Міністерство освіти і науки України

Луцький національний технічний університет
(повне найменування закладу вищої освіти)

Факультет комп'ютерних та інформаційних технологій
(повне найменування факультету)

Кафедра комп'ютерної інженерії та кібербезпеки
(повне найменування кафедри)

КВАЛІФІКАЦІЙНА РОБОТА ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

Е-БІБЛІОТЕКА ЗАСОБАМИ PYTHON ДЛЯ IOS/ANDROID

E-LIBRARY USING PYTHON TOOLS FOR IOS/ANDROID

спеціальність 123 Комп'ютерна інженерія
(шифр і назва спеціальності)

освітня програма Комп'ютерна інженерія
(назва освітньої програми)

Виконав: здобувач вищої освіти
групи КІ-41
Годлевський Захар Едуардович

(підпис)

Керівник:
к.т.н., доцент
Мельник Катерина Вікторівна

(підпис)

Кваліфікаційну роботу
допущено до захисту
« 13 » червня 2024 р.
Гарант освітньої програми:
к.т.н., доцент
Лавренчук Світлана Василівна

(підпис)

Луцьк – 2024 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *комп'ютерної інженерії та кібербезпеки*
Ступінь вищої освіти: *бакалавр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *123 Комп'ютерна інженерія*
Освітня програма: *«Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ проф. Н.Чернящук
« 10 » 01 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Годлевському Захару Едуардовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *Е-бібліотека засобами Python для IOS/Android*

Керівник роботи *к.т.н., доцент Мельник Катерина Вікторівна*

затверджені наказом закладу вищої освіти від «30» грудня 2023 року № 459/01-02
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи 11.06.2024р.

3. Вихідні дані до роботи *Джерелом розробки є науково-технічна література та публікації в періодичних виданнях з даного питання, опубліковані зарубіжні та вітчизняні роботи в даній області та різні інтернет-ресурси технічного спрямування*

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):
Постановка завдань розробки, аналіз інструментів розробки, розробка мобільного додатку

5. Перелік графічного (ілюстративного) матеріалу:

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Індустрія мобільних додатків	*Мельник К.В., доцент*		
Інструменти розробки мобільних додатків	*Мельник К.В., доцент*		
Розробка додатку Е-бібліотеки	*Мельник К.В., доцент*		
Нормоконтроль	*Багнюк Н.В., доцент*		
Гарант ОП	*Лавренчук С.В., доцент*		
Показник запозичень тексту	***___%***		
Академічна доброчесність	*Міскевич О.І.,асистент*		

7. Дата видачі завдання 10.01.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	*Розділ 1. Індустрія мобільних додатків*	до 15.02.2024 р.	Виконано
2.	*Розділ 2. Інструменти розробки мобільних додатків*	до 15.03.2024 р.	Виконано
3.	*Розділ 3. Розробка додатку Е-бібліотеки*	до 04.05.2024 р.	Виконано
4.	*Висновки*	до 07.05.2025 р.	Виконано
5.	*Формування списку використаних джерел*	до 10.05.2024 р.	Виконано
6.	*Формування додатків*	до 15.05.2024 р.	Виконано
7.	*Оформлення ілюстративного матеріалу*	до 20.05.2024 р.	Виконано
8.	*Нормоконтроль*	до 01.06.2024 р.	Виконано
9.	*Інструментальна перевірка на академічний плагіат*	до 04.06.2024 р	Виконано
10.	*Представлення кваліфікаційної роботи бакалавра до захисту*	до 11.06.2024 р.	Виконано

Здобувач вищої освіти _____ Годлевський З.Е.
(підпис) (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ Мельник К.В.
(підпис) (прізвище, ініціали)

АНОТАЦІЯ

Годлевський З.Е. Е-бібліотека засобами Python для IOS/Android. Рукопис.

Кваліфікаційна робота бакалавра ОП «Комп'ютерна інженерія» спеціальності 123 Комп'ютерна інженерія. Луцький національний технічний університет. Луцьк, 2024. 39 с.

Кваліфікаційна робота складається з вступу, трьох розділів, висновків, списку використаних джерел.

Перший розділ детально розглядає індустрію мобільних додатків, її історію та актуальність. Він аналізує ключові аспекти цієї індустрії, включаючи різноманітність додатків, їхню цільову аудиторію. Розділ також розглядає процес розробки.

Другий розділ зосереджений на огляді інструментів для розробки мобільних додатків. Він включає в себе детальний аналіз різних інструментів, їхніх переваг та недоліків, а також вибір інструментарію для конкретного проекту.

Третій розділ присвячений розробці е-бібліотеки за допомогою Kivy та Kivymd. Він описує процес створення мобільного додатку від початку до кінця, включаючи дизайн, інтерфейс, розробку та упакування проекту для платформ. Розділ також розглядає аспекти розробки за допомогою Kivy та Kivymd, створення інтерфейсу користувача, робота з базами даних. Він закінчується детальним описом процесу упаковки проекту.

Ключові слова: мобільний додаток, python, Kivy, Kivymd.

ANNOTATION

Godlevsky Z.E. E-library using Python for IOS/Android. Manuscript. Bachelor's qualifying thesis of the OP "Computer Engineering" specialty 123 Computer Engineering. Lutsk National Technical University. Lutsk, 2024. 39 c.

The qualification work consists of an introduction, three sections, conclusions, and a list of used sources.

The first chapter takes a detailed look at the mobile application industry, its history and relevance. It analyzes key aspects of this industry, including the variety of applications, their target audience. The chapter also examines the development process.

The second section focuses on an overview of tools for mobile application development. It includes a detailed analysis of various tools, their advantages and disadvantages, as well as the selection of tools for a specific project.

The third chapter is devoted to the development of an e-library using Kivy and Kivymd. It describes the process of creating a mobile app from start to finish, including design, interface, development and packaging of the project for the platforms. The chapter also covers aspects of development using Kivy and Kivymd, creating user interfaces, working with databases. It ends with a detailed description of the project packaging process.

Keywords: mobile application, python, Kivy, Kivymd.

ЗМІСТ

ВСТУП

У сучасному світі, де мобільні додатки стрімко розвиваються і стають невід'ємною частиною нашого життя, статистика свідчить: 86% людей на планеті користуються смартфонами та мобільними додатками. Користувачі в середньому встановлюють 40+ додатків, 9 з яких відкривають щодня. У 2020 році щодня завантажувалося 250 млн додатків, у 2025 році ця цифра сягне 300 млн [1]. Ці цифри підтверджують, що мобільні додатки стають все більш важливим інструментом для доступу до інформації та розваг.

Читання за допомогою мобільних додатків стає все більш доступним та зручним, що робить його привабливим для широкого кола людей. Тим не менш, існують й ті, хто все ще віддає перевагу друкованим книгам. Ці люди цінують тактильні відчуття від книги, запах паперу та можливість створювати власні бібліотеки з фізичних книг [2]. Саме для цієї категорії користувачів розроблена дана електронна бібліотека. Її мінімалістичний інтерфейс та базові функції дозволяють зосередитися на читанні, не відволікаючись на зайві елементи.

Ця кваліфікаційна робота присвячена розробці електронної бібліотеки для мобільних платформ (iOS та Android) з використанням мови програмування Python.

Мета розробки є створенні зручного інструменту, який дозволить користувачам зберігати, організовувати та відстежувати прочитані книги, не відволікаючись від процесу читання.

Об'єктом є додаток е-бібліотеки, розроблений дял iOS та Android платформ.

Предметом дослідження є засоби розробки мобільних додатків.

Завдання, які будуть виконані в ході роботи:

– визначити та обґрунтувати використані інструменти для розробки власного додатку;

– створити мінімалістичний та зручний інтерфейс користувача

– розробити базові функції додатку з Python та фреймворків Kivy та KivyMD.

РОЗДІЛ 1
ІНДУСТРІЯ МОБІЛЬНИХ ДОДАТКІВ

1.1 Історія розвитку

Перший етап у розвитку мобільних додатків датується появою технології WAP (Wireless Application Protocol) у 1998 році. WAP відкрив нові горизонти для з'єднання мобільних пристроїв із Інтернетом, що відбулося випуском Nokia 7100 з WAP-браузером у 1999 році. Це була значна подія, оскільки вона відкрила двері до широкого спектру можливостей, які мобільні додатки можуть надати. Технологія WAP дозволила користувачам не лише грати в ігри, але й читати новини, використовувати електронну пошту, завантажувати карти та навіть бронювати квитки. Це відкрило нові горизонти для розвитку мобільних додатків у різних сферах життя. Проте, важливо зазначити, що технологія WAP мала свої обмеження, такі як низька швидкість передачі даних та протокол, що призводив до обмеження можливості прийняття вхідних дзвінків під час завантаження [3].

У 2001 році з'явилася операційна система Symbian, що стала важливою віхою в історії мобільних додатків. Її задум полягав у створенні єдиної платформи для всіх мобільних пристроїв, що мало полегшити розробку та використання програм. Symbian, розроблена компанією Symbian Ltd., здобула популярність завдяки відкритій платформі, яка дозволяла стороннім розробникам створювати програми. Це призвело до розквіту мобільного ринку, де з'явилася безліч корисних та цікавих додатків [4]. Одночасно з Symbian розвивалися Java-додатки, які відіграли значну роль у розвитку мобільних платформ. Java, мова програмування, що відрізняється платформною незалежністю, дозволяла створювати програми, які могли працювати на різних телефонах з підтримкою Java. Зрештою, Symbian не змогла витримати конкуренцію з iOS та Android, і її використання почало згасати. Java-додатки також втратили популярність з появою нативних програм для нових мобільних платформ. Незважаючи на це, Symbian та Java-додатки залишили значний слід в

історії мобільних технологій. Ці платформи заклали фундамент для сучасних мобільних додатків, проклавши шлях до екосистем iOS та Android, які ми використовуємо сьогодні.

2008 рік став поворотним моментом у розвитку мобільних додатків з появою App Store від Apple та Google Play (тоді Android Market) від Google. Ці централізовані платформи для розповсюдження програм відкрили нові можливості для розробників та користувачів, стимулюючи екосистему мобільних додатків та сприяючи їхньому експоненційному зростанню.

App Store та Google Play не просто стали магазинами додатків, вони революціонізували мобільний ринок. Їхня поява спричинила:

- Стрімке зростання смартфонів: зручність та доступність мобільних додатків зробили смартфони незамінними для мільярдів людей.
- Поява нових бізнес-моделей: розробники отримали нові способи монетизації своїх творінь, завдяки рекламі, вбудованим покупкам та підпискам.
- Трансформацію цілих галузей: мобільні додатки з'явилися в усьому, від електронної комерції та банківської справи до освіти та розваг, змінюючи те, як ми живемо, працюємо та спілкуємося.

Ці платформи стали рушійною силою інновацій, даючи можливість розробникам втілювати свої ідеї в життя та мільйонам користувачів отримувати доступ до корисних та цікавих інструментів просто на своїх телефонах.

1.2 Актуальність мобільних додатків

Мобільні додатки проникли до всіх сфер нашого життя, від освіти та прогнозу погоди до розрахунку калорій та корегування режиму сну. Бізнес-аналітики Data.ai прогнозують, що у 2024 році кількість завантажень мобільних додатків в усьому світі виросте до рекордних 257 мільярдів, що на 1% більше, ніж торік [5]. Ця тенденція призвела до зростання попиту на розробку нових програм, адже користувачі стають все більш вибірковими до нового програмного забезпечення.

Зі зростом популярності мобільні додатки стали проникати у різних сфера життя людини. Соціальні мережі та комунікаційні програми зробили спілкування та обмін досвідом легшими, ніж будь-коли. Вони стали не просто платформами для спілкування, а й потужними інструментами для самовираження, бізнесу та просування ідей. Додатки для роботи та навчання відкрили доступ до знань та співпраці в будь-якому місці. Дозволяють нам планувати наш час, організовувати завдання, спілкуватися з колегами та навчатися новому в будь-який час і в будь-якому місці. Мобільні розваги стали персональними кінотеатрами та ігровими кімнатами в кишені. Додатки для здоров'я та фітнесу допомагають відстежують нашу активність, допомагають нам харчуватися здоровіше, скидати вагу, медитувати та краще спати.

Персоналізація стала ключовою особливістю мобільних додатків, адже вони дозволяють користувачам налаштувати їх під свої потреби та вподобання. Майбутнє мобільних додатків сповнене нових можливостей. Зростаючи технології, такі як доповнена та віртуальна реальність, штучний інтелект, обіцяють ще більше змінити наше життя [6-7].

1.3 Типи мобільних додатків

Мобільні додатки – це програмне забезпечення, яке використовується на смартфонах та планшетах. Їх можна встановити на пристрій перед використанням або завантажити пізніше [8].

Нативні мобільні додатки – це програмне забезпечення, розроблене спеціально для певної платформи, такої як Android або iOS. Розробка потребує використання спеціальних інструментів та мов програмування для кожної платформи, що дає їм ряд переваг [9]:

– Завдяки оптимізації під конкретну ОС, нативні додатки працюють значно швидше та плавніше, ніж кросплатформні аналоги.

– Нативні додатки мають повний доступ до всіх функцій смартфону, таких як камера, GPS, акселерометр, мікрофон, список контактів тощо. Це дозволяє їм пропонувати більш глибокий та функціональний досвід.

– Нативні додатки органічно вписуються в інтерфейс та екосистему мобільної операційної системи, забезпечуючи користувачам звичний та комфортний досвід.

– Розробники нативних додатків можуть використовувати стандартні та кастомні жести, роблячи взаємодію з програмою більш інтуїтивною та приємною.

– Деякі нативні додатки можуть частково або повністю функціонувати без підключення до інтернету, що робить їх зручними у ситуаціях з обмеженим доступом до мережі.

– Нативні додатки зазвичай проходять більш суворий контроль в магазинах додатків, що робить їх більш безпечними та надійними, ніж кросплатформні аналоги.

– Нативні додатки можуть використовувати всі можливості інтерфейсу платформи, роблячи дизайн більш естетичним та зручним.

– Користувачі нативних додатків отримують оновлення швидше, адже розробникам не потрібно адаптувати їх під різні платформи.

Приклади популярних нативних додатків:

– Shazam: додаток використовує мікрофон вашого телефону, щоб визначити граючу пісню, а потім надає інформацію про неї, включаючи назву виконавця, альбом та текст пісні. Shazam доступний як для Android, так і для iOS.

– Instagram: соціальна мережа дозволяє користувачам ділитися фотографіями та відео, а також спілкуватися з друзями та підписниками. Instagram доступний як для Android, так і для iOS.

Мобільні веб-додатки – це не те саме, що й класичні мобільні додатки. Насправді, вони ближче до сайтів, оптимізованих для використання на

смартфоні. Їх просто розробляти, вони не потребують завантаження на пристрій та доступні на будь-яких платформах, що дає їм як переваги та недоліки [9]:

Переваги мобільних веб-додатків:

– Кросплатформеність: працюють на будь-яких смартфонах.

– Не потребують завантаження: економлять місце на пристрої.

– Швидка розробка: завдяки сучасним інструментам та фреймворкам.

– Широке використання HTML5: розширює функціонал.

– Не потребують спеціальних знань: для користування.

Недоліки мобільних веб-додатків:

– Залежність від інтернету: не працюють офлайн.

– Середня продуктивність: може бути повільнішою, ніж у нативних додатків.

– Залежність від швидкості інтернету: робота може погіршуватися при слабкому сигналі.

Приклади популярних нативних додатків:

– Last.fm: мобільний веб-додаток пропонує послуги потокової музики та соціальні функції для спільного обміну музичними вподобаннями через веб-браузер.

– Google Maps: по суті, веб-сайт, він також може розглядатися як мобільний веб-додаток, доступний через веб-браузер на мобільних пристроях та пропонує розширений функціонал навігації та пошуку місць.

Гібридні додатки – це мобільні програми, що поєднують в собі елементи як веб-додатків, так і нативних додатків. Вони написані з використанням веб-технологій, таких як HTML, CSS і JavaScript, але також мають доступ до деяких нативних функцій пристрою, таких як камера, GPS та акселерометр [9].

Переваги гібридних додатків:

– Кросплатформеність: працюють на Android та iOS без окремої розробки.

– Швидка розробка: завдяки веб-технологіям та готовим фреймворкам.

– Економія коштів: дешевше, ніж розробка двох нативних додатків.

– Легкі оновлення: без публікації нових версій в магазинах.

Недоліки гібридних додатків:

– Продуктивність: можуть працювати повільніше, ніж нативні додатки.

– Складність розробки: потребують знань веб-технологій та нативних платформ.

– Залежність від інструментів: не завжди мають однакову підтримку, як нативні платформи.

Приклади популярних нативних додатків:

– HeartCamera: гібридний додаток для iOS дозволяє користувачам додавати мальовані серця до своїх фотографій. Він може працювати офлайн, використовує камеру пристрою та надсилає push-повідомлення.

– TripCase: органайзер подорожей доступний на різних платформах, пропонуючи користувачам планування маршрутів, доступ до офлайн-карт, синхронізацію з календарем та push-повідомлення.

1.4 Життєвим циклом розробки додатків

Розробка додатків – це комплексний процес, що керується життєвим циклом розробки додатків (ADLC). ADLC інтегрує врахування потреб користувачів з попередньою оцінкою потенційних ризиків для мінімізації їх впливу на кінцевий продукт.

Він охоплює всі етапи створення додатків, від визначення функціональних можливостей та потреб користувачів до впровадження, тестування та технічної підтримки. ADLC забезпечує структурований підхід до розробки, підвищуючи загальну якість додатків та знижуючи ймовірність критичних помилок.

Важливим аспектом ADLC є наявність різних моделей управління, які визначають методику розробки. Ці моделі можна обирати залежно від

масштабу проекту, складності програми та уподобань команди розробників. Ось декілька найбільш розповсюджених моделей[10]:

– Каскадна модель (Waterfall Model) – традиційний лінійний підхід, що передбачає послідовну розробку по етапах. Кожен етап, наприклад, збір вимог або тестування, повністю завершується перед переходом до наступного. Ця модель ефективна для проектів з чіткими та сталими вимогами, але вона негнучка та не дозволяє легко інтегрувати зміни під час розробки.

– Гнучка модель (Agile Model) – популярний підхід, що передбачає поетапну розробку з урахуванням відгуків користувачів. На відміну від каскадної моделі, гнучка модель дозволяє командам реагувати на зміни вимог та вдосконалювати додаток впродовж усього циклу розробки. Це робить її корисною для проектів, де необхідні швидкі випуски нових версій або коли кінцеві функціональні вимоги можуть змінюватися.

– Інші моделі ADLC – існують й інші варіанти, окрім каскадної та гнучкої моделей, такі як V-подібна модель (V-Shaped Model) з акцентом на тестуванні, ітеративна модель (Iterative Model) для додатків, що потребують змін у майбутньому, спіральна модель (Spiral Model) для врахування ризиків, та модель DevOps (DevOps Model), що наголошує на співпраці між розробниками та IT-операціями. Кожна модель має свої переваги та недоліки, тому оптимальний вибір залежить від конкретного проекту.

1.5 Етапи розробки мобільних додатків

Процес розробки мобільних додатків – це складний та багатоетапний процес, який вимагає залучення фахівців різного профілю. Можна виділити шість ключових етапів [11]:

1. Формування ідеї:

– Визначаємо проблему та цільову аудиторію: на цьому етапі відбувається мозковий штурм, збір інформації від клієнтів, потенційних

користувачів та розробників. Метою є визначення проблеми, яку вирішуватиме додаток, та цільової аудиторії, якій він буде корисний.

– Формуємо технічні вимоги до додатка: на основі зібраної інформації створюється документ, який детально описує функціональні можливості додатку, його взаємодію з користувачем та технічні характеристики.

2. Планування:

– Детально вивчаємо потреби користувачів: на цьому етапі проводиться ретельний аналіз потреб цільової аудиторії. Завдання полягає в тому, щоб зрозуміти, як користувачі будуть взаємодіяти з додатком, які функції їм необхідні, та які проблеми можуть виникнути під час користування.

– Плануємо функціонал, терміни та бюджет: на основі отриманих даних команда проєкту формує детальний план розробки. Він включає визначення пріоритетних функцій додатку, встановлення термінів виконання для кожного етапу та розрахунок бюджету.

3. Дизайн:

– Створюємо прототип додатку: він полягає у створенні візуального представлення того, як виглядатиме додаток та як користувачі будуть з ним взаємодіяти. Прототипи можуть бути простими ескізами або інтерактивними моделями, які дозволяють протестувати основні функції додатку.

– Опрацюємо інтерфейс та користувацький досвід: дизайнери працюють над створенням привабливого та зручного інтерфейсу користувача додатку. Водночас, UX-дизайнери фокусуються на тому, щоб взаємодія з додатком була інтуїтивною та приємною для користувачів.

4. Розробка:

– Розробники пишуть код та реалізовують функціонал додатка: використовуючи обрані технології та мови програмування, розробники пишуть код додатку. Вони реалізовують функції, визначені на етапі планування, та дбають про те, щоб додаток працював стабільно та ефективно.

5. Тестування:

– Тестувальники перевіряють додаток на помилки: після того, як розробка завершена, додаток потрапляє до команди тестувальників. Вони проводять ретельне тестування, щоб виявити будь-які помилки.

РОЗДІЛ 2
ІНСТРУМЕНТИ РОЗРОБКИ МОБІЛЬНИХ ДОДАТКІВ

Розробка додатків для Android та iOS виконується за допомогою спеціалізованих середовищ та мов програмування: Android Studio (Kotlin або Java) для Android, Xcode (Swift) для iOS. Цей підхід називається “рідним” (Native). Однак, це вимагає розробки та підтримки двох окремих додатків, що вимагає більше ресурсів. Для вирішення цієї проблеми були створені кросплатформенні фреймворки, такі як Flutter (від Google) та React Native (від Facebook), які дозволяють розробляти додатки для обох ОС на основі однієї кодової бази. Це спрощує розробку та підтримку, але має свої обмеження, включаючи потенційну повільнішу роботу додатків [12].

Flutter - відкрита крос-платформний фреймворк від Google для розробки додатків з єдиним інтерфейсом на різних платформах, таких як iOS, Android, веб, Windows, MacOS та Linux. Замість того, щоб писати окремий код для кожної платформи, розробники можуть використовувати одну кодову базу з Flutter, що значно економить час та ресурси. Фреймворк використовує власний графічний рушій, який забезпечує візуальну схожість додатку на різних пристроях, а також надає розробникам більше контролю над зовнішнім виглядом [13].

Хоча Flutter і поступається native розробці у показниках абсолютної продуктивності, різниця стає все менш помітною з кожним оновленням. Крім того, переваги Флаттера в швидкості розробки та економії коштів роблять його привабливим вибором для багатьох проєктів.

Окрім цього, Флаттер пропонує інші зручні функції для розробників:

– Гаряче перезавантаження (hot reload): зміни в коді можна миттєво переглянути в додатку без необхідності перезапускати його повністю, що пришвидшує процес розробки.

– Інструменти візуального відлагодження: корисні інструменти допомагають розробникам легко визначати та вирішувати проблеми з компонуванням інтерфейсу.

– Велика спільнота та підтримка: активна спільнота розробників Flutter завжди готова допомогти, а Google постійно оновлює фреймворк та публікує детальну документацію та навчальні матеріали.

React Native - це фреймворк, який дозволяє веб-розробникам писати код один раз, а потім використовувати його для створення мобільних додатків для iOS та Android. Це означає, що ви можете використовувати свої навички JavaScript для створення чудових мобільних додатків без необхідності вивчати нові мови програмування [14].

Переваги використання React Native:

– Швидша розробка: можна створювати мобільні додатки швидше, ніж за допомогою нативної розробки, оскільки вам не потрібно писати код двічі (по одному разу для iOS та Android).

– Простіше обслуговування: потрібно обслуговувати лише одну кодову базу, а не дві, що економить час і гроші.

– Легше знайти розробників: більше людей знають JavaScript, ніж нативні мови розробки мобільних додатків, тому вам буде легше знайти розробників для вашої команди.

– Кращий досвід розробника: React Native пропонує безліч функцій, які роблять розробку мобільних додатків приємнішою, наприклад, гаряче перезавантаження, яке дозволяє негайно бачити зміни коду на пристрої.

Недоліки використання React Native:

– Молодший фреймворк: React Native порівняно новий фреймворк, тому він не такий зрілий, як деякі інші фреймворки мобільної розробки.

– Не всі функції доступні: деякі функції, доступні на нативних платформах, можуть бути недоступні в React Native.

– Можливі проблеми з продуктивністю: у деяких випадках React Native може бути не таким швидким, як нативні мобільні додатки.

Android та iOS мають відмінності в розробці дизайну. Android використовує матеріальний дизайн з багатовимірними площинами, градієнтами та анімаціями, в той час як iOS слідує принципам плоского дизайну з мінімалізмом та геометричністю. Навігація в Android базується на трьох кнопках, а в iOS - на жестах. Розміри та роздільна здатність екранів також відрізняються: Android має велику різноманітність пристроїв, що вимагає адаптивного дизайну, в той час як у iOS лінійка пристроїв обмежена, що дозволяє створювати більш стабільний дизайн [15].

2.1 Інструментарій для розробки проекту

Проект передбачає створення мобільного додатку, спрямованого на допомогу любителям паперових книг управляти своїми колекціями. Цей додаток буде включати основні функції, такі як додавання нових книг до колекції та пошук серед існуючих книг.

Незважаючи на поширення електронних книг, паперові книги все ще залишаються популярними. Згідно з даними Statista Advertising & Media Outlook, 45% населення США купили паперову книгу у 2020 році, у порівнянні з 23% для електронних книг [16]. Дослідження показують, що читання паперових книг може сприяти кращому сприйняттю, меншому відволіканню, створенню емоційного зв’язку та просто бути приємнішим [17]. Це свідчить про великий попит на додаток, який би допоміг людям організувати свої бібліотеки паперових книг. Завдяки простому інтерфейсу та мінімальному набору функцій, додаток буде доступним для користувачів будь-якого рівня технічної підготовки.

2.2 Мова програмування додатку та база даних

Python вибрано для розробки мобільного додатку завдяки своїй універсальності, простоті та швидкості розробки. Синтаксис Python простий та

зрозумілий, що робить його доступним для досвідчених розробників та початківців, полегшуючи процес розробки та співпрацю в команді. Використання інтерпретатора замість компілятора значно прискорює процес розробки, дозволяючи швидше тестувати та вносити зміни до коду.

Python також дозволяє створювати кросплатформні мобільні додатки, які працюють на iOS та Android, що економить час та ресурси, адже не потрібно писати окремі кодові бази для кожної платформи. Мова має безліч бібліотек для машинного навчання, аналізу даних, графіки та інших завдань, що робить її потужним інструментом для створення складних мобільних додатків. Крім того, Python добре справляється з обробкою великих обсягів даних, що робить його ідеальним для додатків, які потребують аналізу даних або машинного навчання.

Приклади успішних мобільних додатків, розроблених на Python, включають Instagram, який використовує Python для свого бекенду, Dropbox для свого синхронізаційного механізму та API, Spotify для алгоритмів персоналізації та рекомендацій, Disqus для масштабованого бекенду та API, та Uber для системи диспетчеризації та маршрутизації. Основними кросплатформними мобільними фреймворками для Python є Kivy, який дозволяє розробляти мобільні додатки з графічним інтерфейсом, та BeeWare, який використовує нативні інструменти для кожної платформи, забезпечуючи автентичний користувацький досвід.

Незважаючи на те, що Python все ще розвивається як мова для мобільної розробки, його переваги та зростаюча популярність свідчать про те, що він буде відігравати значну роль у майбутньому мобільної розробки [18].

Реляційні бази даних (РБД) – один із найпоширеніших типів баз даних, що використовуються для зберігання, організації та управління інформацією. Засновані на реляційній моделі даних, яку було запропоновано Едгаром Коддом у 1970 році [19].

Кожен додаток має свої власні, ізольовані файли для зберігання даних, які ніяк не взаємодіють. Щоб використовувати дані з різних застосунків, потрібно

відкрити щонайменше два різні файли, скопіювати потрібні дані і зберегти в третьому файлі. І так до нескінченності. Звичайно, щойно з'явилася технічна можливість, інженери одразу ж придумали рішення: зберігати дані в таблицях, пов'язаних між собою, relation – це зв'язок, або відношення. Тому таблиці з даними називають реляційними.

Доступ до реляційних таблиць здійснюється за допомогою мови структурованих запитів (SQL). Реляційні бази даних – це гнучкий і структурований спосіб керування даними, який забезпечує вищий рівень абстракції, що дає змогу розробникам ефективніше працювати з даними.

Швидкий доступ і управління даними реалізуються за допомогою SQL-запитів. Це робить реляційні бази даних ідеальними для широкого спектра додатків, від обліку та управління замовленнями до зберігання інформації про клієнтів і багато чого іншого.

Структура реляційних баз даних

– Реляційні таблиці. Данні організовуються у вигляді таблиць із рядами та стовпцями. Кожна таблиця має ім'я та структуру, визначену схемою даних.

– Рядки (кортежі). Кожен рядок у реляційній таблиці представляє окремий запис і містить інформацію про об'єкт. Кожен запис є унікальним та ідентифікується за допомогою ключа.

– Стовпці (атрибути). Стовпці таблиці являють собою атрибути або характеристики об'єктів, що описуються в таблиці. Кожен стовпець має ім'я і тип даних, який визначає, якого роду інформацію можна зберігати в цьому стовпчику (наприклад, текст, числа, дати і так далі).

– Ключі. У реляційній моделі використовуються ключі для унікальної ідентифікації рядків у таблиці. Основний ключ (Primary Key) забезпечує унікальність кожного рядка, а зовнішній ключ (Foreign Key) створює зв'язки між таблицями.

– Зв’язки. Реляційні бази даних дають змогу встановлювати зв’язки між таблицями, що дає змогу об’єднувати дані з різних таблиць для виконання складних запитів.

– SQL (Structured Query Language). SQL використовується для виконання операцій з даними в реляційних базах даних. Він дає змогу створювати, змінювати, видаляти та витягувати дані з таблиць, а також визначати правила для цілісності даних.

Нормалізація даних – це процес організації даних у реляційній БД з метою мінімізації надлишковості даних, поліпшення цілісності даних і забезпечення ефективності виконання запитів. Цей процес дає змогу розбити великі таблиці на дрібніші, щоб уникнути повторення інформації та зменшити ймовірність помилок.

Нормалізація зменшує надмірність даних, оскільки одна й та сама інформація не зберігається в кількох місцях. Це економить місце і зменшує ризик невідповідності даних. Також вона забезпечує вищий рівень цілісності даних, оскільки зміни даних відбуваються тільки в одному місці, що знову-таки зменшує ймовірність. Добре нормалізовані дані дають змогу виконувати запити більш ефективно, оскільки дані розподілені за дрібнішими таблицями, що зменшує необхідність в об’ємних операціях об’єднання даних.

Робота з реляційними базами даних передбачає виконання запитів і операцій з даними з використанням SQL (Structured Query Language), який є стандартною мовою для взаємодії з реляційними СУБД (системами управління базами даних). Ось кілька прикладів запитів і операцій:

– SELECT (Витяг даних): запит використовується для вилучення даних із таблиці.

– INSERT (Додавання даних): операція використовується для вставки нових даних у таблицю.

– UPDATE (Оновлення даних): операція дає змогу оновлювати наявні дані в таблиці.

– DELETE (Видалення даних): операція використовується для видалення даних із таблиці.

Огляд популярних систем керування реляційними базами даних (СУБД) включає такі рішення, як MySQL, PostgreSQL, Microsoft SQL Server, Oracle Database, SQLite та MariaDB. Кожен з них має свої особливості і підходить для різних завдань і проектів.

У практичних застосуваннях реляційні бази даних широко використовуються в різних галузях, включаючи електронну комерцію, фінанси, охорону здоров'я, логістику, управління ланцюгами поставок та аналітику.

Найкращі практики включають в себе індексування для покращення продуктивності, нормалізацію даних для забезпечення цілісності, оптимізацію запитів для ефективного використання ресурсів, кешування для зменшення навантаження на базу даних, а також резервне копіювання та відновлення для забезпечення безпеки даних.

2.3 Фреймворк для розробки та стиль дизайну

Kivy був обраний для розробки додатку з кількох вагомих причин. Перш за все, Kivy – це мультиплатформний фреймворк з відкритим вихідним кодом для розробки додатків на Python, що розповсюджується за ліцензією MIT. Він підтримує всі основні платформи: Windows, Linux, macOS, iOS, Android та Raspberry PI, що забезпечує універсальність та зручність використання на будь-якому пристрої. Це дозволяє створювати складні мобільні та десктопні додатки, що важливо для сучасних додатків.

Kivy використовує налаштовувані візуальні компоненти (віджети), які забезпечують однаковий вигляд додатків на будь-якій платформі, а також графічний інтерфейс OpenGL ES для прискорення GPU та якісного відображення візуальних компонентів та ефектів. Хоча Kivy орієнтований на створення мобільних додатків з сенсорним інтерфейсом, він також підтримує стандартний ввід через клавіатуру та мишу. Це дозволяє розробляти програмне

забезпечення для всіх основних платформ, що значно розширює аудиторію користувачів.

Kivy дозволяє написати код один раз і запускати додатки на будь-якій платформі майже без змін. Це робить його відмінним вибором для розробки сучасних додатків, оскільки економить час і ресурси на адаптацію коду для різних ОС. Фреймворк базується на принципах абстракції та модульності. Абстракція дозволяє спрощувати типові задачі, такі як відкриття вікна, відображення зображень та тексту, відтворення звуку тощо. Модульність дозволяє розширювати функціональність Kivy за допомогою додаткових програмних компонентів, що надає гнучкість у розробці та розширенні додатку.

Ядро фреймворка включає такі функції, як годинник для планування подій, кеш для кешування часто використовуваних подій та функцій, розпізнавання жестів для виявлення різних типів рухів, мова Kivy для опису користувацьких інтерфейсів та спеціальні класи властивостей для зв’язку коду віджетів з інтерфейсом. Модуль UIX містить віджети та макети для швидкого створення інтерфейсу користувача. Віджети - це елементи інтерфейсу, які реагують на дії користувача, а макети використовуються для розміщення віджетів.

Для створення більш стильних і сучасних інтерфейсів ми можемо використовувати додаткову бібліотеку Kivy MD, яка реалізує концепцію Material Design від Google. Цей дизайн включає лаконічні форми, мінімалістичні кольорові комбінації, імітацію реалістичних матеріалів та плавну анімацію об’єктів.

Kivy складається з кількох ключових блоків. Основна ідея полягає в модульності та абстракції. Абстрагує основні завдання, як-от відкриття вікон, відображення зображень і тексту, відтворення аудіо тощо. Це спрощує використання та розширення API, дозволяючи використовувати специфічних провайдерів для різних платформ, таких як macOS, Linux, Windows. Кожна платформа має свої API, і ми використовуємо спеціалізованих провайдерів для ефективного використання можливостей операційної системи.

Такий підхід застосовується і до обробки введення. Провайдер введення додає підтримку різних пристроїв, як-от трекпад Apple, TUIO чи емулятор миші. Для додавання підтримки нового пристрою потрібно створити клас, який читає дані введення і перетворює їх у події Kivy. Графічний API Kivy є абстракцією OpenGL. Kivy виконує команди апаратного прискорення малювання за допомогою OpenGL, але надає спрощений API для малювання. Це автоматично оптимізує команди малювання, роблячи код більш ефективним.

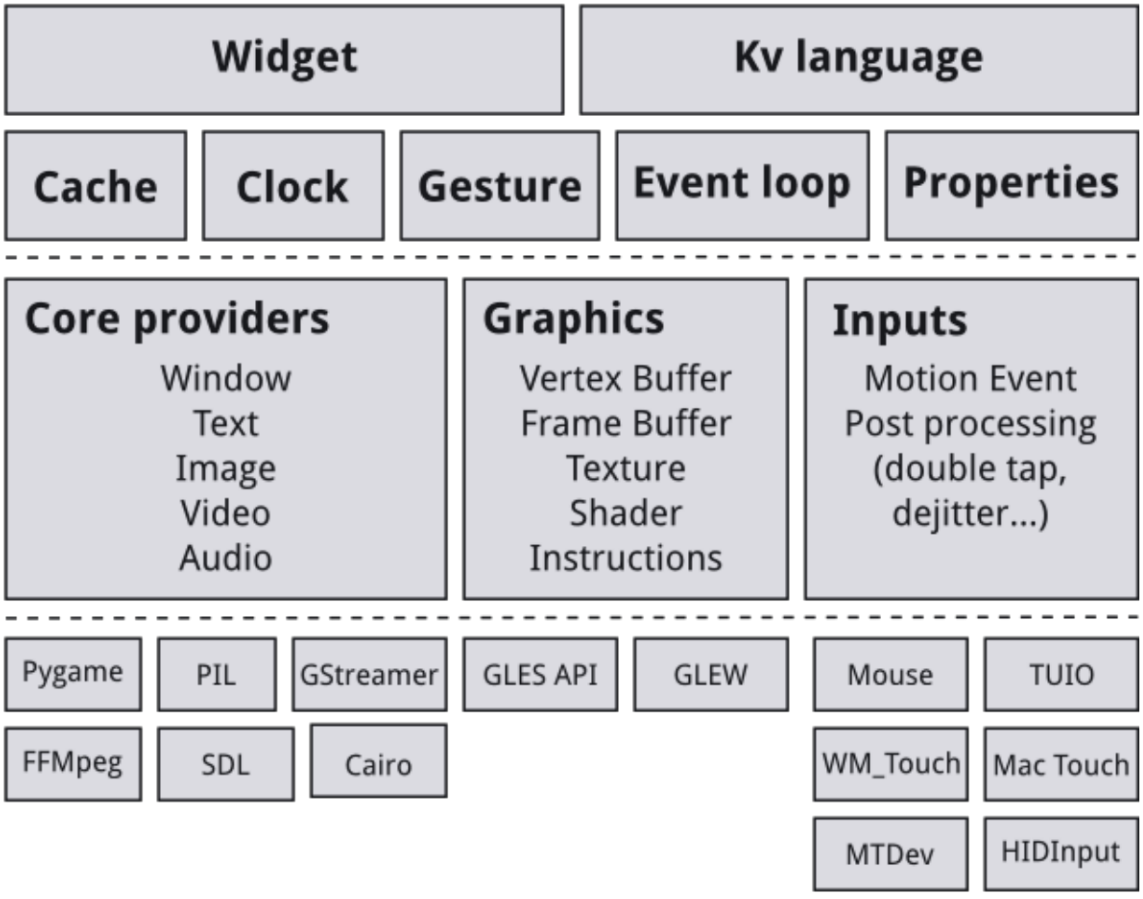

Рисунок 2.1 – Архітектура Kivy [20]

Kivy абстрагує різні типи введення, використовуючи клас Touch для представлення цих подій. Кожен дотик може бути в одному з трьох станів: Down (з'являється), Move (рухається) та Up (зникає). Віджети організовані у дерево, де нові дані введення спочатку надходять до кореневого віджета, а

потім передаються до дочірніх віджетів. Розробка додатків з Kivy проста і дозволяє швидко створювати ефективні додатки.

Kivy підтримує всі поширені ОС і пропонує принцип «напиши код один раз і запускай на будь-якій платформі». Він використовує Python і має власну мову розмітки Kv для створення інтерфейсів. Незважаючи на те, що дизайн віджетів у Kivy може бути спрощеним, а інформаційна підтримка не завжди достатньою, цей фреймворк залишається популярним серед розробників завдяки своїй гнучкості та можливості створювати унікальні інтерфейси.[20].

Зважаючи на те, що мінімалізм у дизайні є одним із ключових підходів в сучасній епохі, варто розглянути його коріння та роль у цифровому середовищі. Історія мінімалізму свідчить про те, що цей дизайн не є новим трендом, але має глибокі корені в різних культурах та архітектурних рухах.

Перший аспект стосується походження мінімалізму, яке можна відстежити в традиційній японській культурі та архітектурних рухах, таких як Bauhaus. Він базується на ідеї "менше – краще", спрямованій на зосередження уваги на суттєвих елементах та вилучення зайвого.

У цифровому дизайні мінімалізм зосереджується на спрощенні та підвищенні ефективності передачі інформації. Застосовуючи мінімалістичний підхід, дизайнери можуть створювати інтуїтивно зрозумілі і цілеспрямовані інтерфейси для користувачів, які дозволяють швидко сприймати інформацію та взаємодіяти з продуктом.

Ключові принципи мінімалістичного дизайну полягають у залишенні лише найнеобхідніших елементів, використанні негативного простору та фокусуванні на одній концепції. Візуально мінімалізм характеризується обмеженою кольоровою палітрою, простими текстурами та драматичною типографією.

Найкращі практики мінімалізму включають ретельний аналіз контенту та елементів інтерфейсу, створення єдиної точки фокусування та використання колірної палітри та шрифтів для створення послідовності та орієнтації для користувачів. Крім того, функціональна анімація та яскраві зображення можуть

доповнити мінімалістичний дизайн, роблячи його більш привабливим та інтуїтивно зрозумілим.

Буде створено продукт, в якому легко орієнтуватися та який приваблює більше нових користувачів, відкидаючи непотрібний вміст і функції, щоб зробити додаток більш простим та зручним для користувача [21].

РОЗДІЛ 3
РОЗРОБКА ДОДАТКУ Е-БІБЛІОТЕКИ

3.1 Постановка завдання та опис функціоналу

Розроблено простий мобільний додаток, який допоможе організувати та керувати своїми колекціями паперових книг. Функціонал додатку включає набір функцій: додавання нової екземплярів книги до списку, список користувача з основною інформацією про книги і пошук по списку за назвою або автором.

Головне вікно додатку було створене за допомогою макету FloatLayout з фреймворку Kivy [20]. Використання цього макету дозволяє розміщувати елементи інтерфейсу відносно розмірів екрану пристрою, на якому запущено додаток. Основна частина інтерфейсу має вертикальну орієнтацію розміщення віджетів. Для навігації по додатку використано віджет MDToolbar з бібліотеки KivyMD [22]. Ця панель містить дві кнопки: ліва кнопка викликає діалогове вікно для введення інформації про нову книгу, а права відкриває додаткове меню з пунктами для пошуку книг, зміни теми та оновлення списку книг (рис 3.1). Назва панелі містить текст, який відображає кількість книг у бібліотеці. Основна область додатку представлена віджетом MDScrollView, який дозволяє прокручувати зміст вікна. Всередині цього віджету розташовано макет MDBoxLayout, який містить список книг. Використання вкладених макетів MDBoxLayout допомагає уникнути помилок та полегшує дизайн інтерфейсу.

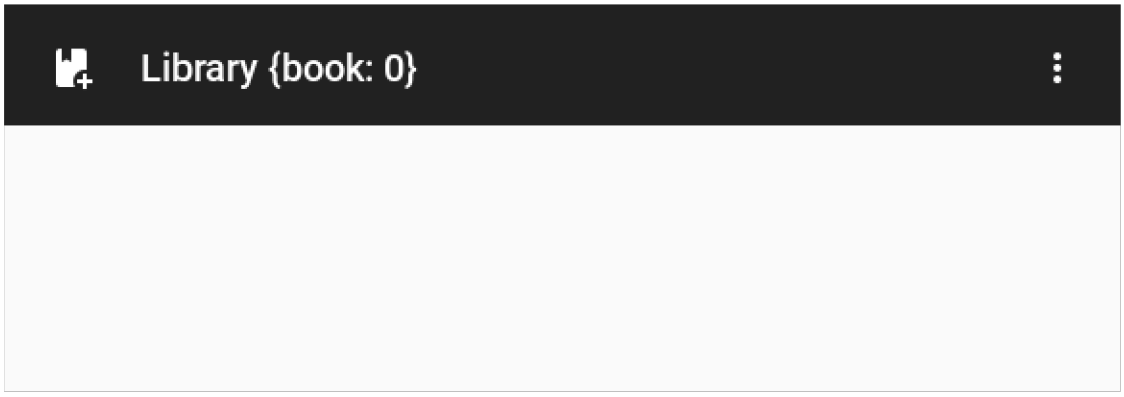

Рисунок 3.1 – Навігаційна панель

При натисканні на ліву кнопку навігаційної панелі відкривається діалогове вікно для введення інформації про нову книгу (рис 3.2). Це вікно реалізоване за допомогою класу ContentDial, який наслідує BoxLayout. Вікно містить поля для введення назви книги, автора, року видання, кількості сторінок, видавництва та ISBN. Також вікно містить дві кнопки: одна для закриття вікна, а інша для підтвердження введених даних.

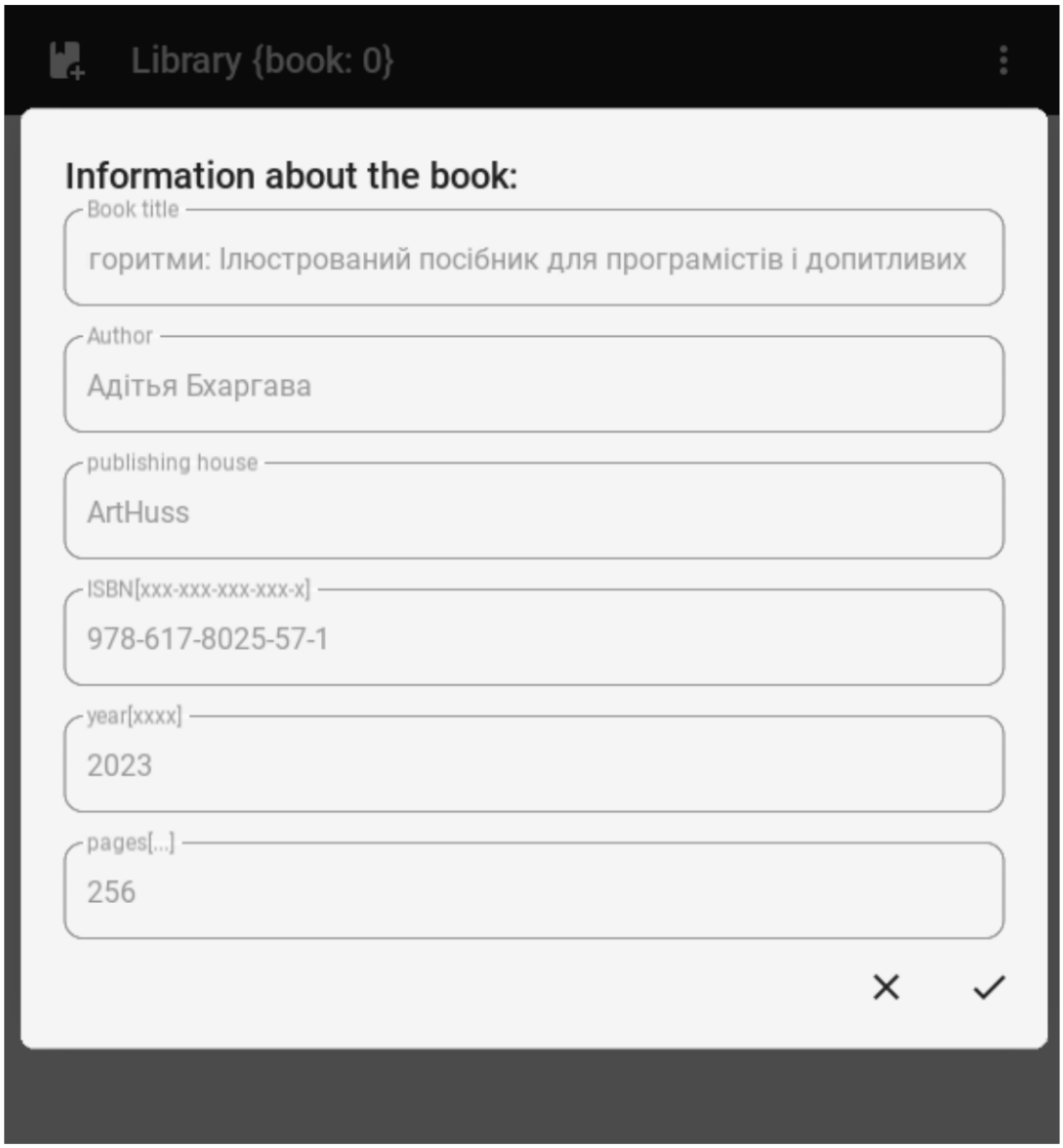

Рисунок 3.2 – Вікно для воду даних про книгу

Список книг реалізований за допомогою віджетів Book та MDExpansionPanel. Кожен елемент списку представляє окрему книгу і містить інформацію про назву книги, автора, рік видання та кількість прочитаних

сторінок. Віджет Book наслідує клас ThreeLineAvatarIconListItem, що дозволяє виводити три рядки інформації та розташовувати додаткові віджети з обох боків елемента. Функція on_add_book виконує додавання нової книги до бази даних та її виведення на екран (рис 3.3). Спершу встановлюється з'єднання з базою даних database.db та створюється курсор для виконання SQL-запитів. Далі виконується SQL-запит для додавання нової книги до таблиці library, де вставляються значення назви книги, автора, кількості сторінок та кількості прочитаних сторінок (яка за замовчуванням встановлюється на 0). Після цього зміни зберігаються у базі даних, з'єднання з базою закривається. Потім викликаються функції update_title для оновлення кількості книг у заголовку відповідно.

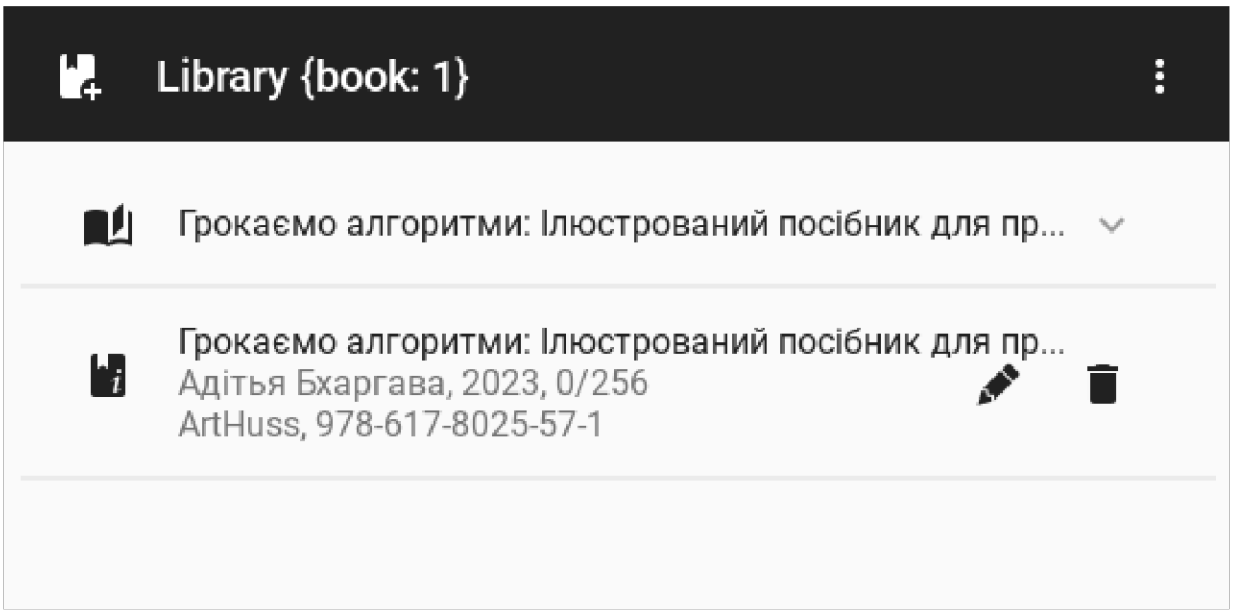

Рисунок 3.3 – Головна сторінка з екземпляром книги

3.2 Реалізація основного функціоналу додатку

Для зберігання інформації про книги використовується реляційна база даних SQLite. Операції включають додавання нових книг, оновлення даних про книги (наприклад, кількість прочитаних сторінок) та видалення книг з бази даних. Всі ці операції виконуються за допомогою SQL-запитів, які

відправляються до бази даних через відповідні функції у коді додатку. База даних містить одну таблицю з назвою library, яка зберігає всі необхідні дані про книги. Таблиця library створюється за допомогою SQL-запиту (лістинг 3.1) з наступними полями:

– id: цілочисельне поле, яке автоматично інкрементується. Використовується як унікальний ідентифікатор для кожного запису в таблиці. Це поле є первинним ключем.

– book_title: текстове поле для зберігання назви книги.

– author: текстове поле для зберігання імені автора книги.

– year: цілочисельне поле для зберігання року видання книги.

– pages: цілочисельне поле для зберігання загальної кількості сторінок у книзі.

– pages_read: цілочисельне поле для зберігання кількості прочитаних сторінок. За замовчуванням встановлено значення 0.

– pub_house: текстове поле для зберігання назви видавництва.

– isbn: текстове поле для зберігання ISBN (Міжнародного стандартного номера книги).

Лістинг 3.1 – Код для створення таблиці

```
query = """CREATE TABLE IF NOT EXISTS library (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            book_title TEXT,
            author TEXT,
            year INTEGER,
            pages INTEGER,
            pages_read INTEGER DEFAULT 0,
            pub_house TEXT,
            isbn TEXT
        )"""
```

Кінець лістингу 3.1

Для введення кількості прочитаних сторінок використовується діалогове вікно (рис 3.4), яке відкривається при натисканні на іконку олівця у списку книг. Це вікно реалізоване за допомогою класу PageInputDialog, який наслідує MDBoxLayout. Вікно містить поле для введення кількості прочитаних сторінок та дві кнопки: одна для закриття вікна, а інша для підтвердження введених даних.

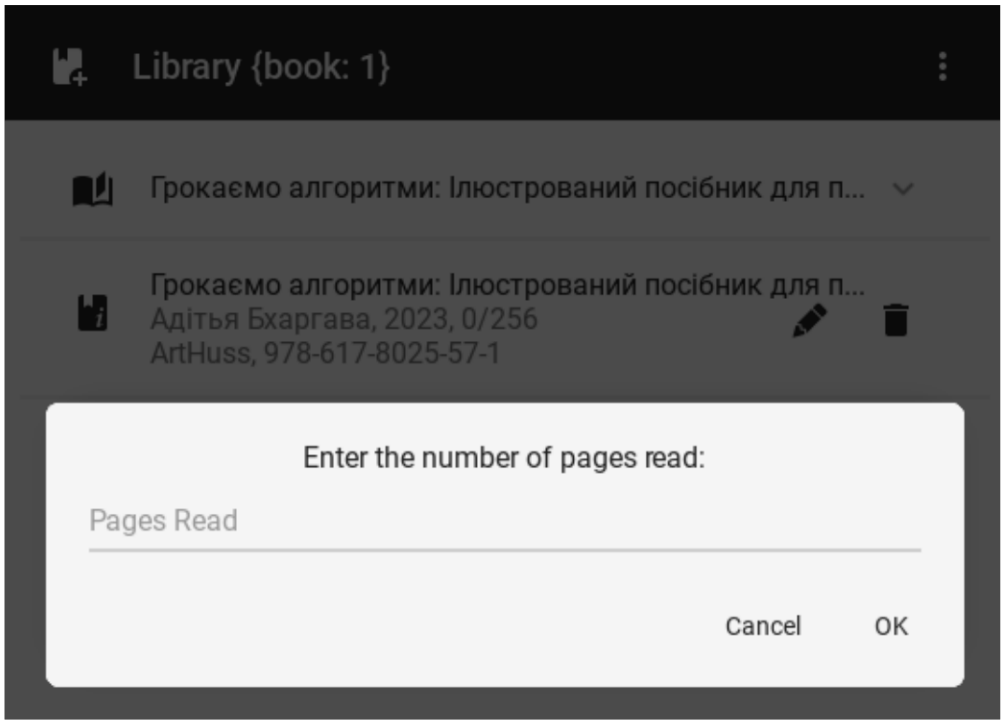

Рисунок 3.4 – Вікно для воду прочитаних сторінок

Функція update_pages (лістинг 3.2) використовується для оновлення кількості прочитаних сторінок книги як в інтерфейсі користувача, так і в базі даних. Ця функція відіграє важливу роль у відстеженні прогресу читання та надає користувачам чітку інформацію про те, скільки сторінок вони прочитали з кожної книги. Параметр pages_read приймає значення типу int, яке являє собою кількість сторінок, прочитаних користувачем з даної книги. Це значення вводиться користувачем вручну. Функція отримує доступ до тексту, який міститься в полі secondary_text. Поле містить інформацію про книгу, автора, рік та кількість сторінок. Текст у полі розділяється на окремі елементи за допомогою методу split. Цей метод розбиває текст на список рядків, де кожен

рядок являє собою окремий елемент інформації про книгу. Розділений текст зберігається у змінній, book_info. Функція update_pages оновлює значення book_info[-1], щоб відобразити нову кількість прочитаних сторінок, отриману з параметра pages_read. Оновлений текст book_info збирається та записується назад до поля secondary_text. Це гарантує, що інтерфейс користувача відображає актуальну інформацію про кількість прочитаних сторінок. Функція update_pages підключається до бази даних, де зберігається інформація про книги та прогрес читання користувачів. Створюється SQL-запит для оновлення кількості прочитаних сторінок для відповідного запису в базі даних. Цей запит використовує назву книги (self.text) як ключ для пошуку запису, а також нове значення pages_read, отримане з параметра функції. SQL-запит виконується, що призводить до оновлення кількості прочитаних сторінок у базі даних. З'єднання з базою даних закривається, щоб гарантувати збереження внесених змін.

Для підтвердження видалення книги зі списку використовується діалогове вікно (рис 3.5), яке відкривається при натисканні на іконку смітника. Це вікно містить текст з підтвердженням та дві кнопки: одна для скасування дії, а інша для підтвердження видалення.

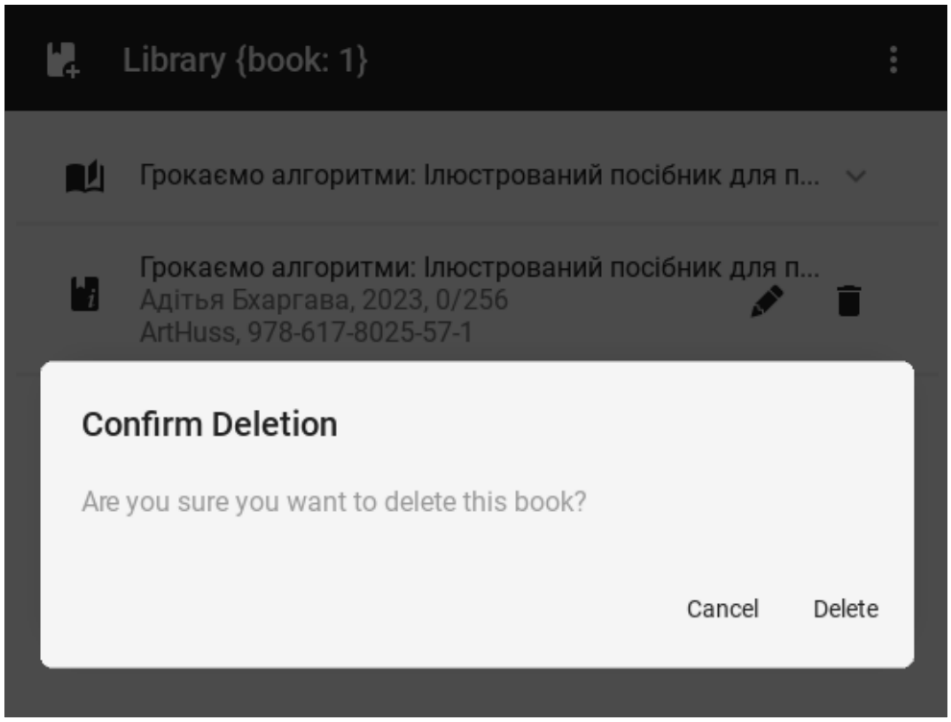

Рисунок 3.5 – Вікно для підтвердження видалення книги

Функція delete_book (лістинг 3.2) відповідає за видалення книги з бази даних та з головного екрана додатку. Відкривається з'єднання з базою даних за допомогою sqlite3.connect, створюється курсор для виконання SQL-запитів. Виконується SQL-запит для видалення запису про книгу з таблиці library, використовуючи назву книги (self.text) для пошуку відповідного запису. З'єднання з базою даних закривається. Функція отримує батьківський елемент для віджету книги (parent_expansion_panel). Видаляє цей елемент з його батьківського контейнера, що призводить до видалення віджету книги з екрану. Зменшується значення лічильника книг у додатку (app.book_count) на одиницю. Викликається функція app.update_title, щоб оновити заголовок панелі інструментів з новою кількістю книг. Таким чином, функція delete_book забезпечує повне видалення книги як з бази даних, так і з інтерфейсу користувача, а також оновлює лічильник книг у додатку.

Лістинг 3.2 – Функції класу Book

```
def update_pages(self, pages_read):
    book_info = self.secondary_text.split(', ')
    total_pages = int(book_info[-1].split('/')[1])
    self.secondary_text = f"{', '.join(book_info[:-1])}, {pages_read}/{total_pages}"
    with sqlite3.connect("database.db") as db:
        cursor = db.cursor()
        query = """UPDATE library SET pages_read = ? WHERE book_title = ?"""
        data = (pages_read, self.text)
        cursor.execute(query, data)
        db.commit()

1 usage
def delete_book(self, *args):
    with sqlite3.connect("database.db") as db:
        cursor = db.cursor()
        query = """DELETE FROM library WHERE book_title = ?"""
        cursor.execute(query, __parameters: (self.text,))
        db.commit()

    parent_expansion_panel = self.parent
    parent_expansion_panel.parent.remove_widget(parent_expansion_panel)

    app = MDApp.get_running_app()
    app.book_count -= 1
    app.update_title()
```

Кінець лістингу 3.2

Додаткове меню містить три пункти: оновлення списку книг, зміна теми та пошук книг (рис 3.6). Оновлення списку очищує поточний список книг на екрані та викликає функцію для завантаження книг з бази даних. Зміна теми дозволяє переключатися між світлою та темною темами інтерфейсу. Пошук книг відкриває діалогове вікно для введення пошукового запиту.

Функція load_books_from_database завантажує дані про книги з бази даних і відображає їх на екрані. Встановлюється з'єднання з базою даних database .db та створюється курсор для виконання SQL-запитів. Виконується запит для отримання всіх записів з таблиці library. Отримані результати зберігаються у змінну. Після цього з'єднання з базою даних закривається. Поточний список книг на екрані очищується для підготовки місця для оновленого списку. Для кожної книги зі списку створюється віджет Book, який додається до списку на екрані.

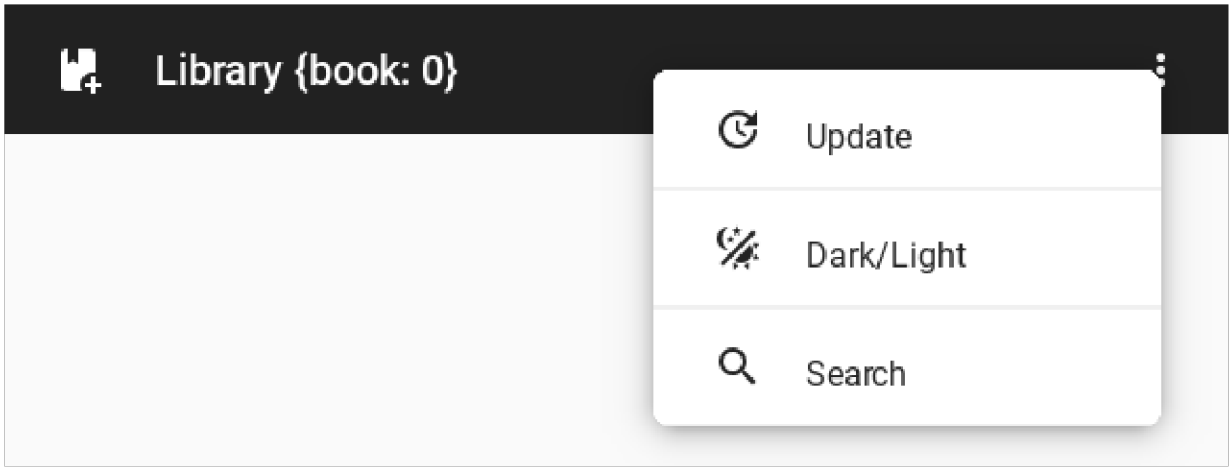

Рисунок 3.6 – Додаткове меню

Додаткове меню на навігаційній панелі містить пункт для пошуку книг. При виборі цього пункту відкривається діалогове вікно для введення запиту на пошук за назвою або автором книги (рис 3.7). Це вікно реалізоване за допомогою класу SearchDialogContent, який наслідує BoxLayout. Вікно містить поле для введення пошукового запиту та дві кнопки: одна для закриття вікна, а інша для підтвердження пошуку.

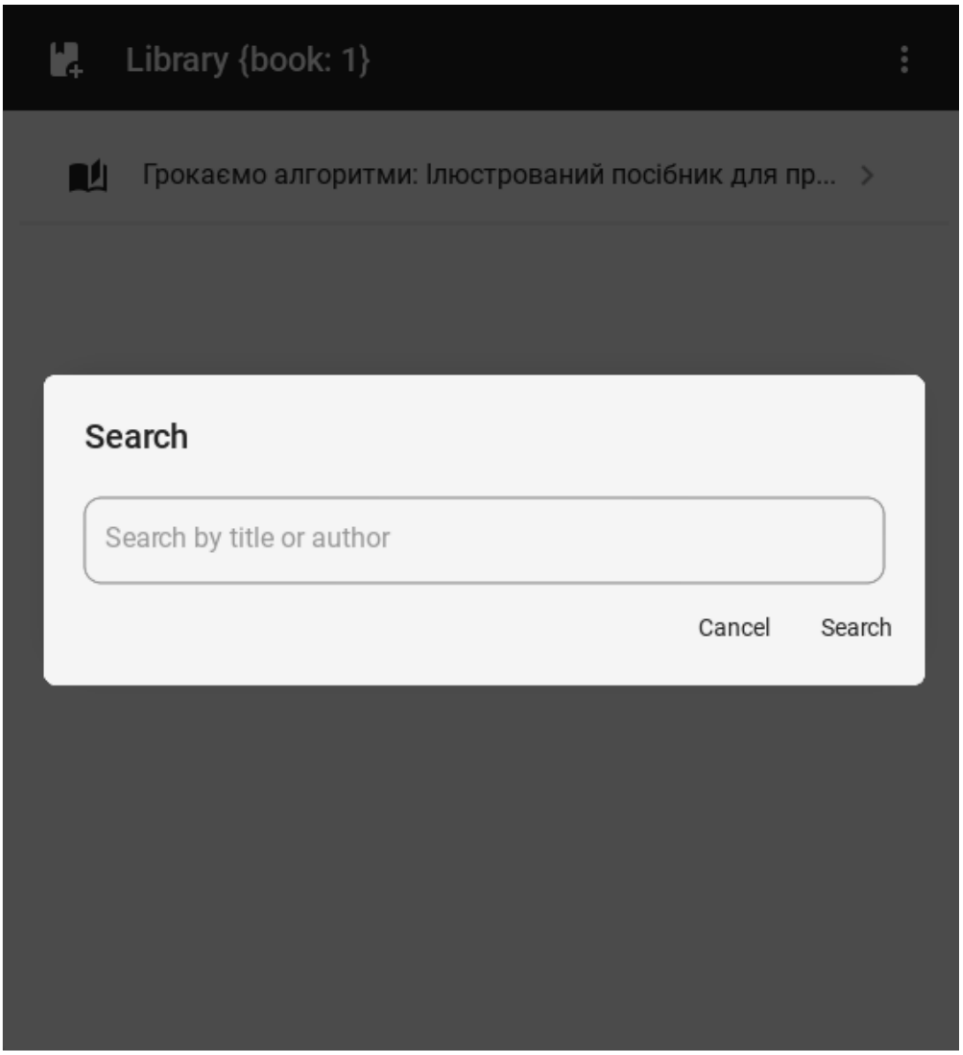

Рисунок 3.7 – Вікно пошуку

Функція perform_search виконує пошук книг за назвою або автором і відображає результати на екрані. Спершу встановлюється з'єднання з базою даних database .db та створюється курсор для виконання SQL-запитів. Виконується запит для пошуку книг, назва або автор яких містить введений користувачем запит. Отримані результати зберігаються у змінну. Після цього з'єднання з базою даних закривається. Поточний список книг на екрані очищується для підготовки місця для відображення результатів пошуку. Для кожної знайденої книги зі списку створюється віджет Book, який додається до списку на екрані. Заголовок навігаційної панелі оновлюється функцією update_title, яка відображає кількість книг у списку. Ця функція викликається

при додаванні нової книги, видаленні книги та кожного разу при запуску додатку.

3.3 Упаковка проекту під мобільні платформи

Buildozer – це інструмент, який використовується для упаковки готового проекту в якості додатку Android. Він автоматизує процес побудови Android-додатку, використовуючи ряд залежностей. Цей інструмент готує середовище з усіма необхідними вимогами для успішної побудови додатку. Ці вимоги включають python-for-android, Android SDK, NDK та інші.

Файл Python буде використовуватися без змін в додатку Android. Але також є ще один важливий файл – buildozer.spec (лістинг 3.3), який необхідний для побудови додатку. Цей файл містить інформацію про додаток Android, таку як назва та версія. Файл buildozer.spec можна згенерувати за допомогою інструмента Buildozer.

Після підготовки всіх файлів, необхідних для побудови додатку Android, можна почати його побудову. Всередині каталогу додатку він може бути зібраний за допомогою наступної команди: buildozer android release. Після виконання команди файл APK буде знайдено в каталозі проекту. Файл APK можна перенести на пристрій Android для його запуску. Також можливо підключити пристрій Android до машини, зібрати, розгорнути та запустити додаток за допомогою однієї команди, яка виглядає так: buildozer android release deploy run [23].

Упакований Kivy-додаток для iOS. Спочатку встановили Xcode та необхідні бібліотеки за допомогою brew, а потім за допомогою pip встановили Cython для інтеграції Python з iOS. Наступним кроком стало завантаження Kivy-ios - інструменту, який перетворив наш Python код на проект Xcode. Під час налаштування Kivy-ios ми переконались, що шлях до використаної версії Python вказано правильно. Після успішного налаштування Kivy-ios ми створили проект Xcode, не забувши назвати головний файл Python main.py. Тепер у нас

з'явилася нова папка з проектом Xcode, готовим до використання. Останнім кроком стала настройка Xcode для роботи з нашим додатком. Варто зазначити, що Kivy-ios копіює всі файли проекту в Xcode, що може бути незручно [20].

Лістинг 3.3 – Вміст файлу buildozer.spec

```
[app]
# (str) Title of your application
title = E-Library

# (str) Package name
package.name = e_library

# (str) Package domain (needed for android/ios packaging)
package.domain = org.e_library

# (str) Source code where the main.py live
source.dir = .

# (list) Source files to include (let empty to include all the files)
source.include_exts = py,png,jpg,kv,atlas

# (str) Application versioning (method 1)
version = 0.5

# (list) Application requirements
requirements = python, kivy, kivymd

# (str) Supported orientation (one of landscape, portrait or all)
orientation = all

# OS X specific
osx.python_version = 3
osx.kivy_version = 2.0.0
osx.kivymd_version = 1.2.0


# (bool) Indicate if the application should be fullscreen or not
fullscreen = 0
```

Кінець лістингу 3.3

ВИСНОВКИ

В ході роботи над проектом було виконано наступні завдання: проаналізовано історію розвитку мобільних додатків та їх вплив на різні аспекти життя людей. Вивчено актуальність мобільних додатків, їх типи та особливості.

Обґрунтовано вибір мови програмування Python для розробки мобільного додатку. Описано фреймворки Kivy та KivyMD, які використовувалися для створення крос-платформного інтерфейсу користувача.

Розроблено простий та зрозумілий інтерфейс користувача, який відповідає принципам мінімалізму. Використано кольорову гаму та шрифти, які приємні для сприйняття. Забезпечено зручну навігацію по додатку.

Створено базовий дизайн мобільного додатку з використанням Float Layout, MDTopAppBar та MDScrollView. Розроблено клас Book для представлення інформації про книги. Реалізовано функціонал додавання нових книг, оновлення списку книг та видалення книг. Додано можливість пошуку книг за назвою або автором. Здійснено інтеграцію з базою даних sqlite3 для зберігання інформації про книги.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Uployal. Все про мобільні застосунки: цифровий світ покупця. *LinkedIn*. URL: https://www.linkedin.com/pulse/все-про-мобільні-застосунки-цифровий-світ-покупця-uployal-3eh8e/ (дата звернення: 17.02.2024).

2. Reads V. E-books vs. paper books: the future of reading. *Medium*. URL: https://medium.com/@valuereads/e-books-vs-paper-books-the-future-of-reading-69eca70991ec (дата звернення: 19.02.2024).

3. Електронний репозитарій ДВНЗ "УжНУ": Історія розвитку мобільних додатків та їх взаємозв'язок із туристичною індустрією. URL: https://dspace.uzhnu.edu.ua/jspui/handle/lib/25531 (дата звернення: 20.02.2024).

4. CBSE class 11 mobile operating systems - symbian, android and ios — geeksforgeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/cbse-class-11-mobile-operating-systems-symbian-android-and-ios/ (дата звернення: 24.02.2024).

5. Ключові тренди у мобільній розробці на 2024 рік Wezom. *Wezom*. URL: https://wezom.com.ua/ua/blog/trendi-mobilnoyi-rozrobki-na-2023-rik (дата звернення: 11.03.2024).

6. Apbundle Technologies. The role of mobile applications in modern society and their impact on productivity and communication. *LinkedIn*. URL: https://www.linkedin.com/pulse/role-mobile-applications-modern-society-impact-productivity- (дата звернення: 28.02.2024).

7. Weburban. The Impact of Mobile Apps on Everyday Life. *LinkedIn*. URL: https://www.linkedin.com/pulse/impact-mobile-apps-everyday-life-weburban (дата звернення: 03.03.2024).

8. Розробка мобільних додатків від А до Я: повний гайд. *DAN IT*. URL: https://dan-it.com.ua/uk/blog/rozrobka-mobilnih-dodatkiv-vid-a-do-ja-povnij-gajd/ (дата звернення: 07.03.2024).

9. Типи мобільних додатків. Онлайн-курси від компанії QATestLab Головна сторінка. URL: https://training.qatestlab.com/blog/technical-articles/types-of-mobile-applications/ (дата звернення: 18.03.2024).

10. Application development life cycle (phases and management models). *The Couchbase Blog*. URL: https://www.couchbase.com/blog/application-development-life-cycle/(дата звернення:19.03.2024).

11. Kissflow T. Application development lifecycle: phases, steps and process explained. *Kissflow Low-Code Application Development Platform*. URL: https://kissflow.com/application-development/application-development-lifecycle/ (дата звернення: 20.03.2024).

12. Які технології використовуються для розробки мобільних додатків – Smart IT Solutions. Smart IT Solutions – Послуги з розробки програмного забезпечення. URL: https://smart-solutions.com.ua/archives/632 (дата звернення: 22.03.2024).

13. What is flutter? - flutter app explained - AWS. *Amazon Web Services, Inc.* URL: https://aws.amazon.com/what-is/flutter/?nc1=h_ls (дата звернення: 24.03.2024).

14. Learning react native. *O'Reilly Online Learning.* URL: https://www.oreilly.com/library/view/learning-react-native/9781491929049/ch01.html (дата звернення: 25.03.2024).

15. Дизайн мобільних додатків: етапи створення та поради Wezom. *Wezom*. URL: https://wezom.com.ua/ua/blog/dizajn-mobilnyh-prilozhenij-pochemu-on-vazhen-i-gde-zakazat (дата звернення: 27.03.2024).

16. Паперові книги залишаються популярнішими за електронні навіть під час пандемії – дослідження. *hromadske.ua*. URL: https://hromadske.ua/posts/paperovi-knigi-zalishayutsya-populyarnishimi-za-elektronni-navit-pid-chas-pandemiyi-doslidzhennya (дата звернення: 27.03.2024).

17. В чому ж успіх та переваги друкованої книги? - Unisoft. *Unisoft*. URL: https://digital.unisoft.ua/v-chomu-zh-uspih-ta-perevagy-drukovanoyi-knygy/#:~:text=Легше%20сприйняття%20тексту:%20Багато%20людей,інформацію%20і%20легко%20робити%20помітки. (дата звернення: 29.03.2024).

18. Створення мобільного додатка на Python – Wezom. *Wezom*. URL: https://wezom.com.ua/ua/blog/sozdanie-mobilnogo-prilozhenija-na-python (дата звернення: 30.03.2024).

19. Реляційні бази даних: структура та застосування у практиці. *FoxmindEd*. URL: https://foxminded.ua/reliatsiini-bazy-danykh/ (дата звернення: 30.03.2024).

20. Welcome to Kivy – Kivy 2.3.0 documentation. Kivy: Cross-platform Python Framework for GUI apps Development. URL: https://kivy.org/doc/stable/ (дата звернення: 31.03.2024).

21. Мам! Я моушн-дизайнер. Що таке мінімалістичний дизайн і як його застосувати. *Друкарня*. URL: https://drukarnia.com.ua/articles/sho-take-minimalistichnii-dizain-i-yak-iogo-zastosuvati-cxDDo (дата звернення: 10.05.2024).

22. Components - KivyMD 1.1.1 documentation. KivyMD 2.0.1.dev0 documentation. URL: https://kivymd.readthedocs.io/en/1.1.1/components/ (дата звернення: 17.05.2024).

23. Ahmed Gad. Python for Android: Start Building Kivy Cross-Platform Applications. *Medium*. URL: https://towardsdatascience.com/python-for-android-start-building-kivy-cross-platform-applications-6cf867d44612 (дата звернення: 18.05.2024).

ДОДАТКИ

Додаток А
Код додатку

```
import sqlite3

from kivy.lang import Builder
from kivy.metrics import dp
from kivy.properties import StringProperty
from kivy.uix.boxlayout import BoxLayout

from kivymd.app import MDApp
from kivymd.uix.boxlayout import MDBoxLayout
from kivymd.uix.button import MDIconButton, MDFlatButton
from kivymd.uix.dialog import MDDialog
from kivymd.uix.expansionpanel import MDExpansionPanel, MDExpansionPanelOneLine
from kivymd.uix.list import IRightBodyTouch, OneLineIconListItem,
ThreeLineAvatarIconListItem
from kivymd.uix.menu import MDDropdownMenu

KV = '''
<Book>
    on_size:
        self.ids._right_container.width = container.width
        self.ids._right_container.x = container.width

    IconLeftWidget:
        icon: "book-information-variant"

    YourContainer:
        id: container

        MDIconButton:
            icon: "lead-pencil"
            on_release: root.show_page_dialog()

        MDIconButton:
            icon: "delete"
            on_release: root.show_confirmation_dialog()

<PageInputDialog>:
    size_hint_y: None
    orientation: "vertical"
    spacing: "5dp"
    height: "50dp"

    MDLabel:
        text: "Enter the number of pages read:"
        halign: "center"

    MDTextField:
        id: pages_input
        hint_text: "Pages Read"
        input_filter: "int"
        multiline: False
```

```
<ContentDial>
    size_hint_y: None
    spacing: "10dp"
    height: "380dp"
    orientation: "vertical"

    MDTextField:
        id: book_title
        hint_text: "Book title"
        mode: "rectangle"

    MDTextField:
        id: author
        hint_text: "Author"
        mode: "rectangle"

    MDTextField:
        id: pub_house
        hint_text: "publishing house"
        mode: "rectangle"

    MDTextField:
        id: isbn
        hint_text: "ISBN[xxx-xxx-xxx-xxx-x]"
        mode: "rectangle"

    MDTextField:
        id: year
        hint_text: "year[xxxx]"
        input_filter: "int"
        mode: "rectangle"

    MDTextField:
        id: pages
        hint_text: "pages[...]"
        input_filter: "int"
        mode: "rectangle"

<IconListItem>
    IconLeftWidget:
        icon: root.icon

<SearchDialogContent>:
    orientation: "vertical"
    spacing: "10dp"
    size_hint_y: None
    height: "50dp"

    MDTextField:
        id: search_input
        hint_text: "Search by title or author"
        mode: "rectangle"

FloatLayout:
    orientation: "vertical"
    adaptive_height: True
```

```
    MDTopAppBar:
        id: toolbar
        title: "Library"
        anchor_title: 'left'
        type_height: "small"
        elevation: 0
        md_bg_color: "#212121"
        left_action_items: [["book-plus", lambda x:
app.show_confirmation_dialog()]]
        right_action_items: [["dots-vertical", lambda x: app.callback(x)]]
        pos_hint: {"top": 1}

    MDBanner:
        id: banner
        type: "three-line"
        over_widget: scroll

    MDScrollView:
        id: scroll
        height: Window.height - toolbar.height
        size_hint_y: None

        MDBoxLayout:
            id: box_book
            orientation: "vertical"
            adaptive_height: True

            MDBoxLayout:
                id: book_list
                orientation: "vertical"
                padding: 10
                spacing: "5dp"
                adaptive_height: True
'''


class IconListItem(OneLineIconListItem):
    icon = StringProperty()


class Book(ThreeLineAvatarIconListItem):
    confirm_dialog = None
    page_dialog = None

    def show_confirmation_dialog(self):
        self.confirm_dialog = MDDialog(
            title="Confirm Deletion",
            text="Are you sure you want to delete this book?",
            buttons=[
                MDFlatButton(
                    text="Cancel",
                    on_release=lambda *args: self.confirm_dialog.dismiss(),
                ),
                MDFlatButton(
                    text="Delete",
                    on_release=lambda *args: [self.confirm_dialog.dismiss(),
```

```
self.delete_book()],
                ),
            ],
        )
        self.confirm_dialog.open()

    def show_page_dialog(self):
        self.page_dialog = MDDialog(
            type="custom",
            content_cls=PageInputDialog(),
            buttons=[
                MDFlatButton(
                    text="Cancel",
                    on_release=lambda *args: self.page_dialog.dismiss(),
                ),
                MDFlatButton(
                    text="OK",
                    on_release=lambda *args: [self.page_dialog.dismiss(),

self.update_pages(self.page_dialog.content_cls.ids.pages_input.text)],
                ),
            ],
        )
        self.page_dialog.open()

    def update_pages(self, pages_read):
        book_info = self.secondary_text.split(', ')
        total_pages = int(book_info[-1].split('/')[1])
        self.secondary_text = f"{', '.join(book_info[:-1])},
{pages_read}/{total_pages}"
        with sqlite3.connect("database.db") as db:
            cursor = db.cursor()
            query = """UPDATE library SET pages_read = ? WHERE book_title = ?"""
            data = (pages_read, self.text)
            cursor.execute(query, data)
            db.commit()

    def delete_book(self, *args):
        with sqlite3.connect("database.db") as db:
            cursor = db.cursor()
            query = """DELETE FROM library WHERE book_title = ?"""
            cursor.execute(query, (self.text,))
            db.commit()

        parent_expansion_panel = self.parent
        parent_expansion_panel.parent.remove_widget(parent_expansion_panel)

        app = MDApp.get_running_app()
        app.book_count -= 1
        app.update_title()

class PageInputDialog(MDBoxLayout):
    pass

class ContentDial(BoxLayout):
    pass
```

```
class YourContainer(IRightBodyTouch, MDBoxLayout):
    adaptive_width = True

class SearchDialogContent(BoxLayout):
    pass

class Example(MDApp):
    dialog = None
    book_count = 0

    def build(self):
        self.theme_cls.theme_style = "Light"

        menu_items = [
            {
                "viewclass": "IconListItem",
                "text": "Update",
                "icon": "update",
                "height": dp(56),
                "on_release": lambda: self.update_list(),
            },
            {
                "viewclass": "IconListItem",
                "text": "Dark/Light",
                "icon": "theme-light-dark",
                "height": dp(56),
                "on_release": lambda: self.switch_theme_style(),
            },
            {
                "viewclass": "IconListItem",
                "text": "Search",
                "icon": "magnify",
                "height": dp(56),
                "on_release": lambda: self.show_search_dialog(),
            }
        ]
        self.menu = MDDropdownMenu(
            items=menu_items,
        )
        return Builder.load_string(KV)

    def on_start(self):
        self.load_books_from_database()
        self.update_title()

    def update_list(self):
        self.root.ids.book_list.clear_widgets()
        self.load_books_from_database()

    def load_books_from_database(self):
        with sqlite3.connect("database.db") as db:
            cursor = db.cursor()
            cursor.execute("SELECT COUNT(*) FROM library")
            book_count = cursor.fetchone()[0]
            self.book_count = book_count
            self.update_title()
            cursor.execute("SELECT * FROM library")
```

```
            books = cursor.fetchall()
            for book in books:
                new_book = Book()
                new_book.text = book[1]
                new_book.secondary_text = f"{book[2]}, {book[3]},
{book[5]}/{book[4]}"
                new_book.tertiary_text = f"{book[6]}, {book[7]}"

                expansion_panel = MDExpansionPanel(
                    icon="book-open-page-variant",
                    content=new_book,
                    panel_cls=MDExpansionPanelOneLine(
                        text=book[1],
                    )
                )

                self.root.ids.book_list.add_widget(expansion_panel)

        if not books:
            self.show_confirmation_dialog()

    def show_confirmation_dialog(self):
        if not self.dialog:
            self.dialog = MDDialog(
                title="Information about the book:",
                type="custom",
                radius=[7, 7, 7, 7],
                content_cls=ContentDial(),
                buttons=[
                    MDIconButton(
                        icon="close",
                        icon_color="#212121",
                        on_release=lambda *args: self.dialog.dismiss(),
                    ),
                    MDIconButton(
                        icon="check",
                        icon_color="#212121",
                        on_release=lambda *args: [self.dialog.dismiss(),
self.on_add_book()],
                    ),
                ],
            )
        self.dialog.open()

    def callback(self, button):
        self.menu.caller = button
        self.menu.open()

    def menu_callback(self):
        self.menu.dismiss()

    def on_add_book(self):
        with sqlite3.connect("database.db") as db:
            cursor = db.cursor()

            book_title = self.dialog.content_cls.ids.book_title.text
            author = self.dialog.content_cls.ids.author.text
```

```
            year = self.dialog.content_cls.ids.year.text
            pages = self.dialog.content_cls.ids.pages.text
            pub_house = self.dialog.content_cls.ids.pub_house.text
            isbn = self.dialog.content_cls.ids.isbn.text

            query = """INSERT INTO library(book_title, author, year, pages,
pub_house, isbn)
                                    VALUES(?, ?, ?, ?, ?, ?)"""
            data = (book_title, author, year, pages, pub_house, isbn)
            cursor.execute(query, data)

            new_book = Book()
            new_book.text = book_title
            new_book.secondary_text = f"{author}, {year}, 0/{pages}"
            new_book.tertiary_text = f"{pub_house}, {isbn}"

            expansion_panel = MDExpansionPanel(
                icon="book-open-page-variant",
                content=new_book,
                panel_cls=MDExpansionPanelOneLine(
                    text=book_title,
                )
            )

            self.root.ids.book_list.add_widget(expansion_panel)

            self.book_count += 1
            self.update_title()

    def update_title(self):
        self.root.ids.toolbar.title = f"Library {{book: {self.book_count}}}"

    def switch_theme_style(self):
        self.theme_cls.theme_style = (
            "Dark" if self.theme_cls.theme_style == "Light" else "Light"
        )

    def show_search_dialog(self):
        if not self.dialog:
            self.dialog = MDDialog(
                title="Search",
                type="custom",
                content_cls=SearchDialogContent(),
                buttons=[
                    MDFlatButton(
                        text="Cancel",
                        on_release=lambda *args: self.dialog.dismiss(),
                    ),
                    MDFlatButton(
                        text="Search",
                        on_release=lambda *args: [self.dialog.dismiss(),

self.perform_search(self.dialog.content_cls.ids.search_input.text)],
                    ),
                ],
            )
        self.dialog.open()
```

```
    def perform_search(self, query):
        self.root.ids.book_list.clear_widgets()

        with sqlite3.connect("database.db") as db:
            cursor = db.cursor()
            cursor.execute("SELECT * FROM library WHERE book_title LIKE ? OR
author LIKE ?",
                           ('%' + query + '%', '%' + query + '%'))
            books = cursor.fetchall()
            for book in books:
                new_book = Book()
                new_book.text = book[1]
                new_book.secondary_text = f"{book[2]}, {book[3]},
{book[5]}/{book[4]}"
                new_book.tertiary_text = f"{book[6]}, {book[7]}"

                expansion_panel = MDExpansionPanel(
                    icon="book-open-page-variant",
                    content=new_book,
                    panel_cls=MDExpansionPanelOneLine(
                        text=book[1],
                    )
                )

                self.root.ids.book_list.add_widget(expansion_panel)


Example().run()
```