ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
МІСЬКОГО ГОСПОДАРСТВА ІМЕНІ О. М. БЕКЕТОВА

Пояснювальна записка
до кваліфікаційної роботи бакалавра

на тему: Розробка програмного застосунку для планування завдань з елементами чат-бота.

Виконав: здобувач вищої освіти 4 курсу,
групи КН 2022-1
спеціальності 122 «Комп'ютерні науки»
(шифр і назва спеціальності)

____Діана ЄМАНОВА____
(прізвище та ініціали)

Керівник: ___Анатолій ЛИТВИНОВ_
(прізвище та ініціали)

Рецензент: Володимир БРЕДІХІН__
(прізвище та ініціали)

м. Харків – 2026 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

«_23_» _червня____2026 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Єманова Діана Дмитрівна
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка програмного застосунку для планування завдань з елементами чат-бота.

керівник роботи Литвинов Анатолій Леонідович, д.т.н., професор
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від «22» травня 2026 р. № 440-03

2. Термін подання роботи здобувачем вищої освіти 16.06.2026 р.
3. Вихідні дані до роботи Об'єкт автоматизації: процес організації особистого тайм-менеджменту та контролю завдань.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Дослідження методологій особистого тайм-менеджменту та порівняльний аналіз існуючих програмних аналогів, обґрунтування та розробка локальної структури бази даних і об'єктно-орієнтованої моделі системи, програмна імплементація модулів інтерактивного календаря та діалогового помічника, конструювання односторінкового інтерфейсу користувача і супровідної документації.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
16 аркушів

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ I	Анатолій ЛИТВИНОВ, професор кафедри КН та ІТ	12.05.2026	17.05.2026
Розділ II	Анатолій ЛИТВИНОВ, професор кафедри КН та ІТ	17.05.2026	20.05.2026
Розділ III	Анатолій ЛИТВИНОВ, професор кафедри КН та ІТ	20.05.2025	22.05.2026
Розділ IV	Вікторія МАЛИШЕВА, к. т., н., доцент кафедри ОП та БЖ	29.05.2026	15.06.2026

7. Дата видачі завдання 12.05.2026

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Вибір тема дипломної роботи	12.05.2025	Виконано
2	Затвердження тем, наукових керівників та календарного плану підготовки кваліфікованої роботи	14.05.2025	Виконано
3	Написання I розділу	18.05.2026	Виконано
4	Написання II розділу	21.05.2026	Виконано
5	Написання III розділу	06.06.2026	Виконано
6	Написання IV розділу	10.06.2026	Виконано
7	Подання дипломної роботи керівнику	11.06.2026	Виконано
8	Робота по усуненню помилок, коригування роботи з урахуванням зауважень керівника, уточнення і доповнення практичного матеріалу.	13.06.2026	Виконано
9	Подання допрацьованого варіанту роботи керівнику	15.06.2026	Виконано
10	Захист матеріалів дипломної роботи на засіданні кафедри	18.06.2026	Виконано
11	Офіційний захист матеріалів дипломної роботи на засіданні екзаменаційної комісії	24.06.2026	Виконано

Здобувач вищої освіти _____ Діана ЄМАНОВА
(підпис) (прізвище та ініціали)

Керівник роботи _____ Анатолій ЛИТВИНОВ
(підпис) (прізвище та ініціали)"

АНОТАЦІЇ

Структура та обсяг роботи. Пояснювальна записка кваліфікаційної роботи бакалавра здобувача вищої освіти групи КН 2022-1 спеціальності 122 «Комп’ютерні науки» Ємановоїї Діани Дмитрівни на тему «Розробка десктопного застосунку тайм-менеджменту з інтегрованим віртуальним асистентом» складається з 4 розділів, містить 96 сторінок, 19 рисунків, 3 таблиць та 18 літературних джерел.

Метою роботи є розробка автономного десктопного планувальника завдань із гібридним розмовним інтерфейсом для оптимізації процесів особистого тайм-менеджменту та зниження когнітивного навантаження користувача.

Об’єктом дослідження є процес організації та управління особистим часом, довгостроковими завданнями і нагадуваннями в ізольованому локальному інформаційному середовищі.

Предметом дослідження є архітектурний підхід Offline-First, інструментальні засоби графічного фреймворку PyQt6, технології управління реляційними базами даних SQLite та алгоритми функціонування кінцевих автоматів для віртуальних помічників.

Основне завдання це реалізувати застосунок «Астра» з функціоналом гібридного чат-асистента, інтерактивного календаря, системою управління планами (з підтримкою прикріплення медіафайлів), механізмом м'якого видалення (Soft Delete) у логічний кошик та можливістю гарячої заміни візуальних тем оформлення.

Результатом роботи є готовий до використання кросплатформний десктопний застосунок «Астра», що поєднує класичні візуальні інтерфейси з інтелектуальним розмовним ботом.

Ключові слова: ДЕСКТОПНИЙ ЗАСТОСУНОК, ТАЙМ-МЕНЕДЖМЕНТ, PYTHON, PYQT6, SQLITE, ВІРТУАЛЬНИЙ АСИСТЕНТ, OFFLINE-FIRST, SINGLE-PAGE APPLICATION.

ANNOTATION

Structure and volume of the work. The explanatory note of the bachelor's qualification thesis of the higher education student of group KN 2022-1 in specialty 122 "Computer Science", Diana Dmytrivna Yemanova, on the topic "Development of a desktop time-management application with an integrated virtual assistant" consists of 4 chapters and contains 96 pages, 18 figures, 3 tables, and 19 references.

The aim of the project is to develop an autonomous desktop task planner with a hybrid conversational interface to optimize personal time management processes and reduce the user's cognitive load.

The object of study is the process of organizing and managing personal time, long-term tasks, and reminders within an isolated local information environment.

The subject of study is the Offline-First architectural approach, PyQt6 graphical framework tools, SQLite relational database management technologies, and finite-state machine algorithms for virtual assistants.

The main task is to implement the "Astra" application featuring a hybrid chat assistant, an interactive calendar, a plan management system (supporting media file attachments), a soft delete mechanism to a logical trash bin, and the ability to hot-swap visual themes.

The result of the work is a ready-to-use cross-platform desktop application "Astra" that combines classic visual interfaces with an intelligent conversational bot.

Keywords: DESKTOP APPLICATION, TIME MANAGEMENT, PYTHON, PYQT6, SQLITE, VIRTUAL ASSISTANT, OFFLINE-FIRST, SINGLE-PAGE APPLICATION.

ЗМІСТ

ВСТУП

Сьогодні інструменти тайм-менеджменту еволюціонують у бік інтелектуальних помічників. Незважаючи на популярність мобільних платформ, персональний комп'ютер залишається головним робочим середовищем для навчання та професійної діяльності. Через багатозадачність користувачам дедалі складніше тримати в голові та фіксувати всі ідеї. Виникає потреба в програмах, які не просто зберігають списки у вигляді сухих таблиць, а пропонують природну взаємодію, наближену до живого спілкування.

Аналіз існуючих планувальників виявляє їхню полярність: це або складні програми з перевантаженим графічним інтерфейсом, або мінімалістичні текстові консолі без підтримки візуального календаря та медіафайлів. Такий підхід ускладнює процес створення нотаток і нагадувань, що зрештою знижує продуктивність користувача замість того, щоб підвищувати її. Розв'язати цю проблему дозволяє нативна розробка десктопних застосунків, які працюють локально та не залежать від іцнтернет-з’єднання. Саме тому актуальним є створення безкоштовної програми категорії Productivity — планувальника, який поєднує класичний календар та списки завдань із вбудованим інтерактивним чат-ботом «Астра».

Метою роботи є проєктування та розробка десктопного застосунку для планування завдань, який швидко обробляє текстові запити користувача, керує медіафайлами та локально зберігає дані без прив'язки до сторонніх хмарних сервісів.

Об’єктом дослідження виступає процес управління часом та планування завдань за допомогою десктопного програмного забезпечення. Предметом дослідження є алгоритми обробки природно-мовних команд чат-бота, методи побудови графічних інтерфейсів та способи локального збереження структурованих даних.

Для досягнення поставленої мети в роботі застосовано системний аналіз предметної області та порівняльний аналіз аналогів. Архітектуру програми побудовано за допомогою методів об'єктно-орієнтованого проєктування. Для перевірки технічних рішень використано експериментальне тестування інтерфейсу та взаємодії з локальною базою даних.

Теоретична цінність дослідження полягає в систематизації підходів до розробки гібридних систем, що поєднують чат-інтерфейс із класичним графічним інтерфейсом користувача, та обґрунтуванні вибору інструментів для роботи з локальними реляційними базами даних.

Практична значущість роботи підтверджується створенням готового до використання, оптимізованого застосунку на базі фреймворку PyQt6. Програма надає користувачам можливість планувати справи через спілкування з віртуальним асистентом, гнучко форматувати завдання, прикріплювати зображення, контролювати дедлайни в календарі та надійно зберігати інформацію виключно на своєму локальному носії.

РОЗДІЛ 1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Опис предметного середовища

Індустрія настільних додатків, призначених для організації особистого часу та ведення записів, наразі переповнена різноманітними готовими сервісами, які здебільшого орієнтовані на хмарну синхронізацію. Попри такий широкий асортимент, люди регулярно відчувають труднощі під час пошуку оптимального софту. Їм потрібен такий цифровий інструмент, який би надійно захищав приватну інформацію завдяки функціонуванню виключно в офлайн-режимі, а також давав змогу блискавично вводити нові дані, маючи при цьому сучасний та привабливий дизайн. Левова частка загальнодоступних безплатних органайзерів змушує людину методично заповнювати жорстко структуровані форми чи незручні таблиці. Через це для банальної генерації звичайнісінького сповіщення доводиться здійснювати безліч непотрібних натискань мишею та блукати складними лабіринтами меню.

Подібний стан речей перетворює процес упорядкування власних справ на виснажливу рутину, через що людина швидко втрачає будь-яке бажання регулярно оновлювати свої переліки завдань. Створення настільної програми під назвою «Астра Асистент» має на меті подолати ці недоліки завдяки впровадженню гібридної концепції. Цей софт гармонійно об'єднує глибоко оптимізований рушій локальної бази даних із діалоговим механізмом взаємодії. Тобто генерація, коригування та відмітка про успішне завершення певних доручень відбуваються шляхом невимушеного текстового спілкування з віртуальним помічником.

Початковим кроком інженерного конструювання такого продукту стало визначення оптимального набору технологій. Враховуючи високі критерії щодо сумісності з різними операційними системами та необхідність швидкої побудови візуальних оболонок, застосування мови Python разом із бібліотекою PyQt6 виявилося найбільш виграшним варіантом. Згаданий фреймворк

відкриває прямий шлях до базових елементів ОС, що дає змогу імплементувати передові графічні ефекти. Наприклад, використання параметричних тіней (через клас QGraphicsDropShadowEffect) та плавних анімацій допомагає досягти візуальної естетики напрямку glassmorphism, причому апаратна частина комп'ютера не зазнає надмірного перевантаження.

Під час розробки структурної побудови додатка окремий акцент робився на ключових критеріях якості софту. Головними орієнтирами тут виступають швидкодія, стабільність функціонування та захищеність збережених відомостей. Швидкодія відповідає за миттєвий відгук інтерфейсу на написані команди в чаті, а також за відсутність затримок під час відмальовування об'ємних графічних файлів, що прикріплюються до карток. Своєю чергою, стабільність гарантує безперебійну роботу застосунку без раптових вильотів, підкріплену абсолютною незалежністю від наявності з'єднання зі світовою павутиною.

Актуальні напрямки дизайну цифрових продуктів диктують правило чіткого розмежування екрана на функціональні сектори. У створеному застосунку цей принцип втілено за допомогою компонента маршрутизації QStackedWidget. Він гарантує миттєву зміну активних вкладок — від діалогового вікна Астри до інтерактивної сітки календаря, архівного переліку або екрана видалених елементів. Принципово важливо, що під час таких переходів загальний стан програми залишається незмінним і не потребує повного перезавантаження сторінок.

Критично вагомим аспектом проєктування є підбір механізму для накопичення та зберігання інформації. Для настільних рішень, написаних на Python, загальноприйнятим стандартом виступає застосування інтегрованої реляційної системи SQLite. Її впровадження дозволяє повністю відповідати строгим стандартам ACID під час проведення транзакцій. Це є запорукою того, що всі тексти доручень, часові координати, нагадування та шляхи до картинок

будуть надійно записані на жорсткий диск без жодної потреби встановлювати чи налаштовувати додатковий локальний сервер.

Специфічною архітектурною знахідкою стала імплементація алгоритму безпечного стирання (відомого як soft delete). Замість того щоб безповоротно знищувати інформаційні блоки з таблиць, система просто модифікує їхній внутрішній статус на deleted, паралельно переміщуючи такі об'єкти до віртуального кошика. Подібна стратегія чудово оберігає оператора від випадкового видалення цінних нотаток, надаючи зручну можливість повернути будь-який запис до життя буквально одним кліком миші. Для сучасних комплексів управління контентом такий підхід є життєво необхідним.

1.1.1 Опис процесу діяльності

Алгоритм функціонування десктопного застосунку базується на концепції діалогової взаємодії, де життєвий цикл планування ініціюється та супроводжується віртуальним асистентом. На відміну від класичних планувальників, які вимагають ручного заповнення багатополівних форм, процес діяльності в системі орієнтований на природний покроковий діалог з Астрою.

Діяльність розпочинається з моменту введення користувачем початкового запиту в текстове поле чату. Астра розпізнає ключові тригери (наприклад, слова «план» або «нагадування») та запускає відповідну гілку ітеративного збору даних. Процес формування нагадувань через Астру поділяється на кілька дискретних кроків, кожен з яких супроводжується уточнюючим питанням асистента: запит назви, визначення текстового контенту, парсинг гнучкої дати та часу (з перевіркою формату), а також

можливість прикріплення медіафайлу (зображення) через виклик системного діалогу QFileDialog.

Після успішного збору всіх атрибутів, Астра автоматично формує об'єкт даних і передає його до модуля database_manager для валідації на унікальність та збереження в SQLite. Для забезпечення зручності управління поточними завданнями безпосередньо з чату реалізовано механізм контекстної пам'яті. При запиті списку справ на поточний день, система генерує нумерований перелік, який зберігається в короткостроковій пам'яті. Це дозволяє користувачу використовувати швидкі команди (наприклад, «план 1 виконано» або «план 2 видалити»), після чого Астра ідентифікує цільовий об'єкт, змінює його статус у базі даних і фоново оновлює всі візуальні компоненти графічного інтерфейсу (вкладки календарю та списків) без необхідності перезавантаження програми.

1.1.2 Опис функціональної моделі

Архітектура взаємодії користувача із застосунком формалізується через функціональну модель, що охоплює комплекс прецедентів використання. Оскільки система спроєктована за принципом Offline-First та гарантує локальне зберігання даних, у ній виділено єдину роль — Локальний користувач (Local User), який має повний адміністративний доступ до всіх функцій без необхідності мережевої авторизації.

Модель використання розгалужується на два основні напрямки: розмовний (Conversational UI) та графічний (GUI).

Прецедент «Взаємодія з Астрою для нагадувань та планів». Користувач здійснює введення текстових запитів. Система забезпечує розпізнавання контексту: виведення списків на конкретну дату, обробку команд завершення завдань або створення нових записів.

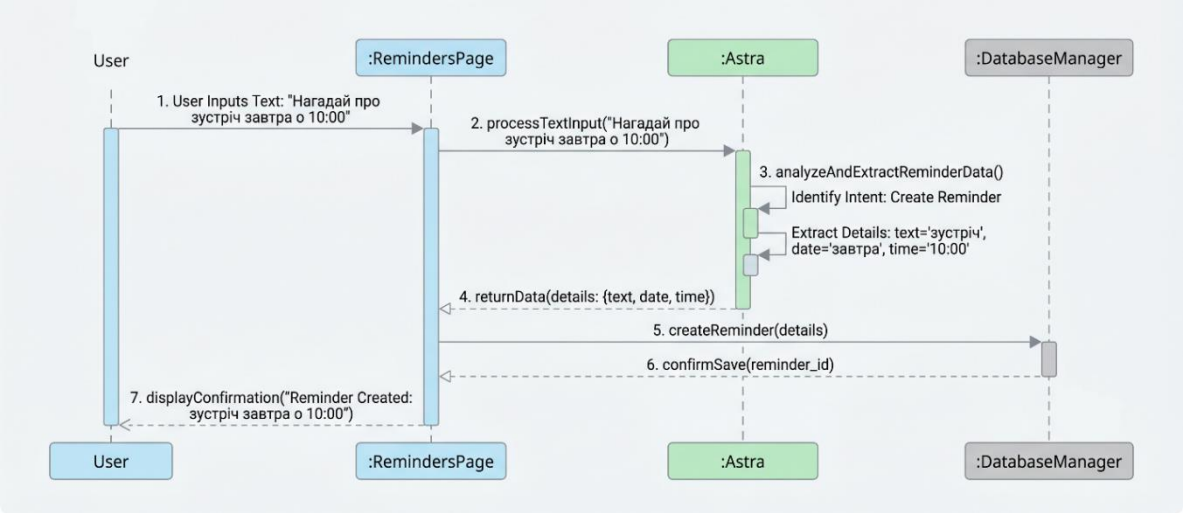

Рисунок 1.1 – UML-діаграма «взаємодія з ботом для нагадувань та планів»

Прецедент «Робота з інтерактивним календарем». Також система автоматично маркує дати, що містять активні завдання. Користувач при виборі дати отримує деталізований перелік планів у вигляді візуальних карток зі стилізованими тінями (елементи glassmorphism-дизайну) та можливістю попереднього перегляду прикріплених зображень.

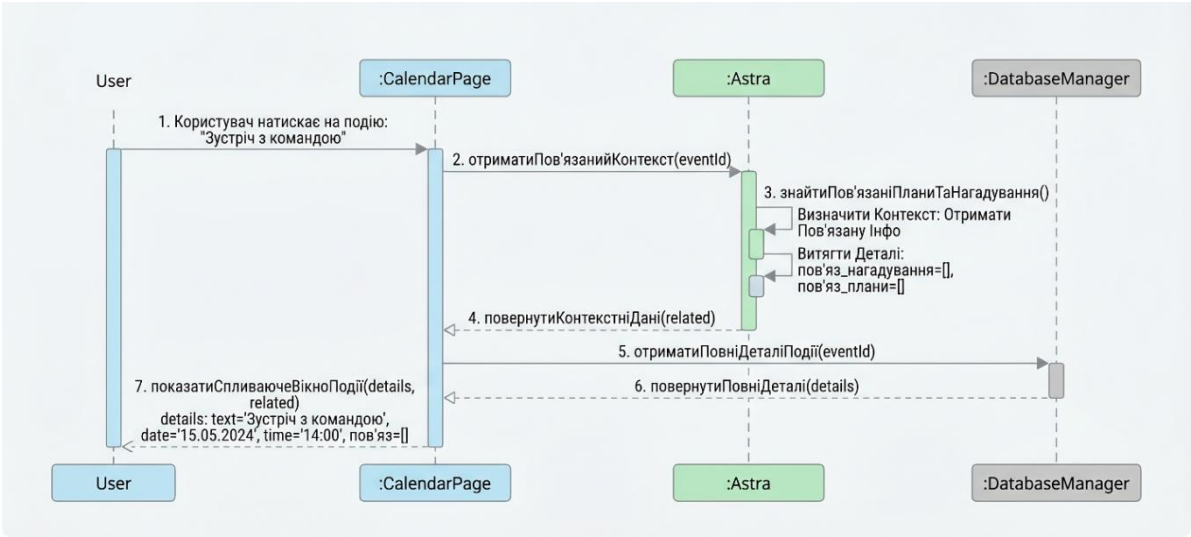

Рисунок 1.2 – UML - діаграма «Робота з інтерактивним календарем»

Прецедент «Управління архівом та кошиком». Реалізовано концепцію «м'якого видалення». Користувач може переглядати виконані плани у відповідній вкладці, а видалені елементи переміщуються до кошика з можливістю подальшого відновлення (Restore) або остаточного знищення (Hard Delete).

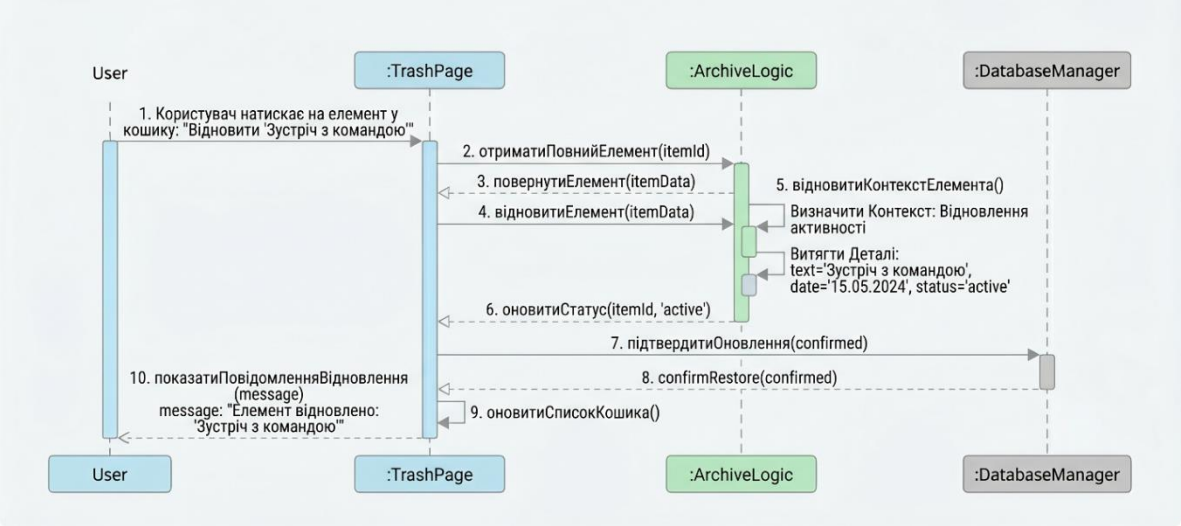

Рисунок 1.3 - UML – діаграма «Управління архівом та кошиком»

Прецедент «Персоналізація робочого простору». Система дозволяє користувачу динамічно змінювати візуальне оформлення (перемикання між світлою та темною темами з миттєвим перемальовуванням каскадних таблиць стилів QSS) та керувати відображенням інформаційних підказок у чаті.

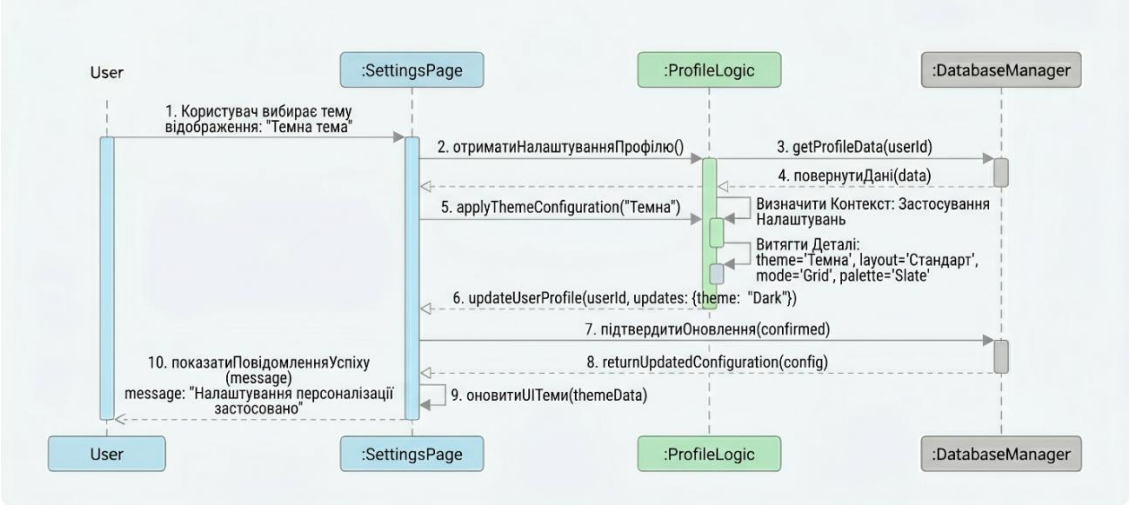

Рисунок 1.4 - UML - діаграма «Персоналізація робочого простору»

1.2 Огляд існуючих аналогів

Щоб довести інноваційність створеного продукту та чітко окреслити його переваги на тлі конкурентів, у межах дослідження проаналізовано три найвідоміші десктопні платформи для організації справ: Todoist, Microsoft To Do та TickTick. Саме ці сервіси сьогодні найчастіше обирають для управління часом на персональних комп'ютерах.

Програма Todoist заслужено займає провідні позиції на ринку, оскільки має чудово налаштований алгоритм розпізнавання природної мови під час друкування дат. Однак під час роботи на десктопі цей інструмент виявляє серйозні вади. Головна з них — абсолютна прив'язка до інтернет-з'єднання та хмарних серверів. До того ж базова версія містить жорсткі рамки. Якщо користувач не оформить платну підписку Premium, він не зможе вільно прикріплювати графічні матеріали чи налаштовувати спеціалізовані сповіщення, що суттєво звужує корисність софту.

Наступний кандидат, Microsoft To Do, виділяється тісним зв'язком із внутрішнім середовищем операційної системи Windows і надає весь свій інструментарій абсолютно безкоштовно. Програма має лаконічний вигляд, проте страждає від нестачі гнучкості в налаштуваннях дизайну, пропонуючи лише бідний набір кольорових тем. Найбільшим мінусом є відсутність розмовного формату взаємодії. Формування нових цілей здійснюється через звичайні нерухомі поля для тексту, що робить процес надто консервативним і позбавляє його будь-якої інтерактивності.

Третій аналог, TickTick, намагається об'єднати звичайний перелік доручень із повноцінною календарною сіткою. Але тут виникає перша перешкода: настільна модифікація змушує обов'язково створювати обліковий запис. Більше того, найкорисніша опція перегляду запланованих подій у

форматі матриці заблокована у вільному доступі. Візуальна оболонка цього додатка перенасичена специфічними інструментами, такими як матриця Ейзенхауера та таймери концентрації Pomodoro. Для людини, яка просто бажає впорядкувати повсякденні справи, такий надлишковий функціонал стає лише тягарем і відвертає увагу від головного.

Таблиця 1.1 - Порівняльний аналіз десктопних застосунків для планування

Параметр порівняння	Todoist	Microsoft To Do	TickTick	Astra Assistent
Формат інтерфейсу	Стандартний (GUI)	Стандартний (GUI)	Перевантажений (GUI)	Гібридний (Чатбот + GUI)
Локальне збереження (Offline)	Відсутнє (тільки кеш)	Частково	Відсутнє	Повністю локальна SQLite БД
Можливість кріплення медіа	Лише у платній версії	Присутня	Лише у платній версії	Вбудована та безкоштовна
Діалогове створення нагадувань	Відсутнє	Відсутнє	Відсутнє	В Наявне (покрокове опитування)

Абонентська плата / Підписки	Наявні обмеження (Premium)	Безкоштовно	Наявні обмеження (Premium)	Повністю безкоштовно
Навігація та дизайн	Класичні меню	Fluent Design	Класичні меню	Адаптивні тіні, QStacked Widget

Аналіз таблиці 1.1 демонструє, що жоден із популярних аналогів не пропонує гібридного текстового інтерфейсу в поєднанні з повною автономністю від серверів та безкоштовним доступом до інструментів інтерактивного календаря. Це підтверджує актуальність нашої розробки.

1.3 Постановка задачі

Головне завдання цього кваліфікаційного проєкту полягає у всебічному науковому обґрунтуванні, створенні алгоритмічного фундаменту та безпосередньому програмуванні кросплатформного настільного комплексу, призначеного для індивідуального планування та організації робочого часу. Створюваний програмний продукт покликаний забезпечити якісно новий формат взаємодії з користувачем завдяки інтеграції текстового діалогового вікна у вигляді віртуального помічника «Астра» з традиційними графічними елементами керування. До цього інтерактивного комп'ютерного співрозмовника висувається ціла низка строгих технічних вимог, серед яких безперебійна обробка інформаційних потоків, чітке внутрішнє сортування створюваних сутностей на довгострокові цілі та поточні сповіщення, а також

формування автономного модуля перевірки медіаданих, що дає змогу долучати графічні зображення до карток конкретних доручень.

У процесі архітектурного конструювання внутрішніх компонентів софту критично важливо врахувати комплекс нефункціональних параметрів, апаратних обмежень та критеріїв стабільності, що висуваються до сучасних прикладних програм настільного типу. Застосунок має функціонувати в межах концепції повної автономності (Offline-First), що передбачає стовідсоткову ізоляцію бізнес-логіки та механізмів інтерпретації текстових команд від глобальної мережі й віддалених серверів для гарантування максимальної конфіденційності особистих відомостей. Нативна розробка за допомогою інструментів мови Python та бібліотеки PyQt6 покликана забезпечити високу адаптивність до різних операційних систем при мінімальному споживанні оперативної пам'яті комп'ютера, що дозволяє повністю відмовитися від ресурсномістких веб-рушіїв. Серйозним викликом на цьому етапі є оптимізація графічного конвеєра, яка потребує раціонального розподілу обчислювальної потужності під час роботи з великими бінарними файлами та їхнього коректного масштабування через алгоритми згладжування, щоб не допустити найменших затримок чи зависань візуальної оболонки додатка.

Окремим технологічним чинником життєздатності виступає формування стійкого шару збереження даних, що здатний ефективно перехоплювати логічні збої та конфлікти на рівні файлової системи. Алгоритм взаємодії з локальним файлом бази даних проєктується з урахуванням автоматичного захисту від дублювання унікальних ключів, що дозволяє коректно обробляти помилки цілісності безпосередньо в СКБД та блокувати екстрене закриття софту в разі виникнення колізій. Окрім того, внутрішній модуль управління інформацією містить механізми валідації локальних адрес прикріплених малюнків, що надає можливість контролювати фізичну присутність об'єктів на накопичувачі та запобігати критичним помилкам

відображення, якщо матеріали були випадково переміщені чи стерті іншими програмами.

Задля успішного втілення поставленої інженерної мети необхідно послідовно виконати низку взаємопов'язаних науково-практичних завдань, що складають загальну структуру дослідження. Передусім потрібно спроєктувати реляційну схему локального сховища на основі СКБД SQLite, яка підтримує автоматичну перевірку структури таблиць, зворотну сумісність модифікацій та гнучке розширення полів за допомогою каскадних запитів без загрози втрати вже існуючих записів користувача. Наступним кроком стає математичний опис та програмне впровадження моделі скінченного автомата для управління внутрішніми станами діалогу асистента, який покроково збирає опис завдань, розпізнає часові координати та здійснює контекстне опрацювання відповідей на основі швидкої навігації за допомогою тимчасової індексації виведених списків.

Поряд із цим постає потреба в розробці модульної багатосторінкової структури графічної оболонки із залученням інструментів внутрішньої маршрутизації для надійної ізоляції та взаємозв'язку окремих екранів застосунку, куди входять розумний чат, візуальна стрічка завдань, картки сповіщень, календарна сітка та панель загальних конфігурацій. На завершальному етапі необхідно реалізувати двофазну схему видалення інформаційних блоків на базі патерну м'якої утилізації, що забезпечує переведення непотрібних елементів у тимчасовий статус із перенесенням до спеціального модуля кошика, зберігаючи шанс на їхнє швидке відновлення або здійснення операції безповоротного фізичного знищення з очищенням пам'яті пристрою.

Підсумковим результатом реалізації всіх зазначених кроків має стати оптимізований, масштабований та стійкий до відмов настільний продукт із чіткою сепарацією рівнів бізнес-логіки та представлення інформації. Програма надасть кінцевому користувачеві повністю приватне, ергономічне

та захищене середовище для ефективного управління часом, яке гармонійно поєднує в собі гнучкість розмовних технологій віртуального помічника із наочністю та зручністю класичних інструментів візуалізації.

Висновки до розділу

Перша частина цієї кваліфікаційної роботи присвячена детальному аналізу індустрії настільних програм, призначених для індивідуального управління часом та інтелектуального розподілу завдань. Проведене дослідження яскраво підтвердило, що аудиторія подібних цифрових продуктів зараз має серйозний дефіцит рішень, які спроможні надійно забезпечити стовідсоткове збереження таємниці особистих відомостей шляхом суто локальної роботи на комп'ютері, але водночас пропонують нестандартні інструменти комунікації. Зокрема, йдеться про інтеграцію комбінованих розмовних оболонок, які мають замінити собою традиційні, надто складні для сприйняття візуальні форми та громіздкі табличні структури.

Спираючись на ретельний розбір інженерних критеріїв щодо темпу відгуку інтерфейсу, стабільного функціонування в різних операційних системах та зовнішньої привабливості, у роботі доведено раціональність вибору базового інструментарію розробки, що складається з мови Python та можливості графічної бібліотеки PyQt6. Головним стовпом захищеності та стабільності створюваного комплексу визначено інтегровану реляційну базу SQLite. Внутрішня будова цього сховища ідеально втілює філософію повної незалежності від мережевих з'єднань Offline-First, а також містить сучасний алгоритм безпечного утилізування записів, який надійно захищає власника пристрою від раптової і незворотної втрати цінної інформації.

Під час теоретичного та прикладного опрацювання теми детально розписано ключові алгоритми життєдіяльності додатку, де центральним

керівним елементом виступає циклічний обмін репліками із цифровим помічником «Астра». Сформована схема варіантів використання повністю охоплює всі основні операції, які може здійснювати людина всередині локальної системи. Сюди відноситься як створення стратегічних цілей чи оперативних сповіщень через послідовні запитання асистента, так і робота з інтегрованими графічними матеріалами разом із легкою зміною активних вікон за допомогою внутрішнього модуля перемикання сторінок.

Здійснене зіставлення параметрів найпопулярніших ринкових конкурентів, зокрема Todoist, Microsoft To Do та TickTick, чітко підтвердило присутність у них вагомих структурних та ідеологічних мінусів. Серед першочергових недоліків виділено тотальну прив'язку до серверної синхронізації, вимогу плати за користування елементарними опціями, наприклад за додавання картинок, та повний брак вбудованих діалогових помічників, що суттєво уповільнює повсякденну взаємодію із софтом. Знайдені слабкі місця наявних програм дозволили детально обґрунтувати унікальні фішки власного програмного продукту та сформувати фінальну інженерну постановку задачі, яка послужить надійною базою для подальшого математичного моделювання та безпосереднього написання вихідного коду додатка.

РОЗДІЛ 2 ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Аналіз предметної області

Нинішній простір цифрових інструментів для індивідуальної продуктивності та розподілу справ зазнає суттєвих змін через лавиноподібне збільшення обсягів повсякденних відомостей і потребу в моментальному записі поточних ідей. Звичні настільні органайзери, які вимагають монотонного заповнення численних граф у таблицях, дедалі сильніше демонструють свою непридатність, оскільки їхні екрани надто захаращені, а людина змушена витрачати купу дорогоцінного часу на технічне введення тексту. Створення зручних умов для ментальної роботи користувача потребує інтеграції гнучких автоматизованих комплексів, що спроможні максимально спростити комунікацію між оператором та комп'ютером. З огляду на це постає потреба в глибокому вивченні специфіки розмовних настільних помічників, які гармонійно поєднують у собі опції безпосереднього текстового обміну репліками та традиційного графічного відображення часових графіків.

Дослідження внутрішніх компонентів аналізованого середовища дає змогу виокремити фундаментальні інформаційні елементи й окреслити специфіку їхніх смислових зв'язків. Головною одиницею додатка виступає надіслана користувачем директива чи конкретна справа, яка, зважаючи на свої особливості та часові межі, поділяється на дві ключові категорії. До початкового блоку належать стратегічні наміри, котрі мають загальний зміст і слугують для глобального планування графіка без жорсткого закріплення за конкретними хвилинами чи годинами. Другу категорію утворюють миттєві оперативні сповіщення, безпосередньо прив'язані до чітких термінів виконання в календарі, які потребують суворої фіксації моменту спрацьовування. Важливим супроводжуючим елементом для обох типів справ

є медіавміст у вигляді збережених на пристрої малюнків, що суттєво підвищує наочність кожної окремої картки та полегшує візуальне сприйняття впорядкованого контенту.

Вагомим нюансом побудови алгоритмів в органайзерах є чіткий опис життєвого шляху доручень та контроль за зміною їхніх поточних параметрів. У межах цього простору будь-яка справа рухається по черзі через серію системних міток, котрі безпосередньо визначають її появу на різних робочих екранах додатка. Статус активного елемента вказує на поточні незавершені справи, за якими можна стежити через сітку календаря або шляхом надсилання повідомлень асистенту. Перенесення запису до категорії завершених дозволяє формувати хронологічний звіт про особисту ефективність. Окрему увагу під час вивчення середовища приділено механізму очищення пам'яті, який працює за принципом двофазного видалення. Замість миттєвого та безповоротного стирання відомостей із накопичувача, програма трансформує статус елемента на деструктивний, завантажуючи його до відокремленого простору кошика. Це повністю усуває загрозу випадкових помилкових дій людини та гарантує збереження особистої бази знань.

Формування загальної теоретичної схеми предметного простору дає можливість точно визначити рамки інженерного конструювання настільної програми та прописати засади збереження внутрішньої несуперечності відомостей на кожному рівні архітектури. Зв'язок між людиною, центральним модулем розмовного помічника, сховищем та візуальними екранами будується на засадах миттєвого відгуку на будь-які трансформації параметрів. Кожне набране повідомлення, яке успішно розпізнає інтерактивний асистент, перетворюється на відповідну зміну характеристик у таблицях реляційної бази даних, що одразу ж відображається на графічних елементах списку справ та часової сітки. Такий комплексний метод гарантує узгодженість усього інформаційного наповнення, а також створює надійну аналітичну платформу

для подальшої математичної та програмної розробки внутрішніх алгоритмів управління створеним комплексом.

2.1.1 Вхідні дані

Вхідні дані розроблюваного програмного комплексу формуються за рахунок інтерактивної взаємодії користувача з текстовим інтерфейсом асистента та графічними елементами управління. На відміну від класичних форм реєстрації, первинний потік інформації генерується ітеративно під час діалогу, після чого трансформується у суворо типізовані структури для передачі до локальної бази даних SQLite. Увесь масив вхідних даних можна класифікувати за їхнім функціональним призначенням, виділяючи семантичні, конфігураційні та системні атрибути.

Основною категорією є текстові дані, що визначають семантику та зміст планування. До них належить параметр унікальної назви завдання (атрибут name строкового типу TEXT), який вводиться користувачем на першому етапі опитування та проходить сувору валідацію на рівні бази даних для уникнення дублікатів. Змістовне наповнення задачі передається через розгорнутий текстовий опис (атрибут content типу TEXT), що дозволяє фіксувати детальну інформацію про завдання без жорстких обмежень за довжиною символів. Хронологічний контекст забезпечується вхідним параметром дати та часу (атрибут reminder_date типу TEXT), який парситься з текстового рядка користувача у форматах «ДД.ММ.РРРР» або «ГГ:ХХ» і виступає ключовим критерієм для позиціонування карток в інтерфейсі інтерактивного календаря.

Для забезпечення програмної класифікації та коректної візуалізації завдань система приймає додаткові конфігураційні вхідні дані. Категорійна належність запису визначається атрибутом типу елемента (атрибут item_type строкового типу), який генерується програмним ядром на основі обраного користувачем сценарію діалогу і набуває статичних значень plan для

безстрокових завдань або reminder для подій з точним дедлайном. У разі потреби розширення інформативності картки, система приймає абсолютний шлях до локального графічного файлу (атрибут image_path типу TEXT). Цей строковий параметр передається з нативного діалогового вікна операційної системи (QFileDialog) та вказує на фізичне розташування зображення у форматах PNG, JPG або BMP на диску пристрою.

На системному рівні до вхідних даних, що забезпечують цілісність бізнес-логіки та навігації, відносяться маркери життєвого циклу об'єкта та глобальні параметри середовища. Маркер стану (атрибут status строкового типу) ініціюється як active під час створення нового запису і може змінюватися на completed або deleted залежно від вхідних керуючих команд користувача, забезпечуючи роботу архітектурного патерну м'якого видалення (soft-delete). Додатково, на етапі запуску застосунку, відбувається зчитування вхідних налаштувань середовища через механізм збереження конфігурацій QSettings, який постачає строковий параметр поточної візуальної теми (light або dark) для ініціалізації відповідних каскадних таблиць стилів та коректного рендерингу графічної оболонки.

2.1.2 Вихідні дані

Вихідні дані проєктувального десктопного комплексу формуються в результаті внутрішньої аналітичної, логічної та алгоритмічної обробки первинних інформаційних потоків ядром програми та персистентним шаром реляційної системи керування базами даних. Головне архітектурне призначення вихідних даних полягає в забезпеченні інтерактивного зворотного зв'язку, інформативності та високої ергономічності інтерфейсу, що дозволяє кінцевому користувачу безперешкодно контролювати стан поточних завдань і взаємодіяти з чат-асистентом. Результуючий інформаційний потік

архітектурно розділяється на кілька взаємопов'язаних рівнів, охоплюючи типізовані структуровані вибірки з локального сховища, динамічні візуальні об'єкти графічної оболонки та системні діалогові й модальні компоненти зворотного зв'язку.

На рівні взаємодії з персистентним шаром сховища вихідні дані представлені дескрипторами та послідовностями об'єктів, які генеруються модулем управління базою даних у відповідь на внутрішні SQL-запити фільтрації та вибірки. Головним вихідним результатом цього рівня є масиви кортежів, що видобуваються методами зчитування активних, виконаних або видалених елементів. Кожен інформаційний рядок у такому масиві містить суворо упорядкований набір вихідних атрибутів: унікальний числовий ідентифікатор первинного ключа id, строкову константу імені завдання name, розгорнутий змістовний текстовий опис content, хронологічну мітку дедлайну reminder_date, категорійний маркер типу item_type та текстову адресу розташування графічного документа image_path. Додатковим пластом вихідних даних дата-рівня є логічні прапорці успішності транзакцій булевого типу, які сигналізують керуючому коду про результати модифікації персистентного стану системи та дозволяють ідентифікувати виняткові ситуації колізій імен на етапі збереження.

На рівні функціонування текстового інтерфейсу вихідний потік трансформується в послідовні репліки чат-асистента Астри, що виводяться у вікні текстового браузера. Ці вихідні дані формуються динамічно залежно від поточного стану кінцевого автомата та підлягають кастомізації за допомогою вбудованих тегів мови розмітки HTML. Програма генерує текстові рядки, які містять стилізоване виділення назв завдань жирним шрифтом, інструктивні повідомлення щодо координації подальших кроків діалогу та системні попередження у разі виникнення помилок валідації. Одночасно з текстовим виведенням, вихідними даними логічного модуля чату є реактивні зміни станів керуючих елементів GUI, що виражається у динамічному блокуванні полів

введення за допомогою булевих параметрів доступності та перемиканні видимості екранних кнопок відправки й завершення кроків формування завдань.

На рівні рендерингу графічних сторінок планувальника вихідні дані транслюються у візуальні компоненти карток планів, нагадувань та утилізованих об'єктів, які динамічно інтегруються у вертикальні шари прокручуваних областей. Вихідними параметрами кожної картки є скомпоновані текстові мітки заголовків, блоки опису із автоматичним переносом слів та стилізовані хронологічні індикатори. Якщо об'єкт містить посилання на медіаконтент, система здійснює вихідну растрову трансляцію зображення, де бінарний потік файлу перетворюється на об'єкт графічного двигуна, оброблений алгоритмами гладкого масштабування із збереженням пропорцій для виключення візуальних деформацій. Додатково до карт застосовуються вихідні параметри графічних ефектів, які формують просторове сприйняття елементів за рахунок накладання м'яких тіней з керованими радіусами розмиття та зміщеннями осей.

Наостанок, специфічну групу вихідних даних становлять параметри просторового маркування календарної сітки та модальні компоненти зворотного зв'язку з користувачем. Вихідні дані модуля календаря представлені об'єктами текстового форматування символів, які передають колірні та шрифтові модифікації конкретним осередкам дат, що містять заплановані події у базі даних, забезпечуючи наочне хронологічне кодування. У разі виникнення критичних або незворотних операцій, система генерує вихідні керуючі сигнали для виклику модальних діалогових вікон попереднього перегляду медіафайлів та вікон підтвердження дій. Результатом роботи цих вікон є повернення вихідних кодів стандартних кнопок вибору, на основі яких бізнес-логіка програми приймає рішення про безповоротне фізичне очищення дискового простору кошика або каскадне скидання конфігураційних параметрів локального середовища.

2.2 Проектування системи

Накопичення та утримання відомостей у створеному настільному додатку безпосередньо покладається на можливості інтегрованої реляційної системи керування базами даних SQLite. Архітектурний каркас цього цифрового сховища розроблено згідно зі стандартами філософії Offline-First. Такий інженерний підхід забезпечує абсолютну незалежність оператора під час взаємодії із софтом, а також гарантує блискавичне зчитування впорядкованих інформаційних блоків без найменшої потреби здійснювати обмін даними через зовнішні мережеві канали чи віддалені сервери.

Конструювання логічної реляційної моделі здійснювалося із суворим дотриманням ключових правил нормалізації, що дозволило повністю усунути ризики появи непотрібного дублювання записів, а також надійно зафіксувати стабільність внутрішніх зв'язків між окремими сутностями програми. Внутрішня будова сховища об'єднує вісім логічно узгоджених між собою таблиць. Вони безпосередньо відповідають за надійне збереження особистих параметрів власника, системних конфігурацій графічної оболонки, списків доручень, хронологічних відміток у календарі та переліку тимчасово утилізованих елементів.

Загальне архітектурне розгалуження зв'язків разом із детальною конфігурацією полів усіх таблиць локального сховища наочно продемонстровано на рисунку 2.1 у форматі діаграми «сутність-зв'язок» (ER-моделі).

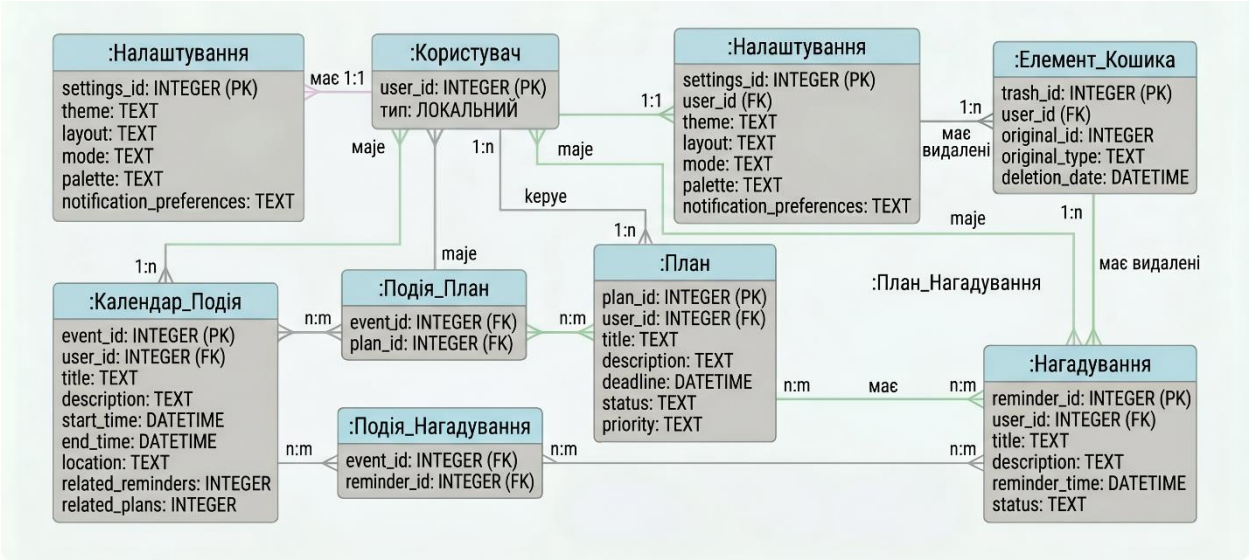

Рисунок 2.1 – ER-модель бази даних застосунку

На рисунку 2.1 наведено концептуальну логічну схему бази даних, центральною базовою сутністю якої виступає таблиця «:Користувач», з якою пов'язані всі інші інформаційні блоки через механізм зовнішніх ключів (Foreign Keys). Відношення між користувачем та його налаштуваннями реалізовано за типом «один до одного» (1:1), тоді як зв'язки з планами, нагадуваннями, календарними подіями та кошиком мають характер «один до багатьох» (1:n). Особливістю схеми є реалізація складних зв'язків «багато до багатьох» (n:m) між календарними подіями та конкретними завданнями через проміжні таблиці-зв'язки «:Подія_План» та «:Подія_Нагадування».

Фізична структура розробленої реляційної бази даних описується призначенням та набором атрибутів таких таблиць:

1. Таблиця «:Користувач» є кореневою сутністю, що ідентифікує локального оператора системи. Вона містить унікальний первинний ключ user_id (типу INTEGER) та маркер типу облікового запису (тип: ЛОКАЛЬНИЙ), що підтверджує офлайн-орієнтованість архітектури.
2. Таблиця «:Налаштування» зберігає глобальні конфігураційні параметри інтерфейсу користувача. Вона містить первинний ключ settings_id та

зовнішній ключ user_id. Додаткові текстові поля theme, layout, mode, palette та notification_preferences зберігають поточний стан візуальної оболонки (наприклад, активну темну чи світлу тему) та преференції системи сповіщень.

3. Таблиця «:План» призначена для збереження довгострокових користувацьких завдань. Запис формується з первинного ключа plan_id, зовнішнього ключа user_id, текстових атрибутів заголовка title та детального опису description. Також таблиця містить часову мітку дедлайну deadline (типу DATETIME), поточний статус виконання status (наприклад, active або completed) та рівень пріоритетності priority.

4. Таблиця «:Нагадування» логує короткострокові завдання із жорсткою прив'язкою до часу. Вона складається з ідентифікатора reminder_id, прив'язки до користувача user_id, текстових полів title та description, а також точної хронологічної мітки спрацьовування reminder_time та поля status.

5. Таблиця «:Календар_Подія» формує сітку інтерактивного календаря. Сутність містить event_id, заголовки та опис події (title, description), чіткі часові межі тривалості (start_time, end_time), текстове поле локації location, а також лічильники або ідентифікатори пов'язаних завдань related_reminders та related_plans.

6. Таблиці зв'язку «:Подія_План» та «:Подія_Нагадування» виступають асоціативними сутностями для розв'язання відношення n:m. Вони складаються виключно із зовнішніх ключів (event_id, plan_id, reminder_id) і дозволяють прив'язувати один план до кількох подій у календарі або об'єднувати декілька нагадувань в одну комплексну календарну подію.

7. Таблиця «:Елемент_Кошика» забезпечує роботу підсистеми утилізації даних та захист від випадкового видалення. Вона містить первинний

ключ trash_id, зовнішній ключ user_id, а також ідентифікатор первинного об'єкта original_id із вказівкою його типу в полі original_type (план чи нагадування). Поле deletion_date фіксує точний час переміщення об'єкта до кошика для можливості його подальшого відновлення.

Запропонована структура бази даних гарантує високу швидкість виконання пошукових запитів (завдяки чіткій типізації) та надійність збереження інформації під час ітеративної взаємодії користувача із чат-асистентом.

2.2.2 Побудова об’єктно-орієнтованої моделі

Об'єктно-орієнтоване моделювання виступає ключовим етапом проєктування архітектури десктопного застосунку, оскільки дозволяє відобразити статичну структуру системи, зафіксувати ієрархію успадкування компонентів та формалізувати логічні зв'язки між різними класами. Для розроблюваного інтелектуального планувальника побудова такої моделі є критично важливою через необхідність координації великої кількості графічних віджетів фреймворку PyQt6, які повинні синхронно реагувати на події текстового інтерфейсу та взаємодіяти з єдиним локальним персистентним шаром бази даних. Головним інструментом специфікації об'єктної структури є уніфікована діаграма класів, яка наочно демонструє розподіл обов'язків всередині програмного коду та інкапсуляцію бізнес-логіки.

Графічне представлення розробленої об'єктно-орієнтованої архітектури, що відображає склад та взаємозв'язки компонентів системи, наведено на рисунку 2.2.

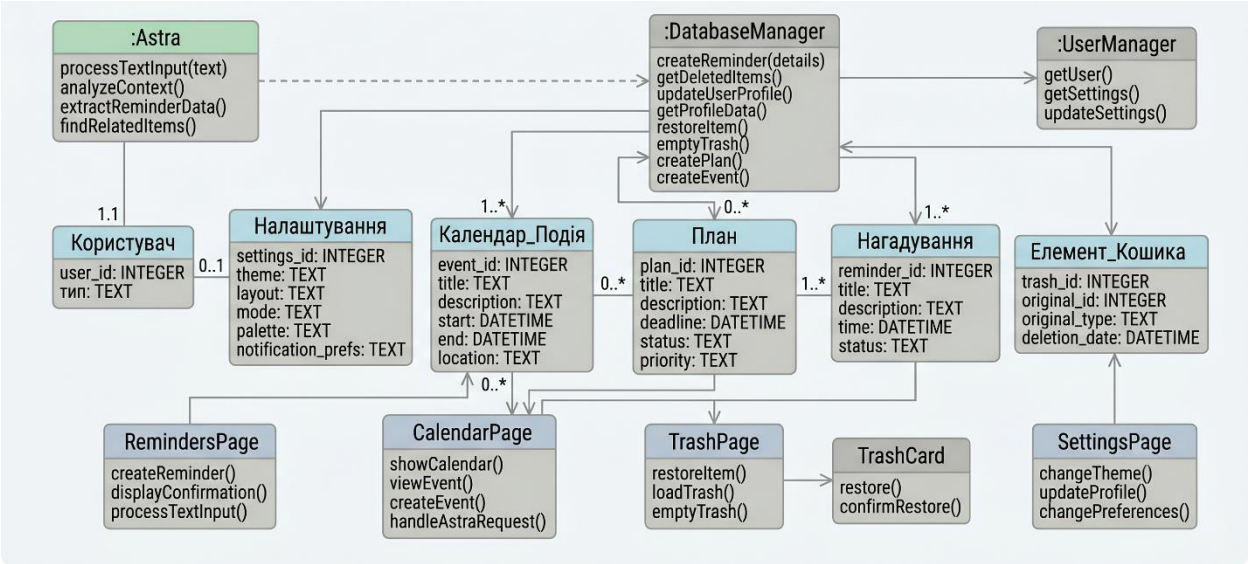

Рисунок 2.2 – Діаграма класів десктопного планувальника

Аналіз побудованої об'єктно-орієнтованої моделі дозволяє виділити чітку ієрархічну підпорядкованість, де базовими елементами виступають нативні класи графічного фреймворку, такі як QWidget, QDialog та QLabel. Центральним ядром та головним координатором всієї системи є клас AstraApp, який успадковує властивості базового вікна користувача та бере на себе функції ініціалізатора довгострокових конфігурацій додатка за допомогою об'єкта QSettings. Цей клас інкапсулює в собі механізм маршрутизації сторінок на основі компонента групування віджетів, здійснюючи каскадне керування життєвим циклом вкладених інтерфейсних вікон. Окрім цього, саме у конструкторі центрального класу відбувається первинне звернення до статичного логічного модуля ініціалізації реляційного сховища, що забезпечує готовність бази даних до виконання транзакцій ще до моменту повного рендерингу графічної оболонки.

Окрему структурну групу в об'єктній моделі становлять класи функціональних сторінок, які безпосередньо формують single-page архітектуру інтерфейсу і взаємодіють з оператором додатка. Класи PlansPage, RemindersPage, CalendarPage, TrashPage та SettingsPage є прямими нащадками візуальних контейнерів і реалізують ізольовані області представлення інформації. Усередині цих класів закладено методи динамічного завантаження та фільтрації контенту, які виконують циклічний рендеринг інформаційних карток на основі поточного стану даних у реляційній базі. Для розширення стандартних можливостей перехоплення подій у графічній системі спроєдовано допоміжний спеціалізований клас ClickableLabel, який розширює базовий текстовий віджет, перевизначає низькорівневий метод обробки натискань миші операційною системою та генерує унікальний високоранговий сигнальний потік для активації модальних вікон попереднього перегляду медіаконтенту.

Проєктні рішення щодо реалізації операцій модифікації та детального аналізу записів відображені через архітектурну групу діалогових компонентів, що функціонують у модальних контекстах. Класи ImageViewerDialog, EditDialog та ReminderEditDialog успадковують властивості системних діалогів і призначені для ізольованого виконання точкових завдань користувача. Зокрема, об'єктна структура ImageViewerDialog оптимізована для обробки важких растрових об'єктів, де логіка ініціалізації включає динамічне зчитування файлових шляхів та застосування алгоритмів гладкого масштабування графічних матриць для запобігання деформаціям інтерфейсу. У той же час класи редагування забезпечують інкапсуляцію полів введення та кнопок підтвердження, виступаючи проміжними контролерами між діями користувача у GUI та методами оновлення записів.

Зв'язок між візуальними класами представлення та шаром даних має характер слабкої зв'язаності (loose coupling), що досягається за рахунок винесення всіх процедур безпосереднього конструювання SQL-запитів у

невізуальний статичний модуль управління базою даних. Графічні сторінки та діалогові вікна не містять у собі явних інструкцій маніпулювання файлом сховища, а натомість виступають клієнтами відносно методів збереження, читання та зміни статусів завдань. Такий архітектурний підхід, відображений на діаграмі класів, забезпечує високий рівень унікальності та масштабованості програмного коду, спрощує процедури подальшого розширення функціоналу системи (наприклад, при додаванні нових типів завдань або кастомізованих віджетів) та повністю унеможливлює виникнення критичних колізій або блокувань графічного потоку під час інтенсивного обміну даними.

2.3 Математичне та алгоритмічне забезпечення

Проєктування дієвого настільного додатка, що використовує комбінований розмовний інтерфейс, потребує залучення ґрунтовної математичної бази та формування швидкісних алгоритмів сортування даних. У процесі роботи такого комплексу левова частка обчислювальних потужностей витрачається на модулі синтаксичного розбору написаних оператором команд, перевірку коректності часових проміжків та координацію мінливих станів цифрового співрозмовника. Для того щоб чітко формалізувати правила діалогу та реакції віртуального помічника «Астра», логіку його поведінки побудовано на основі математичної моделі детермінованого скінченного автомата (Finite State Machine). Впровадження подібного механізму повністю ліквідує ризик появи логічних суперечностей або зависань у ті моменти, коли програма покроково запитує в людини параметри для створення нової картки плану. Як наслідок, застосування скінченного автомата забезпечує стовідсоткову прогнозованість поведінки графічної оболонки та гарантує максимально точну і адекватну відповідь на будь-який текстовий запит. Математично систему управління текстовим

інтерфейсом можна описати як орієнтований граф або кортеж з п'яти елементів $M = (S, X, Y, \delta, \lambda)$, де S - скінченна множина внутрішніх станів асистента (етап очікування та активні кроки збору даних: step = 20, step = 21, step = 21.5, step = 22); X - множина вхідних сигналів (текстові команди та бінарні потоки від вибору файлів); Y - множина вихідних реакцій (кастомізовані текстові відповіді чату та SQL-запити до бази даних); $\delta : S * X \to S$ - функція переходів, яка визначає зміну стану залежно від результатів проміжної валідації; $\lambda : S * X \to Y$ - функція виходів, що генерує відповідну візуальну реакцію у графічній оболонці програми.

На основі розробленої математичної моделі побудовано базовий алгоритм ітеративного функціонування текстового асистента, логічну структуру якого відображено на рисунку 2.3. Алгоритм забезпечує строгу послідовність збору та перевірки інформації, унеможливлюючи потрапляння некоректних або неповних записів до персистентного сховища.

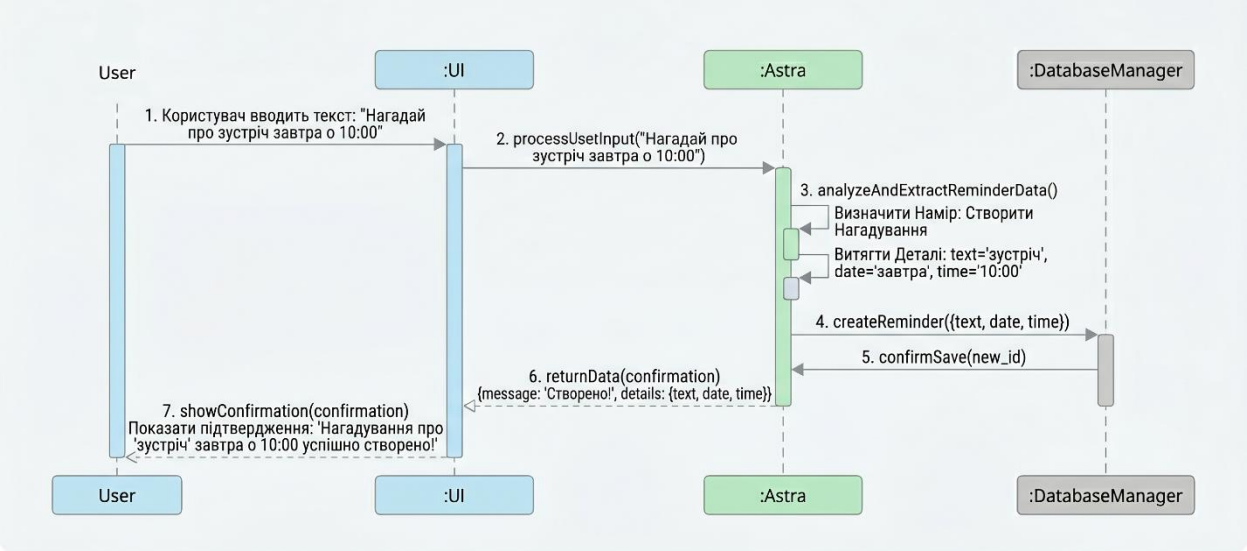

Рисунок 2.3 – Блок-схема алгоритму роботи текстового асистента

Процес виконання алгоритму ініціюється отриманням текстової команди від користувача та модулем ідентифікації наміру на "Створення" об'єкта. Після успішного розпізнавання інтенції, функція переходів переводить систему у стан step = 20, де формується запит на введення

унікальної назви завдання. Отриманий вхідний рядок підлягає негайній перевірці на унікальність шляхом звернення до бази даних SQLite. У разі виникнення колізії (перехоплення винятку порушення цілісності IntegrityError), алгоритм генерує повідомлення про помилку та циклічно повертає систему в початковий стан опитування, блокуючи подальше просування до моменту отримання валідного ідентифікатора.

За умови успішної валідації імені здійснюється детермінований перехід до стану step = 21, де асистент запитує розгорнутий зміст завдання (опис). Після кешування отриманого тексту алгоритм переходить до етапу хронологічного парсингу у стані step = 21.5. На цій ітерації вхідні текстові дані, що містять інформацію про дату та час, проходять валідацію на відповідність підтримуваним системним форматам, формуючи точну часову мітку для майбутнього нагадування.

Завершальним ітеративним кроком збору атрибутів є стан step = 22, що відповідає за опціональну інтеграцію медіаконтенту. Система ініціює виклик нативного модального вікна операційної системи QFileDialog, пропонуючи користувачеві здійснити вибір локального графічного файлу або відмовитися від його додавання. Незалежно від вибору, алгоритм переходить до стадії фіналізації.

Фінальна стадія алгоритму полягає у компіляції зібраного пакета тимчасових даних та ініціації SQL-транзакції збереження в реляційну таблицю. Після отримання підтвердження від СКБД про успішне створення запису, функція виходів λ генерує комплекс керуючих сигналів: відбувається фонове оновлення візуальних компонентів графічного інтерфейсу (GUI) та блокування поля текстового вводу чату для запобігання дублюванню команд. Запропоноване алгоритмічне забезпечення гарантує високу відмовостійкість процесу планування, стійкість до користувацьких помилок та забезпечує когнітивну безперервність діалогової взаємодії.

Висновки до розділу

У другому розділі цього кваліфікаційного проєкту виконано повний цикл дослідницьких та інженерних робіт, необхідних для створення інформаційного, структурного, об'єктного та математичного фундаменту майбутнього планувальника. Детальний розбір напрямків руху інформації дав змогу чітко розмежувати типи вхідних і вихідних параметрів. Завдяки цьому звичайні розмовні фрази користувача перетворюються на строгі машинні формати, які легко зчитуються внутрішніми модулями додатка. Крім того, тут практично обґрунтовано принцип Offline-First. Згідно з ним, обробка будь-яких текстових команд чи об'ємних графічних файлів здійснюється виключно в ізольованій пам'яті самого комп'ютера. Такий підхід робить програму максимально стійкою до збоїв мережі та гарантує абсолютну конфіденційність приватних записів.

Розробка архітектури збереження відомостей завершилася побудовою нормалізованої реляційної моделі на основі СКБД SQLite, яка налічує вісім взаємопов'язаних таблиць. Створена схема «сутність-зв'язок» (ER-діаграма) успішно вирішує проблему координації складних залежностей між параметрами профілю, загальними налаштуваннями дизайну, переліком справ, сповіщеннями та календарними подіями. Для організації цих зв'язків задіяно інструментарій зовнішніх ключів та допоміжні таблиці-зв'язки. Окремо варто виділити механізм логічного відокремлення стертих елементів. Таке рішення безпосередньо втілює принцип безпечної утилізації контенту, зводячи до мінімуму будь-які шанси на випадкове та безповоротне знищення цінних матеріалів користувача.

Побудова об'єктно-орієнтованої структури у форматі діаграми класів дозволила наочно зобразити ієрархію компонентів бібліотеки PyQt6 та

прописати правила їхньої взаємодії всередині односторінкової оболонки. Завдяки наявності головного вікна-координатора та системи модульного перемикання екранів, вдалося чітко розмежувати робочі зони додатка, серед яких інтелектуальний чат, динамічна сітка календаря та панелі для перегляду нотаток. Створена об'єктна схема спирається на правило слабкої зв'язності (loose coupling). Тобто всі прямі звернення до локального сховища виокремлено в самостійний статичний клас-менеджер. Таке архітектурне рішення надійно захищає головний потік відмальовування графіки від мікрозависань у моменти інтенсивного збереження або зчитування матеріалів.

Насамкінець, математичний та алгоритмічний фундамент системи отримав своє логічне обґрунтування завдяки застосуванню моделі скінченного автомата для управління розмовним інтерфейсом. Використаний математичний апарат дав можливість чітко розділити внутрішні стани віртуального помічника «Астра» під час поетапного опитування людини та пов'язати ці кроки з відповідними візуальними реакціями екрана. Складена блок-схема роботи алгоритму в деталях розкриває етапи попередньої перевірки інформації. Сюди входить контроль за унікальністю назв, розпізнавання форматів часу та правильне підключення картинок. Все це разом формує міцну відмовостійкість програмної логіки та гарантує, що до локального сховища ніколи не потраплять помилкові чи напівпорожні записи.

РОЗДІЛ 3 ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Засоби розробки

Створення настільної програми, яка має вбудованого текстового помічника, починається з правильного вибору базових технологій. Цей набір інструментів повинен гарантувати швидку відмальовку графіки, безпечне утримання локальної інформації та миттєвий аналіз введених слів. У якості основної мови для написання коду застосунку «Астра» обрано Python. Головна причина такого рішення полягає у відмінній роботі цієї мови з текстовими рядками. Для реалізованого проєкту це критично важливо, адже віртуальний співрозмовник постійно розбирає та аналізує фрази оператора. Додатково, гнучка типізація та великий набір готових модулів допомогли значно пришвидшити конструювання алгоритмів. Створений скрипт легко запускається на різних операційних системах, що звільняє від необхідності переписувати архітектуру під кожну платформу окремо.

Щоб сформувати комбіновану візуальну оболонку, у проєкті залучено можливості бібліотеки PyQt6. Вона відкриває прямий доступ до функцій потужного фреймворку Qt. Завдяки інструменту QStackedWidget вдалося побудувати зручну навігацію, де екрани миттєво змінюють один одного без повного перезавантаження сторінок. Бібліотека також надала засоби для сучасного оформлення інтерфейсу. Наприклад, картки планів отримали об'ємні тіні за допомогою класу QGraphicsDropShadowEffect, а прикріплені картинки масштабуються без втрати якості завдяки алгоритму SmoothTransformation. Принципово вагомою стала внутрішня система сигналів та слотів. Вона допомогла налаштувати швидкий відгук кнопок і панелей на будь-які дії людини, чи то друкування повідомлення в чаті, чи клік по конкретній даті у віджеті календаря.

Надійне збереження інформації організовано за допомогою системи SQLite. Ця база даних чудово вписалася в концепцію автономної роботи без доступу до глобальної мережі. Вона записує всі таблиці, хронологічні мітки та тексти нотаток у звичайний локальний файл на жорсткому диску комп'ютера. Завдяки цьому повністю зникає потреба налаштовувати додаткові фонові сервери. Оскільки SQLite цілком підтримує стандарти безпечних транзакцій, користувач може бути впевненим, що його записи не зникнуть через раптовий системний збій. Щоб зв'язати логіку програми із таблицями, застосовано стандартний модуль sqlite3. Він чудово справляється з передачею безпечних параметризованих запитів та миттєво реагує на помилки, якщо виникає конфлікт ідентифікаторів під час спроби зберегти нове доручення.

Для безпосереднього написання вихідного коду, пошуку помилок та тестування функціоналу задіяно інтегроване середовище PyCharm. Цей редактор містить потужний аналізатор, який підказує правильний синтаксис та помічає логічні хиби прямо під час набору рядків. Щоб уникнути конфліктів між сторонніми бібліотеками, програма самостійно створює ізольований віртуальний простір. Дуже корисним під час розробки виявився візуальний дебагер. Він дав змогу буквально по кроках спостерігати за тим, як скінченний автомат Астри змінює свої стани після кожної введеної команди. Додатковою перевагою стала наявність панелі Database Tools прямо всередині редактора. Вона дозволяла миттєво переглядати вміст таблиць бази даних і перевіряти, чи правильно зберігаються щойно створені завдання, без необхідності відкривати сторонні утиліти.

3.2 Вимоги до технічного та програмного забезпечення

Щоб створений настільний планувальник працював без збоїв, інтерфейс відмальовувався плавно, а комп'ютерний співрозмовник миттєво реагував на

введені повідомлення, пристрій користувача має відповідати певним технічним критеріям. Оскільки внутрішня будова програми спирається на фреймворк PyQt6 і активно використовує складні графічні ефекти (зокрема, об'ємні тіні, нестандартні шрифти та алгоритми м'якого згладжування картинок SmoothTransformation), до апаратної частини висуваються цілком раціональні, але суворо визначені вимоги.

Мінімальні параметри комп'ютерного обладнання розраховувалися з огляду на те, що системі доведеться паралельно підтримувати роботу віртуального середовища Python та графічного рушія Qt. Для зручної щоденної експлуатації центральний процесор повинен мати 64-розрядну архітектуру (формату x86-64 або ж ARM64) із робочою частотою від 2.0 ГГц. На практиці це означає використання чипів рівня Intel Core i3 чи AMD Ryzen 3 базових поколінь. Обсяг оперативної пам'яті відіграє критичну роль, адже модуль багатосторінкової навігації QStackedWidget зберігає активні вікна у фоновому кеші. Це робиться для того, щоб людина могла без затримок перемикатися між чатом, календарем та стрічкою справ. Тому найнижчою межею оперативної пам'яті є 4 гігабайти. Однак, якщо планується часте завантаження та перегляд об'ємних графічних файлів через інструмент ImageViewerDialog, бажано мати на борту 8 гігабайт.

Вимоги до вільного місця на жорсткому диску можна розділити на два типи: стартові та змінні. Початковий обсяг, якого вистачить для розпакування скомпільованого файлу та створення порожнього сховища astra_data.db, становить менше ніж 200 мегабайтів. Змінна частина безпосередньо залежить від того, наскільки часто власник буде додавати власні малюнки до карток завдань. Щоб гарантувати максимальну швидкість запису та зчитування інформації під час гортання великих списків, найкраще встановлювати програму на твердотільні накопичувачі (SSD). Окрім цього, специфічне компонування екрана, де робоча зона поділена на діалогове вікно і панель перегляду контенту, вимагає наявності дисплея з мінімальною роздільною

здатністю 1280x720 пікселів. Оптимальним варіантом стане Full HD монітор (1920x1080) із підтримкою 32-бітної глибини кольорів, що дозволить максимально коректно передавати відтінки світлої та темної тем оформлення.

Критерії щодо системного програмного забезпечення диктуються мультиплатформною специфікою обраних технологій. Створений застосунок скомпоновано так, що він здатен нативно функціонувати на всіх актуальних настільних операційних системах. Сюди входять останні версії Microsoft Windows (десятого та одинадцятого поколінь), платформи Apple macOS (починаючи від версії 11.0 Big Sur), а також сучасні збірки GNU/Linux (як приклад, Ubuntu 20.04 LTS та новіші релізи). Такий високий рівень сумісності гарантує бібліотека Qt, яка самостійно адаптує внутрішні виклики під конкретну ОС.

Що стосується залежностей середовища виконання, то тут передбачено два можливі сценарії. Якщо програма постачається як готовий до запуску виконуваний файл (згенерований через утиліти на кшталт PyInstaller), кінцевому користувачу взагалі не доведеться встановлювати жодних додаткових компонентів чи інтерпретаторів. У ситуації, коли софт запускатиметься або доопрацьовуватиметься безпосередньо з вихідного коду, цільова машина має бути оснащена інтерпретатором Python версії 3.10 або вище. Також у віртуальному оточенні проєкту обов'язково повинен бути інстальований пакет PyQt6, який відповідає за генерацію візуальної оболонки. Натомість рушій бази даних SQLite не потребує ані завантаження окремих інсталяторів, ані налаштування фонових серверних процесів. Його драйвер відпочатку вбудований у стандартний набір інструментів мови Python, завдяки чому підсистема управління інформацією повністю готова до роботи буквально з першої секунди після запуску органайзера.

3.3 Опис програмної реалізації

Написання вихідного коду для створеного настільного органайзера спирається на модульний принцип конструювання. Кожен самостійний блок логіки розміщено у власному файлі, що значно полегшує читання скриптів, спрощує майбутні оновлення та дозволяє легко розширювати функціонал програми. Запуск усього комплексу починається зі скрипта astra.py. Саме тут оголошено головний клас AstraApp, який переймає базові характеристики стандартного вікна QWidget. Коли софт створює екземпляр цього об'єкта, він найперше звертається до функції init_db(), розміщеної у статичному файлі database_manager.py. Такий крок є обов'язковим, адже він гарантує генерацію файлу сховища та перевіряє наявність усіх потрібних колонок ще до того, як на екрані почнуть малюватися елементи інтерфейсу. Щойно робота з локальною пам'яттю успішно стартувала, застосунок підключає об'єкт QSettings. Він потрібен для миттєвого зчитування збережених налаштувань дизайну з реєстру чи системних каталогів ОС, щоб одразу застосувати правильні каскадні таблиці стилів.

Візуальна оболонка проєкту функціонує за логікою односторінкового додатка (Single-Page Application). Основну роль тут відіграє інструмент маршрутизації QStackedWidget. Усередині конструктора центрального класу відбувається ініціалізація головних робочих екранів: PlansPage, RemindersPage, CalendarPage, TrashPage, а також SettingsPage. Усі ці візуальні блоки послідовно завантажуються до загального стека віджетів. Щоб користувач міг вільно переміщатися між згаданими розділами, передбачено бічну навігаційну панель. Клік по будь-якій кнопці меню створює системний сигнал, який одразу змінює активний індекс стека. Подібна інженерна хитрість дає можливість фіксувати поточний стан будь-якої сторінки. Наприклад, якщо людина ввела текст у пошук або прокрутила довгий список, ці дані не зникнуть після переходу на іншу вкладку. До того ж це суттєво економить ресурси

оперативної пам'яті комп'ютера, оскільки системі не потрібно постійно генерувати нові віконні процеси під час кожної зміни екрана.

Внутрішні механізми комбінованого текстового спілкування приховані у методах об'єкта AstraApp. Роботою скінченного автомата, який керує поведінкою цифрового помічника, завідує спеціальна змінна цілого типу creation_step. Вона постійно фіксує, на якому саме етапі опитування зараз перебуває співрозмовник. Захоплення та аналіз написаних оператором слів здійснюється через функцію handle_input(). Вона витягує текст із рядка QLineEdit і направляє його до алгоритмів логічного розгалуження. Щоб діалог мав гарний вигляд на дисплеї, написано окрему процедуру append_message(). Її завдання — огортати повідомлення оператора та репліки Астри у красиву HTML-розмітку, після чого виводити їх на панель QTextBrowser. Коли черга доходить до етапу прикріплення картинок, скрипт викликає стандартне вікно системи QFileDialog.getOpenFileName. Отримавши повну адресу обраного малюнка на диску, додаток тимчасово тримає цей шлях у словнику temp_command_data, чекаючи на команду остаточного збереження.

Комунікація головного рушія з таблицями SQLite винесена у відокремлений файл database_manager.py. Він відіграє роль своєрідного мосту або проміжного шару доступу до інформації (Data Access Layer). Усередині цього скрипта зібрано колекцію самостійних процедур, на кшталт save_command(), get_items_by_type() або update_status(). Кожна з перелічених функцій вміє самотужки відкривати канал зв'язку з локальним документом astra_data.db. Вони передають туди безпечні параметризовані SQL-команди, використовуючи кортежі для гарантованого захисту від хакерських ін'єкцій. Далі відбувається остаточне підтвердження змін (commit) і безпечне закриття сесії. Щоб програма не "вилітала" через внутрішні конфлікти, впроваджено конструкції перехоплення помилок try-except. Вони миттєво реагують на порушення типу sqlite3.IntegrityError, якщо користувач намагається створити завдання з ідентичною назвою. У таких випадках модуль просто віддає

булевий прапорець про невдачу, дозволяючи графічній оболонці м'яко повідомити людину про проблему.

Базовий набір інструментів бібліотеки PyQt6 довелося розширити шляхом написання унікальних користувацьких класів. Це дозволило зробити відображення контенту більш гнучким. Наприклад, у скриптах для генерації переліку справ та календарної сітки (calendar_page.py) з'явився новий компонент ClickableLabel. Він бере за основу звичайний текстовий блок, однак повністю переписує базову подію mousePressEvent. Тепер, якщо людина клацає по ньому лівою клавішею миші, об'єкт генерує спеціальний сигнал clicked. Завдяки цій прив'язці на екрані одразу з'являється спливаюче вікно ImageViewerDialog, що відповідає за показ завантажених малюнків. Процес запуску цього діалогового блоку включає зчитування бінарного коду картинки за збереженою адресою та перетворення її на формат QPixmap. Для того щоб зображення будь-якого розміру виглядало чітко і без "драбинок" на краях, застосовується алгоритм динамічного масштабування SmoothTransformation. Він якісно згладжує пікселі, ідеально підганяючи графіку під доступні габарити екрана перегляду.

3.3.1 Опис основних класів та функціональних модулів

Програмна архітектура розробленого планувальника завдань базується на строгій об'єктно-орієнтованій декомпозиції, що дозволяє відокремити логіку маршрутизації, обробку бази даних та рендеринг графічних компонентів. Головним координаційним центром системи виступає клас AstraApp, який успадковує базовий контейнер QWidget фреймворку PyQt6. Цей клас відповідає за ініціалізацію персистентного сховища, застосування глобальних параметрів візуальної теми через об'єкт QSettings та організацію багатосторінкового інтерфейсу за допомогою компонента QStackedWidget.

Унікальною особливістю класу є інкапсуляція логіки віртуального помічника, керованого лічильником станів creation_step.

Процес ітеративного збору атрибутів завдання та взаємодії з користувачем через текстовий інтерфейс вимагає складної програмної реалізації парсингу та розгалужень. У лістингу 3.1 наведено фрагмент програмного коду класу AstraApp, який відповідає за фінальний етап формування нагадування (перевірка згоди на додавання медіафайлу) та виклик транзакції збереження.

Лістинг 3.1 – Програмна реалізація фінального етапу скінченного автомата розмовного інтерфейсу

```
elif self.creation_step == 22:
        if text_lower in ['так', 'yes', 'д', 'т']:
            file_name, _ = QFileDialog.getOpenFileName(self, "Виберіть картинку", "",
                                                       "Images (*.png *.jpg *.jpeg *.bmp)")
            if file_name:
                self.temp_command_data['image_path'] = file_name
            else:
                self.temp_command_data['image_path'] = None
        else:
            self.temp_command_data['image_path'] = None
        success = database_manager.save_command(
            self.temp_command_data['name'],
            self.temp_command_data['content'],
            None,
            'reminder',
            self.temp_command_data.get('image_path')
        )
        if success:
```

```
                self.append_message("Астра", f"Чудово! Нагадування збережено. Натисніть 'Готово'.")
                self.user_input.setEnabled(False)
                self.send_btn.hide()
                self.done_btn.show()
            else:
                self.append_message("Астра",
                                    f"Помилка: ім'я «<b>{self.temp_command_data['name']}</b>» вже
зайнято.<br>Давайте почнемо спочатку. Як назвемо нагадування?")
                self.creation_step = 20
```

Основна логіка, показана у лістингу 3.1, алгоритм аналізує текстове введення користувача (text_lower), ідентифікуючи ствердні патерни згоди на імпорт зображення. При позитивному результаті ініціюється виклик нативного модального вікна операційної системи QFileDialog, результати якого кешуються у тимчасовому словнику temp_command_data. Після цього викликається зовнішній метод модуля управління базою даних. Залежно від повернутого булевого прапорця success, графічний інтерфейс динамічно трансформується: у разі успіху поле введення user_input блокується задля уникнення дублювання операцій, а в разі виявлення колізій імен (ім'я вже зайнято) автомат циклічно повертається на етап creation_step = 20, генеруючи відповідне HTML-форматоване повідомлення про помилку у вікні чату.

Ізоляція логіки доступу до даних реалізована через статичний модуль database_manager.py. Це архітектурне рішення гарантує безпеку транзакцій та запобігає прямому маніпулюванню SQL-запитами з боку візуальних класів. У лістингу 3.2 наведено реалізацію методу збереження нового запису до локального файлу SQLite.

Лістинг 3.2 – Метод персистентного збереження завдання у локальну базу даних SQLite

```
def save_command(name, content, reminder_date=None, item_type='plan', image_path=None):
    """Зберігає новий план або нагадування разом із картинкою."""
    conn = sqlite3.connect(DB_NAME)
    cursor = conn.cursor()
    try:
        cursor.execute('''
                    INSERT INTO commands (name, content, reminder_date, item_type, image_path)
                    VALUES (?, ?, ?, ?, ?)
                    ''', (name.lower(), content, reminder_date, item_type, image_path))
        conn.commit()
        success = True
    except sqlite3.IntegrityError:
        success = False
    finally:
        conn.close()
    return success
```

Аналіз лістингу 3.2 демонструє застосування параметризованих інструкцій при виконанні команди INSERT INTO. Використання кортежів параметрів замість прямої конкатенації рядків повністю нівелює вразливості типу SQL Injection. Блок перехоплення винятків try-except фокусується на обробці системної помилки sqlite3.IntegrityError, яка виникає при спробі порушити обмеження унікальності (UNIQUE constraints), накладене на колонку name під час проєктування схеми. Блок finally забезпечує гарантоване закриття з'єднання з базою даних незалежно від результатів виконання транзакції, що є критично важливою вимогою для запобігання витокам пам'яті та блокуванню файлу astra_data.db операційною системою.

Окрему групу в архітектурі складають класи візуального представлення, що відповідають за рендеринг списків. Класи PlansPage та CalendarPage реалізують динамічний життєвий цикл карток завдань. Для забезпечення

інтерактивності графічних елементів, зокрема медіафайлів, стандартний інструментарій PyQt6 було розширено через створення кастомного класу ClickableLabel. Цей клас успадковує властивості текстового віджета QLabel, але перевизначає низькорівневий метод mousePressEvent, генеруючи унікальний сигнал clicked. Ця абстракція дозволяє безперешкодно прив'язувати події кліку до виклику модального діалогу ImageViewerDialog, який здійснює гладке масштабування (SmoothTransformation) растрових матриць QPixmap, забезпечуючи якісний попередній перегляд прикріплених зображень без деформації пропорцій або погіршення візуальної якості в інтерфейсі.

3.3.2 Обробка даних та безпека

Організація надійного та захищеного середовища для обробки користувацької інформації є фундаментальним критерієм якості десктопного програмного комплексу. Оскільки архітектура розробленого планувальника базується на парадигмі Offline-First, ключовим вектором безпеки виступає повна ізоляція інформаційних потоків у межах локальної файлової системи кінцевого пристрою. Відсутність мережевих трансляцій та віддалених серверних сховищ повністю нівелює ризики перехоплення трафіку або несанкціонованого доступу до бази даних з боку зовнішніх зловмисників. Проте локальне функціонування висуває підвищені вимоги до захисту даних від внутрішніх ін'єкційних вразливостей, колізій імен, логічних помилок введення та випадкової деструкції записів.

Для забезпечення транзакційної цілісності та захисту від критичних вразливостей типу SQL-ін’єкцій, усі процедури модифікації даних у персистентному шарі реалізовані за допомогою суворо типізованих параметризованих інструкцій СКБД. У лістингу 3.3 наведено фрагмент

програмного коду з файлу database_manager.py, який демонструє механізм безпечного оновлення параметрів завдань.

Лістинг 3.3 – Реалізація безпечного оновлення записів із захистом від ін'єкцій та колізій у database_manager.py

```
def update_command(command_id, name, content, reminder_date, image_path):
    """Оновлює існуючу команду (з урахуванням картинки)."""
    conn = sqlite3.connect(DB_NAME)
    cursor = conn.cursor()
    try:
        cursor.execute('''
                    UPDATE commands
                    SET name          = ?,
                        content       = ?,
                        reminder_date = ?,
                        image_path    = ?
                    WHERE id = ?
                    ''', (name.lower(), content, reminder_date, image_path, command_id))
        conn.commit()
        success = True
    except sqlite3.IntegrityError:
        success = False
    conn.close()
    return success
```

Детальний аналіз програмного рішення, наведеного у лістингу 3.3, дозволяє виділити кілька рівнів обробки та захисту інформації. По-перше, рядок назви завдання проходить процедуру примусової санітизації та нормалізації за допомогою методу рядкового приведення name.lower(). Це виключає виникнення логічних дублікатів у реляційній таблиці, які могли б

відрізнятися лише регістром символів. По-друге, інструкція UPDATE використовує класичні плейсхолдери значень у вигляді знаків запитання, де масив змінних передається окремим кортежем параметрів. Такий підхід гарантує, що інтерпретатор SQL сприйматиме будь-який введений користувачем текст (включаючи спецсимволи чи службові команди) виключно як статичний рядковий контент, що робить додаток абсолютно стійким до спроб модифікації структури запиту. Крім того, блок try-except здійснює перехоплення системних помилок унікальності sqlite3.IntegrityError, ізолюючи графічну оболонку програми від аварійних збоїв та повертаючи логічний маркер невдачі для подальшої обробки графічним інтерфейсом.

Важливим компонентом підсистеми обробки даних є керування життєвим циклом завдань за принципом мінімізації деструктивного впливу. Замість фізичного видалення записів з дискового носія при першому запиті користувача, архітектура підтримує двостадійну утилізацію контенту на основі патерну м'якого видалення (Soft Delete). Програмна реалізація зміни логічного стану об'єктів без руйнування їхньої структури представлена у лістингу 3.4.

Лістинг 3.4 – Програмний механізм зміни життєвого циклу об'єктів (патерн Soft Delete)

```
def update_status(command_id, status):
    """Змінює статус плану (active <-> completed <-> deleted)."""
    conn = sqlite3.connect(DB_NAME)
    cursor = conn.cursor()
    cursor.execute('UPDATE commands SET status = ? WHERE id = ?', (status, command_id))
    conn.commit()
    conn.close()
```

Як демонструє програмний код лістингу 3.4, операція видалення або завершення завдання зводиться до параметризованої модифікації текстового поля status. Об'єкти зі значенням deleted миттєво виключаються з активних списків рендерингу головних сторінок планів, нагадувань та календарної сітки, переміщуючись до ізольованого сховища кошика TrashPage. Це забезпечує надійний захист інформаційного середовища оператора від випадкових деструктивних дій, надаючи можливість швидкого відновлення первинного стану запису в один клік.

Для повного завершення циклу безпеки, на етапах проведення незворотних операцій (таких як остаточне очищення кошика або повне скидання бази даних через файл settings_page.py), на рівні графічної оболонки впроваджено захисні бар'єри у вигляді перехоплюючих модальних вікон QMessageBox.question. Вони вимагають від користувача свідомого підтвердження дій до моменту виклику фінальних функцій очищення, що спільно з параметризацією SQL-запитів формує комплексне, стійке до збоїв та спроб компрометації програмне забезпечення.

3.3.3 Приклад обробки помилок та виняткових ситуацій

Ефективна експлуатація розробленого програмного комплексу забезпечується завдяки інтуїтивно зрозумілому гібридному графічному інтерфейсу, що суттєво мінімізує поріг входження для нових користувачів. Після успішного запуску виконуваного файлу програми та первинної ініціалізації локальної бази даних, оператор потрапляє на головний екран, який функціонує в режимі інтелектуального чат-асистента. Візуальна архітектура простору побудована за принципом ергономічного розподілу: ліворуч розміщено статичну навігаційну панель для швидкого маршрутного перемикання між логічними модулями, а центральну, найбільшу частину

екрана, займає динамічне вікно діалогу з Астрою. Процес створення нового плану або нагадування ініціюється природним шляхом - через введення базової текстової команди у нижнє поле вводу. На рисунку 3.4 продемонстровано базовий сценарій взаємодії, де користувач проходить ітеративне опитування для генерації завдання, а віртуальний помічник виводить відформатовані підказки та блокує інтерфейс від помилкових дій.

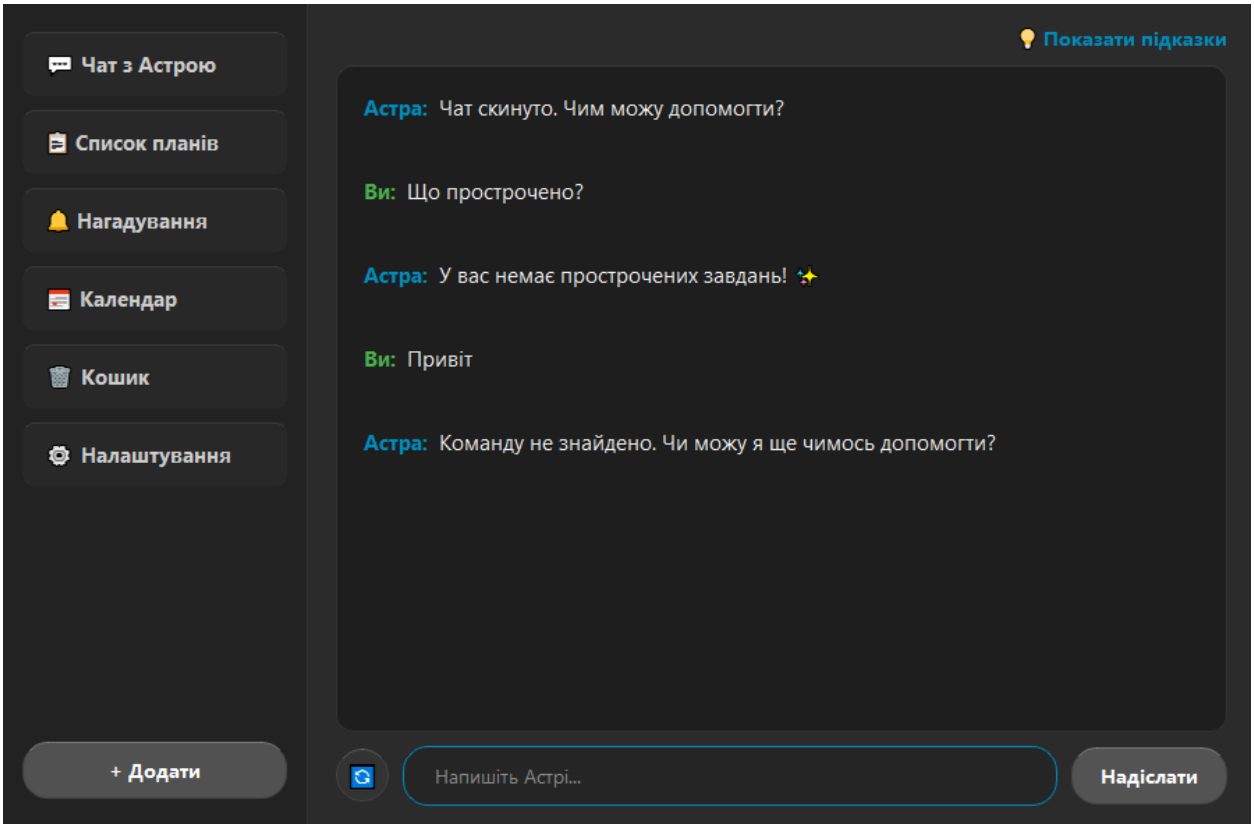

Рисунок 3.4 – Інтерфейс діалогової взаємодії з віртуальним асистентом під час ініціалізації сесії

З технічної точки зору, кожна дія користувача в текстовому полі чату обробляється відповідним внутрішнім методом, який відповідає за зчитування, очищення та передачу рядка до аналітичного ядра програми. Для розуміння того, як саме користувацьке введення трансформується у сигнальні потоки графічної оболонки, у лістингу 3.4 наведено фрагмент програмного

коду, який безпосередньо керує поведінкою поля введення при натисканні кнопки відправки повідомлення або клавіші Enter.

Лістинг 3.5 – Програмний механізм обробки та очищення поля введення в astra.py

```
def handle_input(self):
    text = self.user_input.text().strip()
    if not text:
        return
    self.user_input.clear()
    self.append_message("Ви", text)
    text_lower = text.lower()
```

Як демонструє програмний код лістингу 3.5, інтерфейс миттєво реагує на дію користувача, очищуючи поле user_input та транслюючи введений текст у вікно діалогу із тегом автора «Ви». Цей крок є обов'язковим для створення безперервного та звичного досвіду спілкування з чат-ботом. Після цього очищений та приведений до нижнього регістру рядок надходить до логічного блоку розгалужень скінченного автомата Астри, який наочно відображає поточний етап збору інформації, як це проілюстровано на рисунку 3.5.

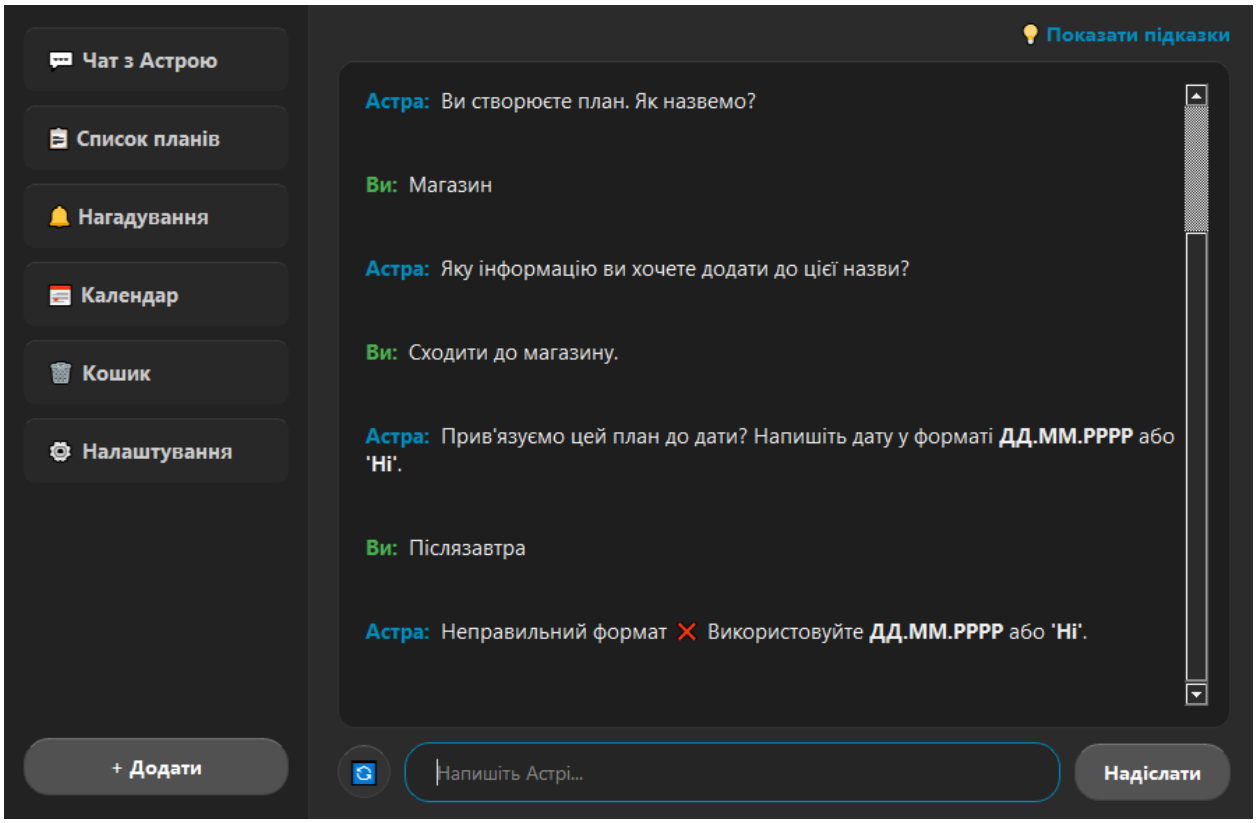

Рисунок 3.5 – Динамічне відображення ходу покрокового опитування у діалоговому вікні та реагування на помилки.

Після успішного збереження інформації через розмовний інтерфейс, система надає можливість здійснювати детальний моніторинг поточних цілей через спеціалізовані графічні панелі. Перехід до розділу «Календар» за допомогою відповідної кнопки бічного меню відкриває інтерактивну хронологічну матрицю. Програмна логіка цього модуля автоматично аналізує базу даних та візуально підсвічує дати, за якими закріплені активні дедлайни. При натисканні лівою кнопкою миші на конкретну дату, у нижній частині вікна динамічно рендериться вертикальний список карток завдань. Кожна картка містить стилізований заголовок, текстовий опис, точну часову мітку та, за наявності, інтерактивну мініатюру прикріпленого медіафайлу, як це детально показано на рисунку 3.6.

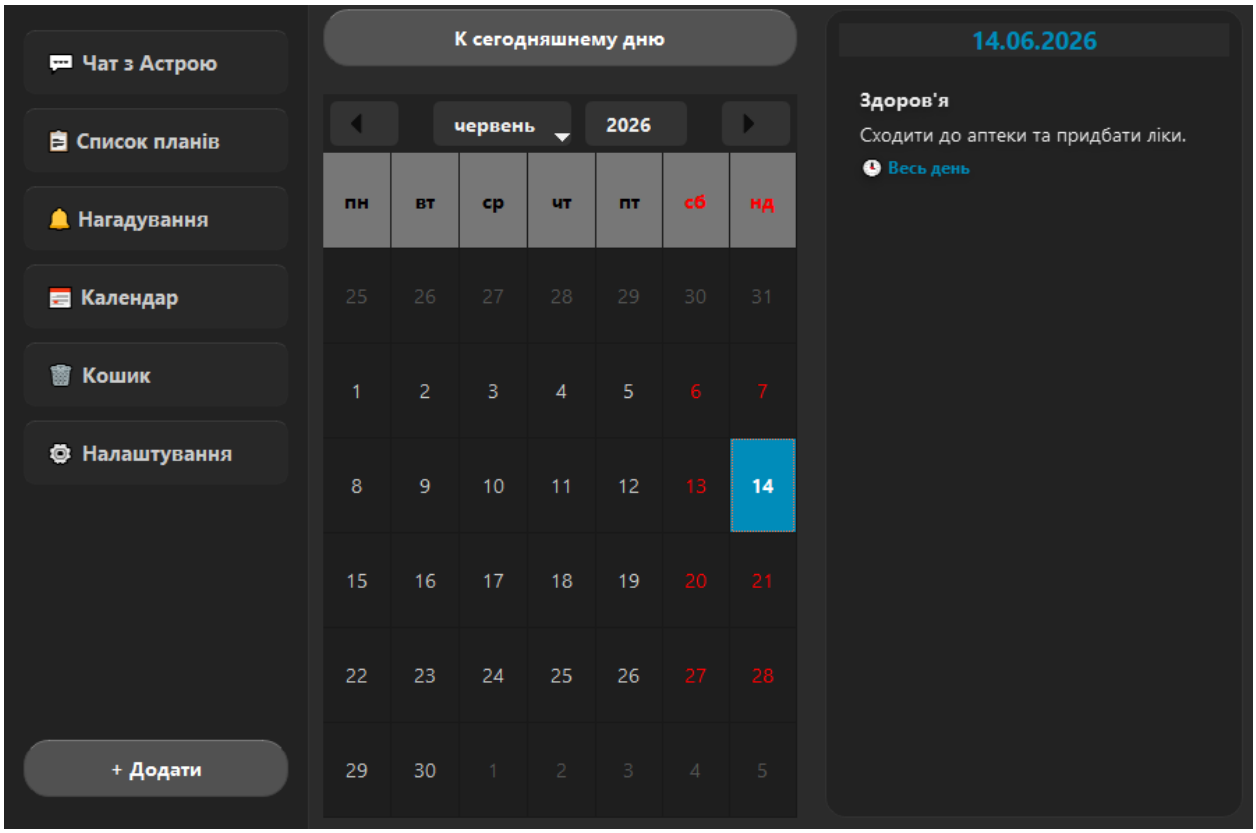

Рисунок 3.6 – Відображення активних завдань в інтерактивному модулі календаря

Для того, щоб забезпечити такий динамічний відгук календарної сітки на кліки користувача, на рівні програмної реалізації було створено спеціальний метод-слухач подій. У лістингу 3.6 наведено фрагмент коду з файлу calendar_page.py, який описує логіку перехоплення вибору дати користувачем та ініціацію процедури повного оновлення списку завдань під актуальний хронологичний контекст.

Лістинг 3.6 – Метод зв'язування подій календаря із процедурою динамічного рендерингу

```
def init_ui(self):
    # Логіка ініціалізації віджетів календаря
    self.calendar = QCalendarWidget()
```

```
self.calendar.setGridVisible(True)
self.calendar.clicked.connect(self.update_day_plans)
```

Завдяки архітектурному рішенню, наведеному в лістингу 3.6, подія clicked нативного віджета QCalendarWidget підключається до внутрішнього слота update_day_plans. Це дозволяє програмі миттєво зчитувати параметри обраного дня, формувати SQL-запит до реляційної таблиці за допомогою менеджера бази даних та перемальовувати картки без фрізів інтерфейсу, що забезпечує високу швидкість відгуку системи на дії оператора.

Окремим важливим аспектом взаємодії є гнучке управління життєвим циклом інформаційних об'єктів та кастомізація робочого середовища. Кожна згенерована картка у вкладках планів оснащена навігаційними кнопками для зміни статусу. При підтвердженні виконання або ініціації видалення, завдання миттєво зникає з головного екрана та переміщується до відповідного архівного розділу або логічного кошика. Якщо до завдання було попередньо прикріплено графічний файл, клік по його мініатюрі викликає модальне вікно попереднього перегляду, де застосовуються алгоритми гладкого масштабування зображення. Окрім управління контентом, користувачу доступна панель глобальних конфігурацій, де реалізовано механізм гарячої підміни каскадних таблиць стилів. Це дозволяє в один клік перемикати візуальну оболонку застосунку між світлою та темною темами для зниження навантаження на зір. Приклад інтерфейсу налаштувань та відображення системи сповіщень наведено на рисунку 3.7.

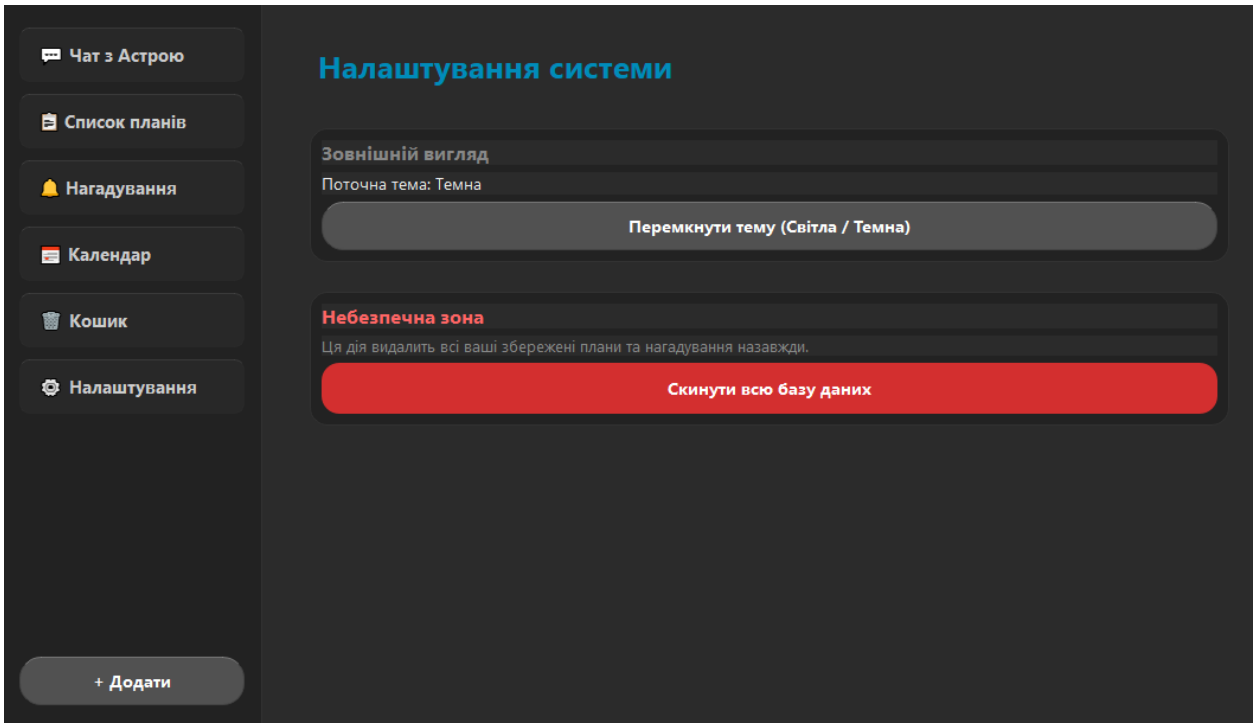

Рисунок 3.7 – Панель глобальних конфігурацій та управління системними параметрами застосунку

Процедура зміни візуального стилю програми користувачем підпорядковується суворому алгоритму динамічної заміни стилістичних констант, що виключає необхідність перезапуску десктопного додатка. Програмне забезпечення, яке відповідає за обробку натискання кнопки перемикання палітри у вкладці конфігурацій, наведено у лістингу 3.7.

Лістинг 3.7 – Програмна логіка гарячої зміни теми оформлення у settings_page.py

```
def toggle_theme(self):
    new_theme = "light" if self.main_app.current_theme == "dark" else "dark"
    self.main_app.change_theme(new_theme)
    status_text = "Світла" if new_theme == "light" else "Темна"
    self.theme_status.setText(f"Поточна тема: {status_text}")
```

Аналіз програмної реалізації з лістингу 3.7 показує, що метод toggle_theme виконує логічну інверсію поточного строкового стану теми програми та делегує виконання каскадної зміни стилів головному класу програми main_app. Після успішного застосування нових параметрів QSS-файлу, текстова мітка theme_status динамічно оновлює свій вміст, інформуючи користувача про поточний режим відображення. Такий комплексний та інтегрований підхід до побудови елементів керування, підкріплений надійними методами обробки подій, забезпечує створення ергономічного, стійкого до помилок та когнітивно комфортного середовища для щоденного планування завдань.

3.4 Керівництво користувача

Ефективна експлуатація розробленого десктопного програмного комплексу забезпечується завдяки інтуїтивно зрозумілому гібридному графічному інтерфейсу, що суттєво мінімізує когнітивне навантаження та поріг входження для нових користувачів. Після успішного запуску програми та автоматичної ініціалізації персистентного локального сховища оператор потрапляє на головний робочий екран, який за замовчуванням функціонує в режимі розмовного інтерфейсу чат-асистента Астри. Візуальна архітектура простору побудована за принципом раціонального розподілу областей: ліворуч розміщено вертикальну навігаційну панель для швидкого маршрутного перемикання між логічними модулями системи, а центральну частину займає динамічне вікно діалогу. Процес створення нового плану або нагадування ініціюється введенням базової текстової команди у нижнє інтерактивне поле. На рисунку 3.8 продемонстровано первинний стан інтерфейсу користувача під час запуску системи та готовності до діалогу.

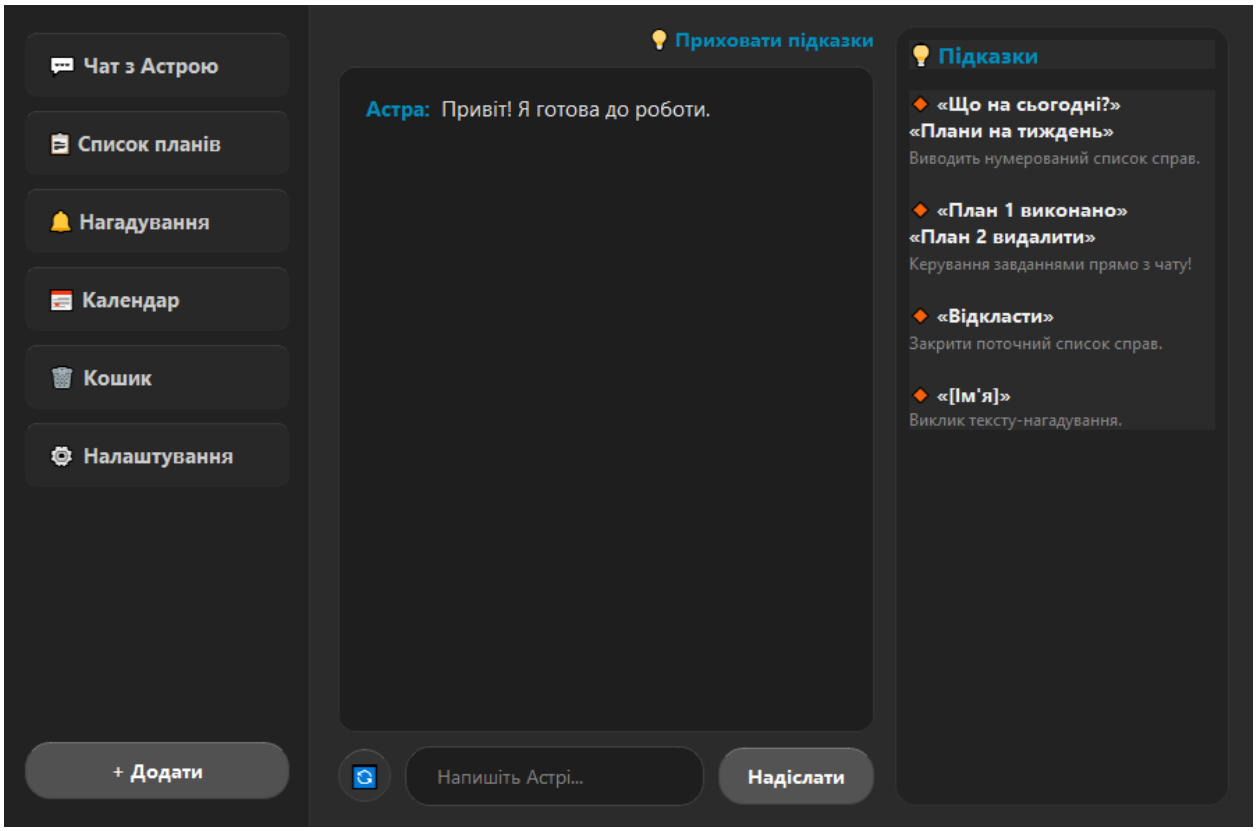

Рисунок 3.8 – Первинний стан інтерфейсу користувача під час ініціалізації робочої сесії

З технічної точки зору, кожна дія користувача у текстовому полі чату обробляється відповідним внутрішнім слотом головного класу, який відповідає за зчитування, очищення та передачу рядка до аналітичного ядра. Для розуміння того, як саме користувацьке введення трансформується у сигнальні потоки графічної оболонки, у лістингу 3.8 наведено фрагмент програмного коду, який керує поведінкою поля введення при натисканні кнопки відправки повідомлення або клавіші Enter.

Лістинг 3.8 – Програмний механізм обробки та очищення поля введення в astra.py

```
def handle_input(self):
    text = self.user_input.text().strip()
```

```
if not text:
    return
self.user_input.clear()
self.append_message("Ви", text)
text_lower = text.lower()
```

Як демонструє програмний код лістингу 3.8, графічна оболонка миттєво реагує на дію оператора, очищуючи віджет user_input та транслюючи введений текст у вікно діалогу із тегом автора. Цей крок є обов'язковим для створення безперервного досвіду спілкування з чат-ботом. Після цього очищений рядок надходить до логічного блоку розгалужень скінченного автомата Астри, який переводить систему у стан покрокового опитування для збору атрибутів завдання, як це проілюстровано на рисунку 3.9.

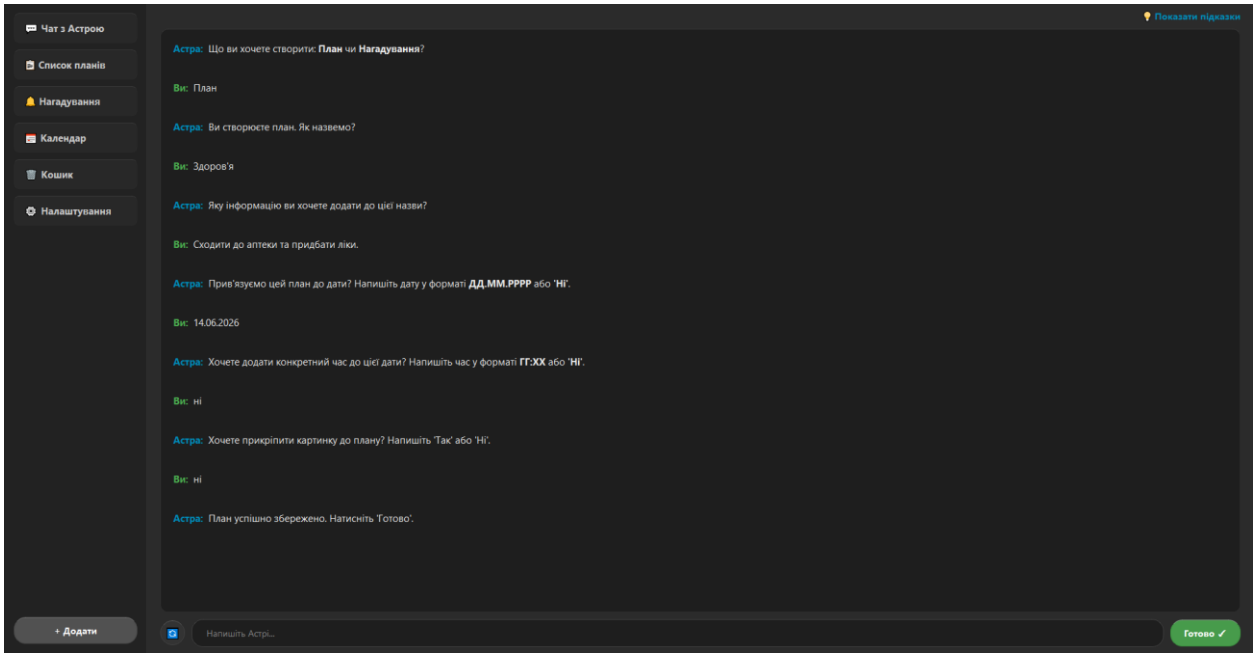

Рисунок 3.9 – Динамічне відображення ходу покрокового опитування у діалоговому вікні

У процесі діалогу скінченний автомат послідовно змінює внутрішній лічильник станів, що фіксує переходи між запитами назви, контенту та хронологічних параметрів. Користувач взаємодіє із системою виключно через текстові відповіді, за винятком етапу інтеграції медіафайлів. Коли Астра запитує про необхідність прикріплення зображення, позитивна відповідь користувача активує нативний системний діалог вибору файлів, повністю інкапсульований у логіці керування станами. Програмне рішення, що забезпечує перехоплення цього кроку та валідацію вибору користувача, наведено у лістингу 3.9.

Лістинг 3.9 – Програмна логіка виклику системного діалогу вибору медіаконтенту в astra.py

```
elif self.creation_step == 22:
        if text_lower in ['так', 'yes', 'д', 'т']:
            file_name, _ = QFileDialog.getOpenFileName(self, "Виберіть картинку", "",
                                                       "Images (*.png *.jpg *.jpeg *.bmp)")
            if file_name:
                self.temp_command_data['image_path'] = file_name
            else:
                self.temp_command_data['image_path'] = None
```

Завдяки реалізації, представленій у лістингу 3.9, користувач отримує можливість візуального вибору графічного додатка за допомогою стандартного вікна операційної системи, що значно спрощує орієнтацію у локальній файловій структурі комп'ютера. Після завершення створення завдання, користувач може переглядати сформовані плани та нагадування у відповідних вкладках або скористатися модулем інтерактивного календаря. Перехід до цього розділу за допомогою бічного меню відкриває календарну матрицю, де дати з активними дедлайнами автоматично маркуються іншим

кольором. При натисканні лівою кнопкою миші на конкретну дату, у нижній частині вікна динамічно рендериться вертикальний список карток завдань, обладнаних ефектами м'яких тіней та мініатюрами зображень, як це детально показано на рисунку 3.10.

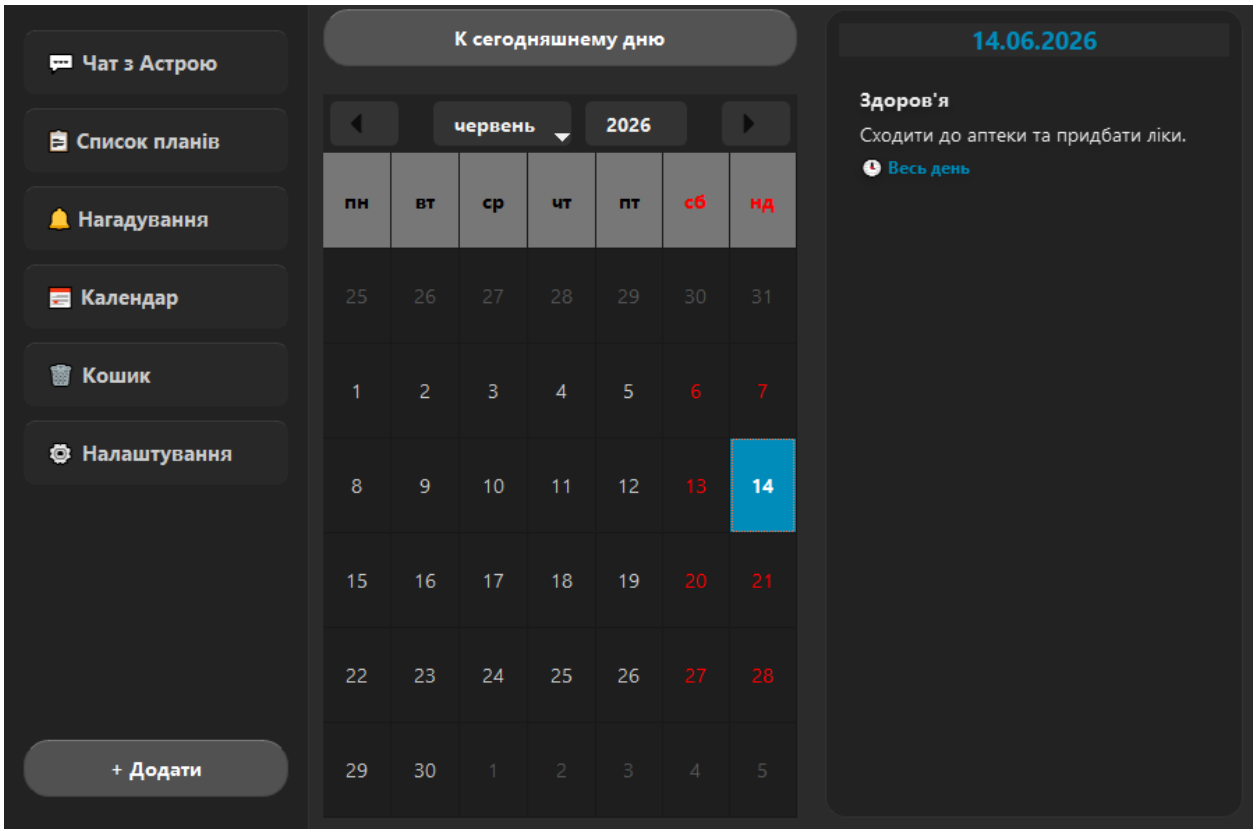

Рисунок 3.10 – Відображення активних завдань в інтерактивному модулі календаря

Для того, щоб забезпечити такий динамічний відгук календарної сітки на кліки користувача, на рівні програмної реалізації було створено спеціальний метод-слухач подій. У лістингу 3.10 наведено фрагмент коду з файлу calendar_page.py, який описує логіку перехоплення вибору дати користувачем та ініціацію процедури повного оновлення списку завдань під актуальний хронологічний контекст.

Лістинг 3.10 – Метод зв'язування подій календаря із процедурою динамічного рендерингу в calendar_page.py

```
def init_ui(self):
    self.calendar = QCalendarWidget()
    self.calendar.setGridVisible(True)
    self.calendar.clicked.connect(self.update_day_plans)
```

Архітектурне рішення, наведене в лістингу 3.10, підключає подію clicked нативного віджета QCalendarWidget до внутрішнього слота update_day_plans. Це дозволяє програмі миттєво зчитувати параметри обраного дня, формувати параметризований SQL-запит до реляційної таблиці за допомогою менеджера бази даних та перемальовувати картки планів без затримок графічного інтерфейсу.

Окремим важливим аспектом взаємодії є можливість персоналізації робочого простору та безпечного очищення сховища. Перехід до панелі налаштувань дозволяє користувачу динамічно змінювати візуальне оформлення. При натисканні кнопки перемикання теми відбувається гаряча заміна каскадних таблиць стилів (QSS) з миттєвим перемальовуванням інтерфейсу між темною та світлою палітрами. Окрім цього, на сторінці розміщено елементи керування для каскадного скидання персистентного стану системи, приклад реалізації яких наведено на рисунку 3.11.

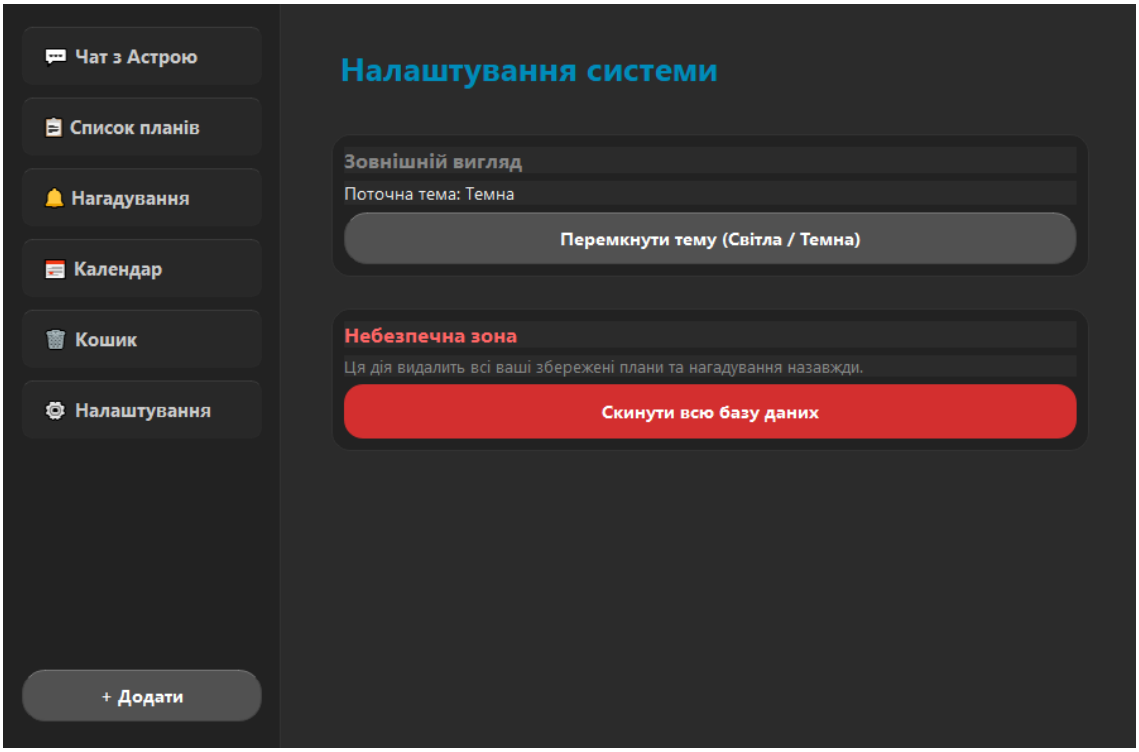

Рисунок 3.11 – Панель глобальних конфігурацій та управління системними параметрами застосунку

Процедура зміни візуального стилю програми користувачем підпорядковується суворому алгоритму динамічної заміни стилістичних констант, що виключає необхідність перезапуску десктопного додатка. Програмне забезпечення, яке відповідає за обробку натискання кнопки перемикання палітри у вкладці конфігурацій, наведено у лістингу 3.11.

Лістинг 3.11 – Програмна логіка гарячої зміни теми оформлення у settings_page.py

```
def toggle_theme(self):
    new_theme = "light" if self.main_app.current_theme == "dark" else "dark"
    self.main_app.change_theme(new_theme)
    status_text = "Світла" if new_theme == "light" else "Темна"
    self.theme_status.setText(f"Поточна тема: {status_text}")
```

Аналіз програмної реалізації з лістингу 3.11 показує, що метод toggle_theme виконує логічну інверсію поточного строкового стану теми програми та делегує виконання каскадної зміни стилів головному класу програми. Після успішного застосування нових параметрів QSS-файлу, текстова мітка динамічно оновлює свій вміст, інформуючи користувача про поточний режим відображення.

У разі, якщо користувач вирішує повністю очистити інформаційну базу або провести остаточну утилізацію контенту, система активує додаткові захисні бар'єри. При натисканні кнопки «Скинути всю базу даних» або спробі повного очищення кошика, бізнес-логіка програми призупиняє виконання операцій та викликає модальне вікно переривання. Програмна реалізація цього процесу безпеки, що захищає користувача від випадкових руйнівних дій, наведена у лістингу 3.12.

Лістинг 3.12 – Метод захисного підтвердження операції очищення персистентного сховища

```
def confirm_reset(self):
    check = QMessageBox.question(
        self, 'Підтвердження видалення',
        "Ви впевнені, що хочете повністю видалити всі плани та нагадування?",
        QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No,
        QMessageBox.StandardButton.No
    )
    if check == QMessageBox.StandardButton.Yes:
        database_manager.clear_db()
        QMessageBox.information(self, "Успіх", "Базу даних успішно очищено.")
```

Як демонструє програмний код лістингу 3.12, модальний віджет QMessageBox.question блокує головне вікно програми та вимагає від

оператора свідомого вибору. Тільки за умови отримання коду прапорця Yes система делегує команду статичному модулю доступу до даних для виконання інструкції повної утилізації записів, що спільно з іншими компонентами графічного інтерфейсу формує надійне, ергономічне та стійке до експлуатаційних помилок програмне середовище тайм-менеджменту.

Важливою функціональною частиною інтерфейсу користувача є спеціалізовані сторінки для роботи з реєстрами довгострокових планів та утилізованих елементів. Перехід до розділу «Плани» за допомогою лівої навігаційної панелі відкриває інтерактивне середовище, призначене для комплексного огляду та пошуку завдань, що не мають суворої часової прив'язки. Ця сторінка обладнана верхньою панеллю швидкої фільтрації, де користувач може миттєво вводити ключові слова для контекстного пошуку, та двома функціональними кнопками перемикання режимів відображення. Залежно від вибору оператора, система перемальовує вміст прокручуваної області, демонструючи або активні завдання, які потребують виконання, або історичний журнал уже завершених цілей. Програмний алгоритм, який забезпечує очищення попереднього графічного контексту та динамічне завантаження карток планів з урахуванням введеного пошукового запиту, наведено у лістингу 3.13.

Лістинг 3.13 – Метод динамічного завантаження та фільтрації карток планів у plans_page.py

```
def load_plans(self, search_query=""):
    if not isinstance(search_query, str): search_query = ""
    for i in reversed(range(self.container_layout.count())):
        widget = self.container_layout.itemAt(i).widget()
        if widget: widget.deleteLater()

    plans = database_manager.get_items_by_type('plan', self.current_view)
```

```
displayed_count = 0
for plan in plans:
    plan_id, name, content, date, image_path = plan
    if search_query.lower() in name.lower() or search_query.lower() in content.lower():
        self.container_layout.addWidget(PlanCard(plan, self, self.current_view))
        displayed_count += 1
```

Як демонструє програмний код лістингу 3.13, під час ініціалізації процедури load_plans система використовує зворотний цикл для безпечного знищення попередніх об'єктів віджетів у контейнері container_layout, запобігаючи витокам оперативної пам'яті. Після цього виконується запит до статичного менеджера бази даних із передачею поточного стану представлення self.current_view. Обробка отриманого масиву супроводжується посимвольним порівнянням рядків у нижньому регістрі, що дозволяє реалізувати гнучку фільтрацію за назвою або змістом планів без перезавантаження інтерфейсу. Візуальне відображення сформованого списку активних довгострокових планів, де кожна картка оснащена індивідуальними елементами управління та підтримує графічні ефекти об’ємних тіней, наведено на рисунку 3.12.

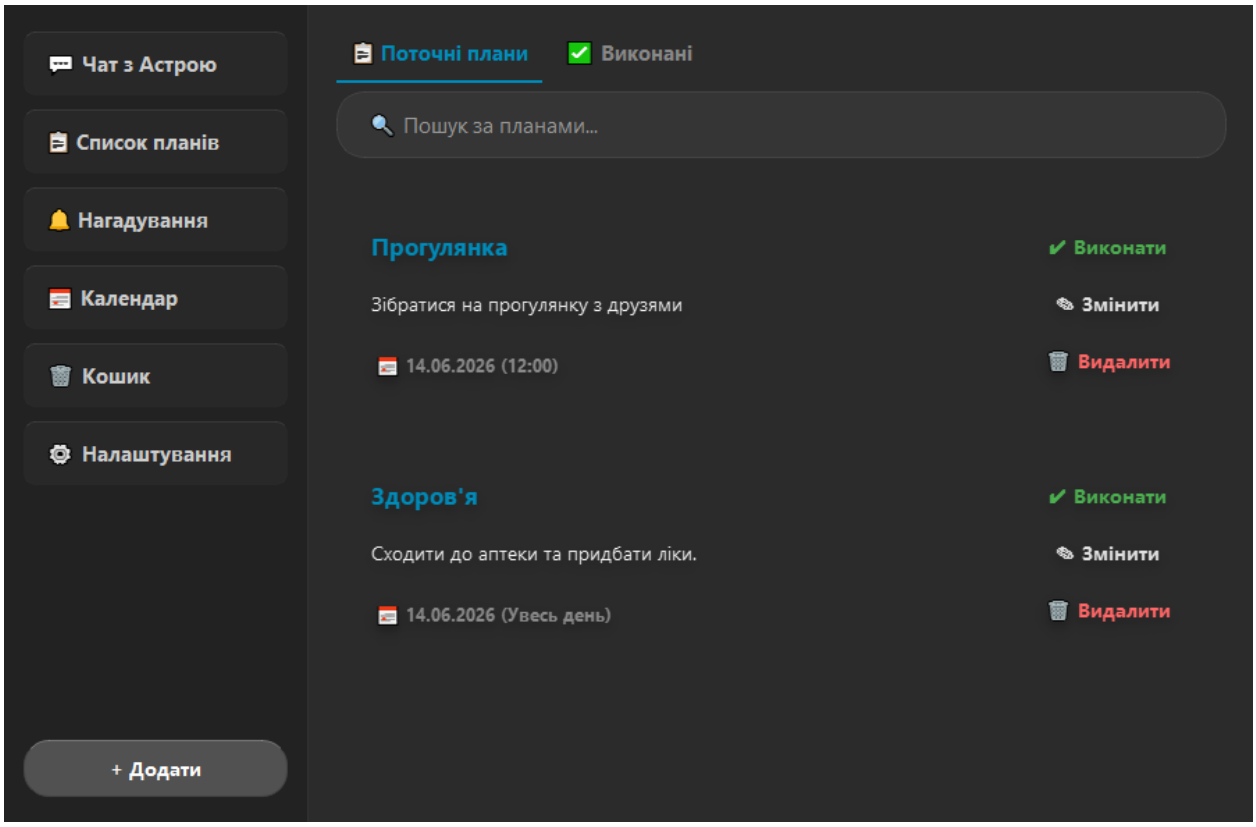

Рисунок 3.12 – Візуальний інтерфейс сторінки відображення активних планів користувача

При перемиканні користувачем фокусу на вкладку виконаних завдань, інтерфейсна логіка програми динамічно модифікує внутрішній стан та ініціює повторний рендеринг. Кнопка «Виконані» змінює свій візуальний статус, а графічне ядро надсилає запит до таблиці бази даних із маркером завершеного життєвого циклу об'єкта. Картки, що виводяться у цій області, автоматично позбавляються кнопок редагування чи маркування успіху, трансформуючись в архівні інформаційні блоки, які підтримують лише операції перегляду прикріплених медіафайлів та логічного видалення. Приклад відображення сторінки завершених планів, що демонструє зафіксовані результати успішної діяльності оператора системи, наведено на рисунку 3.13.

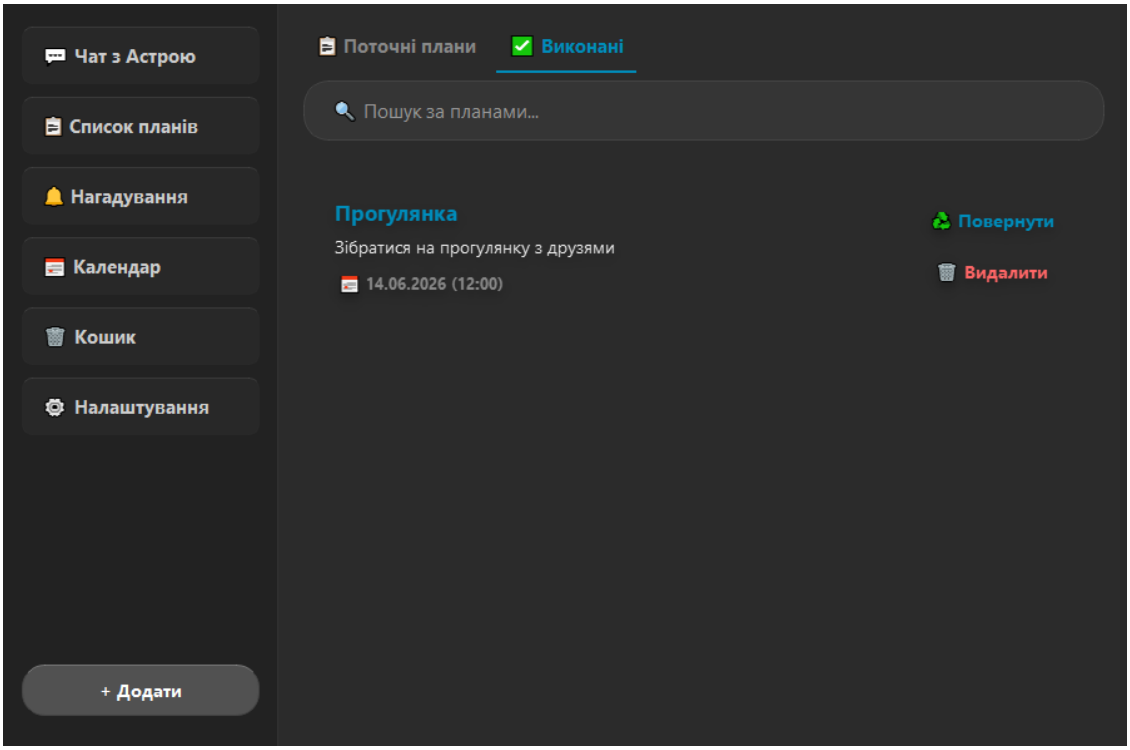

Рисунок 3.13 – Інтерфейс історичного журналу успішно виконаних планів

Окремим логічно ізольованим простором, що забезпечує надійність збереження інформації та реалізацію концепції безпечного тайм-менеджменту, є сторінка «Кошик». Цей модуль призначений для акумуляції всіх утилізованих користувачем об'єктів незалежно від їхнього первинного типу. Візуальне оформлення сторінки підпорядковується загальній стилістиці програми, проте містить у верхній частині критично важливий елемент управління — кнопку повної деструкції сховища. Якщо кошик не містить елементів, програма автоматично рендерить фонове текстове сповіщення з використанням графічних символів для зниження ментального навантаження на оператора. Процедура зчитування видалених елементів та їхнього циклічного відображення в межах сторінки TrashPage наведена у лістингу 3.14.

Лістинг 3.14 – Програмна логіка завантаження утилізованих об'єктів у trash_page.py

```
def load_trash(self, search_query=""):
    if not isinstance(search_query, str): search_query = ""

    for i in reversed(range(self.container_layout.count())):
        w = self.container_layout.itemAt(i).widget()
        if w: w.deleteLater()
    items = database_manager.get_deleted_items()
    displayed_count = 0
    for item in items:
        item_id, name, content, date, item_type, image_path = item
        if search_query.lower() in name.lower() or search_query.lower() in content.lower():
            self.container_layout.addWidget(TrashCard(item, self))
            displayed_count += 1
```

Аналіз лістингу 3.14 підтверджує, що модуль кошика використовує уніфіковані алгоритми фільтрації вхідних потоків, викликаючи метод get_deleted_items. Кожна згенерована картка TrashCard надає користувачеві вибір між двома полярними операціями: миттєвим відновленням первинного статусу завдання у базі даних або ініціацією точкового незворотного видалення. Загальний вигляд сторінки кошика у стані готовності до проведення процедур відновлення або остаточного очищення дискового простору пристрою проілюстровано на рисунку 3.14.

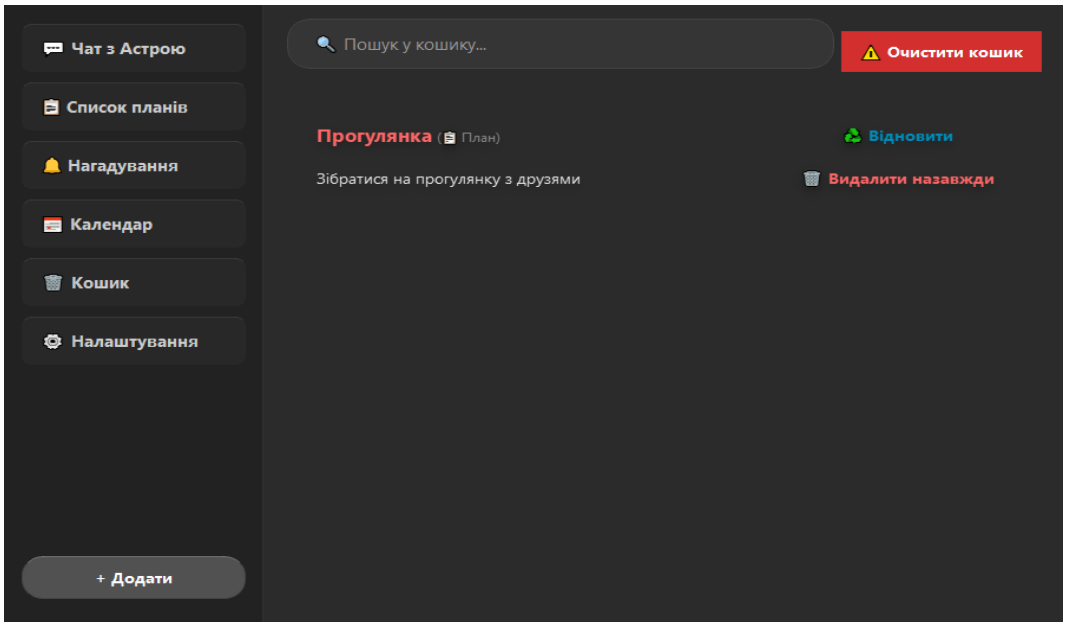

Рисунок 3.14 – Інтерфейс підсистеми двостадійної утилізації контенту та логічного кошика

Висновки до розділу

Третя частина цього проєкту докладно розкриває всі аспекти програмного написання та технічного налаштування настільного додатка, а також містить покрокову інструкцію для майбутніх користувачів. Тут практично пояснено, чому саме комбінація мови Python, бібліотеки PyQt6 та системи SQLite стала фундаментом розробки. Використання цього інструментарію дозволило повною мірою втілити ідеологію Offline-First. Завдяки такому рішенню вдалося досягти високих темпів написання коду, гарантувати роботу софту на різних операційних системах та повністю відмовитися від складного налаштування зовнішніх серверів. Окреслений перелік технічних умов до обладнання доводить, що створений планувальник здатен працювати швидко та стабільно навіть на відносно старих комп'ютерах, споживаючи мінімум обчислювальної потужності.

Архітектура написаного коду цілком спирається на модульний принцип та правила об'єктно-орієнтованого програмування. Застосування віджета

QStackedWidget для внутрішньої навігації дало змогу перетворити класичну програму на сучасний односторінковий застосунок. Цей крок виявився ключовим для того, щоб гармонійно поєднати діалогове вікно комп'ютерного помічника, робочу сітку календаря та стрічку поточних справ у межах одного екрана. Важливим інженерним досягненням стало виокремлення всіх операцій з базою даних у незалежний статичний файл-менеджер. Така ізоляція забезпечила низьку залежність модулів один від одного. Тепер програма легко виконує важкі SQL-запити на фоні, не перериваючи плавної роботи графічного інтерфейсу та не викликаючи зависань картинки.

Окрім візуальної частини, значний обсяг роботи стосувався алгоритмів безпеки, обробки інформації та захисту від критичних збоїв. Перехід на використання виключно параметризованих інструкцій звів до нуля будь-яку небезпеку хакерських SQL-ін'єкцій. Разом із цим, інтеграція алгоритму безпечного стирання контенту та поява спливаючих вікон для підтвердження дій надійно оберігають людину від випадкового видалення власних нотаток. Щоб програма не закривалася аварійно через внутрішні конфлікти, всередині коду діє комплексна система перехоплення виняткових ситуацій. Наприклад, якщо виникає проблема з однаковими назвами завдань (помилка IntegrityError), рушій бази даних не зупиняє роботу софту. Замість цього він м'яко перехоплює збій і виводить на екран зрозуміле попередження для користувача, дозволяючи продовжити планування без перезапуску застосунку.

Написана у цьому ж розділі інструкція з експлуатації наочно показує, наскільки зручним та легким в опануванні вийшов створений комбінований інтерфейс. Зрозумілий опис першого запуску, механіки розмовного створення доручень, додавання картинок та швидкого перемикання колірних тем доводить, що користуватися органайзером зможе людина з будь-яким рівнем комп'ютерної грамотності. Усі перелічені факти та наведені фрагменти вихідного коду підтверджують головну тезу: сконструйований цифровий

помічник є повноцінним, безпечним і абсолютно готовим до щоденного використання продуктом. Він успішно закриває всі технічні вимоги, які висувалися на етапі початкового проєктування цієї системи управління часом.

РОЗДІЛ 4 ОХОРОНА ПРАЦІ

4.1 Організаційно-правові основи забезпечення безпеки праці

Процес інженерного проєктування та безпосереднього написання коду для будь-якої комп'ютерної системи вимагає обов'язкового дотримання правил безпеки на робочому місці. Оскільки створення комплексу на базі Python та PyQt6 передбачало тривалу взаємодію з інтегрованим середовищем розробки, регулярне тестування графічної оболонки та відлагодження алгоритмів, питання збереження здоров'я програміста вийшло на перший план. Правильне облаштування робочої зони дає змогу суттєво зменшити негативний вплив від довгого сидіння за монітором, запобігти швидкій втомлюваності очей та підтримати високий рівень концентрації під час розв'язання складних архітектурних задач.

Загальна стратегія захисту працівників у нашій державі спирається на абсолютний пріоритет людського життя та здоров'я. Будь-яка трудова діяльність у сфері інформаційних технологій має суворо підпорядковуватися базовим нормативним документам. Головним орієнтиром у цій сфері виступає Закон України «Про охорону праці», який гарантує кожному спеціалісту право на безпечні умови виконання своїх професійних обов'язків. Водночас базові сценарії реагування на можливі аварійні чи непередбачувані ситуації в приміщенні, де функціонує обчислювальна техніка, регламентуються положеннями Кодексу цивільного захисту.

Під час безпосереднього створення скриптів та проєктування візуальних інтерфейсів необхідно ретельно контролювати фізичні параметри навколишнього середовища. Зокрема, показники температури, швидкості руху повітря та вологості в кімнаті розробника мають чітко відповідати санітарним нормам мікроклімату, зафіксованим у ДСН 3.3.6.042-99. Щоб уникнути

зайвого навантаження на нервову систему через постійне гудіння вентиляторів системного блоку, рівень акустичного шуму обмежується вимогами ДСН 3.3.6.037-99. Окрім цього, безпечна відстань від дисплея та межі випромінювання електромагнітних полів жорстко контролюються згідно з ДСанПіН 3.3.6.096-2002. Як додатковий критерій надійності, сучасна індустрія розробки програмного забезпечення спирається на міжнародний стандарт управління безпекою ДСТУ ISO 45001:2019, а також на принципи соціальної відповідальності, визначені в ДСТУ ISO 26000:2019.

Проте варто усвідомлювати, що винятково формальне цитування законодавчих актів не здатне автоматично захистити розробника. Справжня безпека під час конструювання та експлуатації десктопних застосунків досягається лише через грамотну практичну організацію робочого процесу. Це означає, що ІТ-фахівець повинен робити регулярні перерви для зняття зорової напруги, правильно налаштовувати ергономіку крісла та освітленість столу, а також постійно перевіряти справність електричної проводки перед увімкненням апаратури. Тільки таке тісне поєднання офіційних правових норм із реальними інженерно-технічними та організаційними заходами формує повноцінно безпечний простір, у якому програміст може працювати максимально продуктивно без жодної шкоди для власного самопочуття.

4.2 Характеристика об’єкта та виявлення потенційних небезпек

У межах цього проєкту створюється настільний органайзер із вбудованим цифровим помічником «Астра». Якщо аналізувати цей процес крізь призму безпеки життєдіяльності, то головним об'єктом вивчення стає робоча зона програміста. Саме тут відбувається архітектурне проєктування, написання скриптів та перевірка готового софту. Крім того, до цієї категорії відноситься

і простір кінцевого споживача, який щодня взаємодіятиме зі встановленою програмою.

Зазвичай розробник облаштовує своє місце в офісі або у відокремленій кімнаті. Головними знаряддями праці виступають стандартні обчислювальні машини: системний блок чи ноутбук, дисплей, а також клавіатура з мишею. Безпосередньо процес програмування проходить у закритих приміщеннях, де мікроклімат підтримується штучно, а денне світло обов'язково доповнюється лампами. Робота ІТ-спеціаліста (сюди входять розробники, тестувальники, системні адміністратори) - це винятково інтелектуальна діяльність. Вона вимагає максимальної концентрації уваги, несе колосальне навантаження на очі та змушує людину годинами сидіти в одній позі.

Специфіка створення десктопних додатків неминуче супроводжується впливом різних негативних чинників, які здатні зашкодити здоров'ю. Зазвичай фахівці класифікують ці загрози залежно від їхньої природи. Під час роботи за комп'ютером на організм найчастіше впливають фізичні, біологічні та психофізіологічні подразники. Що стосується хімічної небезпеки, то у звичайній кімнаті вона майже відсутня, якщо не брати до уваги звичайний побутовий пил чи мізерні викиди озону від працюючого лазерного принтера.

Якщо детальніше розглянути фізичні ризики, то тут на перше місце виходить електромагнітне випромінювання. Його джерелом постійно є матриці моніторів та потужні блоки живлення. До цієї ж категорії варто віднести і відхилення мікроклімату: надто холодне або задушливе повітря, знижену вологість у кімнаті. Серйозною перешкодою для нормальної роботи часто стає погане освітлення, брак сонячних променів або, навпаки, різкі відблиски на склі монітора. Не слід забувати і про акустичне забруднення. Постійне гудіння кулерів всередині системного блока разом із фоновим гомоном вулиці створюють монотонний шум, який швидко втомлює.

Проте найбільшу небезпеку для програміста несуть саме психофізіологічні фактори. Оскільки написання та перевірка коду вимагають

тривалого перебування у сидячому положенні, м'язи спини та шиї зазнають сильного статичного напруження. Паралельно з цим людина стикається з величезним нервовим виснаженням. Необхідність встигнути до дедлайну, постійний пошук прихованих помилок у логіці алгоритмів сильно перевантажують мозок та психіку. Найбільший удар бере на себе зір, адже очі змушені безперервно фокусуватися на дрібних рядках коду та графічних елементах інтерфейсу.

Нарешті, не можна ігнорувати біологічні загрози. Вони ховаються у вигляді патогенних мікроорганізмів (бактерій та вірусів), які дуже швидко накопичуються на кнопках клавіатури, корпусі мишки чи просто на поверхні столу. Це стає реальною проблемою, якщо працівник забуває про елементарні санітарні правила та не очищує свою техніку спеціальними засобами.

Детальну інформацію щодо виявлених небезпечних та шкідливих факторів, джерел їх виникнення та можливих наслідків для здоров'я працівника наведено у таблиці 4.1.

Таблиця 4.1 – Виявлення потенційних небезпек стосовно об'єкту проектування

Сфера проблеми	Потенційна небезпека	Джерело небезпеки	Можливі наслідки
1	Підвищений рівень електромагнітних випромінювань	Монітор, системний блок ПК, периферійне обладнання, кабелі живлення	Зниження імунітету, розлади центральної нервової системи, швидка втомлюваність, головний біль

2	Недостатня освітленість робочої зони, пряма й відбита блискість	Нераціональне розміщення світильників, відблиски від екрана монітора, нестача природного освітлення	Зниження гостроти зору, швидка зорова втомлюваність, різь в очах
3	Статичні фізичні перевантаження	Тривале знаходження у незручній фіксованій позі (сидячи) під час роботи за комп'ютером	Захворювання опорно-рухового апарату (остеохондроз, сколіоз), порушення кровообігу в органах малого тазу, синдром зап'ястного каналу
4	Перенапруження зорових аналізаторів	Тривале безперервне спостереження за екраном монітора під час написання коду та роботи з текстом	Короткозорість, синдром «сухого ока», спазм акомодації
5	Нервово-психічні та емоційні перевантаження, розумове перенапруження	Високе інтелектуальне навантаження, стрес через дедлайни, складні алгоритмічні задачі,	Стрес, безсоння, синдром емоційного вигорання, зниження

		відповідальність за результат	концентрації уваги
6	Підвищений рівень шуму на робочому місці	Робота систем охолодження комп'ютера, сторонні звуки в офісному приміщенні	Зниження уваги, швидка втомлюваність, роздратування
7	Дія патогенних мікроорганізмів	Забруднені поверхні пристроїв введення (клавіатура, миша), недостатня вентиляція приміщення	Ризик виникнення інфекційних та респіраторних захворювань

Аналіз умов праці на об'єкті проектування свідчить про те, що повністю усунути вплив виявлених небезпечних та шкідливих виробничих факторів неможливо через специфіку самої інформаційної діяльності. Проте, застосовуючи комплекс ергономічних, організаційних та інженерно-технічних рішень, цілком можливо суттєво зменшити їхній негативний вплив до прийнятних (гранично допустимих) рівнів. Виходячи з цього, основним завданням для наступного підрозділу є проведення дослідження ризику реалізації виявлених потенційних небезпек та розробка дієвих профілактичних заходів, спрямованих на їх мінімізацію та забезпечення комфортних умов праці інженера-програміста.

4.3 Дослідження ризику реалізації потенційних небезпек на об’єкті проектування та розробка заходів щодо їх попередження

Основою створення безпечного та нешкідливого виробничого середовища для розробника програмного забезпечення та майбутніх користувачів системи є проведення превентивного аналізу потенційних загроз. Для здійснення кількісної та якісної оцінки ризиків реалізації небезпек, ідентифікованих на автоматизованому робочому місці (АРМ) програміста, у даній роботі застосовано метод матричного оцінювання. Цей метод є ефективним інструментом, що дозволяє отримати графічне й текстове описання загроз, класифікувати рівень кожного виявленого фактора та прийняти обґрунтовані інженерно-організаційні рішення щодо його зниження.

Сутність методу полягає у визначенні індексу ризику шляхом перетину двох ключових параметрів: категорії серйозності небезпеки (від I - Катастрофічна до IV - Незначна) та рівня ймовірності її виникнення (від A - Часта до E - Неймовірна). На основі отриманої матриці індекси класифікуються за критеріями припустимості: неприпустимий, небажаний, припустимий із перевіркою та припустимий без перевірки.

Для комплексної візуалізації впливу шкідливих виробничих факторів на об'єкті проектування було побудовано структурно-логічну схему. Вона декомпонує головну подію (негативний вплив на здоров'я при роботі з кодом та інтерфейсами) на конкретні вектори загроз із зазначенням їхньої ймовірності, серйозності наслідків та розрахованого індексу ризику. Результати цього моделювання наведено на рисунку 4.1.

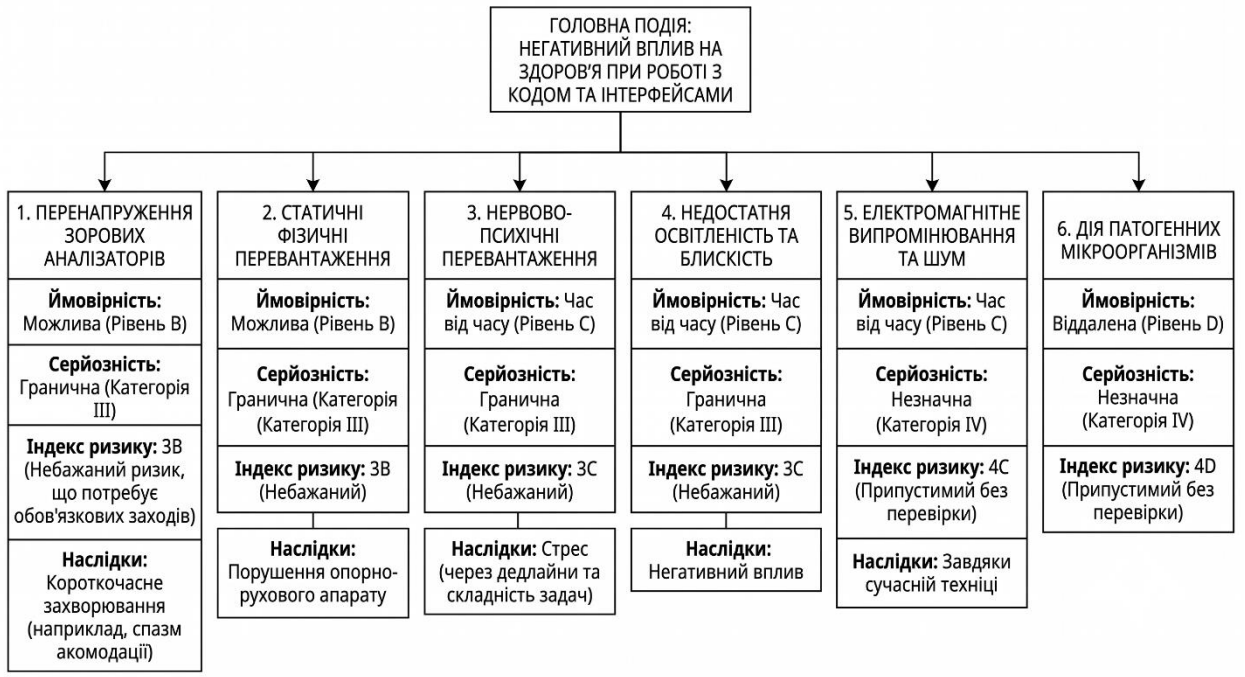

Рисунок 4.1 – Структурно-логічна схема аналізу та оцінювання потенційних ризиків на робочому місці

Аналіз розробленої схеми (рис. 4.1) чітко демонструє диференціацію рівнів небезпеки. Фактори фізичного та біологічного походження, такі як електромагнітне випромінювання, виробничий шум та дія патогенних мікроорганізмів, отримали індекси 4C та 4D. Згідно з критеріями матриці оцінювання, ці значення класифікуються як «Припустимі без перевірки (знехтувані)». Це пояснюється використанням сучасної комп'ютерної техніки з низьким рівнем емісії та віддаленою ймовірністю інфікування в умовах стандартного офісу.

Натомість найбільш критичними векторами, які вимагають негайного реагування, є психофізіологічні фактори та параметри світлового середовища. Перенапруження зорових аналізаторів та статичні фізичні перевантаження отримали індекс 3B, а нервово-психічне перенапруження та недостатня освітленість - індекс 3C. Обидва показники належать до категорії «Небажаний (гранично допустимий) ризик». Для переведення цих загроз у категорію припустимих розроблено багаторівневий комплекс профілактичних заходів:

1. Інженерно-технічні та ергономічні заходи. Для нівелювання ризику статичних перевантажень робоче місце проектується за правилами ергономіки. Висота робочого столу повинна становити 720–750 мм. Працівник забезпечується ергономічним підйомним кріслом із регульованим кутом нахилу спинки та підлокітниками, що знімає напругу з плечового пояса. Екран монітора необхідно розташовувати на відстані 600–700 мм від очей користувача перпендикулярно до лінії погляду. Для нормалізації світлового середовища та усунення прямої блискості (ризик 3С) віконні отвори обладнуються жалюзі, а штучне освітлення забезпечується енергоефективними LED-світильниками із колірною температурою 4000К (нейтральне біле світло).

2. Організаційні заходи та режим праці. Для компенсації нервово-психічних (ризик 3С) та зорових (ризик 3В) навантажень впроваджується суворий режим праці та відпочинку. Встановлюються регламентовані перерви тривалістю 10–15 хвилин кожні 2 години безперервної роботи. Під час перерв обов'язковим є виконання комплексів гімнастики для очей (фокусування на дальніх об'єктах) та легких фізичних вправ для відновлення кровообігу.

3. Програмно-технічні заходи (із залученням об'єкта проектування). Важливо підкреслити, що розроблений десктопний застосунок «Астра» безпосередньо виступає інструментом охорони праці. Його використання для структурування завдань суттєво розвантажує короткочасну пам'ять користувача, ефективно знижуючи рівень нервово-психічного перенапруження та стресу через дедлайни. Крім того, імплементована функція гарячого перемикання на «темну тему» (Dark Mode) є прямою мірою захисту зору: вона суттєво зменшує яскравість екрана та знижує втому очей при роботі в умовах недостатнього зовнішнього освітлення.

Підсумкові заходи щодо зниження рівня ідентифікованих ризиків та очікувані результати від їх впровадження систематизовано у таблиці 4.2.

Таблиця 4.2 – Заходи щодо зниження ризиків

Сфера проблеми	Запропонований захід	Очікуваний результат
Зоровий апарат	Встановлення жалюзі для усунення блискості; використання комбінованого LED-освітлення; застосування «темної» теми розробленого застосунку у вечірній час.	Зниження зорового навантаження, збереження гостроти зору, профілактика синдрому «сухого ока» та спазму акомодації.
Опорно-руховий апарат	Оснащення робочого місця ергономічним кріслом; налаштування висоти столу; проведення фізичної гімнастики під час регламентованих перерв.	Запобігання розвитку остеохондрозу та сколіозу, усунення статичної м'язової напруги, нормалізація кровообігу.
Психоемоційний стан	Використання розробленого планувальника «Астра» для ефективного тайм-менеджменту; дотримання графіку регламентованих перерв.	Зниження рівня робочого стресу, підвищення концентрації уваги, запобігання емоційному та інтелектуальному вигоранню.

Санітарно-гігієнічна сфера	Регулярне вологе прибирання, очищення пристроїв введення антисептиком, наскрізне провітрювання офісного приміщення.	Підтримка імунітету, мінімізація ризику розвитку інфекційних захворювань, покращення загального мікроклімату.

Реалізація запропонованого комплексу рішень дозволить нівелювати вплив шкідливих виробничих факторів та перевести всі виявлені «небажані» ризики у категорію припустимих, забезпечивши високий рівень безпеки та комфорту життєдіяльності.

Висновки до розділу

Четверта частина кваліфікаційного проєкту присвячена розгорнутому вивченню виробничого середовища та розробці засобів безпеки для робочої зони як самого інженера-розробника, так і майбутнього користувача комп'ютерної програми «Астра». Головний орієнтир цього етапу полягав у науковому підтвердженні та практичному впровадженні рішень, котрі здатні заблокувати потенційні загрози під час експлуатації софту, а також послабити дію шкідливих чинників до безпечних санітарних меж.

Під час опрацювання нормативно-правового підґрунтя було детально проаналізовано вітчизняні закони, чинні гігієнічні інструкції та світові стандарти, що встановлюють правила освітленості, межі електромагнітних полів та тепловий комфорт у секторі високих технологій. Це дозволило чітко прописати особливості праці ІТ-фахівців, чия діяльність супроводжується

значною розумовою напругою та багатогодинною нерухомістю за столом. Опираючись на ці документи, вдалося виявити можливі загрози фізичного, психофізіологічного та біологічного характеру. Найбільш небезпечними серед них виявилися втома очей, статичне перевантаження опорно-рухового апарату та ментальний стрес, викликаний необхідністю сувого дотримання встановлених дедлайнів.

Для точного вимірювання та аналізу виявлених чинників у роботі використано інструмент «Матриця оцінювання ризиків» та складено логічну схему взаємодії загроз. Отримані підсумки продемонстрували, що вплив акустичного шуму та магнітних полів від сучасних комп'ютерів не виходить за межі норми. Натомість показники освітлення кімнати та психофізіологічний тиск отримали критичні оціночні індекси 3B і 3C. Подібний результат вказує на гранично допустимий рівень небезпеки, що вимагає негайного розроблення та впровадження профілактичних інженерних кроків.

Щоб нівелювати зафіксовані загрози та суттєво зменшити індекси небезпеки, було сформовано комплекс практичних ергономічних, організаційних та інженерних рекомендацій. Сюди увійшло правильне просторове планування кабінету, яке включає облаштування зручного меблевого куточка та збереження дистанції близько 60–70 сантиметрів від очей до матриці дисплея. Також проведено коригування світлового потоку через встановлення комбінованих світлодіодних ламп і віконних жалюзі. Крім того, передбачено чіткий розпорядок дня, який зобов'язує розробника робити регулярні чвертьгодинні паузи для виконання розминки та спеціальних вправ для відпочинку очей.

Вагому роль у загальній системі профілактики відіграють інструменти захисту, які були закладені безпосередньо у програмний код створюваного додатка. Дослідження підтверджує, що внутрішня логіка органайзера «Астра» помітно зменшує ментальну втому оператора, оскільки софт чітко розбиває завдання та звільняє людину від необхідності утримувати купу інформації в

голові. Також вбудована опція миттєвої активації нічного режиму оформлення (Dark Mode) зменшує світловий потік від монітора на фізичному рівні. Це є чудовим способом уберегти сітківку ока від перенапруження під час роботи в сутінках або за слабкого кімнатного світла.

Перевірка дієвості цих кроків доводить, що застосування запропонованих правил дозволяє повністю заблокувати шкідливі виробничі подразники, трансформувавши небезпечні показники у цілком прийнятні рамки. Такий крок створює повноцінно захищене та максимально зручне середовище для тривалої розумової роботи. Як наслідок, покращення умов праці підвищує стабільність взаємодії в контурі «людина-машина» та забезпечує високу особисту продуктивність під час розв'язання повсякденних та професійних завдань.

ВИСНОВКИ

Підсумком цього кваліфікаційного проєкту стало створення повноцінної настільної програми для планування справ, яка містить вбудованого віртуального помічника «Астра». Аналіз ринку на початкових етапах чітко показав, що користувачам бракує софту, який би об'єднував звичайний візуальний календар та розумний текстовий чат. Фундаментом розробки стала архітектурна парадигма Offline-First. Вона гарантує абсолютну незалежність програми від інтернету та повністю усуває затримки обміну даними. Найголовніше те, що такий підхід забезпечує стовідсоткову конфіденційність особистих записів, адже всі обчислення і файли зберігаються виключно на локальному диску комп'ютера.

Під час інженерного конструювання системи вдалося побудувати оптимізовану базу даних за допомогою інтегрованої СКБД SQLite. Її реляційна структура надійно зв'язує між собою різноманітні елементи: довгострокові цілі, календарні позначки, швидкі сповіщення та прикріплені картинки. Щоб захистити інформацію від випадкового знищення, на рівні бази діє алгоритм м'якого видалення. Замість безповоротного стирання, програма переносить непотрібні записи до спеціального логічного кошика. Це робить застосунок значно стійкішим до помилкових дій оператора і дає змогу гнучко керувати життєвим циклом кожної нотатки.

Графічну оболонку побудовано за допомогою інструментів кросплатформної бібліотеки PyQt6. Завдяки цьому вдалося втілити концепцію односторінкового додатка (Single-Page Application) в умовах десктопного середовища. Внутрішній модуль маршрутизації миттєво перемикає активні вікна без запуску нових фонових процесів, що дуже суттєво заощаджує оперативну пам'ять комп'ютера. Додатково було написано власні графічні класи, які дозволили додати клікабельні текстові блоки та налаштувати якісне

згладжування завантажених фотографій. Усе це разом сформувало сучасний, інтуїтивний та візуально приємний інтерфейс.

Серцем створеного гібридного інтерфейсу є розмовний асистент. Щоб його реакції були безпомилковими, логіку діалогу побудовано на основі математичної моделі детермінованого скінченного автомата. Цей механізм покроково запитує в людини всі необхідні параметри для створення завдання, не пропускаючи жодної важливої деталі. Висока стабільність роботи досягається завдяки використанню виключно параметризованих SQL-запитів, які надійно блокують будь-які спроби хакерських ін'єкцій. Крім того, у коді працює багаторівнева система перехоплення збоїв. Якщо виникає конфлікт ідентифікаторів, програма не зависає, а м'яко виводить на екран зрозумілу текстову підказку.

Дотримуючись стандартів проєктування програмного забезпечення, окрему увагу приділено безпеці робочого місця ІТ-фахівця та майбутнього користувача. Застосувавши метод оцінювання ризиків, вдалося визначити найбільші загрози: перенапруження зору, втому хребта від довгого сидіння та ментальний стрес. Для протидії цим факторам сформовано список практичних порад щодо облаштування столу та правильного освітлення. Важливо підкреслити, що сам додаток «Астра» також працює як інструмент охорони здоров'я. Передача рутини віртуальному помічнику зменшує нервове виснаження, а наявність темної теми оформлення (Dark Mode) береже очі від зайвого світіння монітора у вечірній час.

Готовий програмний комплекс успішно пройшов усі стадії тестування та практичної перевірки. Він цілком відповідає початковим вимогам, працює без критичних збоїв на різних операційних системах і повністю готовий до повсякденного використання. Закладена архітектура залишає широкий простір для подальших оновлень. У майбутньому проєкт можна суттєво вдосконалити, якщо підключити до нього локальні мовні нейромережі (Large Language Models). Це дозволить асистенту набагато глибше розуміти зміст

написаних фраз, вивівши комунікацію на новий рівень інтелекту без жодної шкоди для приватності даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Воротнікова З. Є. Огляд баз даних при розробці програмного забезпечення для різних операційних систем. Вісник Приазовського державного технічного університету. Серія: Технічні науки. 2025. Вип. 51.
2. Грицюк Ю. І. Проєктування інтерфейсу користувача : навч. посіб. Львів : Вид-во ЛТЕУ, 2021. 308 с.
3. Жуковський С. С., Вакалюк Т. А. Об’єктно-орієнтоване програмування мовою Python : навч. посіб. Житомир : Вид-во ЖДУ ім. І. Франка, 2019. 140 с.
4. Піскунова Л. Е., Прилипко В. А., Зубок Т. О. Безпека життєдіяльності : підручник. Київ : Академія, 2014. 224 с.
5. Санітарні норми мікроклімату виробничих приміщень : ДСН 3.3.6.042-99. [Чинний від 1999-12-01]. Київ : МОЗ України, 1999. 21 с.
6. Малишева В. В. Методичні вказівки до виконання розділу «Охорона праці» в дипломних роботах бакалаврів (для галузі знань 12 «Інформаційні технології»). Харків : ХНУМГ ім. О.М. Бекетова, 2025. 14 с.
7. Природне і штучне освітлення : ДБН В.2.5-28:2018. [Чинний від 2019-03-01]. Київ : Мінрегіон України, 2018. 94 с.
8. Про охорону праці : Закон України від 14.10.1992 № 2694-XII. *Офіційний вебпортал парламентів України*. URL: https://zakon.rada.gov.ua/laws/show/2694-12 (дата звернення: 10.05.2026).
9. Celko J. Joe Celko's SQL for Smarties: Advanced SQL Programming. 5th ed. Burlington : Morgan Kaufmann, 2014. 848 p.
10. Fitzpatrick M. Create GUI Applications with Python & Qt6 (PyQt6 Edition). 5th ed. Leanpub, 2021. 811 p.

11. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Reading : Addison-Wesley, 1994. 395 p.

12. Що таке система управління охороною праці на підприємстві? – Режим доступу: https://globynska-gromada.gov.ua/news/1721299162/.

13. Hipp R. SQLite Documentation. *SQLite Consortium*, 2024. URL: https://www.sqlite.org/docs.html (дата звернення: 05.05.2026).

14. Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями : Наказ Мінсоцполітики від 14.02.2018 № 207. *Офіційний вебпортал парламентів України*. URL: https://zakon.rada.gov.ua/laws/show/z0508-18 (дата звернення: 12.05.2026).

15. Hopcroft J. E., Motwani R., Ullman J. D. Introduction to Automata Theory, Languages, and Computation. 3rd ed. Harlow : Pearson, 2006. 560 p.

16. Python Core Team. Python 3.10.12 Documentation. *Python Software Foundation*, 2023. URL: https://docs.python.org/3/ (дата звернення: 02.05.2026).

17. Riverbank Computing Limited. PyQt6 Reference Guide. *Riverbank Computing*, 2024. URL: https://www.riverbankcomputing.com/static/Docs/PyQt6/ (дата звернення: 08.05.2026).

18. Shneiderman B., Plaisant C. Designing the User Interface: Strategies for Effective Human-Computer Interaction. 6th ed. Boston : Pearson, 2016. 624 p.

ДОДАТОК А UML-ДІАГРАМА ІНТЕРФЕЙСІВ КОРИСТУВАЧА

Назва класу	Призначення в архітектурі	Основні атрибути (змінні)	Основні методи (функції)
AstraApp	Головний клас застосунку. Координує маршрутизацію між вікнами, зберігає глобальні налаштування та керує станами діалогу віртуального асистента.	stacked_widget current_theme creation_step temp_command_data	init_ui() handle_input() append_message() change_theme()
PlansPage	Модуль для відображення, контекстного пошуку та фільтрації списку довгострокових планів (активних та виконаних).	current_view container_layout search_input	init_ui() load_plans()
CalendarPage	Модуль інтерактивного календаря. Здійснює моніторинг бази даних для підсвічування дат і виведення завдань на обраний день.	calendar layout	init_ui() update_day_plans()
TrashPage	Модуль логічного кошика. Відповідає за акумуляцію утилізованих об'єктів, їхнє відновлення або	container_layout search_input	init_ui() load_trash() empty_trash()

	безповоротне видалення з пам'яті.		
SettingsPage	Модуль глобальних конфігурацій. Надає інтерфейс для перемикання тем оформлення та каскадного очищення бази даних.	main_app theme_status	init_ui() toggle_theme() confirm_reset()
PlanCard	Динамічний віджет. Використовується для графічного рендерингу індивідуального завдання у списку (містить назву, опис, кнопки управління).	item_id status	init_ui() change_status()
ClickableLabel	Кастомний клас, що розширює стандартний текстовий віджет підтримкою подій кліку мишею для інтерактивних елементів (наприклад, картинок).	clicked *(pyqtSignal)*	mousePressEvent
ImageViewerDialog	Модальне вікно для попереднього перегляду прикріплених медіафайлів із застосуванням алгоритмів гладкого масштабування.	pixmap	init_ui() load_image()