Завантаження стелажів...
                    </div>
                  ) : filteredRacks.length === 0 ? (
                    <div className="px-6 py-14 text-center">
                      <h3 className="text-2xl font-bold text-gray-900 mb-3">
                        Стелажів поки немає
                      </h3>
                      <p className="text-gray-600 max-w-2xl mx-auto leading-
relaxed">
                        Введіть назву стелажа та кількість комірок у формі
вище, а
                        потім натисніть кнопку додавання. Після цього стелаж
з’явиться
                        в списку.
                      </p>
                    </div>
                  ) : (
                    <>
                      <div className="hidden lg:grid grid-cols-
[70px_1.2fr_1.4fr_120px] gap-4 bg-green-50/70 border-b border-green-100 px-6
py-4 text-sm font-bold text-gray-800">
                        <div>№</div>
                        <div>Стелаж</div>
                        <div>Комірки</div>
                        <div className="text-center">Дії</div>
                      </div>

                      <div className="divide-y divide-green-100">
                        {filteredRacks.map((rack, index) => {
                          const cells = rack.cells || [];
                          const busyCells = cells.filter(
                            (cell) => cell.status === "occupied",
                          );
                          const freeCells = cells.filter(
                            (cell) => cell.status === "free",
                          );

                          return (
                            <div
                              key={rack.id}
                              className="px-4 sm:px-6 py-4 hover:bg-green-
50/40 transition-all duration-300"
                            >
                              <div className="hidden lg:grid grid-cols-
[70px_1.2fr_1.4fr_120px] gap-4 items-start">
                                <div className="text-gray-800 font-semibold
pt-1">
                                  {index + 1}
                                </div>

                                <div>
                                  <div className="text-gray-900 font-bold
text-base">
                                    {rack.name}
                                  </div>
                                  <div className="text-sm text-gray-600 mt-1">
                                    Усього комірок: {cells.length}
                                  </div>
```

```
                                    <div className="text-sm text-gray-500 mt-1">
                                      Створено: {formatDate(rack.createdAt)}
                                    </div>
                                  </div>

                                  <div>
                                    <div className="flex flex-wrap gap-2 mb-3">
                                      {cells.map((cell, cellIndex) => {
                                        const productInCell = products.find(
                                          (product) =>
                                            product.rackId === rack.id &&
                                            Array.isArray(product.cellCodes) &&
product.cellCodes.includes(cell.code),
                                        );

                                        return (
                                          <span
                                            key={cellIndex}
                                            title={
                                              productInCell ? productInCell.name
: ""
                                            }
                                            className={`inline-flex items-center
justify-center min-w-[54px] px-2.5 py-1.5 rounded-lg text-xs font-semibold
border ${
                                              cell.status === "occupied"
                                                ? "bg-red-50 text-red-700
border-red-200"
                                                : "bg-green-50 text-green-700
border-green-200"
                                            }`}
                                          >
                                            {cell.code}
                                          </span>
                                        );
                                      })}
                                    </div>

                                    <div className="flex flex-wrap gap-4 text-
sm">
                                      <span className="text-green-700 font-
medium">
                                        Вільні: {freeCells.length}
                                      </span>
                                      <span className="text-red-700 font-
medium">
                                        Зайняті: {busyCells.length}
                                      </span>
                                    </div>
                                  </div>

                                  <div className="flex items-center justify-
center pt-1">
                                    <button
                                      onClick={() => handleDeleteRack(rack)}
                                      className="inline-flex items-center
justify-center w-10 h-10 rounded-xl border border-red-200 text-red-600
hover:bg-red-500 hover:text-white transition-all duration-300"
                                      title="Видалити"
                                    >
                                      <Trash2 size={17} />
                                    </button>
```

```
                                </div>
                              </div>

                              <div className="lg:hidden rounded-2xl border
border-green-100 bg-[#fcfffd] p-4">
                                <div className="flex items-start justify-
between gap-3 mb-3">
                                  <div>
                                    <div className="text-xs text-gray-500 mb-
1">
                                      Стелаж №{index + 1}
                                    </div>
                                    <div className="text-gray-900 font-bold
text-lg">
                                      {rack.name}
                                    </div>
                                    <div className="text-sm text-gray-600 mt-
1">
                                      Усього комірок: {cells.length}
                                    </div>
                                    <div className="text-sm text-gray-500 mt-
1">
                                      Створено: {formatDate(rack.createdAt)}
                                    </div>
                                  </div>

                                  <button
                                    onClick={() => handleDeleteRack(rack)}
                                    className="inline-flex items-center
justify-center w-10 h-10 rounded-xl border border-red-200 text-red-600
hover:bg-red-500 hover:text-white transition-all duration-300"
                                    title="Видалити"
                                  >
                                    <Trash2 size={17} />
                                  </button>
                                </div>

                                <div className="flex flex-wrap gap-2 mb-3">
                                  {cells.map((cell, cellIndex) => (
                                    <span
                                      key={cellIndex}
                                      className={`inline-flex items-center
justify-center min-w-[54px] px-2.5 py-1.5 rounded-lg text-xs font-semibold
border ${
                                        cell.status === "occupied"
                                          ? "bg-red-50 text-red-700 border-
red-200"
                                          : "bg-green-50 text-green-700
border-green-200"
                                      }`}
                                    >
                                      {cell.code}
                                    </span>
                                  ))}
                                </div>

                                <div className="flex flex-wrap gap-4 text-sm">
                                  <span className="text-green-700 font-
medium">
                                    Вільні: {freeCells.length}
                                  </span>
                                  <span className="text-red-700 font-medium">
                                    Зайняті: {busyCells.length}
```

```
                            </span>
                          </div>
                        </div>
                      </div>
                    );
                  })}
                </div>
              </>
            )}
          </div>
        </div>
      </div>
    </main>
  );
}
```

Сторінка руху товарів, компонент InventoryMovements.jsx:

```
import { useCallback, useEffect, useMemo, useState } from "react";
import { ArrowLeftRight, Search, Plus } from "lucide-react";
import { useAuth } from "../context/AuthContext";
import { db } from "../firebase";
import {
  collection,
  doc,
  getDoc,
  getDocs,
  query,
  serverTimestamp,
  where,
  writeBatch,
} from "firebase/firestore";
import Swal from "sweetalert2";

const operationLabels = {
  incoming: "Надходження",
  writeoff: "Списання",
  return: "Повернення",
  transfer: "Переміщення",
};

export default function InventoryMovements() {
  const { user } = useAuth();

  const [activeOperation, setActiveOperation] = useState("incoming");
  const [products, setProducts] = useState([]);
  const [racks, setRacks] = useState([]);
  const [movements, setMovements] = useState([]);
  const [loading, setLoading] = useState(true);

  const [form, setForm] = useState({
    productId: "",
    quantity: "",
    note: "",
    targetRackId: "",
    targetCellCodes: [],
  });

  const [searchTerm, setSearchTerm] = useState("");

  const toast = useCallback((icon, title) => {
    Swal.fire({
```

```
            toast: true,
            position: "top-end",
            icon,
            title,
            showConfirmButton: false,
            timer: 3500,
            timerProgressBar: true,
          });
        }, []);


        const formatDateTime = (timestamp) => {
          if (!timestamp?.toDate) return "—";
          return timestamp.toDate().toLocaleString("uk-UA", {
            day: "2-digit",
            month: "2-digit",
            year: "numeric",
            hour: "2-digit",
            minute: "2-digit",
          });
        };

        const sortByCreatedAtDesc = (items) => {
          return [...items].sort((a, b) => {
            const aSec = a.createdAt?.seconds || 0;
            const bSec = b.createdAt?.seconds || 0;
            return bSec - aSec;
          });
        };

        const fetchData = useCallback(async () => {
          if (!user) return;

          setLoading(true);

          try {
            const [productsSnapshot, racksSnapshot, movementsSnapshot] = await
Promise.all([
              getDocs(query(collection(db, "products"), where("userId", "==",
user.uid))),
              getDocs(query(collection(db, "storageLocations"),
where("userId", "==", user.uid))),
              getDocs(query(collection(db, "stock_movements"), where("userId",
"==", user.uid))),
            ]);

            const productsData = productsSnapshot.docs.map((item) => ({
              id: item.id,
              ...item.data(),
            }));

            const racksData = racksSnapshot.docs.map((item) => ({
              id: item.id,
              ...item.data(),
            }));

            const movementsData = movementsSnapshot.docs.map((item) => ({
              id: item.id,
              ...item.data(),
            }));

            setProducts(sortByCreatedAtDesc(productsData));
            setRacks(sortByCreatedAtDesc(racksData));
```

```
      setMovements(sortByCreatedAtDesc(movementsData));
    } catch (error) {
      toast("error", "Не вдалося завантажити дані руху товарів");
    } finally {
      setLoading(false);
    }
  }, [user, toast]);

  useEffect(() => {
    fetchData();
  }, [fetchData]);

  const selectedProduct = useMemo(() => {
    return products.find((item) => item.id === form.productId) || null;
  }, [products, form.productId]);

  const selectedTargetRack = useMemo(() => {
    return racks.find((item) => item.id === form.targetRackId) || null;
  }, [racks, form.targetRackId]);

  const availableTargetCells = useMemo(() => {
    if (!selectedTargetRack || !selectedProduct) return [];

    return (selectedTargetRack.cells || []).map((cell) => {
      const occupiedByAnother = products.find(
        (item) =>
          item.id !== selectedProduct.id &&
          item.rackId === selectedTargetRack.id &&
          Array.isArray(item.cellCodes) &&
          item.cellCodes.includes(cell.code)
      );

      const currentOwns =
        selectedProduct.rackId === selectedTargetRack.id &&
        Array.isArray(selectedProduct.cellCodes) &&
        selectedProduct.cellCodes.includes(cell.code);

      if (occupiedByAnother) {
        return {
          ...cell,
          status: "occupied",
          productName: occupiedByAnother.name || "",
        };
      }

      if (currentOwns) {
        return {
          ...cell,
          status: "selected",
          productName: selectedProduct.name || "",
        };
      }

      return {
        ...cell,
        status: "free",
        productName: "",
      };
    });
  }, [selectedTargetRack, selectedProduct, products]);

  const filteredMovements = useMemo(() => {
    const value = searchTerm.toLowerCase().trim();
```

```
  return movements.filter(
    (item) =>
      item.productName?.toLowerCase().includes(value) ||
      item.productSku?.toLowerCase().includes(value) ||
      item.typeLabel?.toLowerCase().includes(value) ||
      item.note?.toLowerCase().includes(value)
  );
}, [movements, searchTerm]);

const resetForm = () => {
  setForm({
    productId: "",
    quantity: "",
    note: "",
    targetRackId: "",
    targetCellCodes: [],
  });
};

const handleChange = (e) => {
  const { name, value } = e.target;

  if (name === "productId") {
    const product = products.find((item) => item.id === value);

    setForm((prev) => ({
      ...prev,
      productId: value,
      targetRackId: product?.rackId || "",
      targetCellCodes: product?.cellCodes || [],
    }));
    return;
  }

  if (name === "targetRackId") {
    setForm((prev) => ({
      ...prev,
      targetRackId: value,
      targetCellCodes: [],
    }));
    return;
  }

  setForm((prev) => ({
    ...prev,
    [name]: value,
  }));
};

const handleToggleTargetCell = (code, status) => {
  if (status === "occupied") return;

  setForm((prev) => {
    const exists = prev.targetCellCodes.includes(code);

    return {
      ...prev,
      targetCellCodes: exists
        ? prev.targetCellCodes.filter((item) => item !== code)
        : [...prev.targetCellCodes, code],
    };
  });
```

```
};

const createMovementRecord = async ({
  batch,
  product,
  type,
  quantity,
  note,
  beforeQty,
  afterQty,
  extra = {},
}) => {
  const movementRef = doc(collection(db, "stock_movements"));

  batch.set(movementRef, {
    userId: user.uid,
    productId: product.id,
    productName: product.name,
    productSku: product.sku,
    type,
    typeLabel: operationLabels[type],
    quantity,
    beforeQty,
    afterQty,
    note: note || "",
    createdAt: serverTimestamp(),
    ...extra,
  });
};

const handleSubmitOperation = async () => {
  if (!user) {
    toast("error", "Користувача не знайдено");
    return;
  }

  if (!form.productId || !form.quantity) {
    toast("error", "Оберіть товар і вкажіть кількість");
    return;
  }

  const quantity = Number(form.quantity);

  if (!Number.isInteger(quantity) || quantity <= 0) {
    toast("error", "Кількість має бути додатнім цілим числом");
    return;
  }

  const product = products.find((item) => item.id === form.productId);

  if (!product) {
    toast("error", "Товар не знайдено");
    return;
  }

  try {
    const batch = writeBatch(db);
    const productRef = doc(db, "products", product.id);
    const currentQty = Number(product.quantity) || 0;

    if (activeOperation === "incoming") {
      const nextQty = currentQty + quantity;
```

```
    batch.update(productRef, {
      quantity: nextQty,
    });

    await createMovementRecord({
      batch,
      product,
      type: "incoming",
      quantity,
      note: form.note,
      beforeQty: currentQty,
      afterQty: nextQty,
    });

    await batch.commit();
    toast("success", "Надходження успішно зафіксовано");
  }

  if (activeOperation === "writeoff") {
    if (quantity > currentQty) {
      toast("error", "Недостатня кількість товару для списання");
      return;
    }

    const nextQty = currentQty - quantity;

    batch.update(productRef, {
      quantity: nextQty,
    });

    await createMovementRecord({
      batch,
      product,
      type: "writeoff",
      quantity,
      note: form.note,
      beforeQty: currentQty,
      afterQty: nextQty,
    });

    await batch.commit();
    toast("success", "Списання успішно виконано");
  }

  if (activeOperation === "return") {
    const nextQty = currentQty + quantity;

    batch.update(productRef, {
      quantity: nextQty,
    });

    await createMovementRecord({
      batch,
      product,
      type: "return",
      quantity,
      note: form.note,
      beforeQty: currentQty,
      afterQty: nextQty,
    });

    await batch.commit();
    toast("success", "Повернення успішно зафіксовано");
```

```
            }

            if (activeOperation === "transfer") {
              if (!form.targetRackId || form.targetCellCodes.length === 0) {
                toast("error", "Оберіть стелаж і хоча б одну комірку");
                return;
              }

              const targetRackRef = doc(db, "storageLocations",
form.targetRackId);
              const targetRackSnap = await getDoc(targetRackRef);

              if (!targetRackSnap.exists()) {
                toast("error", "Стелаж не знайдено");
                return;
              }

              const targetRackData = targetRackSnap.data();
              const latestCells = targetRackData.cells || [];

              const busySelected = form.targetCellCodes.some((code) => {
                const occupiedByAnother = products.some(
                  (item) =>
                    item.id !== product.id &&
                    item.rackId === form.targetRackId &&
                    Array.isArray(item.cellCodes) &&
                    item.cellCodes.includes(code)
                );

                const exists = latestCells.some((item) => item.code === code);
                return !exists || occupiedByAnother;
              });

              if (busySelected) {
                toast("error", "Одна або кілька обраних комірок зайняті");
                return;
              }

              if (product.rackId) {
                const oldRackRef = doc(db, "storageLocations",
product.rackId);
                const oldRackSnap = await getDoc(oldRackRef);

                if (oldRackSnap.exists()) {
                  const oldRackData = oldRackSnap.data();
                  const releasedCells = (oldRackData.cells || []).map((cell)
=>
                    Array.isArray(product.cellCodes) &&
product.cellCodes.includes(cell.code)
                      ? { ...cell, status: "free" }
                      : cell
                  );

                  batch.update(oldRackRef, {
                    cells: releasedCells,
                  });
                }
              }

              const occupiedCells = latestCells.map((cell) =>
                form.targetCellCodes.includes(cell.code)
                  ? { ...cell, status: "occupied" }
                  : cell
```

```
            );

            batch.update(targetRackRef, {
              cells: occupiedCells,
            });

            const targetRack = racks.find((item) => item.id ===
form.targetRackId);

            batch.update(productRef, {
              rackId: form.targetRackId,
              rackName: targetRack?.name || "",
              cellCodes: form.targetCellCodes,
              storageLocation: `${targetRack?.name || ""}:
${form.targetCellCodes.join(", ")}`,
            });

            await createMovementRecord({
              batch,
              product,
              type: "transfer",
              quantity,
              note: form.note,
              beforeQty: currentQty,
              afterQty: currentQty,
              extra: {
                fromRackId: product.rackId || "",
                fromRackName: product.rackName || "",
                fromCellCodes: product.cellCodes || [],
                toRackId: form.targetRackId,
                toRackName: targetRack?.name || "",
                toCellCodes: form.targetCellCodes,
              },
            });

            await batch.commit();
            toast("success", "Переміщення успішно виконано");
          }

          resetForm();
          fetchData();
        } catch (error) {
          toast("error", "Не вдалося виконати операцію");
        }
      };

      return (
        <main className="min-h-screen bg-[#f6faf7] px-4 py-10">
          <div className="max-w-7xl mx-auto">
            <div className="rounded-[2rem] border border-green-100 bg-white
shadow-xl shadow-green-100/40 p-6 sm:p-8 md:p-10">
              <div className="flex items-center gap-4 mb-8">
                <div className="w-16 h-16 rounded-full bg-green-100 text-
green-600 flex items-center justify-center shrink-0">
                  <ArrowLeftRight size={30} />
                </div>

                <div>
                  <h1 className="text-3xl sm:text-4xl font-black text-gray-
900 mb-2">
                    Рух товарів
                  </h1>
```

```
                    <p className="text-gray-600 text-base sm:text-lg leading-
relaxed">
                      Створюйте операції руху товарів і переглядайте журнал
усіх змін по складу.
                    </p>
                  </div>
                </div>

                <div className="rounded-[1.5rem] border border-green-100 bg-
[#fcfffd] p-5 sm:p-6 shadow-sm mb-6">
                  <div className="flex flex-wrap gap-3 mb-6">
                    {Object.entries(operationLabels).map(([key, label]) => (
                      <button
                        key={key}
                        type="button"
                        onClick={() => {
                          setActiveOperation(key);
                          resetForm();
                        }}
                        className={`inline-flex items-center justify-center
rounded-xl px-4 py-2.5 text-sm font-semibold transition-all duration-300 ${
                          activeOperation === key
                            ? "bg-green-500 text-white"
                            : "border border-green-500 text-green-600
hover:bg-green-500 hover:text-white"
                        }`}
                      >
                        {label}
                      </button>
                    ))}
                  </div>

                  <div className="grid grid-cols-1 md:grid-cols-2 xl:grid-
cols-3 gap-4 mb-4">
                    <select
                      name="productId"
                      value={form.productId}
                      onChange={handleChange}
                      className="w-full rounded-2xl border border-gray-200 bg-
white px-4 py-3 text-gray-800 focus:outline-none focus:ring-2 focus:ring-
green-500 focus:border-green-500 transition-all duration-300"
                    >
                      <option value="">Оберіть товар*</option>
                      {products.map((item) => (
                        <option key={item.id} value={item.id}>
                          {item.name} ({item.sku})
                        </option>
                      ))}
                    </select>

                    <input
                      type="number"
                      min="1"
                      step="1"
                      name="quantity"
                      value={form.quantity}
                      onChange={handleChange}
                      placeholder="Кількість*"
                      className="w-full rounded-2xl border border-gray-200 bg-
white px-4 py-3 text-gray-800 placeholder:text-gray-400 focus:outline-none
focus:ring-2 focus:ring-green-500 focus:border-green-500 transition-all
duration-300"
                    />
```

```
                    <input
                      type="text"
                      name="note"
                      value={form.note}
                      onChange={handleChange}
                      placeholder="Коментар або примітка"
                      className="w-full rounded-2xl border border-gray-200 bg-
white px-4 py-3 text-gray-800 placeholder:text-gray-400 focus:outline-none
focus:ring-2 focus:ring-green-500 focus:border-green-500 transition-all
duration-300"
                    />
                  </div>

                  {activeOperation === "transfer" && (
                    <div className="rounded-2xl border border-green-100 bg-
white p-4 mb-4">
                      <div className="grid grid-cols-1 md:grid-cols-2 gap-4
mb-4">
                        <select
                          name="targetRackId"
                          value={form.targetRackId}
                          onChange={handleChange}
                          className="w-full rounded-2xl border border-gray-200
bg-white px-4 py-3 text-gray-800 focus:outline-none focus:ring-2 focus:ring-
green-500 focus:border-green-500 transition-all duration-300"
                        >
                          <option value="">Оберіть стелаж*</option>
                          {racks.map((item) => (
                            <option key={item.id} value={item.id}>
                              {item.name}
                            </option>
                          ))}
                        </select>
                      </div>

                      {form.targetRackId && (
                        <>
                          <div className="text-sm font-semibold text-gray-800
mb-3">
                            Оберіть комірки для переміщення
                          </div>

                          <div className="flex flex-wrap gap-2">
                            {availableTargetCells.map((cell, index) => {
                              const checked =
form.targetCellCodes.includes(cell.code);
                              const occupied = cell.status === "occupied";

                              return (
                                <label
                                  key={index}
                                  title={occupied ? cell.productName || "" :
""}
                                  className={`inline-flex items-center gap-2
px-3 py-2 rounded-xl border text-sm font-medium transition-all duration-300
${
                                    occupied
                                      ? "border-red-200 bg-red-50 text-red-600
cursor-not-allowed"
                                      : checked
                                      ? "border-green-500 bg-green-500 text-
white cursor-pointer"
```

```
                                        : "border-green-200 bg-green-50 text-
green-700 cursor-pointer hover:bg-green-100"
                                    }`}
                                  >
                                    <input
                                      type="checkbox"
                                      checked={checked}
                                      disabled={occupied}
                                      onChange={() =>
handleToggleTargetCell(cell.code, cell.status)}
                                      className="hidden"
                                    />
                                    {cell.code}
                                    <span className="text-xs opacity-90">
                                      {occupied ? "зайнята" : checked ? "обрана"
: "вільна"}
                                    </span>
                                  </label>
                                );
                              })}
                            </div>
                          </>
                        )}
                      </div>
                    )}

                    <div className="flex justify-end">
                      <button
                        type="button"
                        onClick={handleSubmitOperation}
                        className="inline-flex items-center justify-center gap-2
rounded-xl border border-green-500 px-4 py-3 text-sm font-semibold text-
green-600 transition-all duration-300 hover:bg-green-500 hover:text-white"
                      >
                        <Plus size={16} />
                        Підтвердити операцію
                      </button>
                    </div>
                  </div>

                  <div className="rounded-[1.5rem] border border-green-100 bg-
[#fcfffd] p-5 sm:p-6 shadow-sm mb-6">
                    <div className="relative">
                      <Search
                        className="absolute left-4 top-1/2 -translate-y-1/2
text-gray-400"
                        size={18}
                      />
                      <input
                        type="text"
                        value={searchTerm}
                        onChange={(e) => setSearchTerm(e.target.value)}
                        placeholder="Пошук по журналу руху"
                        className="w-full rounded-2xl border border-gray-200 bg-
white pl-11 pr-4 py-3 text-gray-800 placeholder:text-gray-400 focus:outline-
none focus:ring-2 focus:ring-green-500 focus:border-green-500 transition-all
duration-300"
                      />
                    </div>
                  </div>

                  <div className="rounded-[1.5rem] border border-green-100 bg-
white shadow-sm overflow-hidden">
```

```
                {loading ? (
                  <div className="px-6 py-12 text-center text-gray-500">
                    Завантаження журналу...
                  </div>
                ) : filteredMovements.length === 0 ? (
                  <div className="px-6 py-14 text-center">
                    <h3 className="text-2xl font-bold text-gray-900 mb-3">
                      Журнал руху поки порожній
                    </h3>
                    <p className="text-gray-600 max-w-2xl mx-auto leading-
relaxed">
                      Створіть першу операцію руху товару, і вона з’явиться
у журналі нижче.
                    </p>
                  </div>
                ) : (
                  <>
                    <div className="hidden xl:grid grid-cols-
[70px_1.2fr_160px_140px_1fr_160px] gap-4 bg-green-50/70 border-b border-
green-100 px-6 py-4 text-sm font-bold text-gray-800">
                      <div>№</div>
                      <div>Товар</div>
                      <div>Операція</div>
                      <div>Кількість</div>
                      <div>Деталі</div>
                      <div>Дата</div>
                    </div>

                    <div className="divide-y divide-green-100">
                      {filteredMovements.map((movement, index) => (
                        <div
                          key={movement.id}
                          className="px-4 sm:px-6 py-4 hover:bg-green-50/40
transition-all duration-300"
                        >
                          <div className="hidden xl:grid grid-cols-
[70px_1.2fr_160px_140px_1fr_160px] gap-4 items-start">
                            <div className="text-gray-800 font-semibold pt-
1">
                              {index + 1}
                            </div>

                            <div>
                              <div className="text-gray-900 font-bold text-
base">
                                {movement.productName}
                              </div>
                              <div className="text-sm text-gray-600 mt-1">
                                Артикул: {movement.productSku}
                              </div>
                            </div>

                            <div className="text-gray-800 font-medium">
                              {movement.typeLabel}
                            </div>

                            <div className="text-gray-700 font-medium">
                              {movement.quantity}
                            </div>

                            <div className="space-y-1 text-sm text-gray-
600">
                              <div>
```

```
                                  Було: {movement.beforeQty} | Стало:
{movement.afterQty}
                                </div>
                                {movement.type === "transfer" && (
                                  <div>
                                    {movement.fromRackName || "—"} →
{movement.toRackName || "—"}
                                  </div>
                                )}
                                {movement.note && <div>{movement.note}</div>}
                              </div>

                              <div className="text-gray-700">
                                {formatDateTime(movement.createdAt)}
                              </div>
                            </div>

                            <div className="xl:hidden rounded-2xl border
border-green-100 bg-[#fcfffd] p-4">
                              <div className="text-xs text-gray-500 mb-1">
                                Запис №{index + 1}
                              </div>
                              <div className="text-gray-900 font-bold text-lg
mb-1">
                                {movement.productName}
                              </div>
                              <div className="text-sm text-gray-600 mb-1">
                                Артикул: {movement.productSku}
                              </div>
                              <div className="text-sm text-gray-700 mb-1">
                                Операція: {movement.typeLabel}
                              </div>
                              <div className="text-sm text-gray-700 mb-1">
                                Кількість: {movement.quantity}
                              </div>
                              <div className="text-sm text-gray-700 mb-1">
                                Було: {movement.beforeQty} | Стало:
{movement.afterQty}
                              </div>
                              {movement.type === "transfer" && (
                                <div className="text-sm text-gray-700 mb-1">
                                  {movement.fromRackName || "—"} →
{movement.toRackName || "—"}
                                </div>
                              )}
                              {movement.note && (
                                <div className="text-sm text-gray-600 mb-1">
                                  {movement.note}
                                </div>
                              )}
                              <div className="text-sm text-gray-500">
                                {formatDateTime(movement.createdAt)}
                              </div>
                            </div>
                          </div>
                        ))}
                      </div>
                    </>
                  )}
                </div>
              </div>
            </main>
```

```
  );
}
```

Сторінка відвантажень, компонент Shipments.jsx:

```
import { useCallback, useEffect, useMemo, useState } from "react";
import { Send, Plus, Search, Trash2, Eye } from "lucide-react";
import { useAuth } from "../context/AuthContext";
import { db } from "../firebase";
import {
  addDoc,
  collection,
  deleteDoc,
  doc,
  getDocs,
  query,
  serverTimestamp,
  updateDoc,
  where,
  writeBatch,
} from "firebase/firestore";
import Swal from "sweetalert2";

const shipmentStatuses = [
  { value: "new", label: "нове" },
  { value: "confirmed", label: "підтверджено" },
  { value: "shipped", label: "відвантажено" },
  { value: "canceled", label: "скасовано" },
];

export default function Shipments() {
  const getShipmentRowClass = (status) => {
    switch (status) {
      case "new":
        return "bg-blue-50/70 hover:bg-blue-100/70";
      case "confirmed":
        return "bg-amber-50/70 hover:bg-amber-100/70";
      case "shipped":
        return "bg-green-50/70 hover:bg-green-100/70";
      case "canceled":
        return "bg-red-50/70 hover:bg-red-100/70";
      default:
        return "hover:bg-green-50/40";
    }
  };

  const getShipmentStatusClass = (status) => {
    switch (status) {
      case "new":
        return "border-blue-200 bg-blue-50 text-blue-700";
      case "confirmed":
        return "border-amber-200 bg-amber-50 text-amber-700";
      case "shipped":
        return "border-green-200 bg-green-50 text-green-700";
      case "canceled":
        return "border-red-200 bg-red-50 text-red-700";
      default:
        return "border-gray-200 bg-white text-gray-800";
    }
  };

  const { user } = useAuth();
```

```
        const [products, setProducts] = useState([]);
        const [shipments, setShipments] = useState([]);
        const [loading, setLoading] = useState(true);
        const [productSearch, setProductSearch] = useState("");
        const [isProductDropdownOpen, setIsProductDropdownOpen] =
useState(false);
        const [invoiceForm, setInvoiceForm] = useState({
          invoiceNumber: "",
          customerName: "",
          customerPhone: "",
          note: "",
        });

        const [itemForm, setItemForm] = useState({
          productId: "",
          quantity: "",
        });

        const [invoiceItems, setInvoiceItems] = useState([]);
        const [searchTerm, setSearchTerm] = useState("");

        const toast = useCallback((icon, title) => {
          Swal.fire({
            toast: true,
            position: "top-end",
            icon,
            title,
            showConfirmButton: false,
            timer: 3500,
            timerProgressBar: true,
          });
        }, []);

        const formatDateTime = (timestamp) => {
          if (!timestamp?.toDate) return "—";
          return timestamp.toDate().toLocaleString("uk-UA", {
            day: "2-digit",
            month: "2-digit",
            year: "numeric",
            hour: "2-digit",
            minute: "2-digit",
          });
        };

        const sortByCreatedAtDesc = (items) => {
          return [...items].sort((a, b) => {
            const aSec = a.createdAt?.seconds || 0;
            const bSec = b.createdAt?.seconds || 0;
            return bSec - aSec;
          });
        };

        const fetchData = useCallback(async () => {
          if (!user) return;

          setLoading(true);

          try {
            const [productsSnapshot, shipmentsSnapshot] = await Promise.all([
              getDocs(
                query(collection(db, "products"), where("userId", "==",
user.uid)),
```

```
            ),
            getDocs(
              query(collection(db, "shipments"), where("userId", "==",
user.uid)),
            ),
          ]);

          const productsData = productsSnapshot.docs.map((item) => ({
            id: item.id,
            ...item.data(),
          }));

          const shipmentsData = shipmentsSnapshot.docs.map((item) => ({
            id: item.id,
            ...item.data(),
          }));

          setProducts(sortByCreatedAtDesc(productsData));
          setShipments(sortByCreatedAtDesc(shipmentsData));
        } catch (error) {
          toast("error", "Не вдалося завантажити відвантаження");
        } finally {
          setLoading(false);
        }
      }, [user, toast]);

      useEffect(() => {
        fetchData();
      }, [fetchData]);

      const selectedProduct = useMemo(() => {
        return products.find((item) => item.id === itemForm.productId) ||
null;
      }, [products, itemForm.productId]);

      const filteredShipments = useMemo(() => {
        const value = searchTerm.toLowerCase().trim();

        return shipments.filter(
          (item) =>
            item.invoiceNumber?.toLowerCase().includes(value) ||
            item.customerName?.toLowerCase().includes(value) ||
            item.statusLabel?.toLowerCase().includes(value) ||
            item.customerPhone?.toLowerCase().includes(value) ||
            item.note?.toLowerCase().includes(value),
        );
      }, [shipments, searchTerm]);
      const filteredProductOptions = useMemo(() => {
      const value = productSearch.toLowerCase().trim();

      return products.filter(
        (item) =>
          item.name?.toLowerCase().includes(value) ||
          item.sku?.toLowerCase().includes(value)
      );
    }, [products, productSearch]);
      const handleInvoiceChange = (e) => {
        const { name, value } = e.target;
        setInvoiceForm((prev) => ({
          ...prev,
          [name]: value,
        }));
      };
```

```
  const handleItemChange = (e) => {
    const { name, value } = e.target;
    setItemForm((prev) => ({
      ...prev,
      [name]: value,
    }));
  };

  const resetForms = () => {
    setInvoiceForm({
      invoiceNumber: "",
      customerName: "",
      customerPhone: "",
      note: "",
    });

    setItemForm({
      productId: "",
      quantity: "",
    });

    setInvoiceItems([]);
    setProductSearch("");
  setIsProductDropdownOpen(false);
  };

  const handleAddInvoiceItem = () => {
    if (!itemForm.productId || !itemForm.quantity) {
      toast("error", "Оберіть товар і вкажіть кількість");
      return;
    }

  if (!invoiceForm.customerPhone.trim()) {
    toast("error", "Вкажіть номер телефону одержувача");
    return;
  }
      const quantity = Number(itemForm.quantity);

      if (!Number.isInteger(quantity) || quantity <= 0) {
        toast("error", "Кількість має бути додатнім цілим числом");
        return;
      }

      const product = products.find((item) => item.id ===
itemForm.productId);

      if (!product) {
        toast("error", "Товар не знайдено");
        return;
      }
      if (quantity > Number(product.quantity || 0)) {
        toast("error", "Не можна додати більше товару, ніж є на складі");
        return;
      }
      const afterQty = Number(product.quantity || 0) - quantity;
      const minStock = Number(product.minStock || 0);

      if (afterQty < minStock) {
        toast(
          "warning",
          "Після відвантаження залишок буде нижчим за мінімальний",
        );
```

```
          }
          const exists = invoiceItems.some((item) => item.productId ===
product.id);

          if (exists) {
            toast("error", "Цей товар вже доданий до накладної");
            return;
          }

          setInvoiceItems((prev) => [
            ...prev,
            {
              productId: product.id,
              productName: product.name,
              productSku: product.sku,
              quantity,
              unit: product.unit,
              currentQuantity: product.quantity,
            },
          ]);

          setItemForm({
            productId: "",
            quantity: "",
          });
        };

        const handleRemoveInvoiceItem = (productId) => {
          setInvoiceItems((prev) =>
            prev.filter((item) => item.productId !== productId),
          );
        };

        const handleCreateShipment = async () => {
          if (!user) {
            toast("error", "Користувача не знайдено");
            return;
          }

          const invoiceNumber = invoiceForm.invoiceNumber.trim();
          const customerName = invoiceForm.customerName.trim();
          const customerPhone = invoiceForm.customerPhone.trim();
          const note = invoiceForm.note.trim();

          if (!invoiceNumber || !customerName) {
            toast("error", "Заповніть номер накладної та одержувача");
            return;
          }

          if (invoiceItems.length === 0) {
            toast("error", "Додайте хоча б один товар у накладну");
            return;
          }
      if (!customerPhone) {
            toast("error", "Вкажіть номер телефону одержувача");
            return;
      }
          const exists = shipments.some(
            (item) =>
              item.invoiceNumber?.toLowerCase() ===
invoiceNumber.toLowerCase(),
          );
```

```
          if (exists) {
            toast("error", "Накладна з таким номером уже існує");
            return;
          }

          try {
            await addDoc(collection(db, "shipments"), {
              userId: user.uid,
              invoiceNumber,
              customerName,
              customerPhone: invoiceForm.customerPhone.trim(),
              note,
              items: invoiceItems,
              status: "new",
              statusLabel: "нове",
              appliedToStock: false,
              createdAt: serverTimestamp(),
            });

            resetForms();
            toast("success", "Накладну успішно створено");
            fetchData();
          } catch (error) {
            toast("error", "Не вдалося створити накладну");
          }
        };

        const handleViewItems = async (shipment) => {
          const rows = (shipment.items || [])
            .map(
              (item, index) => `
                <tr>
                  <td style="padding:8px;border-bottom:1px solid
#e5e7eb;">${index + 1}</td>
                  <td style="padding:8px;border-bottom:1px solid
#e5e7eb;">${item.productName}</td>
                  <td style="padding:8px;border-bottom:1px solid
#e5e7eb;">${item.productSku}</td>
                  <td style="padding:8px;border-bottom:1px solid
#e5e7eb;">${item.quantity} ${item.unit || ""}</td>
                </tr>
              `,
            )
            .join("");

          await Swal.fire({
            title: `Накладна ${shipment.invoiceNumber}`,
            html: `
              <div style="text-align:left;margin-bottom:12px;">
                <div><strong>Одержувач:</strong> ${shipment.customerName || "—
"}</div>
                <div><strong>Телефон:</strong> ${shipment.customerPhone || "—
"}</div>
                <div><strong>Статус:</strong> ${shipment.statusLabel || "—
"}</div>
                <div><strong>Дата:</strong>
${formatDateTime(shipment.createdAt)}</div>
                ${shipment.note ? `<div><strong>Примітка:</strong>
${shipment.note}</div>` : ""}
              </div>
              <div style="overflow:auto;">
                <table style="width:100%;border-collapse:collapse;text-
align:left;">
```

```
                  <thead>
                    <tr>
                      <th style="padding:8px;border-bottom:1px solid
#d1d5db;">№</th>
                      <th style="padding:8px;border-bottom:1px solid
#d1d5db;">Товар</th>
                      <th style="padding:8px;border-bottom:1px solid
#d1d5db;">Артикул</th>
                      <th style="padding:8px;border-bottom:1px solid
#d1d5db;">Кількість</th>
                    </tr>
                  </thead>
                  <tbody>${rows}</tbody>
                </table>
              </div>
            `,
            width: 900,
            confirmButtonColor: "#22c55e",
            confirmButtonText: "Закрити",
          });
        };

        const handleStatusChange = async (shipment, nextStatus) => {
          if (shipment.status === nextStatus) return;

          const nextStatusLabel =
            shipmentStatuses.find((item) => item.value === nextStatus)?.label
||
            nextStatus;

          try {
            if (nextStatus === "shipped") {
              if (shipment.appliedToStock) {
                await updateDoc(doc(db, "shipments", shipment.id), {
                  status: nextStatus,
                  statusLabel: nextStatusLabel,
                });

                toast("success", "Статус накладної оновлено");
                fetchData();
                return;
              }

              const currentProductsSnapshot = await getDocs(
                query(collection(db, "products"), where("userId", "==",
user.uid)),
              );

              const currentProducts = currentProductsSnapshot.docs.map((item)
=> ({
                id: item.id,
                ...item.data(),
              }));

              const insufficient = [];
              const belowMinimum = [];

              for (const line of shipment.items || []) {
                const product = currentProducts.find(
                  (item) => item.id === line.productId,
                );

                if (!product) {
```

```
                insufficient.push({
                  productName: line.productName || "Невідомий товар",
                  currentQty: 0,
                  shipmentQty: Number(line.quantity) || 0,
                });
                continue;
              }

              const currentQty = Number(product.quantity) || 0;
              const shipmentQty = Number(line.quantity) || 0;
              const minStock = Number(product.minStock) || 0;
              const afterQty = currentQty - shipmentQty;

              if (shipmentQty > currentQty) {
                insufficient.push({
                  productName: product.name,
                  currentQty,
                  shipmentQty,
                });
              } else if (afterQty < minStock) {
                belowMinimum.push({
                  productName: product.name,
                  afterQty,
                  minStock,
                });
              }
            }

            if (insufficient.length > 0) {
              toast(
                "error",
                "Недостатня кількість одного з товарів для відвантаження",
              );
              return;
            }

            if (belowMinimum.length > 0) {
              const warningText = belowMinimum
                .map(
                  (item) =>
                    `${item.productName}: залишок буде ${item.afterQty},
мінімум ${item.minStock}`,
                )
                .join("<br>");

              const confirmMinimum = await Swal.fire({
                title: "Увага до залишків",
                html: `Після відвантаження деякі товари опустяться нижче
мінімального залишку:<br><br>${warningText}<br><br>Продовжити?`,
                icon: "warning",
                showCancelButton: true,
                confirmButtonText: "Так, продовжити",
                cancelButtonText: "Скасувати",
                confirmButtonColor: "#22c55e",
                cancelButtonColor: "#9ca3af",
              });

              if (!confirmMinimum.isConfirmed) return;
            }

            const batch = writeBatch(db);

            for (const line of shipment.items || []) {
```

```
          const product = currentProducts.find(
            (item) => item.id === line.productId,
          );
          if (!product) continue;

          const productRef = doc(db, "products", product.id);
          const beforeQty = Number(product.quantity) || 0;
          const afterQty = beforeQty - Number(line.quantity);

          batch.update(productRef, {
            quantity: afterQty,
          });

          const movementRef = doc(collection(db, "stock_movements"));

          batch.set(movementRef, {
            userId: user.uid,
            productId: product.id,
            productName: product.name,
            productSku: product.sku,
            type: "outgoing",
            typeLabel: "Відвантаження",
            quantity: Number(line.quantity),
            beforeQty,
            afterQty,
            note: `Накладна ${shipment.invoiceNumber}`,
            shipmentId: shipment.id,
            shipmentNumber: shipment.invoiceNumber,
            createdAt: serverTimestamp(),
          });
        }

        const shipmentRef = doc(db, "shipments", shipment.id);

        batch.update(shipmentRef, {
          status: nextStatus,
          statusLabel: nextStatusLabel,
          appliedToStock: true,
        });

        await batch.commit();

        toast("success", "Накладну відвантажено");
        fetchData();
        return;
      }

      await updateDoc(doc(db, "shipments", shipment.id), {
        status: nextStatus,
        statusLabel: nextStatusLabel,
      });

      toast("success", "Статус накладної оновлено");
      fetchData();
    } catch (error) {
      toast("error", "Не вдалося змінити статус накладної");
    }
  };

  const handleDeleteShipment = async (shipment) => {
    const result = await Swal.fire({
      title: "Ви впевнені?",
      text: "Накладну буде видалено без можливості відновлення.",
```

```
            icon: "warning",
            showCancelButton: true,
            confirmButtonText: "Так, видалити",
            cancelButtonText: "Скасувати",
            confirmButtonColor: "#ef4444",
            cancelButtonColor: "#9ca3af",
            reverseButtons: true,
          });

          if (!result.isConfirmed) return;

          if (shipment.status === "shipped") {
            toast("error", "Відвантажену накладну видаляти не можна");
            return;
          }

          try {
            await deleteDoc(doc(db, "shipments", shipment.id));
            toast("success", "Накладну видалено");
            fetchData();
          } catch (error) {
            toast("error", "Не вдалося видалити накладну");
          }
        };
        return (
          <main className="min-h-screen bg-[#f6faf7] px-4 py-10">
            <div className="max-w-7xl mx-auto">
              <div className="rounded-[2rem] border border-green-100 bg-white
shadow-xl shadow-green-100/40 p-6 sm:p-8 md:p-10">
                <div className="flex items-center gap-4 mb-8">
                  <div className="w-16 h-16 rounded-full bg-green-100 text-
green-600 flex items-center justify-center shrink-0">
                    <Send size={30} />
                  </div>

                  <div>
                    <h1 className="text-3xl sm:text-4xl font-black text-gray-
900 mb-2">
                      Відвантаження
                    </h1>
                    <p className="text-gray-600 text-base sm:text-lg leading-
relaxed">
                      Створюйте накладні, змінюйте їх статуси та контролюйте
процес
                      відвантаження товарів.
                    </p>
                  </div>
                </div>

                <div className="rounded-[1.5rem] border border-green-100 bg-
[#fcfffd] p-5 sm:p-6 shadow-sm mb-6">
                  <div className="grid grid-cols-1 md:grid-cols-2 xl:grid-
cols-3 gap-4 mb-4">
                    <input
                      type="text"
                      name="invoiceNumber"
                      value={invoiceForm.invoiceNumber}
                      onChange={handleInvoiceChange}
                      placeholder="Номер накладної*"
                      className="w-full rounded-2xl border border-gray-200 bg-
white px-4 py-3 text-gray-800 placeholder:text-gray-400 focus:outline-none
focus:ring-2 focus:ring-green-500 focus:border-green-500 transition-all
duration-300"
```

```
                    />

                    <input
                      type="text"
                      name="customerName"
                      value={invoiceForm.customerName}
                      onChange={handleInvoiceChange}
                      placeholder="Одержувач*"
                      className="w-full rounded-2xl border border-gray-200 bg-
white px-4 py-3 text-gray-800 placeholder:text-gray-400 focus:outline-none
focus:ring-2 focus:ring-green-500 focus:border-green-500 transition-all
duration-300"
                    />

                    <input
                      type="text"
                      name="customerPhone"
                      value={invoiceForm.customerPhone}
                      onChange={handleInvoiceChange}
                      placeholder="Телефон одержувача"
                      className="w-full rounded-2xl border border-gray-200 bg-
white px-4 py-3 text-gray-800 placeholder:text-gray-400 focus:outline-none
focus:ring-2 focus:ring-green-500 focus:border-green-500 transition-all
duration-300"
                    />
                    <input
                      type="text"
                      name="note"
                      value={invoiceForm.note}
                      onChange={handleInvoiceChange}
                      placeholder="Примітка / Коментар"
                      className="w-full md:col-span-2 xl:col-span-3 rounded-
2xl border border-gray-200 bg-white px-4 py-3 text-gray-800 placeholder:text-
gray-400 focus:outline-none focus:ring-2 focus:ring-green-500 focus:border-
green-500 transition-all duration-300"
                    />
                  </div>

                  <div className="rounded-2xl border border-green-100 bg-white
p-4 mb-4">
                    <div className="grid grid-cols-1 md:grid-cols-
[1fr_180px_auto] gap-4 mb-4">
      <div className="relative">
        <input
          type="text"
          value={productSearch}
          onChange={(e) => {
            setProductSearch(e.target.value);
            setIsProductDropdownOpen(true);
          }}
          onFocus={() => setIsProductDropdownOpen(true)}
          placeholder="Пошук товару за назвою або артикулом"
          className="w-full rounded-2xl border border-gray-200 bg-white px-4
py-3 text-gray-800 placeholder:text-gray-400 focus:outline-none focus:ring-2
focus:ring-green-500 focus:border-green-500 transition-all duration-300"
        />

        {isProductDropdownOpen && (
          <div className="absolute z-20 mt-2 w-full max-h-64 overflow-y-auto
rounded-2xl border border-green-100 bg-white shadow-lg">
            {filteredProductOptions.length > 0 ? (
              filteredProductOptions.map((item) => (
                <button
```

```
                  key={item.id}
                  type="button"
                  onClick={() => {
                    setItemForm((prev) => ({
                      ...prev,
                      productId: item.id,
                    }));
                    setProductSearch(`${item.name} (${item.sku})`);
                    setIsProductDropdownOpen(false);
                  }}
                  className="w-full px-4 py-3 text-left text-gray-700
hover:bg-green-50 transition-all duration-200"
                >
                  {item.name} ({item.sku})
                </button>
              ))
            ) : (
              <div className="px-4 py-3 text-sm text-gray-500">
                Нічого не знайдено
              </div>
            )}
          </div>
        )}
      </div>

                      <input
                        type="number"
                        min="1"
                        step="1"
                        name="quantity"
                        value={itemForm.quantity}
                        onChange={handleItemChange}
                        placeholder="Кількість"
                        className="w-full rounded-2xl border border-gray-200
bg-white px-4 py-3 text-gray-800 placeholder:text-gray-400 focus:outline-none
focus:ring-2 focus:ring-green-500 focus:border-green-500 transition-all
duration-300"
                      />

                      <button
                        type="button"
                        onClick={handleAddInvoiceItem}
                        className="inline-flex items-center justify-center
gap-2 rounded-xl border border-green-500 px-4 py-3 text-sm font-semibold
text-green-600 transition-all duration-300 hover:bg-green-500 hover:text-
white"
                      >
                        <Plus size={16} />
                        Додати позицію
                      </button>
                    </div>

                    {selectedProduct && (
                      <div className="text-sm text-gray-600 mb-2">
                        Доступна кількість: {selectedProduct.quantity}{" "}
                        {selectedProduct.unit}
                      </div>
                    )}

                    {invoiceItems.length > 0 && (
                      <div className="rounded-2xl border border-green-100
overflow-hidden">
```

```
                        <div className="hidden md:grid grid-cols-
[1fr_160px_120px_70px] gap-4 bg-green-50/70 border-b border-green-100 px-4
py-3 text-sm font-bold text-gray-800">
                          <div>Товар</div>
                          <div>Артикул</div>
                          <div>Кількість</div>
                          <div></div>
                        </div>

                        <div className="divide-y divide-green-100">
                          {invoiceItems.map((item) => (
                            <div key={item.productId} className="px-4 py-3">
                              <div className="hidden md:grid grid-cols-
[1fr_160px_120px_70px] gap-4 items-center">
                                <div className="text-gray-900 font-medium">
                                  {item.productName}
                                </div>
                                <div className="text-gray-
600">{item.productSku}</div>
                                <div className="text-gray-700">
                                  {item.quantity} {item.unit}
                                </div>
                                <button
                                  type="button"
                                  onClick={() =>
                                    handleRemoveInvoiceItem(item.productId)
                                  }
                                  className="inline-flex items-center justify-
center w-10 h-10 rounded-xl border border-red-200 text-red-600 hover:bg-red-
500 hover:text-white transition-all duration-300"
                                  title="Видалити"
                                >
                                  <Trash2 size={17} />
                                </button>
                              </div>

                              <div className="md:hidden rounded-2xl border
border-green-100 bg-[#fcfffd] p-4">
                                <div className="flex items-start justify-
between gap-3">
                                  <div>
                                    <div className="text-gray-900 font-bold">
                                      {item.productName}
                                    </div>
                                    <div className="text-sm text-gray-600 mt-
1">
                                      Артикул: {item.productSku}
                                    </div>
                                    <div className="text-sm text-gray-700 mt-
1">
                                      Кількість: {item.quantity} {item.unit}
                                    </div>
                                  </div>
                                  <button
                                    type="button"
                                    onClick={() =>
                                      handleRemoveInvoiceItem(item.productId)
                                    }
                                    className="inline-flex items-center
justify-center w-10 h-10 rounded-xl border border-red-200 text-red-600
hover:bg-red-500 hover:text-white transition-all duration-300"
                                    title="Видалити"
                                  >
```

```
                                  <Trash2 size={17} />
                                </button>
                              </div>
                            </div>
                          </div>
                        ))}
                      </div>
                    </div>
                  )}
                </div>

                <div className="flex justify-end">
                  <button
                    type="button"
                    onClick={handleCreateShipment}
                    className="inline-flex items-center justify-center gap-2
rounded-xl border border-green-500 px-4 py-3 text-sm font-semibold text-
green-600 transition-all duration-300 hover:bg-green-500 hover:text-white"
                  >
                    <Plus size={16} />
                    Створити накладну
                  </button>
                </div>
              </div>

              <div className="rounded-[1.5rem] border border-green-100 bg-
[#fcfffd] p-5 sm:p-6 shadow-sm mb-6">
                <div className="relative">
                  <Search
                    className="absolute left-4 top-1/2 -translate-y-1/2
text-gray-400"
                    size={18}
                  />
                  <input
                    type="text"
                    value={searchTerm}
                    onChange={(e) => setSearchTerm(e.target.value)}
                    placeholder="Пошук по накладних"
                    className="w-full rounded-2xl border border-gray-200 bg-
white pl-11 pr-4 py-3 text-gray-800 placeholder:text-gray-400 focus:outline-
none focus:ring-2 focus:ring-green-500 focus:border-green-500 transition-all
duration-300"
                  />
                </div>
              </div>

              <div className="rounded-[1.5rem] border border-green-100 bg-
white shadow-sm overflow-hidden">
                {loading ? (
                  <div className="px-6 py-12 text-center text-gray-500">
                    Завантаження накладних...
                  </div>
                ) : filteredShipments.length === 0 ? (
                  <div className="px-6 py-14 text-center">
                    <h3 className="text-2xl font-bold text-gray-900 mb-3">
                      Накладних поки немає
                    </h3>
                    <p className="text-gray-600 max-w-2xl mx-auto leading-
relaxed">
                      Створіть першу накладну через форму вище, і вона
з’явиться у
                      списку нижче.
                    </p>
```

```
                  </div>
                ) : (
                  <>
                    <div className="hidden xl:grid grid-cols-
[70px_160px_1fr_160px_180px_180px] gap-4 bg-green-50/70 border-b border-
green-100 px-6 py-4 text-sm font-bold text-gray-800">
                      <div>№</div>
                      <div>Накладна</div>
                      <div>Одержувач</div>
                       <div>Номер телефону</div>
                      <div>Статус</div>
                      <div>Дата</div>
                      <div className="text-center">Дії</div>
                    </div>

                    <div className="divide-y divide-green-100">
                      {filteredShipments.map((shipment, index) => (
                        <div
                          key={shipment.id}
                          className={`px-4 sm:px-6 py-4 transition-all
duration-300 ${getShipmentRowClass(shipment.status)}`}
                        >
                          <div className="hidden xl:grid grid-cols-
[70px_160px_1fr_160px_180px_180px] gap-4 items-center">
                            <div className="text-gray-800 font-semibold">
                              {index + 1}
                            </div>

                            <div className="text-gray-900 font-bold">
                              {shipment.invoiceNumber}
                            </div>

                            <div>
                              <div className="text-gray-800 font-medium">
                                {shipment.customerName}
                              </div>
                              <div className="text-sm text-gray-500 mt-1">
                                {shipment.customerPhone}
                              </div>
                              {shipment.note && (
                                <div className="text-sm text-gray-500 mt-1">
                                  {shipment.note}
                                </div>
                              )}
                            </div>

                            <div>
                              <select
                                value={shipment.status}
                                onChange={(e) =>
                                  handleStatusChange(shipment,
e.target.value)
                                }
                                className="w-full rounded-xl border border-
gray-200 bg-white px-3 py-2 text-sm text-gray-800 focus:outline-none
focus:ring-2 focus:ring-green-500 focus:border-green-500 transition-all
duration-300"
                              >
                                {shipmentStatuses.map((status) => (
                                  <option key={status.value}
value={status.value}>
                                    {status.label}
                                  </option>
```

```
                                ))}
                              </select>
                            </div>

                            <div className="text-gray-700">
                              {formatDateTime(shipment.createdAt)}
                            </div>

                            <div className="flex items-center justify-center
gap-3">
                              <button
                                type="button"
                                onClick={() => handleViewItems(shipment)}
                                className="inline-flex items-center justify-
center w-10 h-10 rounded-xl border border-blue-200 text-blue-600 hover:bg-
blue-500 hover:text-white transition-all duration-300"
                                title="Переглянути"
                              >
                                <Eye size={17} />
                              </button>

                              <button
                                type="button"
                                onClick={() =>
handleDeleteShipment(shipment)}
                                className="inline-flex items-center justify-
center w-10 h-10 rounded-xl border border-red-200 text-red-600 hover:bg-red-
500 hover:text-white transition-all duration-300"
                                title="Видалити"
                              >
                                <Trash2 size={17} />
                              </button>
                            </div>
                          </div>

                          <div
                            className={`xl:hidden rounded-2xl border border-
green-100 p-4 ${getShipmentRowClass(shipment.status)}`}
                          >
                            <div className="flex items-start justify-between
gap-3 mb-3">
                              <div>
                                <div className="text-xs text-gray-500 mb-1">
                                  Накладна №{index + 1}
                                </div>
                                <div className="text-gray-900 font-bold
text-lg">
                                  {shipment.invoiceNumber}
                                </div>
                                <div className="text-sm text-gray-600 mt-1">
                                  {shipment.customerName}
                                </div>
                              </div>

                              <div className="flex items-center gap-2">
                                <button
                                  type="button"
                                  onClick={() => handleViewItems(shipment)}
                                  className="inline-flex items-center
justify-center w-10 h-10 rounded-xl border border-blue-200 text-blue-600
hover:bg-blue-500 hover:text-white transition-all duration-300"
                                  title="Переглянути"
                                >
```

```
                                    <Eye size={17} />
                                  </button>

                                  <button
                                    type="button"
                                    onClick={() =>
handleDeleteShipment(shipment)}
                                    className="inline-flex items-center
justify-center w-10 h-10 rounded-xl border border-red-200 text-red-600
hover:bg-red-500 hover:text-white transition-all duration-300"
                                    title="Видалити"
                                  >
                                    <Trash2 size={17} />
                                  </button>
                                </div>
                              </div>

                              <div className="space-y-3 text-sm text-gray-
700">
                                <div>
                                  <span className="font-semibold text-gray-
800">
                                    Статус:
                                  </span>
                                  <div className="mt-1">
                                    <select
                                      value={shipment.status}
                                      onChange={(e) =>
                                        handleStatusChange(shipment,
e.target.value)
                                      }
                                      className={`w-full rounded-xl px-3 py-2
text-sm focus:outline-none focus:ring-2 focus:ring-green-500 focus:border-
green-500 transition-all duration-300
${getShipmentStatusClass(shipment.status)}`}
                                    >
                                      {shipmentStatuses.map((status) => (
                                        <option
                                          key={status.value}
                                          value={status.value}
                                        >
                                          {status.label}
                                        </option>
                                      ))}
                                    </select>
                                  </div>
                                </div>

                                {shipment.note && (
                                  <div>
                                    <span className="font-semibold text-gray-
800">
                                      Примітка:
                                    </span>{" "}
                                    {shipment.note}
                                  </div>
                                )}

                                <div>
                                  <span className="font-semibold text-gray-
800">
                                    Дата:
                                  </span>{" "}
```

```
                                {formatDateTime(shipment.createdAt)}
                              </div>
                            </div>
                          </div>
                        </div>
                      ))}
                    </div>
                  </>
                )}
              </div>
            </div>
          </div>
        </main>
      );
    }
```

Сторінка аналітики, компонент Analytics.jsx:

```
import { useCallback, useEffect, useMemo, useState } from "react";
import {
  BarChart3,
  TrendingUp,
} from "lucide-react";
import { useAuth } from "../context/AuthContext";
import { db } from "../firebase";
import { collection, getDocs, query, where } from "firebase/firestore";
import {
  ResponsiveContainer,
  BarChart,
  Bar,
  XAxis,
  YAxis,
  CartesianGrid,
  Tooltip,
  PieChart,
  Pie,
  Cell,
  LineChart,
  Line,
  Legend,
} from "recharts";

export default function Analytics() {
  const { user } = useAuth();
  const [loading, setLoading] = useState(true);

  const [products, setProducts] = useState([]);
  const [categories, setCategories] = useState([]);
  const [suppliers, setSuppliers] = useState([]);
  const [shipments, setShipments] = useState([]);
  const [movements, setMovements] = useState([]);
  const [storageLocations, setStorageLocations] = useState([]);

  const fetchData = useCallback(async () => {
    if (!user) return;

    setLoading(true);

    try {
      const [
        productsSnapshot,
        categoriesSnapshot,
```

```
              suppliersSnapshot,
              shipmentsSnapshot,
              movementsSnapshot,
              storageSnapshot,
            ] = await Promise.all([
              getDocs(query(collection(db, "products"), where("userId", "==",
user.uid))),
              getDocs(query(collection(db, "categories"), where("userId",
"==", user.uid))),
              getDocs(query(collection(db, "suppliers"), where("userId", "==",
user.uid))),
              getDocs(query(collection(db, "shipments"), where("userId", "==",
user.uid))),
              getDocs(query(collection(db, "stock_movements"), where("userId",
"==", user.uid))),
              getDocs(query(collection(db, "storageLocations"),
where("userId", "==", user.uid))),
            ]);

            setProducts(productsSnapshot.docs.map((doc) => ({ id: doc.id,
...doc.data() })));
            setCategories(categoriesSnapshot.docs.map((doc) => ({ id: doc.id,
...doc.data() })));
            setSuppliers(suppliersSnapshot.docs.map((doc) => ({ id: doc.id,
...doc.data() })));
            setShipments(shipmentsSnapshot.docs.map((doc) => ({ id: doc.id,
...doc.data() })));
            setMovements(movementsSnapshot.docs.map((doc) => ({ id: doc.id,
...doc.data() })));
            setStorageLocations(storageSnapshot.docs.map((doc) => ({ id:
doc.id, ...doc.data() })));
          } finally {
            setLoading(false);
          }
        }, [user]);

        useEffect(() => {
          fetchData();
        }, [fetchData]);

        const formatDate = (timestamp) => {
          if (!timestamp?.toDate) return "";
          return timestamp.toDate().toLocaleDateString("uk-UA", {
            day: "2-digit",
            month: "2-digit",
          });
        };

        const kpis = useMemo(() => {
          const totalProducts = products.length;
          const totalCategories = categories.length;
          const totalSuppliers = suppliers.length;
          const totalShipments = shipments.length;

          const totalQuantity = products.reduce(
            (sum, item) => sum + (Number(item.quantity) || 0),
            0,
          );

          const totalPurchaseValue = products.reduce(
            (sum, item) =>
              sum + (Number(item.quantity) || 0) * (Number(item.purchasePrice)
|| 0),
```

```
            0,
          );

          const totalSaleValue = products.reduce(
            (sum, item) =>
              sum + (Number(item.quantity) || 0) * (Number(item.salePrice) ||
0),
            0,
          );

          const lowStockCount = products.filter(
            (item) => (Number(item.quantity) || 0) <= (Number(item.minStock)
|| 0),
          ).length;

          const activeShipments = shipments.filter(
            (item) => item.status === "new" || item.status === "confirmed",
          ).length;

          return {
            totalProducts,
            totalCategories,
            totalSuppliers,
            totalShipments,
            totalQuantity,
            totalPurchaseValue,
            totalSaleValue,
            lowStockCount,
            activeShipments,
          };
        }, [products, categories, suppliers, shipments]);

        const stockStatusData = useMemo(() => {
          const normal = products.filter(
            (item) => (Number(item.quantity) || 0) > (Number(item.minStock) ||
0),
          ).length;

          const low = products.filter(
            (item) => (Number(item.quantity) || 0) <= (Number(item.minStock)
|| 0),
          ).length;

          return [
            { name: "Норма", value: normal },
            { name: "Низький залишок", value: low },
          ];
        }, [products]);

        const shipmentStatusData = useMemo(() => {
          const map = {
            new: "Нове",
            confirmed: "Підтверджено",
            shipped: "Відвантажено",
            canceled: "Скасовано",
          };

          return Object.keys(map).map((key) => ({
            name: map[key],
            value: shipments.filter((item) => item.status === key).length,
          }));
        }, [shipments]);
```

```
        const categoryAnalytics = useMemo(() => {
          return categories
            .map((category) => {
              const categoryProducts = products.filter(
                (item) => item.categoryId === category.id,
              );

              const quantity = categoryProducts.reduce(
                (sum, item) => sum + (Number(item.quantity) || 0),
                0,
              );

              const value = categoryProducts.reduce(
                (sum, item) =>
                  sum + (Number(item.quantity) || 0) * (Number(item.salePrice)
|| 0),
                0,
              );

              return {
                name: category.name,
                productsCount: categoryProducts.length,
                quantity,
                value,
              };
            })
            .sort((a, b) => b.value - a.value);
        }, [categories, products]);

        const supplierAnalytics = useMemo(() => {
          return suppliers
            .map((supplier) => {
              const supplierProducts = products.filter(
                (item) => item.supplierId === supplier.id,
              );

              return {
                name: supplier.name,
                productsCount: supplierProducts.length,
                quantity: supplierProducts.reduce(
                  (sum, item) => sum + (Number(item.quantity) || 0),
                  0,
                ),
              };
            })
            .sort((a, b) => b.productsCount - a.productsCount)
            .slice(0, 8);
        }, [suppliers, products]);

        const movementChartData = useMemo(() => {
          const grouped = {};

          movements.forEach((movement) => {
            const key = formatDate(movement.createdAt) || "—";

            if (!grouped[key]) {
              grouped[key] = {
                date: key,
                incoming: 0,
                writeoff: 0,
                return: 0,
                transfer: 0,
                outgoing: 0,
```

```
            };
          }

          const qty = Number(movement.quantity) || 0;

          if (movement.type === "incoming") grouped[key].incoming += qty;
          if (movement.type === "writeoff") grouped[key].writeoff += qty;
          if (movement.type === "return") grouped[key].return += qty;
          if (movement.type === "transfer") grouped[key].transfer += qty;
          if (movement.type === "outgoing") grouped[key].outgoing += qty;
        });

        return Object.values(grouped).slice(-10);
      }, [movements]);

      const topDemandProducts = useMemo(() => {
        const grouped = {};

        movements
          .filter((item) => item.type === "outgoing")
          .forEach((movement) => {
            if (!grouped[movement.productId]) {
              grouped[movement.productId] = {
                name: movement.productName,
                qty: 0,
              };
            }

            grouped[movement.productId].qty += Number(movement.quantity) ||
0;
          });

        return Object.values(grouped)
          .sort((a, b) => b.qty - a.qty)
          .slice(0, 8)
          .map((item) => ({
            name: item.name,
            qty: item.qty,
          }));
      }, [movements]);

      const lowStockProducts = useMemo(() => {
        return products
          .filter((item) => (Number(item.quantity) || 0) <=
(Number(item.minStock) || 0))
          .map((item) => ({
            id: item.id,
            name: item.name,
            quantity: Number(item.quantity) || 0,
            minStock: Number(item.minStock) || 0,
            maxStock: Number(item.maxStock) || 0,
            unit: item.unit || "",
            recommendedOrder:
              Math.max(
                (Number(item.maxStock) || 0) - (Number(item.quantity) || 0),
                0,
              ) || 0,
          }))
          .sort((a, b) => a.quantity - b.quantity);
      }, [products]);

      const storageAnalytics = useMemo(() => {
```

```
          const allCells = storageLocations.flatMap((rack) => rack.cells ||
[]);
          const totalCells = allCells.length;
          const occupiedCells = allCells.filter((cell) => cell.status ===
"occupied").length;
          const freeCells = allCells.filter((cell) => cell.status ===
"free").length;

          return [
            { name: "Вільні", value: freeCells },
            { name: "Зайняті", value: occupiedCells },
            { name: "Усього", value: totalCells },
          ];
        }, [storageLocations]);

        const weeklyForecast = useMemo(() => {
          const outgoingMovements = movements.filter((item) => item.type ===
"outgoing");

          const perProduct = {};

          outgoingMovements.forEach((item) => {
            if (!perProduct[item.productId]) {
              perProduct[item.productId] = {
                productId: item.productId,
                name: item.productName,
                totalOutgoing: 0,
                count: 0,
              };
            }

            perProduct[item.productId].totalOutgoing += Number(item.quantity)
|| 0;
            perProduct[item.productId].count += 1;
          });

          return products
            .map((product) => {
              const stats = perProduct[product.id];
              const avgPerMovement = stats
                ? stats.totalOutgoing / Math.max(stats.count, 1)
                : 0;

              const forecast7Days = Math.round(avgPerMovement * 7);
              const currentQty = Number(product.quantity) || 0;
              const daysToStockout =
                avgPerMovement > 0
                  ? Math.floor(currentQty / avgPerMovement)
                  : null;

              return {
                id: product.id,
                name: product.name,
                currentQty,
                forecast7Days,
                daysToStockout,
                unit: product.unit || "",
              };
            })
            .sort((a, b) => (a.daysToStockout ?? 99999) - (b.daysToStockout ??
99999))
            .slice(0, 8);
        }, [movements, products]);
```

```
        const monthlySummary = useMemo(() => {
          const summary = {
            incoming: 0,
            outgoing: 0,
            writeoff: 0,
            return: 0,
            transfer: 0,
          };

          movements.forEach((item) => {
            const qty = Number(item.quantity) || 0;
            if (summary[item.type] !== undefined) {
              summary[item.type] += qty;
            }
          });

          return [
            { name: "Надходження", value: summary.incoming },
            { name: "Відвантаження", value: summary.outgoing },
            { name: "Списання", value: summary.writeoff },
            { name: "Повернення", value: summary.return },
            { name: "Переміщення", value: summary.transfer },
          ];
        }, [movements]);

        const pieColors = ["#22c55e", "#ef4444", "#3b82f6", "#f59e0b",
"#8b5cf6", "#14b8a6"];

        if (loading) {
          return (
            <main className="min-h-screen bg-[#f6faf7] px-4 py-10">
              <div className="max-w-7xl mx-auto">
                <div className="rounded-[2rem] border border-green-100 bg-
white shadow-xl shadow-green-100/40 p-10 text-center text-gray-500">
                  Завантаження аналітики...
                </div>
              </div>
            </main>
          );
        }

        return (
          <main className="min-h-screen bg-[#f6faf7] px-4 py-10">
            <div className="max-w-7xl mx-auto space-y-6">
              <div className="rounded-[2rem] border border-green-100 bg-white
shadow-xl shadow-green-100/40 p-6 sm:p-8 md:p-10">
                <div className="flex items-center gap-4 mb-6">
                  <div className="w-16 h-16 rounded-full bg-green-100 text-
green-600 flex items-center justify-center">
                    <BarChart3 size={30} />
                  </div>

                  <div>
                    <h1 className="text-3xl sm:text-4xl font-black text-gray-
900 mb-2">
                      Аналітика
                    </h1>
                    <p className="text-gray-600 text-base sm:text-lg leading-
relaxed">
                      Повна аналітика складу, руху товарів, статусів
відвантажень, залишків,
                      заповненості комірок і базовий прогноз попиту.
```

```
                  </p>
                </div>
              </div>

              <div className="grid grid-cols-1 sm:grid-cols-2 xl:grid-cols-4
gap-4">
                <div className="rounded-[1.5rem] border border-green-100 bg-
[#fcfffd] p-5 shadow-sm">
                  <div className="flex items-center justify-between mb-3">
                    <span className="text-xs text-gray-500">Товари</span>
                  </div>
                  <div className="text-3xl font-black text-gray-
900">{kpis.totalProducts}</div>
                  <p className="text-sm text-gray-600 mt-2">Загальна
кількість позицій у системі</p>
                </div>

                <div className="rounded-[1.5rem] border border-green-100 bg-
[#fcfffd] p-5 shadow-sm">
                  <div className="flex items-center justify-between mb-3">
                    <span className="text-xs text-gray-500">Запаси</span>
                  </div>
                  <div className="text-3xl font-black text-gray-
900">{kpis.totalQuantity}</div>
                  <p className="text-sm text-gray-600 mt-2">Сумарна
кількість одиниць товару</p>
                </div>

                <div className="rounded-[1.5rem] border border-green-100 bg-
[#fcfffd] p-5 shadow-sm">
                  <div className="flex items-center justify-between mb-3">
                    <span className="text-xs text-gray-500">Ризики</span>
                  </div>
                  <div className="text-3xl font-black text-gray-
900">{kpis.lowStockCount}</div>
                  <p className="text-sm text-gray-600 mt-2">Товарів з
низьким залишком</p>
                </div>

                <div className="rounded-[1.5rem] border border-green-100 bg-
[#fcfffd] p-5 shadow-sm">
                  <div className="flex items-center justify-between mb-3">
                    <span className="text-xs text-gray-500">Накладні</span>
                  </div>
                  <div className="text-3xl font-black text-gray-
900">{kpis.totalShipments}</div>
                  <p className="text-sm text-gray-600 mt-2">
                    Усього накладних, активних зараз: {kpis.activeShipments}
                  </p>
                </div>
              </div>
            </div>

            <div className="grid grid-cols-1 xl:grid-cols-3 gap-6">
              <div className="xl:col-span-2 rounded-[2rem] border border-
green-100 bg-white shadow-xl shadow-green-100/40 p-6">
                <h2 className="text-2xl  text-gray-900 mb-4">Активність руху
товарів</h2>
                <div className="h-[360px]">
                  <ResponsiveContainer width="100%" height="100%">
                    <BarChart data={movementChartData}>
                      <CartesianGrid strokeDasharray="3 3" />
                      <XAxis dataKey="date" />
```

```
                      <YAxis />
                      <Tooltip />
                      <Legend />
                      <Bar dataKey="incoming" name="Надходження"
fill="#22c55e" />
                      <Bar dataKey="outgoing" name="Відвантаження"
fill="#3b82f6" />
                      <Bar dataKey="writeoff" name="Списання" fill="#ef4444"
/>
                      <Bar dataKey="return" name="Повернення" fill="#f59e0b"
/>
                      <Bar dataKey="transfer" name="Переміщення"
fill="#8b5cf6" />
                    </BarChart>
                  </ResponsiveContainer>
                </div>
              </div>

              <div className="rounded-[2rem] border border-green-100 bg-
white shadow-xl shadow-green-100/40 p-6">
                <h2 className="text-2xl  text-gray-900 mb-4">Статуси
залишків</h2>
                <div className="h-[360px]">
                  <ResponsiveContainer width="100%" height="100%">
                    <PieChart>
                      <Pie
                        data={stockStatusData}
                        dataKey="value"
                        nameKey="name"
                        outerRadius={110}
                        label
                      >
                        {stockStatusData.map((entry, index) => (
                          <Cell key={`cell-${index}`} fill={pieColors[index
% pieColors.length]} />
                        ))}
                      </Pie>
                      <Tooltip />
                      <Legend />
                    </PieChart>
                  </ResponsiveContainer>
                </div>
              </div>

              <div className="grid grid-cols-1 xl:grid-cols-2 gap-6">
                <div className="rounded-[2rem] border border-green-100 bg-
white shadow-xl shadow-green-100/40 p-6">
                <h2 className="text-2xl  text-gray-900 mb-4">Статуси
накладних</h2>
                <div className="h-[320px]">
                  <ResponsiveContainer width="100%" height="100%">
                    <PieChart>
                      <Pie
                        data={shipmentStatusData}
                        dataKey="value"
                        nameKey="name"
                        outerRadius={100}
                        label
                      >
                        {shipmentStatusData.map((entry, index) => (
                          <Cell key={`shipment-${index}`}
fill={pieColors[index % pieColors.length]} />
```

```
                        ))}
                      </Pie>
                      <Tooltip />
                      <Legend />
                    </PieChart>
                  </ResponsiveContainer>
                </div>
              </div>

              <div className="rounded-[2rem] border border-green-100 bg-
white shadow-xl shadow-green-100/40 p-6">
                <h2 className="text-2xl  text-gray-900 mb-4">Заповненість
місць зберігання</h2>
                <div className="h-[320px]">
                  <ResponsiveContainer width="100%" height="100%">
                    <PieChart>
                      <Pie
                        data={storageAnalytics.filter((item) => item.name
!== "Усього")}
                        dataKey="value"
                        nameKey="name"
                        outerRadius={100}
                        label
                      >
                        <Cell fill="#22c55e" />
                        <Cell fill="#ef4444" />
                      </Pie>
                      <Tooltip />
                      <Legend />
                    </PieChart>
                  </ResponsiveContainer>
                </div>
                <div className="mt-4 text-sm text-gray-600">
                  Усього комірок:{" "}
                  <span className="font-semibold text-gray-900">
                    {storageAnalytics.find((item) => item.name ===
"Усього")?.value || 0}
                  </span>
                </div>
              </div>
            </div>

            <div className="grid grid-cols-1 xl:grid-cols-2 gap-6">
              <div className="rounded-[2rem] border border-green-100 bg-
white shadow-xl shadow-green-100/40 p-6">
                <h2 className="text-2xl  text-gray-900 mb-4">Топ категорій
за вартістю</h2>
                <div className="h-[360px]">
                  <ResponsiveContainer width="100%" height="100%">
                    <BarChart data={categoryAnalytics.slice(0, 8)}
layout="vertical">
                      <CartesianGrid strokeDasharray="3 3" />
                      <XAxis type="number" />
                      <YAxis dataKey="name" type="category" width={120} />
                      <Tooltip />
                      <Bar dataKey="value" name="Оціночна вартість"
fill="#22c55e" />
                    </BarChart>
                  </ResponsiveContainer>
                </div>
              </div>
```

```
                <div className="rounded-[2rem] border border-green-100 bg-
white shadow-xl shadow-green-100/40 p-6">
                  <h2 className="text-2xl  text-gray-900 mb-4">Топ товарів за
попитом</h2>
                  <div className="h-[360px]">
                    <ResponsiveContainer width="100%" height="100%">
                      <BarChart data={topDemandProducts} layout="vertical">
                        <CartesianGrid strokeDasharray="3 3" />
                        <XAxis type="number" />
                        <YAxis dataKey="name" type="category" width={140} />
                        <Tooltip />
                        <Bar dataKey="qty" name="Відвантажено" fill="#3b82f6"
/>
                      </BarChart>
                    </ResponsiveContainer>
                  </div>
                </div>
              </div>

              <div className="grid grid-cols-1 xl:grid-cols-2 gap-6">
                <div className="rounded-[2rem] border border-green-100 bg-
white shadow-xl shadow-green-100/40 p-6">
                  <h2 className="text-2xl  text-gray-900 mb-4">Підсумок по
руху товарів</h2>
                  <div className="h-[320px]">
                    <ResponsiveContainer width="100%" height="100%">
                      <BarChart data={monthlySummary}>
                        <CartesianGrid strokeDasharray="3 3" />
                        <XAxis dataKey="name" />
                        <YAxis />
                        <Tooltip />
                        <Bar dataKey="value" name="Кількість" fill="#16a34a"
/>
                      </BarChart>
                    </ResponsiveContainer>
                  </div>
                </div>

                <div className="rounded-[2rem] border border-green-100 bg-
white shadow-xl shadow-green-100/40 p-6">
                  <h2 className="text-2xl  text-gray-900 mb-
4">Постачальники</h2>
                  <div className="h-[320px]">
                    <ResponsiveContainer width="100%" height="100%">
                      <BarChart data={supplierAnalytics}>
                        <CartesianGrid strokeDasharray="3 3" />
                        <XAxis dataKey="name" hide />
                        <YAxis />
                        <Tooltip />
                        <Bar dataKey="productsCount" name="К-сть товарів"
fill="#14b8a6" />
                      </BarChart>
                    </ResponsiveContainer>
                  </div>
                </div>
              </div>

              <div className="rounded-[2rem] border border-green-100 bg-white
shadow-xl shadow-green-100/40 p-6">
                <div className="flex items-center gap-3 mb-4">
                  <TrendingUp className="text-green-600" size={24} />
                  <h2 className="text-2xl  text-gray-900">Прогноз і
рекомендації</h2>
```

```
                </div>

                <div className="h-[340px] mb-6">
                  <ResponsiveContainer width="100%" height="100%">
                    <LineChart data={weeklyForecast}>
                      <CartesianGrid strokeDasharray="3 3" />
                      <XAxis dataKey="name" hide />
                      <YAxis />
                      <Tooltip />
                      <Legend />
                      <Line
                        type="monotone"
                        dataKey="currentQty"
                        name="Поточний залишок"
                        stroke="#22c55e"
                        strokeWidth={2}
                      />
                      <Line
                        type="monotone"
                        dataKey="forecast7Days"
                        name="Прогноз попиту на 7 днів"
                        stroke="#3b82f6"
                        strokeWidth={2}
                      />
                    </LineChart>
                  </ResponsiveContainer>
                </div>

                <div className="overflow-x-auto rounded-2xl border border-
green-100">
                  <table className="w-full text-sm">
                    <thead className="bg-green-50/70">
                      <tr className="text-left text-gray-800">
                        <th className="px-4 py-3 font-bold">Товар</th>
                        <th className="px-4 py-3 font-bold">Залишок</th>
                        <th className="px-4 py-3 font-bold">Прогноз 7
днів</th>
                        <th className="px-4 py-3 font-bold">Днів до
вичерпання</th>
                      </tr>
                    </thead>
                    <tbody className="divide-y divide-green-100">
                      {weeklyForecast.map((item) => (
                        <tr key={item.id} className="hover:bg-green-50/40">
                          <td className="px-4 py-3 text-gray-900 font-
medium">{item.name}</td>
                          <td className="px-4 py-3 text-gray-700">
                            {item.currentQty} {item.unit}
                          </td>
                          <td className="px-4 py-3 text-gray-700">
                            {item.forecast7Days} {item.unit}
                          </td>
                          <td className="px-4 py-3 text-gray-700">
                            {item.daysToStockout === null ? "Немає даних" :
item.daysToStockout}
                          </td>
                        </tr>
                      ))}
                    </tbody>
                  </table>
                </div>

                <p className="text-sm text-gray-500 mt-4 leading-relaxed">
```

```
                  Прогноз базується на середній інтенсивності минулих
відвантажень.
                  Це спрощена оцінка, яка допомагає побачити ризики дефіциту
та пріоритети поповнення запасів.
                </p>
              </div>

              <div className="rounded-[2rem] border border-red-100 bg-white
shadow-xl shadow-red-100/30 p-6">
                <h2 className="text-2xl  text-gray-900 mb-4">Товари з
критичним залишком</h2>

                {lowStockProducts.length === 0 ? (
                  <div className="text-gray-600">
                    Усі товари мають залишок вище за мінімальний рівень.
                  </div>
                ) : (
                  <div className="overflow-x-auto rounded-2xl border border-
red-100">
                    <table className="w-full text-sm">
                      <thead className="bg-red-50/70">
                        <tr className="text-left text-gray-800">
                          <th className="px-4 py-3 font-bold">Товар</th>
                          <th className="px-4 py-3 font-bold">Поточний
залишок</th>
                          <th className="px-4 py-3 font-bold">Мінімум</th>
                          <th className="px-4 py-3 font-bold">Бажаний
запас</th>
                          <th className="px-4 py-3 font-bold">Рекомендовано
замовити</th>
                        </tr>
                      </thead>
                      <tbody className="divide-y divide-red-100">
                        {lowStockProducts.map((item) => (
                          <tr key={item.id} className="hover:bg-red-50/40">
                            <td className="px-4 py-3 text-gray-900 font-
medium">{item.name}</td>
                            <td className="px-4 py-3 text-red-700 font-
semibold">
                              {item.quantity} {item.unit}
                            </td>
                            <td className="px-4 py-3 text-gray-700">
                              {item.minStock} {item.unit}
                            </td>
                            <td className="px-4 py-3 text-gray-700">
                              {item.maxStock} {item.unit}
                            </td>
                            <td className="px-4 py-3 text-gray-900 font-
semibold">
                              {item.recommendedOrder} {item.unit}
                            </td>
                          </tr>
                        ))}
                      </tbody>
                    </table>
                  </div>
                )}
              </div>

              <div className="grid grid-cols-1 md:grid-cols-3 gap-6">
                <div className="rounded-[2rem] border border-green-100 bg-
white shadow-xl shadow-green-100/40 p-6">
```

```
                  <div className="text-sm text-gray-500 mb-2">Закупівельна
вартість залишків</div>
                  <div className="text-3xl  text-gray-900">
                    {kpis.totalPurchaseValue.toLocaleString("uk-UA")}
                  </div>
                </div>

                <div className="rounded-[2rem] border border-green-100 bg-
white shadow-xl shadow-green-100/40 p-6">
                  <div className="text-sm text-gray-500 mb-2">Оціночна
вартість продажу</div>
                  <div className="text-3xl  text-gray-900">
                    {kpis.totalSaleValue.toLocaleString("uk-UA")}
                  </div>
                </div>

                <div className="rounded-[2rem] border border-green-100 bg-
white shadow-xl shadow-green-100/40 p-6">
                  <div className="text-sm text-gray-500 mb-2">Категорій /
Постачальників</div>
                  <div className="text-3xl  text-gray-900">
                    {kpis.totalCategories} / {kpis.totalSuppliers}
                  </div>
                </div>
              </div>
            </div>
          </main>
        );
      }
```

Сторінка профілю користувача, компонент Profile.jsx:

```
import { useEffect, useState } from "react";
import { User, Mail, Lock } from "lucide-react";
import { useNavigate } from "react-router-dom";
import { useAuth } from "../context/AuthContext";
import { db } from "../firebase";
import {
  updateProfile,
  updatePassword,
  EmailAuthProvider,
  reauthenticateWithCredential,
  deleteUser,
} from "firebase/auth";
import {
  doc,
  updateDoc,
  collection,
  getDocs,
  query,
  where,
  writeBatch,
} from "firebase/firestore";
import Swal from "sweetalert2";

export default function Profile() {
  const { user } = useAuth();
  const navigate = useNavigate();

  const [nameForm, setNameForm] = useState({
    fullName: "",
  });
```

```
  const [passwordForm, setPasswordForm] = useState({
    currentPassword: "",
    newPassword: "",
  });

  useEffect(() => {
    if (user) {
      setNameForm({
        fullName: user.displayName || "",
      });
    }
  }, [user]);

  const toast = (icon, title) => {
    Swal.fire({
      toast: true,
      position: "top-end",
      icon,
      title,
      showConfirmButton: false,
      timer: 3500,
      timerProgressBar: true,
    });
  };

  const handleNameChange = (e) => {
    setNameForm({ ...nameForm, [e.target.name]: e.target.value });
  };

  const handlePasswordChange = (e) => {
    setPasswordForm({ ...passwordForm, [e.target.name]: e.target.value
});
  };

  const saveName = async (e) => {
    e.preventDefault();

    if (!user) {
      toast("error", "Користувача не знайдено");
      return;
    }

    const fullName = nameForm.fullName.trim();

    if (!fullName) {
      toast("error", "Введіть нове ім’я");
      return;
    }

    try {
      await updateProfile(user, { displayName: fullName });
      await updateDoc(doc(db, "users", user.uid), { fullName });

      toast("success", "Ім’я успішно змінено");
    } catch (error) {
      toast("error", "Не вдалося змінити ім’я");
    }
  };

  const savePassword = async (e) => {
    e.preventDefault();
```

```
          if (!user || !user.email) {
            toast("error", "Користувача не знайдено");
            return;
          }

          const currentPassword = passwordForm.currentPassword.trim();
          const newPassword = passwordForm.newPassword.trim();

          if (!currentPassword || !newPassword) {
            toast("error", "Заповніть усі поля пароля");
            return;
          }

          if (newPassword.length < 6) {
            toast("error", "Новий пароль має містити щонайменше 6 символів");
            return;
          }

          try {
            const credential = EmailAuthProvider.credential(
              user.email,
              currentPassword,
            );
            await reauthenticateWithCredential(user, credential);
            await updatePassword(user, newPassword);

            setPasswordForm({
              currentPassword: "",
              newPassword: "",
            });

            toast("success", "Пароль успішно змінено");
          } catch (error) {
            toast("error", "Старий пароль невірний або сталася помилка");
          }
        };

        const deleteRelatedDocs = async (uid) => {
          const batch = writeBatch(db);

          const collectionsToCheck = [
            { name: "users", fields: [] },
            { name: "products", fields: ["userId", "ownerId", "createdBy"] },
            { name: "categories", fields: ["userId", "ownerId", "createdBy"]
},
            { name: "suppliers", fields: ["userId", "ownerId", "createdBy"] },
            { name: "storageLocations", fields: ["userId", "ownerId",
"createdBy"] },
            {
              name: "inventoryMovements",
              fields: ["userId", "ownerId", "createdBy"],
            },
            { name: "shipments", fields: ["userId", "ownerId", "createdBy"] },
            { name: "analytics", fields: ["userId", "ownerId", "createdBy"] },
          ];

          batch.delete(doc(db, "users", uid));

          for (const item of collectionsToCheck) {
            for (const field of item.fields) {
              const q = query(collection(db, item.name), where(field, "==",
uid));
              const snapshot = await getDocs(q);
```

```
            snapshot.forEach((document) => {
              batch.delete(document.ref);
            });
          }
        }

        await batch.commit();
      };

      const handleDeleteProfile = async () => {
        if (!user) {
          toast("error", "Користувача не знайдено");
          return;
        }

        const result = await Swal.fire({
          title: "Ви впевнені?",
          text: "Профіль і пов’язані дані буде видалено без можливості
відновлення.",
          icon: "warning",
          showCancelButton: true,
          confirmButtonColor: "#d1d5db",
          cancelButtonColor: "#22c55e",
          confirmButtonText: "Так, видалити",
          cancelButtonText: "Скасувати",
          reverseButtons: true,
        });

        if (!result.isConfirmed) return;

        try {
          await deleteRelatedDocs(user.uid);
          await deleteUser(user);

          await Swal.fire({
            icon: "success",
            title: "Профіль видалено",
            text: "Ваш обліковий запис успішно видалено.",
            confirmButtonColor: "#22c55e",
          });

          navigate("/");
        } catch (error) {
          if (error.code === "auth/requires-recent-login") {
            toast("error", "Для видалення профілю потрібно знову увійти в
систему");
          } else {
            toast("error", "Не вдалося видалити профіль");
          }
        }
      };

      return (
        <main className="min-h-screen bg-[#eef6f0] px-4 py-8 sm:py-10">
      <div className="max-w-5xl mx-auto">
        <div className="rounded-[2rem] border border-green-200 bg-white
shadow-xl shadow-green-200/50 p-6 sm:p-8 md:p-12">
          <div className="flex flex-col items-start sm:flex-row sm:items-
center sm:justify-between gap-5 mb-8">
            <div>
              <h1 className="text-3xl sm:text-4xl font-black text-gray-950
mb-2">
```

```
                Профіль користувача
              </h1>

              <p className="text-gray-700 text-base sm:text-lg leading-
relaxed">
                Керуйте основною інформацією профілю, змінюйте ім’я та
пароль
                вашого облікового запису.
              </p>
            </div>
          </div>

          <div className="grid grid-cols-1 lg:grid-cols-2 gap-6">
            <form
              onSubmit={saveName}
              className="rounded-[1.5rem] border border-green-200 bg-white
p-5 sm:p-6 shadow-md"
            >
              <div className="flex items-center gap-3 mb-5">
                <div className="w-11 h-11 rounded-full bg-green-200 text-
green-700 flex items-center justify-center">
                  <User size={20} />
                </div>
                <h2 className="text-xl font-bold text-gray-900">Зміна
імені</h2>
              </div>

              <div className="mb-4">
                <label className="block text-sm font-semibold text-gray-800
mb-2">
                  Нове ім’я
                </label>
                <input
                  type="text"
                  name="fullName"
                  value={nameForm.fullName}
                  onChange={handleNameChange}
                  placeholder="Введіть нове ім’я"
                  className="w-full rounded-2xl border border-gray-300 bg-
white px-4 py-3 text-gray-900 placeholder:text-gray-500 focus:outline-none
focus:ring-2 focus:ring-green-600 focus:border-green-600 transition-all
duration-300"
                />
              </div>

              <div className="mb-6">
                <label className="block text-sm font-semibold text-gray-800
mb-2">
                  Електронна пошта
                </label>
                <div className="relative">
                  <Mail
                    className="absolute left-4 top-1/2 -translate-y-1/2
text-gray-500"
                    size={18}
                  />
                  <input
                    type="email"
                    value={user?.email || ""}
                    readOnly
                    className="w-full rounded-2xl border border-gray-300 bg-
gray-100 pl-11 pr-4 py-3 text-gray-700 cursor-not-allowed"
                  />
```

```
                </div>
              </div>

              <div className="flex justify-center">
                <button
                  type="submit"
                  className="inline-flex items-center justify-center
rounded-xl border border-green-600 px-4 py-2.5 text-sm font-semibold text-
green-700 transition-all duration-300 hover:bg-green-600 hover:text-white"
                >
                  Зберегти зміни
                </button>
              </div>
            </form>

            <form
              onSubmit={savePassword}
              className="rounded-[1.5rem] border border-green-200 bg-white
p-5 sm:p-6 shadow-md"
            >
              <div className="flex items-center gap-3 mb-5">
                <div className="w-11 h-11 rounded-full bg-green-200 text-
green-700 flex items-center justify-center">
                  <Lock size={20} />
                </div>
                <h2 className="text-xl font-bold text-gray-900">
                  Зміна пароля
                </h2>
              </div>

              <div className="mb-4">
                <label className="block text-sm font-semibold text-gray-800
mb-2">
                  Старий пароль
                </label>
                <input
                  type="password"
                  name="currentPassword"
                  value={passwordForm.currentPassword}
                  onChange={handlePasswordChange}
                  placeholder="Введіть старий пароль"
                  className="w-full rounded-2xl border border-gray-300 bg-
white px-4 py-3 text-gray-900 placeholder:text-gray-500 focus:outline-none
focus:ring-2 focus:ring-green-600 focus:border-green-600 transition-all
duration-300"
                />
              </div>

              <div className="mb-6">
                <label className="block text-sm font-semibold text-gray-800
mb-2">
                  Новий пароль
                </label>
                <input
                  type="password"
                  name="newPassword"
                  value={passwordForm.newPassword}
                  onChange={handlePasswordChange}
                  placeholder="Введіть новий пароль"
                  className="w-full rounded-2xl border border-gray-300 bg-
white px-4 py-3 text-gray-900 placeholder:text-gray-500 focus:outline-none
focus:ring-2 focus:ring-green-600 focus:border-green-600 transition-all
duration-300"
```

```
                />
              </div>

              <div className="flex justify-center">
                <button
                  type="submit"
                  className="inline-flex items-center justify-center
rounded-xl border border-green-600 px-4 py-2.5 text-sm font-semibold text-
green-700 transition-all duration-300 hover:bg-green-600 hover:text-white"
                >
                  Оновити пароль
                </button>
              </div>
            </form>
          </div>

          <div className="mt-8 rounded-[1.5rem] border border-red-200 bg-
red-100/80 p-5 sm:p-6">
            <h3 className="text-xl font-bold text-gray-950 mb-2">
              Видалення профілю
            </h3>

            <p className="text-gray-700 leading-relaxed mb-5">
              Ця дія повністю видалить ваш обліковий запис та пов’язані з
ним
              дані із системи. Після підтвердження відновити профіль буде
              неможливо.
            </p>

            <div className="flex justify-center">
              <button
                onClick={handleDeleteProfile}
                className="inline-flex items-center justify-center rounded-
xl border border-red-600 px-4 py-2.5 text-sm font-semibold text-red-700
transition-all duration-300 hover:bg-red-600 hover:text-white"
              >
                Видалити профіль
              </button>
            </div>
          </div>
        </div>
      </div>
    </main>
  );
}
```