ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
МІСЬКОГО ГОСПОДАРСТВА ІМЕНІ О. М. БЕКЕТОВА
Навчально-науковий інститут енергетичної, інформаційної
та транспортної інфраструктури

Кафедра комп'ютерних наук та інформаційних технологій

Пояснювальна записка
до кваліфікаційної роботи бакалавра

на тему: «Розробка Telegram-бота для обробки сервісів поштової доставки»

Виконав: студент 4 курсу, групи КН 2021-1
спеціальності
122 Комп'ютерні науки
(шифр і назва спеціальності)

Дмитро НАУМЕНКО
(ім'я та прізвище)

Керівник: д.ф-м.н., проф. Наталія СІЗОВА
(ім'я та прізвище)

Рецензент: к.т.н., доц. Пахомов Ю.В.
(прізвище та ініціали)

м. Харків – 2025 рік

Харківський національний університет міського господарства імені О. М. Бекетова
(повне найменування закладу вищої освіти)
Навчально-науковий Інститут енергетичної, інформаційної та транспортної інфраструктури
Кафедра комп'ютерних наук та інформаційних технологій
Рівень вищої освіти перший (бакалаврський)
Спеціальність 122 Комп'ютерні науки
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри КНтаІТ

Марина НОВОЖИЛОВА
«_24___» _____06_____2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Науменко Дмитро Геннадійович
(прізвище, ім'я, по батькові)

1. Тема роботи «Розробка Telegram-бота для обробки сервісів поштової доставки»
керівник роботи д.ф-м.н., проф. Сізова Н.Д.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від « 09 » травня 2025 р. № 341-03.
2. Термін подання студентом роботи 24.06.2025
3. Вихідні дані до роботи Рекомендації для розробки Telegram-бота для обробки сервісів поштової доставки.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) проаналізувати предметну область та наявні аналоги; надати аргументоване обґрунтування вибору комплексу інструментального середовища та технічної платформи; визначити ключовий функціонал та провести його системний аналіз; розробити програмний код основних модулів та інтерфейс взаємодії з користувачем; представити опис архітектурної моделі та графічного інтерфейсу користувача Telegram-бота для обробки сервісів поштової доставки та здійснити комплексне тестування програмного забезпечення.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень):
15 слайдів

6. Консультанти розділів роботи

Розділ	Ім'я та Прізвище, посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ I	Наталія СІЗОВА	11.05.2025	10.05.2025
Розділ II	Наталія СІЗОВА	17.05.2025	15.05.2025
Розділ III	Наталія СІЗОВА	21.05.2025	30.05.2025
Розділ IV	Вікторія Малишева	27.05.2025	15.06.2025

7. Дата видачі завдання 10.05.2025

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Вибір теми дипломної роботи	03.05.2025	Викон.
2	Затвердження тем, наукових керівників, завдань та календарного плану підготовки дипломної роботи	05.05.2025	Викон.
3	Написання I розділу	10.05.2025	Викон.
4	Написання II розділу	15.05.2025	Викон.
5	Написання III розділу	20.05.2025	Викон.
6	Написання IV розділу	30.05.2025	Викон.
7	Подання дипломної роботи керівнику	05.06.2025	Викон.
8	Робота по усуненню зауважень керівника, уточнення і доповнення практичного матеріалу, оформлення додатків до роботи	10.06.2025	Викон.
9	Подання доопрацьованого варіанту роботи керівнику	15.06.2025	Викон.
10	Захист матеріалів дипломної роботи на засіданні кафедри	18.06.2025	Викон.
11	Офіційний захист матеріалів дипломної роботи на засіданні екзаменаційної комісії	22.06.2025	Викон.

Студент _____ (підпис) Дмитро НАУМЕНКО (прізвище та ініціали)

Керівник роботи _____ (підпис) Наталія СІЗОВА (прізвище та ініціали)

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка кваліфікаційної роботи бакалавра здобувача вищої освіти групи КН 2021-1 спеціальності 122 «Комп’ютерні науки» Науменко Дмитра Геннадійовича за темою «Розробка Telegram-бота для обробки сервісів поштової доставки» складається з 4 розділів, містить 77 сторінок тексту, 7 рисунків, 1 таблиць, 41 джерело.

Дана робота присвячена розробці Telegram-бота для обробки сервісів поштової доставки задля зручного доступу користувача до сервісів.

Актуальність дослідження. Послуги поштової доставки є невід'ємною частиною сучасного життя, однак взаємодія з численними операторами через окремі додатки та сайти є незручною для користувачів.

Месенджер Telegram, завдяки своїй популярності, надає ідеальну платформу для створення уніфікованих рішень. На ринку бракує ефективного Telegram-бота, який би об'єднував сервіси кількох ключових українських поштових операторів.

Метою роботи є розробка Telegram-бота, що забезпечить уніфікований та зручний доступ до основних сервісів провідних поштових операторів України.

Завдання полягає в проектуванні архітектури Telegram-бота та виборі відповідного технологічного стеку, подальшій розробці механізмів взаємодії з програмними інтерфейсами (API) ключових поштових операторів, реалізації основного користувацького функціоналу, а саме: відстеження відправлень, розрахунку вартості доставки та пошуку відділень, створенні інтуїтивно зрозумілого інтерфейсу в середовищі Telegram і, на завершення, в проведенні ретельного тестування розробленого програмного продукту та підготовці відповідної технічної документації.

Об'єктом дослідження є процес розробки програмного продукту – Telegram-бота для взаємодії з сервісами поштової доставки.

Предметом дослідження є методи, технології та інструментальні засоби розробки Telegram-бота, інтегрованого з програмними інтерфейсами (API) поштових операторів, для надання уніфікованого доступу до їхніх послуг.

Методами дослідження є аналіз предметної області та існуючих аналогічних рішень, методи проектування програмного забезпечення, що включають розробку архітектури та моделювання даних, а також програмна реалізація та тестування розробленого Telegram-бота.

Наукова новизна одержаних результатів полягає у розробці архітектурної моделі Telegram-бота, що забезпечує уніфікований доступ до сервісів кількох поштових операторів України через єдиний програмний інтерфейс, що, на відміну від існуючих рішень, вирішує проблему фрагментації користувацького досвіду.

Практична значимість роботи полягає у створенні готового до впровадження програмного продукту – Telegram-бота, який дозволяє користувачам значно спростити та прискорити взаємодію з сервісами кількох поштових операторів через єдиний, зручний інтерфейс.

У першому розділі виконано огляд середовища розробки, проведено аналіз наявних аналогів, на основі якого обґрунтовано переваги запропонованого рішення, та сформульовано постановку задачі.

Другий розділ присвячено теоретичному обґрунтуванню розробки Telegram-бота. Проведено аналіз предметної області сервісів поштової доставки, на основі якого здійснено проектування системи – визначено архітектуру та моделі даних бота. Також описано ключові алгоритми та математичні підходи, що забезпечують його функціонування.

Третій розділ розкриває практичну реалізацію Telegram-бота. Обґрунтовано вибір засобів розробки та сформульовано технічні й програмні вимоги. Детально описано програмну реалізацію ключових модулів та представлено керівництво для користувача, що пояснює взаємодію з ботом.

У четвертому розділі розглядаються питання охорони праці.

Ключові слова : Telegram-бот, API, поштові сервіси, розробка програмного забезпечення, інтеграція сервісів, уніфікований доступ, Telegram Bot API.

ANNOTATION

Structure and scope of work. Explanatory note to the bachelor's thesis of a higher education applicant from group KN 2021-1, speciality 122 ‘Computer Science’ by Dmytro Hennadiyovych Naumenko on the topic ‘Development of a Telegram bot for processing postal delivery services’ consists of 4 sections and contains – 77 pages of text, 7 pictures, 1 table, 32 sources.

This work is devoted to the development of a Telegram bot for processing postal delivery services for convenient user access to services.

Relevance of the research. Postal delivery services are an integral part of modern life, but interacting with multiple operators through separate applications and websites is inconvenient for users.

Thanks to its popularity, the Telegram messenger provides an ideal platform for creating unified solutions. The market lacks an effective Telegram bot that would combine the services of several key Ukrainian postal operators.

The aim of this work is to develop a Telegram bot that will provide unified and convenient access to the main services of the leading postal operators in Ukraine.

The task is to design the architecture of the Telegram bot and select the appropriate technology stack, further develop mechanisms for interacting with the application programming interfaces (APIs) of key postal operators, implement the main user functionality, namely: tracking shipments, calculating delivery costs and searching for branches, creating an intuitive interface in the Telegram environment and, finally, conducting thorough testing of the developed software product and preparing the relevant technical documentation.

The object of the study is the process of developing a software product – a Telegram bot for interacting with postal delivery services.

The subject of the study is the methods, technologies and tools for developing a Telegram bot integrated with the application programming interfaces (APIs) of postal operators to provide unified access to their services.

The research methods include analysis of the subject area and existing similar solutions, software design methods, including architecture development and data modelling, as well as software implementation and testing of the developed Telegram bot.

The scientific novelty of the results obtained lies in the development of an architectural model of a Telegram bot that provides unified access to the services of several postal operators in Ukraine through a single software interface, which, unlike existing solutions, solves the problem of fragmentation of the user experience.

The practical significance of the work lies in the creation of a ready-to-implement software product – a Telegram bot that allows users to significantly simplify and speed up interaction with the services of several postal operators through a single, convenient interface.

The first chapter provides an overview of the development environment, analyses existing analogues, on the basis of which the advantages of the proposed solution are justified, and formulates the task.

The second section is devoted to the theoretical justification for the development of the Telegram bot. An analysis of the subject area of postal delivery services is carried out, on the basis of which the system is designed – the architecture and data models of the bot are determined. The key algorithms and mathematical approaches that ensure its functioning are also described.

The third section reveals the practical implementation of the Telegram bot. The choice of development tools is justified and technical and software requirements are formulated. The software implementation of key modules is described in detail and a user guide explaining how to interact with the bot is provided.

The fourth section deals with occupational safety issues.

Keywords : Telegram bot, API, mail services, software development, service integration, unified access, Telegram Bot API.

ЗМІСТ

ВСТУП

В епоху стрімкого розвитку цифрових технологій та електронної комерції, послуги поштової доставки відіграють ключову роль у повсякденному житті мільйонів людей та функціонуванні бізнесу. Зростання обсягів онлайн-торгівлі та глобалізація ринків зумовлюють підвищений попит на швидкі, надійні та зручні способи отримання інформації про відправлення та керування ними.

Сучасні користувачі прагнуть мати оперативний доступ до сервісів різних поштових операторів, не витрачаючи час на перемикання між численними веб-сайтами та мобільними додатками. Ця потреба в уніфікації та спрощенні взаємодії з поштовими сервісами визначає актуальність теми даної дипломної роботи.

Популярність месенджерів, зокрема Telegram, як повсякденного інструменту комунікації, відкриває нові можливості для створення інтуїтивно зрозумілих та доступних рішень. Розробка Telegram-бота, що агрегує функціонал кількох поштових операторів, дозволяє значно підвищити зручність для кінцевого користувача, надаючи всю необхідну інформацію в єдиному інтерфейсі. На сьогоднішній день, хоча існують офіційні боти та додатки від окремих операторів, бракує універсального інструменту, який би ефективно об'єднував їхні ключові послуги.

Метою дипломної роботи є розробка Telegram-бота, призначеного для забезпечення зручного доступу користувачів до основних сервісів провідних поштових операторів України, таких як відстеження відправлень, розрахунок орієнтовної вартості доставки та пошук найближчих відділень.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

– Проаналізувати предметну область поштових сервісів та дослідити існуючі аналоги програмних рішень.

- Спроектувати архітектуру Telegram-бота, визначити його основні модулі та моделі даних.
- Розробити механізми взаємодії з API поштових операторів для отримання актуальної інформації.
- Реалізувати основний користувацький функціонал бота, включаючи відстеження відправлень, розрахунок вартості та пошук відділень.
- Створити інтуїтивно зрозумілий користувацький інтерфейс в середовищі Telegram.
- Провести тестування розробленого програмного продукту та підготувати необхідну документацію.

РОЗДІЛ 1
ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Опис предметного середовища

Предметне середовище цієї дипломної роботи охоплює комплекс аспектів, пов'язаних із проектуванням, розробкою, тестуванням та розгортанням програмного продукту – Telegram-бота. Головним завданням цього бота є спрощення взаємодії користувачів із сервісами поштових операторів.

Центральним елементом даного середовища виступає власне Telegram-бот, що розробляється. Його створення нерозривно пов'язане з використанням ключових технологічних ресурсів. Перш за все, це стосується платформи Telegram [38] та її програмного інтерфейсу для ботів (Telegram Bot API) [39]. Зазначений API надає необхідний інструментарій для розробки інтерактивних ботів, що включає обробку команд, обмін повідомленнями та реалізацію елементів користувацького інтерфейсу, таких як кнопки та інші інтерактивні компоненти. Ефективна розробка передбачає глибоке вивчення та оптимальне застосування функціональних можливостей цього API.

Іншою невід’ємною складовою предметного середовища є інтеграція з зовнішніми сервісами, зокрема з програмними інтерфейсами (API) поштових служб (наприклад, "Нова Пошта" [13], "Укрпошта" [28], "Meest Express"). Цей аспект передбачає ретельний аналіз офіційної документації наданих API, імплементацію механізмів для формування та надсилання запитів, а також коректну обробку отриманих відповідей. Важливим завданням є також передбачення та обробка потенційних помилок та обмежень, властивих цим зовнішнім API.

Процес розробки в межах цього середовища охоплює послідовність ключових етапів. Він починається з аналізу вимог, що передбачає чітке

визначення функціоналу майбутнього бота. Далі слідує етап проектування, який включає розробку архітектури програмного забезпечення та дизайну користувацького інтерфейсу в контексті платформи Telegram (визначення команд, структури кнопок, форматів повідомлень). Наступними етапами є безпосереднє кодування заявленого функціоналу, його ретельне тестування для гарантування коректної та надійної роботи, та, у підсумку, розгортання готового програмного продукту.

Таким чином, предметне середовище розробки Telegram-бота для обробки сервісів поштової доставки є багатогранною системою. Вона інтегрує програмні інструменти, зовнішні API, специфіку обраної технологічної платформи та методологічні засади створення програмного продукту, спрямованого на вирішення конкретної прикладної задачі.

1.2 Огляд наявних аналогів

З метою обґрунтування актуальності дослідження та виявлення унікальних конкурентних переваг розроблюваного Telegram-бота, призначеного для оптимізації взаємодії з сервісами поштової доставки, виникає необхідність у проведенні всебічного аналізу існуючих на ринку програмних рішень з аналогічним функціональним призначенням. Такі аналоги доцільно класифікувати за кількома основними категоріями, кожна з яких характеризується специфічними перевагами та недоліками.

Першу категорію аналізованих рішень формують офіційні Telegram-боти, розроблені безпосередньо поштовими операторами. Значна кількість провідних гравців українського ринку поштових послуг, таких як "Нова Пошта" [32] та "Укрпошта", надають своїм клієнтам подібні програмні інструменти. Ключовою перевагою таких ботів виступає їх пряма та надійна інтеграція з внутрішніми інформаційними системами оператора, що забезпечує високий рівень достовірності та оперативності наданих даних. Як правило, функціонал цих ботів охоплює широкий спектр послуг, специфічних

для конкретного оператора, включаючи відстеження поштових відправлень, пошук найближчих пунктів обслуговування, виклик кур'єрської служби, а в окремих випадках – можливість створення електронних транспортних накладних [26]. Водночас, суттєвим обмеженням є їхня прив'язка до конкретного поштового оператора. Це створює незручності для користувачів, які активно взаємодіють з декількома службами доставки, змушуючи їх встановлювати та використовувати декілька окремих ботів, кожен з яких може мати індивідуальний, відмінний від інших, користувацький інтерфейс та логіку функціонування.

Наступну групу становлять неофіційні Telegram-боти [4]. Ці проекти, зазвичай розроблені сторонніми ентузіастами або невеликими незалежними командами, орієнтовані на агрегацію даних від різних поштових служб в єдиному інтерфейсі. Основною їхньою перевагою є прагнення до уніфікації доступу до інформації від декількох операторів. Однак, такі боти часто характеризуються обмеженим функціоналом, концентруючись переважно на функції відстеження відправлень, і не пропонують розширених можливостей, таких як розрахунок вартості доставки, пошук відділень чи інші додаткові сервіси. Окрім того, перелік підтримуваних операторів зазвичай є обмеженим, а також питання, пов'язані з безпекою та конфіденційністю оброблюваних даних користувача, можуть викликати обґрунтовані застереження.

Третю категорію аналізованих рішень репрезентують мобільні додатки, що надаються безпосередньо поштовими операторами. Компанії "Нова Пошта" [11], "Укрпошта", "Meest Express" та інші активно розробляють та підтримують власні мобільні застосунки. Сильною стороною таких додатків є надання найбільш вичерпного набору функцій для конкретного оператора, що може включати створення відправлень, можливість здійснення онлайн-платежів, участь у програмах лояльності та доступ до інших специфічних сервісів. Графічний інтерфейс таких застосунків, як правило, ретельно продуманий та оптимізований для використання на мобільних пристроях. Проте, основним недоліком залишається необхідність інсталяції окремих

додатків для кожної поштової служби, що призводить до використання значного обсягу пам'яті пристрою та вимагає регулярних оновлень. Аналогічно до офіційних ботів, це спричиняє фрагментацію користувацького досвіду та необхідність перемикання між різними програмними середовищами. Для оперативного виконання елементарних завдань, наприклад, перевірки статусу відправлення, використання повнофункціонального мобільного додатку може виявитися менш ергономічним, ніж взаємодія через месенджер.

Останню розглянуту групу аналогів складають веб-сайти поштових операторів та універсальні веб-агрегатори, призначені для відстеження відправлень [3]. Офіційні веб-ресурси поштових служб зазвичай містять вичерпну інформацію щодо послуг, тарифної політики та умов надання сервісів. Також існують спеціалізовані веб-агрегатори, які забезпечують можливість відстеження відправлень численних міжнародних та національних перевізників. Перевагою використання веб-сайтів є доступ до широкого інформаційного масиву. Однак, для користувачів мобільних пристроїв вони часто є менш зручними, оскільки потребують запуску веб-браузера та навігації по сторінках сайту. Крім того, відсутня можливість отримання оперативних push-повідомлень у звичному форматі месенджера, а інтерфейси окремих веб-ресурсів можуть бути переобтяжені інформаційним контентом або рекламними елементами.

Проведений аналіз існуючих аналогічних рішень дозволяє констатувати наявність на ринку незаповненої ніші для Telegram-бота, який би поєднував переваги розглянутих підходів, нівелюючи притаманні їм недоліки. Незважаючи на те, що офіційні боти та мобільні додатки пропонують розширений функціонал для своїх конкретних операторів, вони не забезпечують комплексного вирішення для користувачів, які регулярно взаємодіють з декількома поштовими операторами. Наявні сторонні агрегатори часто характеризуються обмеженим набором функцій,

недостатньою надійністю або неповною підтримкою провідних українських поштових операторів.

1.3 Постановка задачі

Базуючись на проведеному аналізі предметного середовища та огляді існуючих аналогічних рішень, метою даної дипломної роботи є розробка Telegram-бота.

Призначенням даного бота є забезпечення користувачам зручного та уніфікованого доступу до основних сервісів кількох провідних поштових операторів України. Ключовим завданням розроблюваного програмного продукту є агрегація даних від різноманітних поштових служб, що має на меті спростити взаємодію користувачів з їхніми послугами за допомогою платформи Telegram.

Для досягнення окресленої мети передбачається вирішення низки ключових завдань.

Першим завданням є проектування архітектури Telegram-бота. На даному етапі визначаються основні структурні компоненти бота, принципи їх взаємодії та механізми передачі даних. Важливим аспектом є також обґрунтований вибір технологічного стеку: мови програмування, відповідних інструментів (фреймворків, бібліотек) та, за необхідності, системи управління базами даних.

Наступним важливим завданням є розробка функціональних можливостей для взаємодії з API поштових операторів. Передбачається реалізація надійних механізмів інтеграції з API провідних українських поштових служб, таких як "Нова Пошта", "Укрпошта" чи "Meest Express". Ця робота охоплює створення модулів для коректного формування та надсилання запитів на відстеження відправлень за їх трекінг-номерами, отримання даних щодо орієнтовної вартості доставки на основі введених параметрів, а також для реалізації пошуку найближчих до користувача відділень або поштоматів.

Третій блок завдань пов'язаний з реалізацією основного користувацького функціоналу бота. Сюди входить, насамперед, функція відстеження відправлень, яка дозволить користувачеві вводити трекінг-номер та оперативно отримувати актуальну інформацію про поточний статус та місцезнаходження його посилки від обраного поштового оператора.

Також необхідно реалізувати функцію розрахунку вартості доставки, що надасть користувачеві можливість отримати орієнтовну ціну пересилки після введення основних параметрів відправлення, таких як вага, габарити, місто відправлення та місто отримання.

Не менш важливою є функція пошуку відділень/поштоматів, яка дозволить знаходити найближчі пункти обслуговування поштових операторів, використовуючи геолокацію користувача або введену ним адресу чи назву населеного пункту.

Додатково, розглядається можливість реалізації збереження історії запитів, що дозволить користувачеві легко переглядати історію своїх відстежень або збережені трекінг-номери для швидкого повторного доступу, а також розробка механізму автоматичних сповіщень про зміну статусів відстежуваних відправлень.

Четверте завдання полягає у розробці інтуїтивно зрозумілого та ергономічного користувацького інтерфейсу безпосередньо в середовищі Telegram. Це завдання вимагає створення логічної та легкої у використанні системи текстових команд, інтерактивних кнопок та інформативних повідомлень. Інтерфейс має бути спроектований таким чином, щоб забезпечити користувачам легку навігацію в межах функціоналу бота та оперативне отримання необхідної інформації без надмірних зусиль.

Завершальними етапами є тестування розробленого Telegram-бота та підготовка технічної документації. Процес тестування має охоплювати перевірку коректності функціонування всіх реалізованих можливостей, стабільності взаємодії з API поштових служб, а також оцінку зручності користувацького інтерфейсу. За результатами тестування можливе внесення

необхідних коригувань. Підготовка документації передбачатиме опис архітектури системи, перелік використаних технологій, розробку інструкції користувача та надання рекомендацій щодо подальшого розвитку та технічної підтримки бота.

Успішне вирішення поставлених завдань дозволить створити програмний продукт, що відповідатиме заявленій меті та матиме значну практичну цінність для користувачів сервісів поштової доставки.

Висновки до першого розділу

У даному розділі було проведено комплексний аналіз, що заклав теоретичне та практичне підґрунтя для розробки Telegram-бота для обробки сервісів поштової доставки.

По-перше, було визначено предметне середовище розробки. Встановлено, що воно є багатокомпонентною системою, яка включає платформу Telegram та її Bot API, зовнішні API поштових операторів (таких як "Нова Пошта", "Укрпошта" чи "Meest Express"), а також методологію проектування, розробки, тестування та розгортання програмного продукту. Це середовище вимагає глибокого розуміння взаємодії між зазначеними компонентами для успішної реалізації проекту.

По-друге, проведений детальний огляд наявних аналогів дозволив виявити сильні та слабкі сторони існуючих рішень. Аналіз офіційних Telegram-ботів та мобільних додатків поштових операторів, неофіційних ботів-агрегаторів та веб-сайтів показав, що, незважаючи на наявність різноманітних інструментів, на ринку існує незаповнена ніша. Зокрема, відсутнє уніфіковане рішення в рамках популярного месенджера Telegram, яке б надавало зручний доступ до сервісів кількох ключових українських поштових операторів, поєднуючи широкий функціонал з простотою використання.

По-третє, на основі аналізу предметного середовища та виявлених недоліків існуючих рішень, було сформульовано чітку постановку задачі дипломної роботи. Визначено основну мету – розробка Telegram-бота, що забезпечить зручний доступ до сервісів провідних поштових операторів України. Для досягнення цієї мети було окреслено ключові завдання: проектування архітектури бота, розробка механізмів взаємодії з API поштових служб, реалізація основного користувацького функціоналу (відстеження відправлень, розрахунок вартості, пошук відділень, історія запитів та сповіщення), створення інтуїтивно зрозумілого інтерфейсу, а також тестування та підготовка документації.

Таким чином, перший розділ обґрунтовує актуальність обраної теми, визначає специфіку предметної області та формулює конкретні завдання, вирішення яких дозволить створити затребуваний та функціональний програмний продукт.

РОЗДІЛ 2
ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Аналіз предметної області

Для успішного проектування програмного продукту, здатного ефективно взаємодіяти з сервісами поштової доставки, необхідно провести детальний аналіз його предметної області. Цей аналіз полягає у формалізації знань про ключові сутності, їхні атрибути, а також процеси та інформаційні потоки, що виникають під час роботи системи. Предметною областю даної роботи є сукупність процесів, пов'язаних з отриманням інформації від поштових операторів, її обробкою та наданням користувачеві у зручному вигляді через платформу Telegram.

2.1.1 Вхідні дані системи

Для коректного функціонування розробленої системи та виконання поставлених завдань, вона повинна отримувати та обробляти набір вхідних даних. Ці дані можна класифікувати за їхнім джерелом та призначенням.

1. Дані, що надходять від користувача через інтерфейс Telegram:

- Команди:
    - /start: Стандартна команда Telegram для початку або перезапуску діалогу з ботом.
- Текстові повідомлення:
    - Трекінг-номер: Рядок, що складається з 11 або 14 цифр, для відстеження відправлень "Нової Пошти".
    - Назва населеного пункту: Текстовий рядок (напр., "Куп’янськ", "Чорнобай") для пошуку відділень за назвою.
    - Параметри для розрахунку вартості: Послідовність текстових повідомлень, що містять назви міст відправлення та отримання, а також числові значення ваги та оголошеної вартості посилки.

- Натискання на кнопки ReplyKeyboardMarkup:
    - Текстові повідомлення, що точно відповідають написам на кнопках головного меню ("Відстежити посилку", "Мої посилки", "Знайти відділення", "Розрахувати вартість посилки") або кнопки скасування ("Повернутись в меню").
- Дані про геолокацію:
    - Об'єкт Location від Telegram, що містить географічні координати користувача (широту та довготу), який надсилається при натисканні на відповідну кнопку.
- Дані CallbackQuery:
    - Спеціальні дані, що генеруються при натисканні на Inline-кнопки (кнопки під повідомленням). Вони містять унікальний ідентифікатор (callback_data), який використовується для таких дій, як збереження, видалення або оновлення статусу конкретної посилки.

2. Дані, що надходять від зовнішніх API (у нашому випадку API "Нової Пошти") [7]:

Ці дані є результатом запитів, які бот робить до зовнішніх сервісів.

- Дані про статус відправлення: JSON-об'єкт, що містить повну інформацію про посилку, з якого система вилучає ключові поля: Status, ScheduledArrivalDate, DocumentWeight тощо.
- Список відділень: Масив JSON-об'єктів, де кожен об'єкт представляє одне відділення та містить його опис (Description), координати (Latitude, Longitude) та графік роботи (Schedule).
- Дані про розрахунок вартості: JSON-об'єкт, що містить розраховану вартість доставки (Cost).
- Довідкові дані: JSON-об'єкти, що містять системні ідентифікатори (Ref) для населених пунктів, необхідні для формування коректних запитів на розрахунок вартості.

3. Дані з внутрішньої бази даних (SQLite):

Ці дані використовуються для реалізації довготривалих функцій, таких як збереження посилок та автоматичні сповіщення.

- Список збережених посилок користувача: Результат SQL-запиту SELECT, що повертає всі трекінг-номери, пов'язані з конкретним user_id.
- Список всіх активних посилок: Результат SQL-запиту SELECT, що повертає всі посилки з бази, які ще не позначені як доставлені. Використовується фоновою задачею для періодичної перевірки статусів.

2.1.2 Вихідні дані системи

Вихідні дані системи – це вся інформація, яку генерує та надсилає бот у відповідь на запити користувача або в результаті роботи фонових процесів. Основною метою формування вихідних даних є надання користувачеві необхідної інформації у зрозумілому, структурованому та зручному вигляді. Всі вихідні дані передаються через Telegram Bot API.

1. Текстові повідомлення:

Це основний спосіб комунікації бота з користувачем.

- Інформація про статус посилки: Структуроване повідомлення, що містить поточний статус відправлення, його номер та назву поштового оператора. Повідомлення форматується за допомогою HTML-тегів (<b>, <code>) для кращої читабельності.
- Результат розрахунку вартості: Повідомлення, що містить орієнтовну вартість доставки в гривнях.
- Інформаційні та сервісні повідомлення: Текстові відповіді, що супроводжують діалоги, наприклад: "Будь ласка, надішліть трекінг-номер", "Шукаю інформацію...", "Дію скасовано", "Невідома команда" тощо.
- Повідомлення про помилки: Тексти, що інформують користувача про неможливість виконати запит (напр., "Інформація про посилку не

знайдена", "Неправильний формат трекінг-номера", "Помилка з'єднання з сервером").

2. Повідомлення зі списками та спеціальним форматуванням:

- Список відділень: Послідовність окремих повідомлень, де кожне повідомлення містить інформацію про одне відділення: його опис, відстань до користувача та актуальний статус роботи (відчинено/зачинено/скоро зачиняється).
- Список збережених посилок: Послідовність повідомлень, де кожне містить номер збереженої посилки та її поточний статус.

3. Клавіатури (Keyboards):

Бот надсилає різні типи клавіатур для спрощення взаємодії.

- ReplyKeyboardMarkup (основна клавіатура):
  - Головне меню: Набір кнопок з основними функціями ("Відстежити посилку", "Мої посилки" тощо), який повертається після завершення будь-якої дії.
  - Клавіатура запиту геолокації: Спеціальна клавіатура з кнопкою, що ініціює вбудований в Telegram інтерфейс надсилання геолокації.
  - Клавіатура скасування: Кнопка "Повернутись в меню", що дозволяє перервати будь-який покроковий сценарій.
- InlineKeyboardMarkup (кнопки під повідомленням):
  - Кнопки керування посилкою: Кнопки "Оновити статус" та "Видалити", що прикріплюються до повідомлення про збережену посилку.
  - Кнопка збереження посилки: Кнопка "Зберегти цю посилку", що з'являється після успішного відстеження нового номера.
  - Кнопка "Показати на карті": Кнопка з URL, що веде на Google Maps з координатами конкретного відділення.
  - Кнопки вибору способу пошуку: Кнопки "За геолокацією" та "За назвою міста".

4. Автоматичні Push-сповіщення:

Повідомлення про зміну статусу: Асинхронне повідомлення, яке бот надсилає користувачеві у фоновому режимі, коли статус однієї з його збережених посилок змінюється. Воно містить номер посилки та її новий статус.

5. Спливаючі сповіщення :

Короткі, тимчасові сповіщення, що з'являються у верхній частині екрану або в центрі після натискання на кнопку.

- Підтвердження дій: "Посилку збережено!", "Посилку видалено.", "Оновлюю статус...".
- Повідомлення про помилки: "Ця посилка вже є у вашому списку.", "Неможливо зберегти посилку з помилкою.".

2.2 Проєктування системи

На основі аналізу предметної області та сформульованих вимог було проведено проектування програмної системи. Метою цього етапу є визначення архітектури майбутнього Telegram-бота, його основних компонентів, зв'язків між ними та моделей даних, що забезпечить надійну, масштабовану та легку в підтримці реалізацію.

2.2.1 Архітектура системи

Для реалізації Telegram-бота була обрана модульна архітектура, що передбачає розподіл програми на самостійні компоненти, кожен з яких відповідає за конкретну функціональність. Такий підхід значно полегшує розробку, спрощує тестування та полегшує подальше масштабування та підтримку системи [24].

Система складається з наступних ключових модулів (рис. 2.1):

1. Головний модуль (main.py): Точка входу в програму. Відповідає за ініціалізацію та запуск бота, диспетчера та фонових задач. Реєструє всі обробники з інших модулів.
2. Модуль обробників (handlers): Містить логіку, що безпосередньо реагує на дії користувача. Цей модуль можна далі розділити на підмодулі за функціональною ознакою (обробка команд, сценаріїв відстеження, пошуку тощо).
3. Модуль клієнтів API (api_clients.py): Інкапсулює всю логіку взаємодії з зовнішніми сервісами (в даному випадку, з API "Нової Пошти"). Це дозволяє ізолювати залежність від зовнішніх API і легко додавати нових операторів у майбутньому.
4. Модуль бази даних (database.py): Відповідає за всі операції з базою даних SQLite: створення таблиць, додавання, отримання, оновлення та видалення записів.
5. Модуль клавіатур (keyboards.py): Містить функції для генерації всіх типів клавіатур (як Reply, так і Inline), що робить код обробників чистішим.
6. Модуль фонових задач (background_tasks.py): Містить логіку, що виконується за розкладом (наприклад, перевірка статусів посилок), і не залежить від прямої взаємодії з користувачем.

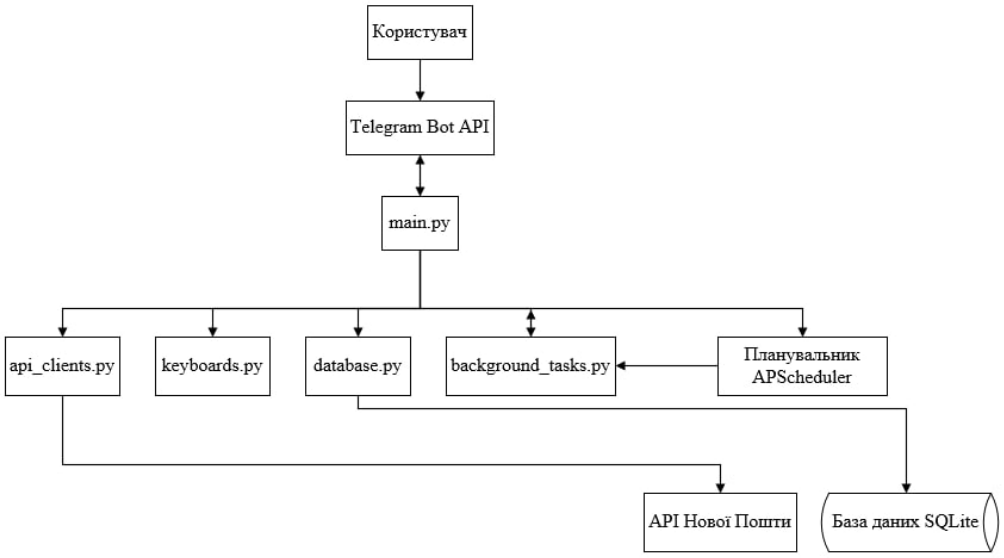


Рисунок 2.1 – Блок-схема програмного продукту

2.2.2 Проектування моделі даних

Для зберігання інформації про відстежувані користувачами посилки була спроектована модель даних, реалізована у вигляді таблиці в базі даних SQLite [23].

Таблиця tracked_packages

Назва поля	Тип даних	Призначення
id	INTEGER	Унікальний ідентифікатор запису (первинний ключ).
user_id	INTEGER	Ідентифікатор користувача в Telegram.
tracking_number	TEXT	Трекінг-номер посилки.
last_status	TEXT	Останній відомий статус посилки, для відстеження змін.
is_delivered	INTEGER	Прапорець (0 або 1), що вказує, чи доставлена посилка.

Обмеження:

- PRIMARY KEY (id): Забезпечує унікальність кожного запису.
- UNIQUE (user_id, tracking_number): Гарантує, що один користувач не може зберегти один і той самий трекінг-номер двічі.

2.2.3 Проєктування діалогів за допомогою FSM

Для реалізації покрокових сценаріїв взаємодії з користувачем, таких як розрахунок вартості доставки, було вирішено застосувати механізм машини скінченних станів (Finite State Machine, FSM) [20], наданий бібліотекою aiogram [27].

FSM дозволяє "запам'ятовувати", на якому етапі діалогу знаходиться користувач, і відповідно реагувати на його наступні повідомлення. Такий

підхід дозволяє створювати складні, розгалужені та надійні діалоги, легко керуючи контекстом розмови з кожним окремим користувачем.

2.3 Математичне та алгоритмічне забезпечення

Для реалізації функціоналу розробленого Telegram-бота було використано низку алгоритмів та математичних підходів, які забезпечують обробку даних, керування діалогами та взаємодію з зовнішніми системами.

2.3.1 Алгоритм обробки оновлень від Telegram

В основі роботи бота лежить алгоритм обробки вхідних оновлень, реалізований за допомогою бібліотеки aiogram. Цей процес можна описати наступною послідовністю кроків:

1. Отримання оновлення: Бот встановлює довготривале з'єднання з серверами Telegram (метод Long Polling) і очікує на нові події (повідомлення, натискання кнопок тощо).
2. Диспетчеризація: Отримане оновлення передається до головного диспетчера (Dispatcher).
3. Фільтрація та маршрутизація: Диспетчер послідовно перевіряє оновлення на відповідність зареєстрованим фільтрам (наприклад, фільтр команди /start, фільтр тексту кнопки, фільтр поточного стану FSM).
4. Виклик обробника: Як тільки знайдено перший фільтр, що відповідає оновленню, диспетчер викликає пов'язану з ним асинхронну функцію-обробник (handler).
5. Виконання логіки: Обробник виконує бізнес-логіку: звертається до функцій API, бази даних, формує відповідь.
6. Відправка відповіді: Сформована відповідь надсилається користувачеві через Telegram Bot API.

Цей алгоритм забезпечує асинхронну та неблокуючу обробку запитів від багатьох користувачів одночасно.

2.3.2 Машина скінченних станів (FSM) для керування діалогом

Для реалізації покрокових сценаріїв, таких як розрахунок вартості доставки, було застосовано модель машини скінченних станів (Finite State Machine). Ця модель описує систему, яка може перебувати в одному з кількох скінченних станів у будь-який момент часу.

- Стани (States): Кожен етап діалогу представлений як окремий стан (напр., waiting_for_departure_city, waiting_for_arrival_city).
- Переходи (Transitions): Перехід зі стану в стан відбувається при отриманні певного вхідного сигналу (повідомлення від користувача).
- Дії (Actions): При кожному переході виконуються певні дії: збереження отриманих даних у тимчасове сховище (state.update_data), надсилання наступного запитання користувачеві.
- Кінцевий стан: Після отримання всіх необхідних даних система переходить у кінцевий стан (або скидає стан - state.clear()), виконує фінальну дію (розрахунок вартості) і повертається у звичайний режим очікування.

Використання FSM дозволяє надійно керувати контекстом розмови з кожним користувачем окремо, не плутаючи діалоги між собою.

2.3.3 Алгоритм пошуку найближчих відділень

Для реалізації функції пошуку найближчих пунктів обслуговування було розроблено комбінований алгоритм, що включає взаємодію з API та математичні обчислення на стороні клієнта:

1. Отримання координат користувача: Система отримує географічні координати (широту φ_1 та довготу γ_1) від користувача.
2. Запит до API: Формується HTTP-запит до API "Нової Пошти" з отриманими координатами для отримання списку всіх відділень у заданому радіусі (напр., 20 км).

3. Обчислення відстані: Для кожного відділення і зі списку, отриманого від API, система:
    - Вилучає його координати (φ_2, γ_2).
    - Обчислює точну відстань d_i між точкою користувача та точкою відділення за допомогою формули Гаверсинуса [21], яка враховує кривизну Землі. Формула має вигляд:

$$a = \sin^2(\frac{\Delta\varphi}{2}) + \cos(\varphi_1) * \cos(\varphi_1) * \sin^2(\frac{\Delta\gamma}{2})$$

$$c = 2 * a \tan 2(\sqrt{a}, \sqrt{1-a}$$

$$d = R * c$$

де $\Delta\varphi = \varphi_2 - \varphi_1, \Delta\gamma = \gamma_2 - \gamma_1$, а R - радіус Землі (приблизно 6371 км). Для реалізації цих обчислень використано бібліотеку geopy [41].

4. Сортування: Список усіх отриманих відділень сортується за зростанням обчисленої відстані d_i.
5. Вивід результату: Користувачеві надсилається інформація про перші N елементи відсортованого списку.

Цей алгоритм забезпечує високу точність результатів, оскільки фінальне сортування за відстанню відбувається на стороні бота, що нівелює можливі неточності сортування в самому API.

2.3.4 Алгоритм роботи фонової задачі

Для реалізації автоматичних сповіщень використовується планувальник APScheduler [29], який за розкладом запускає наступний алгоритм:

1. Запуск за таймером: Планувальник активує функцію перевірки через заданий інтервал часу (напр., кожні 30 хвилин).
2. Отримання активних посилок: Система робить SQL-запит до бази даних для отримання списку всіх посилок, що не мають позначки "доставлено" (is_delivered = 0).
3. Ітеративна перевірка: Для кожної посилки зі списку виконується:
    - Запит до API "Нової Пошти" для отримання її поточного статусу.

- Порівняння отриманого статусу зі статусом, збереженим у базі даних (last_status).

4. Прийняття рішення:
   - Якщо статуси відрізняються, система надсилає сповіщення користувачеві (за його user_id) та оновлює запис у базі даних новим статусом.
   - Якщо статуси збігаються, жодних дій не виконується.
5. Завершення: Після перевірки всіх посилок функція завершує роботу до наступного виклику планувальником.

Висновки до другого розділу

У другому розділі дипломної роботи було виконано комплекс робіт з проектування інформаційної системи та розробки її математичного й алгоритмічного забезпечення.

По-перше, було проведено аналіз предметної області, в ході якого ідентифіковано ключові сутності (Користувач, Поштове відправлення, Пункт обслуговування), визначено їхні атрибути та описано основні функціональні процеси, такі як відстеження, розрахунок вартості, пошук відділень та керування збереженими посилками. Це дозволило формалізувати вимоги до майбутньої системи.

По-друге, на основі проведеного аналізу було спроектовано програмну систему. Розроблено модульну архітектуру, що розділяє логіку на функціональні компоненти, забезпечує гнучкість та масштабованість проекту. Також було спроектовано модель даних, реалізовану у вигляді таблиці в базі даних SQLite, для зберігання інформації про відстежувані посилки.

По-третє, було детально описано математичне та алгоритмічне забезпечення системи. Розглянуто основний алгоритм обробки оновлень від Telegram, застосування моделі машини скінченних станів (FSM) для реалізації складних покрокових діалогів, а також розроблено комбінований алгоритм

пошуку найближчих відділень з використанням формули Гаверсинуса для точного обчислення відстаней. Окрім цього, описано алгоритм роботи фонової задачі для автоматичних сповіщень про зміну статусів.

Таким чином, у даному розділі було закладено повне теоретичне та технічне підґрунтя для подальшої програмної реалізації Telegram-бота. Створена архітектура, модель даних та розроблені алгоритми є фундаментом для створення надійного та функціонального продукту.

РОЗДІЛ 3
ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Засоби розробки

Для створення програмного продукту було використано набір сучасних та поширених у галузі технологій та інструментів, які забезпечили ефективний процес розробки та надійність кінцевого рішення.

1. Мова програмування: Python 3.11 [34]

В якості основної мови програмування було обрано Python. Це високорівнева, інтерпретована мова з динамічною типізацією, яка ідеально підходить для швидкої розробки прототипів та створення складних систем завдяки своєму чистому синтаксису та великій стандартній бібліотеці [25].

2. Середовище розробки (IDE): JetBrains PyCharm

Розробка велася в інтегрованому середовищі розробки PyCharm Community Edition [31]. Дана IDE надає потужні інструменти для написання та зневадження коду, включаючи інтелектуальне автодоповнення, статичний аналіз коду, вбудований термінал, інструменти для роботи з базами даних та інтеграцію з системами контролю версій [22].

3. Основні бібліотеки та фреймворки:

Aiogram 3. Сучасний, потужний та асинхронний фреймворк для створення Telegram-ботів на Python. Його архітектура, побудована на базі asyncio, дозволяє створювати високопродуктивні та відзивчиві додатки. Були використані ключові компоненти, такі як Dispatcher для маршрутизації оновлень, фільтри для гнучкої обробки повідомлень та FSM (Finite State Machine) для реалізації покрокових діалогів [20].

Requests: Стандарт де-факто в екосистемі Python для виконання HTTP-запитів. Ця бібліотека використовувалася для взаємодії з REST API "Нової Пошти" завдяки своєму простому та інтуїтивно зрозумілому інтерфейсу.

APScheduler: Потужна бібліотека для планування фонових завдань. В проекті використано її асинхронну версію (AsyncIOScheduler) для періодичного запуску функції перевірки статусів посилок без блокування основного циклу роботи бота.

python-dotenv: Бібліотека, що дозволяє завантажувати змінні середовища (такі як токени та API-ключі) з файлу .env [33]. Це забезпечує безпечне зберігання конфігураційних даних окремо від основного коду.

Geopy: Бібліотека для виконання географічних обчислень. У проекті вона використовувалася для точного розрахунку відстані між координатами користувача та відділень за формулою Гаверсинуса.

4. Система управління базами даних: SQLite 3

Для зберігання даних про відстежувані посилки була використана легка, файлова СУБД SQLite. Вона не вимагає окремого серверного процесу і ідеально підходить для додатків середнього розміру. Взаємодія з базою даних здійснювалася за допомогою вбудованого в Python модуля sqlite3 [35].

3.2 Вимоги до технічного та програмного забезпечення

Для успішної розробки, розгортання та функціонування розробленого Telegram-бота необхідно дотримуватися певних вимог до апаратного та програмного забезпечення. Ці вимоги можна розділити на дві категорії: вимоги до середовища розробки та вимоги до середовища експлуатації.

3.2.1 Вимоги до середовища розробки

Середовище розробки – це конфігурація комп'ютера, на якому ведеться написання, тестування та зневадження коду [19].

Мінімальні апаратні вимоги:

- Процесор: Будь-який сучасний процесор.
- Оперативна пам'ять (RAM): 2 ГБ.

- Вільне місце на диску: 5 ГБ (для IDE, інтерпретатора, бібліотек та файлів проекту).
- Мережеве підключення: Стабільне підключення до мережі Інтернет для завантаження бібліотек та взаємодії з API.

Програмне забезпечення:

- Операційна система: Windows 10/11, macOS 10.15+, або будь-який сучасний дистрибутив Linux (напр., Ubuntu 20.04+).
- Інтерпретатор Python: Версія 3.9 або вище.
- Середовище розробки (IDE): JetBrains PyCharm 2023.x або новіше, або будь-який інший редактор коду з підтримкою Python (напр., Visual Studio Code).
- Система контролю версій: Git версії 2.25 або вище.

Додаткові інструменти (опціонально): Програма для роботи з базами даних SQLite, наприклад, DB Browser for SQLite.

3.2.2 Вимоги до середовища експлуатації

Середовище експлуатації – це сервер, на якому бот буде запущений для постійної роботи 24/7.

Мінімальні апаратні вимоги:

- Процесор (CPU): 2 vCPU (віртуальне процесорне ядро).
- Оперативна пам'ять (RAM): 4 ГБ.
- Вільне місце на диску: 40 ГБ.
- Мережеве підключення: Постійне та стабільне підключення до мережі Інтернет.

Такі вимоги задовольняє більшість сучасних хмарних провайдерів (напр., DigitalOcean, Heroku, Vultr, AWS, Google Cloud [30]) на мінімальних тарифних планах.

Програмне забезпечення:

- Операційна система: Рекомендовано серверний дистрибутив Linux (напр., Ubuntu Server, Debian, CentOS).

- Інтерпретатор Python: Версія 3.9 або вище.
- Встановлені залежності: Усі бібліотеки, перелічені у файлі requirements.txt проекту, зокрема: aiogram, requests, apscheduler, python-dotenv, geopy
- Менеджер процесів (опціонально, рекомендовано): Інструмент, що забезпечить автоматичний перезапуск бота у випадку збою, наприклад, systemd або supervisor.

3.2.3 Вимоги до користувацького середовища

Для взаємодії з розробленим програмним продуктом користувачеві не потрібно встановлювати жодного додаткового програмного забезпечення, окрім клієнта месенджера Telegram.

Апаратні вимоги [40]:

Будь-який пристрій (смартфон, планшет, персональний комп'ютер), здатний запустити додаток Telegram.

Програмні вимоги:

Операційна система: Будь-яка ОС, що підтримується Telegram (Android, iOS, Windows, macOS, Linux).

Додаток: Встановлений та активований клієнт месенджера Telegram актуальної версії.

Мережеве підключення: Активне підключення до мережі Інтернет для обміну повідомленнями з серверами Telegram.

3.3 Опис програмної реалізації

Програмна реалізація Telegram-бота була виконана на мові Python з використанням асинхронного фреймворку aiogram. Розробка велася відповідно до спроектованої модульної архітектури, що дозволило чітко розділити відповідальність між компонентами системи. У цьому підрозділі детально розглянуто імплементацію основних функціональних блоків.

3.3.1 Ініціалізація та запуск системи

Точкою входу в програму є файл main.py, який виконує послідовність ініціалізаційних дій, необхідних для запуску всіх компонентів системи.

1. Завантаження конфігурації: Секретні ключі та токени завантажуються з файлу .env за допомогою бібліотеки python-dotenv.
2. Ініціалізація бази даних: Викликається функція init_db(), яка створює файл бази даних postal_bot.db та таблицю tracked_packages, якщо вони відсутні.
3. Ініціалізація бота: Створюються основні об'єкти aiogram: Bot та Dispatcher.
4. Реєстрація обробників: Усі функції-обробники повідомлень, команд та callback-запитів реєструються в диспетчері за допомогою декораторів (@dp.message, @dp.callback_query). Порядок реєстрації є важливим, оскільки диспетчер перевіряє фільтри послідовно.
5. Запуск фонових задач: Створюється та запускається планувальник AsyncIOScheduler, до якого додається асинхронна задача check_packages_status з модуля background_tasks для періодичної перевірки статусів посилок.
6. Запуск бота: Викликається метод dp.start_polling(bot), який запускає нескінченний цикл отримання оновлень від серверів Telegram.

3.3.2 Реалізація покрокових сценаріїв за допомогою FSM

Для керування складними діалогами, такими як розрахунок вартості, було використано механізм машини скінченних станів (FSM) з aiogram. Для цього було створено клас UserDialog, що наслідується від StatesGroup та містить перелік усіх можливих станів, в яких може перебувати користувач.

Перехід між станами відбувається в функціях-обробниках. Обробник, що розпочинає сценарій, встановлює початковий стан за допомогою await state.set_state(...). Наступні обробники прив'язуються до конкретного стану і спрацьовують лише тоді, коли користувач знаходиться в цьому стані. Дані, отримані від користувача на кожному кроці, зберігаються в тимчасовому

сховищі за допомогою await state.update_data(...), а після завершення діалогу стан скидається викликом await state.clear().

3.3.3 Реалізація взаємодії з API та базою даних

Вся логіка роботи з зовнішніми та внутрішніми даними винесена в окремі модулі api_clients.py та database.py.

Функції в api_clients.py формують JSON-payload для HTTP-запитів до API "Нової Пошти" і відправляють їх за допомогою бібліотеки requests. Обов'язковим елементом є обробка винятків try-except, що дозволяє коректно реагувати на мережеві помилки або помилки на стороні сервера API.

Функції в database.py використовують вбудований модуль sqlite3 для виконання SQL-запитів. Для запобігання SQL-ін'єкціям використовуються параметризовані запити (з символом ?). З'єднання з базою даних відкривається на початку кожної функції і обов'язково закривається в блоці finally.

3.4 Керівництво користувача

Для ефективної взаємодії кінцевого користувача з розробленим програмним продуктом було спроектовано інтуїтивно зрозумілий інтерфейс, що базується на системі команд та інтерактивних кнопок. Цей підрозділ описує основні сценарії використання Telegram-бота "PostalHelperBot" та надає послідовність дій для доступу до його ключових функціональних можливостей.

3.4.1 Початок роботи

Для початку роботи з ботом необхідно знайти його в Telegram за ім'ям користувача @PostalHelperBot та натиснути кнопку "Start" або ввести команду /start.

У відповідь бот надішле вітальне повідомлення та відобразить головне меню у вигляді кнопок внизу екрана (рис. 3.1).

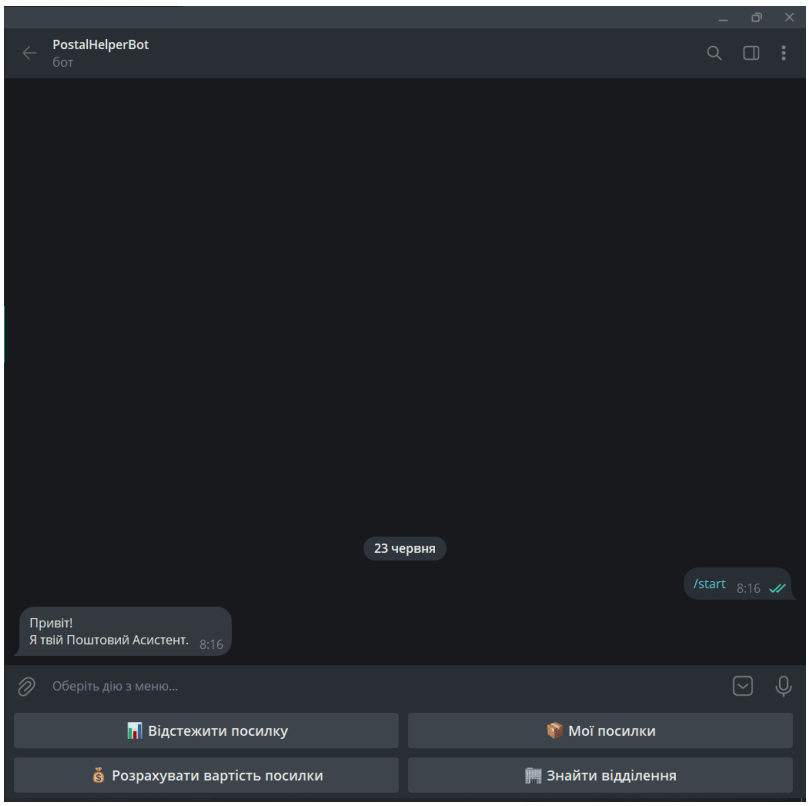


Рисунок 3.1 – Головне меню бота

Головне меню містить наступні опції:

Відстежити посилку: Перевірити статус посилки за її трекінг-номером.

Мої посилки: Переглянути список збережених посилок.

Розрахувати вартість: Дізнатися орієнтовну вартість доставки.

Знайти відділення: Знайти найближче поштове відділення за геолокацією або ручний пошук за населеним пунктом.

3.4.2 Відстеження посилки

Натисніть кнопку "Відстежити посилку" в головному меню.

Бот запропонує надіслати трекінг-номер. Введіть номер накладної (11 або 14 цифр) та надішліть його боту.

Після короткого очікування бот надішле повідомлення з актуальним статусом вашої посилки (рис. 3.2).

Рисунок 3.2 – Результат відстеження посилки

3.4.3 Робота зі збереженими посилками

Якщо після відстеження посилки ви бажаєте додати її до списку для подальшого моніторингу, натисніть на кнопку "Зберегти цю посилку", що з'явиться під повідомленням про статус.

Для перегляду списку збережених посилок:

Натисніть кнопку "Мої посилки" в головному меню.

Бот надішле окремими повідомленнями інформацію про кожну збережену посилку.

Під кожним таким повідомленням будуть кнопки для керування (рис. 3.3):

Оновити статус: Миттєво перевіряє та оновлює статус посилки.

Видалити: Видаляє посилку з вашого списку.

Після того, як ви зберегли посилку, бот буде самостійно, з певною періодичністю, перевіряти її статус. Якщо статус зміниться (наприклад, з "В дорозі" на "Прибуло у відділення"), бот автоматично надішле вам повідомлення з оновленою інформацією.

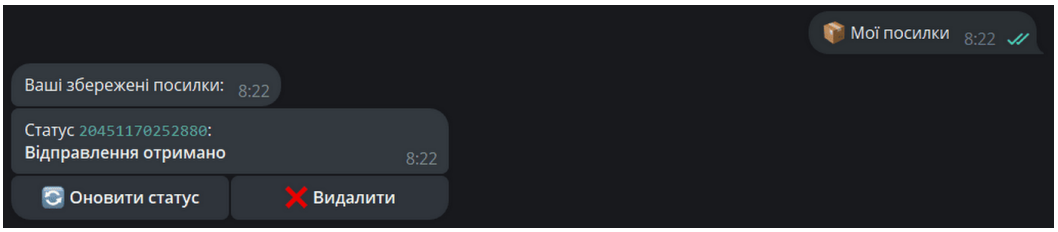


Рисунок 3.3 – Меню керування збереженою посилкою

3.4.4 Розрахувати вартість посилки

Натисніть на кнопку "Розрахувати вартість посилки".

Бот розпочне покроковий діалог, послідовно запитуючи у вас наступну інформацію:

- Місто відправлення.
- Місто отримання.
- Вагу посилки в кілограмах (можна вводити десяткові числа, напр., 0.5).
- Оголошену вартість посилки в гривнях.

Після надання всіх даних бот надішле повідомлення з орієнтовною вартістю доставки. (рис. 3.4)

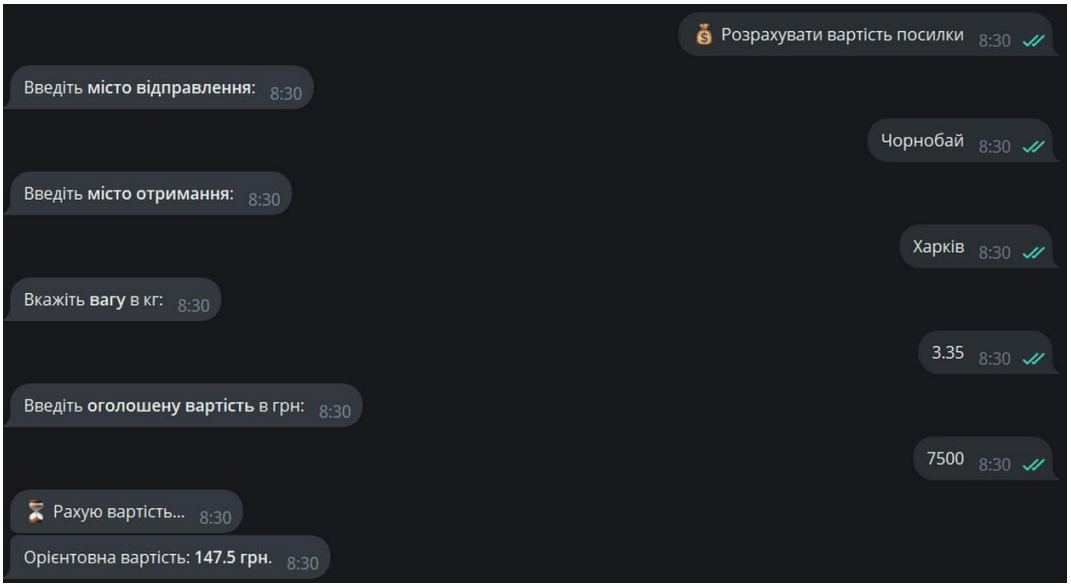


Рисунок 3.4 – Результат пошуку відділень

3.4.5 Пошук відділень

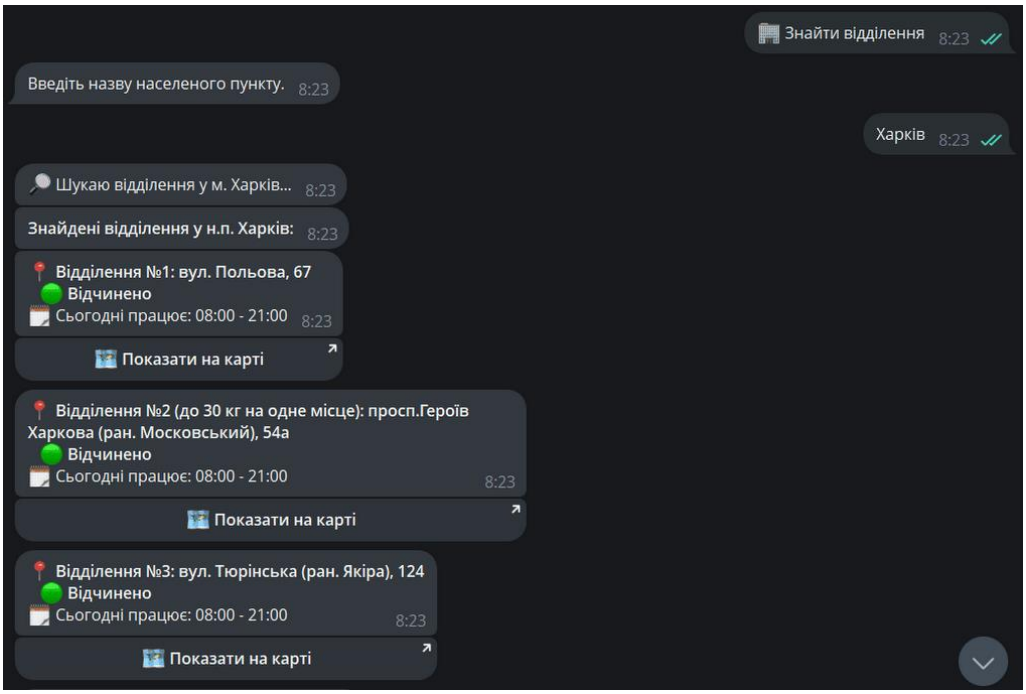
Натисніть на кнопку "Знайти відділення".

Бот запропонує вам два способи пошуку за допомогою кнопок під повідомленням:

За моєю геолокацією: Натисніть на цю кнопку. З'явиться клавіатура з пропозицією "Надіслати мою геолокацію". Натисніть на неї та підтвердіть надсилання. Бот знайде 3 найближчі до вас відділення, розрахувавши відстань.

За назвою міста: Натисніть на цю кнопку. Бот попросить вас ввести назву населеного пункту. Надішліть назву (наприклад, "Харків"), і бот покаже список відділень у цьому місті.

Для кожного знайденого відділення буде вказана його адреса, поточний статус роботи (відчинено/зачинено/скоро зачиняється) та кнопка " Показати на карті" для перегляду його розташування в Google Maps. (рис. 3.5, рис. 3.6)



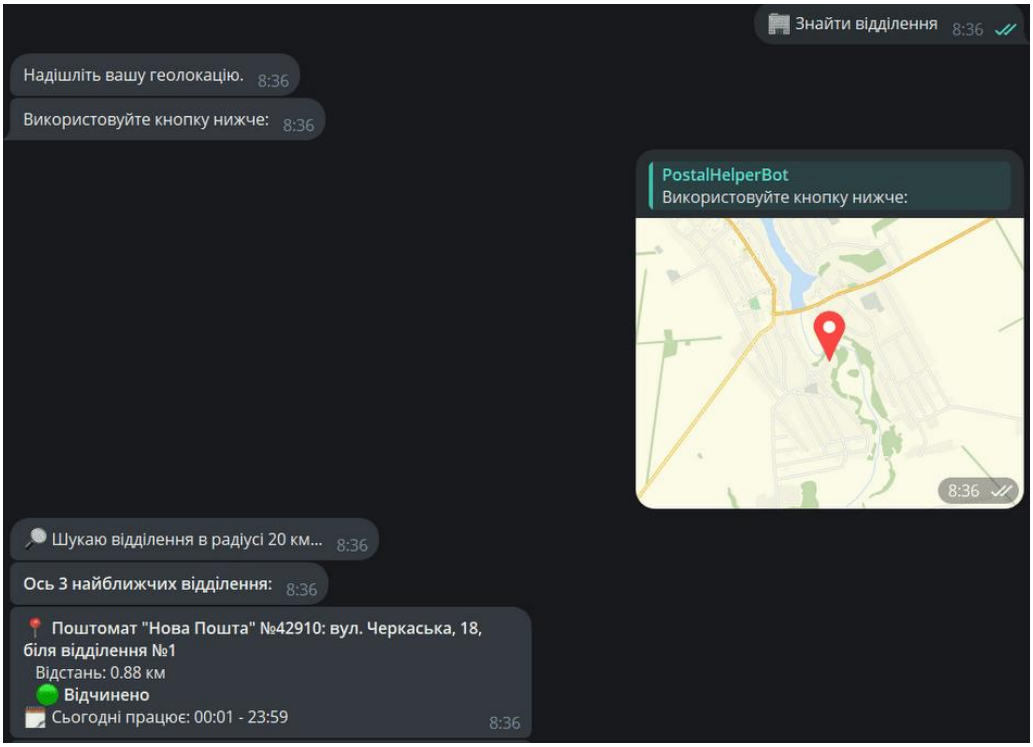


Рисунок 3.5, 3.6 – Результат пошуку відділень

3.4.6 Скасування дії

Якщо ви розпочали якийсь сценарій (наприклад, розрахунок вартості або пошук відділення) і бажаєте його перервати, ви можете в будь-який

момент натиснути на кнопку "Повернутись в меню". Бот скасує поточну операцію та поверне вас до головного меню.

Висновки до третього розділу

У третьому розділі дипломної роботи було виконано практичну реалізацію програмного продукту − Telegram-бота для обробки сервісів поштової доставки, а також описано всі технічні та програмні аспекти, пов'язані з його створенням та функціонуванням.

По-перше, було обґрунтовано вибір засобів розробки, що склали технологічний стек проекту. В якості основних інструментів було обрано мову програмування Python, асинхронний фреймворк aiogram для роботи з Telegram Bot API, СУБД SQLite для зберігання даних та бібліотеку APScheduler для реалізації фонових задач. Такий вибір дозволив створити високопродуктивний, асинхронний та функціональний додаток.

По-друге, було сформульовано вимоги до програмного та технічного забезпечення, необхідного для розгортання та експлуатації бота. Визначено мінімальні системні вимоги до серверного середовища, а також перелік програмних залежностей, що забезпечує можливість відтворення робочого оточення.

По-третє, було детально описано програмну реалізацію ключових модулів системи. Розглянуто структуру проекту, процес ініціалізації та запуску бота. Особливу увагу приділено імплементації складних функціональних блоків: реалізації покрокових діалогів за допомогою механізму машини скінченних станів (FSM), взаємодії з зовнішнім API "Нової Пошти", а також архітектурі фонової задачі для автоматичного сповіщення користувачів про зміну статусів їхніх посилок.

На завершення було розроблено керівництво користувача, яке в доступній формі описує всі основні сценарії взаємодії з ботом: від початку

роботи та відстеження посилок до використання розширених функцій, таких як пошук відділень та керування списком збережених відправлень.

Таким чином, у ході виконання робіт по третьому розділу було успішно створено програмний продукт, що повністю відповідає поставленим завданням, є стабільним, функціональним та готовим до практичного використання.

РОЗДІЛ 4
ОХОРОНА ПРАЦІ

4.1 Регулювання питань охорони праці на законодавчому рівні

Охорона праці є невід'ємною складовою трудової діяльності та ключовим елементом державної соціальної політики, спрямованої на збереження життя, здоров'я і працездатності людини [10, 12]. Ефективне функціонування системи охорони праці неможливе без чіткого та всеосяжного законодавчого регулювання, яке встановлює єдині норми та правила для всіх суб'єктів трудових відносин. Законодавча база в цій сфері формує правове поле, що гарантує створення безпечних та нешкідливих умов праці, а також визначає відповідальність за порушення встановлених норм [2].

Основним нормативно-правовим актом, що регулює відносини у сфері охорони праці в Україні, є Закон України "Про охорону праці" [16]. Цей закон визначає основні положення щодо реалізації конституційного права громадян на охорону їхнього життя і здоров'я в процесі трудової діяльності, встановлює єдиний порядок організації охорони праці та регулює відносини між роботодавцем, працівником і державою. Окрім цього, важливу роль відіграють положення Кодексу законів про працю України, який закріплює базові права працівників на безпечні умови праці та обов'язки роботодавця щодо їх забезпечення.

Реалізація законодавчих положень на рівні конкретного підприємства чи організації покладається на декількох ключових суб'єктів. Центральною фігурою в цьому процесі є роботодавець. Згідно зі статтею 13 Закону "Про охорону праці", саме роботодавець зобов'язаний створити на робочому місці в кожному структурному підрозділі умови праці відповідно до нормативно-правових актів, а також забезпечити додержання вимог законодавства щодо прав працівників у галузі охорони праці [16]. Він несе безпосередню

відповідальність за функціонування системи управління охороною праці, фінансування відповідних заходів, організацію навчання та інструктажів для працівників.

Важливу роль в організації роботи з охорони праці відіграє служба охорони праці або відповідний фахівець, створення якої є обов'язковим на підприємствах з кількістю працюючих 50 і більше осіб, згідно з НПАОП 0.00-4.21-04 [17]. Ця служба підпорядковується безпосередньо роботодавцю і виконує контрольно-методичні функції: розробляє інструкції, проводить інструктажі, контролює дотримання працівниками вимог нормативних актів, бере участь у розслідуванні нещасних випадків. Водночас, відповідальність за порушення несе не лише роботодавець, але й сам працівник, який, відповідно до статті 14 Закону "Про охорону праці", зобов'язаний дбати про особисту безпеку і здоров'я, а також про безпеку і здоров'я оточуючих людей в процесі виконання будь-яких робіт [16].

Державний нагляд та контроль за додержанням законодавства про охорону праці здійснює Державна служба України з питань праці (Держпраці). Цей центральний орган виконавчої влади має право проводити перевірки на підприємствах, видавати приписи про усунення порушень, притягати до відповідальності посадових осіб та, у разі виявлення грубих порушень, що загрожують життю працівників, призупиняти роботу виробництв чи окремого обладнання. Таким чином, законодавство створює багаторівневу систему регулювання та контролю, де кожен учасник трудових відносин має свої чітко визначені права та обов'язки, спрямовані на досягнення головної мети – створення безпечного робочого середовища [6, 9].

4.2 Виявлення потенційних небезпек стосовно об'єкту проектування

Діяльність, пов'язана з розробкою програмного забезпечення, зокрема Telegram-бота, зазвичай не належить до робіт з підвищеною небезпекою. Проте, робоче місце програміста, як і будь-яке інше, є середовищем, де можуть

виникати шкідливі та небезпечні виробничі фактори, що здатні негативно впливати на здоров'я та працездатність працівника [1]. Відповідно до класифікації, для робочого місця розробника характерні наступні групи небезпек.

До фізичних небезпечних і шкідливих виробничих факторів належить підвищений рівень електромагнітного випромінювання від комп'ютерної техніки [5], недостатня або надмірна освітленість робочої зони, а також підвищена яскравість світла та пульсація світлового потоку від монітора [14]. Недотримання оптимальних параметрів мікроклімату (температури, вологості повітря) [18] також є вагомим фактором ризику. Окрім цього, існує небезпека ураження електричним струмом у випадку несправності обладнання чи ізоляції електромереж.

До психофізіологічних факторів належать нервово-психічні та фізичні перевантаження. Розумове перенапруження, висока емоційна напруженість, пов'язана з дотриманням термінів проекту, та монотонність праці можуть призводити до стресу та професійного вигорання. Статичні фізичні перевантаження, спричинені тривалою роботою в сидячому положенні, можуть викликати захворювання опорно-рухового апарату [1].

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

Виявлення потенційних небезпек є лише першим кроком у створенні безпечного робочого середовища. Для ефективного управління охороною праці необхідно провести оцінку ризиків, пов'язаних з цими небезпеками. Оцінка ризику – це систематичний процес аналізу ймовірності виникнення небезпечної події та тяжкості її можливих наслідків. Головною задачею цієї процедури є ідентифікація найбільш значущих ризиків, їх ранжування за ступенем пріоритетності та розробка обґрунтованих заходів для їх усунення або мінімізації. Цей процес дозволяє перейти від простого реагування на

інциденти до проактивного управління безпекою, запобігаючи нещасним випадкам ще до їх виникнення.

Для проведення оцінки ризику на робочому місці розробника програмного забезпечення будемо використовувати матрицю оцінювання ризиків, наведену у Додатку 2 методичних вказівок. Цей метод полягає у визначенні рівня ризику як комбінації двох параметрів: категорії серйозності небезпеки та рівня ймовірності небезпеки.

Проведемо класифікацію та оцінку ризику для двох виявлених небезпек: "Підвищене напруження зору" та "Пожежна небезпека".

1. Оцінка ризику "Підвищене напруження зору":

Категорія серйозності небезпеки: За Таблицею 1 Додатку 2, наслідки у вигляді стійкого захворювання (погіршення зору) відповідають II категорії (Критична).

Рівень ймовірності небезпеки: За Таблицею 2, оскільки робота за комп'ютером є щоденною, велика ймовірність того, що ця подія відбудеться, відповідає рівню A (Часта).

Індекс ризику та критерій припустимості: Згідно з Таблицею 3, на перетині категорії II та рівня A знаходиться індекс 2A. Цей індекс відповідає критерію "Неприпустимий (надмірний)", що вимагає негайного впровадження заходів контролю.

2. Оцінка ризику "Пожежна небезпека":

Категорія серйозності небезпеки: За Таблицею 1, наслідки у вигляді смерті або зруйнування системи відповідають I категорії (Катастрофічна).

Рівень ймовірності небезпеки: За Таблицею 2, за умови дотримання правил експлуатації електроприладів та справної проводки, ця подія є малоймовірною, але можливою протягом життєвого циклу. Це відповідає рівню D (Віддалена).

Індекс ризику та критерій припустимості: Згідно з Таблицею 3, на перетині категорії I та рівня D знаходиться індекс 1D. Цей індекс відповідає

критерію "Небажаний (гранично допустимий)", що також потребує розробки заходів для зниження ризику.

На основі проведеної оцінки ризиків розроблено наступні заходи щодо їх попередження та мінімізації:

Заходи щодо зниження ризику підвищеного напруження зору (індекс 2A):

Організаційні: Впровадження регламентованих перерв у роботі з ПК (наприклад, 10-15 хвилин кожні 2 години). Проведення інструктажів для працівників щодо необхідності виконання гімнастики для очей.

Технічні: Забезпечення робочих місць сучасними моніторами з високою роздільною здатністю та низьким рівнем мерехтіння [14,18]. Організація правильного освітлення робочого місця, що унеможливлює відблиски на екрані [14].

Медичні: Організація періодичних медичних оглядів для своєчасного виявлення погіршення зору [15].

Заходи щодо попередження пожежної небезпеки (індекс 1D):

Організаційні: Призначення відповідальної особи за пожежну безпеку. Проведення регулярних інструктажів та тренувань з персоналом щодо дій у разі пожежі. Розробка та розміщення на видних місцях планів евакуації.

Технічні: Забезпечення приміщення справними первинними засобами пожежогасіння (вогнегасниками) та автоматичною пожежною сигналізацією. Регулярна перевірка стану електромереж та електрообладнання. Заборона використання несправних електроприладів та куріння у невідведених для цього місцях [8].

Комплексне впровадження цих заходів дозволить суттєво знизити рівень ідентифікованих ризиків та забезпечити безпечні та комфортні умови праці для розробника.

Висновки до четвертого розділу

У даному розділі було проведено комплексний аналіз питань охорони праці та безпеки життєдіяльності в контексті діяльності розробника програмного забезпечення.

По-перше, було розглянуто законодавче регулювання у сфері охорони праці. Проаналізовано ключові нормативно-правові акти, такі як Закон України "Про охорону праці" та Кодекс законів про працю, а також визначено роль та відповідальність основних суб'єктів цього процесу: роботодавця, працівника та державних контролюючих органів.

По-друге, було проведено виявлення та класифікацію потенційних небезпек, характерних для робочого місця програміста. На основі наданих методичних матеріалів було ідентифіковано основні фізичні, психофізіологічні та загальні небезпечні фактори, серед яких підвищене напруження зору, статичні навантаження, ризик ураження електричним струмом та пожежна небезпека.

По-третє, було виконано дослідження та оцінку ризиків за допомогою матричного методу. Для найбільш значущих небезпек – "підвищене напруження зору" та "пожежна небезпека" – було визначено категорії серйозності та рівні ймовірності, що дозволило розрахувати індекси ризику та класифікувати їх як "неприпустимий" та "небажаний" відповідно.

На основі проведеного аналізу було розроблено та рекомендовано комплекс організаційних, технічних та медичних заходів, спрямованих на мінімізацію виявлених ризиків. Запропоновані рішення, такі як впровадження регламентованих перерв, використання сучасного обладнання, організація правильного освітлення та дотримання норм пожежної безпеки, дозволять створити безпечні та сприятливі умови для трудової діяльності розробника.

ЗАГАЛЬНІ ВИСНОВКИ

У ході виконання даної дипломної роботи було успішно досягнуто поставленої мети.

Розроблено Telegram-бота для обробки сервісів поштової доставки, що надає користувачам зручний та уніфікований доступ до ключових послуг сервісів поштової доставки.

Проведено комплексний аналіз предметної області та існуючих на ринку аналогічних рішень. Дослідження показало, що, незважаючи на наявність офіційних додатків від окремих операторів, існує незаповнена ніша для універсального агрегатора в середовищі Telegram. Це дозволило сформулювати основні вимоги до програмного продукту та поставити конкретні задачі для його розробки.

Розроблено гнучку модульну архітектуру, що забезпечує легкість підтримки та масштабування системи. Було обґрунтовано вибір технологічного стеку, зокрема мови програмування Python та асинхронної бібліотеки aiogram, а також спроектовано структуру бази даних SQLite для зберігання відстежуваних посилок.

В рамках практичної реалізації було імплементовано весь запланований функціонал.

Створено надійні механізми взаємодії з API "Нової Пошти", реалізовано покрокові діалоги для збору даних від користувача за допомогою технології FSM та розроблено інтуїтивно зрозумілий користувацький інтерфейс. Особливу увагу було приділено розширеним можливостям, таким як система збереження посилок та проактивна система автоматичних сповіщень про зміну статусів за допомогою планувальника фонових задач.

Проведено аналіз потенційних небезпек на робочому місці розробника та оцінку ризиків. На основі отриманих результатів було запропоновано

комплекс заходів з охорони праці та пожежної безпеки, спрямованих на створення безпечних умов праці.

Таким чином, у результаті виконання дипломної роботи створено повноцінний, стабільний та функціональний програмний продукт, що вирішує актуальну прикладну задачу. Розроблений Telegram-бот є готовим до практичного використання та має потенціал для подальшого розвитку, зокрема шляхом інтеграції з API інших поштових операторів та розширення набору доступних користувачеві функцій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бедрій Я.І. Основи охорони праці користувачів персональних комп’ютерів / Я.І. Бедрій : навч. посіб. – Тернопіль. : Навчальна книга – Богдан, 2014. – 144 с.
2. Бедрій Я.І. Охорона праці та пожежна безпека / Я.І. Бедрій : навч. посіб. – Тернопіль. : Навчальна книга – Богдан, 2013. – 184 с.
3. Відстежити посилку. *Postal Ninja*. URL: https://postal.ninja/uk/track (дата звернення: 24.06.2025).
4. Де посилка? Відстежити посилку з Аліекспрес. *Telegram*. URL: https://t.me/DePosylka_bot (дата звернення: 24.06.2025).
5. Державні санітарні норми і правила при роботі з джерелами електромагнітних полів : ДСанПіН 3.3.6.096-2002. – [Чинний від 2002-12-18] . – Київ : МОЗ, 2002. – 24 с.
6. Директива Ради від 12 червня 1989 року про запровадження заходів, покликаних заохочувати до покращення безпеки та охорони здоров'я працівників на роботі (89/391/ЄЕС) : Директива Європ. екон. співтовариства від 12.06.1989 № 89/391/ЄЕС : станом на 11 груд. 2008 р. URL: https://zakon.rada.gov.ua/laws/show/994_b23#Text (дата звернення: 24.06.2025).
7. Документація API Нова пошта. URL: https://developers.novaposhta.ua/documentation (дата звернення: 24.06.2025).
8. Кодекс цивільного захисту України : Кодекс України від 02.10.2012 № 5403-VI : станом на 1 січ. 2025 р. URL: https://zakon.rada.gov.ua/laws/show/5403-17#Text (дата звернення: 24.06.2025).
9. Конвенція про основи, що сприяють безпеці та гігієні праці N 187 : Конвенція Міжнар. орг. пр. від 15.06.2006 № 187. URL: https://zakon.rada.gov.ua/laws/show/993_515#Text (дата звернення: 24.06.2025).
10. Коцан І.Я. Безпека життєдіяльності / І.Я. Коцан, О.Ю.Дмитрук, Є.П. Желібо : підручн. – Х. :Фоліо, 2014. – 463 с.
11. Мобільний додаток Нова пошта. URL: https://novaposhta.ua/mobile_app_new/ (дата звернення: 24.06.2025).
12. Піскунова Л.Е. Безпека життєдіяльності / Л.Е. Піскунова, В.А. Прилипко, Т.О. Зубок : підручн. – К. : Академія, 2014. – 224 с.

13. Портал для розробників API Нова пошта. URL: https://developers.novaposhta.ua (дата звернення: 24.06.2025).
14. Природне і штучне освітлення : ДБН В.2.5-28:2018. – [Чинний від 2019-03-01] . – Київ : ДП «НДІ БК», 2018. – 94 с.
15. Про затвердження переліку професій, виробництв та організацій, працівники яких підлягають обов'язковим профілактичним медичним оглядам, порядку проведення цих оглядів та видачі особистих медичних книжок : Постанова Каб. Міністрів України від 23.05.2001 № 559 : станом на 17 січ. 2014 р. URL: https://zakon.rada.gov.ua/laws/show/559-2001-п#Text (дата звернення: 24.06.2025).
16. Про охорону праці : Закон України від 14.10.1992 № 2694-XII : станом на 4 квіт. 2025 р. URL: https://zakon.rada.gov.ua/laws/show/2694-12#Text (дата звернення: 24.06.2025).
17. Про Типове положення про службу охорони праці : Наказ Держ. ком. України по нагляду за охорон. пр. від 03.08.1993 № 73 : станом на 12 груд. 2004 р. URL: https://zakon.rada.gov.ua/laws/show/z0140-93#Text (дата звернення: 24.06.2025).
18. Санітарні норми мікроклімату виробничих приміщень : ДСН 3.3.6.042-99. – [Чинний від 1999-12-01] . – Київ : МОЗ, 1999. – 21 с.
19. Середовище розробки - TestMatick. *TestMatick*. URL: https://testmatick.com/uk/glossary/seredovyshhe-rozrobky/ (дата звернення: 24.06.2025).
20. Учасники проектів Вікімедіа. Скінченний автомат – Вікіпедія. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/Скінченний_автомат (дата звернення: 24.06.2025).
21. Учасники проектів Вікімедіа. Формула гаверсинуса – Вікіпедія. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/Формула_гаверсинуса (дата звернення: 24.06.2025).
22. Учасники проектів Вікімедіа. PyCharm – Вікіпедія. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/PyCharm (дата звернення: 24.06.2025).
23. Учасники проектів Вікімедіа. SQLite – Вікіпедія. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/SQLite (дата звернення: 24.06.2025).
24. Що таке модульна архітектура. *JavaRush*. URL: https://javarush.com/ua/quests/lectures/ua.questservlets.level14.lecture05 (дата звернення: 24.06.2025).
25. Яка в Telegram мова програмування?. *Lemon School*. URL: https://lemon.school/blog/yaka-u-telegram-mova-programuvanya (дата звернення: 24.06.2025).

26. Як правильно працювати інтернет-магазину зі службами доставки: Нова Пошта, Укрпошта, Meest, Justin | CityHost. *Хостинг в Украине от Cityhost – лучший хостинг сайта*. URL: https://cityhost.ua/uk/blog/obzor-sluzhb-dostavki-novaya-pochta-ukrpochta-meest-justin.html (дата звернення: 24.06.2025).
27. Aiogram. *PyPI*. URL: https://pypi.org/project/aiogram/ (дата звернення: 24.06.2025).
28. API від Укрпошти. *Ukrposhta*. URL: https://dev.ukrposhta.ua (дата звернення: 24.06.2025).
29. APScheduler. *PyPI*. URL: https://pypi.org/project/APScheduler/ (дата звернення: 24.06.2025).
30. CPU, RAM, and storage requirements | Google Distributed Cloud (software only) for VMware | Google Cloud. *Google Cloud*. URL: https://cloud.google.com/kubernetes-engine/distributed-cloud/vmware/docs/how-to/cpu-ram-storage (дата звернення: 24.06.2025).
31. JetBrains. Download PyCharm: The Python IDE for data science and web development by JetBrains. *JetBrains*. URL: https://www.jetbrains.com/pycharm/download/?section=windows (дата звернення: 24.06.2025).
32. Nova Poshta. *Telegram*. URL: https://t.me/novaposhtahelpbot (дата звернення: 24.06.2025).
33. Python-dotenv. *PyPI*. URL: https://pypi.org/project/python-dotenv/ (date of access: 24.06.2025).
34. Python Release Python 3.11.0. *Python.org*. URL: https://www.python.org/downloads/release/python-3110/ (дата звернення: 24.06.2025).
35. Sqlite3 DB-API 2.0 interface for SQLite databases. *Python documentation*. URL: https://docs.python.org/3/library/sqlite3.html (дата звернення: 24.06.2025).
36. System requirements for remote development | PyCharm. *PyCharm Help*. URL: https://www.jetbrains.com/help/pycharm/prerequisites.html#min_requirements (дата звернення: 24.06.2025).
37. Team S.-L. Exploring finite state machine in aiogram 3: a powerful tool for telegram bot development. *Medium*. URL: https://medium.com/sp-lutsk/exploring-finite-state-machine-in-aiogram-3-a-powerful-tool-for-telegram-bot-development-9cd2d19cfae9 (дата звернення: 24.06.2025).
38. Telegram – a new era of messaging. *Telegram*. URL: https://telegram.org (дата звернення: 24.06.2025).

39. Telegram bot API. *Telegram APIs*. URL: https://core.telegram.org/bots/api (дата звернення: 24.06.2025).
40. Telegram FAQ. *Telegram*. URL: https://telegram.org/faq#yakimi-pristroyami-ya-mozhu-koristuvatisya (date of access: 24.06.2025).
41. Welcome to GeoPy's documentation! – GeoPy 2.4.1 documentation. *Welcome to GeoPy's documentation! – GeoPy 2.4.1 documentation*. URL: https://geopy.readthedocs.io/en/stable/ (дата звернення: 24.06.2025).

ЛІСТИНГ КОДУ

Код файлу main.py

```
import asyncio
from datetime import datetime, timedelta

from aiogram import Bot, Dispatcher, types, F
from aiogram.filters import CommandStart
from aiogram.fsm.context import FSMContext
from aiogram.fsm.state import State, StatesGroup, any_state
from aiogram.types import CallbackQuery
from dotenv import load_dotenv
from apscheduler.schedulers.asyncio import AsyncIOScheduler
from geopy.distance import geodesic

from keyboards import *
from database import *
from api_clients import *
from background_tasks import check_packages_status

load_dotenv()
API_TOKEN = os.getenv("BOT_TOKEN")
logging.basicConfig(level=logging.INFO)
if not API_TOKEN: raise ValueError("Не знайдено токен бота.")


class UserDialog(StatesGroup):
    waiting_for_city_name = State()
    waiting_for_departure_city = State()
    waiting_for_arrival_city = State()
    waiting_for_weight = State()
```

```
    waiting_for_cost = State()
    waiting_for_tracking_number = State()


bot = Bot(token=API_TOKEN)
dp = Dispatcher()


def parse_schedule(schedule: dict) -> str:
    days_map = {0: "Monday", 1: "Tuesday", 2: "Wednesday", 3: "Thursday",
4: "Friday", 5: "Saturday", 6: "Sunday"}
    try:
        now = datetime.now()
        current_day_name = days_map[now.weekday()]
        today_schedule_str = schedule.get(current_day_name)
        if not today_schedule_str or ":" not in today_schedule_str:
            return "📅 Графік: вихідний"

        open_time_str, close_time_str = today_schedule_str.split('-')
        open_time = datetime.strptime(open_time_str, "%H:%M").time()
        close_time = datetime.strptime(close_time_str, "%H:%M").time()

        schedule_info = f"Сьогодні працює: {open_time_str} -
{close_time_str}"
        current_time = now.time()

        if open_time <= current_time < close_time:
            close_datetime = now.replace(hour=close_time.hour,
minute=close_time.minute, second=0, microsecond=0)
            if (close_datetime - now) <= timedelta(hours=1):
```

```
                return f"□ <b>Відчинено</b> (скоро зачиняється)\n🗓 {schedule_info}"
            else:
                return f"□ <b>Відчинено</b>\n🗓 {schedule_info}"
        else:
            return f"🔘 <b>Зачинено</b>\n🗓 {schedule_info}"
    except Exception as e:
        logging.error(f"Помилка парсингу графіка: {e}")
        return "🗓 Графік не вдалося визначити"


@dp.callback_query(F.data.startswith("save_"))
async def save_package_callback(callback: CallbackQuery):
    user_id, tn = callback.from_user.id, callback.data.split("_")[1];
    status = get_np_package_status(tn)
    if "Помилка" in status or "не знайдена" in status: await callback.answer("Неможливо зберегти посилку з помилкою.",
                                                                              show_alert=True); return
    if add_package(user_id, tn, status):
        await callback.answer("Посилку збережено!", show_alert=True); await callback.message.edit_reply_markup()
    else:
        await callback.answer("Ця посилка вже є у списку.", show_alert=True); await callback.message.edit_reply_markup()


@dp.callback_query(F.data.startswith("delete_"))
async def delete_package_callback(callback: CallbackQuery):
    user_id, tn = callback.from_user.id, callback.data.split("_")[1]
```

```
        if delete_package(user_id, tn):
            await callback.answer("Посилку видалено."); await callback.message.delete()
        else:
            await callback.answer("Помилка видалення.", show_alert=True)


    @dp.callback_query(F.data.startswith("update_"))
    async def update_package_status_callback(callback: CallbackQuery):
        tn = callback.data.split("_")[1];
        await callback.answer("Оновлюю статус...")
        status = get_np_package_status(tn)
        try:
            await callback.message.edit_text(f"Статус <code>{tn}</code>:\n<b>{status}</b>", parse_mode="HTML",
                                             reply_markup=get_package_management_kb(tn))
        except Exception as e:
            logging.error(f"Помилка оновлення: {e}"); await callback.answer("Статус не змінився.", show_alert=True)


    @dp.callback_query(F.data == "search_by_location")
    async def search_by_location_callback(callback: CallbackQuery):
        await callback.message.edit_text("Надішліть вашу геолокацію.", reply_markup=None)
        await callback.message.answer("Використовуйте кнопку нижче:", reply_markup=location_kb)
        await callback.answer()
```

```
@dp.callback_query(F.data == "search_by_city")
async def search_by_city_callback(callback: CallbackQuery, state: FSMContext):
    await callback.message.edit_text("Введіть назву населеного пункту.", reply_markup=None)
    await state.set_state(UserDialog.waiting_for_city_name)
    await callback.answer()


@dp.message(CommandStart())
async def send_welcome(message: types.Message, state: FSMContext): await state.clear(); await message.answer(
    "Привіт!\nЯ твій Поштовий Асистент.", reply_markup=main_kb)


@dp.message(F.text == "🔙 Повернутись в меню", any_state)
async def cancel_handler(message: types.Message, state: FSMContext): await state.clear(); await message.answer(
    "Дію скасовано.", reply_markup=main_kb)


@dp.message(F.text == "📦 Мої посилки")
async def show_my_packages(message: types.Message):
    packages = get_user_packages(message.from_user.id)
    if not packages: await message.answer("У вас ще немає збережених посилок."); return
    await message.answer("Ваші збережені посилки:")
    for pkg_num in packages:
        status = get_np_package_status(pkg_num)
```

```
            await message.answer(f"Статус <code>{pkg_num}</code>:\n<b>{status}</b>", parse_mode="HTML",
                                 reply_markup=get_package_management_kb(pkg_num))
            await asyncio.sleep(0.2)


        @dp.message(F.text == "📊 Відстежити посилку")
        async def start_tracking(message: types.Message, state: FSMContext): await message.answer("Надішліть трекінг-номер.",

reply_markup=cancel_kb); await state.set_state(
            UserDialog.waiting_for_tracking_number)


        @dp.message(F.text == "🏢 Знайти відділення")
        async def start_office_search(message: types.Message):
            await message.answer("Як ви бажаєте шукати відділення?", reply_markup=office_search_choice_kb)


        @dp.message(F.text == "💰 Розрахувати вартість посилки")
        async def start_cost_calculation(message: types.Message, state: FSMContext): await message.answer(
            "Введіть <b>місто відправлення</b>:", parse_mode="HTML", reply_markup=cancel_kb); await state.set_state(
            UserDialog.waiting_for_departure_city)


        @dp.message(UserDialog.waiting_for_tracking_number)
```

```
async def process_tracking_number(message: types.Message, state: FSMContext):
    tn = message.text.strip()
    if not tn.isdigit() or len(tn) not in [11, 14]:
        await message.answer(
            "<b>Неправильний формат.</b> Номер має складатися з 11 або 14 цифр. Спробуйте ще раз.",
            parse_mode="HTML"
        )
        return

    await state.clear()

    await message.answer("🔎 Шукаю інформацію...", reply_markup=main_kb)
    status = get_np_package_status(tn)

    reply_markup = None


    if "Помилка" not in status and "не знайдена" not in status:
        reply_markup = get_save_package_kb(tn)

    await message.answer(
        f"Оператор: <b>Нова Пошта</b>\n"
        f"Статус посилки <code>{tn}</code>:\n<b>{status}</b>",
        parse_mode="HTML",
        reply_markup=reply_markup
    )
```

```
@dp.message(F.location)
async def handle_location(message: types.Message, state: FSMContext):
    if await state.get_state() is not None: return
    user_coords = (message.location.latitude, message.location.longitude)
    await message.answer("🔎 Шукаю відділення в радіусі 20 км...", reply_markup=main_kb)
    warehouses = get_np_warehouses_in_radius(user_coords[0], user_coords[1])
    if not warehouses: await message.answer("Не вдалося знайти відділення поруч."); return
    for warehouse in warehouses:
        wh_coords = (float(warehouse['Latitude']), float(warehouse['Longitude']))
        distance = geodesic(user_coords, wh_coords).kilometers
        warehouse['calculated_distance'] = distance
    sorted_warehouses = sorted(warehouses, key=lambda x: x['calculated_distance'])
    await message.answer("<b>Ось 3 найближчих відділення:</b>", parse_mode="HTML")
    for w in sorted_warehouses[:3]:
        schedule_dict = w.get('Schedule', {})
        working_status = parse_schedule(schedule_dict)
        text = (f"📍 <b>{w['Description']}</b>\n"
                f"  Відстань: {w['calculated_distance']:.2f} км\n"
                f"  {working_status}")
        markup = get_map_keyboard(float(w['Latitude']), float(w['Longitude']))
        await message.answer(text, parse_mode="HTML", reply_markup=markup)
```

```
@dp.message(UserDialog.waiting_for_city_name)
async def process_city_name(message: types.Message, state: FSMContext):
    city_name = message.text
    await state.clear()
    await message.answer(f"🔎 Шукаю відділення у м. {city_name}...", reply_markup=main_kb)
    warehouses = get_np_warehouses_by_city(city_name)
    if not warehouses: await message.answer("Не вдалося знайти відділення. Перевірте назву."); return
    await message.answer(f"<b>Знайдені відділення у н.п. {city_name}:</b>", parse_mode="HTML")
    for w in warehouses:
        schedule_dict = w.get('Schedule', {})
        working_status = parse_schedule(schedule_dict)
        text = (f"📍 <b>{w['Description']}</b>\n"
                f"   {working_status}")
        lat, lon = w.get('Latitude'), w.get('Longitude')
        markup = get_map_keyboard(float(lat), float(lon)) if lat and lon else None
        await message.answer(text, parse_mode="HTML", reply_markup=markup)


@dp.message(UserDialog.waiting_for_departure_city)
async def process_departure_city(message: types.Message, state: FSMContext): await state.update_data(
    departure_city=message.text); await message.answer("Введіть <b>місто отримання</b>:",
```

```
                                        parse_mode="HTML"); await
state.set_state(
        UserDialog.waiting_for_arrival_city)


    @dp.message(UserDialog.waiting_for_arrival_city)
    async def process_arrival_city(message: types.Message, state: FSMContext):
await state.update_data(
        arrival_city=message.text); await message.answer("Вкажіть <b>вагу</b>
в кг:",
                                        parse_mode="HTML"); await
state.set_state(
        UserDialog.waiting_for_weight)


    @dp.message(UserDialog.waiting_for_weight)
    async def process_weight(message: types.Message, state: FSMContext):
        try:
            await state.update_data(weight=float(message.text.replace(',', '.')));
await message.answer(
                "Введіть <b>оголошену вартість</b> в грн:",
parse_mode="HTML"); await state.set_state(
                UserDialog.waiting_for_cost)
        except ValueError:
            await message.answer("Неправильний формат. Введіть число.")


    @dp.message(UserDialog.waiting_for_cost)
    async def process_cost_and_calculate(message: types.Message, state:
FSMContext):
```

```
            try:
                cost = float(message.text.replace(',', '.'));
                user_data = await state.get_data();
                await state.clear()
                await message.answer("⏳ Рахую вартість...", reply_markup=main_kb)
                result = calculate_np_delivery_cost(city_sender=user_data['departure_city'],
                                                    city_recipient=user_data['arrival_city'],
weight=user_data['weight'],
                                                    cost=cost)
                await message.answer(result, parse_mode="HTML")
            except ValueError:
                await message.answer("Неправильний формат. Введіть число.")


        @dp.message()
        async def handle_unknown_message(message: types.Message): await
message.answer("Невідома команда.",
                                                            reply_markup=main_kb)


        async def main():
            init_db()
            scheduler = AsyncIOScheduler(timezone="Europe/Kiev")
            scheduler.add_job(check_packages_status, 'interval', minutes=30,
args=(bot,))
            scheduler.start()
            await bot.delete_webhook(drop_pending_updates=True)
            await dp.start_polling(bot)
```

```
if __name__ == '__main__':
    try:
        asyncio.run(main())
    except KeyboardInterrupt:
        print("Бот зупинений.")
```

Код файлу api_clients.py

```
import logging
import os
import requests
from dotenv import load_dotenv

load_dotenv()
NOVA_POSHTA_API_KEY = os.getenv("NOVA_POSHTA_API_KEY")

def get_np_package_status(tracking_number: str) -> str:
    url, payload = "https://api.novaposhta.ua/v2.0/json/", {"apiKey": NOVA_POSHTA_API_KEY, "modelName": "TrackingDocument", "calledMethod": "getStatusDocuments", "methodProperties": {"Documents": [{"DocumentNumber": tracking_number}]}}
    try:
        r = requests.post(url, json=payload, timeout=5); r.raise_for_status(); d = r.json()
        if d['success']: return d['data'][0]['Status'] if d['data'] else "Інформація про посилку не знайдена."
        else: return d.get('errors', ['Невідома помилка'])[0]
    except Exception as e: logging.error(f"Помилка API НП (статус): {e}"); return "Помилка з'єднання."
```

```
    def get_np_warehouses_in_radius(lat: float, lon: float) -> list:
        url, payload = "https://api.novaposhta.ua/v2.0/json/", {"apiKey": NOVA_POSHTA_API_KEY, "modelName": "Address", "calledMethod": "getWarehouses", "methodProperties": {"Latitude": str(lat), "Longitude": str(lon), "Radius": 20}}
        try:
            r = requests.post(url, json=payload, timeout=7); r.raise_for_status(); d = r.json()
            if d['success']: return d.get('data', [])
            else: return []
        except Exception as e: logging.error(f"Помилка API НП (відділення): {e}"); return []

    def get_np_warehouses_by_city(city_name: str) -> list:
        url, payload = "https://api.novaposhta.ua/v2.0/json/", {"apiKey": NOVA_POSHTA_API_KEY, "modelName": "Address", "calledMethod": "getWarehouses", "methodProperties": {"CityName": city_name, "Limit": 15}}
        try:
            r = requests.post(url, json=payload, timeout=7); r.raise_for_status(); d = r.json()
            if d['success']: return d.get('data', [])
            else: return []
        except Exception as e: logging.error(f"Помилка API НП (відділення за містом): {e}"); return []

    def calculate_np_delivery_cost(city_sender: str, city_recipient: str, weight: float, cost: float) -> str:
        url = "https://api.novaposhta.ua/v2.0/json/"
        def get_city_ref(city_name):
```

```
            p = {"apiKey": NOVA_POSHTA_API_KEY, "modelName": "Address", "calledMethod": "getSettlements", "methodProperties": {"FindByString": city_name, "Limit": 1}}
            try:
                r = requests.post(url, json=p, timeout=5); d = r.json()
                if d['success'] and d['data']: return d['data'][0]['Ref']
            except Exception: return None
            return None
        sender_ref, recipient_ref = get_city_ref(city_sender), get_city_ref(city_recipient)
        if not sender_ref or not recipient_ref: return "Не вдалося знайти населенний пункт."
        cost_payload = {"apiKey": NOVA_POSHTA_API_KEY, "modelName": "InternetDocument", "calledMethod": "getDocumentPrice", "methodProperties": {"CitySender": sender_ref, "CityRecipient": recipient_ref, "Weight": str(weight), "ServiceType": "WarehouseWarehouse", "Cost": str(cost), "CargoType": "Parcel"}}
        try:
            r = requests.post(url, json=cost_payload, timeout=7); r.raise_for_status(); d = r.json()
            if d['success'] and d['data']: return f"Орієнтовна вартість: <b>{d['data'][0]['Cost']} грн</b>."
            else: return f"Помилка розрахунку: {d.get('errors', ['невідомо'])[0]}"
        except Exception as e: logging.error(f"Помилка API НП (вартість): {e}"); return "Помилка з'єднання."
```

Код файлу background_tasks.py

```
import asyncio
import logging
from aiogram import Bot
```

```
from database import get_all_active_packages, update_package_status
from api_clients import get_np_package_status


async def check_packages_status(bot: Bot):
    logging.info("Запуск фонової перевірки статусів...")
    packages_to_check = get_all_active_packages()

    for package in packages_to_check:
        user_id = package['user_id']
        tracking_number = package['tracking_number']
        last_known_status = package['last_status']

        current_status = get_np_package_status(tracking_number)

        if current_status != last_known_status and "Помилка" not in current_status and "не знайдена" not in current_status:
            logging.info(f"Статус для {tracking_number} змінився! Новий статус: {current_status}")
            try:
                await bot.send_message(
                    chat_id=user_id,
                    text=f"🔔 <b>Оновлення статусу!</b>\n"
                         f"Посилка <code>{tracking_number}</code>\n"
                         f"Новий статус: <b>{current_status}</b>",
                    parse_mode="HTML"
                )
                update_package_status(tracking_number, current_status)
            except Exception as e:
```

```
                logging.error(f"Не вдалося надіслати сповіщення користувачеві {user_id}: {e}")
            await asyncio.sleep(1)
        logging.info("Фонова перевірка завершена.")
```

Код файлу keyboards.py

```
from aiogram.types import ReplyKeyboardMarkup, KeyboardButton, InlineKeyboardMarkup, InlineKeyboardButton

track_button = KeyboardButton(text="📊 Відстежити посилку")
my_packages_button = KeyboardButton(text="📦 Мої посилки")
calculate_button = KeyboardButton(text="💰 Розрахувати вартість посилки")
offices_button = KeyboardButton(text="🏢 Знайти відділення")
cancel_button = KeyboardButton(text="🔙 Повернутись в меню")
location_request_button = KeyboardButton(text="📍 Надіслати мою геолокацію", request_location=True)

main_kb = ReplyKeyboardMarkup(
    keyboard=[
        [track_button, my_packages_button],
        [calculate_button, offices_button]
    ],
    resize_keyboard=True,
    one_time_keyboard=False,
    input_field_placeholder="Оберіть дію з меню..."
)

location_kb = ReplyKeyboardMarkup(
    keyboard=[
```

```
            [location_request_button],
            [cancel_button]
        ],
        resize_keyboard=True,
        one_time_keyboard=True,
        input_field_placeholder="Надішліть геолокацію або поверніться в меню..."
    )

    cancel_kb = ReplyKeyboardMarkup(
        keyboard=[[cancel_button]],
        resize_keyboard=True
    )

    def get_map_keyboard(lat: float, lon: float) -> InlineKeyboardMarkup:
        keyboard = [[
            InlineKeyboardButton(
                text="🗺️ Показати на карті",
                url=f"https://www.google.com/maps/search/?api=1&query={lat},{lon}"
            )
        ]]
        return InlineKeyboardMarkup(inline_keyboard=keyboard)

    def get_package_management_kb(tracking_number: str) -> InlineKeyboardMarkup:
        buttons = [
            [
                InlineKeyboardButton(text="🔄 Оновити статус", callback_data=f"update_{tracking_number}"),
```

```
                InlineKeyboardButton(text="❌ Видалити",
callback_data=f"delete_{tracking_number}")
            ]
        ]
        return InlineKeyboardMarkup(inline_keyboard=buttons)

    def get_save_package_kb(tracking_number: str) -> InlineKeyboardMarkup:
        buttons = [
            [InlineKeyboardButton(text="💾 Зберегти цю посилку",
callback_data=f"save_{tracking_number}")]
        ]
        return InlineKeyboardMarkup(inline_keyboard=buttons)

    office_search_choice_kb = InlineKeyboardMarkup(
        inline_keyboard=[
            [
                InlineKeyboardButton(text="📍 За моєю геолокацією",
callback_data="search_by_location"),
                InlineKeyboardButton(text="✍️ За назвою міста",
callback_data="search_by_city")
            ]
        ]
    )
```

Код файлу database.py

```
import sqlite3
import logging

DB_FILE = "postal_bot.db"
```

```
def init_db():
    try:
        conn = sqlite3.connect(DB_FILE)
        cursor = conn.cursor()
        cursor.execute("""
            CREATE TABLE IF NOT EXISTS tracked_packages (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                user_id INTEGER NOT NULL,
                tracking_number TEXT NOT NULL,
                last_status TEXT,
                is_delivered INTEGER DEFAULT 0, -- 0 = ні, 1 = так
                UNIQUE(user_id, tracking_number)
            );
        """)
        conn.commit()
        logging.info("База даних успішно ініціалізована.")
    except sqlite3.Error as e:
        logging.error(f"Помилка при ініціалізації БД: {e}")
    finally:
        if conn: conn.close()


def add_package(user_id: int, tracking_number: str, current_status: str) -> bool:
    try:
        conn = sqlite3.connect(DB_FILE)
        cursor = conn.cursor()
        cursor.execute(
            "INSERT INTO tracked_packages (user_id, tracking_number, last_status) VALUES (?, ?, ?)",
            (user_id, tracking_number, current_status)
```

```
            )
            conn.commit()
            return True
        except sqlite3.IntegrityError:
            logging.warning(f"Спроба додати дублікат: user_id={user_id}, tn={tracking_number}")
            return False
        finally:
            if conn: conn.close()

    def get_all_active_packages() -> list:
        try:
            conn = sqlite3.connect(DB_FILE)
            conn.row_factory = sqlite3.Row
            cursor = conn.cursor()
            cursor.execute("SELECT user_id, tracking_number, last_status FROM tracked_packages WHERE is_delivered = 0")
            return [dict(row) for row in cursor.fetchall()]
        finally:
            if conn: conn.close()

    def update_package_status(tracking_number: str, new_status: str):
        try:
            conn = sqlite3.connect(DB_FILE)
            cursor = conn.cursor()
            is_delivered = 1 if "Отримано" in new_status or "вручено" in new_status.lower() else 0
            cursor.execute(
                "UPDATE tracked_packages SET last_status = ?, is_delivered = ? WHERE tracking_number = ?",
```

```
                (new_status, is_delivered, tracking_number)
            )
            conn.commit()
        finally:
            if conn: conn.close()

    def get_user_packages(user_id: int) -> list:
        try:
            conn = sqlite3.connect(DB_FILE)
            cursor = conn.cursor()
            cursor.execute("SELECT tracking_number FROM tracked_packages
WHERE user_id = ?", (user_id,))
            return [item[0] for item in cursor.fetchall()]
        finally:
            if conn: conn.close()

    def delete_package(user_id: int, tracking_number: str) -> bool:
        try:
            conn = sqlite3.connect(DB_FILE)
            cursor = conn.cursor()
            cursor.execute("DELETE FROM tracked_packages WHERE user_id =
? AND tracking_number = ?", (user_id, tracking_number))
            conn.commit()
            return cursor.rowcount > 0
        finally:
            if conn: conn.close()
```