Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

**Бакалаврська дипломна робота**

на тему:

«Розробка кросплатформного мобільного застосунку для управління читацькою активністю»

Виконала: студентка 4 курсу
групи ІПІ-20б спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки спеціальності)
Олійник І.М.
(прізвище та ініціали)
Керівник: к.т.н., доцент каф. ПЗ Романюк О.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент: к.т.н., доцент каф. ЗІ Войтович О.П.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

**Допущено до захисту**

Завідувач кафедри ПЗ

д.т.н., проф. Романюк О.Н.

(ініціали та прізвище)

« 10 » червня 2024 р.

ВНТУ – 2024

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
О. Н. Романюк
« 12 » березня 2024 р.

**З А В Д А Н Н Я**
**НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТЦІ**

Олійник Ірині Миколаївні

1. Тема роботи – «Розробка кросплатформного мобільного застосунку для управління читацькою активністю».

Керівник роботи: Романюк Оксана Володимирівна, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 11 березня 2024 р. № 80.

2. Строк подання студентом роботи 3 червня 2024.

3. Вихідні дані до роботи: тип програми – кросплатформний мобільний застосунок; модель розробки – ітеративна; засоби організації даних програми – фреймворки Flutter, Firebase Auth та Firestore; вхідні дані: вподобані користувачем книжки; вихідні дані: аналітика читацької активності; середовище розробки – Visual Studio Code; мова програмування – Dart; програмне забезпечення для симуляції мобільних пристроїв – Xcode.

4. Зміст текстової частини: вступ; обґрунтування вибору методу розробки та постановка задач дослідження; розробка методів та алгоритмів роботи програмного продукту; розробка програмного забезпечення; тестування програмного застосунку; висновки; список використаних джерел; додатки.

5. Перелік ілюстративного матеріалу: титульний слайд; актуальність теми дослідження; мета, об'єкт та предмет розробки; задачі дослідження; новизна отриманих результатів; практична цінність отриманих результатів; порівняльний аналіз програм-аналогів; модель системи; структура застосунку; метод і блок-схема алгоритму пошуку та збереження книги; метод і блок-схема алгоритму для відслідковування читання книги; блок-схема алгоритму збереження запису в календар; головна сторінка застосунку; сторінка бібліотеки користувача;

тестування програми; апробація та публікація матеріалів бакалаврської дипломної роботи; висновки; фінальний слайд.

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | Завдання видала | Завдання прийняла |
| 1-4 | Романюк О.В., к.т.н., доцент кафедри ПЗ | 13.03.24 | 03.06.24 |

7. Дата видачі завдання 13 березня 2024 р.

## КАЛЕНДАРНИЙ ПЛАН

| Ч.ч. | Назва етапів бакалаврської дипломної роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Аналіз стану питання та постановка задач дослідження | 14.03.24 – 25.03.2024 | виконано |
| 2 | Розробка структури та моделі застосунку | 26.03.2024 – 02.04.2024 | виконано |
| 3 | Розробка методів та алгоритмів пошуку і зберігання книг з використанням Google Books API та відслідковування читання книг | 03.04.2024 – 20.04.2024 | виконано |
| 4 | Розробка алгоритму збереження запису в календар | 21.04.2024 – 28.04.2024 | виконано |
| 5 | Аналіз та вибір засобів для реалізації програмного продукту | 29.04.2024 – 02.05.2024 | виконано |
| 6 | Програмна реалізація | 03.05.2024 – 16.05.2024 | виконано |
| 7 | Тестування програмного застосунку | 17.05.2024 – 23.05.2024 | виконано |
| 8 | Оформлення матеріалів до захисту БДР | 24.05.2024 – 30.05.24 | виконано |

**Студентка** _____ **Олійник І.М.**
(підпис) (ініціали та прізвище)

**Керівник бакалаврської дипломної роботи** _____ **Романюк О.В.**
( підпис ) (ініціали та прізвище)

# АНОТАЦІЯ

Бакалаврська дипломна робота складається з 60 сторінок формату А4, на яких є 35 рисунків, 3 таблиці, список використаних джерел містить 21 найменування.

У бакалаврській дипломній роботі проведено аналіз стану мобільних застосунків управління читацькою активністю. Сформульовано мету досліджень – підвищення якості моніторингу та управління читацькою активністю за рахунок розробки та використання спеціалізованої мобільної системи, яка дозволяє відслідковувати прогрес читання та аналізувати читацькі звички.

Уперше запропоновано відображати прогрес читання книг у вигляді календаря, який дозволяє користувачеві візуально відстежувати свої щоденні читацькі сесії та переглядати їх у зручному форматі. Удосконалено метод пошуку та зберігання книг, який на відміну від відомих, використовує Google Books API, що дозволило уникнути необхідності збереження книг на власному сервері. Удосконалено метод відслідковування читання книг, який на відміну від відомих методів, дозволяє вибирати сторінки з яких починати та завершувати читання та надає можливість поставити таймер на паузу, що забезпечує більш точне та ефективне визначення читацького прогресу.

Розроблено алгоритми авторизації та пошуку книги, а також методи та алгоритми пошуку і зберігання книг з використанням Google Books API та відслідковування читання книг

Застосунок розроблено з використанням мови Dart та інтегрованого середовища розробки Visual Studio Code. Для забезпечення організації системи зберігання даних використовуються хмарні технології Firebase.

Отримані в бакалаврській дипломній роботі результати можуть бути використані для вдосконалення відстеження процесу читання книг, включаючи сфери освіти та розваг.

Ключові слова: кросплатформна розробка, читання, читацька активність.

# ABSTRACT

The bachelor thesis consists of 60 pages of A4 format, on which there are 35 pictures, 3 tables, the list of used sources contains 21 items.

The bachelor's thesis analyzed the state of mobile applications for managing readership. The goal of the research is formulated – to improve the quality of monitoring and management of reading activity due to the development and use of a specialized mobile system that allows monitoring the progress of reading and analyzing reading habits.

For the first time it is proposed to display the progress of reading books in the form of a calendar, which allows the user to visually track his daily reading sessions and view them in a convenient format. The method of searching and storing books has been improved, which, unlike the known ones, uses the Google Books API, which made it possible to avoid the need to store books on your own server. The method of tracking book reading has been improved, which, unlike known methods, allows you to choose the pages from which to start and finish reading and provides the ability to pause the timer, which ensures a more accurate and effective determination of reading progress.

Algorithms for authorizing and searching for books, as well as methods and algorithms for searching and storing books using the Google Books API and tracking book reading have been developed

The application is developed using the Dart language and the Visual Studio Code integrated development environment. Firebase cloud technologies are used to ensure the organization of the data storage system.

The results obtained in the bachelor's thesis can be used to improve the tracking of the book reading process, including the fields of education and entertainment.



Keywords: cross-platform development, reading, reading activity.

ЗМІСТ

# ВСТУП

**Обґрунтування вибору теми дослідження.** Читання є невід'ємною складовою людського життя. Читання – ідеальний спосіб розслабитися і заспокоїти переживання, які існують всередині людини. Нещодавнє дослідження показало, що лише шість хвилин на день читання зменшують м'язову напругу, а також уповільнюють частоту серцевих скорочень читачів, загалом зменшуючи рівень стресу на 68% [1].

Популярність читання в сучасному світі залишається актуальною темою, особливо на тлі технологічного прогресу та зміни культурних звичок [2]. Сприяючи розвитку мислення, розширенню словникового запасу та розвитку емоційного інтелекту, читання залишається невід'ємною частиною культурного прогресу.

Незважаючи на технологічні та культурні зміни, популярність читання залишається важливим елементом особистого та соціального розвитку, сприяючи збагаченню розумових здібностей, розвитку емоційного інтелекту та глибшому розумінню культури. Люди шукають способи ефективного використання свого часу та підтримки своїх особистих цілей, використовуючи мобільні трекери для відстеження прогресу читання книг, які допомагають користувачам контролювати свою читацьку активність та відстежувати зміни у їхніх читацьких звичках.

Одним з найкращих рішень, яке читач може прийняти для покращення свого читацького життя, є ведення читацького щоденника. На багатьох рівнях це допомагає організувати книги, а ведення власного обліку читання може звільнити розум від організаційних питань, щоб мати можливість зосередитися на більшій кількості читання. Більше часу на читання означає менше часу, витраченого на стрес з приводу організації [3].

Наявні аналоги застосунків управління читацькою активністю не надають достатнього функціоналу. Тому, розробка власної реалізації застосунку управління читацькою активністю є актуальною, оскільки буде надавати більше

можливостей для персоналізації та адаптації, тому що буде розроблена з урахуванням конкретних уподобань та стилів читання аудиторії.

**Зв'язок роботи з науковими програмами, планами, темами.** Робота виконувалася згідно з планом виконання наукових досліджень на кафедрі програмного забезпечення.

**Мета та завдання дослідження.** Метою є підвищення якості моніторингу та управління читацькою активністю за рахунок розробки та використання спеціалізованої мобільної системи, яка дозволяє відслідковувати прогрес читання та аналізувати читацькі звички.

Відповідно до поставленої мети потрібно вирішити такі **завдання**:

– провести аналіз стану предметної області та аналогічних застосунків для відстеження читацької активності;

– розробити модель та структуру програмного застосунку;

– розробити метод і алгоритм пошуку та зберігання книг з використанням Google Books API;

– розробити метод та алгоритм відслідковування читання книг;

– реалізувати відображення прогресу читання книг у вигляді календаря;

– розробити графічний інтерфейс мобільного застосунку;

– розробити програмне забезпечення мобільного застосунку для відстеження читацької активності;

– провести тестування мобільного застосунку згідно поставлених задач;

– розробити інструкцію користувача та встановити користувацькі системні вимоги для програми управління читацькою активністю.

**Об'єктом дослідження є** процес управління читацькою активністю.

**Предметом дослідження є** методи і засоби реалізації кросплатформного мобільного застосунку для управління читацькою активністю.

**Методи дослідження.** У процесі досліджень використовувалися: порівняння функціональних характеристик програм-аналогів для виявлення можливих недоліків та формулювання завдань на їх вдосконалення; теорія алгоритмів для розробки та візуалізації алгоритмів роботи програмного

забезпечення; комп’ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

**Новизна отриманих результатів.**

1. Уперше запропоновано відображення прогресу читання у вигляді календаря, який дозволяє користувачеві візуально відстежувати свої щоденні читацькі сесії та переглядати їх у зручному форматі.

2. Удосконалено метод пошуку та зберігання книг, який на відміну від відомих, використовує технологію Google Books API, що дозволило запобігти необхідності збереження книг на власному сервері.

3. Удосконалено метод відслідковування читання книг, який на відміну від відомих методів, дозволяє вибирати сторінки, з яких починати та завершувати читання та надає можливість поставити таймер на паузу, що забезпечує більш точне та ефективне визначення читацького прогресу.

**Практична цінність отриманих результатів.** Практична цінність отриманих результатів полягає в тому, що на основі отриманих в бакалаврській дипломній роботі теоретичних положень запропоновано алгоритми та розроблено програмні засоби для управління читацькою активністю.

**Особистий внесок здобувача.** Усі наукові результати, що викладені у бакалаврській дипломній роботі, отримано автором самостійно. У наукових працях, опублікованих у співавторстві, автору належать такі результати: переваги кросплатформної розробки над нативною [4]; імплементація трекінг-процесу для управління читацькою активністю [5].

**Апробація матеріалів бакалаврської дипломної роботи**. Результати роботи доповідалися на:

– LIII Всеукраїнській науково-технічній конференції факультету інформаційних технологій та комп'ютерної інженерії (Вінниця, 2024 р.) [4];

– Молодь в науці: дослідження, проблеми, перспективи (Вінниця, 2024 р.) [5].

**Публікації.** За тематикою дослідження опубліковано 2 наукові праці у збірниках матеріалів конференцій.

# 1 ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗРОБКИ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

## 1.1 Аналіз стану мобільних застосунків управління читацькою активністю

Мобільні застосунки для управління читацькою активністю в останні роки зазнали значного розвитку. Існують різноманітні платформи, що надають користувачам можливість відстежувати свою читацьку активність, визначати прочитані книги, вести список побажань, отримувати рекомендації та багато іншого. Деякі з них пропонують соціальні функції, які дозволяють користувачам ділитися своїм прогресом з друзями та родиною, створюючи атмосферу змагання та підтримки.

З появою цифрового читання та мобільних пристроїв програми для щоденників читання стають все більш популярними серед книголюбів. Вони забезпечують зручний спосіб запису та зберігання інформації про прочитані книги, не потребуючи ручки та паперу [6].

Ключовий аспект аналізу стану засобів мобільного застосунку в цій галузі – це їх функціональність. Деякі застосунки пропонують базовий набір функцій, таких як ведення списку прочитаних книг та оцінювання їх, тоді як інші можуть мати розширений функціонал, включаючи можливість обміну враженнями про прочитані книги з іншими користувачами, участь у відгуках та дискусіях, а також персоналізовані рекомендації.

Переваги використання мобільних застосунків для моніторингу та управління читацькою активністю:

1. Постійний доступ до інформації: доступ до списку прочитаних книг, оцінок, цитат, нотаток; відстеження прогресу читання.
2. Персоналізація: налаштування інтерфейсу читання; формування власної бібліотеки.
3. Соціальна взаємодія: обмін враженнями про прочитані книги з іншими користувачами; участь у читацьких клубах; спілкування з авторами

4. Додаткові можливості: використання словників та перекладачів; розвиток та навчання навичок читання.

Отже, аналіз стану засобів мобільних застосунків управління читацькою активністю підкреслює важливість наявності різноманітних інструментів з різним функціоналом та рівнями зручності використання. Вибір конкретного застосунку має відповідати індивідуальним потребам та вимогам кожного користувача, забезпечуючи ефективне та комфортне ведення їх читацького життя.

### 1.2 Порівняльний аналіз аналогів

Порівняльний аналіз функціоналу застосунків для управління читацькою активністю дозволить з’ясувати переваги та недоліки кожного із них, а також визначити, які саме функції є найбільш корисними для користувачів. Такий підхід допоможе створити базу для подальшого вдосконалення та розвитку мобільних застосунків для читачів, забезпечуючи їм зручний та ефективний інструмент для ведення читацького життя.

Для визначення конкурентних переваг мобільного застосунку розглянемо три популярні мобільні застосунки для відслідковування прочитаного:

1) Goodreads;
2) TBR – Bookshelf;
3) The Storygraph.

«Goodreads» – це соціальна мережа для любителів книг, де користувачі можуть відкрити обліковий запис, додавати книги до свого списку прочитаних, відгуків, а також оцінювати та рецензувати книги. Заснований в 2006 році і згодом придбаний компанією Amazon, «Goodreads» став однією з найбільш популярних платформ для читачів по всьому світу.

Кожна книга у «Goodreads» має спеціальну сторінку, на якій показано загальну оцінку, яку користувачі дали книзі, короткий зміст, інформацію про автора та відгуки від спільноти застосунку [7] (рисунок 1.1).

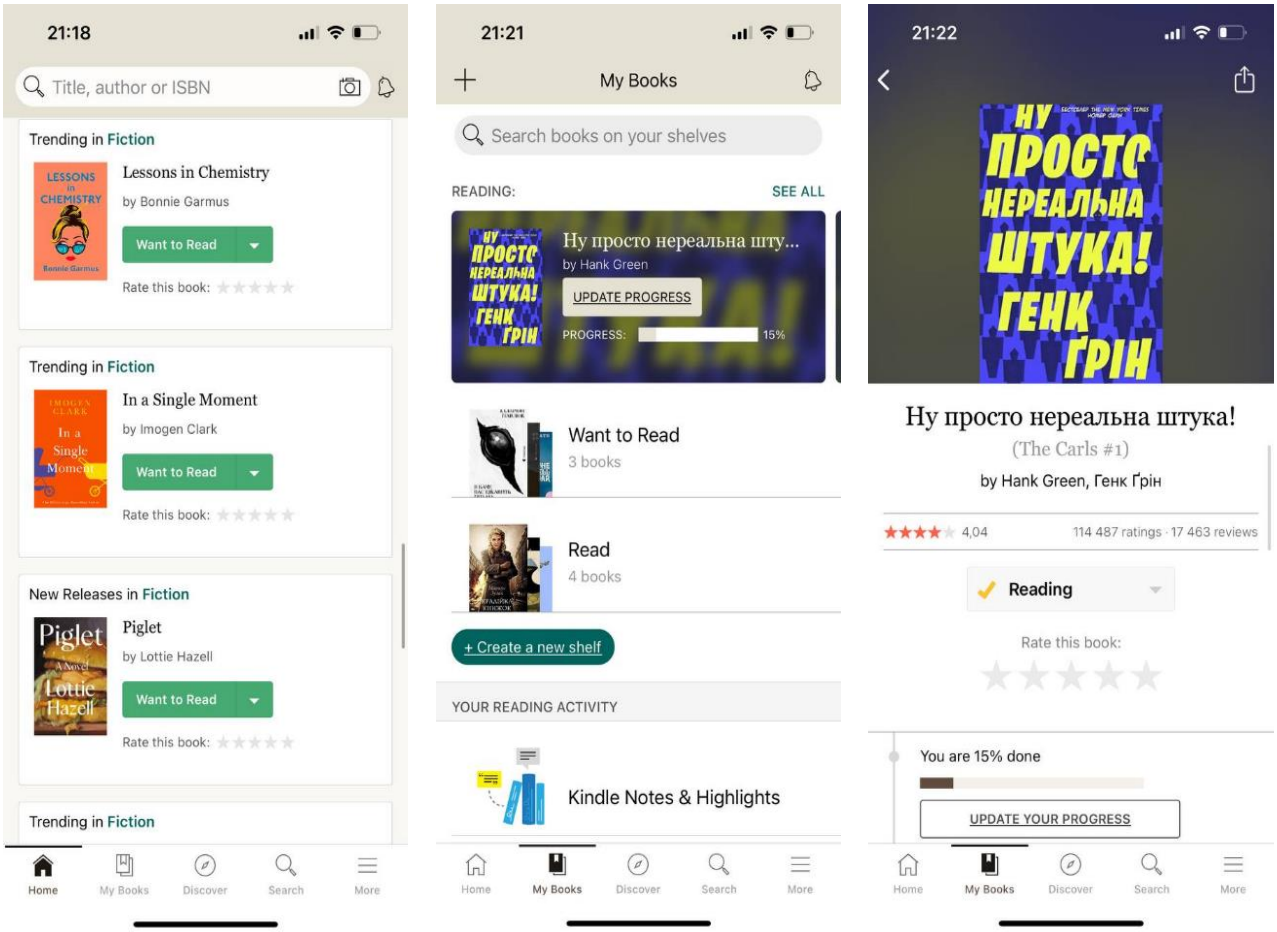



Рисунок 1.1 – Застосунок «Goodreads»

«TBR – Bookshelf» – це інноваційний мобільний застосунок, який створено з метою полегшення організації та керування списком книг, які користувачі планують прочитати у майбутньому [8]. Користувач має можливість бачити загальний вигляд книги, які «тропи» вона містить, частоту повторного читання та 4 рейтинги за жанрами.

Основні функції застосунку включають в можливість додавання книг за допомогою сканування штрих-кодів або пошуку за назвою, автором чи ISBN. Крім того, користувач може встановлювати пріоритети для кожної книги, вказуючи, яка з них потрібна більше.

Відображення роботи програмного продукту «TBR – Bookshelf» показане на рисунку 1.2.

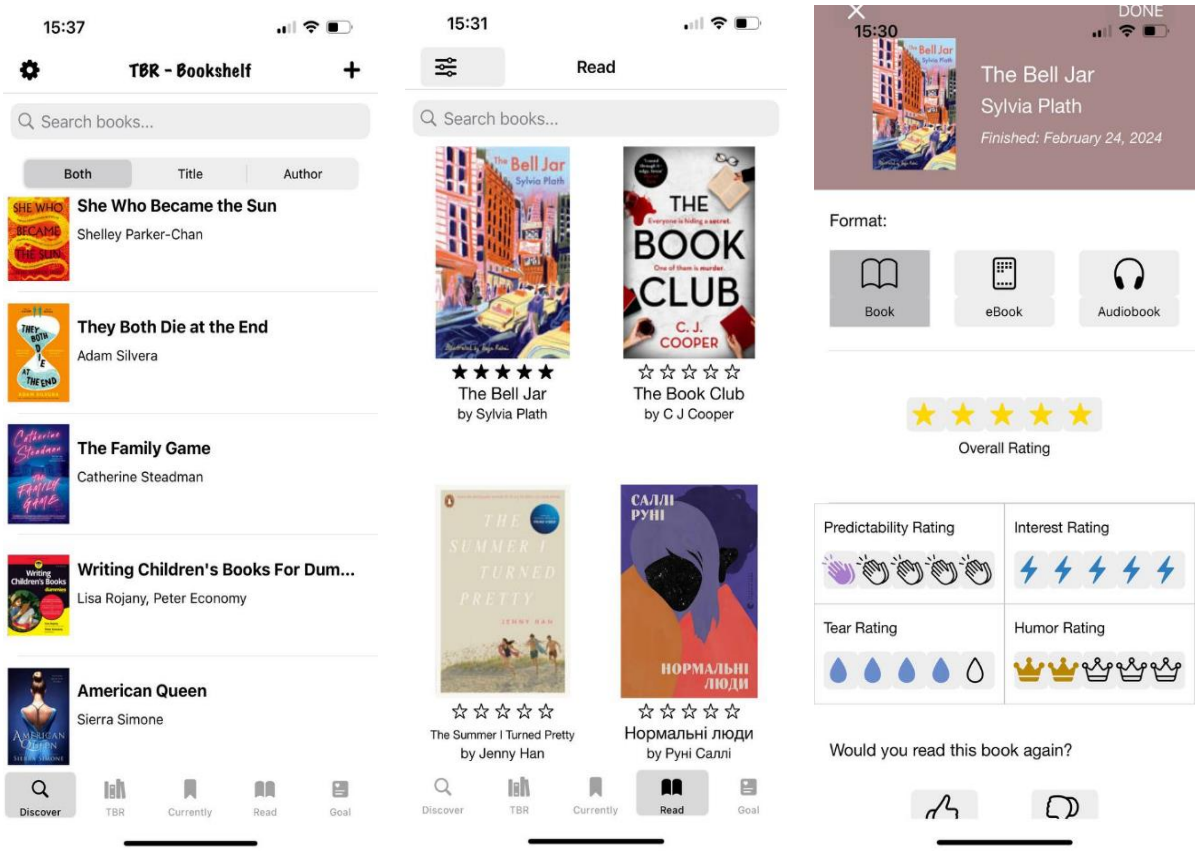



Рисунок 1.2 – Застосунок «TBR - Bookshelf»

«The Storygraph» – це інтерактивна платформа для читачів, яка дозволяє створювати особистий читацький журнал. Користувачі можуть відстежувати свій прогрес у прочитанні книг та залишати відгуки та оцінки. Одним із ключових функціональних елементів є можливість легко відстежувати та дізнаватися про звички читання за допомогою широкого вибору діаграм і графіків [9].

«The Storygraph» надає детальну статистику щодо кількості книг, які прочитав користувач за певний період, його переваги щодо жанрів, а також середню тривалість читання кожної книги. Такі дані допомагають користувачам краще розуміти свої читацькі звички та знаходити нові способи насолоджуватися світом літератури.

Відображення роботи програмного продукту «The Storygraph» наведено на рисунку 1.3.

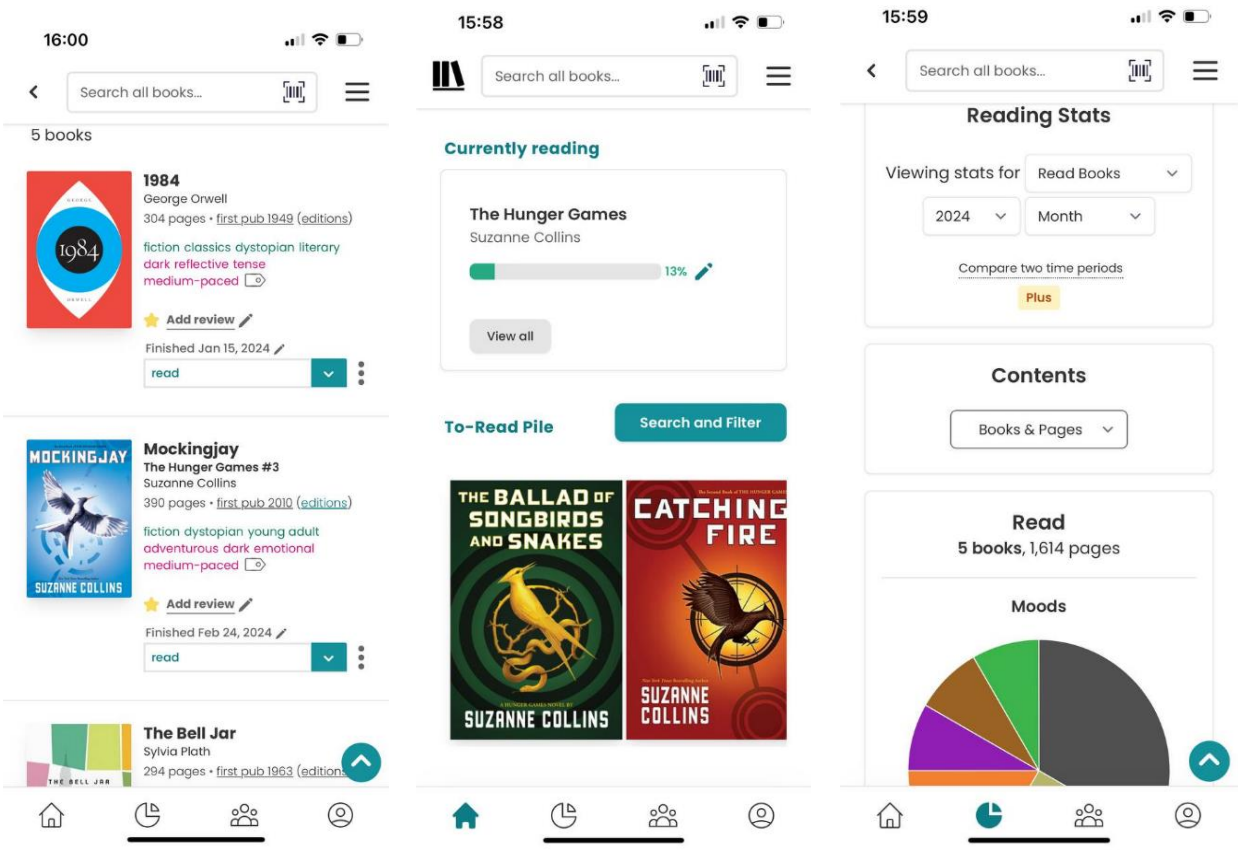



Рисунок 1.3 – Застосунок «The Storygraph»

У результаті розгляду та аналізу аналогів було створено таблицю 1.1 для порівняння їхнього функціоналу із розроблюваним мобільним застосунком «BooBook».

Таблиця 1.1 – Функціональні характеристики мобільних застосунків

| Критерій | Goodreads | TBR - Bookshelf | The Storygraph | BooBook |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |
| Зміна критерію кількості сторінок у книзі | - | - | - | + |
| Відслідковування прочитаного за кількістю сторінок | + | - | + | + |

Продовження таблиці 1.1

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| Наявність календаря | - | - | - | + |
| Розширена аналітика читацької активності | - | + | + | + |
| Можливість створювати персоналізовані списки за певними критеріями | + | - | - | + |
| Можливість переглядати відгуки читачів певної книги | + | - | - | + |
| Сумарний коефіцієнт | 3 | 1 | 2 | 6 |

Аналізуючи вищенаведені дані, «BooBook» має значно вищий сумарний коефіцієнт відносно аналогів «Goodreads», «The Storygraph» та «TBR - Bookshelf». Тому розробка мобільного застосунку «BooBook» є актуальною, оскільки отриманий продукт зможе покрити недоліки існуючих рішень та відкрити нові можливості для покращення досвіду користувачів та застосування мобільного застосунку управління читацькою активністю.

Отже, розробка програмних засобів для мобільного застосунку, спрямованих на управління читацькою активністю, не лише корисна для самого користувача, але й може мати значний позитивний вплив на бізнесову діяльність. Зокрема, застосунок відкриває перед підприємствами широкі можливості для взаємодії зі своїми клієнтами. Збираючи дані про читацькі вподобання та активність, бізнес може створювати персоналізовані пропозиції, рекомендації та промо-акції, які краще відповідатимуть потребам своїх клієнтів. Крім того, величезна кількість зібраних даних може бути використана для аналізу ринку та прогнозування тенденцій, що дозволить бізнесу більш ефективно планувати свою стратегію розвитку та реагувати на зміни в попиті та потребах споживачів. Таким чином, програмні засоби моніторингу та управління читацькою

активністю можуть стати потужним інструментом для підвищення ефективності та конкурентоспроможності бізнесу.

### 1.3 Аналіз методів розв’язання задачі

Розробка реалізації мобільної системи для управління читацькою активністю вимагає ретельного аналізу і розгляду різних підходів до вирішення проблеми. Існують різноманітні методи та рішення підходу розробки, які мають свої переваги та недоліки.

Одним з варіантів вирішення проблеми розробки програмних модулів для реалізації мобільної системи управління читацькою активністю є створення окремого сервісу, який буде відповідати за всю логіку пов’язану з потоком даних (збереження, обробка). Такий сервіс доволі легко інтегрувати, проте він вимагає додаткових зусиль для реалізації та більшого обсягу роботи. Крім того, цей підхід вимагає використання сторонній сервісів або платформ для розміщення серверів, на яких буде відбуватись обробка та збереження всіх даних. Також цей варіант потребує більше місця на серверах задля збереження додаткової інформації та зв’язків. Тому такий метод не варто застосовувати.

Інший підхід полягає в інтегруванні вже готового сервісу, який реалізує обробку та збереження даних. Такий підхід дозволяє розробникам зосередитись саме на розробці модулів мобільного застосунку. Користувачі такого сервісу можуть інтегрувати вже готове рішення, що дозволить зменшити ресурси на розробку та обсяг часу порівняно з попереднім методом. Однак цей метод не є дуже ефективним, через застосування великої кількості необхідної пам’яті. Користувачі сервісу не можуть накласти обмеження на створення додаткових даних і через це база даних може бути наповнена непотрібними або дубльованими даними, що в подальшому призведе до уповільнення роботи сервісу. Цей метод краще застосовувати у випадках, коли дані можна створити в одному екземплярі. Цей метод не підходить для реалізації системи, дані якої постійно будуть оновлюватись та збільшуватись у обсязі.

Один з поширених рішень є делегування неоднакових задач різним сервісам. Такий метод дозволяє інтегрувати сервіси в відповідні модулі, для подальшого застосування в конкретних цілях, завдяки цьому є можливість уникнути дублювання та накопичення непотрібної інформації завдяки розподілу задач. Також цей метод є зручним для масштабування, розширення функціональності продукту. З недоліків варто зазначити витрату часу для пошуку ідеальних готових сервісів, які націлені для вирішення конкретної проблеми користувача, а також необхідність залучення додаткових ресурсів. Проте такий метод дозволяє створити унікальну платформу з різноманітним функціоналом.

Розглянувши переваги та недоліки кожного методу, було вирішено використати метод з делегуванням неоднакових задач різним сервісам для реалізації мобільної системи управління читацькою активністю. Таких підхід дозволяє максимально швидко та комфортно оперувати необхідними даними без потреби зберігати дублікати та зайві дані. Більше того, такий метод розробки дозволить створити унікальну платформу з унікальним функціоналом.

## 1.4 Постановка задач для розробки застосунку управління читацькою активністю

Враховуючи аналіз функціоналу застосунків управління читацькою активністю, було визначено завдання, які необхідно виконати для розробки мобільної системи, а саме:

- провести аналіз стану предметної області та аналогічних застосунків для відстеження читацької активності;
- розробити модель та структуру програмного застосунку;
- розробити метод і алгоритм пошуку та зберігання книг з використанням Google Books API;
- розробити метод та алгоритм відслідковування читання книг;
- реалізувати відображення прогресу читання книг у вигляді календаря;
- розробити графічний інтерфейс мобільного застосунку;

– розробити програмне забезпечення мобільного застосунку для відстеження читацької активності;

– провести тестування мобільного застосунку згідно поставлених задач;

– розробити інструкцію користувача та встановити користувацькі системні вимоги для програми управління читацькою активністю.

Технічне завдання на розробку наведено в додатку А.

### 1.5 Висновки

У розділі було проаналізовано поточний стан мобільних застосунків для управління читацькою активністю, а саме: «Goodreads», «TBR – Bookshelf» та «The Storygraph». У результаті порівняння було оцінено їх переваги й недоліки порівняно з запропонованим рішенням, і, таким чином, була виявлена доцільність створення власного програмного рішення. Враховуючи цю інформацію було сформовано перелік завдань, які необхідно виконати для розробки власного мобільного застосунку.

# 2 РОЗРОБКА МЕТОДІВ ТА АЛГОРИТМІВ РОБОТИ ПРОГРАМНОГО ПРОДУКТУ

## 2.1 Визначення даних для роботи програмного застосунку

Вхідні та вихідні дані відіграють важливу роль у функціонуванні будь-якої програми, забезпечуючи її зв’язок із зовнішнім середовищем. Вхідні дані є інформаційним потоком, який надходить до програми від користувача. Вихідні дані, у свою чергу, навпаки, є результатом роботи програми, що представлений у зрозумілому для користувача форматі. Розглянемо їх детальніше.

Вхідні дані програми управління читацькою активністю складаються з інформації про обрану користувачем книгою та даних про процес читання. Інформація про книгу включає назву книги, ім’я автора, зображення палітурки, кількістю сторінок та коментарі інших читачів. Ці дані дозволяють користувачам легко ідентифікувати книги, які вони зараз читають або планують прочитати.

Після вибору книги для читання відбувається відслідковування процесу читання. За допомогою інтерфейсу програми користувач активовує таймер, вводить початковий номер сторінки, а після завершення читання – номер крайньої сторінки. Аналізуючи введені дані, програма показує орієнтовний час завершення читання, а також відображає перелік всіх сесій читання обраної книги.

Крім функції відслідковування читання книги користувач має можливість переглядати підсумки своїх щоденних сесій читання у вигляді календаря. Користувач може бачити свій читацький прогрес у вигляді відображення поточного тижня або місяця.

Вихідні дані програми формуються на основі вхідних даних і надають аналітику прочитаної книги, що містить інформацію про кількість сторінок та загальний час читання, середню швидкість читання, дату початку та закінчення читання. Також користувач може переглядати перелік всіх добавлених книг та сортувати їх за критерієм прочитанності. Вихідні дані дозволяють користувачам аналізувати їх читацький прогрес, оцінювати свої читацькі звички та створювати

плани на майбутнє читання. Вони надають корисну інформацію для саморозвитку та удосконалення читацьких навичок, дозволяючи користувачам порівнювати свій прогрес у читанні різних книг, визначати, скільки часу вони витрачають на кожну книгу в порівнянні з іншими, а також відстежувати зміни у своїй читацькій активності з часом.

Отже, вхідні дані визначають потреби користувачів та вимоги до програми, тоді як вихідні дані дозволяють користувачам отримувати результати та використовувати їх у своїй діяльності. Кросплатформний мобільний застосунок для управління читацькою активністю надає можливість користувачам легко та зручно відстежувати свою читацьку активність на будь-якому пристрої, незалежно від їх операційної системи.

### 2.2 Розробка моделі системи

Модель системи розроблюваного застосунку може бути представлена за допомогою UML-діаграми прецедентів. Діаграма прецедентів – це структурна діаграма, яка використовується в моделюванні систем для візуалізації функціональних вимог до системи з точки зору користувачів. Вона показує взаємодії між системою та її користувачами [10].

Основними компонентами діаграми прецедентів є:

– актори – суб’єкти, які взаємодіють з системою, наприклад користувачі, інші системи або зовнішні сервіси;

– прецеденти – окремі функціональні можливості або дії, які виконує система;

– взаємодії – стрілки або лінії, що показують взаємодію між акторами та прецедентами.

На рисунку 2.1 зображено діаграму прецедентів, що відображає, як користувач взаємодіє з застосунком для управління читацькою активністю та як він може користуватися різноманітними функціями для ефективного управління своїм читацьким досвідом.

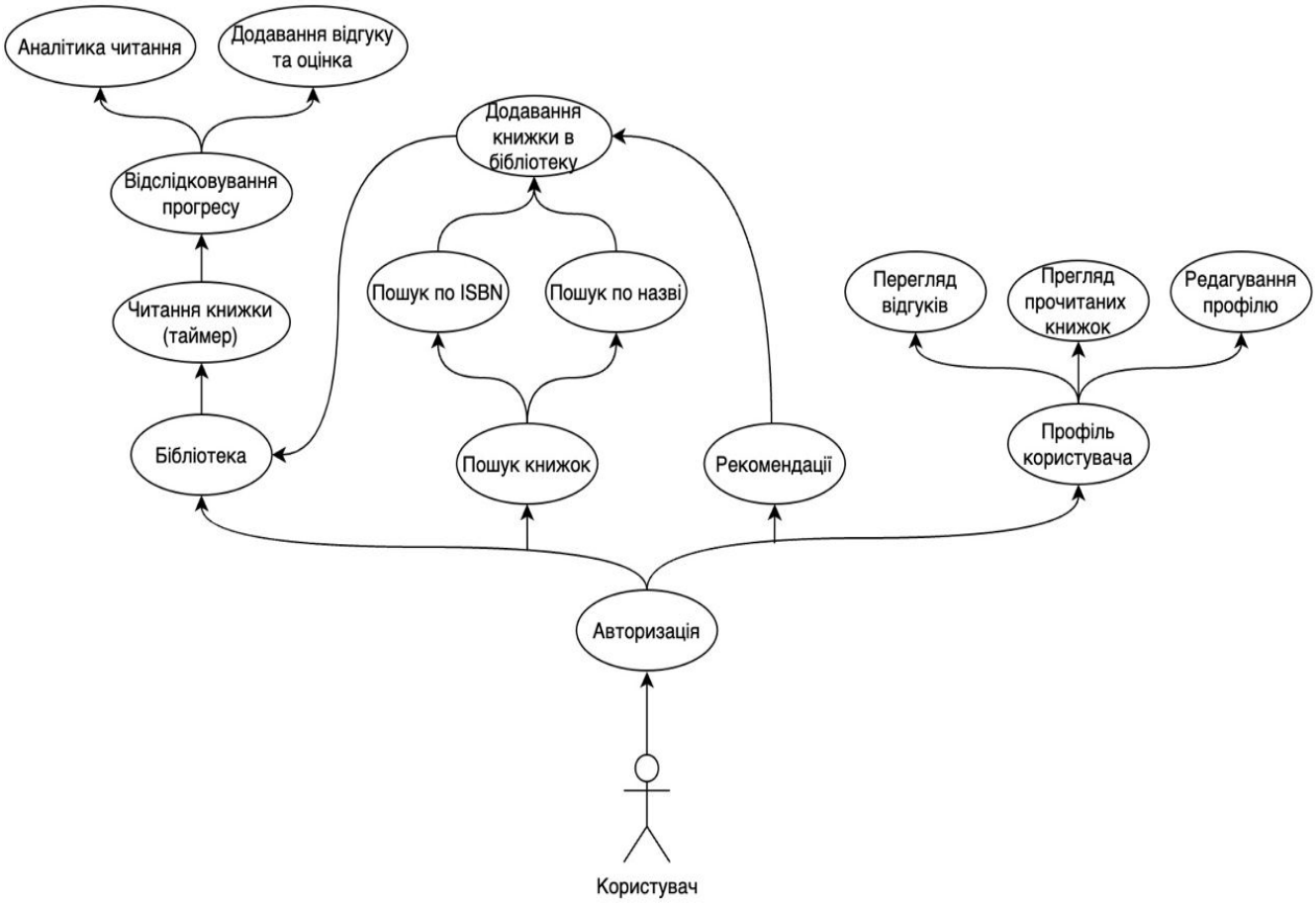



Рисунок 2.1 – Діаграма прецедентів

Користувач – це основний актор, який використовує мобільний застосунок для управління читацькою активністю. Його прецедентами є:

1. Авторизація: користувач повинен авторизуватися в системі, щоб отримати доступ до більшості функцій. Система використовує дані, збережені в сервісі Firebase.

2. Бібліотека: відображає всі книги, які користувач додав до своєї бібліотеки.

3. Читання книжки: відображає книгу, яку користувач зараз читає, а також таймер, який показує, скільки часу користувач читає цю книгу.

4. Відслідковування прогресу: дозволяє користувачу відслідковувати, скільки книг він прочитав, скільки сторінок він прочитав і скільки часу він витратив на читання.

5. Аналітика читання: система надає користувачеві аналітику його читацьких звичок, що включає такі дані, як скільки книг користувач читає щомісяця, які жанри книг користувач читає найчастіше та скільки часу користувач витрачає на читання.

6. Додавання оцінки та відгуку: користувач може залишати відгуки та оцінки про прочитані книги.

7. Пошук книжок: дозволяє користувачу шукати книги за ISBN або назвою.

8. Додавання книжки в бібліотеку: користувач може додати знайдену книжку до бібліотеки.

9. Рекомендації: можливість отримувати персональні рекомендації щодо книг на основі попередніх читань та оцінок.

10. Профіль користувача: можливість користувача переглядати та редагувати особистий профіль в системі, включаючи інформацію про себе та перегляд прочитаних книг разом з залишеними відгуками.

Таким чином, діаграми прецедентів допомагають визначити всі основні функції, які повинна виконувати система, та визначити обсяг робіт для розробки програмного забезпечення.

### 2.3 Розробка структури програмного застосунку

Навігаційний граф – це структура даних, що представляє собою граф, де вузли відображають сторінки або розділи програмного забезпечення, а ребра відображають переходи між цими сторінками. Цей граф допомагає відстежувати шляхи переміщення користувачів по системі та планувати оптимальні маршрути.

Навігаційний граф визначає структуру застосунку, показуючи всі доступні екрани, сторінки або функції, а також зв’язки між ними. Інтерфейс застосунку втілює цю структуру, надаючи користувачам засоби для навігації між різними екранами та взаємодії з функціями. Інтерфейс віртуального пристрою може існувати у формі програмного інтерфейсу і/або протоколу взаємодії двох частин графічної системи [11]. Якщо навігаційний граф добре організований, то

допомагає створити зручний та інтуїтивно зрозумілий інтерфейс для користувачів.

Навігаційний граф мобільного застосунку управління читацькою активністю зображений на рисунку 2.2.

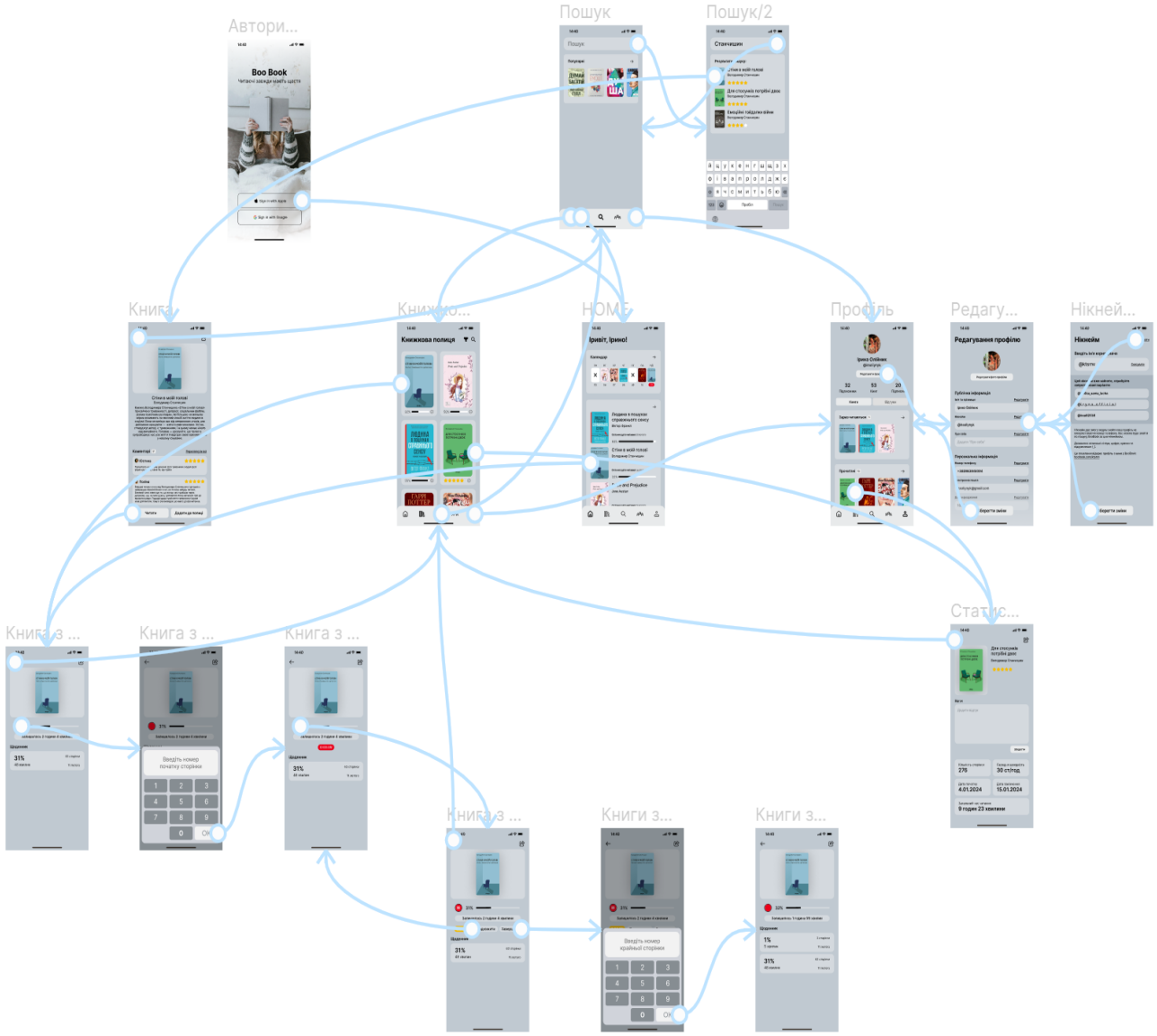



Рисунок 2.2 – Навігаційний граф

Отже, навігаційний граф і інтерфейс застосунку тісно взаємопов’язані: перший допомагає визначити структуру та зв’язки між елементами застосунку, в той час як другий реалізовує цю структуру, забезпечуючи користувачам зручну

навігацію та взаємодію з функціями.

### 2.4 Розробка методу та алгоритму пошуку та зберігання книг

Для розробки мобільного застосунку управління читацької активності необхідно розробити метод та алгоритм для пошуку та зберігання книг, згідно якого буде працювати застосунок.

Відповідно до цього методу користувач вже має спершу авторизуватися для подальшого роботи з застосунком. Щоб почати процес пошуку, необхідно перейти на відповідну сторінку. Система використовує Google API, щоб не зберігати в базі даних список усіх існуючих книг, а застосовувати вже готову.

Після того, як користувач знаходить бажану книгу, система створює в базі даних користувача запис з цією книжок для подальшого відслідковування прогресу.

Блок-схема алгоритму пошуку та зберігання книг зображена на рисунку 2.3. Опишемо її детальніше.

Користувач має увійти в систему, використовуючи Google або Apple акаунт, для подальшого користування. Якщо під час авторизації виникла помилка, система повідомляє користувача про необхідність пройти процес авторизації ще раз.

Щоб додати книжку до свого списку, користувачу необхідно перейти на екран пошуку та ввести назву книги або її автора. Після отримання назви, система одразу робить запит на Google Books API, щоб знайти схожі книжки. Якщо система не знаходить жодного відповідного варіанту, відображається повідомлення, що результатів не знайдено і користувач повинен повторити процес пошуку.

У випадку знаходження книг, з назвою яка схожа на запит, система відображає список книги, які надходять з бази даних Google Books API.

Користувач може вибрати будь-яку книгу для додавання до своєї колекції, що призведе до автоматичного оновлення бази даних задля подальшого відстеження прогресу.

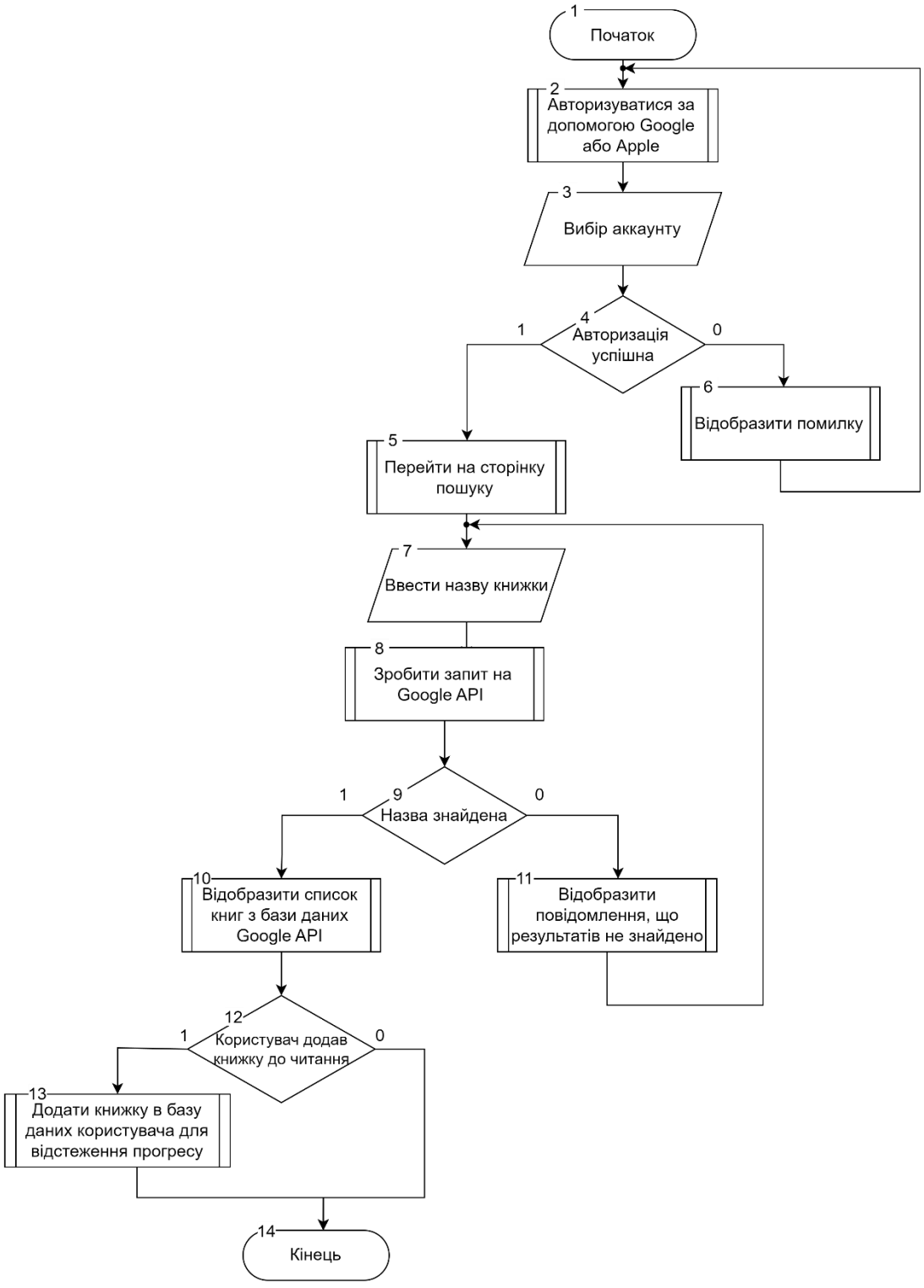



Рисунок 2.3 – Блок-схема алгоритму пошуку та зберігання книг

Таким чином, розроблений метод та алгоритм пошуку та зберігання книг дозволяє повністю автоматизувати систему для пошуку та зберігання книжок, значно підвищуючи ефективність управління читацькою активністю. Завдяки

цьому методу та алгоритму користувачі можуть швидко та легко знаходити необхідну літературу, а потім зберігати її.

### 2.5 Розробка методу та алгоритму відслідковування читання книги

Для кращого розуміння роботи модуля для управління читацькою активністю було розроблено додатковий метод і алгоритм відслідковування читання книги. Цей алгоритм зумовлює процес збору даних задля подальшого показу аналітики читання.

Щоб почати відслідковувати прогрес, користувач має перейти на відповідну сторінку. Система показує приблизний час, за скільки користувач може закінчити читання книжки, задля кращого усвідомлення прогресу та швидкості, а після цього починається сам процес читання. Система запускає таймер та відображає кнопки «Завершити» та «Пауза». Після завершення читання отримується кількість прочитаних сторінок та час. Відносно зібраних даних далі буде відображатись аналітика.

Діаграма класів алгоритму відстеження читання показана на рисунку 2.4.

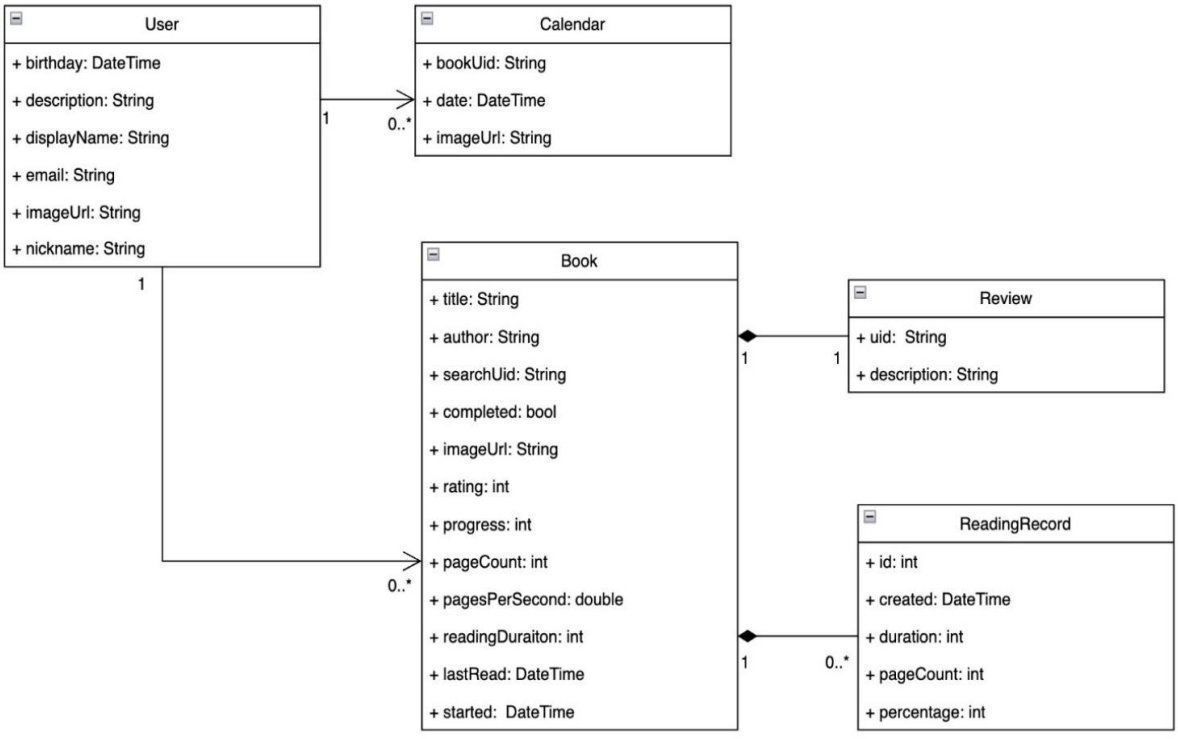



Рисунок 2.4 – Діаграма класів для відстеження процесу читання

Діаграма класів слугує вихідною точкою для розробки блок-схеми алгоритму, особливо якщо потрібно визначити, які методи класів будуть викликані та в якому порядку вони будуть виконуватися для досягнення певного функціонального результату. Тобто, діаграма класів допомагає визначити структуру програми, а блок-схема – послідовність операцій або алгоритмів, які можуть бути реалізовані у межах цієї структури.

Клас User містить такі атрибути:

– birthday – дата народження;
– description – опис профілю;
– displayName – ім’я яке відображатиметься іншим користувачам;
– email – електронна пошта;
– imageUrl – посилання на аватар;
– nickname – унікальне ім’я користувача.

Клас Calendar містить такі атрибути:

– bookUid – унікальне значення для ідентифікування в базі;
– date – дата читання;
– imageUrl – посилання на зображення обкладинки книжки.

Клас Book містить такі атрибути:

– title – назва книжки;
– author – ім’я автора;
– searchUid – унікальне значення для ідентифікування при пошуку;
– completed – чи закінчив користувач читання;
– imageUrl – посилання на зображення обкладинки книжки;
– rating – рейтинг, який залишив користувач;
– progress – прогрес читання;
– pageCount – загальна кількість сторінок;
– pagesPerSecond – кількість прочитаних сторінок за секунду;
– readingDuration – тривалість читання;
– lastRead – дата останнього разу читання;

– started – дата початку читання.

Клас Review містить такі атрибути:

– uid – унікальне значення для ідентифікації в базі даних;

– description – текст відгуку.

Клас ReadingRecord містить такі атрибути:

– id – унікальне значення для ідентифікації в базі даних;

– created – дата створення запису;

– duration – тривалість читання;

– pageCount – кількість прочитаних сторінок;

– percentage – відцоток прочитаних сторінок.

Отже, враховуючи дані, отримані з діаграми класів, розробимо блок-схему алгоритму відслідковування читання книги.

Перш за все, користувачеві необхідно відкрити сторінку з конкретною книгою. Якщо книжку вже прочитано, відбувається перехід на сторінку з аналітикою, в іншому випадку – на сторінку з прогресом читання. Якщо користувач вже читав цю книгу, система розраховує приблизний час, необхідний для завершення, та відображає записи щодо прогресу читання.

Для того, щоб створити запис читання, користувач має розпочати читати книжку. Після цього активовується таймер та з’являється кнопка паузи, при натисканні на яку, таймер зупиняється – і з’являються кнопки «Продовжити» та «Завершити».

При натисканні користувачем на «Завершити», створюється відповідний запис з прогресом читання. Також система визначає, чи прочитав користувач повністю книжку, якщо так, тоді він переходить на сторінку з аналітикою.

Сторінка аналітики включає в себе дані про кількість сторінок, час читання, середню швидкість читання, дата початку та закінчення читання.

Блок-схему алгоритму для екрану відслідковування читання книги продемонстровано на рисунку 2.5.


Рисунок 2.5 – Блок-схема алгоритму відслідковування читання книги

Отже, метод та алгоритм процесу відслідковування читання книги є ключовим етапом для розуміння та аналізу читацьких звичок. Цей процес включає в себе запис часу, витраченого на кожну сесію читання. Використання цього методу та алгоритму допомагає читачам зрозуміти свої переваги, покращити ефективність читання та збагатити свій досвід літературного сприйняття.

### 2.6 Висновки

Отже, у другому розділі було визначено необхідні дані для роботи програмного застосунку. Також розроблено методи та алгоритми роботи мобільного застосунку для управління читацькою активністю, що визначають логіку роботи системи та способи вирішення поставлених задач, а саме: пошуку та зберігання книг та відслідковування читання книги та відображення. Крім того, було розроблено структуру системи, яка забезпечує зручну та ефективну взаємодію з мобільним застосунком.

# 3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

Для розробки мобільного застосунку управління читацькою активністю вибір технологій є ключовим. Важливо обирати ефективні, зручні та сумісні з платформами рішення, враховуючи потреби користувачів та безпеку даних.

Використання бібліотек та фреймворків є важливим елементом розробки програмного забезпечення. Ці інструменти надають готові компоненти, модулі та інші ресурси, які допомагають розробникам зменшити час, необхідний для створення програми, а також підвищити її якість та функціональність. Розглянемо особливості трьох кросплатформних мов програмування: Dart, Python та JavaScript.

Dart є об’єктно-орієнтованою мовою програмування, тобто вона зосереджена на створенні багаторазово використовуваних фрагментів коду, які називаються об’єктами, і підтримує об’єкти, успадкування та поліморфізм, дозволяючи розробникам структурувати свій код у модульний та організований спосіб [12]. Dart має великий набір вбудованих бібліотек і пакетів, які надають готові до використання функції для стандартних завдань.

Python – це високорівнева мова програмування, яку можна використовувати його для розробки програмного забезпечення, серверної сторони веброзробки, машинного навчання та математики [13]. Python підтримує кілька парадигм програмування, включаючи об'єктно-орієнтоване, процедурне та функціональне програмування. Асинхронне програмування в Python дозволяє ефективно обробляти операції введення-виведення, що є важливим для створення високонавантажених серверів і мережевих застосунків.

JavaScript –це високорівнева мова програмування, яка є однією з основних технологій веброзробки поряд з HTML і CSS [14]. JavaScript має динамічну типізацію і підтримує об’єктно-орієнтоване, імперативне і функціональне

програмування. Вона використовується не тільки для створення інтерактивних вебінтерфейсів, але й для серверної розробки завдяки платформі Node.js.

Всі три мови підтримують розробку кросплатформних застосунків. Dart із Flutter дозволяє створювати програми для iOS, Android та вебплатформ. Python можна використовувати для розробки програм, що працюють на різних операційних системах, за допомогою інструментів як Kivy або BeeWare. JavaScript є основною мовою для веброзробки, і з використанням технологій як React Native та Electron можна створювати мобільні та настільні застосунки.

Для порівняння обраних мов програмування складемо таблицю 3.1.

Таблиця 3.1 – Функціональні характеристики мов програмування

| Характеристика | Мова програмування | | |
|---|---|---|---|
| | Dart | Python | JavaScript |
| Статична типізація | + | +/- | - |
| Компіляція в машинний код | + | - | - |
| Можливість визначення перевантаження операторів | + | + | - |
| Множинне наслідування класів | +/- | + | - |
| Відповідальність системи за механізм виявлення помилок під час виконання | + | + | +/- |
| Сумарний коефіцієнт | 4,5 | 3,5 | 0,5 |

Розглянемо детальніше функціональні характеристики таблиці:

1. Статична типізація – це підхід до типізації в програмуванні, де типи даних визначаються на етапі компіляції програми і залишаються незмінними під час виконання коду. У Python статична типізація реалізована за допомогою анотацій типів, але вона не є обов’язковою і не перевіряється під час виконання програми.

2. Компіляція в машинний код є процесом перетворення вихідного коду програми у машинний код.

3. Можливість визначення перевантаження операторів – це можливість визначати, які дії повинні виконуватися під час використання стандартних операторів (+, -, *, /, тощо) зі своїми власними типами даних.

4. Множинне наслідування класів – це можливість класу успадковувати властивості та методи від більш ніж одного батьківського класу. Dart не підтримує множинне наслідування класів у традиційному розумінні – він використовує концепцію під назвою міксини. Міксини дозволяють додавати можливості класу без множинного наслідування. Клас може «змішувати» кілька міксинів для досягнення подібного результату.

5. Відповідальність системи за механізм виявлення помилок під час виконання – це забезпечення виявлення різних типів помилок, таких як синтаксичні помилки, помилки часу виконання, тощо, та надання механізмів для їх обробки або виведення повідомлень про помилки для користувача. У JavaScript механізм виявлення помилок також відбувається під час виконання програми, і помилки виводяться в консоль браузера. Однак через динамічну природу JavaScript, де типи даних визначаються автоматично, деякі помилки можуть виникати під час виконання програми і можуть бути важко виявити під час розробки.

Проаналізувавши дані таблиці, можна зробити висновки, що використання мови програмування Dart є найбільш оптимальним вибором для розробки. Dart можна використовувати для створення автономних програм або для розробки застосунків за допомогою фреймворків, таких як Flutter. Flutter-розробки відомі своєю високою продуктивністю і плавними анімаціями [15]. Flutter використовує графічний двигун під назвою Skia для відтворення інтерфейсу, що забезпечує стабільну продуктивність на різних пристроях.

Крім мови програмування для кросплатформного мобільного застосунку для управління читацькою активністю необхідні ще додаткові засоби розробки. Firebase Auth – це служба автентифікації користувачів, яку розробники можуть

інтегрувати у мобільні та вебзастосунки. Вона надає низку методів автентифікації, починаючи від власних постачальників, таких як звичайна електронна адреса та пароль, або соціальних постачальників, таких як Facebook, Twitter, LinkedIn, GitHub і Google [16].

Firestore – це хмарна документо-орієнтована база даних, яку надає Google в рамках платформи Firebase. Вона призначена для зберігання та синхронізації даних для розробки клієнтських і серверних застосунків, що дозволяє легку масштабованість та синхронізацію даних в реальному часі між клієнтськими застосунками та серверами [17].

Поєднання різних технологій у розробці мобільної системи дозволяє розширити можливості додатка та підвищити його продуктивність. Dart та Flutter мають зручний синтаксис та широку підтримку спільноти, що спрощує розробку та підтримку мобільних застосунків. Firebase Auth та Firestore легко інтегруються з Flutter, що дозволяє швидко розширювати функціональність застосунків з використанням хмарних сервісів.

Таким чином, використання Flutter та Dart разом з Firebase Auth та Firestore є ефективним вибором для швидкої розробки кросплатформних застосунків з використанням хмарних сервісів для автентифікації та збереження даних.

### 3.2 Розробка модуля управління читацькою активністю

Головною метою створення застосунку є управління читацькою активністю. Для того, щоб система знала, яку книжку обрав користувач для перегляду або читання, потрібно передати відповідний об’єкт. На рисунку 3.1 зображена сторінка книжки.

Після того, як система отримала потрібну книжку, відбувається створення полів для зберігання та редагування відповідної інформації. Аналогічно створенню полів для об’єкта в програмуванні можна порівняти з підготовкою простору для зберігання інформації. Це можна порівняти з підготовкою аркуша електронної таблиці, де створюються різні колонки для різних типів інформації. Надалі створені поля будуть змінюватися при натисканні різних кнопок.

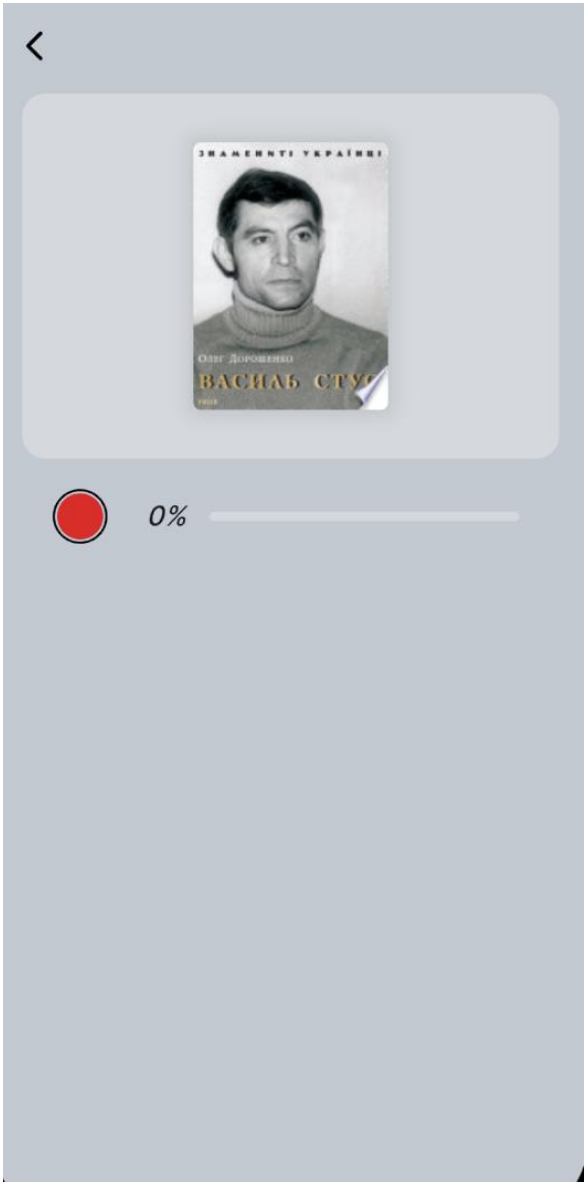



Рисунок 3.1 – Сторінка книжки

Для того, щоб користувач міг відстежувати свій прогрес та бачити швидкість читання, система використовує таймер. Цей таймер відслідковує, скільки часу користувач витрачає на читання книги, допомагаючи оцінити його читацький темп та відстежити прогрес в прочитанні книги. Таймер запускається, коли користувач починає читати книгу. При завершені читання таймер зупиняється і система зберігає загальний час читання для цього сеансу. Ці дані використовуються для визначення середньої швидкості читання. Для виконання цієї задачі створюються відповідні методи та підписки, які викликаються при зміні відліку часу (рисунок 3.2).

```
final timerSubscription = Stream.periodic(
  const Duration(seconds: 1),
  (value) {
    return value;
  },
).listen((event) { // Stream.periodic
  final currentValue = timerSeconds.value ?? 0;

  timerSeconds.changeValue(currentValue + 1, forceChange: true)
})
  ..pause();

addSubscriptions([
  timerSubscription,
  timerSeconds.extraDataStream.listen((timerStatus) {
    switch (timerStatus) {
      case TimerStatus.active:
        timerSubscription.resume();
      case TimerStatus.paused:
        timerSubscription.pause();
      case TimerStatus.disabled:
      // Do nothing
    }
  }),
]);
```

Рисунок 3.2 – Функціональна реалізація таймеру

Щосекунди таймер, якщо він ввімкнутий, збільшується на одиницю. Також є підписка на зміну стану таймера. Якщо новий статус таймеру «active», тоді він активується і починається відлік (рисунок 3.3), якщо статус «paused», тоді відлік часу зупиняється (рисунок 3.4), якщо статус «disabled», тоді система не робить нічого.

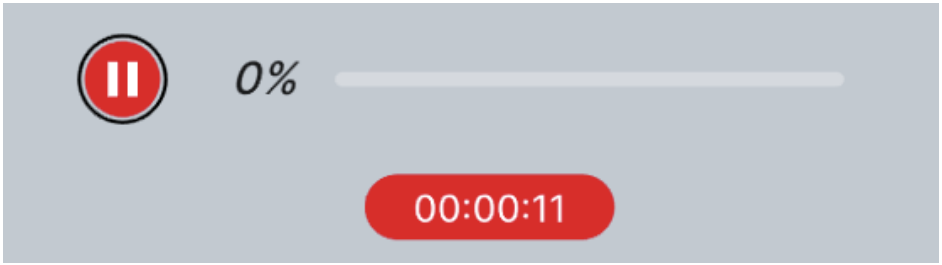



Рисунок 3.3 – Активний таймер

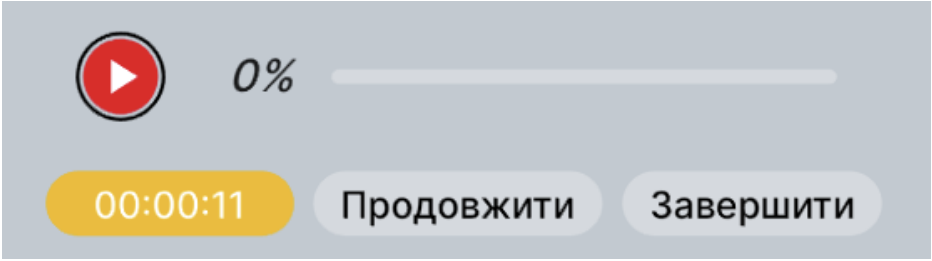



Рисунок 3.4 – Зупинений таймер

Після того, як користувач завершив читання, його прогрес потрібно зберігати для подальшого використання, що може бути зроблено шляхом збереження часу, який користувач витратив на читання та номера сторінки на якій він зупинився. Зберігання прогресу дозволяє користувачеві повернутися до книги пізніше та продовжити з того місця, де він зупинився. Таким чином, система може забезпечити зручність та продуктивність для користувача, дозволяючи йому легко відновити своє читання.

На рисунку 3.5 зображено збережений прогрес читання.

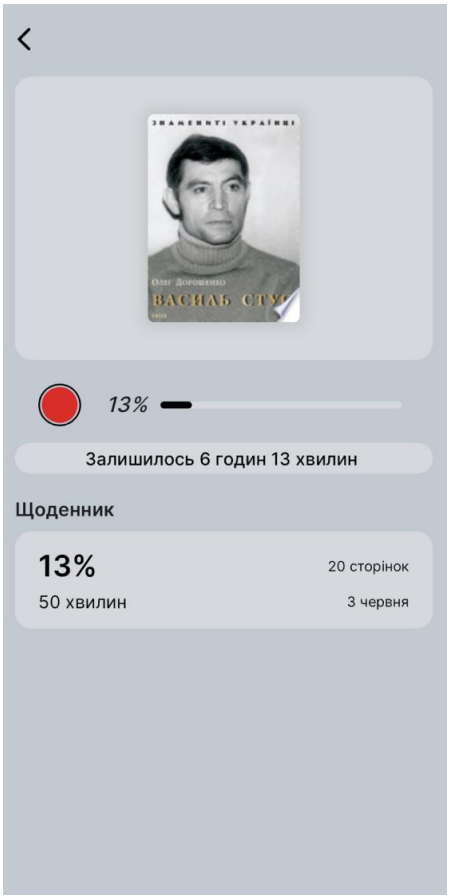



Рисунок 3.5 – Збереження прогресу читання

Таким чином, для забезпечення персоналізованого досвіду користувачів було розроблено модуль для управління читацькою активністю, який дозволяє системі ефективно відстежувати вибрані книги користувачами, їх читацькі звички та вподобання. Цей модуль надає можливість персоналізувати рекомендації, оптимізувати вміст та покращувати загальний досвід використання застосунку.

### 3.3 Розробка сервісу для пошуку книжок

Основною метою модуля для пошуку книжок є надання користувачеві можливості знаходити книги, що відповідають його запиту у полі пошуку. Коли користувач вводить запит, система аналізує його і видає результати, які найбільш відповідають введеному тексту.

На рисунку 3.6 наведено сторінку з пошуком.

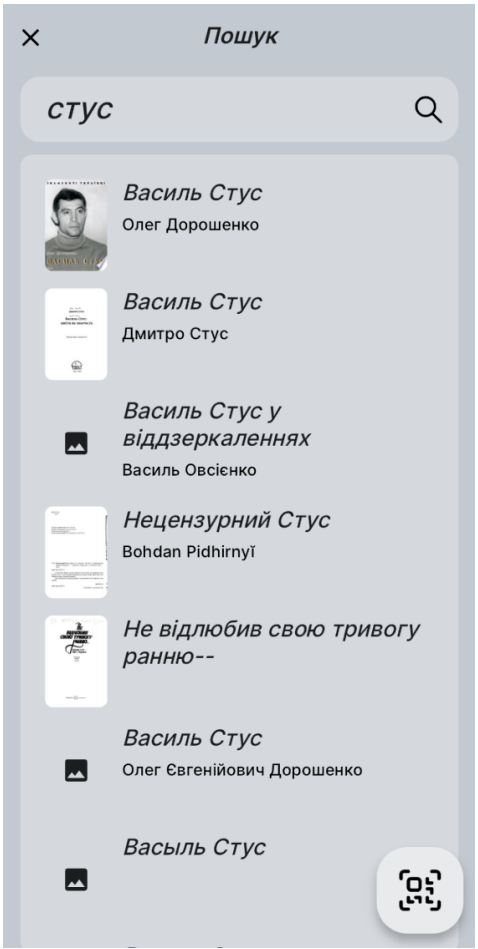



Рисунок 3.6 – Сторінка пошуку

Для того, щоб не зберігати величезну базу книжок, яку потрібно часто оновлювати, оскільки з’являються нові або перевидаються в новому виданні, при розробці даного модуля було реалізовано сервіс для пошуку використовуючи Google Books API [18], що надає вже готову базу з книжками. Сервіс отримує рядок запиту в якості параметру, що містить назву книжки для пошуку. Після цього робиться запит на Google Books API та обробляється результат.

Для уникнення повторного відображення книжок на сторінці пошуку, система використовує фільтрацію списку за ключовим ідентифікатором (ID). Коли користувач виконує пошук, система відображає результати, виключаючи будь-які книжки, які вже були показані раніше. Це дозволяє уникнути дублювання та забезпечує користувачам актуальні результати пошуку. На рисунку 3.7 зображена реалізація фільтрації.

```
@override
SearchState onStateChanged(DataChangeReason reason, SearchState state) {
  final visibleData = state.data.unique(id: (book) => book.id);

  return state.copyWith(visibleData: visibleData);
}

@override
bool equals(BookSearchModel item1, BookSearchModel item2) {
  return item1.id == item2.id;
}
```

Рисунок 3.7 – Фільтрація книжок за ключовим ідентифікатором

Після того, як користувач знайшов потрібну книжку, система надає йому можливість додати її в свою колекцію для подальшого швидкого доступу та уникнення повторного пошуку. Після додавання книги в колекцію вона стає доступною для користувача у розділі «Книжкова полиця».

Отже, модуль для пошуку книжок дозволяє знаходити книги за запитом користувача за допомогою полів пошуку. Користувач може ввести ключові слова, назву книги, автора або інші параметри, щоб отримати список відповідних результатів. Для цієї реалізації було використано Google Books API, що дозволяє

модулю отримувати доступ до широкого асортименту книжок із бази даних Google Books.

### 3.4 Розробка сервісу відслідковування читання у вигляді календаря

Відслідковування прогресу читання у вигляді календаря – це не тільки про взаємодію з користувачем. Така функція сприяє розвитку звичок читання, особистої відповідальності та загальної взаємодії з програмою.

Коли користувач бачить обкладинку книги в календарі, це дає візуальне уявлення про прогрес, даючи відчуття досягнення та заохочує продовжувати читати. Крім того, ця функція підтримує формування звичок читання. Календар допомагає встановити та підтримувати регулярний розпорядок читання, заохочуючи користувачів читати щодня. Можливість спостерігати за прогресом день за днем змушує продовжувати читати, щоб відповідати їхнім поставленим цілям і мотивує їх не відставати від розкладу читання.

Коли користувач заходить в застосунок, на головному екрані з’являється календар, який показує 7 днів тижня, який триває. Поточний день відображається червоним кольором. Якщо користувач нічого ще не читав, календар матиме порожні клітинки без книжок (рисунок 3.8).

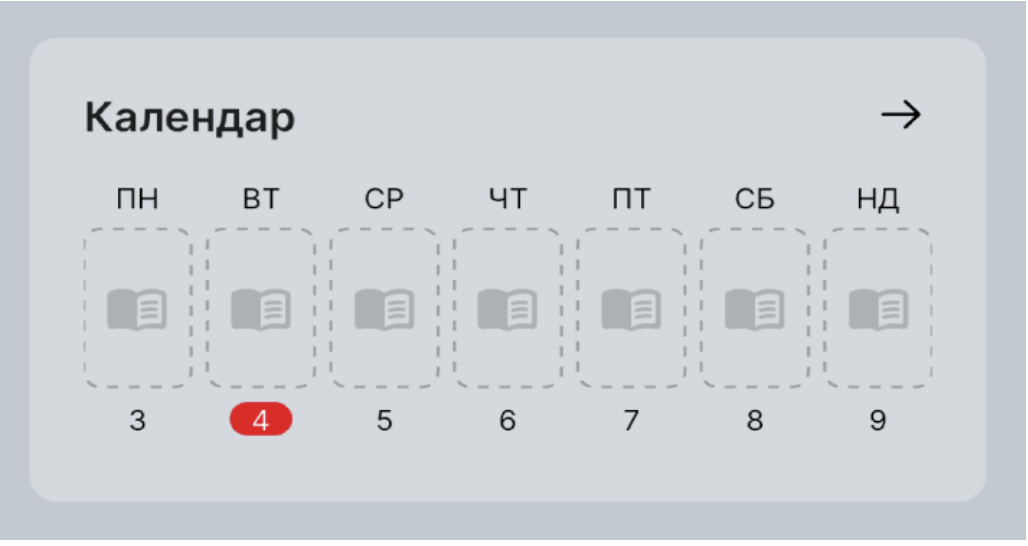



Рисунок 3.8 – Порожній календар на 7 днів

Дивлячись на такий календар користувач може побачити скільки він читав в поточному тижні. Для зручності та кращого відслідковування прогресу користувач може натиснути на стрілку, яка розгорне календар і збільшить видимість в поточний місяць (рисунок 3.9).

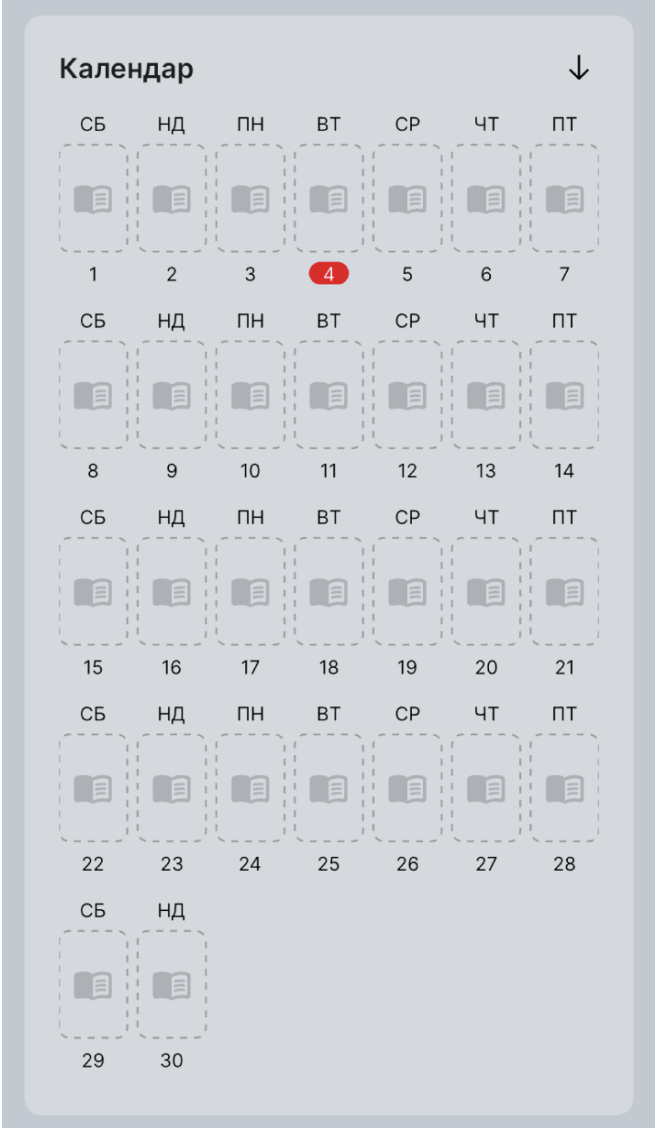



Рисунок 3.9 – Розгорнутий календар

Для того, щоб в календарі з’явилася книжка, користувач має почати читати. Для цього потрібно вибрати книжку, перейти на її сторінку та запустити таймер, а через деякий час, після завершення читання, книжка з’явиться в календарі.

На рисунку 3.10 зображено блок-схему алгоритму збереження запису в календар.

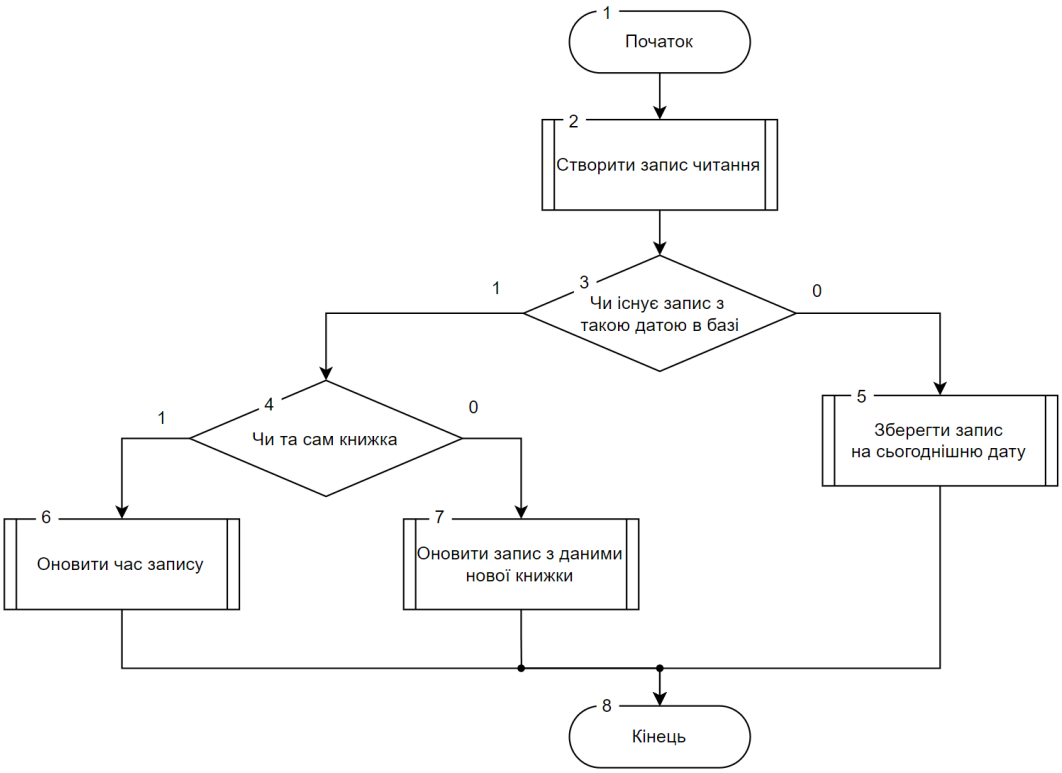



Рисунок 3.10 – Блок-схема алгоритму збереження запису в календар

Після завершення читання, користувач може повернутися на головну сторінку і побачити оновлений календар, який буде містити обкладинку книжки, яку він читав. На рисунку 3.11 зображено оновлений календар після читання.

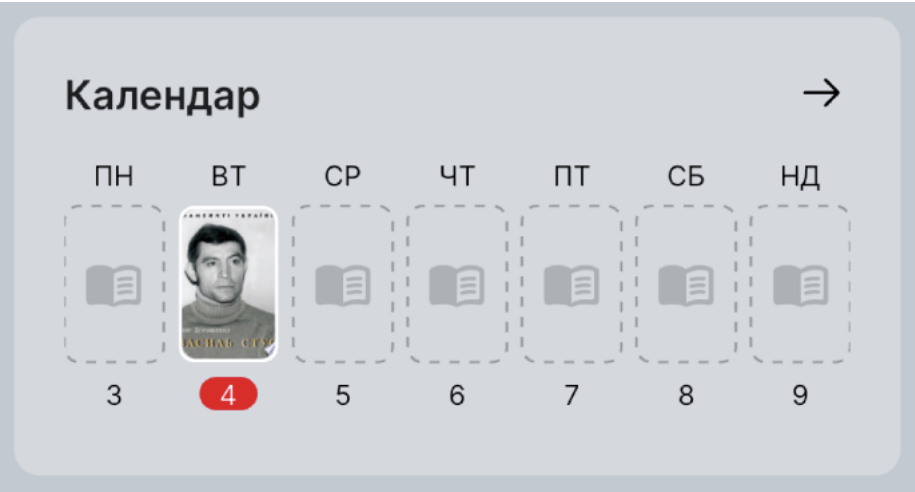



Рисунок 3.11 – Оновлений календар

Отже, сервіс відслідковування читання у вигляді календаря покращує застосунок для відстеження читання, забезпечуючи візуальну мотивацію та підтримує формування звичок, посилюючи залученість і полегшуючи встановлення цілей. Переваги такого модуля сприяють приємнішому та ефективнішому читанню, що робить застосунок цінним інструментом для читачів. Лістинг програми наведено в додатку В.

### 3.5 Висновки

У третьому розділі було обґрунтовано вибір засобів реалізації програмного продукту, а саме: Flutter та Dart, Firebase Auth та Firestore. Також наведено опис створення модуля управління читацькою активністю, розробки сервісу для пошуку книжок та сервісу відслідковування читання у вигляді календаря.

# 4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСТОСУНКУ

## 4.1 Аналіз методів тестування програмного забезпечення

Тестування, як завершальний етап розробки сайту, відіграє важливу роль у процесі створення якісного програмного забезпечення.

Фреймворк Flutter забезпечує надійну підтримку для автоматизованого тестування програми. Для повної перевірки програми використовують три типи автоматичного тестування [19]:

- модульне тестування;
- інтеграційне тестування;
- тестування віджетів.

Розглянемо детальніше кожний з них.

Модульне тестування (англ. Unit Testing) є методом тестування програмного забезпечення, де окремі модулі програми, такі як функції, методи класів або окремі файли, перевіряються на коректність роботи в ізольованому середовищі. Тобто кожен модуль тестується незалежно від інших частин програми, що дозволяє зосередитися на перевірці їхньої функціональності.

Крім того, модульні тести є важливою документацією для розробників, описуючи очікувану поведінку кожного модуля і надаючи швидкий засіб перевірки, чи відповідає він цій функціональності. Такий підхід сприяє покращенню якості програмного забезпечення та забезпечує більшу впевненість у його працездатності.

Після успішного проходження модульних тестів проводяться інтеграційні тести. Інтеграційне тестування (англ. Integration Testing) – це процес перевірки взаємодії між різними компонентами програмного забезпечення, наприклад, модулями, класами або сервісами. Під час цього виду тестування перевіряється, як добре ці компоненти працюють разом та чи відповідають вимогам та специфікаціям програми. Виконання інтеграційних тестів допомагає забезпечити, що весь додаток працює як одне ціле, а компоненти взаємодіють між собою коректно та ефективно.

Тестування віджетів (англ. Widget Testing) – це процес автоматизованої перевірки коректності та працездатності віджетів користувацького інтерфейсу. Тестування віджетів є ключовим етапом в розробці застосунків на платформі Flutter, оскільки віджети є основними будівельними блоками інтерфейсу. Автоматизовані тести віджетів дозволяють перевіряти правильність відображення кожного елемента інтерфейсу та його взаємодію з користувачем в різних сценаріях використання, що допомагає виявити та виправити можливі проблеми з відображенням, взаємодією та зручністю використання інтерфейсу ще до випуску застосунку в продуктивне середовище.

Усі результати тестування записуються у відповідну документацію та аналізуються розробниками для виявлення можливих проблем та вдосконалення якості програмного забезпечення. Звіт про тестування є важливим засобом комунікації, який допомагає всім зацікавленим сторонам, керівникам проектів та іншим членам команди зрозуміти результати тестування та прийняти обґрунтовані рішення стосовно готовності програмного забезпечення до випуску [20]. Документація також може містити описи тестових сценаріїв, виявлені помилки, відповідні виправлення та інші важливі дані, які допомагають у підтримці та подальшому розвитку програми.

## 4.2 Тестування мобільного застосунку управління читацькою активністю

Результати тестування продуктивності системи включають в себе оцінку швидкості відповіді системи на запити користувачів, завантаження ресурсів, а також моніторинг витрат ресурсів при різних умовах використання. Ці результати дозволяють визначити, чи відповідає продуктивність системи вимогам та чи необхідно проводити оптимізацію для забезпечення заданих рівнів продуктивності.

Результат тестування продуктивності наведено на рисунку 4.1.

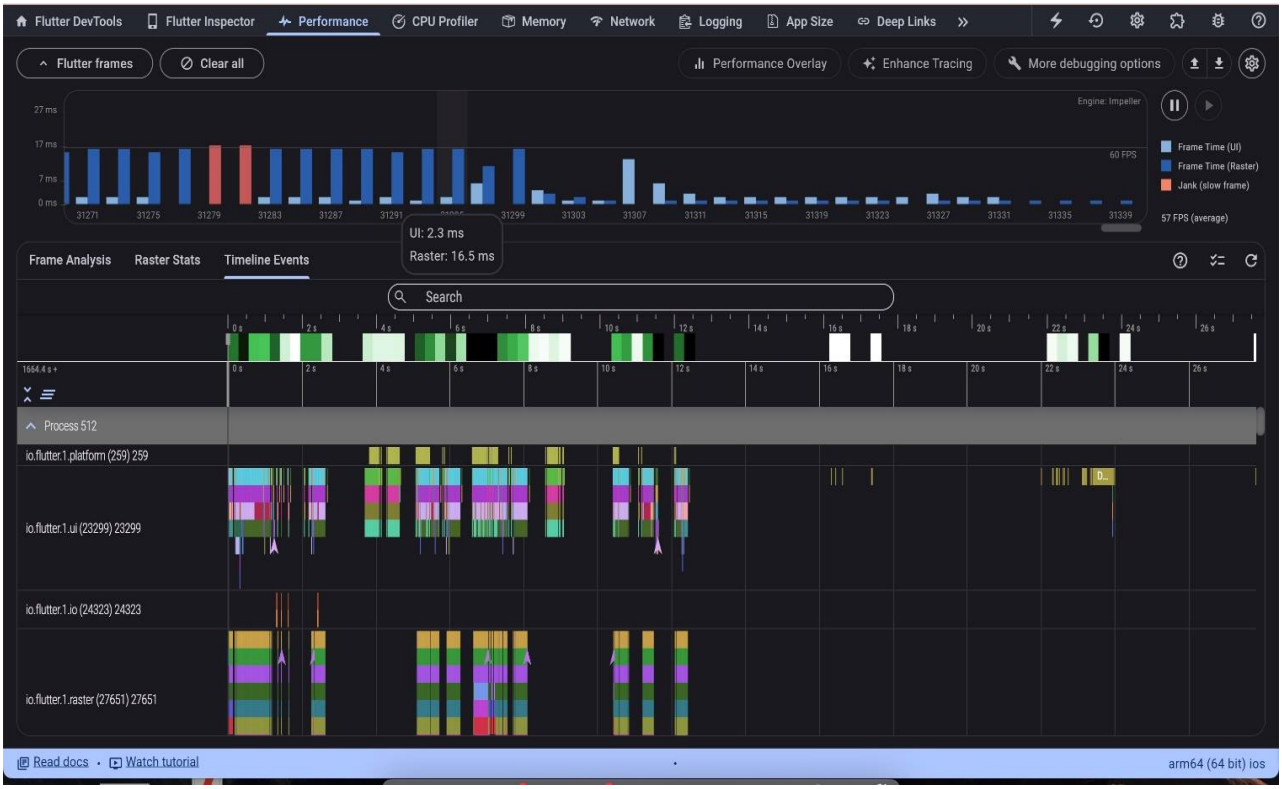



Рисунок 4.1 – Результати тестування продуктивності системи

Першою можливою помилкою при користуванні користувачем застосунком є перевірка його підключення до інтернету. У разі відсутності інтернет-з’єднання, система відображає повідомлення про помилку.

Результат тестування зображено на рисунку 4.2.

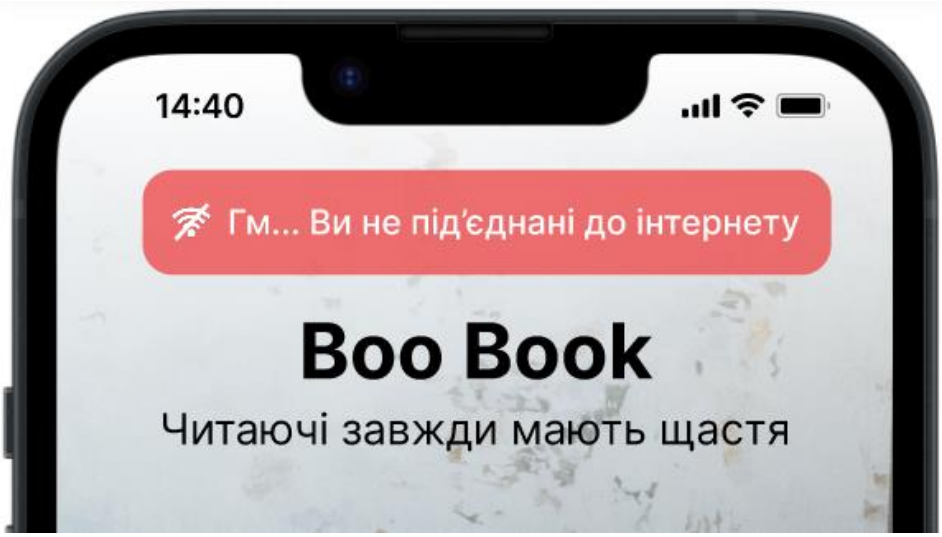



Рисунок 4.2 – Підключення користувача без інтернет-з’єднання

Після перевірки підключення користувача до Інтернету система виконує додаткові дії для забезпечення зручності та повідомлення про стан зв’язку, а саме система автоматично намагається відновити зв’язок, перевіряючи доступність мережі знову через певний інтервал часу.

Крім того, система зберігає локальні дані для можливого використання в офлайн-режимі, якщо доступ до Інтернету втрачено.

При пошуку книги можливий випадок помилки, якщо користувач не ввів необхідні дані, тобто здійснює пошук пустого поля. Оскільки система не отримує від користувача жодних конкретних критеріїв пошуку, вона не може визначити, які книги точно він шукає і показує повідомлення про помилку (рисунок 4.3).

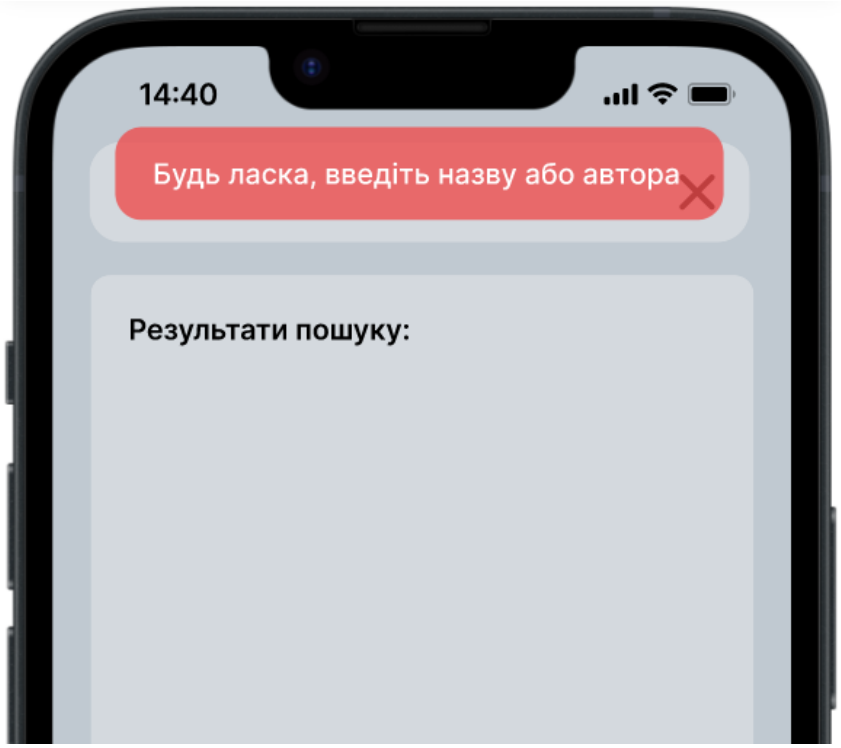



Рисунок 4.3 – Пусте поле пошуку книги

Іншою можливою помилкою при пошуку книги є та, коли книгу не було знайдено – або неправильно введені її назва чи автор, або ж цієї книги немає в базі даних. Тоді користувач отримує повідомлення про те, що книгу не було знайдено і користувачеві краще ввести іншу назву або автора (рисунок 4.4).

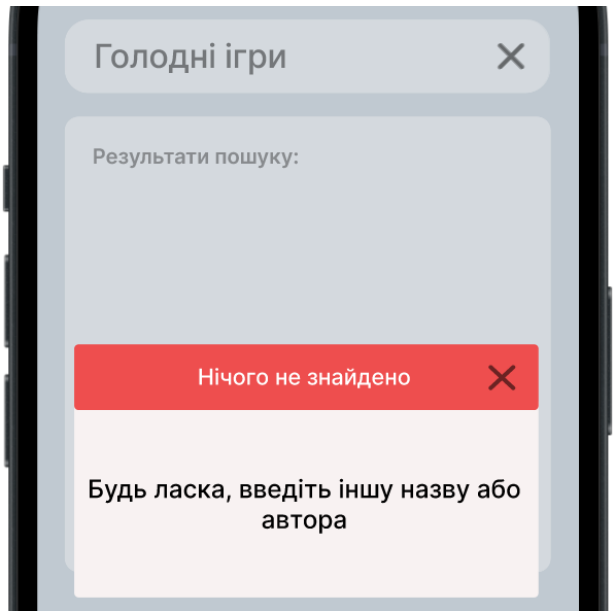



Рисунок 4.4 – Помилка пошуку книги

Після вибору книги та початку читання користувач вводить номер початкової сторінки, вмикаючи таймер, і по завершенню вводить номер кінцевої сторінки. Очевидно, що цей номер не може бути нулем, тому якщо користувач хоче зберегти дані про нулеву крайню сторінку, система відображає помилку про це. Повідомлення про помилку показано на рисунку 4.5.

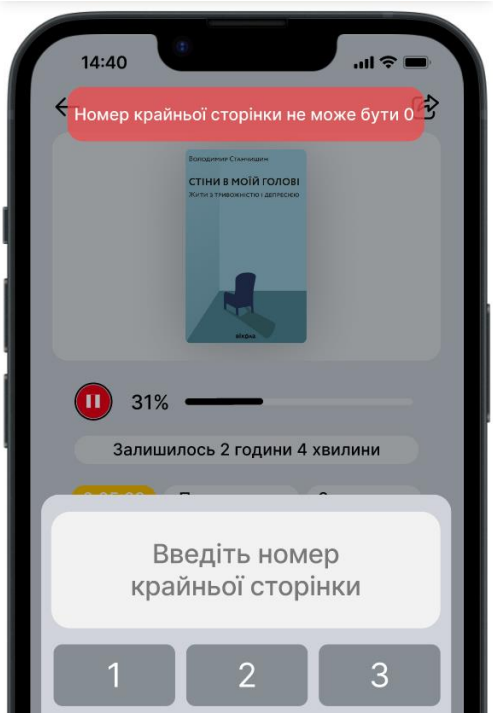



Рисунок 4.5 – Некоректне введення крайньої сторінки

Отже, тестування системи є важливим етапом для того, щоб переконатися в її надійності та коректному функціонуванні перед випуском в експлуатацію. Здійснене тестування включало проведення різних тестів, таких як модульне тестування, інтеграційне тестування та тестування віджетів, які дозволяли перевірити різні аспекти системи.

### 4.3 Розробка інструкції користувача

Інструкція користувача передбачає визначення основних кроків та процедур, які користувач повинен виконати для успішного використання програмного забезпечення.

При відкритті застосунку користувач бачить екран «Авторизація». Використовуючи цей екран користувач може увійти в свій акаунт за допомогою акаунтів Apple та Google (рисунок 4.6).

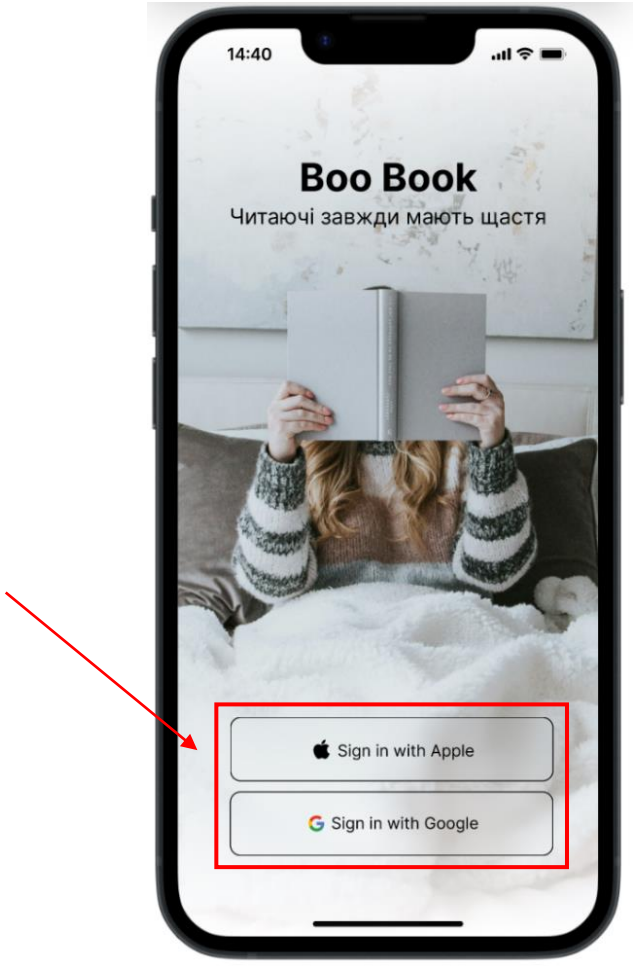


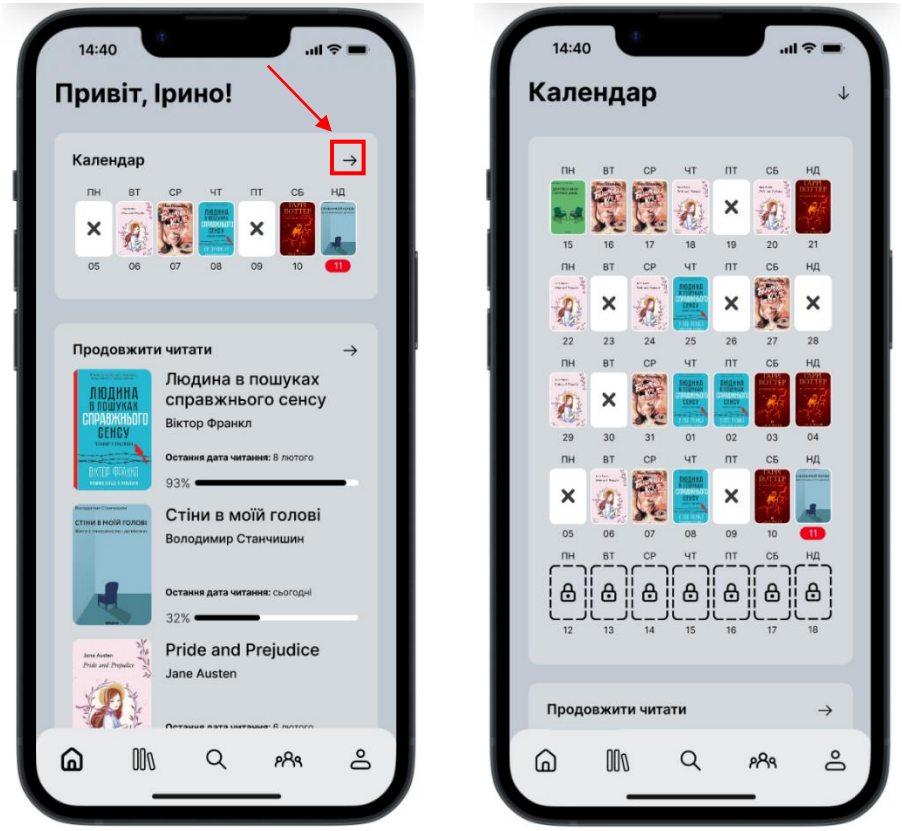

Рисунок 4.6 – Сторінка входу користувача

Після входу в застосунок користувач бачить екран «Головна» та доступні для нього функції (рисунок 4.7, а):

– календар, який показує обкладинки книг, які користувач читав певного дня;

– відображення книг, які зараз читає користувач.

Календар показує поточний тиждень та містить кнопку, при натисканні на яку він розгортається (рисунок 4.7, б). Користувач має швидкий доступ до книги, тобто він може натиснути на обкладинку книги, щоб відкрити її. Це дозволяє швидко розпочати читання книги, не шукаючи її в свій «бібліотеці». Також під обкладинкою книги зображено, який відсоток книги вже прочитано.




а) б)

Рисунок 4.7 – Головна сторінка застосунку (а) та розгорнутий календар (б)

На рисунку 4.8 наведено панель навігації застосунку, що містить 5 кнопок: «Головна», «Книжкова полиця», «Пошук», «Рекомендації» та «Профіль».

Рисунок 4.8 – Панель навігації

При першому вході в мобільний застосунок користувач ще не має переліку книг, тому інтерфейс даних є пустим, оскільки користувач ще не має збережених даних або відомостей про їхню активність на платформі. Зображення екрану наведено на рисунку 4.9.

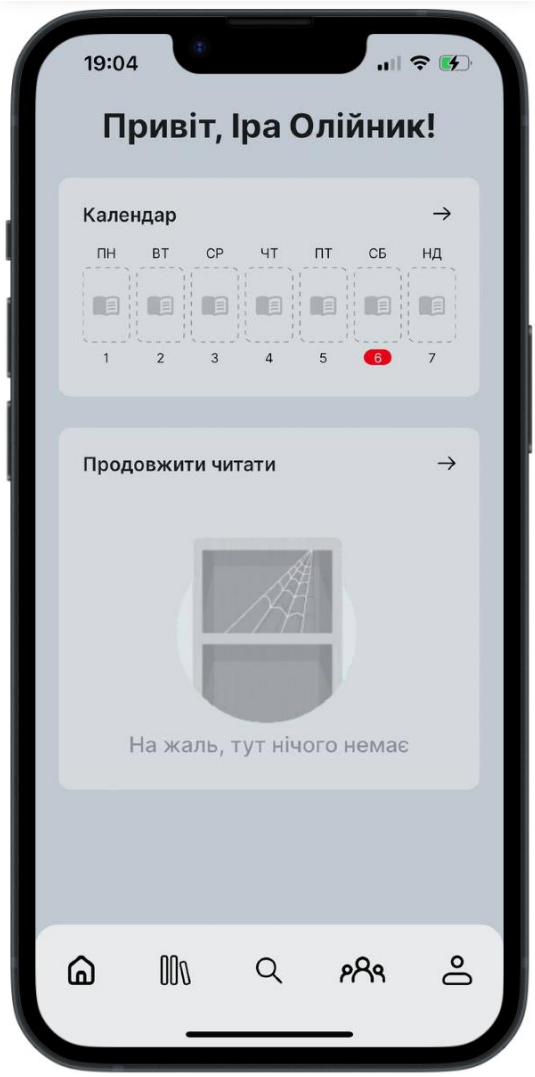



Рисунок 4.9 – Головний екран при першому вході користувача

Наступним розділом застосунку є екран «Книжкова полиця», що показує книги, які користувач добавив до своєї колекції. Користувач може бачити обкладинку, назву книги, автора та прогрес читання. Також користувач може виконувати різні дії зі своєю книжковою колекцією, такі як додавання нових книг, видалення або редагування інформації про наявні книги.

На рисунку 4.10 наведено графічний інтерфейс екрану «Книжкова полиця».

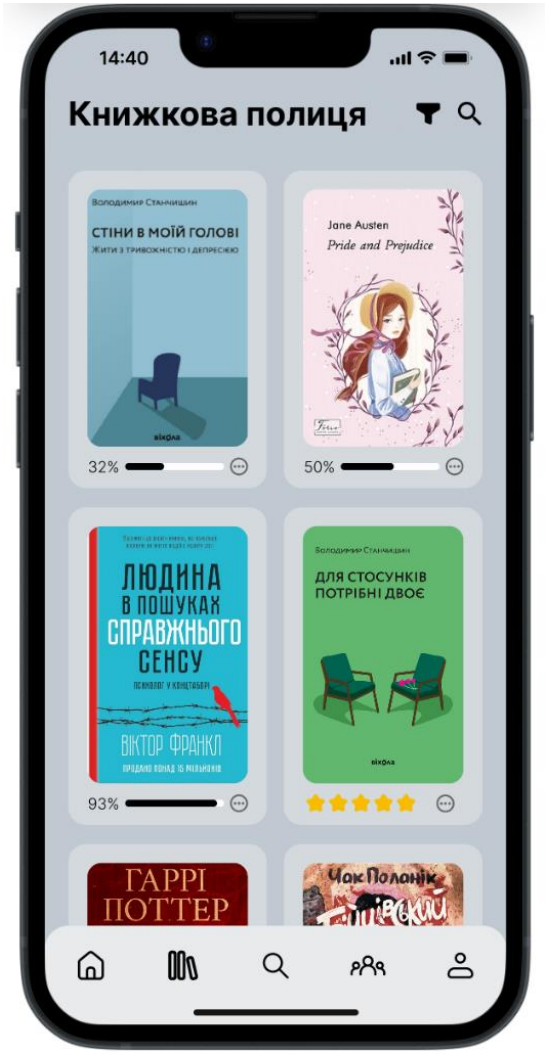
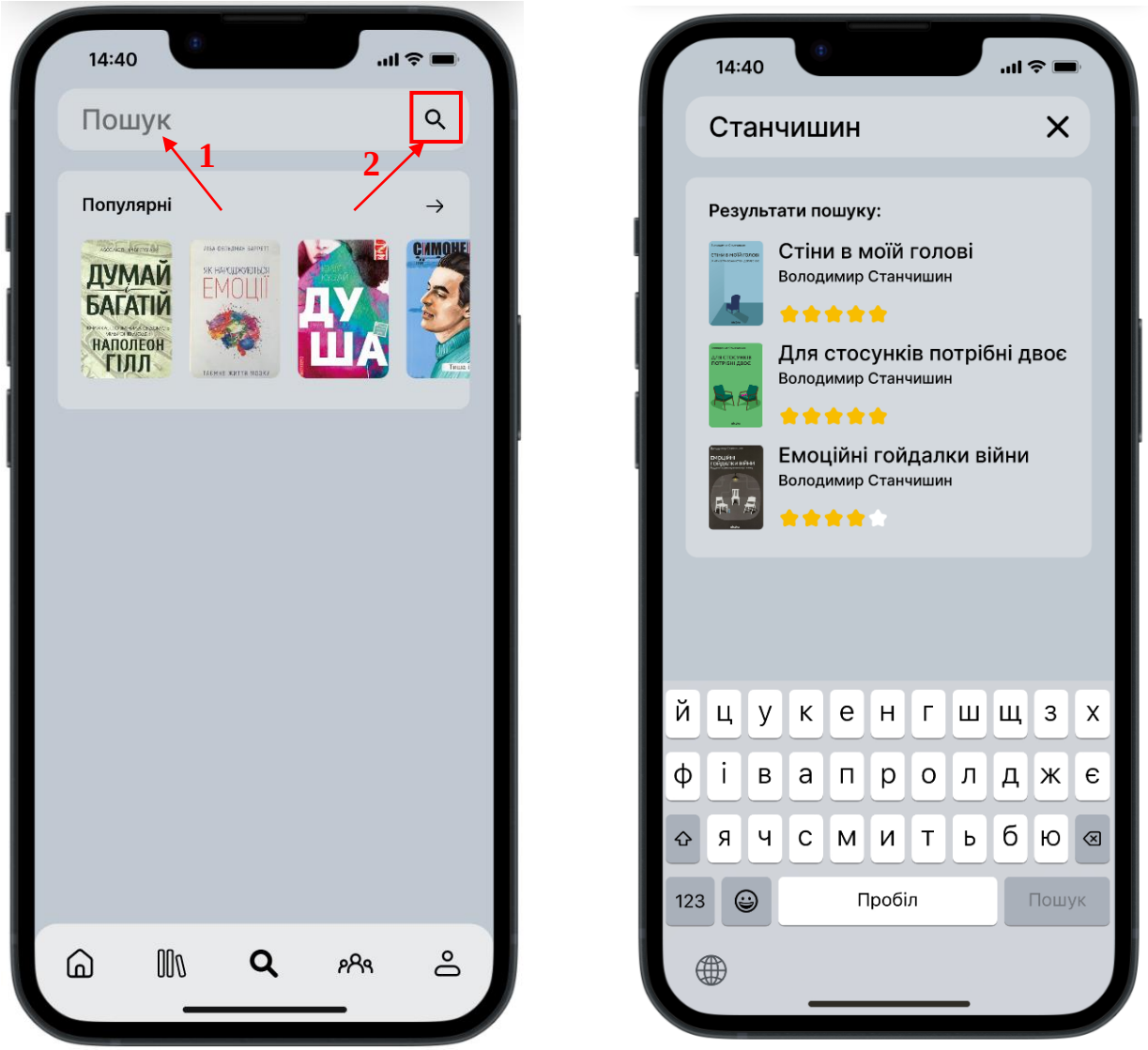



Рисунок 4.10 – Сторінка «Книжкова полиця»

Також користувач може шукати необхідні для нього книги у розділі «Пошук» (рисунок 4.11, а). Пошук є важливою функцією, що допомагає користувачам знаходити книги, які вони цікавляться, швидко і ефективно.

Пошук може здійснюватися як за назвою книги, так і за її автором. Після введення запиту користувач отримує список книг разом з їхніми назвами та авторами, а також можливість переглянути додаткову інформацію, таку як опис книги, обкладинка та рейтинг. Такий процес дозволяє користувачам швидко знайти необхідну інформацію про книги та оцінити їхню популярність, допомагаючи їм прийняти рішення про покупку або читання (рисунок 4.11, б).




а) б)

Рисунок 4.11 – Сторінка пошуку (а) та результат пошуку книги (б)

Використовуючи перелік книг на головній сторінці (див. рисунок 4.7), сторінці «Книжкова полиця» (див. рисунок 4.10), пошуку (див. рисунок 4.11, б)

або сторінці профіля користувача, користувач може обрати книгу і побачити повну інформацію про неї – обкладинку, назву, автора, анотацію та відгуки інших користувачів з їх оцінками.

Користувач може почати читати обрану книжку або додати її на свою «полицю». Інтерфейс сторінки інформації про книжку показано на рисунку 4.12.

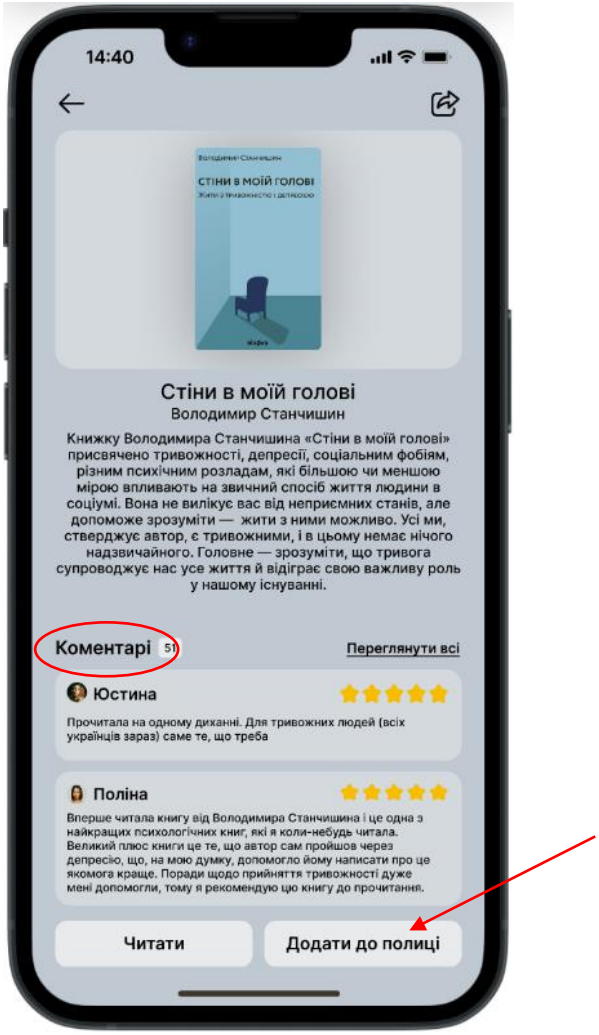



Рисунок 4.12 – Сторінка інформації про книжку

Після вибору книги для прочитання за допомогою червоної кнопки користувач вмикає таймер і вводить номер сторінки початку. Після завершення читання користувач знову натискає червону кнопку та вводить номер крайньої сторінки (рисунок 4.13).

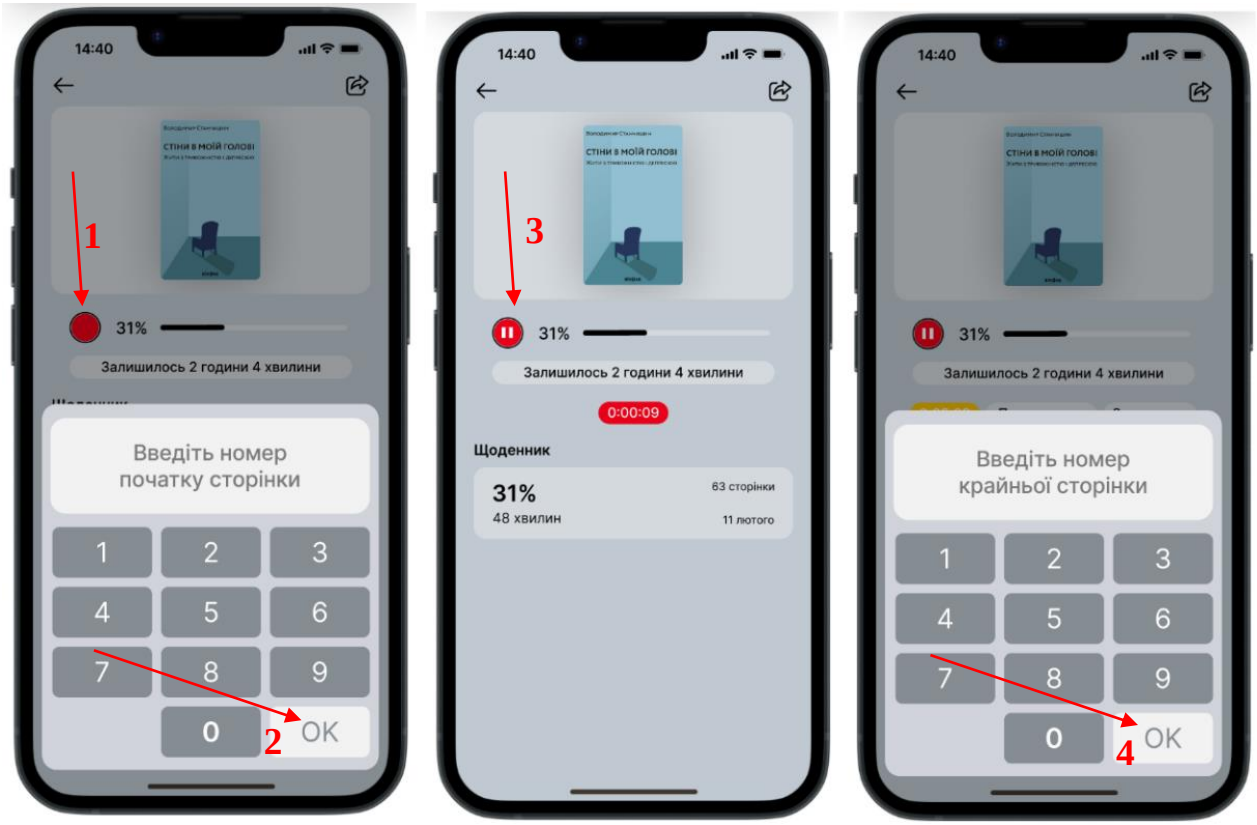


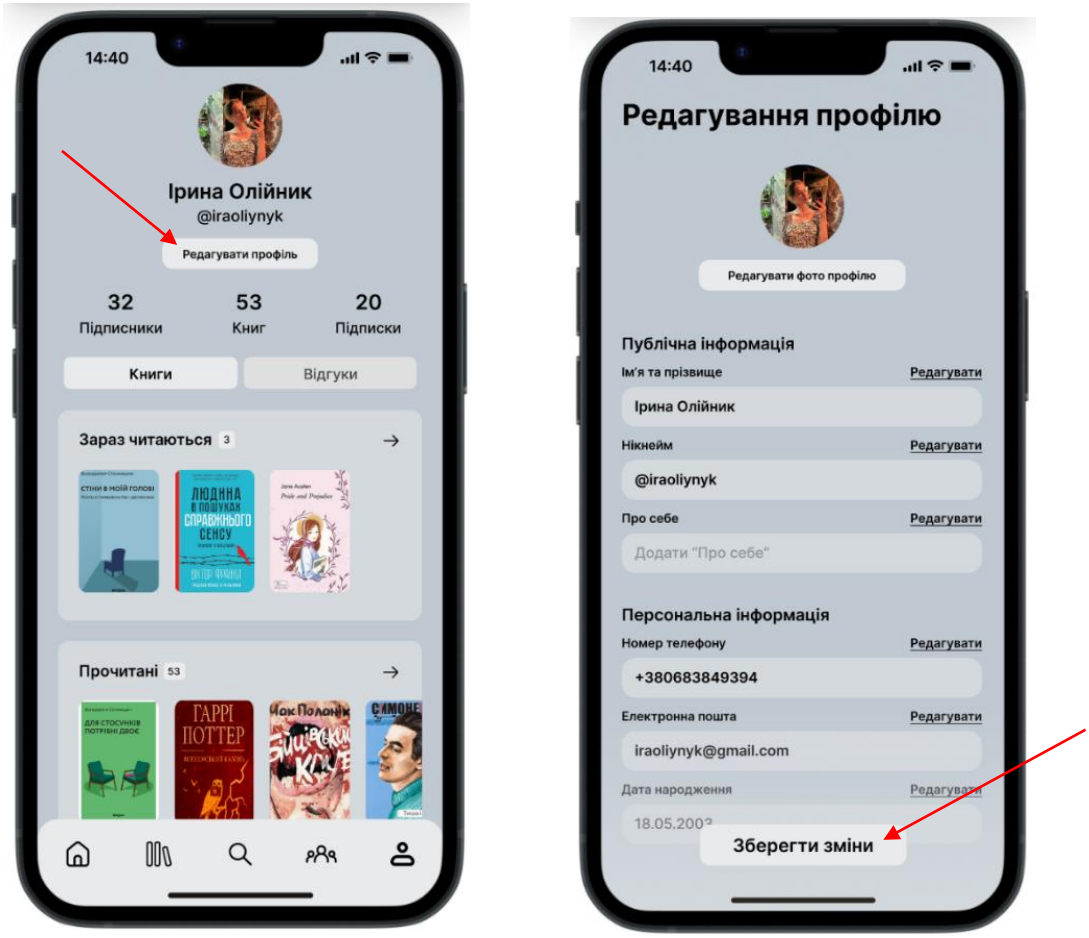

Рисунок 4.13 – Відслідковування читання

Профіль користувача відображає важливі розділи, що допомагають зрозуміти його інтереси та враження від читання. Він містить розділи «Книги» і «Відгуки».

У розділі «Книги» користувач може переглядати список прочитаних та книг, які він читає зараз (рисунок 4.14, а). «Відгуки» відображають враження користувача від прочитаних творів.

Крім того, профіль має кнопку «Редагувати профіль» (рисунок 4.14, б) для зміни інформації про користувача та іконки трьох крапок для додаткових параметрів.

При натисканні на кнопку «Редагувати профіль» користувач може змінити своє ім’я, нікнейм, фото профілю, інформацію «Про себе», номер телефону та інші персональні дані. Після внесення всіх змін користувачеві необхідно натиснути на кнопку «Зберегти зміни».




а) б)

Рисунок 4.14 – Профіль користувача (а) та редагування профілю (б)

Після того, як користувач завершує прочитання книги, він може бачити поля для оцінки книги та відгуку, а також статистику (рисунок 4.15), яка включає в себе:

– кількість сторінок книги;
– загальний час читання;
– середню швидкість читання;
– дату початку читання;
– дату закінчення читання.

Після написання відгуку користувачеві потрібно натиснути на кнопку «Зберегти», щоб опублікувати свій відгук до прочитаної книги.

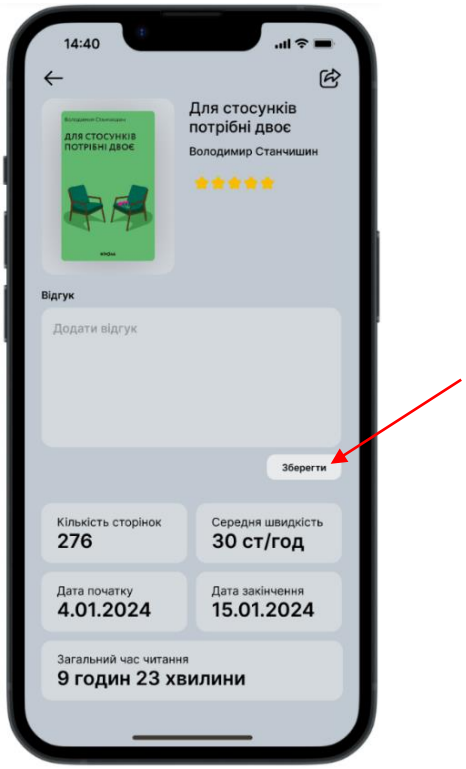



Рисунок 4.15 – Аналітика прочитаної книги

Після прочитання книги також користувач може бачити сторінку «Рекомендації», вхід на яку відбувається з головної сторінки (рисунок 4.16).

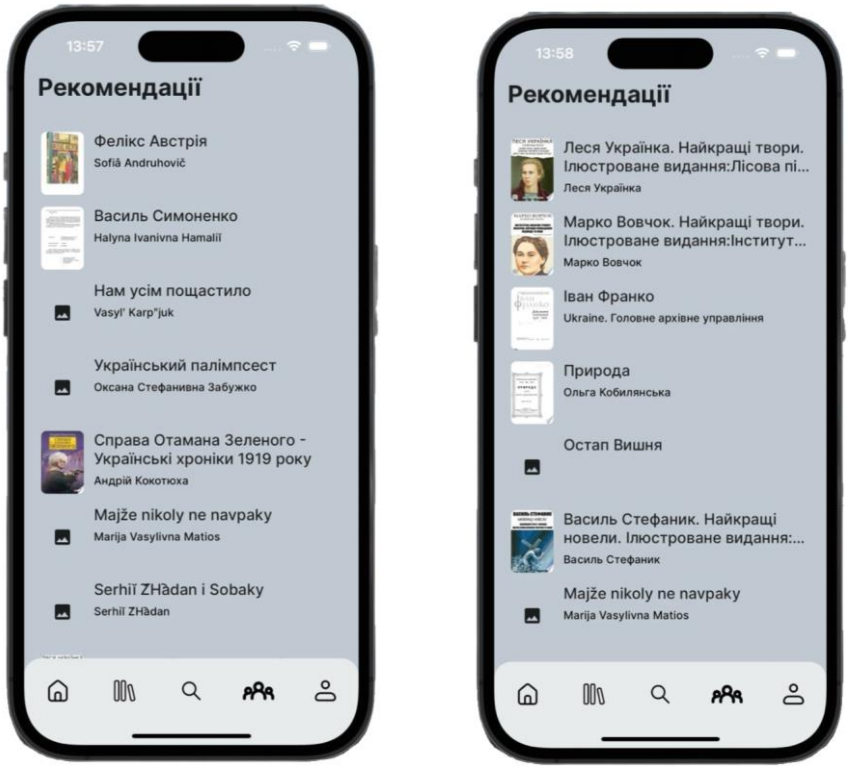



Рисунок 4.16 – Сторінка «Рекомендації»

На сторінці «Рекомендації» відображається список книг, які підбираються згідно з попереднім читацьким досвідом користувача. Підбір виконується за допомогою алгоритму рекомендацій на основі попередніх оцінок читання, а також аналізу його читацьких вподобань та інших параметрів, щоб запропонувати список книг, які можуть зацікавити його найбільше.

### 4.4 Системні вимоги

Системні вимоги – це набір технічних характеристик, які необхідні для встановлення та нормальної роботи програмного забезпечення на комп’ютері чи іншому пристрої.

Деталі щодо мінімальної та рекомендованої конфігурації наведено в таблиці 4.1.

Таблиця 4.1 – Мінімальні конфігурації

| | Мінімальна конфігурація | | Рекомендована конфігурація | |
|---|---|---|---|---|
| Операційна система | Android | IOS | Android | IOS |
| Тип процесора | 2-ядерний | Apple A8 | 4-ядерний або більше | Apple A12 Bionic або новіше |
| Об’єм оперативної пам’яті | 1 ГБ | 1 ГБ | 2 ГБ або більше | 2 ГБ або більше |
| Місце на жорсткому диску | 256 Мб | 256 Мб | 1 ГБ або більше | 1 ГБ або більше |
| Операційна система | Android 7.0 | iOS 12 | Остання версія Android | Остання версія iOS |

Після входу в систему користувача за допомогою Google або Apple аккаунту, передбачена перевірка етапів для забезпечення безпеки та правильності процесу: перевірка правильності введення облікових даних користувача, а також перевірка наявності активного інтернет-з’єднання для доступу до онлайн-сервісів. Використовуючи панель навігації застосунку він може перейти на розділи: «Книжкова полиця», «Пошук», «Рекомендації» та «Профіль». За допомогою пошуку користувач може знайти потрібну книгу, додати її в свою бібліотеку та читати, використовуючи функцію таймеру. Після прочитання книги користувач отримує аналітику прочитання та рекомендації для майбутнього вибору книг.

Отже, інструкція користувача допомагає користувачам уникнути помилок і неправильного використання програмного забезпечення, що може призвести до втрати даних або проблем з продуктивністю.

### 4.5 Висновки

У четвертому розділі було проведено тестування функціональних та графічних компонентів мобільного застосунку. Після проведення тестування було сформовано системні вимоги, що встановлюють мінімальні потрібні технічні характеристики для стабільної роботи програми, та описано інструкцію користувача, яка пояснює, як правильно використовувати застосунок для управління читацькою активністю.

# ВИСНОВКИ

У бакалаврській дипломній роботі було розроблено кросплатформний мобільний застосунок для управління читацькою активністю.

Бакалаврська дипломна робота оформлена згідно з методичними вказівками [21].

Було проаналізовано стан мобільних застосунків управління читацькою активністю, виконано порівняльний аналіз аналогів, поставлено задачі та розроблено модель системи.

Було створено структуру програмного застосунку та розроблено методи і алгоритми пошуку та збереження книг, а також для відслідковування читання книги. Розроблені методи та алгоритми надають покращені методи пошуку та збереження книг, що використовують Google Books API, усуваючи потребу в зберіганні книг на власному сервері. Модифіковані методи відслідковування читання, на відміну від існуючих, дозволяють обирати конкретні сторінки для початку і завершення читання, надаючи можливість ставити таймер на паузу.

Крім того, було проведено варіантний аналіз і обґрунтувано вибір засобів для реалізації програмного забезпечення. Під час аналізу технологій розробки було обґрунтовано вибір мови програмування Dart та фреймворку Flutter. Також було розглянуто їх переваги використання разом з функцією Firebase Auth та базою даних Firestore. Для розробки було використано середовище програмної розробки Visual Studio Code.

Розроблено сервіс для пошуку книжок, модуль управління читацькою активністю, сервіс відслідковування читання у вигляді календаря, який дозволяє користувачам візуально відслідковувати свої щоденні читацькі сесії та зручно переглядати їх.

Здійснено тестування розробленого кросплатформного мобільного застосунку. Тестування програми виявило, що всі основні функції працюють коректно і відповідають вимогам, встановленим у технічному завданні. Крім того, під час тестування були перевірені різні сценарії використання програми.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Вплив читання на стрес [Електронний ресурс] – Режим доступу до ресурсу: https://www.telegraph.co.uk/news/health/news/5070874/Reading-can-help-reduce-stress.html (дата звернення: 15.03.2024).

2. Популярність читання книг [Електронний ресурс] – Режим доступу до ресурсу: https://sens.in.ua/chomu-potribno-chytaty-knyhy/ (дата звернення: 15.03.2024).

3. How to keep track of books read [Електронний ресурс] – Режим доступу до ресурсу: https://cheqmark.io/blog/best-book-reading-trackers/ (дата звернення: 17.03.2024).

4. Олійник І.М. Переваги кросплатформної розробки над нативною / І.М. Олійник, О.В. Романюк // Матеріали LIII Науково-технічної конференції підрозділів Вінницького національного технічного університету, Вінниця, 2024. [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2024/paper/view/20154

5. Олійник І.М. Імплементація трекінг-процесу для управління читацькою активністю / І.М. Олійник, О.В. Романюк // Молодь в науці: дослідження, проблеми, перспективи, Вінниця, 2024. [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/mn/mn2024/paper/view/21691

6. Покращення досвіду читання користувача [Електронний ресурс] – Режим доступу до ресурсу: https://www.cottagenotebook.ie/6-reading-journal-apps-to-enhance-your-reading-experience/ (дата звернення: 18.03.2024).

7. Goodreads [Електронний ресурс] – Режим доступу до ресурсу: https://www.businessinsider.com/guides/tech/what-is-goodreads (дата звернення: 18.03.2024).

8. Особливості розробки «TBR - Bookshelf» [Електронний ресурс] – Режим доступу до ресурсу: https://www.cisin.com/growth-hacks/cost-and-feature-to-develop-software-like-tbr-bookshelf/ (дата звернення: 18.03.2024).

9. The Storygraph [Електронний ресурс] – Режим доступу до ресурсу: https://www.thestorygraph.com/ (дата звернення: 18.03.2024).

10. Larman C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis & Design and the Unified Process (3rd ed.). London : Pearson Education, 2021. 627 p.

11. «Комп'ютерна графіка» : електронний навч. посіб. [Електронний ресурс] / О. Н. Романюк, О. В. Романюк, Р. Ю. Чехместрук. – Вінниця :ВНТУ, 2023. – 147 с.

12. Deven Joshi. Building Cross-Platform Apps with Flutter and Dart . BPB Publications, Delhi 2023, 378 p.

13. Python [Електронний ресурс] – Режим доступу до ресурсу: https://www.freecodecamp.org/news/learn-python-book/ (дата звернення: 24.03.2024).

14. Federico Kereki. Mastering JavaScript Functional Programming. Birmingham : Packt Publishing; 3rd ed. Edition, 2023. 614 p.

15. Jonathan Sande. Dart Apprentice: Beyond the Basics. Dart Apprentice Series. New York City : Kodeco Inc, 2021, 256 p.

16. Firebase Auth [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@Adekola_Olawale/firebase-authentication-413626c5234d (дата звернення: 25.03.2024).

17. Sangapu, S., Panyam, D., Marston, J. The Definitive Guide to Modernizing Applications on Google Cloud (1st ed.). Birmingham : Packt Publishing, 2022. 488 p.

18. Google Books API [Електронний ресурс] – Режим доступу до ресурсу: https://developers.google.com/books (дата звернення: 27.03.2024).

19. Тестування на Flutter [Електронний ресурс] – Режим доступу до ресурсу: https://www.javatpoint.com/flutter-testing (дата звернення: 28.03.2024).

20. Тестова документація [Електронний ресурс] – Режим доступу до ресурсу: https://drukarnia.com.ua/articles/testova-dokumentaciya-G9sV1 (дата звернення: 28.03.2024).

21. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів студентів спеціальності 121 "Інженерія програмного забезпечення" / Уклад. О. Н. Романюк, Г. О. Черноволик – Вінниця: ВНТУ, 2022. – 51с.

# ДОДАТКИ

# ДОДАТОК А

## Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ

д.т.н., проф. О. Н. Романюк

"12" березня 2024 р.

**Технічне завдання**
**на бакалаврську дипломну роботу**
**«Розробка кросплатформного мобільного застосунку для управління читацькою активністю»**
**за спеціальністю 121 – Інженерія програмного забезпечення**

Керівник бакалаврської дипломної роботи:

к.т.н., доцент каф. ПЗ Романюк О.В.

"12" березня 2024 р.

Виконала:

студентка гр. 1ПІ-20б Олійник І.М.

"12" березня 2024 р.

Вінниця – 2024 року

**1. Найменування та галузь застосування.**

Бакалаврська дипломна робота: «Розробка кросплатформного мобільного застосунку для управління читацькою активністю».

Галузь застосування – мобільні технології.

**2. Підстава для розробки.**

Підставою для виконання бакалаврської дипломної роботи (БДР) є індивідуальне завдання на БДР та наказ №80 від «11» березня 2024 р. ректора по ВНТУ про закріплення тем БДР.

**3. Мета та призначення розробки.**

Метою роботи є підвищення якості моніторингу та управління читацькою активністю за рахунок розробки та використання спеціалізованої мобільної системи, яка дозволяє відслідковувати прогрес читання та аналізувати читацькі звички.

Призначення роботи – розробка та реалізація кросплатформного мобільного застосунку для управління читацькою активністю

**4. Вихідні дані для проведення НДР.**

Перелік основних літературних джерел, на основі яких буде виконуватись БДР.

1. Популярність читання книг [Електронний ресурс] – Режим доступу до ресурсу: https://sens.in.ua/chomu-potribno-chytaty-knyhy/

2. Deven Joshi. Building Cross-Platform Apps with Flutter and Dart . BPB Publications, Delhi 2023, 378 p.

3. Sangapu, S., Panyam, D., Marston, J. The Definitive Guide to Modernizing Applications on Google Cloud (1st ed.). Birmingham : Packt Publishing, 2022. 488 p.

**5. Технічні вимоги**

Тип програми – кросплатформний мобільний застосунок; модель розробки

– ітеративна; засоби організації даних програми – фреймворки Flutter, Firebase Auth та Firestore; вхідні дані: вподобані користувачем книжки; вихідні дані: аналітика читацької активності; середовище розробки – Visual Studio Code; мова програмування – Dart; програмне забезпечення для симуляції мобільних пристроїв – Xcode.

**6. Конструктивні вимоги.**

Графічна та текстова документація повинна відповідати діючим стандартам України.

**7. Перелік технічної документації, що пред'являється по закінченню робіт:**

– пояснювальна записка до БДР;
– технічне завдання;
– лістинги програми.

**8. Вимоги до рівня уніфікації та стандартизації.**

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

**9. Стадії та етапи розробки:**

| Ч.ч. | Назва етапів бакалаврської дипломної роботи | Строк виконання етапів роботи |
|---|---|---|
| 1 | Аналіз стану питання та постановка задач дослідження | 14.03.24 – 25.03.2024 |
| 2 | Розробка структури та моделі застосунку | 26.03.2024 – 02.04.2024 |
| 3 | Розробка методів та алгоритмів пошуку і зберігання книг з використанням Google Books API та відслідковування читання книг | 03.04.2024 – 20.04.2024 |
| 4 | Розробка алгоритму збереження запису в календар | 21.04.2024 – 28.04.2024 |
| 5 | Аналіз та вибір засобів для реалізації програмного продукту | 29.04.2024 – 02.05.2024 |
| 6 | Програмна реалізація | 03.05.2024 – 16.05.2024 |
| 7 | Тестування програмного застосунку | 17.05.2024 – 23.05.2024 |
| 8 | Оформлення матеріалів до захисту БДР | 24.05.2024 – 30.05.24 |

**10. Порядок контролю та прийняття.**

Виконання етапів бакалаврської дипломної роботи контролюється керівником згідно з графіком виконання роботи. Прийняття бакалаврської дипломної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

# ДОДАТОК Б

## Протокол перевірки бакалаврської дипломної роботи на наявність текстових запозичень

ПРОТОКОЛ ПЕРЕВІРКИ БАКАЛАВРСЬКОЇ ДИПЛОМНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: «Розробка кросплатформного мобільного застосунку для управління читацькою активністю»

Тип роботи: БДР

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник: Романюк О.В.

| | |
|---|---|
| Оригінальність | 96% |
| Схожість | 4% |

**Аналіз звіту подібності**

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи Олійник І.М.

Керівник роботи Романюк О.В.

# ДОДАТОК В

## Лістинг програми

```
import 'package:flutter/material.dart';

import 'package:boo_book/localization/index.dart';
import 'app.dart';
import 'app_initialization.dart';

void main() async {
  await initializeApp();

  runApp(
    EasyLocalization(
      path: CodegenLoader.path,
      supportedLocales: CodegenLoader.supportedLocales,
      fallbackLocale: CodegenLoader.supportedLocales.last,
      assetLoader: const CodegenLoader(),
      child: BooBook(),
    ),
  );
}

import 'package:flutter/material.dart';

import 'package:flutter_bloc/flutter_bloc.dart';

import 'package:boo_book/blocs/index.dart';
import 'package:boo_book/services/index.dart';

class AppStateWrapper extends StatelessWidget {
  const AppStateWrapper({
    super.key,
    required this.child,
  });

  final Widget child;

  @override
  Widget build(BuildContext context) {
    return MultiBlocProvider(
      providers: [
        BlocProvider(
          create: (context) => getIt<AuthBloc>(),
        ),
      ],
      child: child,
```

```
    );
  }
}

import 'dart:ui';

import 'package:boo_book/core/index.dart';
import 'package:flutter/material.dart';

import 'package:easy_localization/easy_localization.dart';
import 'package:firebase_core/firebase_core.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:flutter_dotenv/flutter_dotenv.dart';
import 'package:flutter_native_splash/flutter_native_splash.dart';

import 'package:boo_book/blocs/observer.dart';
import 'package:boo_book/services/index.dart';
import 'package:hive/hive.dart';
import 'package:path_provider/path_provider.dart';

Future<void> initializeApp() async {
  final widgetsBinding = WidgetsFlutterBinding.ensureInitialized();

  initializeSplashScreen(widgetsBinding);
  await initializeDotenv();
  await initializeFirebase();
  await Future.wait([
    initializeLogger(),
    initializeLocalization(),
    initHive(),
  ]);
  initializeCrashlytics();
  initializeBlocObserver();
  configureAuthDependencies();
}

void initializeSplashScreen(WidgetsBinding widgetsBinding) {
  FlutterNativeSplash.preserve(widgetsBinding: widgetsBinding);
}

Future<void> initializeFirebase() {
  return Firebase.initializeApp();
}

Future<void> initHive() async {
  final applicationPath = (await getApplicationDocumentsDirectory()).path;

  Hive.init(applicationPath);
```

```
  await Hive.openBox(EnvConstants.recommendationsBox);
}

Future<void> initializeLogger() {
  return LoggerService.instance.init();
}

Future<void> initializeLocalization() {
  return EasyLocalization.ensureInitialized();
}

Future<void> initializeDotenv() {
  return dotenv.load();
}

void initializeCrashlytics() {
  FlutterError.onError = (errorDetails) {
    LoggerService.instance.logError(errorDetails.exception, errorDetails.stack);
  };

  PlatformDispatcher.instance.onError = (error, stack) {
    LoggerService.instance.logError(error, stack);

    return true;
  };
}

void initializeBlocObserver() {
  Bloc.observer = SimpleBlocObserver(LoggerService.instance);
}

import 'package:flutter/material.dart';

import 'package:auto_route/auto_route.dart';
import 'package:flutter_bloc/flutter_bloc.dart';

import 'package:boo_book/blocs/auth/auth_bloc.dart';
import 'package:boo_book/core/index.dart';
import 'package:boo_book/styles/index.dart';
import 'widgets/index.dart';

@RoutePage()
class LoginScreen extends StatelessWidget {
  const LoginScreen({super.key});

  @override
  Widget build(BuildContext context) {
    final authBloc = context.read<AuthBloc>();
    final screenHeight = context.sizeOf.height;
```

```
    return LoginScaffold(
      shadowHeight: screenHeight * 0.45,
      body: SafeArea(
        child: Padding(
          padding: EdgeInsets.symmetric(
            vertical: screenHeight * 0.08,
            horizontal: 42,
          ),
          child: Column(
            children: [
              const Text(
                'Boo Book',
                style: TextStyle(
                  fontSize: 36,
                  fontWeight: FontWeight.bold,
                  color: Colors.black,
                ),
              ),
              const SizedBox(
                height: 3,
              ),
              const Text(
                'Читаючі завжди мають щастя',
                style: TextStyle(
                  fontSize: 20,
                ),
              ),
              const Spacer(),
              LoginButton(
                icon: Assets.icons.apple.svg(),
                text: 'Вхід з Apple',
                onTap: () {},
              ),
              const SizedBox(height: 10),
              LoginButton(
                icon: Assets.icons.google.svg(),
                text: 'Вхід з Google',
                onTap: authBloc.signInWithGoogle,
              ),
            ],
          ),
        ),
      ),
    );
  }
}

import 'dart:async';
```

```
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:freezed_annotation/freezed_annotation.dart';
import 'package:injectable/injectable.dart';

import 'package:boo_book/models/index.dart';
import 'package:boo_book/repositories/index.dart';

part 'auth_bloc.freezed.dart';
part 'auth_event.dart';
part 'auth_state.dart';

enum AuthenticationStatus {
  initial,
  unauthenticated,
  authenticated,
}

@Singleton(scope: 'auth')
class AuthBloc extends Bloc<_AuthEvent, AuthState> {
  final AuthRepository authRepository;
  final UserRepository userRepository;

  late StreamSubscription<User?> _subscription;

  AuthBloc({
    required this.authRepository,
    required this.userRepository,
  }) : super(const AuthState()) {
    _subscription = authRepository.authStateChanges.listen((user) {
      add(_AuthenticationStatusChanged(user));
    });

    on<_AuthenticationStatusChanged>(_authenticationStatusChanged);

    on<_SignInWithGoogleEvent>(_onSignInWithGoogle);
    on<_SignInWithAppleEvent>(_onSignInWithApple);

    on<_SignOut>(_signOut);
  }

  void signInWithGoogle() => add(_SignInWithGoogleEvent());
  void signInWithApple() => add(_SignInWithAppleEvent());

  FutureOr<void> _authenticationStatusChanged(
    _AuthenticationStatusChanged event,
    Emitter<AuthState> emit,
  ) async {
```

```
  final firebaseUser = event.user;

  if (firebaseUser == null) {
    return emit(AuthState.unauthenticated());
  }

  var userProfile = await userRepository.getUserProfile(firebaseUser.uid);

  if (userProfile == null) {
    userProfile = UserProfile.fromFirebaseUser(firebaseUser);

    await userRepository.createUserProfile(userProfile);
  }

  emit(
    AuthState.authenticated(userProfile),
  );
}

FutureOr<void> _onSignInWithGoogle(
  _SignInWithGoogleEvent event,
  Emitter<AuthState> emit,
) async {
  // TODO(Ira): Add error handling
  try {
    await authRepository.signInWithGoogle();
  } catch (e, stackTrace) {
    addError(e, stackTrace);

    emit(AuthState.unauthenticated());
  }
}

FutureOr<void> _onSignInWithApple(
  _SignInWithAppleEvent event,
  Emitter<AuthState> emit,
) async {
  // TODO(Ira): Add login with Apple
}

FutureOr<void> _signOut(_SignOut event, Emitter<AuthState> emit) {
  return authRepository.signOut();
}

@override
Future<void> close() async {
  await _subscription.cancel();
  return super.close();
}
```

```
}

import 'package:firebase_auth/firebase_auth.dart';
import 'package:google_sign_in/google_sign_in.dart';
import 'package:injectable/injectable.dart';

@Injectable(scope: 'auth')
class AuthRepository {
  final _client = FirebaseAuth.instance;

  Stream<User?> get authStateChanges => _client.authStateChanges();

  Future<UserCredential?> signInWithGoogle() async {
    final googleUser = await GoogleSignIn().signIn();

    final googleAuth = await googleUser?.authentication;

    final credentials = GoogleAuthProvider.credential(
      accessToken: googleAuth?.accessToken,
      idToken: googleAuth?.idToken,
    );

    return _client.signInWithCredential(credentials);
  }

  Future<void> signOut() {
    return _client.signOut();
  }
}

import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:injectable/injectable.dart';

import 'package:boo_book/models/books/models.dart';
import 'package:boo_book/repositories/mixin/firestore_mixin_repo.dart';

@injectable
class BooksRepository extends FirestoreMixinRepo<UserBookModel> {
  BooksRepository({
    @Named('userUid') required super.userUid,
  });

  @override
  CollectionReference<UserBookModel> collectionReference() {
    return userDoc().collection('books').withConverter(
          fromFirestore: (snapshot, _) => UserBookModel.fromSnapshot(snapshot),
          toFirestore: (payload, _) => payload.toJson(),
        );
  }
```

```
  Future<List<UserBookModel>> getUncompletedBooks() async {
    final response = await collectionReference()
        .where('completed', isNotEqualTo: true)
        .get();

    return response.docs.map((doc) => doc.data()).toList();
  }

  Future<List<UserBookModel>> getUserBooks() async {
    final response = await collectionReference().get();

    return response.docs.map((doc) => doc.data()).toList();
  }

  Future<UserBookModel> addBookToCollection(
    UserBookModel payload,
  ) async {
    final response = await collectionReference().add(payload);

    return payload.copyWith(uid: response.id);
  }

  Future<void> updateBook(UserBookModel payload) {
    return collectionReference().doc(payload.uid).set(payload);
  }
}
```

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:injectable/injectable.dart';

import 'package:boo_book/models/index.dart';
import 'mixin/firestore_mixin_repo.dart';

@injectable
class CalendarRepository extends FirestoreMixinRepo<CalendarBookModel> {
  CalendarRepository({
    @Named('userUid') required super.userUid,
  });

  @override
  CollectionReference<CalendarBookModel> collectionReference() {
    return userDoc().collection('calendar').withConverter(
          fromFirestore: (snapshot, _) =>
              CalendarBookModel.fromSnapshot(snapshot),
          toFirestore: (payload, _) => payload.toJson(),
        );
  }
```

```
  Future<List<CalendarBookModel>> getCalendarData({
    required DateTime from,
    required DateTime to,
  }) async {
    final fromTimestamp = Timestamp.fromDate(from);
    final toTimestamp = Timestamp.fromDate(to);

    final response = await collectionReference()
        .where(
          'date',
          isGreaterThan: fromTimestamp,
          isLessThan: toTimestamp,
        )
        .get();

    return response.docs.map((doc) => doc.data()).toList();
  }

  Future<CalendarBookModel> createCalendarRecord(
    CalendarBookModel payload,
  ) async {
    final response = await collectionReference().add(payload);

    return payload.copyWith(bookUid: response.id);
  }

  Future<void> replaceCalendarRecord(
    String uidToReplace,
    CalendarBookModel replaceWith,
  ) async {
    return collectionReference().doc(uidToReplace).set(replaceWith);
  }
}

import 'dart:convert';

import 'package:boo_book/core/index.dart';
import 'package:boo_book/models/index.dart';
import 'package:boo_book/models/recommendations/models.dart';
import 'package:chat_gpt_sdk/chat_gpt_sdk.dart';
import 'package:hive/hive.dart';
import 'package:injectable/injectable.dart';

@injectable
class RecommendationsRepository {
  final OpenAI gptClient;

  RecommendationsRepository(this.gptClient);
```

```
  Future<void> cacheRecommendations(List<BookSearchModel> books) async {
    final box = Hive.box(EnvConstants.recommendationsBox);

    final saveData = books.map((book) => jsonEncode(book.toJson())).toList();

    return box.put('test', saveData);
  }

  Future<List<BookSearchModel>> getSavedRecommendations() async {
    final box = Hive.box(EnvConstants.recommendationsBox);

    final data = box.get('test') as List?;

    return data?.map((str) {
          final json = jsonDecode(str) as DynamicMap;

          return BookSearchModel.fromJson(json);
        }).toList() ??
        <BookSearchModel>[];
  }

  // TODO(Ira): Think
  Future<List<RecommendationResponse>> getRecommendations(
    List<UserBookModel> userBooks,
  ) async {
    final bookList =
        userBooks.map((book) => '"${book.title}" - ${book.rating}').join(', ');

    final requestText = 'Я читач і хочу знайти для себе нові книжки'
        'Порекомендуй мені якісь ще книжки.'
        'Я прочитав книжки і поставив їм оцінки за 5 бальною шкалою.'
        'Згенеруй 8 рекомендації українською мовою на основі цього списку без російських
письменників не повторюючі прочитані книжки'
        ' $bookList у форматі JSON [{"author": ..., "book": ...}]';

    final request = ChatCompleteText(
      model: GptTurbo1106Model(),
      maxToken: 500,
      temperature: 1.2,
      messages: [
        {'role': 'user', 'content': requestText},
      ],
    );

    final response = await gptClient.onChatCompletion(request: request);

    final choice = response?.choices.firstOrNull;

    if (choice == null || (choice.message?.content.isEmpty ?? true)) {
```

```
      return [];
    }

    final data = jsonDecode(choice.message!.content) as List;

    return data.map((json) => RecommendationResponse.fromJson(json)).toList();
  }
}

import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:injectable/injectable.dart';

import 'package:boo_book/models/index.dart';

// TODO(Ira): Refactor to collection refference
@injectable
class ReviewsRepository {
  final _client = FirebaseFirestore.instance;

  Future<List<BookReviewModel>> getBookReviews(String bookSearchId) async {
    final response = await _client
        .collection('reviews')
        .where('bookSearchId', isEqualTo: bookSearchId)
        .where('description', isNotEqualTo: '')
        .get();

    return response.docs
        .map((doc) => BookReviewModel.fromSnapshot(doc))
        .toList();
  }

  Future<BookReviewModel> createBookReview(BookReviewModel payload) async {
    final response = await _client.collection('reviews').add(payload.toJson());

    return payload.copyWith(uid: response.id);
  }

  Future<void> updateReviewRating(String uid, double rating) {
    return _client.collection('reviews').doc(uid).update({'rating': rating});
  }

  Future<void> updateBookReview(BookReviewModel payload) {
    return _client
        .collection('reviews')
        .doc(payload.uid)
        .update(payload.toJson());
  }
}
```

```
import 'dart:convert';

import 'package:http/http.dart' as http;
import 'package:injectable/injectable.dart';

import '../models/index.dart';

@injectable
class SearchRepository {
 String get endpointUrl => 'https://www.googleapis.com/books/v1/';

 BookSearchModel _converter(DynamicMap json) {
  final itemJson = Map<String, dynamic>.from(json['volumeInfo'])
   ..addAll({
    'id': json['id'],
   });

  return BookSearchModel.fromJson(itemJson);
 }

 Future<BookSearchModel> getBookBySearchId(String id) async {
  final queryUrl = '$endpointUrl' 'volumes/$id';

  final response = await http.get(Uri.parse(queryUrl));
  final json = jsonDecode(response.body);

  return _converter(json);
 }

 Future<BookSearchModel?> searchBook(String title, String author) async {
  final queryUrl = '$endpointUrl'
     'volumes?q=intitle:${title.trim()}'
     '&maxResults=1'
     '&langRestrict=uk'
     '&printType=books';

  final response = await http.get(Uri.parse(queryUrl));

  final data = jsonDecode(response.body) as Map<String, dynamic>;

  final items = ((data['items'] ?? []) as List)
     .cast<DynamicMap>()
     .map((itemsJson) => _converter(itemsJson))
     .toList();

  return items.firstOrNull;
 }

 Future<PaginationResponse<BookSearchModel>> searchBooks({
```

```
    required String query,
    PaginationModel pagination = const PaginationModel.initial(),
  }) async {
    final formattedQuery = query.trim().replaceAll(' ', '+');

    final queryUrl = '$endpointUrl'
        'volumes?q=intitle:$formattedQuery'
        '&maxResults=${pagination.limit}'
        '&startIndex=${pagination.startOffset}'
        '&langRestrict=uk'
        '&printType=books';

    final response = await http.get(Uri.parse(queryUrl));

    final data = jsonDecode(response.body) as Map<String, dynamic>;

    final totalItems = data['totalItems'] as int;
    final items = ((data['items'] ?? []) as List)
        .cast<DynamicMap>()
        .map((itemsJson) => _converter(itemsJson))
        .toList();

    final newPagination = PaginationModel(
      totalItems: totalItems,
      limit: pagination.limit,
      startOffset: items.length,
    );

    return (
      items: items,
      pagination: newPagination,
    );
  }
}

import 'package:flutter/material.dart';

import 'package:boo_book/localization/index.dart';
import 'package:boo_book/router/index.dart';
import 'package:boo_book/styles/index.dart';

@RoutePage()
class MainScreen extends StatelessWidget {
  const MainScreen({super.key});

  @override
  Widget build(BuildContext context) {
    EasyLocalization.of(context);
```

```
const iconHeight = 26.0;

final bottomPadding = MediaQuery.viewPaddingOf(context).bottom;

return AutoTabsScaffold(
 routes: const [
  HomeRoute(),
  LibraryRoute(),
  RecommendationsRoute(),
  ProfileRoute(),
 ],
 bottomNavigationBuilder: (_, tabsRouter) {
  return Container(
   height: 68 + bottomPadding,
   alignment: Alignment.bottomCenter,
   decoration: const BoxDecoration(
    color: AppColors.navigationBar,
    borderRadius: BorderRadius.vertical(
     top: Radius.circular(20),
    ),
   ),
   child: BottomNavigationBar(
    iconSize: 32,
    elevation: 0,
    showSelectedLabels: false,
    showUnselectedLabels: false,
    backgroundColor: Colors.transparent,
    currentIndex: tabsRouter.activeIndex,
    type: BottomNavigationBarType.fixed,
    onTap: (index) {
     if (index == 2) {
      context.pushRoute(const SearchRoute());
     } else {
      tabsRouter.setActiveIndex(
       index > 2 ? index - 1 : index,
      );
     }
    },
    items: [
     BottomNavigationBarItem(
      icon: tabsRouter.activeIndex == 0
         ? Assets.icons.homeBold.svg(height: iconHeight)
         : Assets.icons.home.svg(height: iconHeight),
      label: 'Home',
     ),
     BottomNavigationBarItem(
      icon: tabsRouter.activeIndex == 1
         ? Assets.icons.libraryBold.svg(height: iconHeight)
         : Assets.icons.library.svg(height: iconHeight),
```

```
            label: 'Library',
          ),
          BottomNavigationBarItem(
            icon: Assets.icons.search.svg(),
            label: 'Search',
          ),
          BottomNavigationBarItem(
            icon: tabsRouter.activeIndex == 2
                ? Assets.icons.communityBold.svg(height: iconHeight - 3)
                : Assets.icons.community.svg(height: iconHeight - 3),
            label: 'Community',
          ),
          BottomNavigationBarItem(
            icon: tabsRouter.activeIndex == 3
                ? Assets.icons.profileBold.svg(height: iconHeight)
                : Assets.icons.profile.svg(height: iconHeight),
            label: 'Profile',
          ),
        ],
      ),
    );
  },
 );
 }
}

import 'dart:async';

import 'package:injectable/injectable.dart';
import 'package:stx_flutter_form_bloc/stx_flutter_form_bloc.dart';

import 'package:boo_book/blocs/index.dart';
import 'package:boo_book/models/index.dart';
import 'package:boo_book/repositories/index.dart';

@injectable
class CompletedBookModalBloc extends FormBloc<UserBookModel, String> {
 late final InputFieldBloc<double> rating;
 late final TextFieldBloc review;

 UserBookModel initial;

 final AuthBloc authBloc;
 final BooksRepository booksRepository;
 final ReviewsRepository reviewsRepository;

 CompletedBookModalBloc({
  @factoryParam required this.initial,
  required this.authBloc,
```

```
  required this.booksRepository,
  required this.reviewsRepository,
 }) {
  rating = InputFieldBloc(
   defaultValue: 0,
   initialValue: initial.rating,
  );

  review = TextFieldBloc(
   initialValue: initial.review?.description,
  );

  addFields([
   rating,
   review,
  ]);
 }

 Future<void> saveReview() async {
  final reviewText = review.value ?? '';
  final userProfile = authBloc.state.user!;

  try {
   emitLoading();

   final payload = BookReviewModel(
    uid: initial.review?.uid ?? '',
    bookSearchId: initial.searchUid,
    created: DateTime.now(),
    description: reviewText,
    rating: rating.value,
    userPreview: UserReviewModel.fromProfile(userProfile),
   );

   final BookReviewModel response;

   if (initial.review == null) {
    response = await reviewsRepository.createBookReview(payload);
   } else {
    await reviewsRepository.updateBookReview(payload);

    response = payload;
   }

   review.updateInitial(reviewText);

   final userReview = Review(
    uid: response.uid,
    description: reviewText,
```

```
      );

      initial = initial.copyWith(
        review: userReview,
        rating: rating.value,
      );

      await booksRepository.updateBook(initial);

      emitSuccess(initial);
    } catch (e, stackTrace) {
      addError(e, stackTrace);

      emitFailure('Щось пішло не так');
    }
  }

  @override
  FutureOr<void> onSubmit() async {
    emitLoading();

    try {
      final payload = initial.copyWith(
        rating: rating.value,
      );

      final request = [
        booksRepository.updateBook(payload),
      ];

      if (initial.review != null) {
        request.add(
          reviewsRepository.updateReviewRating(
            initial.review!.uid,
            rating.value,
          ),
        );
      }

      await Future.wait(request, eagerError: true);

      emitSuccess(payload);
    } catch (e, stackTrace) {
      addError(e, stackTrace);

      emitFailure('Щось пішло не так');
    }
  }
}
```

```
import 'package:flutter/material.dart';

import 'package:cached_network_image/cached_network_image.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:flutter_rating_bar/flutter_rating_bar.dart';

import 'package:boo_book/blocs/index.dart';
import 'package:boo_book/core/index.dart';
import 'package:boo_book/models/index.dart';
import 'package:boo_book/router/index.dart';
import 'package:boo_book/services/index.dart';
import 'package:boo_book/styles/index.dart';
import 'package:boo_book/widgets/index.dart';
import 'completed_book_modal_bloc.dart';
import 'widgets/index.dart';

@RoutePage()
class CompletedBookModalScreen extends StatelessWidget
   implements AutoRouteWrapper {
 const CompletedBookModalScreen({
  super.key,
  required this.book,
 });

 final UserBookModel book;

 @override
 Widget wrappedRoute(BuildContext context) {
  final formBloc = getIt<CompletedBookModalBloc>(param1: book);

  return BlocProvider(
   create: (_) => formBloc,
   child: this,
  );
 }

 @override
 Widget build(BuildContext context) {
  final textTheme = context.textTheme;
  final formBloc = context.read<CompletedBookModalBloc>();

  final book = formBloc.initial;

  final freeSpace =
     context.screenHeight - 646 - AppStyleConstants.landingTopPadding;

  return CustomFormBlocListener(
   formBloc: formBloc,
```

```
onSuccess: (context, state) {
 final book = state.response as UserBookModel?;

 if (book != null) {
  context.read<LibraryBloc>().editItem(book);
 }
},
child: LandingPageScaffold(
 appBar: AppBar(
  leading: BackButton(
   onPressed: context.popRoute,
  ),
 ),
 body: SingleChildScrollView(
  child: Column(
   children: [
    SizedBox(
     height: 200,
     child: Row(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
       Container(
        width: 154,
        padding: const EdgeInsets.symmetric(
         vertical: 16,
         horizontal: 22,
        ),
        decoration: BoxDecoration(
         color: AppColors.card,
         borderRadius: BorderRadius.circular(16),
        ),
        child: ClipRRect(
         borderRadius: BorderRadius.circular(6),
         child: CachedNetworkImage(
          fit: BoxFit.fill,
          imageUrl: book.imageUrl,
         ),
        ),
       ),
       const SizedBox(width: 16),
       Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
         Text(
          book.title,
          style: textTheme.titleMedium,
         ),
         const SizedBox(height: 10),
         Text(
```

```
                book.author,
                style: textTheme.titleSmall,
              ),
              const SizedBox(height: 20),
              RatingBar.builder(
                itemSize: 22,
                initialRating: book.rating,
                unratedColor: Colors.grey,
                itemPadding: const EdgeInsets.only(right: 4),
                itemBuilder: (_, index) {
                  return Assets.icons.start.svg();
                },
                onRatingUpdate: (rating) {
                  formBloc.rating.changeValue(rating);
                  formBloc.submit();
                },
              ),
            ],
          ),
        ],
      ),
    ),
    const SizedBox(height: 16),
    BookReviewFormBuilder(
      fieldBloc: formBloc.review,
      onSave: formBloc.saveReview,
    ),
    SizedBox(height: freeSpace),
    BookAnalytics(
      book: book,
    ),
  ],
 ),
),
),
);
}
}

import 'dart:async';

import 'package:injectable/injectable.dart';
import 'package:stx_flutter_form_bloc/stx_flutter_form_bloc.dart';

import 'package:boo_book/blocs/calendar/calendar_bloc.dart';
import 'package:boo_book/blocs/index.dart';
import 'package:boo_book/core/index.dart';
import 'package:boo_book/models/index.dart';
import 'package:boo_book/repositories/index.dart';
```

```
@injectable
class ReadingBookModalBloc extends FormBloc<UserBookModel, String> {
  late final SelectFieldBloc<UserBookModel> currentBook;

  late final NumberFieldBloc timerSeconds;
  late final NumberFieldBloc remainingTime;

  late final NumberFieldBloc startPage;

  late final ListFieldBloc<BookReadingRecord> records;

  final UserBookModel initial;
  final CalendarBloc calendarBloc;
  final BooksRepository booksRepository;

  ReadingBookModalBloc({
    @factoryParam required this.initial,
    required this.calendarBloc,
    required this.booksRepository,
  }) {
    currentBook = SelectFieldBloc(
      initialValue: initial,
      forceValueToSet: true,
    );

    remainingTime = NumberFieldBloc(
      initialValue: initial.pagesPerSecond != 0
          ? initial.pageCount ~/ initial.pagesPerSecond
          : null,
    );

    startPage = NumberFieldBloc(
      initialValue: initial.readingRecords.firstOrNull?.pageCount,
    );

    timerSeconds = NumberFieldBloc(
      initialValue: 0,
      extraData: TimerStatus.disabled,
    );

    records = ListFieldBloc(
      initialValue: initial.readingRecords,
    );

    addFields([
      timerSeconds,
      remainingTime,
      startPage,
```

```
    currentBook,
    records,
  ]);

  // All subscriptions which are added through [addSubscriptions]
  // Are going to be cancelled in bloc.onClose
  // ignore: cancel_subscriptions
  final timerSubscription = Stream.periodic(
    const Duration(seconds: 1),
    (value) {
      return value;
    },
  ).listen((event) {
    final currentValue = timerSeconds.value ?? 0;

    timerSeconds.changeValue(currentValue + 1, forceChange: true);
  })
    ..pause();

  addSubscriptions([
    timerSubscription,
    timerSeconds.extraDataStream.listen((timerStatus) {
      switch (timerStatus) {
        case TimerStatus.active:
          timerSubscription.resume();
        case TimerStatus.paused:
          timerSubscription.pause();
        case TimerStatus.disabled:
        // Do nothing
      }
    }),
  ]);
}

void finishReading(int pageCount) {
  final userRed = pageCount - (startPage.value ?? 0);
  final remaining = userRed / initial.pageCount;

  final newRecords = BookReadingRecord(
    id: records.length,
    created: DateTime.now(),
    duration: timerSeconds.value ?? 0,
    pageCount: pageCount,
    percentage: (remaining * 100).toInt(),
  );

  timerSeconds.changeValue(0);
  records.insert(0, newRecords);
```

```
 submit();
}

@override
FutureOr<void> onSubmit() async {
 emitLoading();

 final recordsSum =
    records.value.fold((pageCount: 0, duration: 0), (prev, record) {
  return (
   pageCount: prev.pageCount + record.pageCount,
   duration: prev.duration + record.duration
  );
 });

 final pagesPerSecond = recordsSum.pageCount / recordsSum.duration;

 try {
  final now = DateTime.now();
  final book = currentBook.value!;
  final latestRecords = records.value.first;
  final completed = latestRecords.pageCount >= book.pageCount;

  final remaining = latestRecords.pageCount / book.pageCount;

  final payload = book.copyWith(
   readingRecords: records.value,
   completed: completed,
   started: book.started ?? now,
   lastRed: now,
   progress: (remaining * 100).toInt(),
   pagesPerSecond: pagesPerSecond,
   readingDuration: recordsSum.duration,
  );

  await booksRepository.updateBook(payload);
  calendarBloc.addRecordAsync(
   latestRecords.toCalendarModel(
    bookUid: book.uid,
    imageUrl: payload.imageUrl,
   ),
  );

  remainingTime.changeValue(book.pageCount ~/ pagesPerSecond);
  currentBook.changeValue(payload);

  emitSuccess(payload);
 } catch (e, stackTrace) {
  addError(e, stackTrace);
```

```
      emitFailure('Something went wrong!');
    }
  }
}

import 'package:flutter/material.dart';

import 'package:flutter_bloc/flutter_bloc.dart';

import 'package:boo_book/blocs/index.dart';
import 'package:boo_book/models/index.dart';
import 'package:boo_book/router/index.dart';
import
'package:boo_book/screens/search/modals/search_book_details_modal/widgets/index.da
rt';
import 'package:boo_book/services/index.dart';
import 'package:boo_book/widgets/index.dart';
import 'reading_book_modal_bloc.dart';
import 'widgets/index.dart';

@RoutePage()
class ReadingBookModalScreen extends StatelessWidget
    implements AutoRouteWrapper {
  const ReadingBookModalScreen({
    super.key,
    required this.book,
  });

  final UserBookModel book;

  @override
  Widget wrappedRoute(BuildContext context) {
    final formBloc = getIt<ReadingBookModalBloc>(
      param1: book,
    );

    return BlocProvider(
      create: (_) => formBloc,
      child: this,
    );
  }

  @override
  Widget build(BuildContext context) {
    final formBloc = context.read<ReadingBookModalBloc>();

    return CustomFormBlocListener<ReadingBookModalBloc, UserBookModel>(
      formBloc: formBloc,
```

```
onSuccess: (_, state) {
 final book = state.response;

 if (book != null) {
  context.read<HomeBloc>().editItem(book);
  context.read<LibraryBloc>().editItem(book);

  if (book.completed) {
   context.replaceRoute(
    CompletedBookModalRoute(
     book: book,
    ),
   );
  }
 }
},
child: LandingPageScaffold(
 appBar: AppBar(
  leading: BackButton(
   onPressed: () => context.popRoute(),
  ),
 ),
 safeAreaBottom: false,
 body: CustomScrollView(
  slivers: [
   SliverToBoxAdapter(
    child: Column(
     children: [
      BookAvatarFrame(
       imageUrl: formBloc.initial.imageUrl,
       useChangedImage: true,
      ),
      const SizedBox(height: 20),
      ReadingProgressActionFormBuilder(
       formBloc: formBloc.timerSeconds,
       currentBook: formBloc.currentBook,
       startFrom: () {
        return formBloc.records.firstOrNull?.pageCount;
       },
       onStartReading: (pageCount) {
        formBloc.startPage.changeValue(pageCount);
       },
      ),
      TimeRemainingFormBuilder(
       formBloc: formBloc.remainingTime,
      ),
      const SizedBox(height: 20),
      ReadingTimerFormBuilder(
       formBloc: formBloc.timerSeconds,
```

```
                onFinish: formBloc.finishReading,
              ),
            ],
          ),
        ),
        RecordsTitleBuilder(
          formBloc: formBloc,
        ),
        RecordsListFormBuilder(
          formBloc: formBloc,
        ),
      ],
    ),
  ),
 );  } }

import 'package:flutter/material.dart';

import 'package:flutter_native_splash/flutter_native_splash.dart';
import 'package:provider/provider.dart';

import 'package:boo_book/blocs/index.dart';
import 'package:boo_book/router/index.dart';

@RoutePage()
class AuthScreen extends StatelessWidget {
 const AuthScreen({super.key});

 void _removeSplashScreen() {
  FlutterNativeSplash.remove();
 }

 @override
 Widget build(BuildContext context) {
  final authStatus = context.watch<AuthBloc>().state.status;

  return AutoRouter.declarative(
   routes: (_) {
    switch (authStatus) {
     case AuthenticationStatus.initial:
      return [];
     case AuthenticationStatus.unauthenticated:
      _removeSplashScreen();
      return [const LoginRoute()];
     case AuthenticationStatus.authenticated:
      _removeSplashScreen();
      return [const HomeRoute()];
    }
   },   );  }  }
```

# ДОДАТОК Г

## Ілюстративна частина

## ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА КРОСПЛАТФОРМНОГО МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ УПРАВЛІННЯ ЧИТАЦЬКОЮ АКТИВНІСТЮ

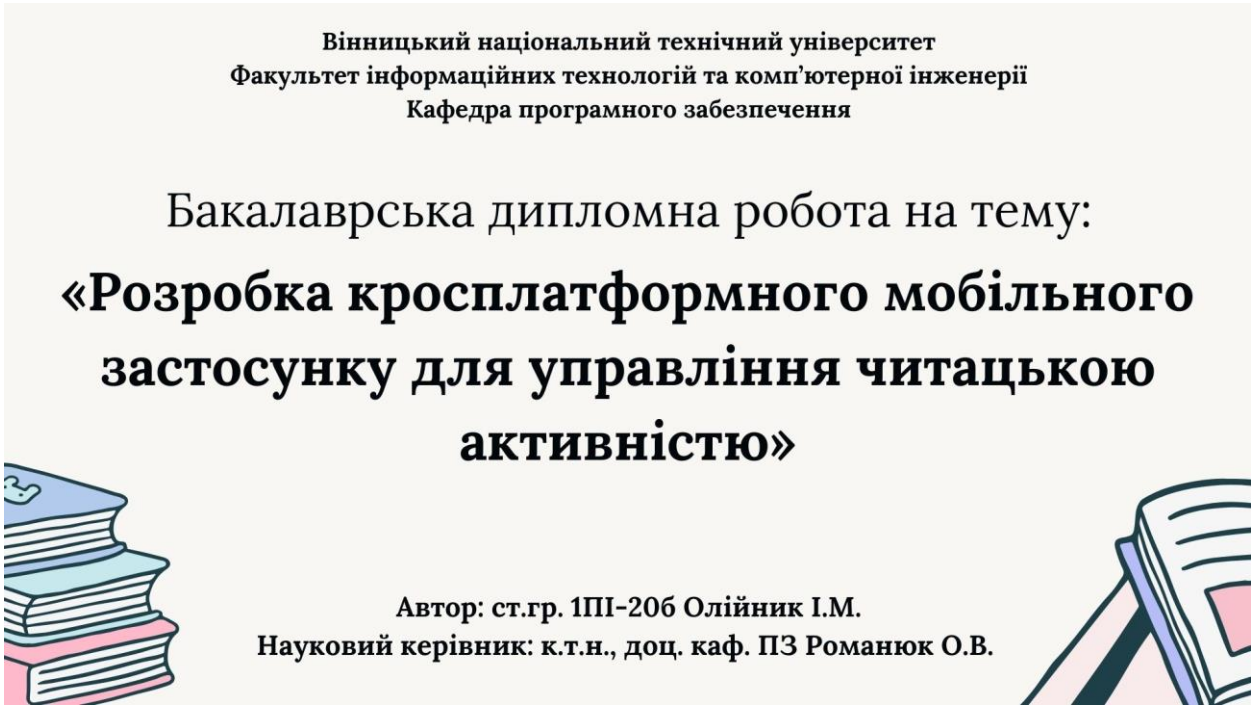



Рисунок Г.1 – Титульний слайд

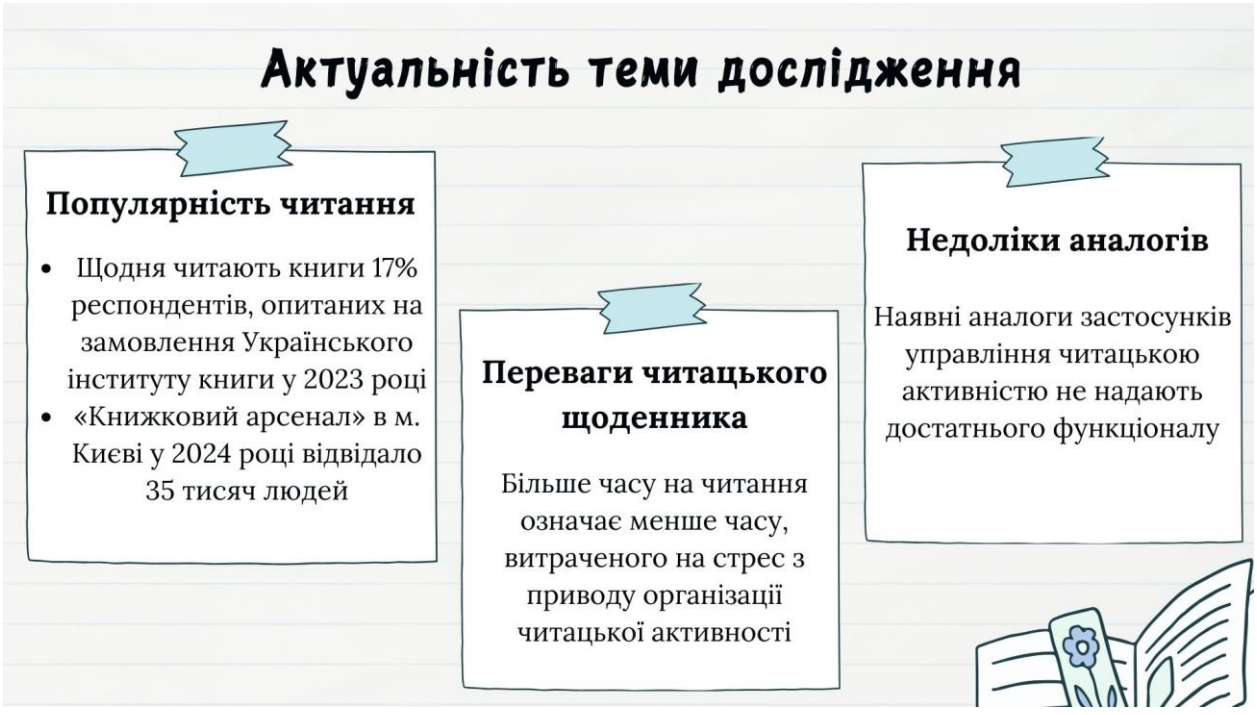



Рисунок Г.2 – Актуальність теми

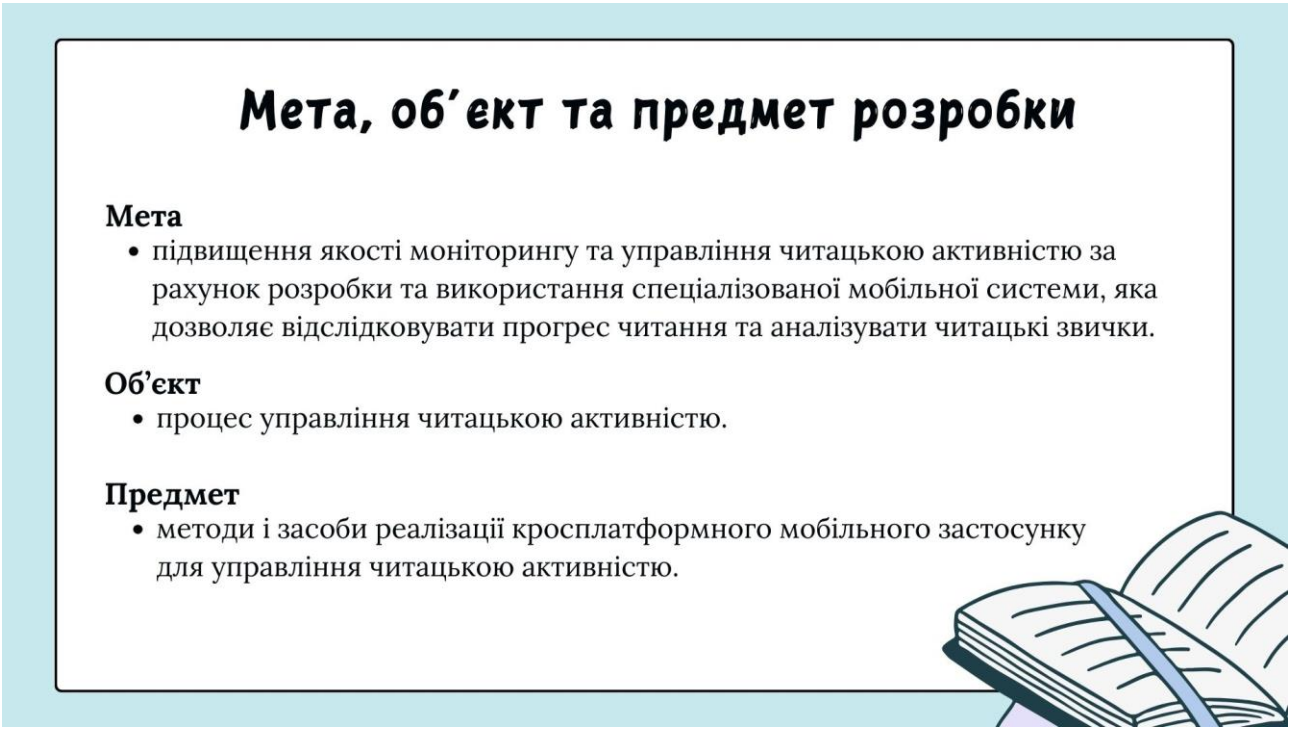



Рисунок Г.3 – Мета, об’єкт та предмет розробки

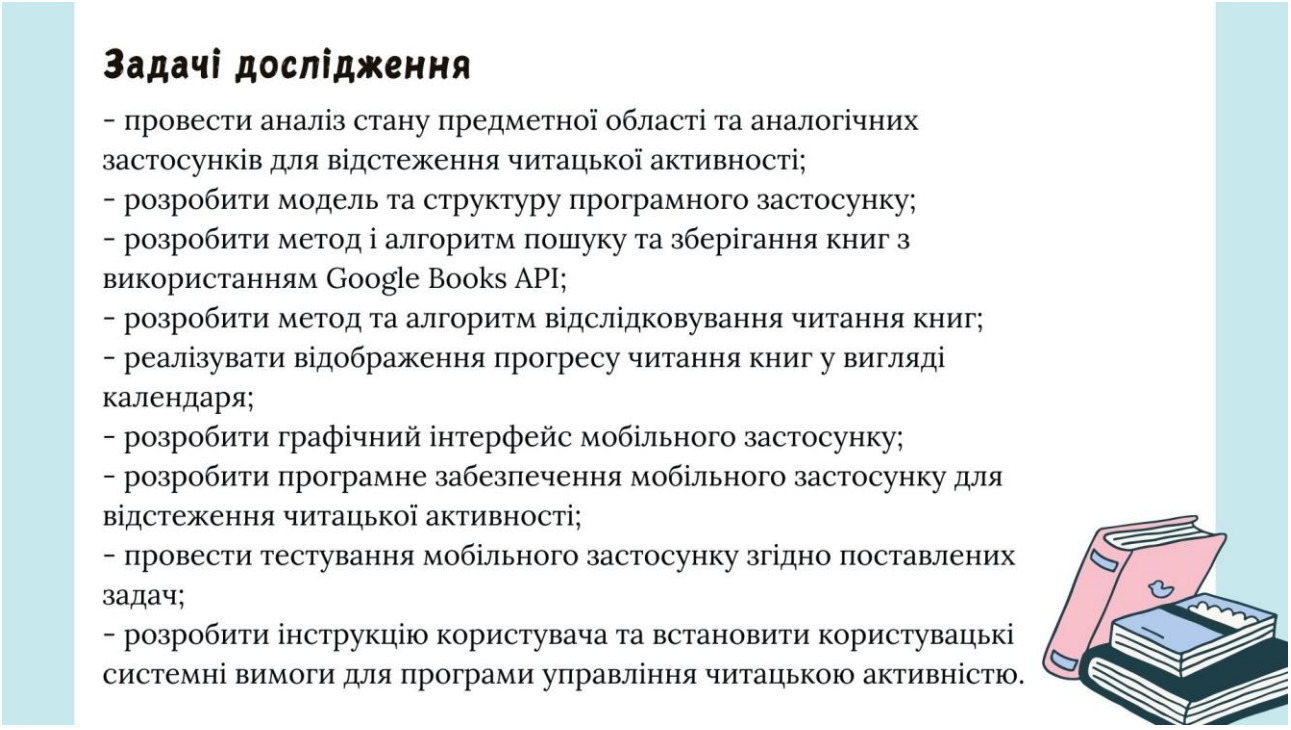



Рисунок Г.4 – Задачі дослідження

## Новизна отриманих результатів

1. Уперше запропоновано відображення прогресу читання у вигляді календаря,який дозволяє користувачеві візуально відстежувати свої щоденні читацькі сесії та переглядати їх у зручному форматі.

2. Удосконалено метод пошуку та зберігання книг, який на відміну від відомих, використовує Google Books API, що дозволило уникнути необхідності збереження книг на власному сервері.

3. Удосконалено метод відслідковування читання книг, який на відміну від відомих методів, дозволяє вибирати сторінки з яких починати та завершувати читання та надає можливість поставити таймер на паузу, що забезпечує більш точне та ефективне визначення читацького прогресу.


Рисунок Г.5 – Новизна отриманих результатів

## Практична цінність отриманих результатів

Практична цінність отриманих результатів полягає в тому, що на основі отриманих в бакалаврській дипломній роботі теоретичних положень запропоновано алгоритми та розроблено програмні засоби для управління читацькою активністю.

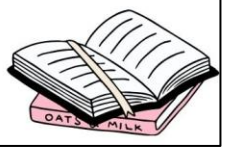



Рисунок Г.6 – Практична цінність отриманих результатів

## Порівняльний аналіз програм-аналогів

| Критерій | Goodreads | TBR - Bookshelf | The Storygraph | Власна реалізація BooBook |
|---|---|---|---|---|
| Зміна критерію кількості сторінок у книзі | - | - | - | + |
| Відслідковування прочитаного за кількістю сторінок | + | - | + | + |
| Наявність календаря | - | - | - | + |
| Розширена аналітика читацької активності | - | + | + | + |
| Можливість створювати персоналізовані списки за певними критеріями | + | - | - | + |
| Можливість переглядати відгуки читачів певної книги | + | - | - | + |
| **Сумарний коефіцієнт** | **3** | **1** | **2** | **6** |

Рисунок Г.7 – Порівняльний аналіз програм-аналогів

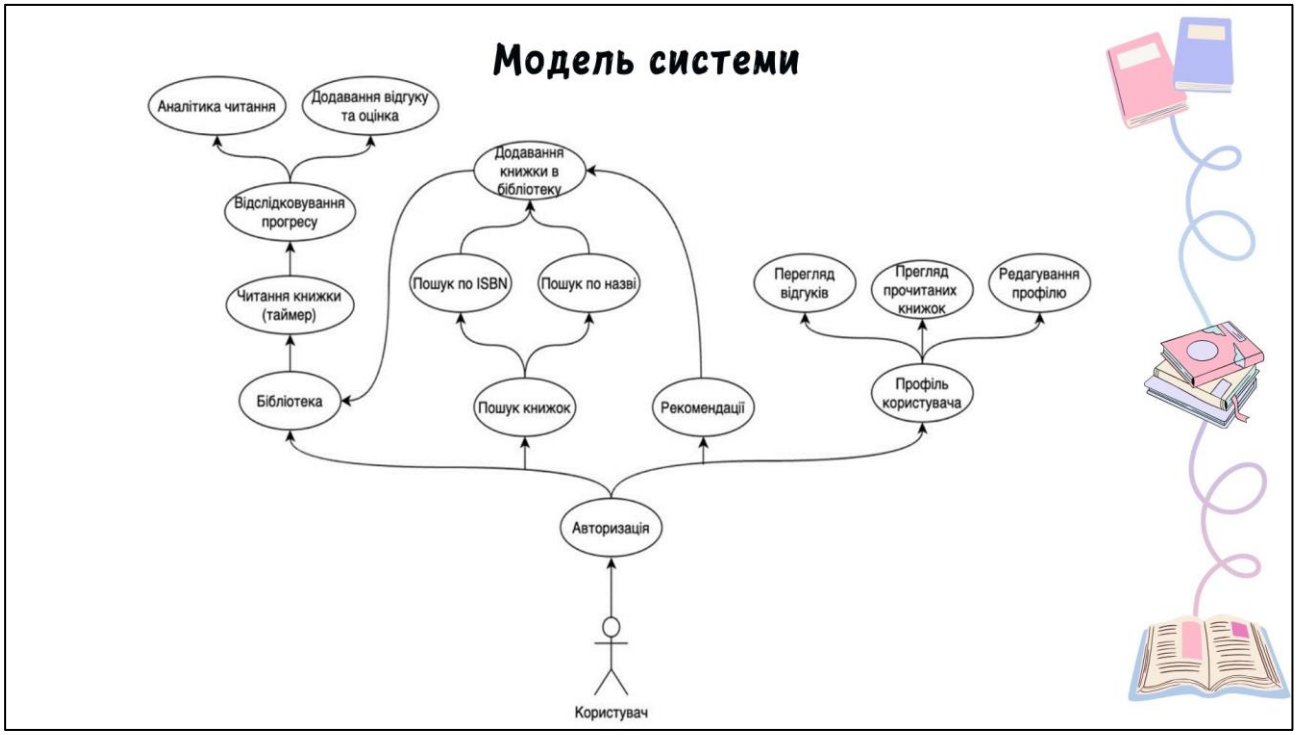



Рисунок Г.8 – Модель системи


Рисунок Г.9 – Структура застосунку

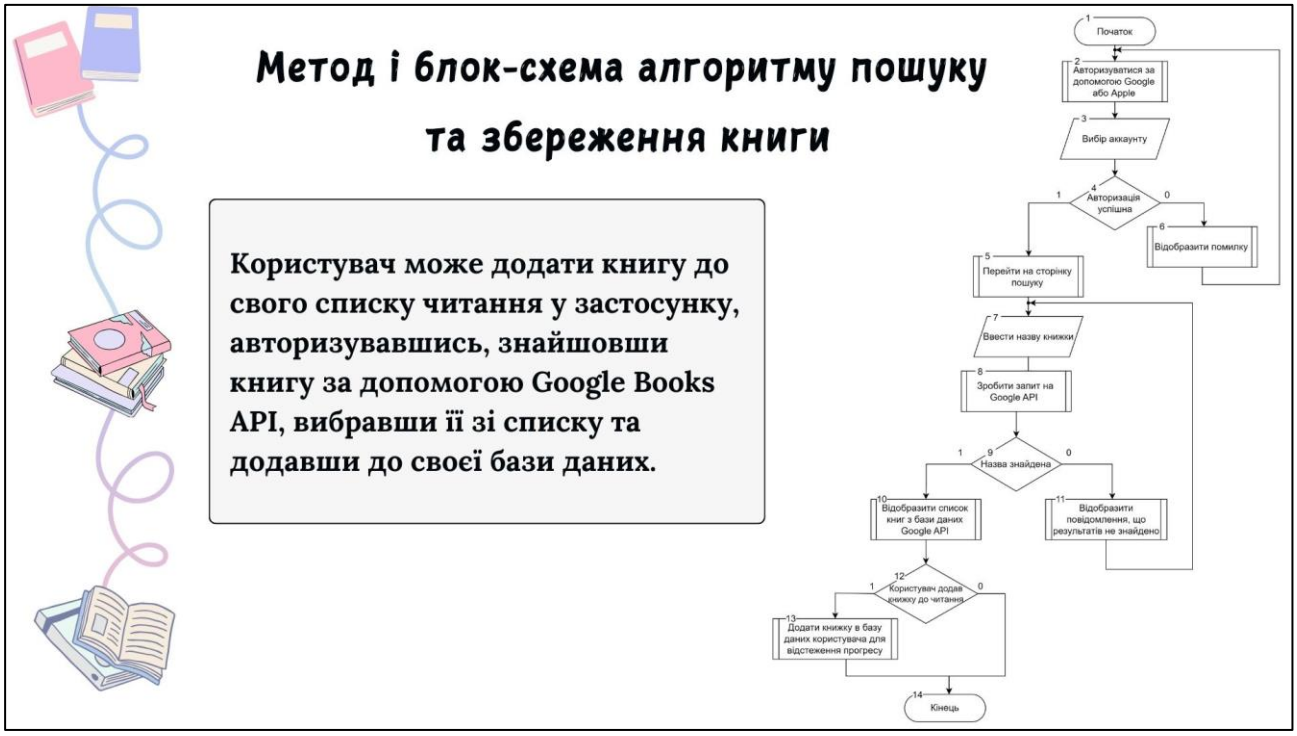



Рисунок Г.10 – Метод і блок-схема алгоритму пошуку та збереження книги


Рисунок Г.11 – Метод і блок-схема алгоритму для відслідковування читання книги


Рисунок Г.12 – Блок-схема алгоритму збереження запису в календар

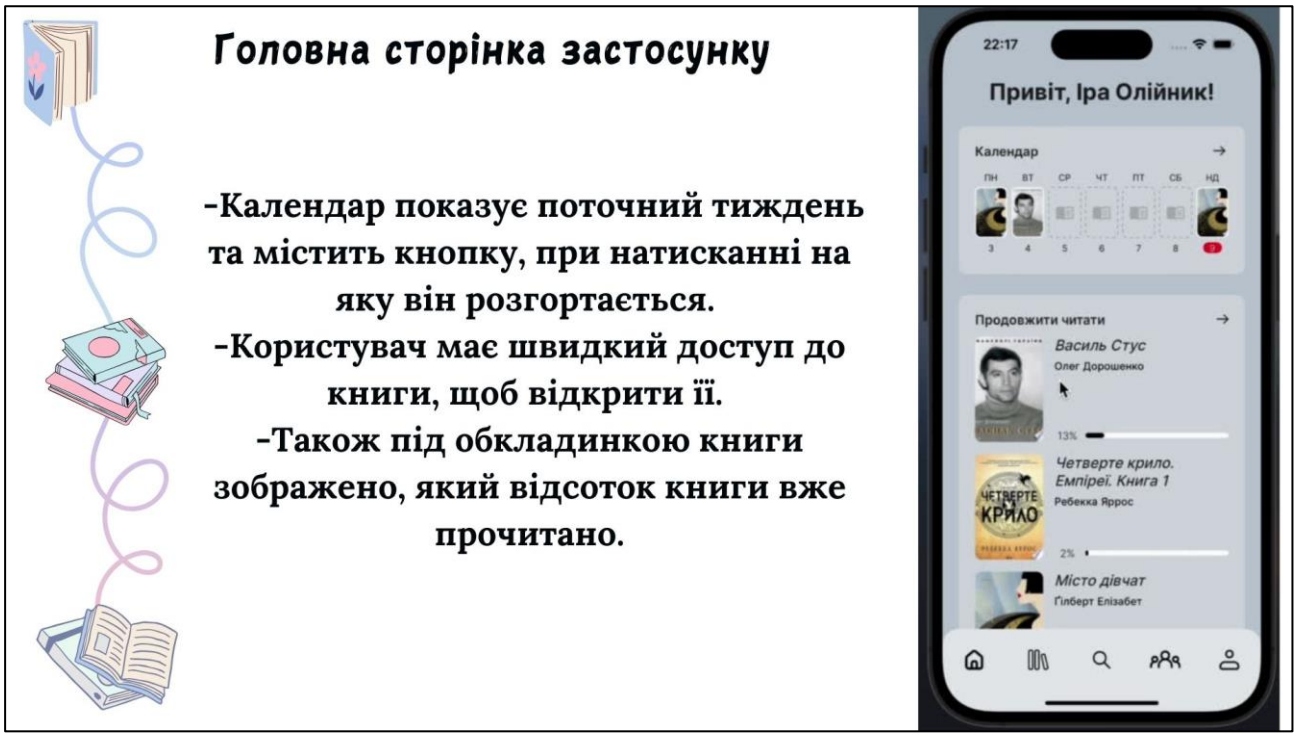



Рисунок Г.13 – Головна сторінка застосунку


Рисунок Г.14 – Сторінка бібліотеки користувача

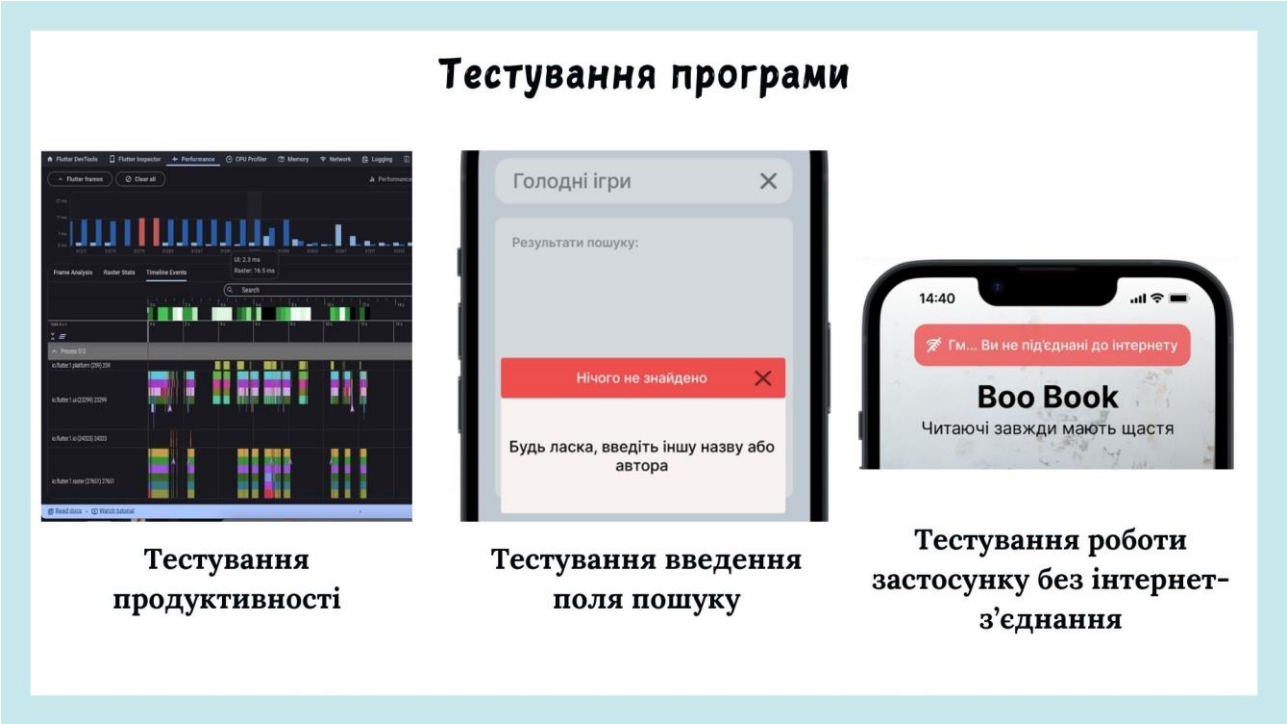



Рисунок Г.15 – Тестування програми

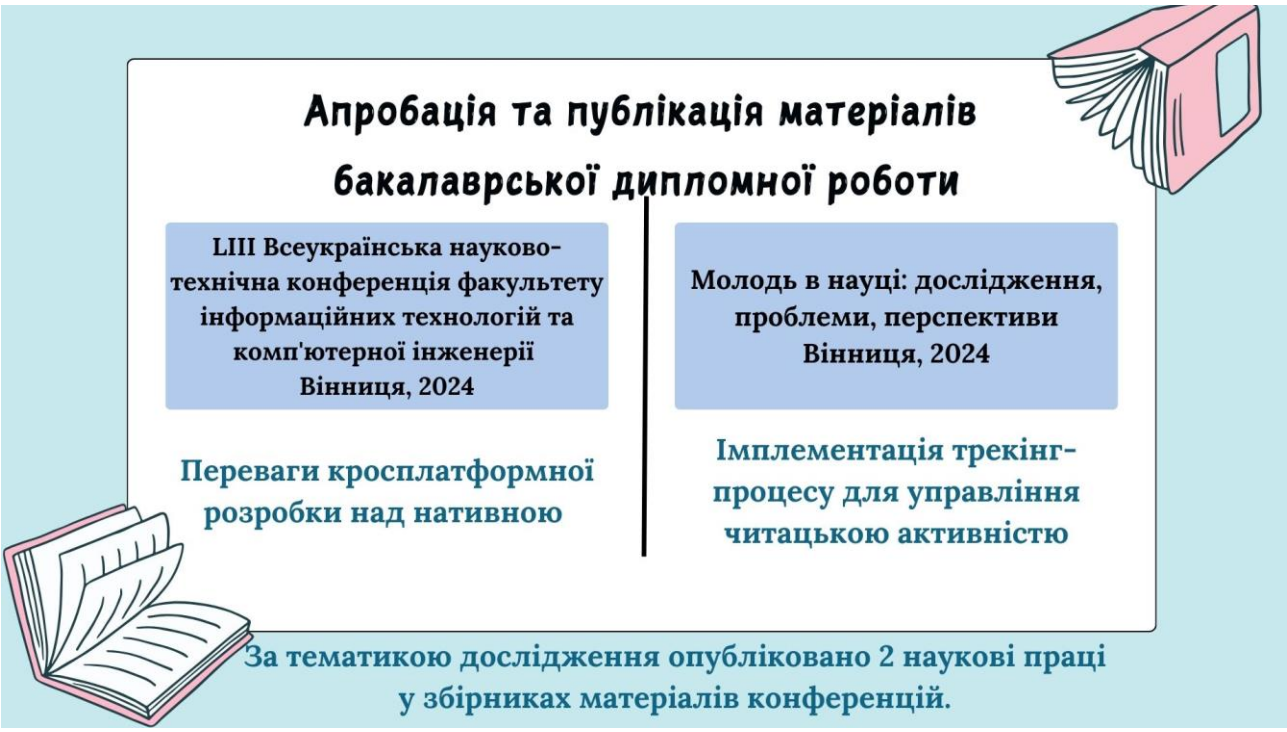



Рисунок Г.16 – Апробація та публікація матеріалів бакалаврської дипломної роботи

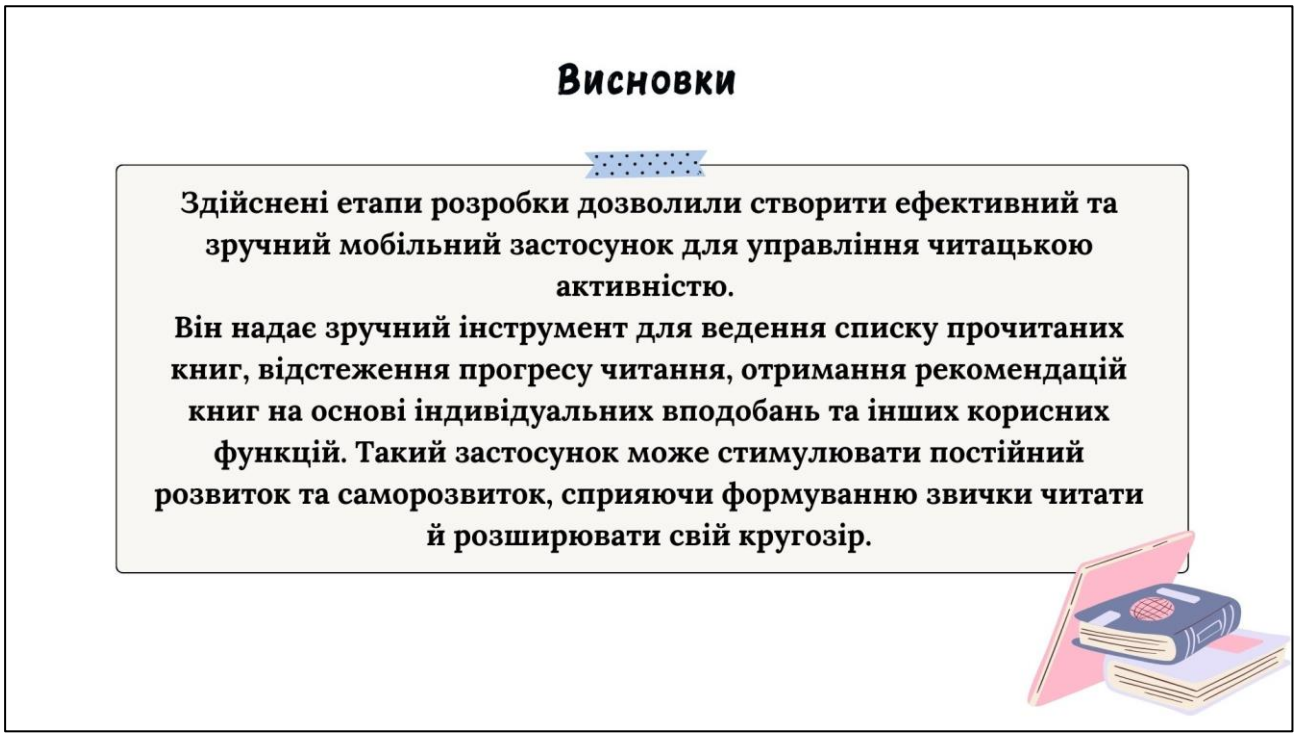



Рисунок Г.17 – Висновки


Рисунок Г.18 – Фінальний слайд