Галицький фаховий коледж імені В’ячеслава Чорновола

відділення комп’ютерних технологій

циклова комісія інформатики та комп’ютерних дисциплін

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач відділення

комп’ютерних технологій

Наталія СТЕФУРАК /_____/

підпис

«___» _____ 202__ р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи

освітньо-професійного ступеня «фаховий молодший бакалавр»

зі спеціальності 122 «Комп’ютерні науки»

на тему: «Мобільний застосунок для управління особистими завданнями»

Студент групи КН-41	Андрій ВЕЛЕЩУК	_____ (підпис)
Керівник роботи	Надія ГАВРИШКІВ	_____ (підпис)
Консультанти:		
з техніко-економічного обґрунтування	Любов МЕЛЕНЧУК	_____ (підпис)
нормоконтролер	Наталія КУЛЬЧИНСЬКА	_____ (підпис)

Тернопіль – 2025

Галицький фаховий коледж імені В’ячеслава Чорновола

відділення комп’ютерних технологій

циклова комісія інформатики та комп’ютерних дисциплін

ЗАТВЕРДЖУЮ

Завідувач відділення комп’ютерних технологій

Наталія СТЕФУРАК /_____/

підпис

«___» _____ 2024 р.

ЗАВДАННЯ

на кваліфікаційну роботу

на здобуття освітньо-професійного ступеня «фаховий молодший бакалавр»

_____студенту Велещуку Андрію Руслановичу_____

(прізвище, ім’я та по-батькові студента)

1. Тема роботи мобільний застосунок для управління особистими завданнями затверджена наказом по коледжу від “25” листопада 2024р., №253а-н

2. Термін здачі студентом завершеної роботи “__” _____ 2025 р.

3. Вихідні дані до роботи результати дослідження мобільних застосунків для управління особистими завданнями, аналіз ринку додатків за цією тематикою та технологій їх реалізацій _____

4. Перелік питань, які повинні бути розроблені:

а) основна частина Аналіз предметної області та постановка завдань. Проєктування системи. Реалізація та тестування системи._____

б) техніко-економічнеобгрунтування Аналіз ринку. Розрахунок витрат на проєктування. Обґрунтування необхідності розробки._____

5. Перелік графічного матеріалу діаграма класів, ER-діаграма, діаграма послідовності_____

Розділ	Консультанти	Підпис, дата	
		Завдання видано	Завдання прийнято
3 техніко-економічного обґрунтування	Любов МЕЛЕНЧУК		

КАЛЕНДАРНИЙ ПЛАН

виконання кваліфікаційної роботи

№ п/п	Найменування етапу	Терміни	
		початку	завершення
1	Пошук теми роботи, вивчення вимог для кваліфікаційної роботи	16.11.2024	25.11.2024
2	Аналіз існуючих рішень та визначення їхніх переваг і недоліків	26.11.2024	06.12.2024
3	Вивчення технологій Flutter, Dart, Hive та інструментів реалізації.	07.12.2024	10.12.2024
4	Формування архітектури застосунку, визначення функціональних вимог.	11.12.2024	20.12.2024
5	Підготовка та налаштування програмного середовища	21.12.2024	07.01.2025
6	Проєктування структури застосунку, бази даних та інтерфейсу користувача	08.01.2025	01.02.2025
7	Реалізація основної логіки застосунку	02.02.2025	25.04.2025
8	Оптимізація коду та вдосконалення окремих модулів	26.04.2025	05.05.2025
9	Тестування застосунку та усунення помилок	06.05.2025	09.05.2025
10	Аналіз ринку збуту, визначення користувачів і оцінка економічної доцільності	10.05.2025	15.05.2025
11	Оформлення пояснювальної записки.	16.05.2025	11.06.2025
12	Попередній захист кваліфікаційної роботи	13.06.2025	13.06.2025
13	Підготовка до захисту кваліфікаційної роботи	14.06.2025	23.06.2025
14	Захист кваліфікаційної роботи	24.06.2025	24.06.2025

Дата видачі “___” _____ 202_ р. Керівник _____/ Гавришків Н.Г.

Завдання прийняв до виконання _____/ Велещук А.Р.

Реферат

Кваліфікаційна робота. Велещук Андрій. Застосунок для управління особистими завданнями. Пояснювальна записка містить 69 сторінок, 5 таблиць, 32 рисунки, 3 додатки, 7 джерел посилання.

Метою роботи є створення To-Do застосунку з можливістю локального зберігання інформації, управління завданнями та отримання push-нагадувань.

У ході написання кваліфікаційної роботи буде проаналізовано існуючі рішення мобільних застосунків для керування завданнями, визначено їх основні переваги та недоліки. Буде створено макети інтерфейсу та модель локальної бази даних із урахуванням вимог до зберігання та обробки особистих завдань.

Для розробки мобільного застосунку буде використано середовище Visual Studio Code. Основна мова програмування – Dart, основний фреймворк – Flutter. Для зберігання даних обрано Hive як локальну NoSQL базу. Для реалізації функціоналу нагадувань використано бібліотеку flutter_local_notifications. Інтерфейс реалізовано з використанням елементів Material Design, а оновлення вмісту забезпечено засобами реактивного програмування (ValueListenableBuilder). Застосунок орієнтований на Android-платформу з можливістю подальшої кросплатформної адаптації.

МОБІЛЬНИЙ ЗАСТОСУНОК, TO-DO, FLUTTER, HIVE, DART, ЗАВДАННЯ, ОСОБИСТИЙ ЧАС, NOTIFICATION, ER-ДІАГРАМА, ДІАГРАМА КЛАСІВ, ДІАГРАМА ПОСЛІДОВНОСТІ.

Abstract

Qualification work. Andrii Veleshchuk. An application for managing personal tasks. The explanatory note contains 69 pages, 5 tables, 32 figures, 3 appendices, 7 references.

The purpose of the work is to create a to-do application with the ability to store information locally, manage tasks, and receive push reminders.

In the course of writing the qualification work, existing mobile application solutions for task management will be analyzed, their main advantages and disadvantages will be identified. Interface layouts and a model of a local database will be created, taking into account the requirements for storing and processing personal tasks.

The Visual Studio Code environment will be used to develop the mobile application. The main programming language is Dart, and the main framework is Flutter. Hive was chosen as a local NoSQL database for data storage. The flutter_local_notifications library was used to implement the notification functionality. The interface is implemented using Material Design elements, and content updates are provided by means of reactive programming (ValueListenableBuilder). The application is focused on the Android platform with the possibility of further cross-platform adaptation.

MOBILE APPLICATION, TO-DO, FLUTTER, HIVE, DART, TASKS, PERSONAL TIME, NOTIFICATION, ER-DIAGRAM, CLASS DIAGRAM, SEQUENCE DIAGRAM.

ЗМІСТ

Вступ 7

1 Аналіз предметної області та постановка завдань 8

1.1 Опис предметної області 8

1.2 Аналіз наявних рішень 9

1.3 Аналіз вимог до програмного засобу та постановка завдання 16

2 Проєктування системи застосунку керування особистими завданнями . 17

2.1 Проєктування структури системи 17

2.2 Аргументація вибору засобів та середовища 19

2.3 Проєктування бази даних 20

2.4 Проєктування інтерфейсу застосунку 23

3 Реалізація та тестування системи 29

3.1 Підготовка програмного середовища 29

3.2 Реалізація основних функцій застосунку 31

3.3 Створення та налаштування бази даних 41

3.4 Реалізація логіки нагадування 44

3.5 Реалізація інтерфейсу користувача 47

3.6 Тестування застосунку 54

4 Техніко-економічне обгрунтування 60

4.1 Аналіз ринку збуту продукту 60

4.2 Розрахункова частина 62

4.3 Обгрунтування необхідності розробки 66

Висновки 68

Перелік джерел посилання 69

Додатки 70

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

AVD – Android Virtual Device.

CRUD – Create, Read, Update, Delete (Створити, прочитати, оновити, видалити).

IDE – Integrated Development Environment.

SDK – Software Development Kit.

UI – User Interface.

UX – User Experience.

YAML – Yet Another Markup Language.

ВСТУП

Стрімкий розвиток цифрових технологій та зростання темпу життя, створюють потребу в засобах, які допомагають ефективно організовувати повсякденну діяльність. Одним із таких інструментів є To-Do застосунки [1] – мобільні програми для створення та керування списками завдань. Вони дозволяють користувачам структурувати свої справи, планувати час, фіксувати важливі події та контролювати виконання завдань у зручному та доступному форматі. Подібні рішення активно використовуються в особистому, освітньому та професійному середовищі, сприяючи підвищенню продуктивності й дисципліни.

Метою даної дипломної роботи є розробка повноцінного мобільного застосунку для керування завданнями, що забезпечуватиме користувачу інтуїтивно зрозумілий інтерфейс, зручну взаємодію зі списками справ, стабільну систему збереження даних, а також функціональну систему нагадувань.

Кваліфікаційна робота буде охоплювати повний процес розробки програмного продукту: від аналізу користувацьких потреб до тестування готового рішення. У процесі роботи розглядатимуться сучасні підходи до проєктування мобільних інтерфейсів, принципи організації локального зберігання даних та методи оптимізації продуктивності застосунків. Отримані результати будуть корисними для подальших досліджень у сфері розробки користувацьких додатків та впровадження інноваційних рішень в галузі персональної організації часу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАНЬ

1.1 Опис предметної області

В умовах сьогодення здатність ефективно організовувати справи та керувати завданнями стала критичною для професійного успіху та особистісного зростання. З кожним днем обсяг інформації, який потрібно обробляти, лише збільшується, а багатозадачність стає невід'ємною частиною життя більшості людей. Потужний інформаційний потік, велика кількість термінових справ та постійна необхідність жонглювати різноманітними завданнями часто породжують безлад, пропуск важливих термінів та неоптимальне використання часового ресурсу. У результаті люди стикаються з перевантаженням, неефективним плануванням, зниженням концентрації та навіть професійним вигоранням. Така ситуація неминуче призводить до втрати внутрішньої мотивації, падіння продуктивності та надмірного психологічного навантаження, що негативно впливає як на професійну діяльність, так і на особисте життя.

Мобільний To-Do застосунок розробляється для того, щоб спростити керування завданнями та зробити процес організації справ більш зрозумілим і доступним. У ньому можна легко створювати, редагувати та виконувати завдання без зайвих складнощів. Інтерфейс інтуїтивно простий, тому користуватися застосунком буде зручно навіть тим, хто раніше не працював із подібними інструментами. Вбудовані нагадування допоможуть не забувати про важливі дедлайни, а зручні функції планування сприятимуть ефективному розподілу часу. Це дозволить уникати плутанини у справах і підвищити продуктивність без зайвого стресу.

Основна мета To-Do застосунку – не лише допомогти користувачам підвищити продуктивність, а й створити гнучку та зручну систему управління завданнями, яка дозволить легко організовувати свій робочий процес, уникати зайвого стресу та ефективно досягати поставлених цілей. Завдяки поєднанню простоти, зручності та широкого функціоналу, застосунок стане незамінним

інструментом для тих, хто прагне впорядкувати свої справи, покращити самодисципліну та досягти балансу між роботою та особистим життям.

1.2 Аналіз наявних рішень

Для аналізу наявних рішень у сфері управління завданнями було розглянуто популярні застосунки, які вже використовуються користувачами. Важливими критеріями є оцінка їх функціоналу, інтерфейсу, переваг та недоліків, щоб визначити, які аспекти можна покращити в To-Do застосунку.

TickTick – це популярний планувальник завдань, який забезпечує користувачам можливість ефективного управління своїми справами. Однією з головних особливостей є вбудований Pomodoro-таймер, що допомагає підвищити концентрацію та ефективність роботи. Застосунок доступний на різних платформах, включаючи iOS, Android, Windows, macOS та веб-версію, що робить його зручним для синхронізації між пристроями. Користувачі можуть створювати списки завдань, використовувати мітки, підзадачі та нагадування для більш зручного планування (рис.1.1).

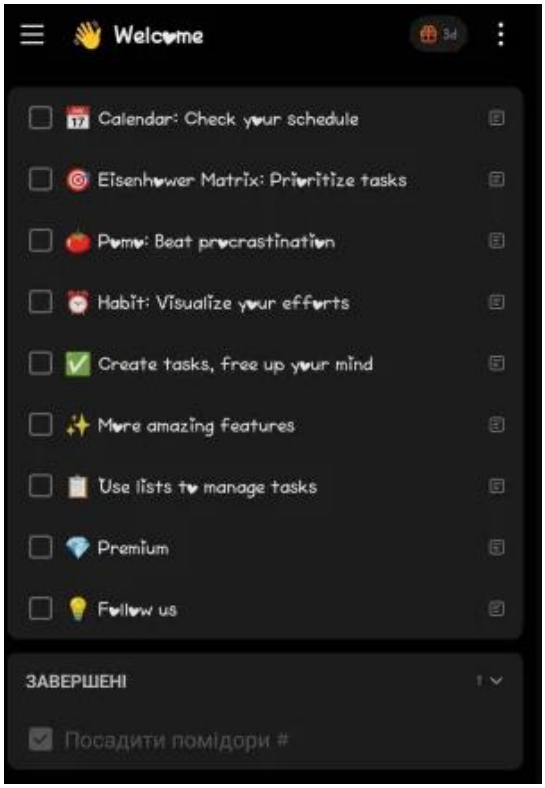

Рисунок 1.1 – Списки завдань

Також однією з переваг цього застосунку є інтуїтивно простий інтерфейс, а також великий спектр пунктів для кастомізації інтерфейсу під користувача, що зображено на рисунку 1.2.

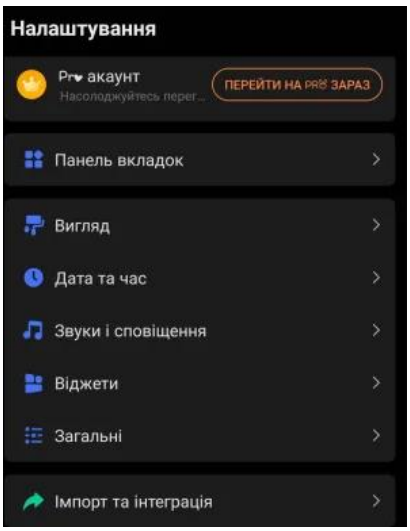

Рисунок 1.2 – Кастомізація під користувача

TickTick також підтримує інтеграцію з календарем, що дозволяє легко стежити за дедлайнами. Проте, одним з недоліків є те, що безкоштовна версія має певні обмеження, зокрема кількість списків і завдань, що може бути незручним для активних користувачів, для використання повного функціоналу необхідно придбати преміум підписку, що показано на рисунку 1.3.

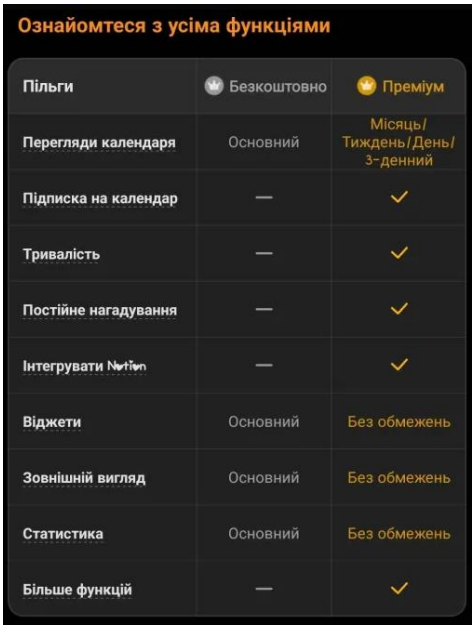

Рисунок 1.3 – Преміум підписка

Проаналізувавши цей застосунок можна окреслити його інтуїтивно простий інтерфейс, зручність, а також достатній функціонал. Однак, одразу після завантаження повним функціоналом користуватись не вийде. В застосунку є як і свої переваги, так і свої недоліки. Він є хорошим прикладом застосунку такого жанру.

Any.do (рис.1.4) – це один із найпопулярніших застосунків для управління завданнями, який поєднує класичний список справ із розширеним функціоналом. Однією з головних переваг є інтеграція з голосовими помічниками (Google Assistant, Alexa), що дозволяє швидко створювати завдання за допомогою голосових команд.

Застосунок доступний на iOS, Android, Windows, macOS та у веб-версії, що забезпечує синхронізацію між пристроями. Інтерфейс Any.do простий та мінімалістичний, що робить його зручним для швидкого доступу до списків справ.

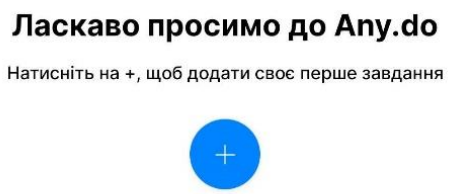

Рисунок 1.4 – Застосунок Any.do

Однією з ключових функцій є інтеграція з календарем (рис. 1.5), яка дозволяє бачити всі свої завдання у вигляді графіка. Це допомагає краще розподіляти час і уникати перевантаження робочого дня.

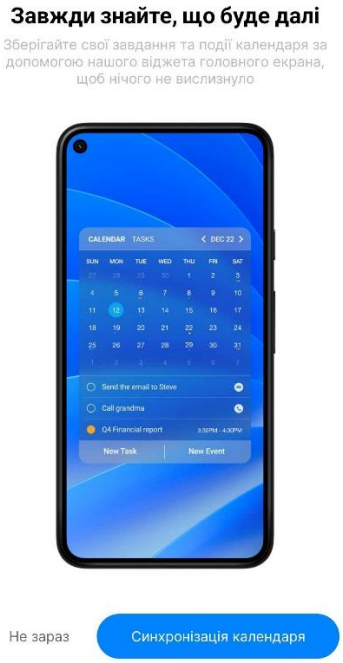

Рисунок 1.5 – Інтеграція з календарем

Попри зручність, у безкоштовній версії застосунку є певні обмеження. Наприклад, для використання повторюваних завдань, налаштовуваних нагадувань та спільної роботи необхідно оформити преміум-підписку (рис. 1.6).

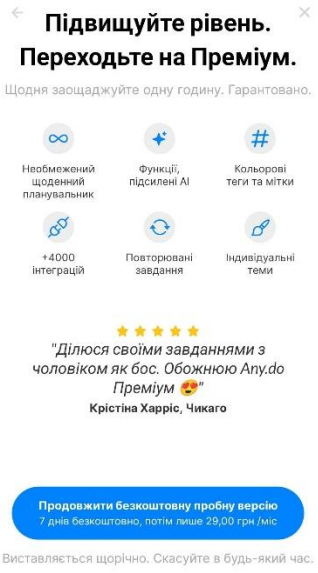

Рисунок 1.6 – Преміум підписка

Проаналізувавши цей застосунок, основними недоліками можна виділити обмежений безкоштовний функціонал – користувачам доводиться платити за можливість використовувати повторювані завдання, розширені нагадування та роботу в команді, Низьку кастомізацію – не можна змінювати теми чи гнучко налаштовувати інтерфейс, повільну синхронізацію – іноді дані не одразу оновлюються між пристроями, обмежену кількість підзадач – у безкоштовній версії можна створювати лише прості списки без глибокої ієрархії, рекламу в безкоштовній версії – може дратувати користувачів.

Загалом, Any.do – це чудовий вибір для тих, хто шукає простий і зручний планувальник завдань, проте для розширеного використання доведеться сплатити за преміум-функції.

Todoist (рис 1.7) – це ще один потужний застосунок для управління завданнями, який відзначається мінімалістичним дизайном та широкими можливостями кастомізації. Основною перевагою є система пріоритетів, що дозволяє розподіляти завдання за рівнем важливості.

Інтерфейс застосунку зручний та не перевантажений зайвими елементами. У Todoist можна створювати не лише прості завдання, а й комплексні проєкти, що робить його чудовим вибором для командної роботи.

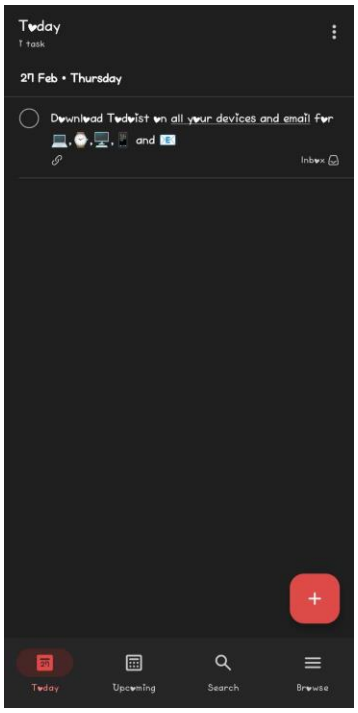

Рисунок 1.7 – Застосунок Todoist

Застосунок також підтримує інтеграцію з популярними сервісами, такими як Google Calendar (рис 1.8), Slack і Dropbox, що дозволяє легко синхронізувати завдання та отримувати нагадування у потрібний момент.

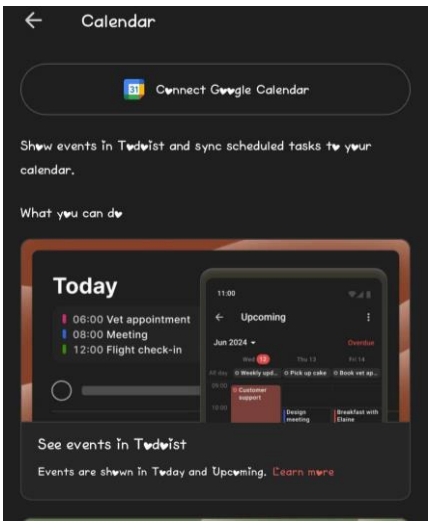

Рисунок 1.8 – Інтеграція з Google Calendar

Як і у випадку з Any.do, у безкоштовній версії є певні обмеження: користувачі не мають доступу до розширених аналітичних звітів, фільтрів і додаткових опцій нагадувань. Основні переваги преміум версії зображено на рисунку 1.9.

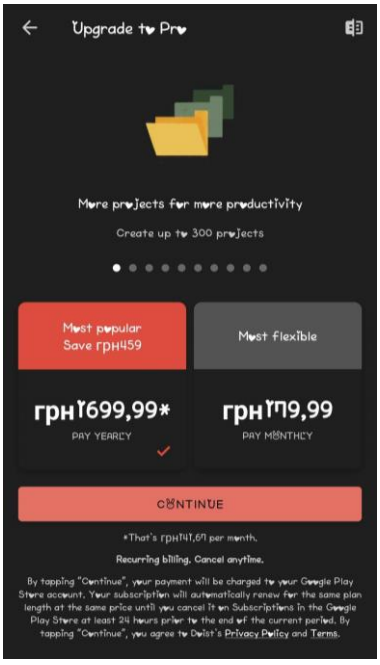

Рисунок 1.9 – Переваги преміум версії

Після аналізу цього застосунку можна виділити кілька основних недоліків, наприклад висока ціна преміум-версії – багато корисних функцій, таких як розширені фільтри, нагадування та аналітика, доступні лише за передплатою, складний інтерфейс для новачків – деякі користувачі скаржаться, що система проєктів і підзадач не дуже інтуїтивна, немає календарного режиму у безкоштовній версії – відсутність повноцінного перегляду завдань у форматі календаря без передплати, синхронізація не завжди миттєва – особливо в багатокористувацьких проєктах.

Однак, можна зазначити, що Todoist – це чудове рішення для тих, хто шукає простий, але потужний інструмент для управління завданнями, особливо якщо потрібен зручний спосіб розставлення пріоритетів та синхронізації з іншими сервісами.

Проаналізувавши існуючі застосунки для управління завданнями, можна зробити висновок, що вони пропонують широкий функціонал, зручну інтеграцію з календарями та навіть додаткові можливості, такі як Pomodoro-таймер або система тегів.

Проте значна частина з них має суттєві недоліки: складний або перевантажений інтерфейс, обмеження у безкоштовній версії, необхідність реєстрації для доступу до всіх функцій, а також нав’язливі нагадування про підписку. Це ускладнює використання застосунку, особливо для тих, хто шукає простий та ефективний інструмент для організації справ.

Щоб усунути ці проблеми, у застосунку буде максимально швидкий старт без обов’язкової реєстрації, простий і зрозумілий інтерфейс без зайвих елементів та всі основні функції без штучних обмежень. Користувачі зможуть одразу створювати завдання, керувати ними та отримувати нагадування без необхідності проходити складні налаштування або оформлювати підписку для базового використання.

1.3 Аналіз вимог до програмного засобу та постановка завдання

Аналізуючи потреби користувачів та особливості предметної області, до мобільного застосунку висуваються такі основні вимоги. Застосунок має забезпечувати ефективне управління завданнями, включаючи їх створення, редагування, відстеження статусу та виконання. Він повинен мати зручний та інтуїтивно зрозумілий інтерфейс, що дозволить користувачам швидко взаємодіяти із системою.

Основні функціональні вимоги передбачають підтримку створення списків завдань, встановлення пріоритетів, налаштування нагадувань і дедлайнів, а також можливість додавання підзадач і міток для кращої організації. Це дозволить користувачам не лише зберігати необхідну інформацію, а й структурувати її відповідно до власних потреб і стилю роботи.

Архітектура застосунку включатиме кілька ключових модулів, серед яких модуль управління завданнями, система сповіщень, календар для візуалізації дедлайнів, та модуль користувачів, що відповідає за персоналізацію інтерфейсу та взаємодії. Така модульна структура сприятиме масштабованості й підтримці різних сценаріїв використання.

Таким чином, застосунок повинен стати зручним інструментом для підвищення продуктивності, що дозволить користувачам ефективно організовувати свої справи та дотримуватися запланованих дедлайнів.

2 ПРОЄКТУВАННЯ СИСТЕМИ ЗАСТОСУНКУ КЕРУВАННЯ ОСОБИСТИМИ ЗАВДАННЯМИ

2.1 Проєктування структури системи

Структура проєкту To-Do спроєктована таким чином, щоб забезпечити чітке поділення на функціональні модулі та полегшити масштабування і супровід коду в майбутньому. Оскільки застосунок реалізовано за допомогою фреймворку Flutter[2], структура проєкту побудована відповідно до принципів модульності та гнучкості, що є однією з переваг цього фреймворку. Основні класи системи та їх взаємозв'язки представлені на діаграмі класів (рис. 2.1). Клас User відповідає за взаємодію з користувачем і створення завдань. Клас Task описує сутність завдання, зокрема, його атрибути та стан. Клас Category організовує завдання за категоріями, що допомагає користувачу ефективно структурувати завдання за різними типами. Для централізованого керування завданнями використовується клас TaskManager, який реалізує логіку фільтрації, додавання та видалення завдань, а також взаємодіє з іншими компонентами для забезпечення належного відображення завдань на екрані.

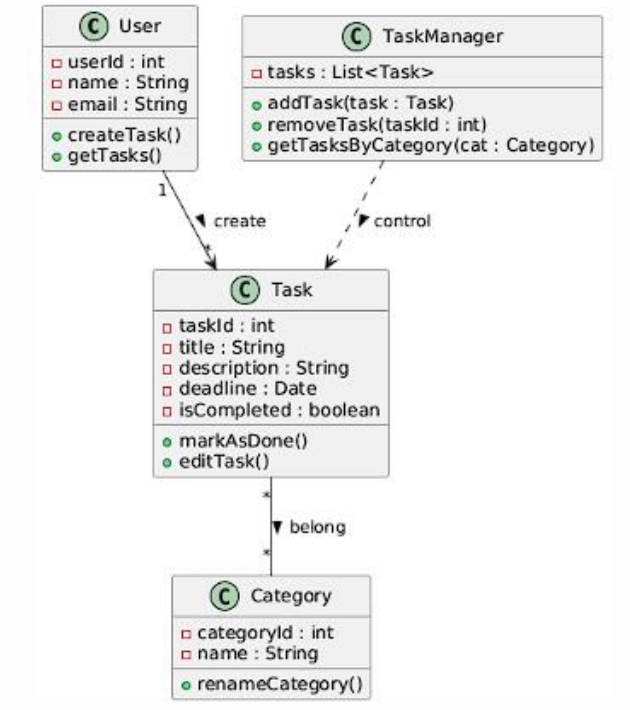

Рисунок 2.1 – Діаграма класів

Проєкт розбитий на кілька основних папок, кожна з яких відповідає за конкретну частину застосунку. Відповідно до архітектурного підходу MVVM, структура має чітке розподілення між моделями, віджетами інтерфейсу та логікою управління станом. Кожна функціональна частина застосунку, така як екран перегляду завдань, екран створення нових завдань чи модуль нагадувань, знаходиться у власній окремій папці з відповідними ресурсами.

У найвищому рівні структури проєкту знаходяться основні файли конфігурації, включаючи налаштування фреймворку та плагінів. Внутрішня організація проєкту побудована таким чином, щоб мінімізувати залежності між модулями. Це дозволяє змінювати окремі частини застосунку або додавати нові функціональні можливості без впливу на інші компоненти. Зокрема, екран для перегляду завдань, екран створення нового завдання та модуль нагадувань мають власні незалежні сервіси, які взаємодіють лише через чітко визначені інтерфейси.

Усередині кожної папки модулів структурується код за принципом "одна відповідальність". Наприклад, у папці, що містить логіку управління завданнями, зберігаються моделі для збереження та обробки даних, сервіси для взаємодії з базою даних Hive, а також контролери стану, що керують відображенням інформації на екрані. Кожен компонент взаємодіє з іншими через абстракції, що дозволяє зберігати модульність і гнучкість.

Використання бази даних Hive для збереження завдань передбачає створення окремого модуля для роботи з даними. Вся логіка взаємодії з базою даних організована через сервіси, що абстрагують операції з даними, дозволяючи змінювати джерела збереження без зміни основної бізнес-логіки. Це особливо важливо для підтримки функціональності застосунку в автономному режимі.

Структура проєкту дозволяє легко додавати нові екрани чи функціональні можливості, наприклад, інтеграцію з іншими сервісами або додаткові типи завдань. Завдяки чіткій організації коду і розподілу обов’язків між модулями, застосунок може масштабуватись без необхідності

переписувати значні частини коду. Крім того, структура забезпечує простоту для тестування окремих компонентів. Кожен модуль має свої окремі одиничні тести, що дозволяє легко перевіряти коректність роботи окремих частин застосунку без необхідності перевіряти всю систему цілком. Це сприяє кращому контролю за якістю коду і дозволяє швидко виявляти й усувати помилки.

У результаті проєкт має зручну та масштабовану структуру, що сприяє легкості в супроводі та розширенні, а також забезпечує високу якість коду при подальших змінах і доповненнях.

2.2 Аргументація вибору засобів та середовища

Для реалізації мобільного застосунку було обрано фреймворк Flutter, який дозволяє створювати кросплатформерні застосунки для платформ Android та iOS з єдиним кодовим базисом. Такий підхід суттєво знижує витрати часу та ресурсів на розробку та підтримку двох окремих застосунків для кожної з платформ. Flutter забезпечує можливість розробляти застосунки, які виглядають та працюють як нативні, тобто мають високу продуктивність і швидкість відгуку, що є критичним для мобільних застосунків. Оскільки головним завданням цього застосунку є надання простого та інтуїтивно зрозумілого інтерфейсу для користувача, використання Flutter дозволяє легко реалізувати всі необхідні елементи взаємодії з користувачем, при цьому не втрачаючи продуктивності. Крім того, можливість використовувати одну кодову базу для обох платформ значно спрощує процес тестування і оновлення застосунку, що важливо для його довгострокової підтримки.

Завдяки тісній інтеграції з фреймворком Flutter та високій продуктивності мовою програмування було обрано Dart[6]. Вона дозволяє писати чистий та структурований код, який легко тестується та підтримується. Мова підтримує асинхронне виконання, що дає можливість ефективно обробляти запити до бази даних або виконувати інші фонові операції без блокування основного потоку виконання застосунку. Це особливо важливо

для мобільних застосунків, де швидкість реагування на дії користувача має першочергове значення. Завдяки використанню Dart розробка застосунку стає швидшою, а код більш зрозумілим і легким для підтримки в довгостроковій перспективі.

Для збереження даних в застосунку було обрано базу даних Hive, яка є високопродуктивним і легким рішенням для локального зберігання даних на мобільних пристроях. Hive[3] – це NoSQL база даних, яка не вимагає наявності серверної частини, що дозволяє застосунку працювати в автономному режимі без необхідності підключення до інтернету. Це особливо важливо для To-Do застосунку, де користувачі очікують, що всі функціональні можливості будуть доступні навіть при відсутності стабільного інтернет-з'єднання. Hive забезпечує швидкий доступ до даних, оптимізуючи час відгуку при додаванні, редагуванні або видаленні завдань. Оскільки застосунок не має складної структури даних, Hive є ідеальним вибором, оскільки вона не потребує складної конфігурації та дозволяє зберігати дані у вигляді простих об'єктів без використання складних моделей. Це значно спрощує розробку і підвищує продуктивність застосунку.

Комбінація Flutter, Dart і Hive забезпечує створення стабільного, високопродуктивного та кросплатформерного застосунку, який відповідає вимогам до простоти використання, швидкості та ефективності роботи з даними. Такий вибір технологій дозволяє створити застосунок, який є масштабованим, добре підтримується і готовий до подальшого розвитку, а також відповідає сучасним вимогам до мобільних застосунків.

2.3 Проєктування бази даних

Оскільки Hive є легким та швидким рішенням для локального зберігання, особливо в мобільних застосунках, де критично важливо працювати з даними швидко і ефективно, це рішення було обране для забезпечення швидкого доступу до даних навіть при обмежених ресурсах мобільного пристрою. Важливим є те, що Hive не вимагає серверної

інфраструктури, а всі дані зберігаються локально, що дозволяє застосунку працювати автономно без постійного підключення до інтернету.

База даних організована як набір окремих колекцій, кожна з яких відповідає за зберігання конкретного типу даних. Основними елементами бази даних є три ключові колекції: завдання, категорії та нагадування. Кожен з цих елементів зберігається в окремому боксі Hive, що дозволяє організувати дані у вигляді простих об'єктів, що підвищує ефективність роботи з ними.

Завдання є основною сутністю в застосунку. Для кожного завдання зберігаються такі атрибути, як назва, опис, дата створення, дата виконання, статус (виконано чи не виконано). Крім того, кожне завдання має унікальний ідентифікатор, що дозволяє ефективно знаходити та редагувати конкретні записи. Завдання зберігаються в спеціальному боксі "Task Box", який містить список всіх завдань користувача.

Нагадування є важливою частиною функціональності застосунку. Кожне завдання може мати одне або кілька нагадувань, що спрацьовують у заданий час. Для кожного нагадування в базі даних зберігається дата та час сповіщення, а також посилання на завдання, до якого воно прив'язане. Це дозволяє користувачу отримати нагадування про необхідність виконання конкретного завдання. Нагадування зберігаються в боксі "Reminder Box", де для кожного запису є дата і час нагадування, а також зв'язок із конкретним завданням.

Взаємодія з базою даних реалізується через репозиторії. Репозиторії є абстракцією для доступу до бази даних, що дозволяє зручно виконувати операції створення, читання, оновлення та видалення даних (CRUD). Наприклад, при створенні нового завдання додається новий об'єкт класу Task, який зберігається в відповідному боксі. Для редагування або видалення завдання використовуються методи репозиторію, що дозволяють змінювати або видаляти записи в базі даних без необхідності безпосередньо працювати з низькорівневими деталями зберігання.

Взаємозв'язки між користувачами, завданнями та категоріями відображено на ER-діаграмі (рис. 2.2). Кожен користувач може створювати багато завдань. Завдання можуть належати до однієї або кількох категорій, а зв'язок між завданнями та категоріями реалізовано через проміжну таблицю TaskCategory.

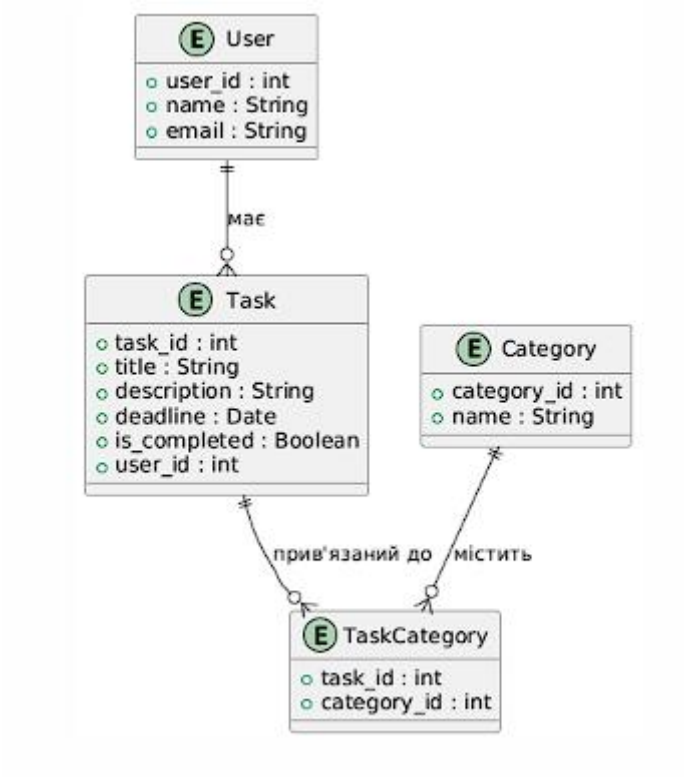

Рисунок 2.2 – ER-діаграма

Оскільки застосунок працює з особистими даними користувачів, важливо забезпечити безпеку збережених даних. Hive підтримує шифрування даних, що дозволяє захистити інформацію при її збереженні на мобільному пристрої. При ініціалізації бази даних можна вказати параметри шифрування, що дозволяє забезпечити конфіденційність даних навіть у випадку фізичного доступу до пристрою.

Для забезпечення швидкості роботи з базою даних важливо також оптимізувати запити. Hive має вбудовані механізми кешування, що дозволяють значно прискорити доступ до часто використовуваних даних. Крім того, для складніших запитів може бути використана індексація, що дозволяє швидко знаходити необхідні записи в базі даних.

Таким чином, проєктування бази даних у цьому мобільному застосунку з використанням Hive забезпечує ефективне зберігання та обробку даних користувачів, зокрема завдань, категорій та нагадувань, при цьому гарантуючи високу швидкість роботи та безпеку збережених даних.

2.4 Проєктування інтерфейсу застосунку

Проєктування інтерфейсу користувача для To-Do мобільного застосунку відіграватиме ключову роль у забезпеченні зручності, інтуїтивності та ефективності взаємодії користувача з функціональністю програми.

У процесі розробки буде зроблено акцент на створенні простого, логічного та естетично привабливого інтерфейсу, який не перевантажуватиме користувача зайвою інформацією та дозволятиме швидко орієнтуватися у доступних можливостях. Особлива увага приділятиметься ергономічності елементів керування, контрастності шрифтів і кольоровій палітрі, що сприятиме зменшенню втоми очей під час тривалого користування застосунком.

Головним екраном застосунку стане список завдань на поточний день. Він буде розроблений таким чином, щоб відображати лише найнеобхіднішу інформацію, як-от назву завдання, час нагадування та статус виконання. При цьому інші деталі, такі як опис, дата створення, будуть доступні у розгорнутому вигляді після натискання на конкретне завдання.

Це дозволить зберігати чистоту інтерфейсу, не відволікаючи користувача зайвими візуальними елементами.

Під час створення нового завдання користувачу буде запропоновано ввести назву, вибрати дату та час нагадування, додати необов’язковий опис. Усі ці дії виконуватимуться на одній зручній формі, без необхідності переходити між кількома екранами.

На рисунку 2.3 представлено схему інтерфейсу користувача із зазначенням основних екранів та логіки переходів між ними.

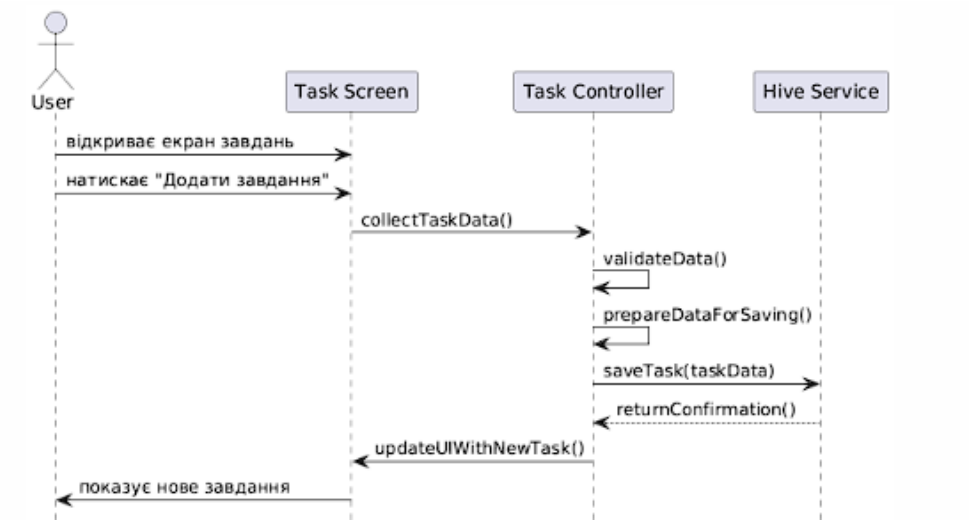

Рисунок 2.3 – Діаграма послідовності

Зміни у завданнях відображатимуться миттєво, без потреби перезавантаження сторінки, що позитивно вплине на загальне враження від роботи з застосунком. Для наглядності було створено макети сторінки головного екрана, форми створення нового завдання, а також вікна редагування, які відображають передбачувану структуру інтерфейсу, розміщення елементів керування та загальний стиль візуального оформлення. Ці макети виконують роль візуального прототипу та допомагають краще уявити, як виглядатиме застосунок під час використання. Вони слугуватимуть орієнтиром на етапі реалізації інтерфейсу у середовищі розробки.

Перший наведений макет – макет головної сторінки мобільного додатку, який є центральним елементом всього додатку та служить основною точкою взаємодії користувача з системою управління завданнями. Дизайн виконаний у мінімалістичному стилі з акцентом на функціональність та зручність використання на мобільних пристроях.

У верхній частині екрану розміщено заголовок "Мої завдання" з підписом, який відображає загальну кількість завдань у форматі "x/x завдань". Це забезпечує користувача миттєвою інформацією про поточний стан його списку завдань, праворуч – кнопка видалення (іконка смітника) для швидкого видалення всіх завдань.

Центральна частина екрану відведена під список завдань. Завдання будуть сформовані у вигляді списку, в якому користувач зможе керувати окремо кожним завданням, яке він обере і натисне на нього. Зліва від кожного завдання розміщено кругле поле для позначення статусу виконання (checkbox), яке дозволяє користувачу швидко відмічати виконані завдання.

Особливу увагу приділено кнопці додавання нового завдання, яка розміщена у правому нижньому куті екрану. Така позиція є інтуїтивно зрозумілою для користувачів мобільних додатків та забезпечує швидкий доступ до функції створення завдання. Макет головної сторінки наведений на рисунку 2.4.

Рисунок 2.4 – Макет головної сторінки

Другий розроблений макет – макет спеціалізованої сторінки "Додати нове завдання", яка представляє собою чітко структуровану форму для введення інформації про нове завдання. Дизайн сторінки відповідає принципам мобільного інтерфейсу з акцентом на простоту заповнення та мінімізацію кількості дій користувача.

У верхній частині екрану розміщено стрілку назад для швидкого повернення до головної сторінки, що забезпечує інтуїтивну навігацію. Заголовок "Додати нове завдання" чітко вказує на призначення сторінки та орієнтує користувача в контексті поточної дії.

Форма містить три основні поля для введення інформації. Перше поле "Що плануєте?" призначене для введення назви або короткого опису завдання. Це поле є обов'язковим та служить основним ідентифікатором завдання в системі. Друге поле "Додати нотатку" дозволяє користувачу додати детальну інформацію, пояснення або додаткові деталі, що можуть бути корисними для виконання завдання. Нижня частина форми містить два поля для встановлення часових параметрів: "Час" та "Дата". Ці поля забезпечують можливість планування завдань на конкретний час та дату, що є критично важливим для ефективного управління часом та пріоритетами. У нижній частині екрану розміщена чорна кнопка "Додати завдання", яка виділяється контрастним кольором та забезпечує чітке позначення основної дії на сторінці. Така кольорова схема робить кнопку помітною та інтуїтивно зрозумілою для користувача. Макет цієї сторінки зображений на рисунку 2.5.

Рисунок 2.5 – Макет сторінки додавання завдання

Третій макет представляє сторінку "Оновити завдання", яка призначена для редагування та управління вже існуючими завданнями.

Дизайн цієї сторінки базується на тій же структурі, що й сторінка створення завдання, забезпечуючи консистентність користувацького досвіду та інтуїтивність використання.

Верхня частина сторінки містить знайому стрілку назад для навігації та заголовок "Оновити завдання", який чітко вказує на можливість внесення змін до існуючого завдання. Форма включає ті ж поля, що й сторінка створення: "Що плануєте?" для редагування назви завдання, "Додати нотатку" для оновлення додаткової інформації, а також поля "Час" та "Дата" для корегування часових параметрів.

Ключовою відмінністю цієї сторінки є розширена функціональність в нижній частині екрану. Замість однієї кнопки для збереження, користувач має доступ до двох основних дій: "Видалити завдання" та "Оновити завдання".

Така двокнопкова система дозволяє користувачу як зберегти внесені зміни, так і повністю видалити завдання, якщо воно втратило актуальність.

Розміщення двох кнопок поруч забезпечує швидкий доступ до обох функцій, але при цьому потребує від користувача уважності при виборі відповідної дії. Макет цієї сторінки зображений на рисунку 2.6.

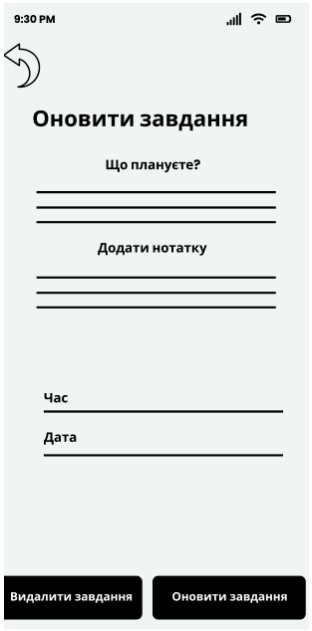

Рисунок 2.6 – Макет сторінки оновлення завдання

Розроблений мобільний інтерфейс поєднує три ключові принципи: простоту використання, функціональну повноту та адаптивність до мобільних пристроїв. Мінімалістичний дизайн з інтуїтивною навігацією дозволяє швидко отримувати доступ до всіх функцій без зайвих дій, що дає змогу користувачам зосередитись на виконанні завдань.

Використання монохромної кольорової схеми з контрастними акцентами (чорні кнопки, лінії завдань) забезпечує чітку читабельність на будь-яких пристроях та при різному освітленні.

Стандартні UI-елементи (круглі кнопки, стрілки навігації, зрозумілі іконки) роблять інтерфейс звичним для користувачів незалежно від їх технічного досвіду, підвищуючи ефективність взаємодії з додатком.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

3.1 Підготовка програмного середовища

Для реалізації мобільного застосунку управління завданнями було обрано сучасний стек технологій, який забезпечує швидку, надійну та кросплатформену розробку. Основним середовищем реалізації став фреймворк Flutter, який дозволяє створювати застосунки з єдиною кодовою базою для платформ Android та iOS, що суттєво прискорює процес розробки та спрощує подальшу підтримку продукту. Мовою програмування було обрано Dart, яка тісно інтегрується з Flutter та має зрозумілий синтаксис, що сприяє зменшенню кількості помилок під час написання коду.

Перед початком реалізації функціоналу було встановлено та налаштовано всі необхідні компоненти середовища розробки. Основу становить IDE Android Studio, яка містить інструменти для розробки, запуску, налагодження та тестування Flutter-застосунків. Через Android Studio було встановлено Flutter SDK останньої стабільної версії, а також Dart SDK, який входить до його складу. Додатково були налаштовані необхідні середовища емуляції мобільних пристроїв – Android Virtual Device (AVD) – для тестування застосунку без потреби фізичного смартфона.

Для керування залежностями та зовнішніми пакетами використовувався файл pubspec.yaml, який є стандартним конфігураційним файлом у Flutter-проєктах. До проєкту було додано кілька зовнішніх пакетів, що значно розширили функціональні можливості системи. Одним із ключових пакетів є Hive, який слугує як легка та ефективна NoSQL-база даних, що зберігає дані безпосередньо у пам’яті пристрою. Hive був обраний через його високу продуктивність, простоту у використанні та відсутність залежності від SQL-синтаксису.

Після додавання Hive було також встановлено додаткові плагіни: hive_flutter для інтеграції з Flutter та path_provider, який дозволяє Hive знаходити місце для збереження файлів у файловій системі мобільного

пристрою. Щоб дозволити Hive працювати з користувацькими моделями даних, було підключено генератор адаптерів та відповідну команду build_runner, яка дозволяє генерувати типобезпечні серіалізатори об’єктів.

Крім базової підсистеми зберігання даних, у застосунку також реалізовано механізм нагоджень, для чого використовувався пакет flutter_local_notifications. Цей пакет дозволяє створювати локальні сповіщення, які не потребують підключення до мережі. Він був обраний з огляду на його гнучкість, підтримку різних платформ і можливість точного планування сповіщень на задану дату і час. Налаштування цього пакету включало інтеграцію з Android Notification Channel, дозвіл на показ сповіщень на рівні системи та ініціалізацію об'єкта NotificationPlugin у кореневому класі застосунку.

Для візуального оформлення інтерфейсу застосунку було використано бібліотеку Flutter Material Components, що містить готові стилізовані елементи інтерфейсу – кнопки, списки, текстові поля тощо. Крім того, були застосовані власні стилі, створені за допомогою темізації, що дозволило створити єдиний візуальний стиль застосунку відповідно до обраного дизайну.

Для організації структури коду було прийнято рішення дотримуватись архітектурного підходу, що базується на поділі відповідальностей (Separation of Concerns). Уся логіка застосунку була поділена на окремі модулі: моделі, репозиторії, провайдери стану, сервіси сповіщень, екрани та віджети. Це забезпечило зручність масштабування, повторне використання коду та спрощене тестування окремих компонентів.

Для зберігання поточного стану застосунку, зокрема списку завдань, був використаний пакет provider, який реалізує шаблон управління станом у Flutter. Через нього здійснюється передача стану між віджетами без необхідності пропускати дані через кілька рівнів ієрархії. Це рішення дозволило зберегти логіку взаємодії з даними централізовано та ізольовано від UI-компонентів.

Крім основних компонентів, у процесі підготовки середовища було також створено і налаштовано проєктну структуру з відповідними директоріями: lib/models, lib/services, lib/providers, lib/screens, lib/widgets тощо. Такий підхід спрощує навігацію у коді, дозволяє легше орієнтуватися серед файлів і пришвидшує командну розробку.

Отже, підготовка програмного середовища включала встановлення Flutter SDK, Android Studio [4], емуляторів, додаткових плагінів, підключення бази даних Hive, реалізацію системи сповіщень та структурування коду згідно з архітектурними принципами. Такий підхід створив міцну основу для подальшої реалізації функціоналу застосунку, а також забезпечив можливість ефективного тестування, масштабування та підтримки проєкту.

3.2 Реалізація основних функцій застосунку

Після завершення налаштування середовища розробки та базової структури проєкту розпочалась реалізація основних функціональних можливостей мобільного застосунку для створення, зберігання, редагування, сортування, нагадування та видалення особистих завдань. На цьому етапі були реалізовані ключові компоненти логіки застосунку: моделі даних, інтеграція з базою даних Hive, створення користувацького інтерфейсу, взаємодія з локальними нагадуваннями та логіка керування станом.

Особливу увагу приділено адаптації інтерфейсу під мобільне середовище та забезпеченню стабільної роботи зі збереженням даних між сеансами. Основна логіка застосунку побудована навколо роботи з локальною базою завдань, яка зберігає важливу інформацію для кожного завдання: назву, опис, дедлайн та необов'язковий час нагадування. Для реалізації нагадувань використовується взаємодія з локальними системними повідомленнями Android через плагін flutter_local_notifications.

Перед початком розробки функціональності застосунку, було визначено, як саме будуть виглядати дані, з якими працюватиме додаток. Основою всього

застосунку є завдання користувача, тому створення правильної моделі даних стало першочерговим завданням.

Модель "Завдання" (Task) розроблена як Dart-клас, який містить всю необхідну інформацію про окреме завдання користувача. Для забезпечення можливості збереження цих даних у локальній базі даних Hive було використано спеціальні анотації, які дозволяють системі автоматично перетворювати об'єкти в формат, придатний для зберігання. Програмний код класу “Завдання” наведено в додатку А.

Після створення моделі даних наступним важливим кроком стала підготовка системи зберігання даних до роботи. База даних Hive потребує спеціальної ініціалізації перед тим, як застосунок зможе почати створювати, читати або змінювати завдання користувача.

Процес ініціалізації відбувається на самому початку роботи застосунку, ще до відображення користувацького інтерфейсу. Це критично важливо, оскільки без правильно налаштованого сховища застосунок не зможе функціонувати належним чином. У лістингу 3.1 представлено фрагмент коду ініціалізації сховища Hive.

Лістинг 3.1 – Ініціалізація сховища Hive

```
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Hive.initFlutter();
  Hive.registerAdapter(TaskAdapter());
  await Hive.openBox<Task>('tasks');
  runApp(const MyApp());
}
```

Наступним кроком стала реалізація функціональності додавання нових завдань користувачем. Ця можливість є основою всього застосунку, оскільки без неї користувач не зміг би створювати власні списки справ.

Для збору інформації від користувача використовуються стандартні контролери Flutter, які дозволяють отримувати текстові дані з полів введення. Як показано в лістингу 3.2, спочатку ініціалізуються контролери для назви та опису завдання.

Лістинг 3.2 – Ініціалізація контролерів для введення даних

```
    TextEditingController titleController =
TextEditingController();
    TextEditingController subtitleController =
TextEditingController();
```

Процес створення нового завдання відбувається через виклик спеціального методу, який збирає дані з форми та зберігає їх у локальній базі даних. В лістингу 3.3 показано реалізацію цього функціоналу.

Лістинг 3.3 – Метод додавання нового завдання

```
void addTask() {
  final task = Task.create(
    title: titleController.text,
    subTitle: subtitleController.text,
  );

  final box = Hive.box<Task>('tasks');
  box.add(task);
}
```

Використання фабричного методу Task.create() спрощує процес створення об'єкта, автоматично генеруючи унікальний ідентифікатор та встановлюючи поточну дату і час створення.

Після створення об'єкта він додається до відкритої раніше коробки Hive, що забезпечує миттєве збереження даних на пристрої користувача.

Для забезпечення зручного перегляду всіх створених завдань було реалізовано динамічний список, який автоматично оновлюється при будь-яких змінах у базі даних.

Ця функціональність критично важлива для користувацького досвіду, оскільки дозволяє миттєво бачити результати своїх дій.

Основою реалізації є компонент ValueListenableBuilder, який прослуховує зміни в коробці Hive та автоматично перебудовує інтерфейс при необхідності.

В лістингу 3.4 наведено повну реалізацію списку завдань.

Лістинг 3.4 – Відображення динамічного списку завдань

```
ValueListenableBuilder(
  valueListenable: Hive.box<Task>('tasks').listenable(),
  builder: (context, Box<Task> box, _) {
    final tasks = box.values.toList();
    return ListView.builder(
      itemCount: tasks.length,
      itemBuilder: (context, index) {
        final task = tasks[index];
        return ListTile(
          title: Text(task.title),
          subtitle: Text(task.subTitle),
          trailing: Checkbox(
            value: task.isCompleted,
            onChanged: (value) {
              task.isCompleted = value!;
              task.save();
            },
          ),
        ); },  );  },)
```

Цей підхід забезпечує швидкодію інтерфейсу - список автоматично оновлюється при додаванні, зміні або видаленні завдань. Кожне завдання відображається у вигляді окремого елемента списку з можливістю позначення як виконане через чекбокс, що дозволяє користувачам легко відстежувати свій прогрес.

Візуалізація прогресу виконання завдань також є важливим елементом користувацького досвіду.

Для відображення загального стану виконання всіх завдань використовується спеціальний віджет, який поєднує круговий індикатор прогресу з текстовою інформацією.

Цей компонент надає користувачам миттєвий візуальний зворотний зв'язок про їхні досягнення та мотивує до завершення поставлених цілей.

Програмний код для візуалізації прогресу завдань наведено в лістингу 3.5.

Лістинг 3.5 – Візуалізація прогресу завдань

```
Widget _buildProgressIndicator(List<Task> tasks) {
  return Row(
    mainAxisAlignment: MainAxisAlignment.center,
    children: [
      SizedBox(
        width: 30,
        height: 30,
        child: CircularProgressIndicator(
          value: checkDoneTask(tasks) /
valueOfIndicator(tasks),
          backgroundColor: Colors.grey,
          valueColor: const
AlwaysStoppedAnimation(AppColors.primaryColor),
        ),
      ),
      const SizedBox(width: 25),
      Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Text("Мої завдання", style:
textTheme.displayLarge),
          Text("${checkDoneTask(tasks)} з ${tasks.length}
завдань",
              style: textTheme.titleMedium),
        ],
      )
    ],
  );
}
```

Цей підхід забезпечує швидкодія інтерфейсу - список автоматично оновлюється при додаванні, зміні або видаленні завдань. Кожне завдання відображається у вигляді окремого елемента списку з можливістю позначення як виконане через чекбокс, що дозволяє користувачам легко відстежувати свій прогрес.

Для забезпечення логічного порядку відображення завдань в інтерфейсі користувача застосовується механізм автоматичного сортування за датою створення. Цей підхід гарантує, що найстаріші завдання відображаються першими в списку, що відповідає природній логіці виконання справ за хронологічним порядком. Сортування відбувається динамічно при кожному оновленні даних, забезпечуючи актуальність порядку завдань. Програмний код для сортування завдань наведено в лістингу 3.6.

Лістинг 3.6 – Реалізація сортування завдань

```
    var tasks = box.values.toList();
    tasks.sort((a, b) =>
a.createdAtDate.compareTo(b.createdAtDate));
```

Представлений код демонструє ефективну реалізацію сортування колекції завдань. Спочатку всі завдання з бази даних Hive перетворюються в список за допомогою методу toList(), що дозволяє застосувати до них операції сортування. Метод sort() використовує компаратор, який порівнює дати створення двох завдань через compareTo(), забезпечуючи впорядкування від найстаршого до найновішого. Цей механізм інтегрований у ValueListenableBuilder, що означає автоматичне пересортування при будь-яких змінах у базі даних, підтримуючи консистентний порядок відображення завдань.

Можливість редагування вже створених завдань є невід'ємною частиною функціональності застосунку. Користувачі часто потребують внесення змін до своїх завдань, будь то уточнення опису, зміна назви або оновлення статусу виконання.

Процес редагування реалізовано через безпосередню зміну властивостей об'єкта Task з подальшим збереженням змін. Завдяки наслідуванню від HiveObject, кожен об'єкт Task має вбудований метод save(), який синхронізує зміни з базою даних. В лістингу 3.7 наведено фрагмент програмного коду для редагування існуючих завдань.

Лістинг 3.7 – Редагування існуючого завдання

```
    void updateTask(Task task, String newTitle, String
newSubtitle) {
      task.title = newTitle;
      task.subTitle = newSubtitle;
      task.save();}
```

Цей підхід забезпечує простоту та ефективність операцій оновлення. Зміни відразу відображаються в інтерфейсі завдяки ValueListenableBuilder, що створює плавний користувацький досвід без необхідності ручного оновлення списку.

Функціонал видалення завдань дозволяє користувачам підтримувати свій список актуальним, прибираючи завершені або неактуальні завдання. Ця можливість реалізована максимально просто завдяки використанню вбудованих методів Hive. В лістингу 3.8 наведено фрагмент програмного коду для видалення завдань з Hive бази даних.

Лістинг 3.8 – Видалення завдання з бази даних

```
void deleteTask(Task task) {
  task.delete();
}
```

Метод delete() автоматично видаляє об'єкт з коробки Hive та ініціює оновлення всіх пов'язаних з нею інтерфейсних компонентів. Це означає, що видалене завдання миттєво зникає зі списку без потреби в додаткових діях з боку розробника.

Для підвищення зручності управління завданнями в додатку імплементовано інтуїтивний жест "змахнути для видалення", який дозволяє користувачам швидко видаляти непотрібні завдання горизонтальним рухом пальця. Цей функціонал реалізований за допомогою віджета Dismissible, який забезпечує плавну анімацію видалення та автоматичну синхронізацію змін з базою даних. Користувач отримує візуальний зворотний зв'язок у вигляді фонового повідомлення про дію видалення. Програмний код для реалізації swipe-to-delete функціональності наведено в лістингу 3.9.

Лістинг 3.9 – Програмний код реалізації видалення завдань жестом

```
Dismissible(
  direction: DismissDirection.horizontal,
  onDismissed: (_) => base.dataStore.deleteTask(task: task),
  background: Row(
    mainAxisAlignment: MainAxisAlignment.center,
    children: const [
      Icon(Icons.delete_outline, color: Colors.grey),
      SizedBox(width: 8),
      Text(AppStr.deletedTask, style: TextStyle(color:
Colors.grey)),
    ],
  ),
  key: Key(task.id),
  child: TaskWidget(task: task),
)
```

Також для того, щоб система нагадувань функціонувала належним чином було налаштовано канал сповіщень Android. Це критично важливий крок, оскільки сучасні версії Android вимагають явного створення каналів для категоризації повідомлень та надання користувачам контролю над їх відображенням.

Канал сповіщень визначає, як система буде обробляти повідомлення від застосунку, включаючи їх пріоритет, звуки, вібрацію та інші параметри відображення. Налаштування каналу відбувається один раз при запуску застосунку та зберігається в системі до видалення програми. Фрагмент програмного коду для ініціалізації каналу сповіщень наведено в лістингу 3.10.

Лістинг 3.10 – Ініціалізація каналу сповіщень

```
    const AndroidNotificationDetails
androidPlatformChannelSpecifics = AndroidNotificationDetails(
      'task_channel_id',
      'Task Notifications',
      channelDescription: 'Notifications for task reminders',
      importance: Importance.max,
      priority: Priority.high,
      enableVibration: true,
    );
```

Ця конфігурація каналу забезпечує максимальну ефективність доставки нагадувань. Параметр importance: Importance.max гарантує відображення сповіщень навіть у режимі "Не турбувати", що особливо важливо для нагадувань про завдання. Увімкнена вібрація посилює користувацький досвід, забезпечуючи тактильний відгук навіть коли пристрій знаходиться в беззвучному режимі.

Після налаштування каналу постало завдання реалізації безпосереднього планування сповіщень. Ця функціональність потребувала особливої уваги до роботи з часовими зонами, оскільки користувачі можуть створювати завдання в одному часовому поясі, а отримувати нагадування в іншому.

Систему планування побудовано таким чином, щоб автоматично враховувати локальний час пристрою та забезпечувати точну доставку повідомлень незалежно від зміни часових поясів або переходу на літній/зимовий час. Програмний код для планування сповіщення в залежності від часової зони наведено в лістингу 3.11.

Лістинг 3.11 – Планування сповіщення з часовою зоною

```
Future<void> scheduleNotification({
  required int id,
  required String title,
  required String body,
  required DateTime scheduledTime,
}) async {
  await flutterLocalNotificationsPlugin.zonedSchedule(
    id,
    title,
    body,
    tz.TZDateTime.from(scheduledTime,     tz.local),     //
Конвертація в локальний час
    NotificationDetails(android:
androidPlatformChannelSpecifics),
    androidAllowWhileIdle:  true,  //  Доставка  у  режимі
енергозбереження
    uiLocalNotificationDateInterpretation:
        UILocalNotificationDateInterpretation.absoluteTime,
  );
}
```

Реалізація використовує метод zonedSchedule, який автоматично враховує часовий пояс пристрою через параметр tz.local[5]. Особливо важливим виявився параметр androidAllowWhileIdle, який забезпечує доставку сповіщень навіть коли Android-пристрій перебуває в режимі енергозбереження або "глибокого сну".

Інтеграція системи нагадувань з основною логікою застосунку здійснюється через виклик цього методу при створенні або редагуванні завдань. Унікальний ідентифікатор сповіщення генерується на основі ключа завдання в Hive, що дозволяє ефективно керувати життєвим циклом нагадувань та уникати конфліктів між різними завданнями.

Для покращення користувацького досвіду при відсутності завдань у списку реалізовано привабливий анімований інтерфейс пустого стану. Цей

підхід замінює стандартне порожнє відображення на інтерактивну анімацію, яка інформує користувача про успішне виконання всіх завдань. Використання бібліотеки Lottie дозволяє відтворювати складні векторні анімації з мінімальним впливом на продуктивність додатку. Додаткові ефекти появи створюють плавний та професійний перехід до стану "всі завдання виконано". Програмний код для анімації пустого стану наведено в лістингу 3.12.

Лістинг 3.12 – Реалізація анімованого інтерфейсу пустого стану

```
Column(
  mainAxisAlignment: MainAxisAlignment.center,
  children: [
    FadeIn(
      child: SizedBox(
        width: 300,
        height: 350,
        child: Lottie.asset(
          lottieURL,
          animate: tasks.isNotEmpty ? false : true,
        ),
      ),
    ),
    FadeInUp(
      from: 30,
      child: const Text(AppStr.doneAllTask),
    ),
  ],
)
```

Представлена реалізація демонструє інтеграцію кількох анімаційних бібліотек для створення ефектного користувацького інтерфейсу. Основний елемент - Lottie-анімація, розміщена в контейнері фіксованого розміру (300x350 пікселів), з умовною логікою відтворення через параметр animate. Анімація активується лише при порожньому списку завдань (tasks.isNotEmpty ? false : true), що забезпечує оптимальне використання ресурсів. Ефект FadeIn застосовується до всієї анімації, створюючи плавну появу, тоді як FadeInUp з параметром from: 30 надає тексту ефект підйому знизу вгору на 30 пікселів. Вертикальне вирівнювання по центру через MainAxisAlignment.center забезпечує гармонійне розташування елементів на екрані.

3.3 Створення та налаштування бази даних

Однією з базових функціональних складових мобільного застосунку для керування особистими завданнями є внутрішня система зберігання даних, яка забезпечує швидкий доступ до завдань, збереження історії дій, а також автономну роботу без постійного інтернет-з'єднання. Для досягнення цієї мети було обрано технологію Hive – локальну NoSQL базу даних, оптимізовану для використання у Flutter-застосунках.

Hive є бібліотекою, розробленою спеціально для Dart-середовища, яка дозволяє зберігати дані у вигляді пар ключ-значення без потреби у складному конфігуруванні або написанні SQL-запитів. Вона працює напряму з типізованими Dart-об'єктами, підтримує генерацію адаптерів для кастомних моделей і забезпечує високу продуктивність навіть на слабких пристроях. На відміну від традиційних реляційних баз, Hive не має фіксованої схеми таблиць, що дозволяє швидко змінювати структуру моделі без ризику втрати даних.

Після підключення Hive до проєкту і додавання необхідних залежностей у pubspec.yaml, наступним етапом стало створення моделі даних Task, яка описує основну сутність у системі. Модель включає унікальний ідентифікатор, заголовок, опис, дату та час створення, а також статус виконання. Фрагмент визначення класу Task наведено в додатку Б.

Цей клас використовує анотації @HiveType та @HiveField для серіалізації, що дозволяє Hive зберігати об'єкти в локальному сховищі. Генерація адаптера відбувається автоматично за допомогою пакета build_runner. Після цього Hive може зчитувати й записувати об'єкти типу Task.

Ще одним важливим кроком стало відкриття боксу, у якому зберігатимуться завдання. Ініціалізація Hive виконується під час запуску застосунку, після чого відкривається відповідний бокс. Програмний код ініціалізації наведено в лістингу 3.13.

Лістинг 3.13 – Ініціалізація Hive

```
await Hive.initFlutter();
Hive.registerAdapter(TaskAdapter());
await Hive.openBox<Task>('tasks');
```

Кожен бокс у Hive можна розглядати як окрему таблицю, іменовану та типізовану, що зберігає об’єкти певного класу. У даному випадку, відкритий бокс 'tasks' служить єдиним джерелом правди для всіх записів про завдання користувача.

Для збереження нового завдання у базу було реалізовано просту логіку. Після створення нового об’єкта Task через фабричний конструктор, він додається до боксу. Програмний код цієї логіки наведено в лістингу 3.14.

Лістинг 3.14 – Програмний код логіки

```
final taskBox = Hive.box<Task>('tasks');
final newTask = Task.create(title: 'Новий запис', subTitle: 'Опис');
await taskBox.add(newTask);
```

Так само працює оновлення чи видалення записів: достатньо змінити властивість об’єкта та викликати save(), або застосувати метод delete(), доступний усім об’єктам, що успадковують HiveObject.

Оскільки застосунок передбачає взаємодію з базою в реальному часі, усі компоненти інтерфейсу працюють у режимі швидкого відображення. При зміні стану сховища оновлення інтерфейсу відбувається автоматично. Це реалізовано за допомогою ValueListenableBuilder, який слухає відповідний бокс і програмний код якого наведено в лістингу 3.15.

Лістинг 3.15 – Програмний код ValueListenableBuilder

```
ValueListenableBuilder(
  valueListenable: Hive.box<Task>('tasks').listenable(),
  builder: (context, Box<Task> box, _) {
    final tasks = box.values.toList();
    return ListView.builder(
      itemCount: tasks.length,
      itemBuilder: (context, index) {
        final task = tasks[index];
        return Text(task.title);
      },  );  },);
```

Завдяки такому підходу забезпечується плавна, безперервна робота застосунку, де кожне оновлення даних відображається без потреби вручну викликати методи перезавантаження.

Крім основних операцій збереження, зчитування та видалення, важливою складовою стало коректне управління життєвим циклом боксу, особливо в контексті мультиекранного інтерфейсу. Ініціалізація Hive та відкриття необхідного боксу відбувається один раз при запуску застосунку, після чого отриманий об’єкт боксу доступний для будь-яких маніпуляцій упродовж усього сеансу роботи. Це дозволяє уникнути зайвих повторних звернень до файлової системи пристрою, що могло б спричинити втрату продуктивності або неочікувані виключення.

Ще один аспект, що заслуговує на увагу, – це організація логіки фільтрації та сортування завдань безпосередньо у застосунку. Наприклад, при необхідності відобразити лише завершені завдання або відсортувати список за датою створення, використовується стандартна колекційна логіка Dart. Такий підхід є ефективним, зважаючи на невеликі обсяги локальних даних, характерні для особистих To-Do застосунків. Він дозволяє реалізувати гнучке й адаптивне представлення даних без потреби у складних запитах або додаткових індексах.

Важливо також враховувати можливість розширення структури даних у майбутньому. Hive не вимагає жорсткої схеми таблиць, проте будь-яке оновлення моделі, таке як додавання нових полів, вимагає синхронного оновлення адаптерів і потенційної міграції існуючих даних. Це можливо реалізувати за допомогою продуманих стратегій версіонування моделей або ручної обробки при відкритті боксу. У межах даного проєкту зміни структури моделі контролювалися вручну, із обов’язковою генерацією оновленого адаптера та перевіркою на зворотну сумісність.

Ще однією важливою деталлю є забезпечення конфіденційності даних користувача. Hive підтримує вбудоване шифрування боксів із використанням AES-шифрування, що є вагомим фактором при розробці застосунків, які можуть опрацьовувати персональні або чутливі дані. У поточному застосунку реалізація шифрування не була першочерговою вимогою, однак архітектура передбачає можливість його інтеграції на подальших етапах розробки,

зокрема, у поєднанні з Flutter Secure Storage або іншими захищеними сховищами ключів.

Також було враховано потребу в очищенні пам’яті та закритті боксів після завершення роботи застосунку або при зміні користувача, якщо така функціональність передбачена. Hive дозволяє явно закривати відкриті бокси, що є важливим етапом для запобігання витокам пам’яті та забезпечення стабільної роботи системи у довготривалому користуванні.

У підсумку, реалізація та налаштування локальної бази даних за допомогою Hive не лише задовольнила функціональні вимоги, а й створила гнучке, стабільне та масштабоване середовище для подальшого розвитку. Такий підхід дозволив зосередитися на зручності користувача, мінімізувати технічні ризики та забезпечити сучасний рівень ефективності у керуванні персональними завданнями. Використання Hive є обґрунтованим технічним вибором, що забезпечує баланс між зручністю розробки, ефективністю використання ресурсів і комфортом для кінцевого користувача.

3.4 Реалізація логіки нагадування

Логіка нагадування у розробленому мобільному застосунку реалізовуватиметься за допомогою бібліотеки flutter_local_notifications, яка забезпечує можливість надсилати як миттєві, так і відкладені локальні сповіщення. Основна мета цієї підсистеми – нагадувати користувачеві про створені завдання у визначений ним час. Щоб реалізувати зазначену функціональність, було створено окремий файл notification_helper.dart, який містить три основні асинхронні функції: планування, скасування одного сповіщення та скасування всіх запланованих сповіщень.

Перед початком роботи із самими сповіщеннями у головному файлі main.dart виконується ініціалізація об'єкта FlutterLocalNotificationsPlugin. Під час запуску застосунку відбувається виклик функції initializeNotifications, у якій виконується налаштування каналу сповіщень для Android, ініціалізація часових зон через бібліотеку timezone та створення необхідного каналу через

платформозалежну реалізацію. Таким чином, забезпечується підтримка роботи нагадувань у фоновому режимі та під час сну пристрою.

Безпосереднє планування сповіщення відбувається у функції scheduleNotification, програмний код якої знаходиться в додатку В. Функція приймає п’ять параметрів: ідентифікатор (id), заголовок (title), текст повідомлення (body), об'єкт DateTime з часом нагадування (scheduledTime) і необов’язкове поле payload. У тілі функції викликається метод flutterLocalNotificationsPlugin.zonedSchedule, у якому час сповіщення конвертується до формату TZDateTime, програмний код якого наведено в лістингу 3.16.

Лістинг 3.16 – Метод flutterLocalNotificationsPlugin.zonedSchedule

```
tz.TZDateTime.from(scheduledTime, tz.local)
```

Канал сповіщення налаштовується у вигляді об'єкта NotificationDetails, де вказується високий пріоритет, відтворення звуку та вібрація. Програмний код наведено в лістингу 3.17.

Лістинг 3.17 – Налаштування каналу сповіщення

```
const NotificationDetails(
  android: AndroidNotificationDetails(
    'task_channel_id',
    'Task Notifications',
    channelDescription: 'Notifications for task reminders',
    importance: Importance.max,
    priority: Priority.high,
    playSound: true,
    enableVibration: true,
  ),
)
```

Також використовується параметр matchDateTimeComponents: DateTimeComponents.time, що дозволяє реалізувати щоденне повторення сповіщення у заданий час.

У разі потреби скасувати конкретне сповіщення реалізовано функцію cancelNotification, програмний код якої наведено в лістингу 3.18.

Лістинг 3.18 – Програмний код функції cancelNotification

```
Future<void> cancelNotification(int id) async {
  await flutterLocalNotificationsPlugin.cancel(id);
}
```

Вона викликає метод flutterLocalNotificationsPlugin.cancel(id), що дозволяє скасувати конкретне сповіщення за його id.

Для скасування всіх запланованих нагадувань доступна функція cancelAllNotifications, програмний код якої наведено в лістингу 3.19.

Лістинг 3.19 – Програмний код функції cancelAllNotifications

```
Future<void> cancelAllNotifications() async {
  await flutterLocalNotificationsPlugin.cancelAll();
}
```

Вона викликає метод flutterLocalNotificationsPlugin.cancelAll(), який дозволяє скасувати абсолютно всі сповіщення.

Ці функції дозволяють динамічно керувати сповіщеннями на основі дій користувача – наприклад, якщо завдання було змінено або видалено.

Таким чином, реалізована система нагадувань повністю інтегрується із механізмом створення завдань і забезпечує своєчасне інформування користувача. Це значно покращує зручність користування застосунком і дозволяє ефективно організовувати особистий час.

У перспективі, в майбутньому функціонал буде розширено. Буде реалізовано періодичні або повторювані нагадування з використанням параметра matchDateTimeComponents.

Також для забезпечення кращого користувацького буде додано можливість керування нагадуваннями через окремий екран налаштувань.

Загалом, реалізація логіки нагадування базуватиметься на принципах надійного планування подій, обробки часових зон та інтеграції з системними механізмами Android/iOS, що дозволить досягти високої точності та зручності використання нагадувань у мобільному To-Do застосунку.

3.5 Реалізація інтерфейсу користувача

Інтерфейс користувача мобільного To-Do застосунку розроблений відповідно до принципів інтуїтивної взаємодії, мінімалізму та адаптивності. Особлива увага приділялася тому, щоб користувач міг без зайвих зусиль створювати, переглядати та редагувати свої завдання, а також отримувати своєчасні нагадування.

Головна сторінка застосунку (рис. 3.1) реалізована у вигляді списку завдань із простим та зрозумілим інтерфейсом.

У верхній частині розміщено заголовок "Мої завдання", який вказує на поточний розділ. Нижче відображається індикатор прогресу у форматі "0 з 0 завдань", оскільки на екрані поки немає жодного завдання.

Індикатор дозволяє користувачу швидко оцінити загальну кількість завдань та їх статус.

Оскільки активні завдання відсутні система відображає повідомлення "Ви виконали всі завдання! ", що супроводжується символічною посмішкою. Цей елемент виконує роль візуального підказу, запобігаючи відчуттю порожнечі та мотивуючи користувача до створення нових завдань.

Дизайн сторінки витриманий у мінімалістичному стилі: використано чітку типографіку, нейтральний фон та мінімальну кількість елементів, що забезпечує зручність користування.

Для швидкого додавання нових завдань передбачено кнопку у нижній частині екрану, розташовану в зоні легкого доступу.

Загальна композиція інтерфейсу сприяє швидкій орієнтації та ефективній роботі з завданнями, відповідаючи сучасним стандартам UI/UX-дизайну.

Рисунок 3.1 – Головна сторінка застосунку

Для зручного додавання нових пунктів у нижній частині екрану розміщено круглу кнопку з іконкою "+", виконану у кольоровій гамі, що відповідає загальній темі застосунку. При натисканні на неї відкривається сторінка створення нового завдання, що забезпечує швидкий та інтуїтивний доступ до основної функції програми. Розташування кнопки дозволяє комфортно користуватися застосунком однією рукою.

Сторінка додавання нового завдання (рис. 3.2) реалізована у вигляді зрозумілої та мінімалістичної форми для швидкого введення інформації. У верхній частині екрану розміщено “стрілочку” для повернення на головну сторінку, а також заголовок "Додати нове Завдання", що чітко вказує на призначення цієї сторінки. Під заголовком знаходиться підказка "Що плануєте?", яка направляє користувача у процесі створення завдання. Форма містить поле для додавання нотатки, яка необхідна для додаткового опису

завдання. Час виконання завдання відображається у зручному 12-годинному форматі з позначкою AM/PM, а дата представлена у вигляді зрозумілого скорочення з днем тижня, числом, місяцем та роком. У нижній частині екрану розташована кнопка "Додати завдання", виконана у кольорі, що гармонійно поєднується з загальною кольоровою схемою застосунку. Вся форма виконана у єдиній стилістиці з головним екраном, з використанням аналогічної типографіки та мінімалістичного дизайну. Елементи інтерфейсу мають чіткі межі та достатні відступи, що забезпечує зручність взаємодії. Вибір часу та дати реалізовано через стандартні компоненти платформи, що робить процес інтуїтивно зрозумілим для користувача. Загальний дизайн сторінки спрямований на максимально швидке та безпомилкове додавання нового завдання при мінімальній кількості дій з боку користувача.

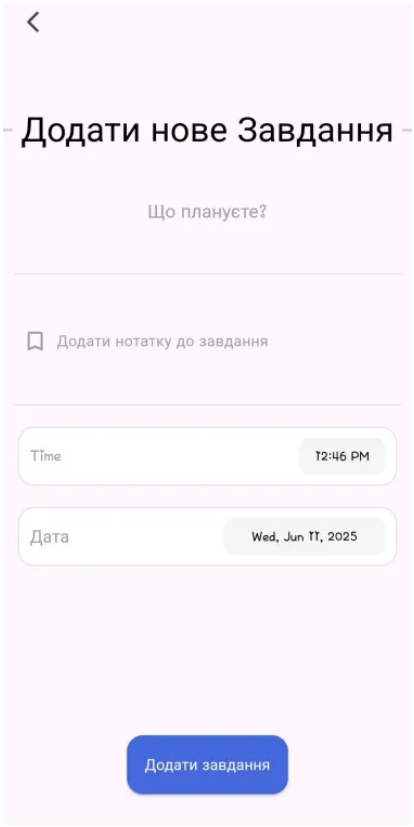

Рисунок 3.2 – Сторінка додавання завдання

Вікно вибору дати (рис. 3.3) організоване з урахуванням зручності та функціональності. У верхній частині інтерфейсу розташовані дві чіткі кнопки управління - "Cancel" зліва та "Done" справа, що дозволяє швидко підтвердити

або скасувати вибір. Центральна зона вікна містить інтуїтивно зрозумілу сітку даних, де роки, місяці та числа організовані у логічній послідовності. Поточний рік, місяць та число виділені у верхньому рядку. Вибір конкретної дати здійснюється простим дотиком до відповідного осередку, при цьому система надає візуальний зворотний зв'язок. Дизайн вікна витриманий у стилістиці всього додатку - мінімалістичний, з чіткими контурами елементів та оптимальним контрастом між текстом і фоном. Таке рішення дозволяє користувачеві швидко орієнтуватися у календарі та ефективно планувати свої завдання на будь-який термін, включаючи віддалені роки.

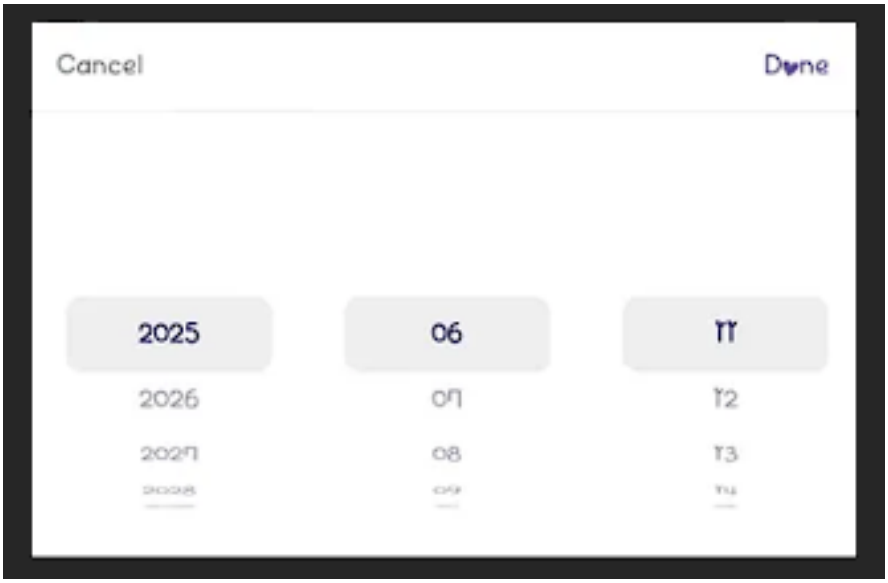

Рисунок 3.3 – Вікно вибору дати

Вікно вибору часу (рис. 3.4) організоване за тими ж принципами мінімалізму та функціональності, що й інтерфейс вибору дати, формуючи єдину систему взаємодії. Як і в календарному модулі, тут передбачено дві чіткі кнопки управління – "Cancel" зліва та "Done" справа, що підтримує інтуїтивну послідовність роботи з параметрами часу. Основна зона призначена для вибору числового значення через механізм плавної прокрутки, де кожен пункт відображається у вигляді чітко окресленого елементу з оптимальним контрастом. Специфікою цього інтерфейсу є розширений діапазон значень, який охоплює всі можливі варіанти для точного та гнучкого планування. Подібно до календарного вікна, тут реалізовано принцип миттєвого

візуального відгуку – при прокручуванні списку активний елемент візуально виділяється, що дозволяє користувачеві безпомилково ідентифікувати свій вибір. Дизайн витриманий у спільній стилістиці: такі ж м'які переходи, аналогічна типографіка та узгоджена кольорова схема, що забезпечує гармонійну інтеграцію з іншими елементами системи.

Особливу увагу приділено ергономіці – розміри елементів та відстані між ними розраховані для комфортної роботи на сенсорних екранах. Цей модуль, як і інтерфейс вибору дати, демонструє продуману систему взаємодії, де кожна деталь спрямована на максимальне спрощення процесу введення часу при збереженні точності та функціональності.

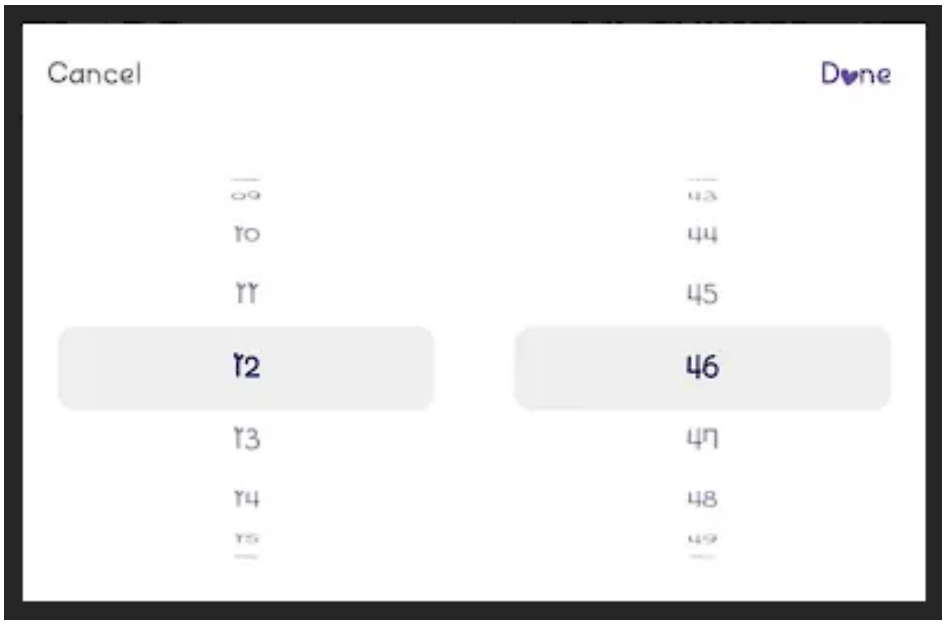

Рисунок 3.4 – Вікно вибору часу

Сторінка оновлення наявного завдання (рис. 3.5) стилістично схожа до сторінки для створення нового завдання, для кращого користувацького досвіду. У полях можна змінити текст завдання, додати або редагувати нотатку, а також оновити дату та час виконання. На відміну від екрану створення завдання, тут додано кнопку "Видалити завдання" поряд з кнопкою "Оновити завдання", що дозволяє повністю видалити запис. Всі елементи інтерфейсу збережені в єдиній стилістиці додатку.

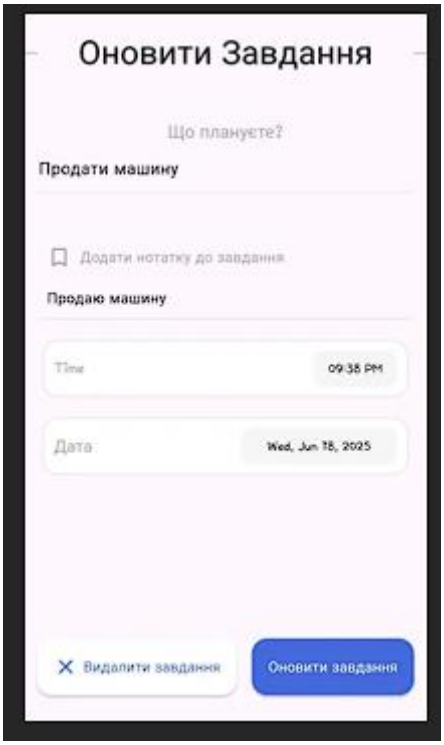

Рисунок 3.5 – Сторінка для оновлення існуючого завдання

Після того як користувач додає завдання (рис. 3.6), воно відображається на головній сторінці у вигляді списку, з його назвою, нотаткою, та датою виконання. Зліва є checkbox, який призначений для позначення вже виконаного завдання. В такому випадку завдання змінює колір фону, а також перекреслюється основний текст.

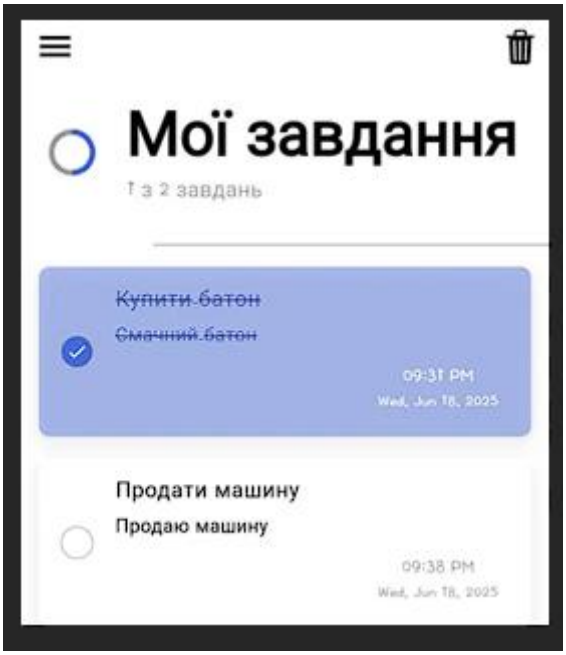

Рисунок 3.6 – Додані завдання

Якщо користувач захоче видалити всі завдання за допомогою кнопки, розташованої зправа зверху, для нього відкриється вікно з попередженням (рис 3.7) про цю дію. Якщо завдань не буде, користувач натиснувши на цю кнопку отримає відповідне повідомлення (рис. 3.8).

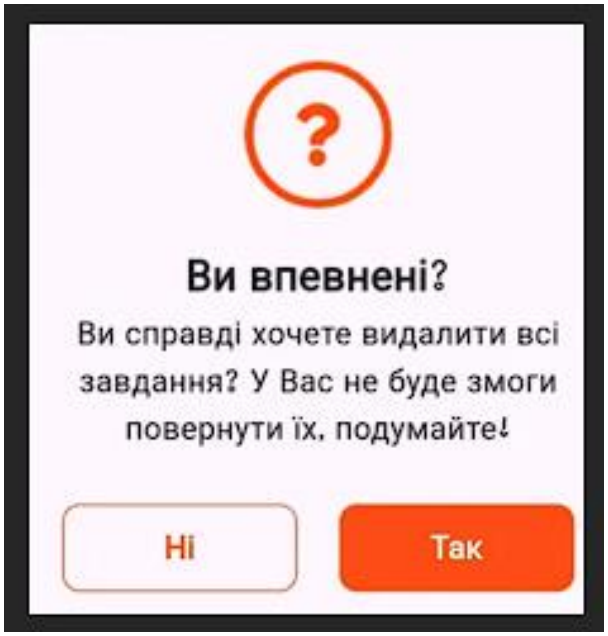

Рисунок 3.7 – Попередження перед видаленням

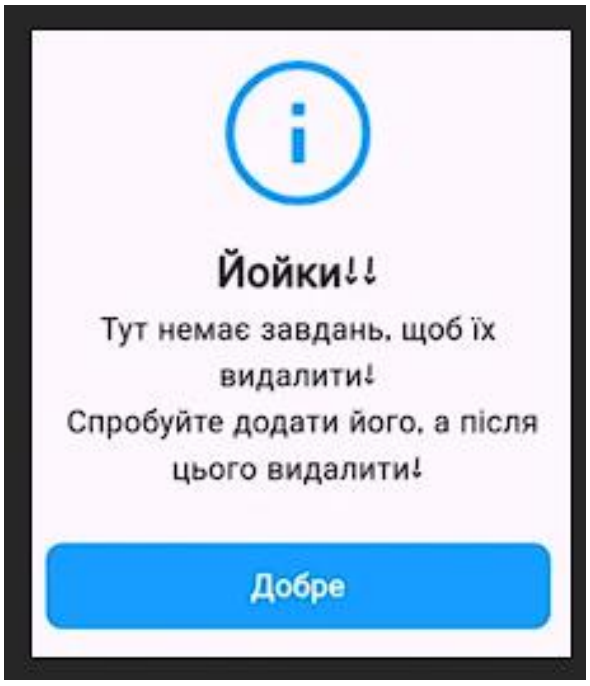

Рисунок 3.8 – Помилка при видаленні завдання

В загальному, інтерфейс застосунку реалізовано у мінімалістичному стилі з чіткою типографікою та інтуїтивною навігацією, що забезпечує зручність користування. Використано єдину стилістику для всіх екранів, з оптимальними відступами та контрастними кольорами для кращої візуалізації. Кожен елемент інтерфейсу спроєктовано з урахуванням ергономіки, що дозволяє користувачам швидко та без зусиль керувати завданнями.

3.6 Тестування застосунку

Тестування програмного забезпечення є невід'ємною частиною життєвого циклу розробки, особливо коли мова йде про продуктивність та зручність використання.

У випадку з To-Do додатком, який є інструментом для організації повсякденних завдань, важливість ретельного тестування зростає в рази. Це пов'язано з тим, що користувачі очікують від такого рішення абсолютної надійності - втрата записів, некоректне збереження стану задач або проблеми з синхронізацією між пристроями можуть призвести до втрати довіри до продукту.

Спочатку було протестовано базовий функціонал застосунку. Для прикладу, було взято сценарій, де користувач відкриває застосунок і додає завдання.

Після запуску користувача переправляє на головну сторінку застосунку, з Lottie-анімацією, і в списку немає жодного завдання. Сторінка аналогічна, як і на рисунку 3.1.

Після натискання на кнопку з позначкою “+” користувач переходить до сторінки додавання завдання. Користувачу потрібно створити завдання з заголовком “Купити молоко”, нотаткою “Молокія 3,5%”, на 12.06.25, 12:00. Заповнені поля за сценарієм зображено на рисунку 3.9.

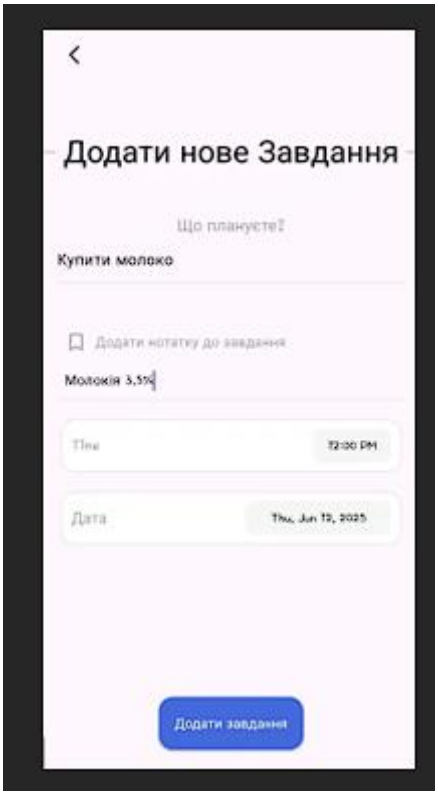

Рисунок 3.9 – Додавання завдання

Після того, як користувач перевірив правильність заповнення всіх полів, натиснувши на кнопку “Додати завдання”, яка знаходиться внизу по центру екрану – додає його. Це завдання з’являється на головній сторінці (рис. 3.10).

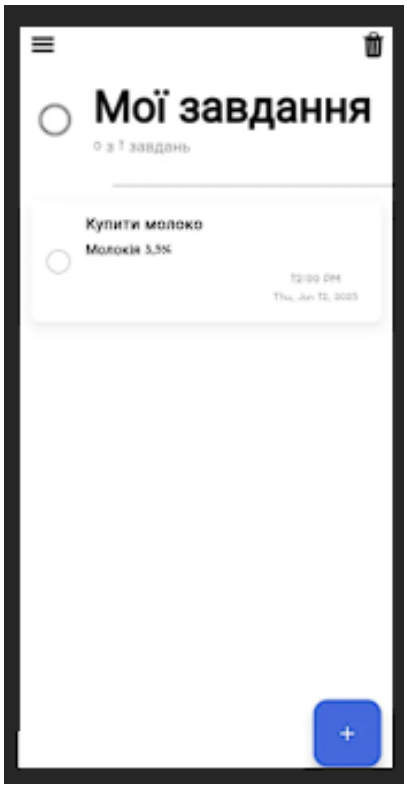

Рисунок 3.10 – Головна сторінка з завданням

Було припущено, що користувач вже виконав це завдання, але не захотів видаляти його з загального списку. Для цього зліва від завдання є спеціальний checkbox, натиснувши на який користувач позначив це завдання як виконане. Це зображено на рисунку 3.11.

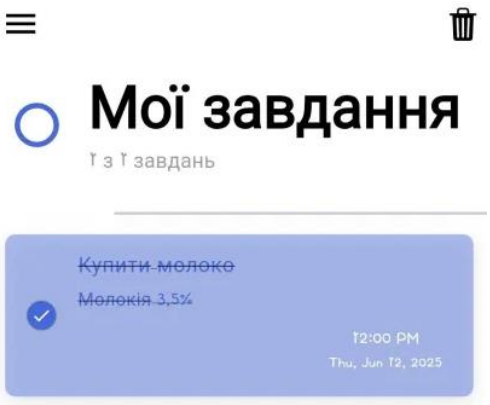

Рисунок 3.11 – Виконане завдання

Також, як зображено на рисунку вище, шкала прогресу повністю заповнилась, оскільки виконано всі завдання. Далі було припущено, що користувач також додав нове випадкове завдання. Тепер шкала прогресу буде заповнена наполовину, оскільки виконана тільки половина завдань зі списку, що відображено на рисунку 3.12.

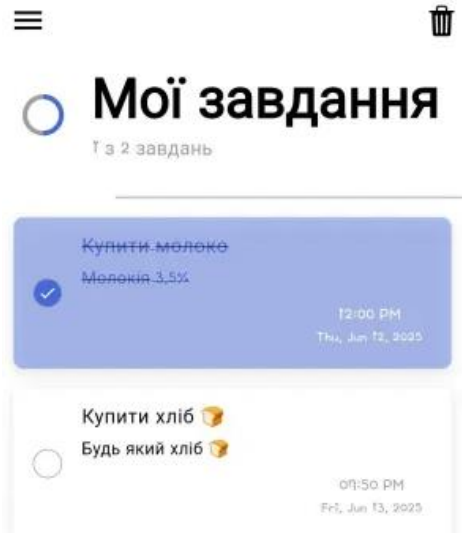

Рисунок 3.12 – Додавання іншого завдання

У користувача було два способи для видалення завдання:

– видалити перетягнувши його в бік екрану;

– натиснути на верхню інтуїтивно зрозумілу кнопку з зображенням “Смітника”.

В першому сценарії, де користувач видаляє завдання перетягуванням на його місці з’являється надпис “Це завдання було видалено” (рис. 3.13), після чого завдання назавжди видаляється зі сховища телефону.

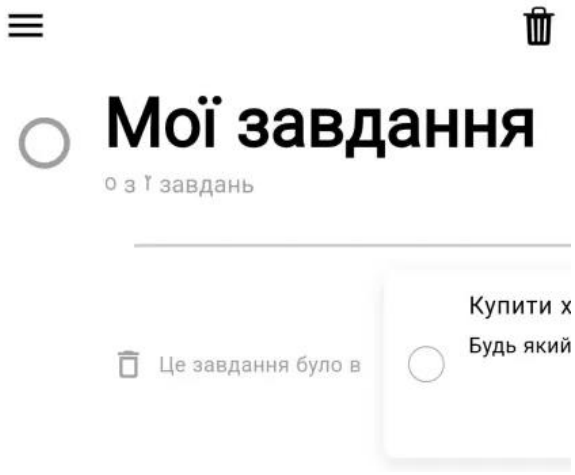

Рисунок 3.13 – Видалення завдання перетягуванням

В наступному сценарії користувач хоче видалити всі завдання за допомогою відповідної кнопки. Передбачено, якщо в списку не буде жодного завдання, то користувачу відкриється вікно помилки, яке зображено на рисунку 3.8. Якщо ж все-таки завдання в списку є, то користувачу відкриється вікно (рис. 3.7), щоб він підтвердив свої дії. Якщо він все підтверджує, то всі завдання видаляються з локального сховища і все повертається до Lottie-анімації, під якою пише, що всі завдання вже виконані.

Також передбачено, якщо на сторінці додавання завдання користувач не заповнить жодного поля, то відповідна помилка (рис. 3.14).

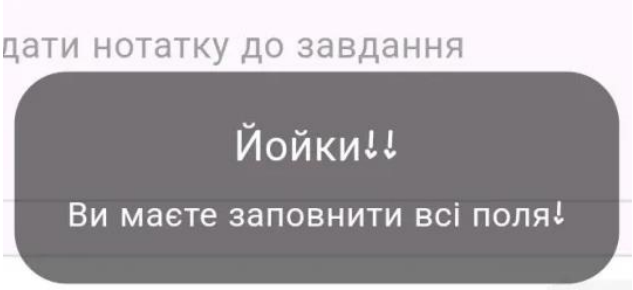

Рисунок 3.14 – Помилка пустих полів

В наступному сценарії користувач відредагував вже існуюче завдання. Для цього було створено нове завдання, яке зображено на рисунку 3.15.

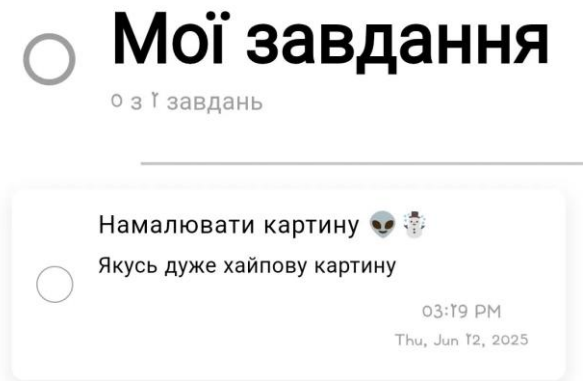

Рисунок 3.15 – Нове завдання

Після цього, натиснувши на нього користувача перенесло на сторінку з редагуванням завданням. Якщо не заповнити на ній якесь поле, то з’являється аналогічна помилка, як на рисунку 3.14. Коли користувач відредагував завдання, далі було натиснуто на кнопку “Оновити завдання” (рис. 3.16).

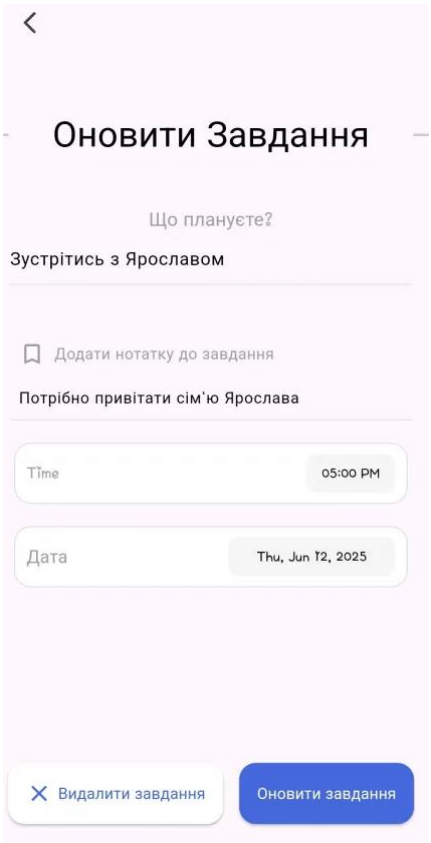

Рисунок 3.16 – Оновлення завдання

Після цього завдання було успішно відредаговано і в новому вигляді з’явилось на головній сторінці. Це показано на рисунку 3.17.

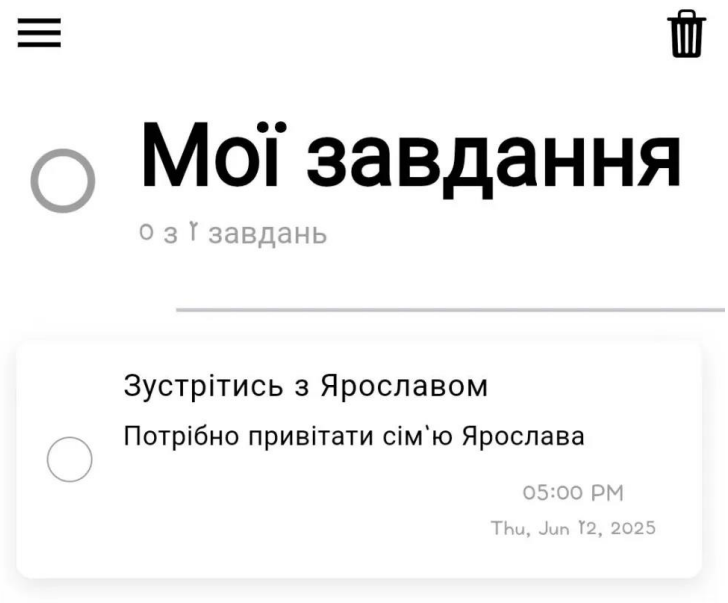

Рисунок 3.17 – Оновлене завдання

У даному розділі було повністю реалізовано функціональну частину мобільного застосунку для управління завданнями, що включає підготовку середовища розробки, створення бази даних, розробку основних сценаріїв використання та впровадження логіки нагадувань. Особливу увагу приділено інтерфейсу користувача, який забезпечує інтуїтивну взаємодію та зручність у повсякденному використанні. Інтерфейс було протестовано в різних сценаріях, зокрема при створенні, редагуванні, видаленні завдань і роботі з push-нагадуваннями.

Тестування підтвердило стабільну та коректну роботу системи відповідно до технічного завдання. Реалізовані функції працюють згідно з очікуваннями, інтерфейс залишається адаптивним і зрозумілим, а нагадування надійно доставляються відповідно до заданого часу. Отже, можна зробити висновок, що поставлені завдання реалізовано повною мірою, а розроблений застосунок є готовим до подальшого впровадження, розширення або використання в реальних умовах.

4 ТЕХНІКО-ЕКОНОМІЧНЕ ОБГРУНТУВАННЯ

4.1 Аналіз ринку збуту продукту

To-do мобільний застосунок, створений у межах кваліфікаційної роботи, має привабливі економічні перспективи завдяки поєднанню низьких витрат на підтримку та широкої потенційної аудиторії. Використання кросплатформенного підходу дає змогу охопити як користувачів Android, так і iOS без необхідності дублювання витрат на розробку окремих версій, що суттєво знижує початкові інвестиції. Локальне зберігання даних через вбудовану базу дозволяє обійтися без витрат на серверну інфраструктуру, обслуговування хостингу чи зовнішніх баз даних, що робить продукт економічно вигідним у довгостроковій перспективі.

Цей застосунок не є унікальним за своїм функціоналом – він має багато аналогів на ринку, однак він є вдосконаленою модифікацією, яка усуває недоліки багатьох конкурентів: відсутність офлайн-доступу, складний інтерфейс, висока ціна або необхідність підписки. Основна конкурентна перевага – простота, швидкість, конфіденційність та доступність. Застосунок зберігає дані лише на пристрої користувача, що гарантує приватність, не потребує реєстрації та доступний одразу після встановлення.

Цільова аудиторія – активні користувачі смартфонів віком від 16 до 50 років: студенти, фрілансери, офісні працівники, менеджери, які мають потребу в простому інструменті для планування особистих завдань. Продукт також підходить тим, хто віддає перевагу мінімалістичним застосункам без зайвого функціоналу.

Ринок збуту охоплює мобільні платформи Android і iOS, розповсюдження здійснюється через Google Play та App Store. Враховуючи кросплатформенність, можливість локалізації інтерфейсу та простоту використання, продукт має потенціал виходу на міжнародний ринок. На першому етапі основний акцент робиться на український ринок, з подальшим розширенням на країни Європи.

Очікується стабільний попит, оскільки сегмент To-Do застосунків постійно поповнюється новими користувачами. Особливо це стосується осіб, які переходять від традиційного планування (нотаток, блокнотів) до цифрових засобів. Перевагою продукту є відсутність потреби в підключенні до інтернету, що робить його доступним для ширшого кола користувачів, зокрема в регіонах з нестабільним покриттям.

Продаж здійснюватиметься за freemium-моделлю: основний функціонал безкоштовний, розширені функції (аналітика, додаткові нагадування, теми оформлення) – за разову плату або через рекламу. Така модель дозволяє охопити більше користувачів і поступово монетизувати продукт. Платна версія буде конкурентною за ціною порівняно з аналогами (Todoist, TickTick), де застосовується підписна модель.

Прогнозований обсяг продажів у перший рік – 10–20 тисяч інсталяцій, з них близько 10% користувачів можуть перейти на платну версію. Це дає змогу швидко отримати перші прибутки й продовжити розвиток продукту.

Серед основних конкурентів – Google Tasks, Todoist, Any.do, Microsoft To Do. Продукція конкурентів має широку функціональність, проте часто вимагає авторизації, онлайн-доступу, має складніші інтерфейси, або побудована на щомісячних підписках. Дизайн конкурентів є професійним, але перевантаженим – наш продукт робить ставку на лаконічність, мінімалізм та зручність.

Рівень цін на ринку аналогічних продуктів варіюється: безкоштовні – з обмеженнями, підписні – від 80 до 200 грн/міс. Запропонована ціна в 300 грн одноразово за повний доступ до розширених функцій є вигідною альтернативою.

Основні конкурентні переваги продукту є проста автономна робота без облікового запису, швидкий запуск і використання з мінімальними вимогами, прозора цінова модель (freemium + разова покупка), гарантоване збереження конфіденційності даних, адаптивність і легкість масштабування.

З урахуванням потреб користувачів, тенденцій ринку та конкурентного аналізу можна зробити висновок про доцільність і потенційний успіх розробленого To-Do застосунку як в Україні, так і на міжнародному ринку. Життєвий цикл продукту прогнозується на 3–5 років із можливістю регулярного оновлення та вдосконалення функціоналу.

4.2 Розрахункова частина

Процес створення мобільного To-Do застосунку охоплює комплекс витрат, які необхідно оцінити для аналізу економічної доцільності проєкту. Розрахунок охоплює ключові напрями витрат: оплата праці, нарахування, витрати на матеріали, електроенергію, амортизацію обладнання, та накладні витрати. Додаткові витрати (зв'язок, ПЗ, підтримка) будуть враховані в наступному підпункті.

Для розрахунку витрат на розробку було взято формулу "Вартість розробки = Зарплата+Електроенергія+Придбання/Оренда+Інші".

На сайті міністерства фінансів зазначено, що місячна мінімальна зарплата в Україні з 01.04.2024 складає 8000 грн [7].

Розробка здійснюється одним спеціалістом – розробником, що працює над застосунком повний місяць. Ураховується погодинна ставка, кількість відпрацьованих годин та преміювання за формулою "Зп = T_год * G * D_прем", де T_год – погодинна ставка (150 грн), G – кількість робочих годин (120), D_прем – коефіцієнт премії (1.5). Таким чином, було визначено, що заробітна плата складає 150*120*1.5 = 27000 грн.

Також є додаткова заробітна плата, яка передбачає в собі відпускні, лікарняні, різні доплати. Вона складає близько 20% від загальної заробітньої плати, тобто 27000*0.2 = 5400.

Єдиний соціальний внесок (ЄСВ) обов'язково сплачується роботодавцем від нарахованої заробітної плати. Станом на 25.04.2025 цей внесок становить 22% від заробітної плати. Тобто цей внесок складає 32400*0.22= 7128 грн.

ПДФО(податок на дохід фіз. особи) станом на 25.04.2025 складає 18% від заробітної плати, тобто 32400*0.18 = 5832 грн.

Військовий збір станом на 25.04.2025 складає 5% від заробітної плати, що становить 32400*0.05 = 1620 грн.

Після всіх розрахувань було визначено, що загальні витрати на заробітну плату складають 27000+5400+7128+5832+1620 = 46980 грн.

Основні витрати на електроенергію для кожного використаного приладу наведено в таблиці 4.1. Тариф на електроенергію станом на 25.04.2025 складає 4.32 грн/кВт*год

Таблиця 4.1 – Витрати на електроенергію

Прилад	Потужність(Вт)	Час роботи(год)	Витрати(грн)
Комп’ютер	300	210	272,16
Монітор	50	210	45,36
Настільна лампа	15	80	5,2
Wi-Fi роутер	15	360	23,3

Розрахунки витрати електроенергії приладами наведені нижче:

– комп’ютер – 0,3*210*4,32 = 272,16 грн;
– монітор – 0.05*210*4,32 = 45,36 грн;
– настільна лампа – 0.015*80*4,32 = 5,2 грн;
– wi-fi роутер – 0.015*360*4,32 = 23,3 грн.

Отже, після проведених розрахунків можна визначити, що Сел (витрати на електроенергію) складаються 272,16+45,36+5,2+23,3 = 346,02 грн.

Основні витрати на оренду/придбання ПЗ та різних технічних засобів наведено в таблиці 4.2.

Таблиця 4.2 – Витрати на оренду/придбання ПЗ та ТЗ

Прилад/засіб	Кількість	Вартість	Сума
Figma Pro	1	600 грн / міс.	600 грн
Потужний ноутбук	1	45000 грн	45000 грн
Visual Studio Code	1	0 грн	0 грн
Хмарне сховище Google Drive	100 ГБ	150 грн	150 грн

Отже, за даними, що наведені в таблиці вище витрати на оренду/придбання ПЗ та ТЗ склали 600+45000+150 = 45750 грн.

До інших витрат належать витрати на зв'язок, що наведено в таблиці 4.3.

Таблиця 4.3 – Витрати на зв'язок

Тип	Кількість	Вартість	Сума
Wi-Fi	1	250 грн/міс.	250 грн
Поповнення мобільного	1	250 грн/міс.	250 грн

Отож, загалом витрати на зв'язок склали 250+250=500 грн.

Також в інші витрати входять витрати на комунальні послуги, які зображені в таблиці 4.4.

Таблиця 4.4 – Комунальні послуги

Тип	Вартість	Сума
Оренда офісного робочого місця	3000 грн/міс.	3000 грн
Опалення	350 грн/міс.	350 грн
Прибирання	300 грн/міс.	300 грн
Водопостачання	400 грн/міс.	400 грн

Отже, витрати на комунальні послуги склали 3000+350+300+400 = 4050.

В інші витрати також було додано витрати на різні канцелярські потреби, розхідний матеріал. Орієнтовано сума складає 1000 грн в місяць, а також виділено фіксований бюджет на різні "імпульсивні"витрати, а саме 5000 грн в місяць.

Загалом інші витрати складають 500+4050+1000+5000 = 10550 грн.

Отож, загальні витрати на розробку проєкту (Срозр) складають 46980+346,02+45750+10550= 103626,02 грн.

Економічний ефект від впровадження мобільного To-Do застосунку розраховується шляхом порівняння очікуваної вигоди від його використання для кінцевих користувачів і потенційного прибутку від його комерціалізації.

Користувачі, які будуть використовувати мобільний застосунок для організації своїх завдань, зможуть заощадити час на плануванні та організації,

а також підвищити ефективність виконання завдань. Якщо, наприклад, середній час на організацію завдань в тиждень становить 2 години, а за допомогою застосунку цей час зменшується до 1 години, то кожен користувач зможе зекономити до 52 годин на рік. Зважаючи на 5000 користувачів, загальна економія часу становитиме 260 000 годин на рік, що можна перевести в грошовий еквівалент, оцінивши вартість робочої години для кінцевих користувачів.

Також однією з вигод для користувачів є відсутність потреби в платній підписці, характерній для конкурентних сервісів (Todoist, TickTick, Microsoft To Do Premium). Середня вартість підписки становить 100 грн/міс.

В такому випадку, економія одного користувача(Еод) становить Еод = 100*12 = 1200 грн на рік. Якщо застосунок будуть використовувати, до прикладу, 5000 користувачів, то економія на рік складатиме 1200*5000 = 6000000 грн/рік.

Комерційна стратегія передбачає freemium-модель. Додаткові функції (аналітика, розширені нагадування) доступні за разову оплату. Припущено, що загальна кількість користувачів – 5000, частка користувачів, що придбала додаткові функції – 10% (500 користувачів), разова плата за доступ до функцій – 300 грн.

Отже, очікуваний дохід буде складати 500*300 = 150000 грн.

Враховуючи, що застосунок дозволяє зекономити кошти на платних підписках, також можна оцінити позитивний вплив на підвищення лояльності користувачів до продукту. Це може сприяти більшому утриманню користувачів і покращенню їхньої загальної задоволеності.

У майбутньому застосунок може залучити більше користувачів через маркетингові кампанії, оптимізацію функціоналу та розширення бази користувачів, що дозволить збільшити доходи від придбання додаткових функцій.

Для визначення економічної ефективності розробки мобільного To-Do застосунку необхідно розрахувати строк окупності проєкту. Строк окупності

– це період, за який дохід, отриманий від реалізації продукту, компенсує витрати на його розробку.

Загальні економічні витрати на розробку проєкту склали близько 130626 грн, очікуваний дохід складає 150000 грн.

Тобто, Ток(термін окупності) = 130626/150000 = 0,87 року, що рівно близько 10 місяцям.

Отже, проєкт окупається менш ніж за один рік від моменту виходу на ринок, що свідчить про його високу економічну ефективність навіть за помірного обсягу реалізації.

Для зручності це було наведено в таблиці 4.5.

Таблиця 4.5 – Окупність проєкту

Показник	Значення
Загальні витрати	130626 грн
Очікуваний дохід	150000 грн
Строк окупності	0,87 року
Строк окупності(міс.)	~10 місяців

4.3 Обгрунтування необхідності розробки

Розробка мобільного To-Do застосунку зумовлена зростаючою потребою в цифрових інструментах для ефективного управління особистим часом та планування завдань. В умовах швидкого темпу життя все більше людей стикається з проблемою інформаційного перевантаження, відсутності систематизації повсякденних справ та неефективного використання часу. Ці фактори стимулюють попит на програмні рішення, що дозволяють організувати та контролювати щоденні завдання.

Однією з головних причин створення цього продукту є незадоволеність частини користувачів існуючими аналогами. Більшість популярних застосунків мають складний інтерфейс, вимагають реєстрації, постійного підключення до інтернету або працюють виключно в межах підписної моделі. Такий підхід обмежує можливості частини користувачів, які шукають простий, швидкий та безпечний інструмент для ведення нотаток і списків

справ. Запропонований застосунок усуває ці обмеження завдяки автономності, мінімалістичному інтерфейсу, локальному зберіганню даних і прозорій ціновій політиці.

Варто також враховувати техніко-економічні чинники. Розробка виконана з використанням Flutter, що дає змогу швидко масштабувати проєкт на різні платформи з мінімальними витратами. База даних Hive забезпечує збереження даних без потреби в сервері, що дозволяє мінімізувати витрати на інфраструктуру. Завдяки цьому продукт є економічно доцільним і має високу рентабельність, що підтверджується коротким терміном окупності (менше одного року) та позитивним економічним ефектом від використання застосунку користувачами.

Крім того, ринок мобільних застосунків демонструє стабільне зростання, зокрема в сегменті To-Do рішень. За останніми статистичними даними, понад 50% користувачів смартфонів використовують інструменти для планування завдань. В умовах цифрової трансформації освіти, роботи та особистого життя очікується подальше зростання попиту на подібні застосунки. Це створює сприятливе середовище для виведення на ринок нових конкурентоспроможних продуктів, що орієнтовані на зручність, простоту та безпеку.

З огляду на наведене, можна стверджувати, що створення To-Do застосунку є обґрунтованим як з економічної, так і з практичної точки зору. Проєкт поєднує у собі актуальність, технічну ефективність, комерційний потенціал і соціальну корисність. Його впровадження здатне задовольнити запити користувачів, покращити якість організації часу та водночас забезпечити стабільний прибуток розробнику чи компанії, яка його підтримуватиме.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було реалізовано мобільний застосунок для керування особистими завданнями. Основна мета полягала у створенні зручного інструменту для організації повсякденних справ, що забезпечує користувачам можливість ефективно планувати час, встановлювати нагадування, групувати завдання та відстежувати прогрес виконання.

У результаті роботи було розроблено повноцінний програмний продукт, що відповідає сучасним вимогам до застосунків продуктивності. Реалізовано ключові функції, такі як створення списків справ, додавання підзадач, встановлення дедлайнів , а також система нагадувань з push-сповіщеннями.

Технічна реалізація була спрямована на досягнення простоти у використанні, стабільної роботи та доступності для широкої аудиторії. Було використано сучасні засоби, що дозволили забезпечити плавну роботу застосунку та автономне збереження даних без необхідності у серверній інфраструктурі. Завдяки цьому продукт не потребує постійного інтернет-з’єднання, що підвищує його зручність у повсякденному використанні.

Отримане рішення демонструє високий рівень відповідності поставленим вимогам і потенціал до подальшого розвитку. Застосунок може використовуватись як повсякденний цифровий помічник, а також як основа для подальших досліджень або комерційного розширення. В майбутньому, застосунок буде вдосконалений шляхом впровадження синхронізації між пристроями, підтримки спільного доступу до завдань, інтеграції з календарями та іншими сервісами планування. Таким чином, результати виконаної роботи засвідчують доцільність і актуальність розробки в контексті сучасних потреб користувачів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Що таке To-Do List. *Checkify.com*: вебсайт. URL: https://checkify.com/blog/what-is-a-todo-list/ (дата звернення 09.01.25).
2. Що таке Flutter. *Foxminden.ua*: вебсайт. URL: https://foxminded.ua/flutter-shcho-tse/ (дата звернення 13.02.25).
3. Пояснення за Apache Hive. *Techukraine*.net: вебсайт. URL: https://surl.li/ufnxye (дата звернення 13.02.25).
4. Android Studio – вікіпедія. *uk.wikipedia.org*: вебсайт. URL: https://uk.wikipedia.org/wiki/Android_Studio (дата звернення 13.02.25).
5. Tz.local – Dart Package. *Pub.dev*: вебсайт. URL: https://pub.dev/packages/timezone (дата звернення 14.05.25).
6. Dart documentation. *Dart.dev*: вебсайт. URL: https://dart.dev/docs (дата звернення 13.02.25).
7. Мінімальна заробітна плата в Україні станом на 2025 рік. *index.minfin.com*: вебсайт. URL: https://index.minfin.com.ua/ua/labour/salary/min/ (дата звернення 16.05.25).

ДОДАТКИ

Додаток А

Лістинг програмного коду класу “Завдання”

```
@HiveType(typeId: 0)
class Task extends HiveObject {
  @HiveField(0)
  final String id;

  @HiveField(1)
  String title;

  @HiveField(2)
  String subTitle;

  @HiveField(3)
  DateTime createdAtTime;

  @HiveField(4)
  DateTime createdAtDate;

  @HiveField(5)
  bool isCompleted;

  Task({
    required this.id,
    required this.title,
    required this.subTitle,
    required this.createdAtTime,
    required this.createdAtDate,
    required this.isCompleted,
  });

  factory Task.create({
    required String? title,
    required String? subTitle,
    DateTime? createdAtTime,
    DateTime? createdAtDate,
  }) => Task(
        id: const Uuid().v1(),
        title: title ?? "",
        subTitle: subTitle ?? "",
        createdAtDate: createdAtDate ?? DateTime.now(),
        createdAtTime: createdAtTime ?? DateTime.now(),
        isCompleted: false,
      );
}
```

Додаток Б

Лістинг програмного коду фрагменту визначення класу Task

```
@HiveType(typeId: 0)
class Task extends HiveObject {
  @HiveField(0)
  final String id;
  @HiveField(1)
  String title;
  @HiveField(2)
  String subTitle;
  @HiveField(3)
  DateTime createdAtTime;
  @HiveField(4)
  DateTime createdAtDate;
  @HiveField(5)
  bool isCompleted;

  Task({
    required this.id,
    required this.title,
    required this.subTitle,
    required this.createdAtDate,
    required this.createdAtTime,
    required this.isCompleted,
  });

  factory Task.create({
    required String? title,
    required String? subTitle,
    DateTime? createdAtTime,
    DateTime? createdAtDate,
  }) =>
      Task(
        id: const Uuid().v1(),
        title: title ?? "",
        subTitle: subTitle ?? "",
        createdAtDate: createdAtDate ?? DateTime.now(),
        createdAtTime: createdAtTime ?? DateTime.now(),
        isCompleted: false,
      );
}
```

Додаток В

Лістинг програмного коду функції scheduleNotification

```
Future<void> scheduleNotification({
  required int id,
  required String title,
  required String body,
  required DateTime scheduledTime,
  String? payload,
}) async {
  ...
  await flutterLocalNotificationsPlugin.zonedSchedule(
    id,
    title,
    body,
    tz.TZDateTime.from(scheduledTime, tz.local),
    const NotificationDetails(...),
    androidAllowWhileIdle: true,
    uiLocalNotificationDateInterpretation:
        UILocalNotificationDateInterpretation.absoluteTime,
    matchDateTimeComponents: DateTimeComponents.time,
    payload: payload,
  );
  ...
}
```