Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Програмна система для обміну кулінарними рецептами. Мобільний додаток
(тема)

Виконав:
студент 4 курсу, групи ПЗПІ-20-9

Захарченко Я.В.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник ст.викл. кафедри ПІ Онищенко К.Г.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ З.В.Дудар
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
Кафедра Програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми освітньо-професійна
Освітня програма Програмна інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Захарченко Ярослав Вікторович
(прізвище, ім'я, по батькові)

1. Тема роботи Програмна система для обміну кулінарними рецептами. Мобільний додаток

Затверджена наказом по університету від 20.05.2024 № 471 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 14.06.2024

3. Вихідні дані до роботи Розробити клієнтську частину програмної системи обміну кулінарних рецептів для платформи Android, мовою програмування Kotlin, використовуючи Android SDK та набір бібліотек Android Jetpack.

4. Перелік питань, що потрібно опрацювати в роботі

Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	08.04.2024	*виконано*
2	Створення специфікації ПЗ	22.04.2024	*виконано*
3	Проектування ПЗ	28.04.2024	*виконано*
4	Розробка ПЗ	20.05.2024	*виконано*
5	Тестування ПЗ	20.05.2024	*виконано*
6	Оформлення пояснювальної записки	04.06.2024	*виконано*
7	Підготовка презентації та доповіді	06.06.2024	*виконано*
8	Попередній захист	09.06.2024	*виконано*
9	Нормоконтроль, рецензування	06.06.2024	*виконано*
10	Здача роботи у електронний архів	10.06.2024	*виконано*
11	Допуск до захисту у зав. кафедри	17.06.2024	*виконано*
12	Захист	17.06.2024	

Дата видачі завдання 08 квітня 2024р.

Студент (ка) _____ Захарченко Я.В.
(підпис)

Керівник роботи _____ ст.викл. кафедри ПІ Онищенко К.Г.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до кваліфікаційної роботи бакалавра, 65 стор., 32 рис., 2 табл., 12 джерел.

КУЛІНАРІЯ, МОБІЛЬНИЙ ЗАСТОСУНОК, ОБМІН, РЕЦЕПТ, ANDROID SDK, KOTLIN, JETPACK COMPOSE

Об’єкт розробки – клієнтська частина програмної системи для обміну кулінарними рецептами.

Мета розробки – створити систему, де люди зможуть обмінюватися кулінарним досвідом, реалізувати інтерфейс для зручного перегляду рецептів та взаємодії з іншими користувачами.

Метод рішення – набір інструментів Android SDK, мова програмування Kotlin, UI бібліотеки Jetpack Compose.

У результаті розробки спроектовано зручний мобільний застосунок, який дозволяє створювати рецепти, обмінюватися ними з іншими людьми, та оцінювати інші.

CULINARY, MOBILE APPLICATION, SHARING, RECIPE, ANDROID SDK, KOTLIN, JETPACK COMPOSE

The object of development is the client part of the software system for sharing culinary recipes.

The goal of development is to create a system where people can share culinary experience, implement an interface for convenient viewing of recipes and interaction with other users.

The solution method is a set of Android SDK tools, the Kotlin programming language, and the UI of the Jetpack Compose library.

As a result of the development, a user-friendly mobile application was designed that allows you to create recipes, share them with other people, and rate others.

Я, Захарченко Ярослав Вікторович, студент гр. ПЗПІ-20-9, здобувач вищої освіти на першому (бакалаврському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Програмна система для обміну кулінарними рецептами. Мобільний додаток», що буде представлена до екзаменаційної комісії для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Усі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови до допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

В світі люди завжди мали потребу в їжі та постійно шукали нові способи комбінування інгредієнтів для поліпшення смаку та корисності страв. Історично кулінарія вимагала багато експериментів та вивчення і для спрощення цього процесу, люди почали обмінюватися рецептами, передавати їх поколіннями, або просто навчати інших людей напряму. У сучасності, з розвитком інформаційних технологій, зацікавленість у кулінарії збільшилася, бо люди з усього світу могли обмінюватися своїм досвідом і не треба було витрачати багато зусиль на приготування чогось нового і цікавого. Існує багато кулінарних блогів, відеоуроків, онлайн-спільнот, сайтів та застосунків які об'єднують любителів готування з усього світу, але у них достатньо обмежена функціональність, що не дозволяє зручно знаходити різні рецепти або ділитися своїми з іншими.

Саме через це створення програмної системи для обміну кулінарними рецептами і зараз є актуальною. Ця система дозволить спростити процеси поширення та перегляду різних рецептів.

Однією з ключових задач системи є створення інтуїтивно зрозумілого та зручного інтерфейсу програмної системи. Користувачі повинні легко знаходити необхідні рецепти, додавати нові та взаємодіяти з іншими учасниками спільноти. Система повинна забезпечувати можливість зберігання, категоризації та організації кулінарних рецептів у зручний спосіб. Це включає в себе можливість пошуку за категоріями, типами страв тощо. Також користувачам має бути надана можливість додавати власні рецепти до системи, розширюючи її базу даних та сприяючи обміну кулінарними ідеями. Система повинна створювати сприятливе середовище для взаємодії між користувачами, наприклад, через можливість коментування та обговорення рецептів, оцінювання, обміну порадами та рекомендаціями.

У висновку, програмна система для обміну кулінарними рецептами не лише полегшить процес пошуку необхідного рецепту, а ще й допоможе людям обмінюватися своїм кулінарним досвідом.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

Кулінарна галузь розвивалась з початком з початку виникнення людства і люди завжди змішуючи різні інгредієнти і різні способи приготування щоб отримати щось смачне, корисне, енергійне, особливе. Для розвитку людини у цій сфері потрібно було багато експериментів і часу, і щоб оптимізувати цей процес люди ділилися своїм досвідом з іншими.

З розвитком технологій стало набагато простіше обмінюватися кулінарним досвідом за допомогою інтернету і відповідних платформ, форумів, застосунків. За допомогою цих інструментів люди з різних куточків світу можуть показувати свої рецепти іншим, а інші спробувати їх відтворити і написати відгук або порадити покращення для цього рецепту.

Популярність кулінарії серед людства показує безліч популярних кулінарних блогів, різних сайтів та застосунків. Вони спрощують обмін досвідом з кулінарії, мотивують людей готувати різні рецепти вдома, що може бути більш економним і корисним варіантом.

Через популярність цієї галузі існує і висока конкурентність серед різноманітних кулінарних застосунків. Однак в більшості застосунків мала кількість інструментів або обмеженість в рецептах через ігнорування соціальної сторони кулінарії, що заважає людям ділитися своїм досвідом. Розглянемо деякі популярні рішення кулінарних застосунків і проаналізуємо їх.

Tasty є одним з найпопулярніших кулінарних застосунків на мобільний телефон, який дозволяє продивлятися багато кулінарних рецептів, оцінювати їх, та зберігати (див. рис. 1.1). Проаналізуємо основні функції та недоліки цього застосунку.

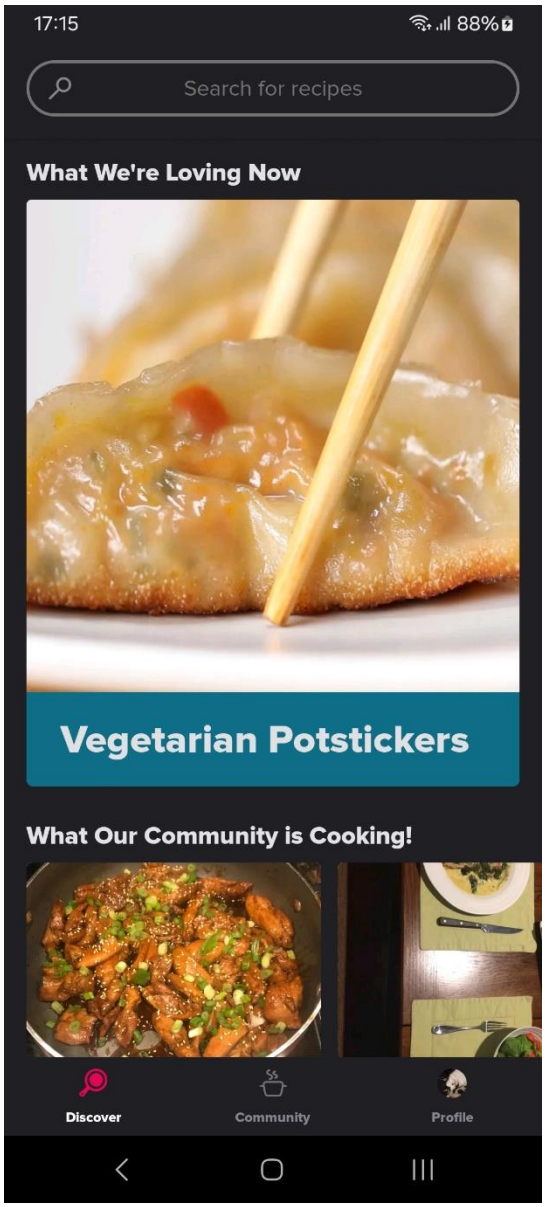

Рисунок 1.1 – Знімок екрану застосунку Tasty (виконано самостійно)

Основні функції:

– пошук потрібних рецептів: користувачі можуть шукати потрібні рецепти за такими параметрами, як назва, категорія, складність, кухня;
– перегляд повної інформації про рецепт: користувач може продивитися повну інформацію про рецепт, а саме відео приготування, інгредієнти, поради від інших людей для рецепту, час приготування і кроки приготування;
– зберігання рецептів: у користувача є можливість зберегти рецепти по різним категоріям і потім за потреби швидко повернутися до них.

Недоліки:

– у застосунка немає можливості створення рецепту і всі рецепти додають адміністратори;
– кроки можуть бути розписані недостатньо детально.

Хоча у Tasty є величезна база рецептів і непоганий функціонал для пошуку, перегляду і зберігання рецептів, він не забезпечує достатнього зворотного відгука від глядача до автора, який створив рецепт, бо рецепти додаються не самим автором.

Cookpad є одним із популярних застосунків для кулінарних рецептів, який має широкий спектр функціоналу (див. рис. 1.2).

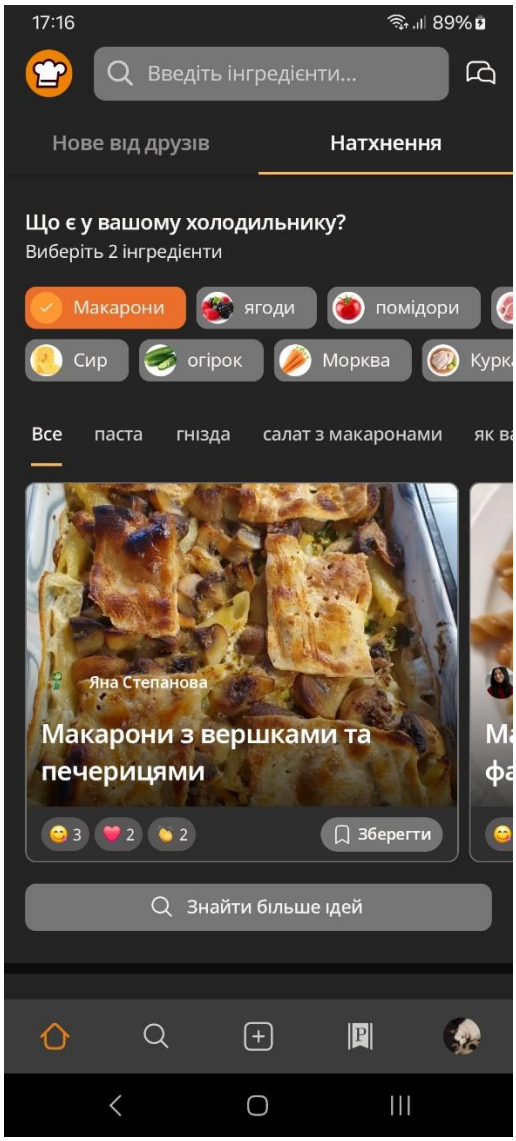

Рисунок 1.2 – Знімок екрану застосунку Cookpad (виконано самостійно)

Серед основних функцій можна виділити наступні:

- пошук потрібних рецептів: користувачі можуть шукати потрібні рецепти за різними параметрами та інгредієнтами;
- перегляд повної інформації про рецепт: користувач може продивитися повну інформацію про рецепт, від потрібних інгредієнтів до кроків приготування;
- зберігання рецептів: користувач може зберегти рецепт який сподобався і швидко його знайти при потребі;
- створення рецептів: є можливість створювати власні рецепти, додавши різні фото, інгредієнти і кроки, і ділитися ними зі спільнотою;
- перегляд статистики: користувач може продивлятися статистику для створених ним рецептів задля розуміння зацікавленості аудиторії в ваших рецептах.

Недоліки:

- інтерфейс може бути трохи перевантажений і інтуїтивно не зрозумілий, особливо інтерфейс пошуку;
- під час створення рецепту не можна додавати відео приготування.

Як можна побачити, більшість застосунків мають свої недоліки в різних аспектах свого використання, що ускладнює процес користування застосунком і отримання належного досвіду, тому існує потреба в розробці програмної системи яка буде об’єднувати увесь потрібний функціонал в зручному для користувача інтерфейсі.

1.2 Виявлення та вирішення проблем

Аналіз предметної галузі показав, що існує багато проблем, які можна вирішити для покращення користувацького досвіду. Система повинна враховувати потреби користувачів, забезпечуючи їм можливість легко знаходити та використовувати необхідні функції, що підвищить їхню задоволеність та залученість. Завдяки цьому, новий додаток зможе стати лідером на ринку

кулінарних застосунків, пропонуючи унікальний та повноцінний користувацький досвід.

У кулінарних додатках взаємодія з іншими користувачами та обмін думками має важливе значення. Відсутність можливості залишати відгуки та коментарі до рецептів позбавляє користувачів можливості спілкування та обговорення. Щоб вирішити цю проблему, можна розглянути створення функціоналу коментування авторських рецептів, де користувачі зможуть ділитися своїми враженнями, порадами та питаннями. Така функція не тільки сприятиме покращенню рецептів через обмін досвідом, але й створить спільноту однодумців, що захоплюються кулінарією. Це допоможе залучити більше користувачів до додатку та підвищить їхню лояльність завдяки активному спілкуванню та підтримці.

Можливість створення власних рецептів є ключовою для користувачів, оскільки вона дає їм можливість виражати свою творчість та поділитися своїми улюбленими стравами з іншими. Відсутність або обмеженість функціоналу у цьому напрямку обмежує користувача та знижує його зацікавленість у застосунку. Додавання інструментів для створення власного рецепту, можливо, включаючи функції додавання інгредієнтів, опису кроків приготування та фотографій, може значно поліпшити користувацький досвід. Крім того, можливість оцінювати та коментувати рецепти інших користувачів сприятиме активному обміну ідеями та створенню спільноти однодумців, що додатково стимулюватиме залученість та активність користувачів.

Інтерфейс застосунку впливає на сприйняття користувачем та його здатність ефективно використовувати його. Складний або незрозумілий інтерфейс може ускладнити користувачам насолоджуватися всіма можливостями застосунку. Вирішити цю проблему можна шляхом розробки інтуїтивного інтерфейсу, який відповідає потребам та очікуванням користувачів. Це може включати в себе використання зрозумілої логіки розташування елементів, чітку навігацію та дружній до користувача дизайн. Використання сучасних принципів дизайну допоможе створити естетично привабливий та функціональний інтерфейс, що значно підвищить задоволеність користувачів і їхню лояльність до застосунку.

В результаті вирішення цих проблем користувачі отримають зручний та цікавий застосунок, який задовольнятиме їх потреби у кулінарних експериментах та спілкуванні. Вони матимуть змогу насолоджуватися персоналізованими рекомендаціями, ділитися своїми власними рецептами та спілкуватися з іншими користувачами, що підвищить загальний рівень задоволення від використання застосунку.

1.3 Постановка задачі

На основі отриманих результатів аналізу та виявлення проблем було виявлено, що сучасний етап розвитку цифрових технологій і мобільних пристроїв характеризується постійним зростанням попиту на зручні та функціональні мобільні застосунки. Враховуючи цей тренд, було поставлено завдання створити зручний мобільний застосунок, який не лише задовольнятиме базові потреби користувачів у пошуку та збереженні кулінарних ідей, але й створюватиме інтерактивну спільноту любителів кулінарії. Ця система має на меті вирішити основні недоліки існуючих програм та забезпечити користувачів зручними й ефективними інструментами для обміну кулінарними рецептами. Щоб краще зрозуміти потреби користувачів, необхідно провести дослідження цільової аудиторії застосунка.

Розуміння, хто саме буде користуватися нашим продуктом, дозволяє краще адаптувати його функціональність, інтерфейс та контент під потреби різних груп користувачів. Це, в свою чергу, сприяє більш ефективному задоволенню запитів та очікувань користувачів, що підвищує загальну привабливість та корисність застосунка. Далі розглянемо детальніше цільову аудиторію нашого продукту. Цільова аудиторія нашого мобільного застосунка для обміну рецептами охоплює широкий спектр користувачів, які мають спільний інтерес до кулінарії та приготування їжі. Основними групами користувачів є:

– домогосподарки та домогосподарі: люди, які регулярно готують їжу для своєї родини і шукають нові рецепти для різноманітності домашнього меню. Їм потрібен легкий доступ до різноманітних рецептів, можливість

зберігати улюблені рецепти, обмінюватися порадами з іншими користувачами;

- молоді професіонали: люди, які нещодавно почали жити самостійно і прагнуть навчитися готувати різні страви. Їм потрібні прості та швидкі рецепти, пошук за інгредієнтами, які є обмеженими для них та опис кроків;
- кулінарні ентузіасти: люди, які захоплюються кулінарією на аматорському або професійному рівні і хочуть ділитися своїми рецептами та досвідом. Їм потрібні можливість публікувати власні рецепти, отримувати фідбек та оцінки від інших користувачів;
- люди з дієтичними обмеженнями: користувачі, які дотримуються певних дієт через здоров'я або етичні міркування. Їм потрібні рецепти, адаптовані до їхніх дієтичних потреб, можливість фільтрувати рецепти за певними критеріями, рекомендації щодо заміни інгредієнтів;
- сім'ї з дітьми: батьки, які шукають рецепти, які сподобаються їхнім дітям, а також залучають дітей до процесу приготування їжі. Їм потрібні прості та здорові рецепти, рецепти для вибагливих їдців, кулінарні ідеї для дітей.

На основі отриманих результатів аналізу та ідентифікації основних проблем, пов'язаних з існуючими програмами для обміну кулінарними рецептами, було визначено ключові вимоги до нашого мобільного застосунка. Аналіз існуючих рішень показав, що користувачі часто стикаються з рядом недоліків, таких як складність інтерфейсу, обмежені можливості для взаємодії між користувачами та недостатня кількість функцій для зручного обміну рецептами. Основні завдання, які необхідно вирішити в рамках розробки програмної системи:

- створити застосунок який буде об'єднувати увесь основний функціонал кулінарного застосунку, щоб отримати комплексне рішення, яке надасть користувачу все необхідне для ефективного використання програми;
- розробити зручну авторизацію задля забезпечення безпеки, збереження даних та персоналізації досвіду кожного користувача;

- розробити зручні інструменти пошуку, які включають в себе пошук по назві та категоріям, а також перегляд інформації про рецепт, в яку входять опис, інгредієнти, кроки приготування та фотографії для забезпечення отримання користувачем повної інформації про рецепт ;
- реалізувати систему оцінювання рецептів та можливість залишити відгуки до них, що дозволить користувачам обмінюватися думками та порадами стосовно рецептів, що покращить взаємозв'язок у спільноті;
- розробити можливість збереження рецепту та оформлення підписки на автора, щоб мати швидкий доступ до улюблених рецептів і авторів, а також покращити персоналізований досвід користувача;
- реалізувати зручний інструмент для створення рецепту, з можливістю вказання усієї необхідної інформації, щоб інші користувачі мали доступ до детальної інформації про рецепт;
- реалізувати інтуїтивно зрозумілий, зручний та привабливий інтерфейс, який забезпечить ефективну взаємодію користувача з застосунком;
- розробити сучасну та масштабовану архітектуру програми, яка дозволить швидко вносити зміни та легко додавати новий функціонал в застосунок за потреби.

Розробка програмної системи для кулінарного застосунку вимагає вирішення численних завдань, спрямованих на створення зручного, функціонального та привабливого продукту для користувачів. Впровадження зручної авторизації, пошукового механізму, персоналізованих рекомендацій, системи оцінювання та відгуків, збереження та підписки на рецепти, зручного інструменту для створення рецептів, а також привабливого та ергономічного інтерфейсу [7] дозволить створити продукт, який забезпечить користувачам повний функціонал та приємний досвід використання. Врахування всіх цих аспектів допоможе створити конкурентоспроможний кулінарний застосунок, який задовольнить потреби та очікування користувачів.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

Клієнтський мобільний додаток є ключовою програмою, яка забезпечує повноцінну взаємодію з користувачем та гарантує йому задоволення від використання системи. Для забезпечення комфортного та приємного користувацького досвіду необхідно оптимізувати програму, ретельно працювати над дизайном та забезпечити високу якість коду для швидкого впровадження нового функціоналу.

2.1 Головні технології та інструменти

Для розробки мобільного застосунку будуть використані наступні технології, бібліотеки та інструменти:

- Android SDK – набір різноманітних інструментів, бібліотек і API, які дозволяють розробникам створювати мобільні додатки для платформи Android і без якого не обходиться ні єдиний застосунок під Android;
- Kotlin – сучасна мова програмування, яка стала офіційною мовою розробки для платформи Android. Kotlin пропонує компактний і зрозумілий синтаксис, безпеку типів і високу продуктивність. Вона інтегрується з Android SDK і надає широкі можливості для розробки додатків;
- Jetpack Compose – декларативна бібліотека для створення користувацьких інтерфейсів в Android за допомогою Kotlin [6]. Вона спрощує процес створення UI, забезпечуючи більш простий та зрозумілий код;
- Jetpack Navigation – компонент бібліотеки Jetpack [2], який допомагає у керуванні навігацією в мобільному додатку. Забезпечує простий та зручний спосіб переходу між різними екранами застосунку;
- Retrofit – бібліотека для роботи з мережевими запитами в Android. Вона дозволяє здійснювати HTTP-запити [8] до сервера і обробляти отримані відповіді. Retrofit спрощує роботу з мережею і дозволяє ефективно взаємодіяти з віддаленими джерелами даних;

– Hilt – бібліотека для реалізації залежностей в Android-додатках з використанням принципів ін'єкції залежностей. Hilt дозволяє забезпечити чистий та модульний код, зменшуючи зв'язаність між компонентами додатку;
– Coil – бібліотека для завантаження та відображення зображень в Android-додатках. Coil дозволяє ефективно обробляти зображення з використанням кешування та асинхронного завантаження, що полегшує розробку додатків, які працюють з великими обсягами мультимедійних даних.

2.2 Архітектура

Для розробки мобільного застосунку буде використовуватися багатомодульна архітектура [4] з використанням Clean Architecture [1], що забезпечить якість коду, швидку збірку проекту та зручну ізоляцію коду. Для шару інтерфейсу буде використовуватися архітектурний патерн MVVM та компонентний підходу написання UI, що забезпечить повторне використання коду. Кожен модуль застосунку буде відповідати за окремий функціонал:

– модуль застосунку – об’єднуватиме функціонал всіх модулів застосунка;
– головний модуль – буде зберігати усі основні функції, стилі, компоненти і класи які будуть використовуватися у всьому застосунку;
– модуль авторизації та реєстрації – відповідатиме за авторизацію та реєстрацію користувача в системі;
– модуль особистого профілю користувача – дозволить користувачам продивлятися особисту інформацію та редагувати її, а також продивлятися свої та збережені рецепти та підписи на авторів;
– модуль рецептів – буде зберігати функціональність пошуку та фільтрації рецептів, показ інформації про них, а також створення особистих рецептів.

2.3 Взаємодія з серверною частиною програмної системи

Мобільне застосування буде взаємодіяти з сервером за допомогою інтерфейсу RESTful API [5], через який застосунок буде отримувати різну інформації про рецепти, особистий профіль або виконувати операцій, такі як створення рецепту, реєстрація та інші. Взаємодія буде будуватися через запити протоколу HTTP та бібліотеку Retrofit. Дані з серверу будуть приходити у JSON-форматі і потім перетворюватися на класи з використанням бібліотеки JSONConverter, для можливості обробки інформації мовою програмування Kotlin. Для забезпечення захисту інформації від зміни сторонніми особами буде використовуватися токен доступу при кожному запиті.

2.4 Захист від помилок та оптимізація застосунку

Для захисту програми від критичних помилок в програмі будуть використовуватися офіційні і перевірені часом бібліотеки, що набагато зменшує можливість виникнення непередбачуваних помилок. Для впровадження залежностей буде використовуватися бібліотека Hilt з генерацією коду під час компіляції, яка має деякі переваги над бібліотеками які впроваджують залежності динамічно під час роботи застосунку, а саме унеможливлює помилки під час роботи застосунку і всі помилки пов’язані з його роботою будуть виникати тільки під час компіляції, а також має продуктивність, так як всі залежності в класах вже описані і їх не треба шукати під час роботи програми. Також використання мови програмування Kotlin може захистити від деяких помилок, які були в Java, бо Kotlin має безпеку типів.

Для оптимізації застосунку необхідно використати використовуватися різноманітні підходи:

– для завантаження фотографій і іконок будуть використовуватися відповідні для Android формати файлів, а також забезпечуватиметься кешування [9] фотографій за допомогою бібліотеки Coil;

- зменшення кількості запитів в інтернет на екрані, що дозволить швидко завантажувати необхідні дані;
- запобігання витіку пам’яті використанням усіх рекомендацій для захисту від них а також постійно перевірка роботи застосунку;
- використання асинхронного коду під час будь-яких операцій які займають багато часу.

Отже, для забезпечення надійності та продуктивності програми передбачається використання перевірених бібліотек і сучасних підходів до розробки. Бібліотека Hilt забезпечить статичну перевірку залежностей під час компіляції, що мінімізує ризик виникнення помилок під час виконання. Використання мови програмування Kotlin підвищить безпеку коду завдяки типобезпеці. Оптимізація програми буде досягнута за рахунок використання ефективних форматів файлів для зображень, кешування за допомогою бібліотеки Coil, зменшення кількості інтернет-запитів, запобігання витоку пам'яті та застосування асинхронних операцій для обробки ресурсомістких завдань. Всі ці заходи сприятимуть стабільній роботі застосунку і забезпечать високий рівень продуктивності.

3 АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 UML проєктування програмного забезпечення

В цьому підрозділі проєктування будуть показані діаграми, які візуалізують поведінку програмної системи для обміну кулінарними рецептами за допомогою UML [3].

Для показу функціональних можливостей і взаємодії користувача із застосунком використана діаграма прецедентів за допомогою UML (див. рис. 3.1). На цій діаграмі зображено між акторами, які є користувачами системи, і прецедентами, які є функціями системи.

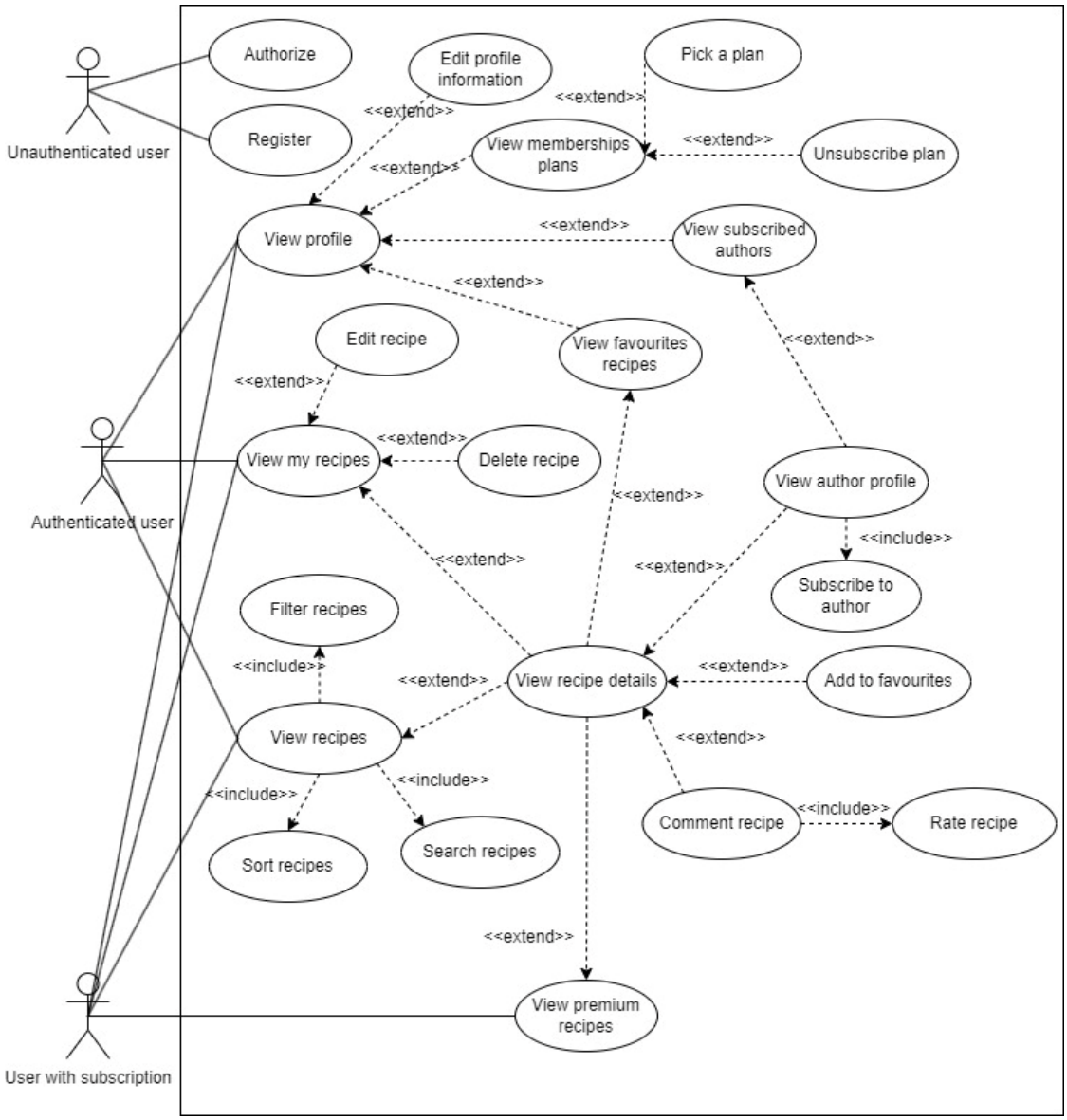

Рисунок 3.1 – Діаграма прецедентів (виконано самостійно)

На діаграмі показано 3 актори, які будуть в застосунку, неавторизований, авторизований користувачі та користувач з підпискою. Неавторизований користувач має дві можливості: зареєструватися в системі, якщо у нього немає акаунту, або авторизуватися, якщо акаунт є.

Авторизований користувач отримує доступ до великого спектру функціоналу, одним з яких є перегляд рецептів. Користувач може шукати потрібні йому рецепти за назвою, сортувати за рейтингом та фільтрувати їх за тегами, що дозволяє знайти найбільш відповідний рецепт. Вони можуть переглядати деталі кожного рецепту, включаючи опис, час приготування, складність, кількість порцій список інгредієнтів, покрокову інструкцію з приготування, фото та інформацію про наявні теги рецепту.

Також користувачі можуть додавати рецепти в список улюблених для швидкого доступу в майбутньому. Окрім цього, вони можуть оцінювати рецепти за допомогою система оцінювання від 1 до 5, залишати коментарі під рецептами, де можна поділитися своїми враженнями, запитаннями або порадами щодо приготування. Користувач також може перейти на сторінку автора рецепту, щоб дізнатися більше про нього, переглянути інші його рецепти та підписатися на оновлення від цього автора.

Авторизованому користувачу відкривається доступ до створення свого рецепту. Він може додати свій власний рецепт, включаючи опис, фото, список інгредієнтів та покрокову інструкцію з приготування, прикріпити потрібні теги. Свої рецепти можна буде редагувати та видаляти за необхідності. В особистій інформації користувач може переглядати збережені рецепти, підписки на авторів, проглядати плани та редагувати інформацію про себе.

Користувачу з оформленою підпискою відкривається можливість проглядати преміум рецепти від шефів-партнерів. Також у нього є можливість дивитися детальну інформацію про цей рецепт і підписуватися на автора цього рецепту.

Загалом діаграма прецедентів показує можливості, які має неавторизований, авторизований користувачі та користувачі з підпискою в застосунку. Це допомагає

побачити повноцінний функціонал застосунку і зробити висновки, що вже є і що можна доробити за потреби.

Для демонстрування можливої послідовності дій користувача в застосунку була розроблена діаграма діяльності (див. рис. 3.2). Вона допоможе краще розуміти дії користувача в системі і прослідковувати їх послідовність.

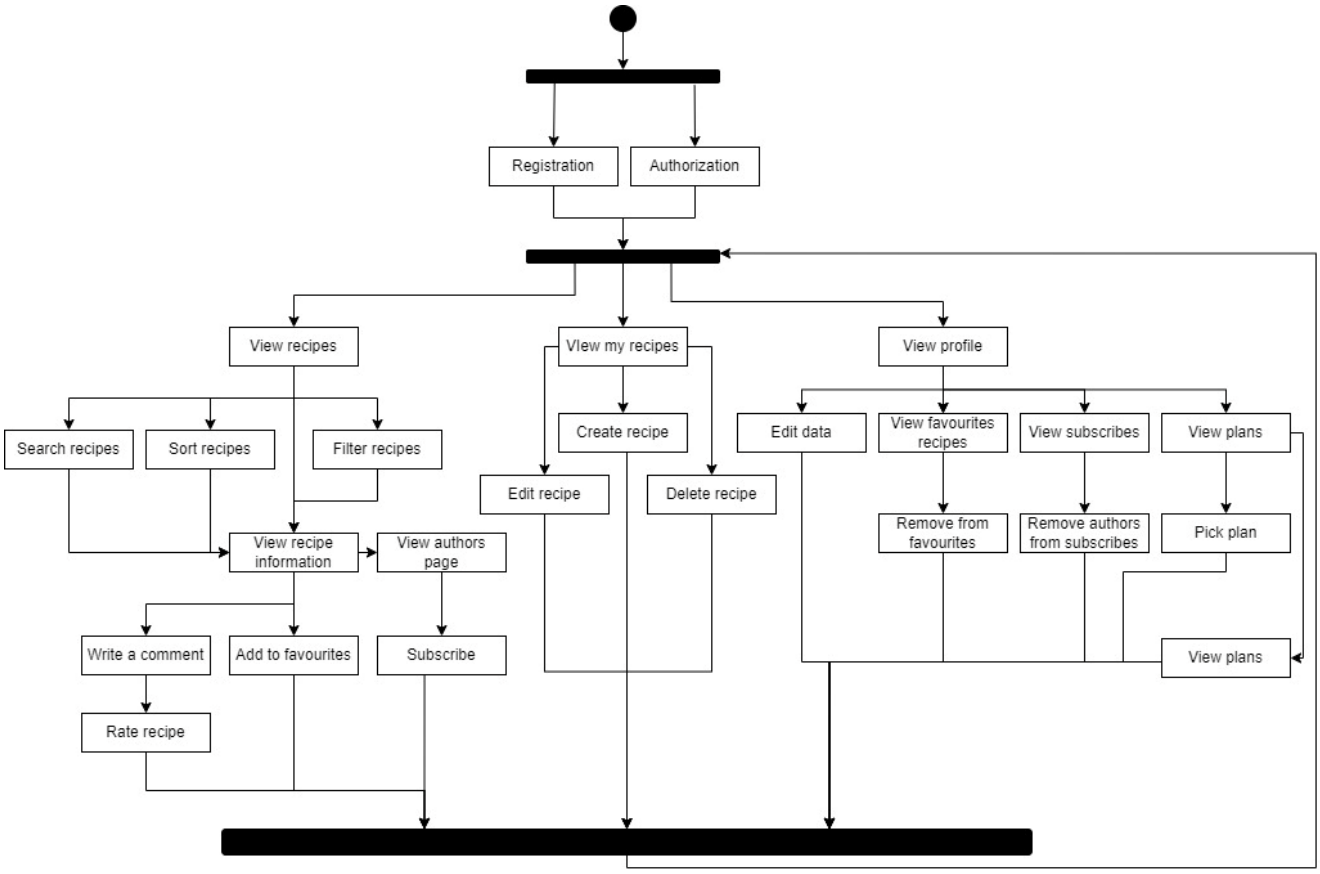

Рисунок 3.2 – Діаграма діяльності (виконано самостійно)

Діаграма починається з точки входу – запуску застосунку, де у користувача є вибір, зареєструватися або авторизуватися, після чого дані про користувача завантажаться і почнеться нова точка входу – головний екран застосунку. З цієї точки користувач може перейти на екран з усіма рецептами, екран профілю та екран зі створеними ним рецептами.

З головного екрану застосунку у користувач може перейти на сторінку з усіма рецептами. На ній є можливість шукати рецепти за назвою, сортувати їх за рейтингом та фільтрувати за вибраними тегами. Також по натисненню по рецепту

користувач перейде до детальної інформації про нього. На цій сторінці користувач має можливість продивитися повну інформацію про рецепт, включаючи назву, опис, коротку інформацію про автора, час приготування, кількість порцій, складність його приготування, детальну інформацію про інгредієнти, необхідні кроки для приготування рецепту з описом та фотографіями. На цій сторінці у користувача буде можливість залишити коментар з оцінкою і додати його до улюблених. Також з цієї сторінки при натисканні на інформацію про автора користувача перекину на сторінку з деталями автора, де у користувача є можливість підписатися на цього автора та переглянути інші його рецепти.

Також з головного екрану у користувача э можливість перейти на сторінку зі своїми рецептами, на якій можна побачити всі створені рецепти. На цьому екрані користувач може перейти на сторінку з детальною інформацією про рецепт, видалити його або перейти на сторінку редагування інформації про рецепт. Також можна перейти на сторінку створення рецепту і заповнити всі поля для його створення.

Іншої важливою функцією до якої користувач може перейти є особистий профіль, де він може продивитися повну інформацію про себе. Також на цьому екрані він зможе редагувати свою інформацію, перейти на сторінку з улюбленими рецептами, де користувач може продивитися улюблені рецепти і потім видалити їх зі списку, за необхідністю, продивитися свої підписки на авторів і потім відписатися від певного автора. Також він може перейти на сторінку з планами, обрати план, або якщо користувач вже підписаний на якийсь план, то відмінити його.

Ця діаграма показує якою повинна бути послідовність дій користувача при вході в застосунок. Це допомагає розробити розділити функціонал на коректні групи, для того щоб зробити правильну навігацію, та прослідкувати за можливими діями користувача.

3.2 Проєктування архітектури програмного забезпечення

Клієнтський мобільний застосунок буде мати багатомодульну архітектуру, що забезпечить кращу швидкість збірки, ізольованість коду та легку масштабованість коду. Архітектура буде будуватися за принципом Clean Architecture, яка поділяє модуль на три основних шари:

- доменний шар – містить бізнес-логіку програми та основні концепції домену. Тут знаходяться сутності, які відображають основні об'єкти домену, такі як користувачі або рецепти. Крім того, у цьому шарі знаходиться опис роботи репозиторіїв за допомогою інтерфейсів;
- шар даних – містить всі механізми доступу до даних, такі як бази даних, мережеві запити або кешування. Він відповідає за збереження та отримання даних з різних джерел. Тут реалізовані інтерфейси репозиторіїв з доменного шару, робота з API та більшість операцій над даними;
- шар презентації – відповідає за відображення даних користувачам та обробку їх вводу. Він містить компоненти інтерфейсу користувача, які відповідають за відображення даних на екрані, а також за обробку подій користувача.

Для шару презентації реалізується архітектурний патерн MVVM (див. рис. 3.3). В цьому патерні Model відображає бізнес логіку та дані, ViewModel відображає шар посередник між даними та екраном, щоб обробити дані для потреб екрану та зберігати стан екрану, а View відображає екрани з побудованими UI елементами, які слідкують за змінами у ViewModel. ViewModel забезпечує двосторонню прив'язку даних, дозволяючи View автоматично оновлюватися при зміні даних в Model, і навпаки, зміни в інтерфейсі користувача автоматично відображаються в Model через ViewModel.

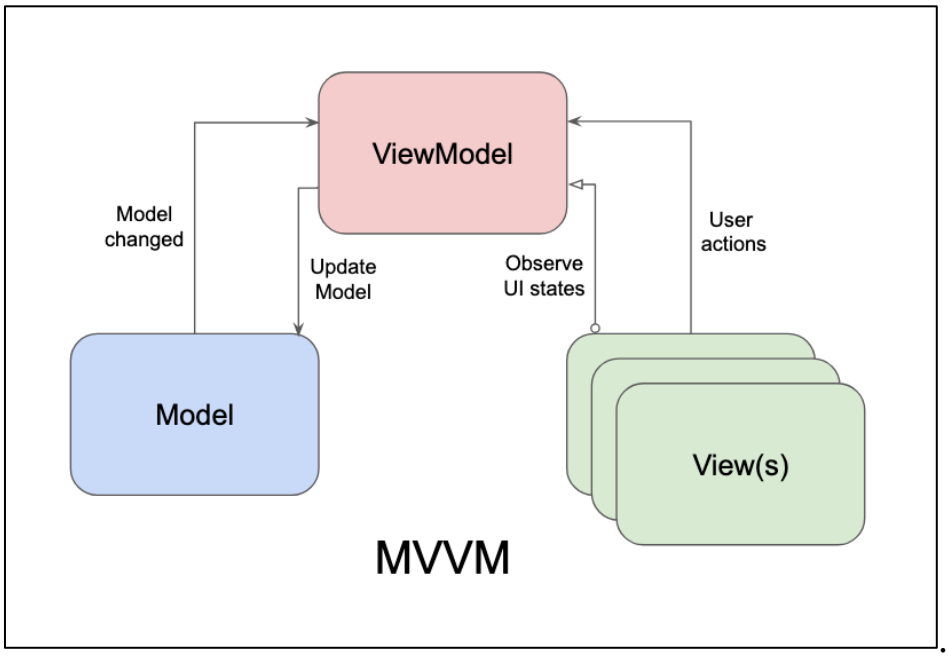

Рисунок 3.3 – Архітектура патерну MVVM (за даними [11])

3.3 Проєктування структури зберігання, управління даними

Оскільки мобільний застосунок буде частиною клієнт-серверної архітектури, усі дані будуть оброблятися та зберігатися на сервері, а потім передаватися застосунку через API. Застосунок візьме на себе відповідальність за перетворення даних у зручний для обробки формат на клієнтській стороні та забезпечить їх зручне відображення користувачу.

Після авторизації, для подальшої роботи з додатком, користувачу буде необхідно зберегти токен доступу до сервера. Це забезпечиться використанням технології DataStore, яка зберігатиме дані у форматі ключ-значення в локальному файлі застосунку.

Щоб оптимізувати завантаження фотографій для різних рецептів, бібліотека Coil буде використовувати кешування цих зображень. Фотографії будуть зберігатися як локальні кеш-файли в каталозі додатку, що дозволить їх завантажувати не з інтернету, а з локальних файлів, що прискорить процес їх відображення.

Також на платформі Android є проблема, що під час перестворення екрану введені дані зникають, тому для збереження стану екрану використовуватимуться ViewModels, які будуть зберігати інформацію після різних взаємодій з екранами і поліпшать користувацький досвід використання застосунка.

3.4 Огляд алгоритмів та методів

Після аналізу мобільних кулінарних застосунків, були обрані наступні ключові функції для реалізації:

- авторизація та реєстрація нових користувачів. Система повинна забезпечити можливість реєстрації нових користувачів та вхід в систему існуючих та керування їх правами доступу. Це дозволить ідентифікувати користувачів і надавати їм доступи до функціоналу, який доступний тільки їм;
- створення своїх рецептів. Користувачі повинні мати можливість створювати власні рецепти за допомогою відповідної функції у додатку. Це дозволить користувачам виражати свою творчість та ділитися власними кулінарними ідеями з іншими користувачами;
- отримання даних з сервера та їх кешування. Система повинна взаємодіяти з сервером для отримання актуальних даних, таких як рецепти, користувацька інформація тощо. Крім того, система повинна забезпечити кешування цих даних на пристрої користувача, щоб забезпечити швидкий доступ до них та зменшити навантаження на сервер;
- перегляд усіх рецептів. Користувачі повинні мати можливість переглядати всі доступні рецепти в застосунку. Система має забезпечити зручний інтерфейс для пошуку, сортування за рейтингом та фільтрації рецептів за тегами. Це дозволить користувачам швидко знаходити цікаві для них рецепти;
- додавання рецептів в улюблене. Користувачі повинні мати можливість додавати рецепти до списку улюблених. Це дозволить їм швидко

знаходити та повертатися до рецептів, які їм сподобалися, без необхідності повторного пошуку;

– підписки на авторів. Користувачі повинні мати можливість підписуватися на авторів, рецепти яких їм подобаються. Це забезпечить автоматичне отримання листів на пошту про нові рецепти та оновлення від улюблених авторів, що сприятиме більшій взаємодії та утриманню користувачів;
– підписки на плани. Додаток повинен підтримувати можливість підписки на плани після яких користувачам відкриється доступ до преміум рецептів від шефів-партнерів, що дозволить користувачам дізнаватися про більш елітні та професійні рецепти.

3.5 Створення UI/UX дизайну системи

Першим, на що звертає увагу користувач – це дизайн системи, її зручність використання та простота, а вже потім на сам функціонал застосунку, тому розробка ефективного UI/UX дизайну [10] відіграє важливу роль у створенні привабливого та інтуїтивно зрозумілого користувацького інтерфейсу.

Проаналізувавши багато мобільних застосунків в конкурентній галузі та актуальні принципи створення дизайну в індустрії, можна назвати наступні рекомендації:

– інтуїтивно зрозуміла навігація: навігаційні елементи повинні бути розміщені логічно і доступно, забезпечуючи швидкий доступ до всіх функцій додатку. Важливі дії мають легко знаходитись та бути доступними за кілька натискань;
– використання заокруглених форм: заокруглені форми сприяють зменшенню візуального напруження і створюють більш приємний естетичний вигляд. Такий дизайн виглядає більш сучасно та привабливо;
– простота дизайну і акцент на основному функціоналі: не потрібно перевантажувати екран зайвою інформацією. Ставити акцент потрібно на основних функціях та забезпечуйте їх легкий доступ. Кожен елемент дизайну повинен мати свою чітко визначену функцію і місце;

- добре описаний зворотній зв’язок: потрібно інформувати користувача про результати його дій. Якщо виникають помилки або потрібні додаткові дії, необхідно надавати користувачу чітку інформацію та рекомендації щодо подальших кроків;
- зручність введення даних: необхідно забезпечити простоту та зручність введення даних. Використовувати інтерактивні елементи, такі як автодоповнення або підказки, щоб полегшити користувачеві завдання;
- інтуїтивна послідовність компонентів: розташовувати елементи інтерфейсу потрібно так, щоб послідовність їх використання була логічною для користувача. Це дозволить зменшити час на навчання та покращити загальний досвід використання додатку.

Проаналізувавши всі рекомендації були створені деякі прототипи екранів застосунку. Одним з найголовніших екранів буде головний екран (див. рис. 3.4), на якому зверху у вигляді іконок будуть доступні функції пошуку за назвою, фільтрації за тегами та сортування рецептів за їх рейтингом, в центрі буде доступний перегляд списку рецептів, на кожному елементі якої буде відображена фотографія страви та головна інформація про рецепт, така як назва та частина опису, а також буде показано рейтинг страви. В свою чергу низу буде зроблена навігації і можливість перейти на екрани профілю та перегляду своїх створених рецептів.

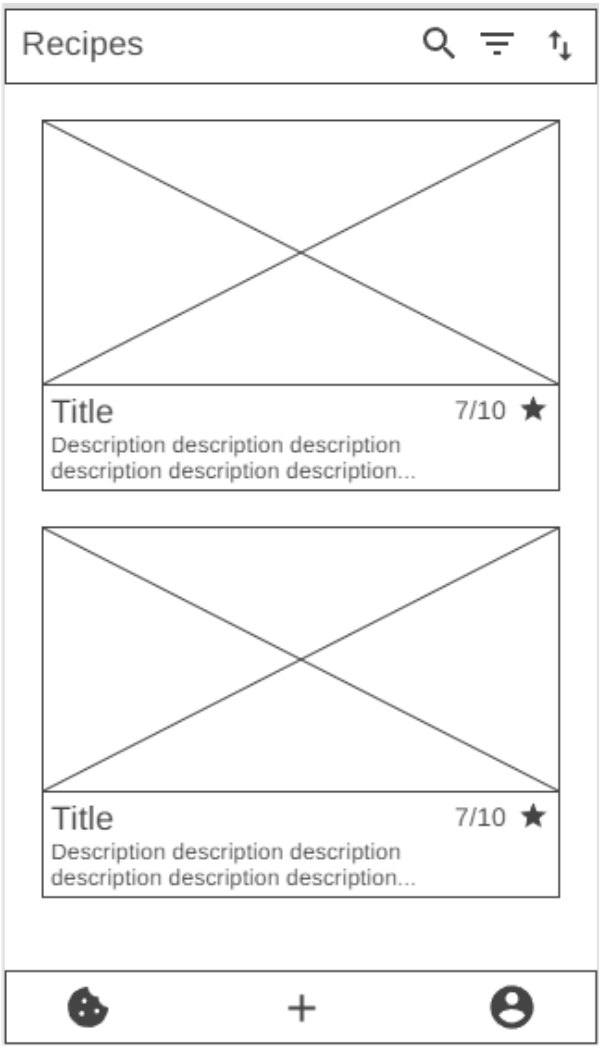

Рисунок 3.4 – Прототип головного екрану (виконано самостійно)

В кулінарному застосунку екран повної інформації про рецепт (див. рис. 3.5) буде використовуватися користувачем найчастіше, тому йому потрібно приділити найбільше уваги. Зверху будуть знаходитися іконки навігації назад та додавання до улюблених. Після іконок буде знаходитися головна інформація про рецепт, а саме: головна фотографія страви, назва, оцінка, автор з його фотографією, дата створення та опис. Далі буде показуватися список інгредієнтів, після якого показуватимуться кроки виконання цього рецепту з фотографією та описом.

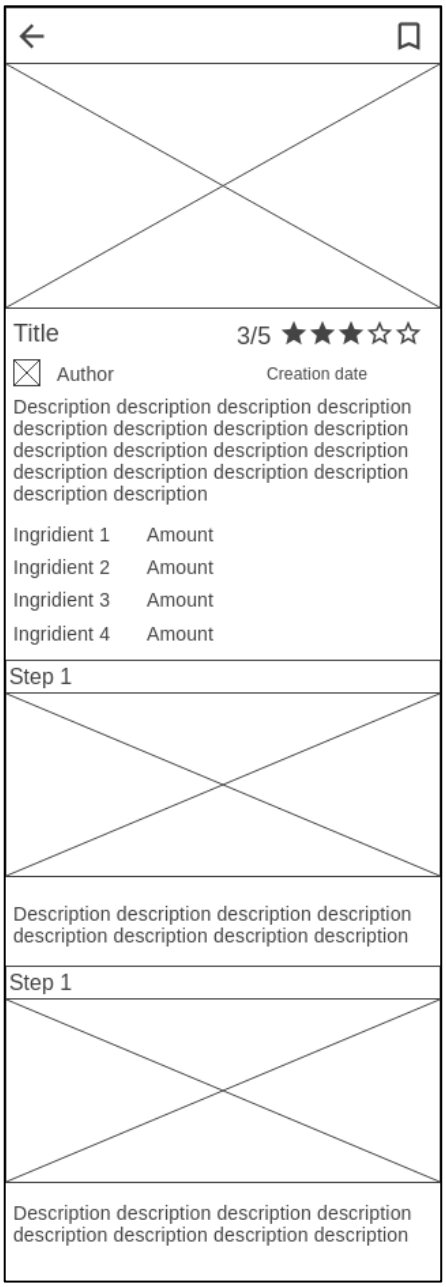

Рисунок 3.5 – Прототип екрану повної інформації про рецепт (виконано самостійно)

Екран створення рецепту в кулінарному застосунку (див. рис. 3.6) є важливим компонентом, оскільки саме тут користувачі вводитимуть інформацію для нових рецептів. Цей екран повинен бути інтуїтивно зрозумілим і зручним для використання. Зверху будуть розміщені іконки навігації назад та збереження рецепту. Нижче знаходиться поле для завантаження головної фотографії страви. Після цього користувачам потрібно буде ввести назву рецепту, його опис, порції, час приготування та складність. Далі буде розміщено розділ для додавання

інгредієнтів, де користувачі зможуть вводити кожен інгредієнт окремо. Після інгредієнтів буде представлено розділ для покрокового опису приготування страви, де користувачі можуть додавати фотографії та текстові описи кожного кроку. І в кінці можна буде додавати теги для доповнення інформації про рецепт.

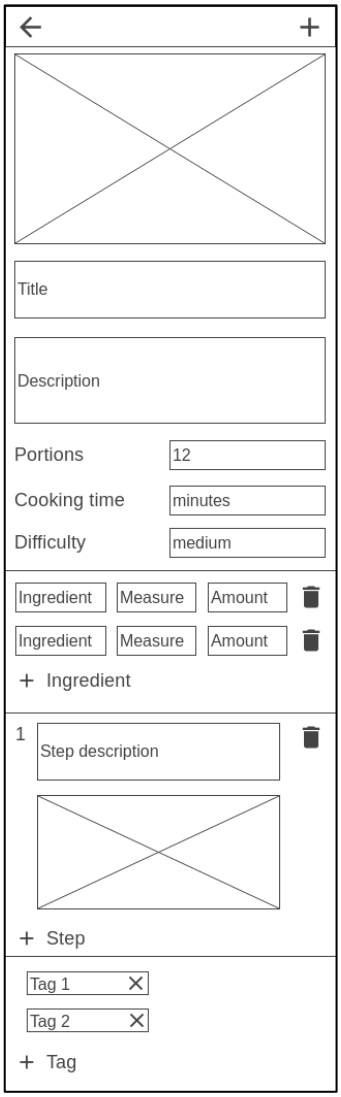

Рисунок 3.6 – Прототип екрану створення рецепту (виконано самостійно)

Створення UI/UX дизайну кулінарного застосунку є критичним етапом, який визначає успіх кінцевого продукту з точки зору користувацького досвіду. В процесі розробки було враховано всі ключові аспекти, що забезпечують зручність та інтуїтивність інтерфейсу. Завдяки ретельному підходу до дизайну вдалося створити систему, яка не лише відповідає вимогам користувачів, але й сприяє їх залученню та активній взаємодії з кулінарним контентом.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Опис вибору архітектурних рішень

Для розробки мобільного застосунка під платформу Android було обрано багатомодульну архітектуру. Вона забезпечує високу модульність і розділення відповідальностей між різними частинами застосунка, що сприяє створенню легко підтримуваного та масштабованого коду. Кожен модуль відповідає за певну функціональність або компонент додатка, що дозволяє розробникам зосереджуватися на конкретних завданнях і знижує складність проекту в цілому. Було вирішено розділити застосунок на 6 основних модулів (див. рис. 4.1).

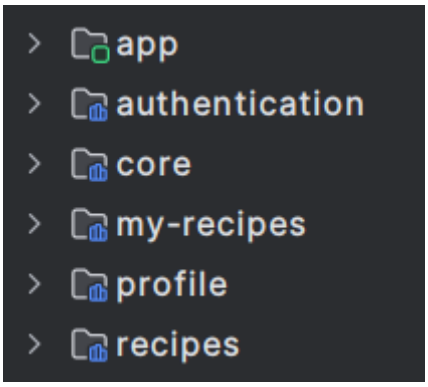

Рисунок 4.1 – Модулі застосунка (виконано самостійно)

Модулі було розділено за різними потоками навігації, а саме:

– app: головний модуль, який відповідає за об’єднання усього функціоналу застосунка;
– authentication: відповідає да екрани та логіку реєстрації та авторизації в застосунку;
– core: зберігає в собі основні компоненти, функції та логіку, яка буде доступна усім іншим модулям;
– my-recipes: відповідає за перегляд рецептів користувача, їх створення, редагування та видалення;
– profile: відповідає за перегляд особистої інформації про користувача, її редагування, перегляд обраних рецептів та підписок на авторів;

– recipes: відповідає за перегляд усіх рецептів, пошук за назвою, сортуванням та фільтрацією.

Однією з головних переваг багатомодульної архітектури є підвищення ефективності розробки. Завдяки чіткому розділенню коду на модулі, різні команди розробників можуть працювати над окремими модулями незалежно один від одного. Це сприяє паралельній розробці та скорочує час на інтеграцію та тестування. Крім того, кожен модуль може бути розроблений, протестований і розгорнутий окремо, що дозволяє швидше впроваджувати нові функції та виправлення.

Багатомодульна архітектура також забезпечує кращу підтримуваність коду. Завдяки чіткій структурі проекту, нові члени команди можуть легше зрозуміти архітектуру додатка та швидше долучитися до розробки. Кожен модуль має чітко визначені інтерфейси та залежності, що знижує ризик виникнення помилок при внесенні змін і спрощує рефакторинг коду.

Крім того, багатомодульна архітектура сприяє повторному використанню коду. Окремі модулі можуть бути легко використані в інших проектах або додатках, що підвищує ефективність розробки та знижує витрати на створення нових рішень.

Ще однією важливою перевагою багатомодульної архітектури є покращення продуктивності та зменшення часу збірки додатка. Розділення проекту на менші модулі дозволяє Gradle виконувати інкрементальні збірки, що значно зменшує час, необхідний для компіляції змін. Це прискорює процес розробки та підвищує продуктивність команди.

Для організації коду в кожному модулі було обрано архітектурний підхід Clean Architecture. Дотримуючись його принципів, можна чітко розділити бізнес-логіку, логіку взаємодії з даними та презентаційний рівень, що дозволяє уникнути складних та монолітних структур. Кожен модуль було поділено на декілька шарів відповідальності (див. рис. 4.2).

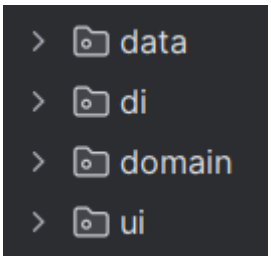

Рисунок 4.2 – Шари відповідальності (виконано самостійно)

Кожен з цих шарів відповідає за своє місце у структурі, а саме:

– data: відповідає за зв’язок з серверною частиною, обробкою інформації та помилок;
– di: відповідає за впровадження залежностей;
– domain: відповідає за правила бізнес логіки і охарактеризує, як ui шару потрібно взаємодіяти з data;
– ui: відповідає за отримання інформації з data та відображення її на екрані застосунку.

Однією з головних переваг Clean Architecture є незалежність від конкретних фреймворків та платформ. Це означає, що основні бізнес-правила та логіка додатка не залежать від деталей реалізації та можуть бути легко адаптовані для різних платформ або змін фреймворків. Це підвищує гнучкість та забезпечує довготривалу підтримку додатка. Чітка структура проекту та розповсюдженість серед розробників робить його легшим для розуміння новими членами команди, що скорочує час на розуміння коду та знижує ймовірність виникнення помилок при внесенні змін до нього.

Ще однією важливою перевагою Clean Architecture є можливість повторного використання коду. Завдяки чітко визначеним інтерфейсам та абстракціям можна легко використовувати окремі компоненти в різних проектах або модулях, що підвищує ефективність розробки та знижує витрати на створення нових додатків.

Таким чином, вибір Clean Architecture для розробки нашого мобільного застосунка був стратегічним рішенням, спрямованим на створення високоякісного, масштабованого та підтримуваного коду. Завдяки своїм перевагам, таким як

модульність, незалежність від фреймворків, полегшення тестування, зниження технічного боргу та можливість повторного використання коду, Clean Architecture дозволяє забезпечити високий рівень задоволеності як розробників, так і кінцевих користувачів.

Для організації частини користувацького інтерфейсу було обрано архітектурний патерн Model-View-ViewModel (MVVM). Він забезпечує чітке розділення відповідальностей між різними компонентами додатка, що полегшує підтримку та масштабування коду. У моделі MVVM модель (Model) відповідає за управління даними та бізнес-логікою, представлення (View) відповідає за відображення інформації користувачу, а модель представлення (ViewModel) виступає посередником, який забезпечує зв'язок між моделлю та представленням.

MVVM забезпечує високу модульність та повторне використання компонентів. Завдяки чітко визначеним інтерфейсам та абстракціям можна легко використовувати ті самі ViewModel у різних частинах додатка або в різних проектах, що підвищує ефективність розробки та знижує витрати на створення нових додатків.

Крім того, MVVM інтегрується з іншими компонентами екосистеми Android, такими як LiveData та ViewModel з Jetpack, що забезпечує зручну та ефективну роботу з даними та управління життєвим циклом компонентів. Використання Flow дозволяє автоматично оновлювати UI при зміні даних, що спрощує роботу розробників та забезпечує кращий користувацький досвід. На рисунку 4.3 продемонстровано екрану з усіма рецептами та показано взаємодію ViewModel та екрану в застосунку. За допомогою бібліотеки Hilt впроваджується RecipeViewModel у екран рецептів і за допомогою StateFlow на стороні ViewModel та функції collectAsState() організовано реактивне оновлення цього екрану.

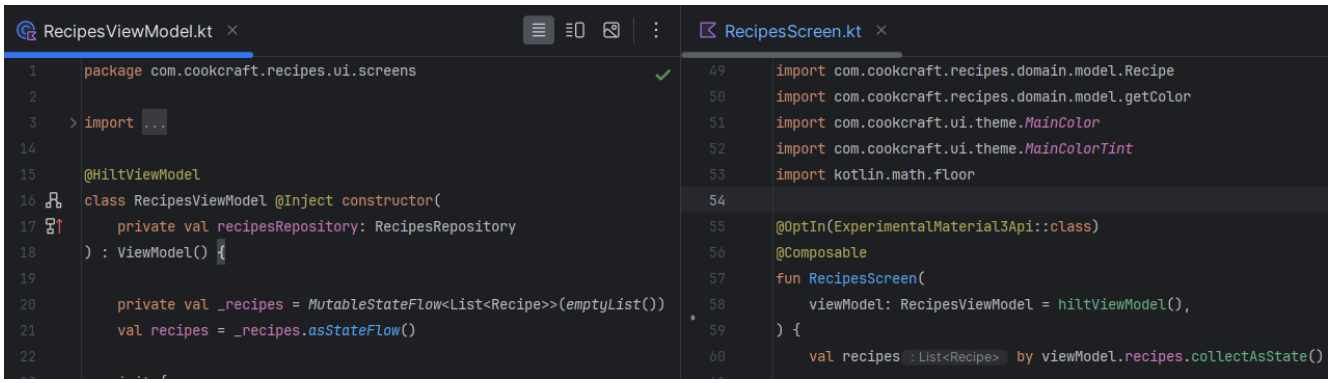

Рисунок 4.3 – Приклад взаємодії екрану та ViewModel в застосунку (виконанно самостійно)

Ще однією важливою перевагою MVVM є покращення підтримуваності та зрозумілості коду. Завдяки чіткій структурі та розділенню відповідальностей код стає легшим для розуміння новими членами команди, що скорочує час на навчання та знижує ймовірність виникнення помилок при внесенні змін до коду.

Завдяки своїм перевагам, таким як покращення тестованості, висока модульність, інтеграція з екосистемою Android, підтримуваність та зручність, MVVM дозволяє забезпечити високий швидкості розробки та впровадження зрозумілого коду.

4.2 Опис вибору технологій

Для розробки мобільного застосунка під платформу Android було обрано мову програмування Kotlin. По-перше, Kotlin є сучасною мовою програмування, яка враховує найкращі практики та тенденції в області розробки програмного забезпечення. Її синтаксис та конструкції спрощують написання коду, роблячи його більш читабельним і зрозумілим.

Другою важливою причиною є повна сумісність Kotlin з Java. Це дозволяє використовувати існуючі бібліотеки та фреймворки, написані на Java, що значно спрощує написання коду під Android та прискорює процес розробки. Крім того, Google офіційно підтримує Kotlin як основну мову для розробки під Android, що забезпечує наявність великої кількості ресурсів, документації та інструментів,

спеціально адаптованих для роботи з Kotlin на платформі Android. Крім того, Kotlin має велику та активну спільноту розробників, що забезпечує доступ до численних навчальних матеріалів, форумів, конференцій та інших ресурсів, полегшуючи навчання та вирішення проблем, що виникають під час розробки.

Kotlin також має вбудовані механізми для запобігання багатьом типам помилок, зокрема NullPointerException, що підвищує надійність та стабільність застосунка. Підтримка функціонального програмування дозволяє використовувати лямбда-функції, високорівневі функції та інші конструкції, які спрощують написання ефективного та компактного коду. Завдяки менш громіздкому синтаксису порівняно з Java, Kotlin дозволяє писати менше коду для виконання тих самих задач, що прискорює процес розробки та робить код більш підтримуваним та легшим для розуміння.

Таким чином, обрання Kotlin як основної мови програмування для розробки мобільного застосунка під Android було правильним рішенням, спрямованим на створення сучасного, безпечного та ефективного продукту. Завдяки своїм перевагам, таким як сучасність, сумісність з Java, офіційна підтримка Google, безпечність, функціональні можливості, зручність та активна спільнота, Kotlin дозволяє швидко і якісно реалізувати всі заплановані функції та забезпечити високий рівень задоволеності користувачів.

Також для розробки було обрано Jetpack Compose як основний інструмент для створення інтерфейсу користувача. Jetpack Compose є сучасним інструментом для побудови інтерфейсів, який дозволяє створювати UI за допомогою декларативного підходу. Це значно спрощує процес розробки, оскільки розробники можуть описувати, як має виглядати інтерфейс, без необхідності турбуватися про його стан та зміни (див. рис. 4.4).

```kotlin
@Composable
fun CraftButton(
    text: String,
    onClick: () -> Unit,
    modifier: Modifier = Modifier,
    enabled: Boolean = true,
) {
    Button(
        shape = RoundedCornerShape(ButtonRadius),
        enabled = enabled,
        colors = ButtonDefaults.buttonColors(containerColor = MainColor),
        onClick = onClick,
        modifier = modifier.height(ButtonHeight),
    ) {
        Text(
            text = text,
            style = CraftButtonStyle,
            modifier = Modifier.fillMaxWidth()
        )
    }
}
```

Рисунок 4.4 – Приклад компоненту написаного за допомогою Jetpack Compose (виконано самостійно)

Jetpack Compose інтегрується з іншими компонентами екосистеми Jetpack, що забезпечує зручне використання сучасних архітектурних підходів та бібліотек, таких як ViewModel, LiveData, Navigation та інших. Крім того, Jetpack Compose забезпечує високу продуктивність та гнучкість у розробці інтерфейсів. Використовуючи Kotlin як основну мову програмування, Compose дозволяє писати менш громіздкий та більш читабельний код, що полегшує його підтримку та розширення. Це також знижує кількість потенційних помилок та підвищує стабільність додатка. Jetpack Compose, як і Kotlin активно підтримується Google, що гарантує постійні оновлення, нові можливості та покращення, наявність документації, прикладів та навчальних матеріалів, які допомагають розробникам швидко освоїти та ефективно використовувати цей інструмент.

Ще однією важливою перевагою є можливість створення адаптивних інтерфейсів, які добре виглядають на різних розмірах екранів та пристроях. Це дозволяє забезпечити користувачам оптимальний досвід використання незалежно

від їхнього пристрою. Також важливою перевагою є можливість перегляду вигляду компонента з різними станами без необхідності запуску застосунка, яке виконується за допомогою анотації @Preview (див. рис. 4.5).

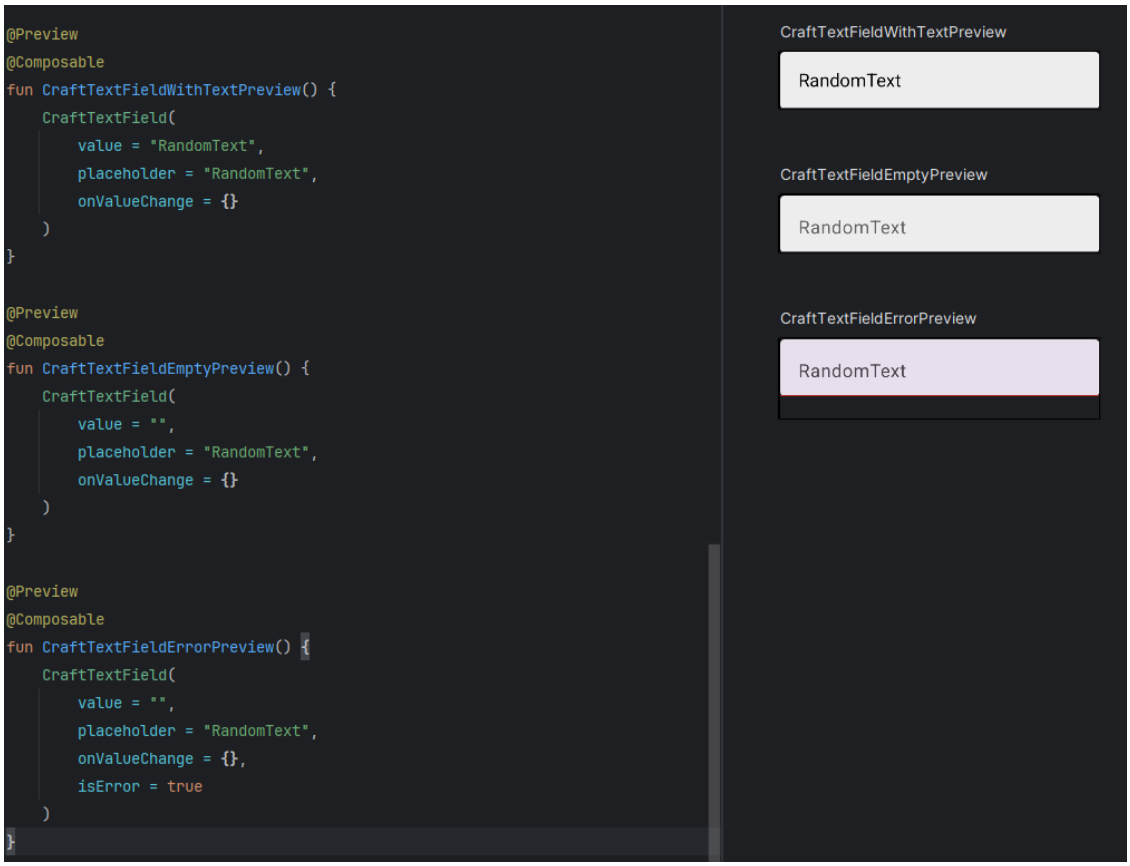

Рисунок 4.5 – Приклад використання Preview (виконано самостійно)

Для роботи з асинхронними операціями було обрано бібліотеки Kotlin Coroutines та Flow. Вони дозволяють писати асинхронний код, який виглядає та працює як синхронний, що значно спрощує його читання і розуміння. Використання корутин зменшує кількість коду, необхідного для управління потоками, та допомагає уникнути багатьох проблем. На рисунку 4.6 продемонстровано приклад отримання даних за допомогою цих двох технологій.

```
private val _recipes = MutableStateFlow<List<Recipe>>(emptyList())
val recipes = _recipes.asStateFlow()

init {
    getAllMyRecipes()
}

private fun getAllMyRecipes() {
    viewModelScope.launch(Dispatchers.IO) {
        val result : RequestState<List<Recipe>>  = recipesRepository.getAllRecipes()
        if (result is RequestState.Success) {
            _recipes.value = result.data
        }
    }
}
```

Рисунок 4.6 – Приклад взаємодії корутин та Flow

Корутини інтегруються з багатьма іншими бібліотеками і фреймворками, що робить їх гнучким і універсальним інструментом для різноманітних задач, таких як мережеві запити, робота з базами даних та обробка подій користувача. Вони також забезпечують високий рівень продуктивності, оскільки дозволяють ефективно використовувати ресурси системи.

Flow, як частина бібліотеки Kotlin Coroutines, додає можливості реактивного програмування. Flow забезпечує простий і потужний спосіб роботи з потоками даних, дозволяючи легко маніпулювати даними, трансформувати їх та реагувати на зміни. Використання Flow дозволяє створювати гнучкі та масштабовані системи обробки подій та даних.

Flow також забезпечує підтримку таких функцій, як об'єднання потоків, обробка помилок та управління ресурсами, що робить його ідеальним для складних сценаріїв асинхронної обробки даних. Це допомагає створювати більш надійні та стійкі додатки, які можуть ефективно реагувати на зміни в реальному часі.

Вибір Kotlin Coroutines та Flow дозволяє зменшити складність асинхронного програмування, підвищити продуктивність та гнучкість додатка, а також забезпечити кращу підтримку сучасних архітектурних підходів, таких як реактивне

програмування. Це стратегічне рішення спрямоване на створення високоякісного, ефективного та надійного коду, який легко підтримувати та розширювати.

Для розробки було вирішено використовувати технологію впровадження залежностей, оскільки сучасна розробка без цього інструменту майже неможлива. Основні переваги цієї технології включають зменшення зв’язності коду, спрощення управління залежностями, підвищення гнучкості та масштабованості додатка, а також полегшення управління життєвим циклом об’єктів. Це робить код більш модульним, тестованим і підтримуваним, що особливо важливо для створення масштабованих і надійних застосунків.

Для використання впровадження залежностей було обрано бібліотеку Hilt як основний інструмен. Hilt є офіційним фреймворком від Google для впровадження залежностей у додатки Android, що забезпечує тісну інтеграцію з іншими компонентами екосистеми Jetpack та Android.

Однією з головних переваг Hilt є спрощення процесу впровадження залежностей. Завдяки автоматичному управлінню життєвим циклом компонентів, Hilt зменшує обсяг ручної роботи, необхідної для налаштування залежностей у додатку, що дозволяє розробникам зосередитися на логіці додатка, а не на конфігурації DI-контейнерів.

Hilt також сприяє кращій організації коду та підвищенню його підтримуваності. Використовуючи Hilt, можна легко керувати залежностями між різними модулями додатка, що робить архітектуру більш модульною та гнучкою. Це дозволяє легко тестувати окремі компоненти застосунка завдяки можливості замінювати залежності на тестові реалізації. Крім того, Hilt підтримує всі основні архітектурні компоненти Android, такі як ViewModel, Navigation, WorkManager та інші, що забезпечує послідовність та узгодженість у використанні залежностей в різних частинах додатка. Це також дозволяє зменшити кількість коду, який потрібно писати вручну, завдяки автоматичній генерації коду для впровадження залежностей. На рисунку 4.7 показано приклад впровадження залежностей за допомогою інструментів бібліотеки Hilt.

```
@Module
@InstallIn(ViewModelComponent::class)
class ProfileApiModule {

    @Provides
    fun provideProfileApi(retrofit: Retrofit): ProfileApi {
        return retrofit.create(ProfileApi::class.java)
    }

    @Provides
    fun provideSubscriptionsApi(retrofit: Retrofit): SubscriptionsApi {
        return retrofit.create(SubscriptionsApi::class.java)
    }
}
```

Рисунок 4.7 – Приклад впровадження залежностей

Hilt також активно підтримується та розвивається спільнотою Google, що гарантує регулярні оновлення, виправлення помилок та додавання нових можливостей. Ще однією важливою перевагою Hilt є його сумісність з іншими популярними бібліотеками та фреймворками для Android, що забезпечує легку інтеграцію в існуючі проекти та дозволяє використовувати Hilt разом з іншими інструментами, такими як Retrofit, Room та інші.

Для здійснення мережевих запитів як основний інструмент було обрано бібліотеку Retrofit. Вона є популярною та надійною бібліотекою, розробленою компанією Square, яка значно спрощує інтеграцію з RESTful API. Вона забезпечує зручний та інтуїтивно зрозумілий спосіб роботи з мережевими запитами, що дозволяє розробникам легко обмінюватися даними між клієнтською частиною додатка та сервером.

Однією з головних переваг Retrofit є його простота та зручність використання. Бібліотека автоматично перетворює HTTP API в інтерфейси Kotlin, що дозволяє розробникам визначати кінцеві точки API як методи інтерфейсу (див. рис. 4.8). Це значно зменшує обсяг коду, необхідного для створення та обробки мережевих запитів, а також робить код більш читабельним та підтримуваним.

```
interface ProfileApi {

    @GET("users/profile")
    suspend fun getProfile(): Response<ProfileApiModel>

    @Multipart
    @POST("users/update")
    suspend fun updateUserPhoto(
        @Part profileImage: MultipartBody.Part,
    ): Response<ProfileApiModel>

    @Multipart
    @POST("users/update")
    suspend fun updateProfile(
        @Part("firstName") firstName: RequestBody,
        @Part("lastName") lastName: RequestBody,
        @Part("email") email: RequestBody,
        @Part("birthDate") birthDate: RequestBody,
        @Part("gender") gender: RequestBody,
    ): Response<ProfileApiModel>

    @POST("auth/logout")
    suspend fun logout(): Response<LogoutApiModel>
}
```

Рисунок 4.8 – Приклад інтерфейсу для HTTP-запитів (виконано самостійно)

Retrofit підтримує широкий спектр форматів даних, таких як JSON, XML та протоколу об'єктів, що дозволяє легко обробляти різні типи відповідей від сервера. Для автоматичного перетворення JSON файлів, які приходять з серверної частини, було використано бібліотеку перетворювач Gson, і використовуючи її, Retrofit може автоматично серіалізувати та десеріалізувати дані, що додатково спрощує роботу з мережевими запитами.

Крім того, Retrofit підтримує асинхронні запити, що дозволяє виконувати мережеві операції у фоновому режимі без блокування головного потоку додатка. Це забезпечує плавність роботи додатка та кращий користувацький досвід. Асинхронні запити в Retrofit легко реалізуються за допомогою зворотних викликів або використанням Kotlin Coroutines, що робить код ще більш зрозумілим та легким для обробки.

Retrofit також інтегрується з іншими популярними бібліотеками для Android, такими як OkHttp, що дозволяє налаштовувати та розширювати функціональність

мережевих запитів. OkHttp забезпечує додаткові можливості, такі як кешування відповідей, управління підключеннями та інтерцептори для модифікації запитів та відповідей. Також значно спрощує роботу з запитами інтерфейс TokenInterceptor, який автоматично вбудовує API токен в кожен запит.

Таким чином, вибір Retrofit для здійснення мережевих запитів спрямовано на спрощення та прискорення процесу розробки, підвищення якості та продуктивності додатка, а також забезпечення кращого досвіду для кінцевих користувачів.

Для зберігання постійних даних у мобільному застосунку під було обрано DataStore. DataStore є сучасним інструментом, який забезпечує більш безпечне та ефективне зберігання даних у порівнянні з попередніми рішеннями, такими як SharedPreferences. Ця бібліотека пропонує декілька типів зберігання, що дозволяє розробникам вибирати найбільш підходящий варіант для конкретних потреб додатка. Для даної програмної системи було обрано Preferences DataStore, для зберігання токену користувача (див. рис. 4.9).

```
val Context.dataStore: DataStore<Preferences> by preferencesDataStore(name = DATA_STORE_PREFERENCES)

class DataStoreRepositoryImpl @Inject constructor(
    private val context: Context,
) : DataStoreRepository {
    private val TOKEN = stringPreferencesKey(TOKEN_KEY)

    override val tokenFlow: Flow<String> = context.dataStore.data
        .map { preferences ->
            preferences[TOKEN] ?: ""
        }

    override suspend fun saveToken(token: String) {
        context.dataStore.edit { settings ->
            settings[TOKEN] = token
        }
    }

    companion object {
        private const val TOKEN_KEY = "token"

        const val DATA_STORE_PREFERENCES = "settings"
    }
}
```

Рисунок 4.9 – Збереження токену з використанням DataStore (виконано самостійно)

Однією з головних переваг DataStore є асинхронна природа операцій збереження та зчитування даних, що забезпечує плавний користувацький досвід без блокування головного потоку додатка. Крім того, DataStore використовує Kotlin Coroutines та Flow, що робить його інтеграцію з іншими сучасними бібліотеками та компонентами Android простою та зручною.DataStore також забезпечує збереження даних у структурованому та типобезпечному форматі, що підвищує надійність та захищеність даних. Це робить його ідеальним вибором для зберігання налаштувань користувача та інших критично важливих даних.

Для завантаження та відображення зображень у мобільному застосунку було обрано Coil. Ця бібліотека є сучасною, яка використовує Kotlin Coroutines для асинхронного завантаження зображень, що забезпечує високу продуктивність та плавність роботи додатка.

Coil інтегрується з Jetpack Compose, що дозволяє легко використовувати його для завантаження та відображення зображень у декларативних інтерфейсах. Використовуючи простий та зрозумілий API, Coil дозволяє розробникам швидко налаштовувати завантаження зображень, включаючи кешування, трансформації та обробку помилок.

Крім того, Coil оптимізований для мінімального використання пам'яті та ресурсів, що робить його ефективним вибором для мобільних додатків. Він також підтримує різні формати зображень та надає можливості для роботи з анімаціями, що дозволяє створювати більш привабливі та інтерактивні інтерфейси.

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Для тестування мобільного застосунка було обрано мануальне тестування, оскільки воно забезпечує детальну перевірку функціональності, юзабіліті та загальної якості додатка. Мануальне тестування дозволяє тестувати застосунок, як це робили б реальні користувачі, що допомагає виявити приховані проблеми та оцінити візуальні аспекти інтерфейсу. Кроки тестування функціоналу реєстрації та отримані результати наведені в таблиці 5.1.

Таблиця 5.1 – Тест-кейс №1 (таблиця виконана самостійно)

Інформація про тест-кейс			
Ідентифікатор тесту:	Тест-кейс №1		
Опис функції:	Реєстрація		
Власник тесту:	Захарченко Ярослав Вікторович		
Дата створення:	03.06.2023		
Мета тесту:	Перевірити коректність реалізації реєстрації користувача		
Передумова			
№	Опис випадку	Очікуваний результат	Висновок
1	Відкрити мобільний застосунок	Користувач має доступ до застосунку, який відкритий	Пройдено
Створення рецепту			
№	Опис випадку	Очікуваний результат	Висновок
1	Натиснути на кнопку “Зареєструватися”	Перенаправлено на сторінку створення акаунту	Пройдено
2	Заповнити обов’язкові поля «Ім’я» та «Прізвище»	Поля заповнені коректно без зайвих символів	Пройдено
3	Обрати «Дата народження»	Обрана дата не більше теперішньої дати	Пройдено
4	Обрати «Гендер»	Обрано одно з двух наявних гендерів	Пройдено
5	Натиснути кнопку «Наступне»	Перевірено на правильність введених даних та перенаправлено на наступний екран реєстрації	Пройдено
6	Заповнити обов’язкове поле «Електронна пошта»	Поле заповнено відповідно вимогам для електронної пошти	Пройдено
7	Заповнити обов’язкові поля «Пароль» та «Підтвердження паролю»	Поля заповнені літерами та цифрами та символи в полях ідентичні	Пройдено

Кінець таблиці 5.1

8	Натиснути кнопку «Зареєструватися»	Перевірена валідація полей, створено акаунт та перенаправлено на екран з усіма рецептами	Пройдено
Результати тестування			
Тестувальник: Захарченко Я.В.	Дата прогону тесту: 03.06.2023	Результат тесту (P/F/B): ПРОЙДЕНО (P)	

Одним з найважливішого функціоналу застосунку є створення рецепту, тому було вирішено протестувати цей функціонал. Кроки тестування функціоналу створення рецепту та отримані результати наведені в таблиці 5.2.

Таблиця 5.2 – Тест-кейс №2 (таблиця виконана самостійно)

Інформація про тест-кейс			
Ідентифікатор тесту:	Тест-кейс №2		
Опис функції:	Створення рецепту		
Власник тесту:	Захарченко Ярослав Вікторович		
Дата створення:	08.06.2023		
Мета тесту:	Перевірити коректність реалізації створення рецепту користувачем		
Передумова			
№	Опис випадку	Очікуваний результат	Висновок
1	Відкрити мобільний застосунок	Користувач має доступ до застосунку, який відкритий	Пройдено
2	Має обліковий запис та хоче авторизуватися	Перенаправлено на сторінку профілю	Пройдено
Створення рецепту			
№	Опис випадку	Очікуваний результат	Висновок
1	Відкрити екран зі своїми рецептами	Перенаправлено на сторінку з власними рецептами	Пройдено
2	Натиснути на кнопку для створення рецепту	Перенаправлено на сторінку для створення рецепту	Пройдено
3	Натиснути на шаблон фото і обрати фото з галереї	Обране фото відображається на екрані	Пройдено
4	Заповнити обов’язкові поля «Назва» та «Опис»	Поля заповнені правильно	Пройдено
5	Заповнити обов’язкові поля «Час приготування» та «Порції»	Поля заповнені правильно	Пройдено

Кінець таблиці 5.2

6	Обрати «Складність»	Обрано одну з трьох складностей	Пройдено
7	Створити необхідну кількість інгредієнтів та заповнити необхідні поля «Інгредієнт», «Міра» та «Кількість» в кожному інгредієнті	Поля заповнені правильними типами даних	Пройдено
8	Створити необхідну кількість кроків, заповнити необхідне поле «Опис» та обрати фото для кожного кроку	Поля заповнені правильно та обрані фотографії видно на екрані	Пройдено
9	Обрати відповідні теги для рецепту	Обрані теги відображаються на екрані	Пройдено
10	Перемкнути прапорець «Публікувати»	Поля заповнені правильно	Пройдено
11	Заповнити обов’язкові поля «Час приготування» та «Порції»	Збережено вибір чи публікувати рецепт	Пройдено
12	Натиснути на кнопку створення рецепту	Перенаправлено на екран зі списком усіх створених рецептів, де показується створений рецепт	Пройдено
Результати тестування			
Тестувальник: Захарченко Я.В.	Дата прогону тесту: 03.06.2023	Результат тесту (P/F/B): ПРОЙДЕНО (P)	

6 ВПРОВАДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

За тематикою кваліфікаційної роботи було підготовлено наукову статтю у вигляді тез для V Всеукраїнської студентської наукової конференції «Розвиток сучасної науки: актуальні питання теорії та практики», що відбувся 19 квітня 2024 року в онлайн режимі. Представлену роботу було опубліковано у збірнику конференції під заголовком «Багатомодульність як спосіб оптимізації розробки та підтримки андроїд застосунку» [12].

У цій статті обговорювалися основні аспекти удосконалення розробки програмного забезпечення під платформу Android за допомогою впровадження багатомодульної архітектури. Були переглянуті основні проблеми, з якими стикаються Android розробники, та яким чином багатомодульність їх вирішує. Були надані основні поради та переваги впровадження модульної архітектури та підкреслювала її важливість для роботи в команді.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи бакалавра був проведений глибокий аналіз ринку кулінарних застосунків, були виявлені недоліки існуючих рішень та їх переваги, для покращеного розуміння тенденцій розвитку цієї сфери. На основі проведених аналізів було виявлено основні вимоги та поставлені завдання для розробки зручного мобільного застосунку для обміну кулінарними рецептами.

Для реалізації мобільного застосунка для платформи Android (клієнтської частини) системи, була обрана мова програмування Kotlin, яка є найпоширенішою для розробки під платформу Android, набір інструментів та бібліотек Android SDK. Для структуризації застосунку було обрано багатомодульну архітектуру, використовуючи підхід Clean Architecture в кожному з модулів та архітектурний патерн MVVM, який відповідає за організацію коду шару користувацького інтерфейсу з використанням сучасної бібліотеки Jetpack Compose для декларативної побудови екранів застосунку.

Впровадження надійних архітектурних рішень та перевірених бібліотек, а також використання основних способів оптимізації та запобігання помилок для розробки мобільного застосунку дозволили створити якісний застосунок для обміну рецептами, а зручний інтерфейс відповідає тенденціям ринку та здатний задовільнити потреби цільової аудиторії.

Отже, розроблений мобільний застосунок не лише відповідає потребам галузі та використовує передові рішення для забезпечення високоякісного користувацького досвіду, але й має потенціал для розширення та вдосконалення, що дозволить привернути більше користувачів з кулінарної галузі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Посібник з архітектури застосунку [Електронний ресурс] – URL: https://developer.android.com/topic/architecture (дата звернення: 10.05.2024)

2. Android Jetpack [Електронний ресурс] – URL: https://developer.android.com/jetpack (дата звернення: 10.05.2024).

3. Як будувати UML-діаграми. Розбираємо три найпопулярніші варіанти [Електронний ресурс] – URL: https://dou.ua/forums/topic/40575/ (дата звернення: 10.05.2024)

4. Посібник з модуляризації додатків для Android [Електронний ресурс] – URL: https://developer.android.com/topic/modularization (дата звернення: 10.05.2024)

5. Леонард Річардсон. RESTful Web API: Сервіси для мінливого світу. – 1-ше вид. – Каліфорнія: O'Reilly, 2013. – 406 с.

6. Пітер Шпет. Вивчайте Kotlin для розробки на Android: Мова наступного покоління для програмування сучасних додатків для Android. Нью-Йорк: APress, 2019. – 508 с.

7. Джейсон Морріс. Розробка користувацького інтерфейсу Android: Посібник для початківців. Бірмінгем: Packt Pub Ltd, 2011. – 287 с.

8. Таль Атер. Створення прогресивних веб-додатків: Використання можливостей нативної мови в браузері. – 1-е видання. – Каліфорнія: O'Reilly, 2017. – 288 с.

9. Ніл Форд, Марк Річардс. Основи архітектури програмного забезпечення: Вичерпний посібник з патернів, характеристик та найкращих практик. – 1-е видання. – Каліфорнія: O'Reilly, 2020. – 500 с.

10. Майк Монтейро. Дизайн - це робота. Нью-Йорк: A Book Apart, 2012. – 135 с.

11. Model-View-ViewModel [Електронний ресурс] – URL: https://uk.wikipedia.org/wiki/Model-View-ViewModel (дата звернення: 10.05.2024)

12. Матеріали V Всеукраїнської студентської наукової конференції [Електронний ресурс] – URL: https://archive.liga.science/index.php/conference-proceedings/issue/view/ukr-19.04.2024/75 (дата звернення: 10.05.2024)

ДОДАТОК А

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ

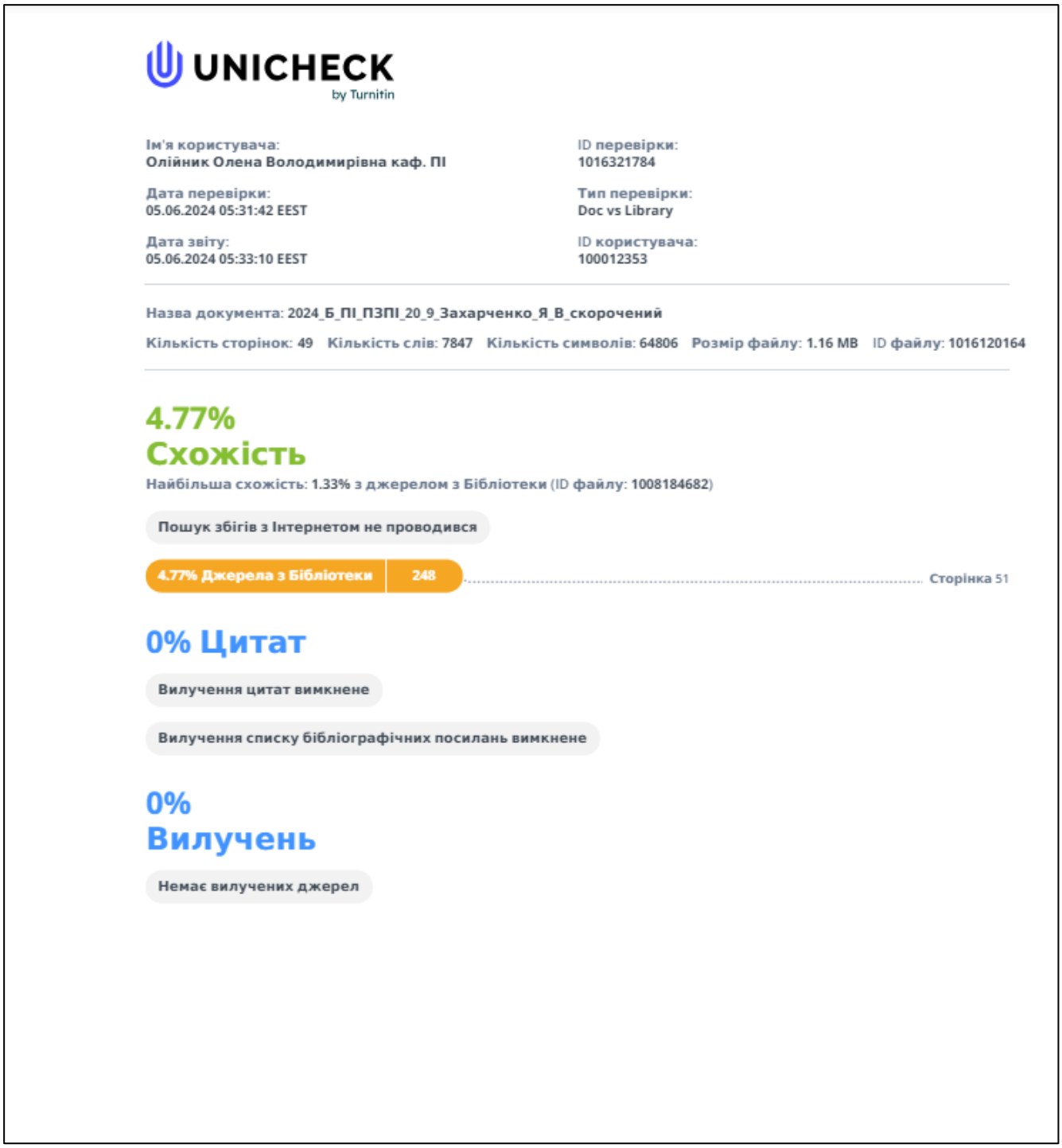

Рисунок А.1 – Звіт з результату перевірки на унікальність тексту

ДОДАТОК Б

Слайди презентації

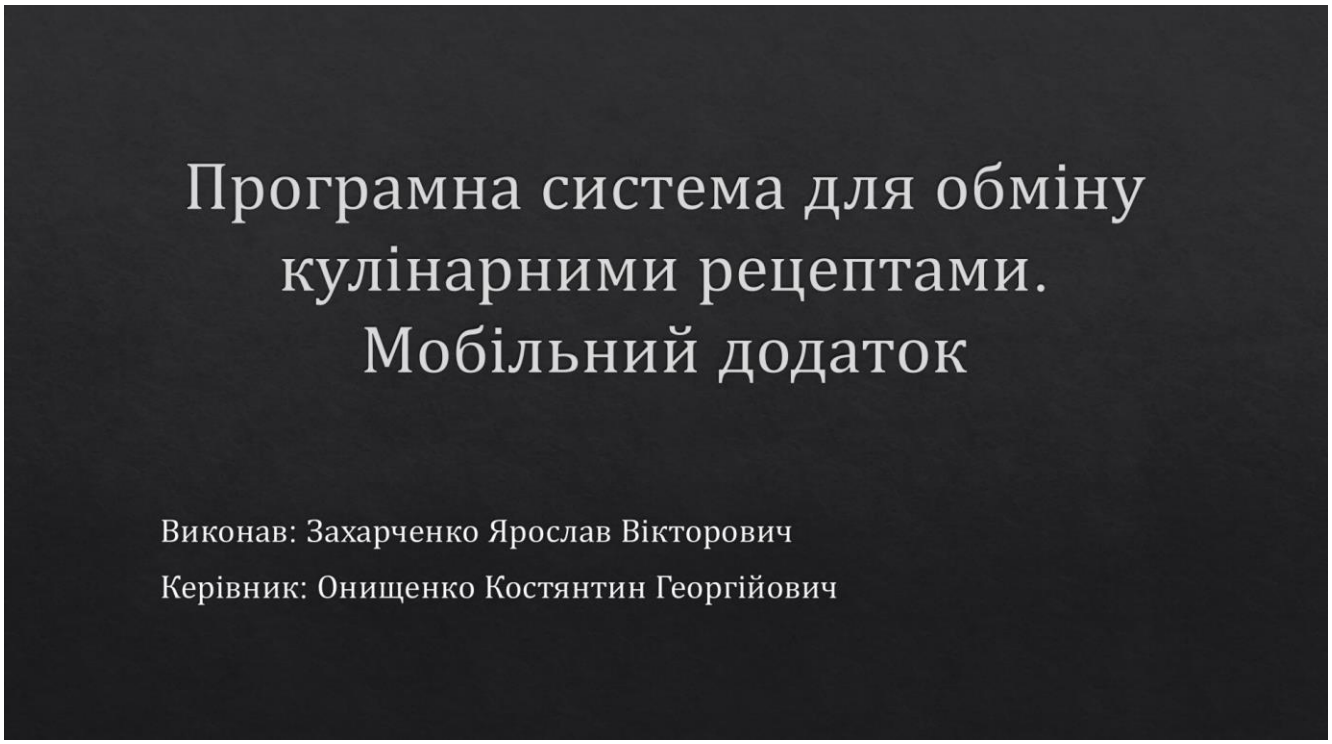

Рисунок Б.1 – Слайд 1

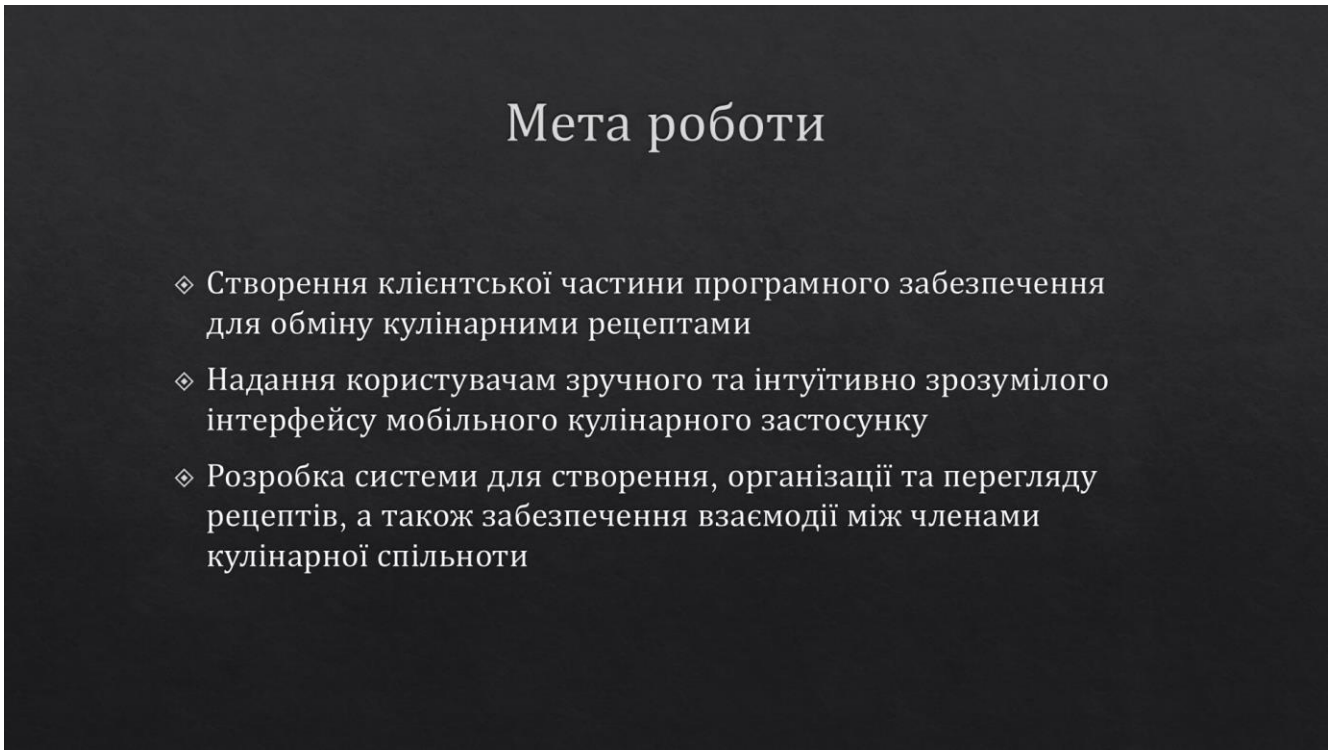

Рисунок Б.2 – Слайд 2

Рисунок Б.3 – Слайд 3

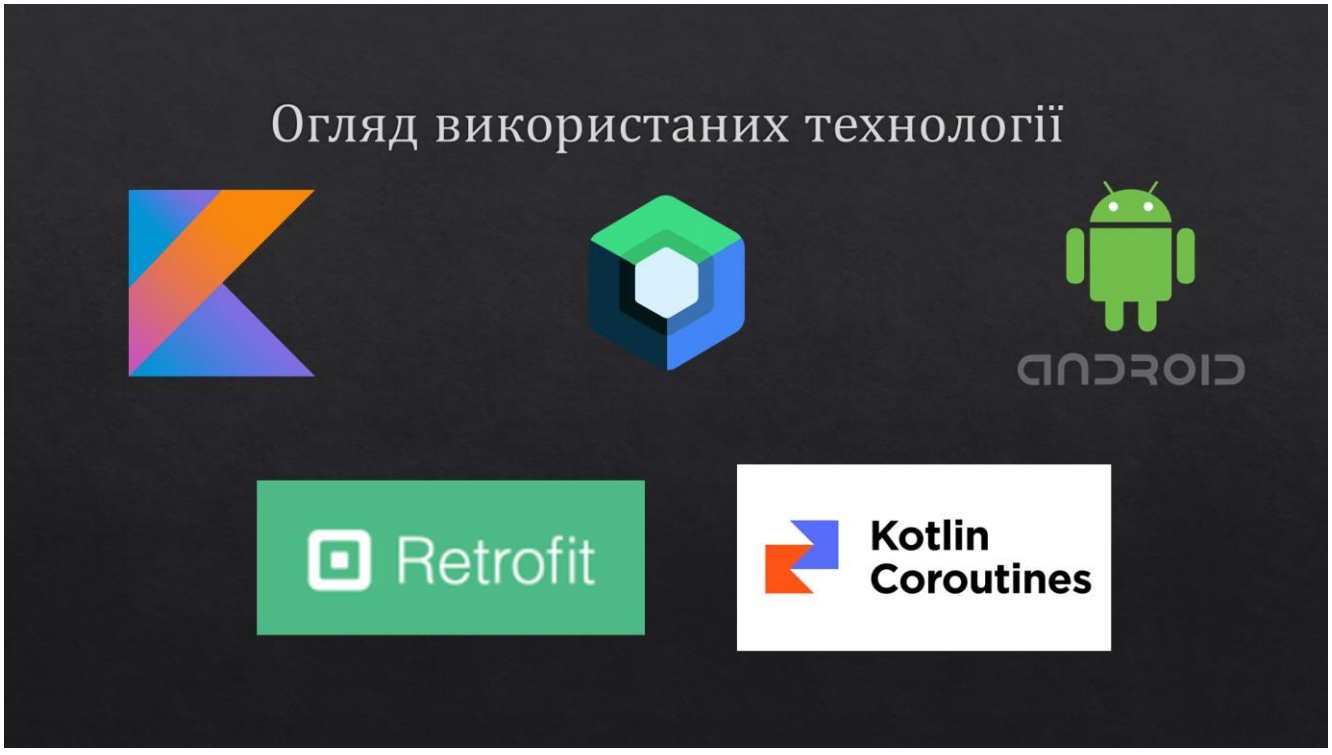

Рисунок Б.4 – Слайд 4

Рисунок Б.5 – Слайд 5

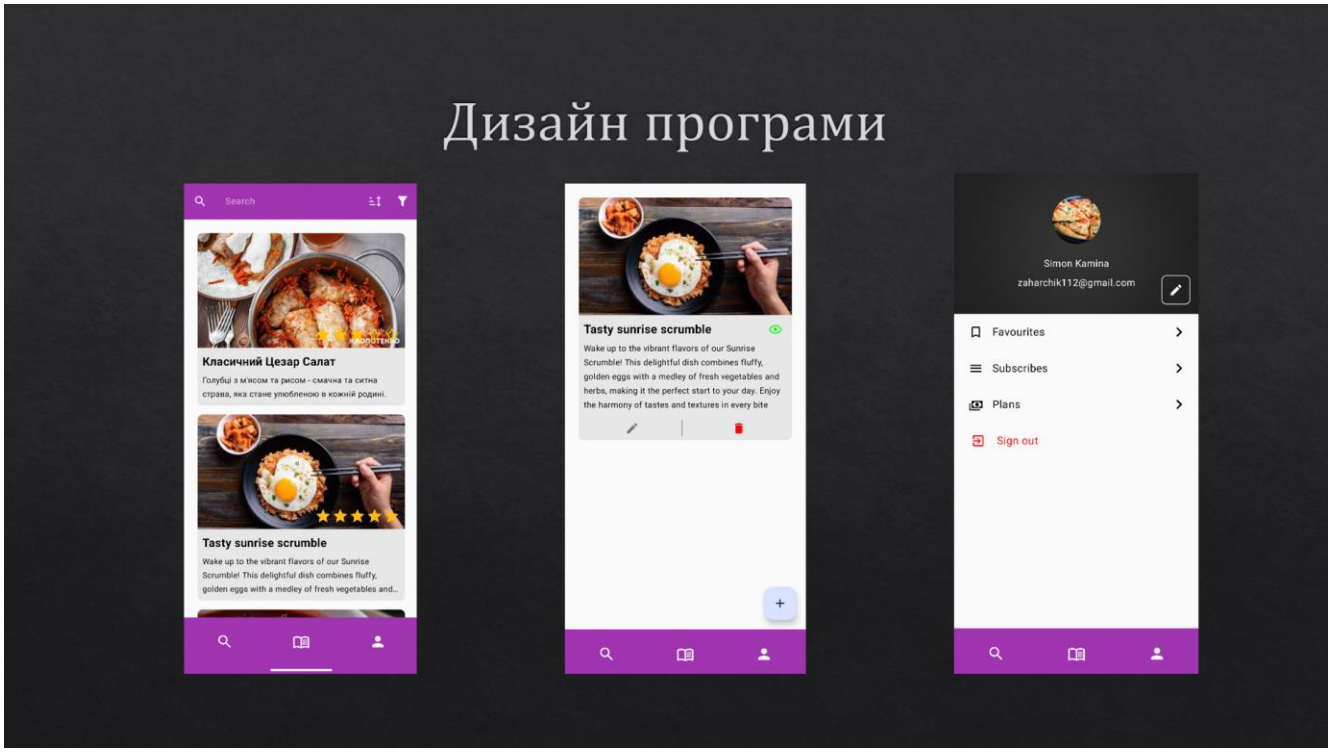

Рисунок Б.6 – Слайд 6

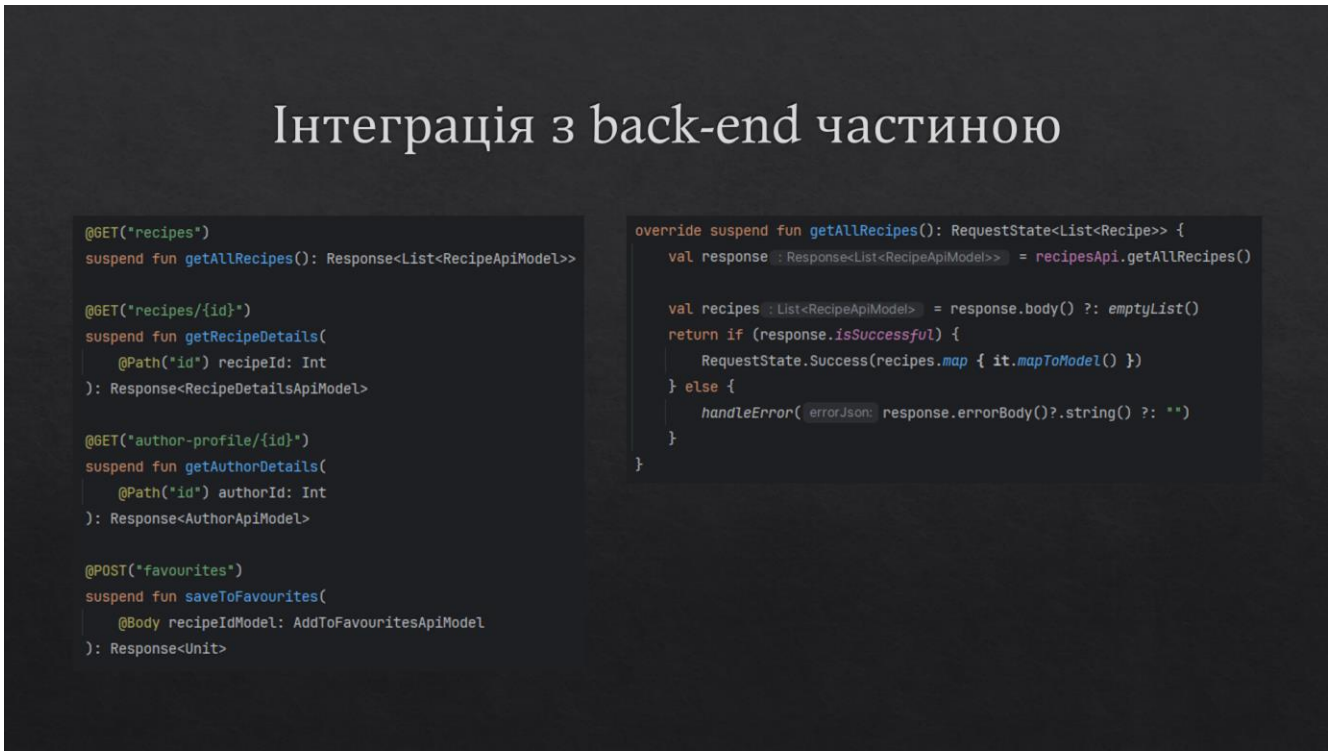

Рисунок Б.7 – Слайд 7

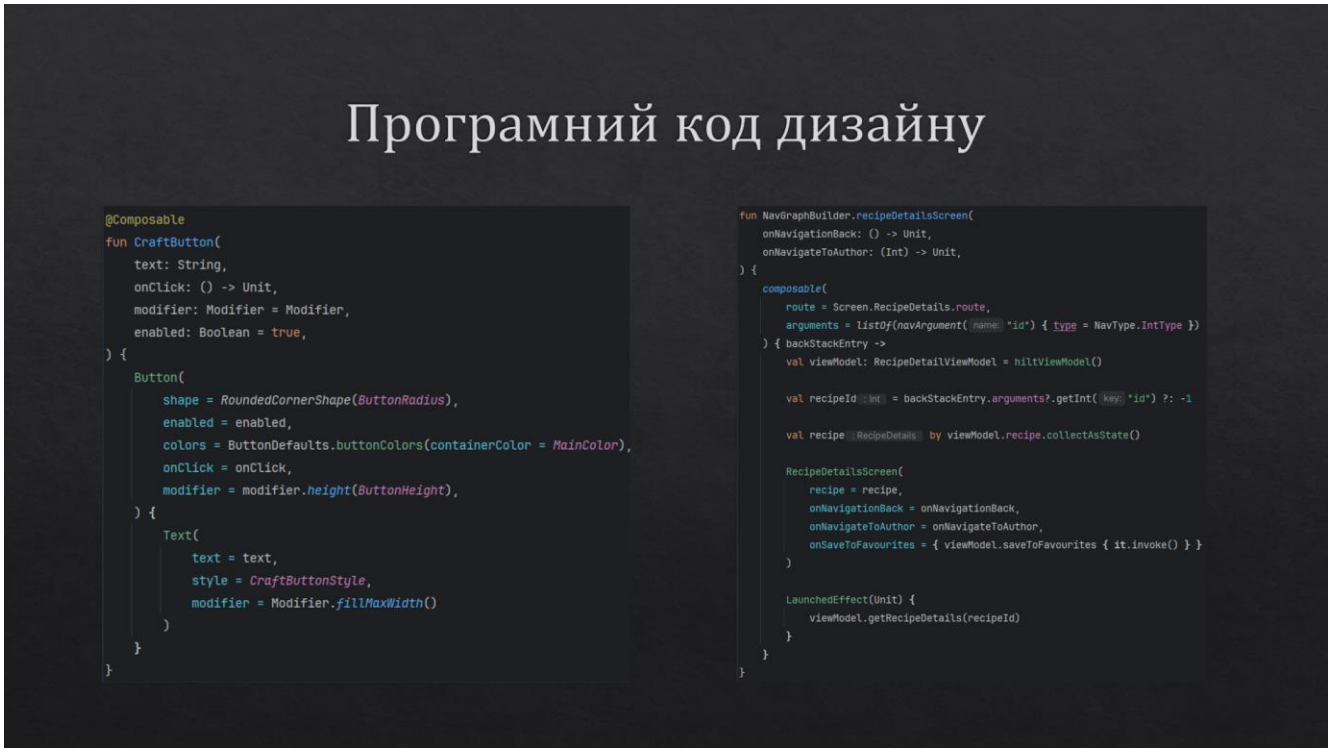

Рисунок Б.8 – Слайд 8

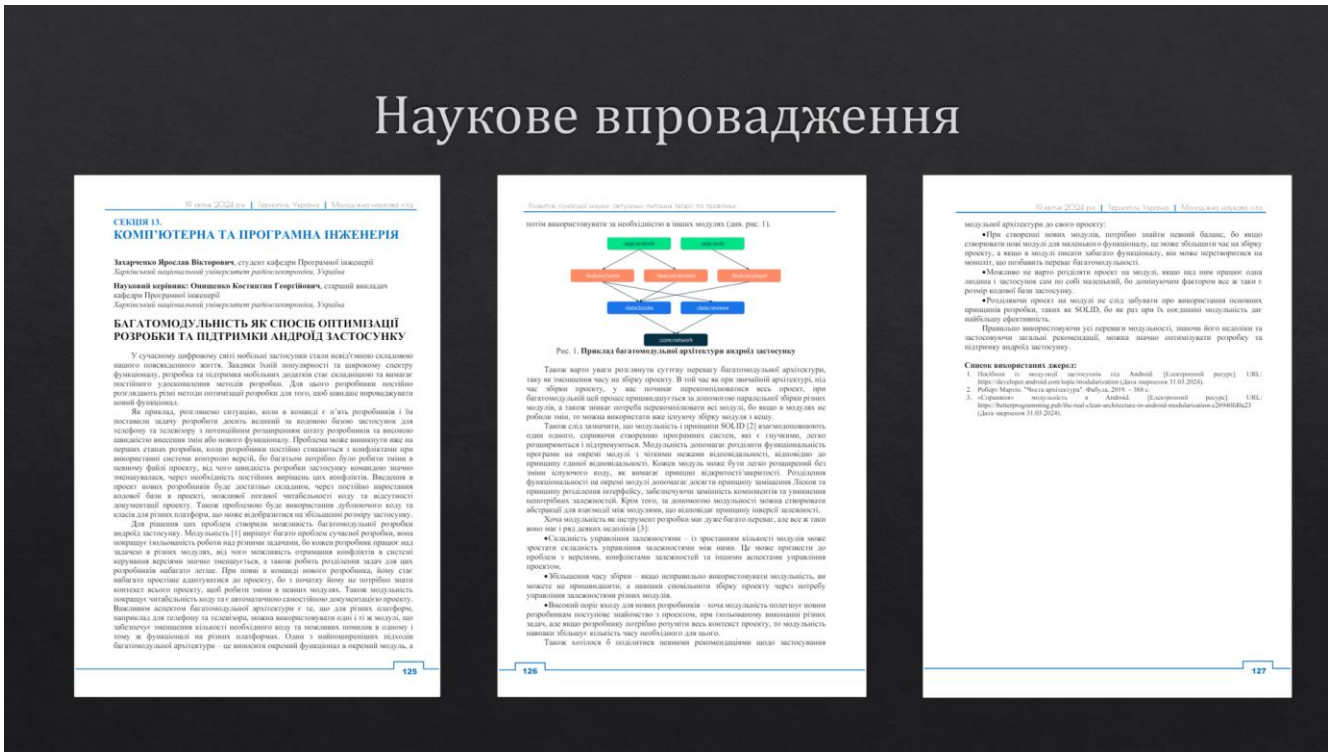

Рисунок Б.9 – Слайд 9

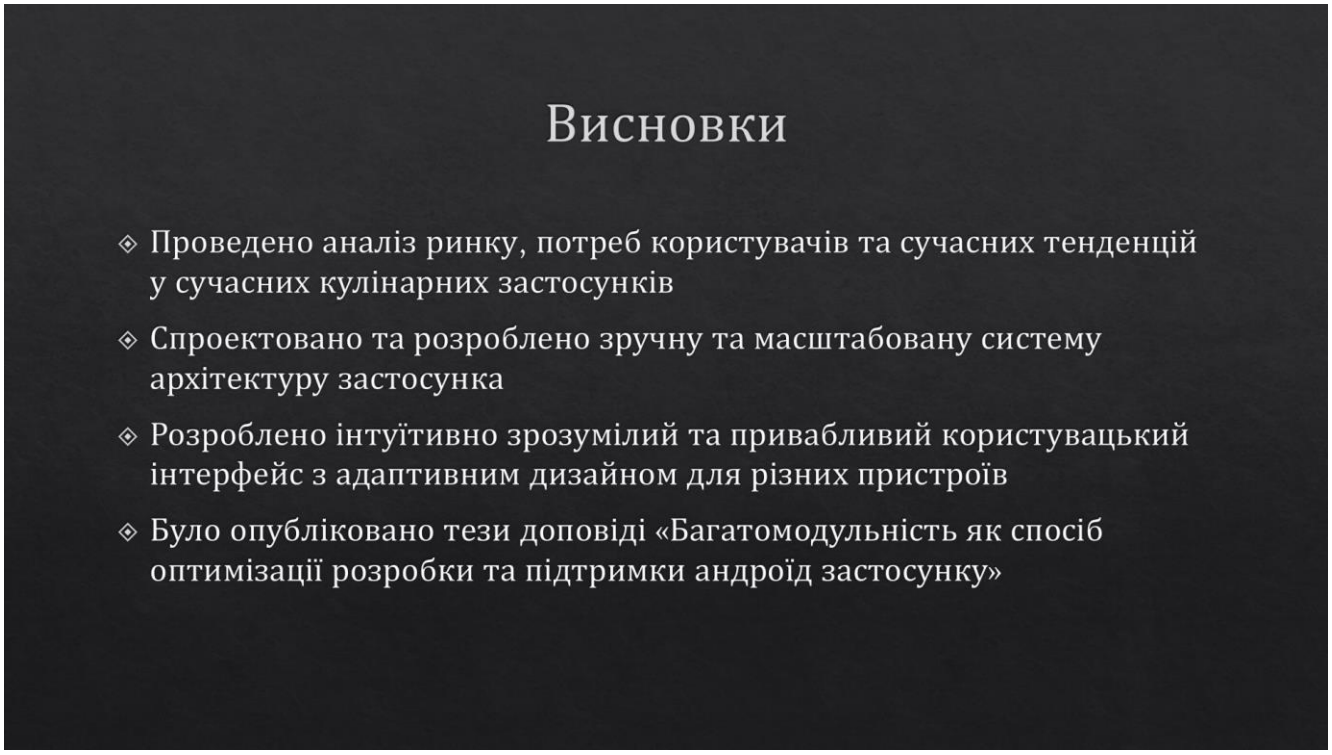

Рисунок Б.10 – Слайд 10

ДОДАТОК В

Тези доповіді для науково-практичної наукової конференції

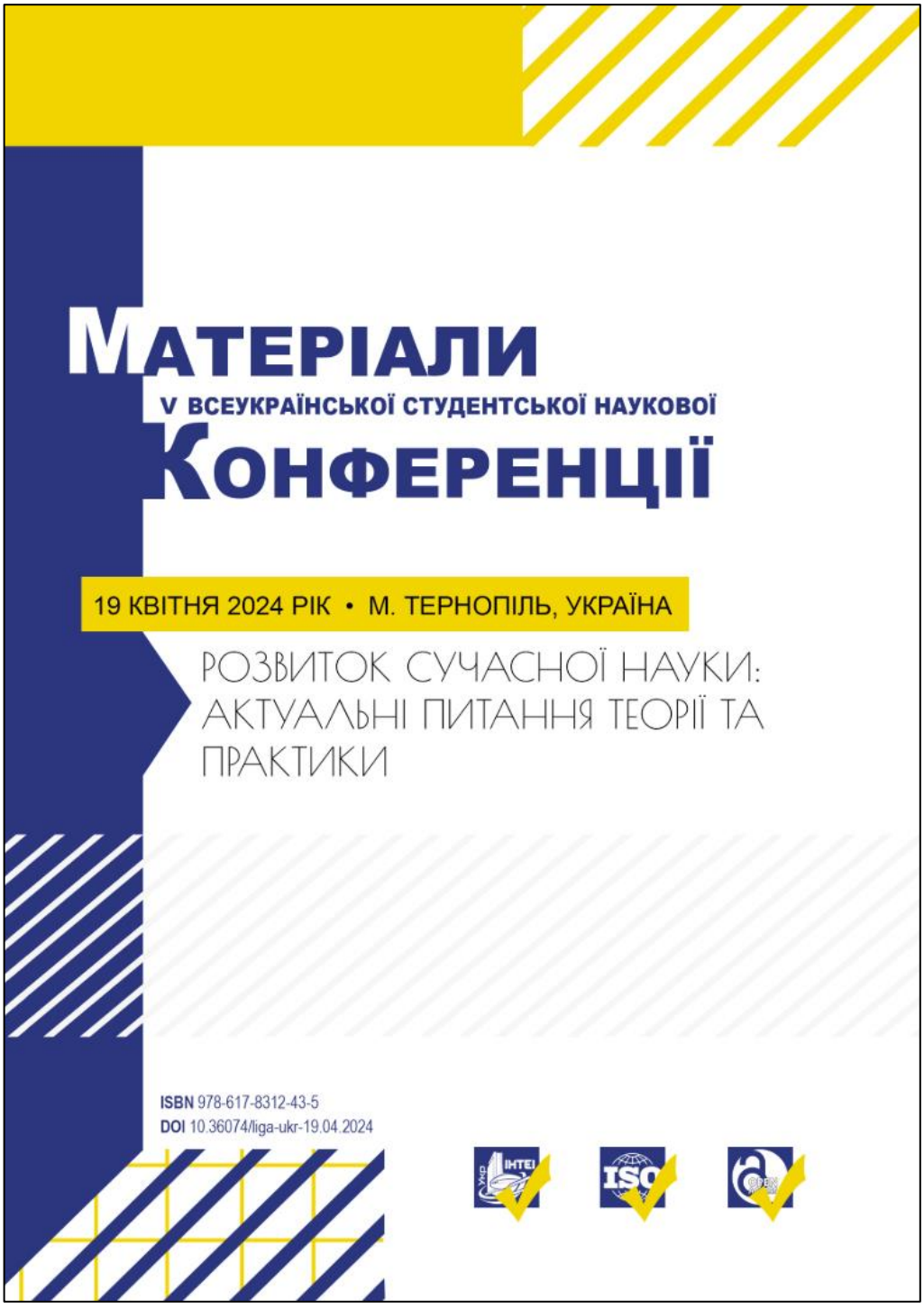

Рисунок А.1 – Обкладинка збірника

СЕКЦІЯ 13.
КОМП'ЮТЕРНА ТА ПРОГРАМНА ІНЖЕНЕРІЯ

Захарченко Ярослав Вікторович, студент кафедри Програмної інженерії
Харківський національний університет радіоелектроніки, Україна

Науковий керівник: Онищенко Костянтин Георгійович, старший викладач кафедри Програмної інженерії
Харківський національний університет радіоелектроніки, Україна

БАГАТОМОДУЛЬНІСТЬ ЯК СПОСІБ ОПТИМІЗАЦІЇ РОЗРОБКИ ТА ПІДТРИМКИ АНДРОЇД ЗАСТОСУНКУ

У сучасному цифровому світі мобільні застосунки стали невід'ємною складовою нашого повсякденного життя. Завдяки їхній популярності та широкому спектру функціоналу, розробка та підтримка мобільних додатків стає складнішою та вимагає постійного удосконалення методів розробки. Для цього розробники постійно розглядають різні методи оптимізації розробки для того, щоб швидше впроваджувати новий функціонал.

Як приклад, розглянемо ситуацію, коли в команді є п'ять розробників і їм поставили задачу розробити досить великий за кодовою базою застосунок для телефону та телевізору з потенційним розширенням штату розробників та високою швидкістю внесення змін або нового функціоналу. Проблема може виникнути вже на перших етапах розробки, коли розробники постійно стикаються з конфліктами при використанні системи контролю версій, бо багатьом потрібно було робити зміни в певному файлі проекту, від чого швидкість розробки застосунку командою значно зменшувалася, через необхідність постійних вирішень цих конфліктів. Введення в проект нових розробників буде достатньо складним, через постійно наростання кодової бази в проекті, можливої поганої читабельності коду та відсутності документації проекту. Також проблемою буде використання дублюючого коду та класів для різних платформ, що може відобразитися на збільшенні розміру застосунку.

Для рішення цих проблем створили можливість багатомодульної розробки андроїд застосунку. Модульність [1] вирішує багато проблем сучасної розробки, вона покращує ізольованість роботи над різними задачами, бо кожен розробник працює над задачею в різних модулях, від чого можливість отримання конфліктів в системі керування версіями значно зменшується, а також робить розділення задач для цих розробників набагато легше. При появі в команді нового розробника, йому стає набагато простіше адаптуватися до проекту, бо з початку йому не потрібно знати контекст всього проекту, щоб робити зміни в певних модулях. Також модульність покращує читабельність коду та є автоматичною самостійною документацією проекту. Важливим аспектом багатомодульної архітектури є те, що для різних платформ, наприклад для телефону та телевізора, можна використовувати одні і ті ж модулі, що забезпечує зменшення кількості необхідного коду та можливих помилок в одному і тому ж функціоналі на різних платформах. Один з найпоширеніших підходів багатомодульної архітектури – це виносити окремий функціонал в окремий модуль, а

Рисунок А.2 – Матеріал тез (сторінка 1)

потім використовувати за необхідністю в інших модулях (див. рис. 1).

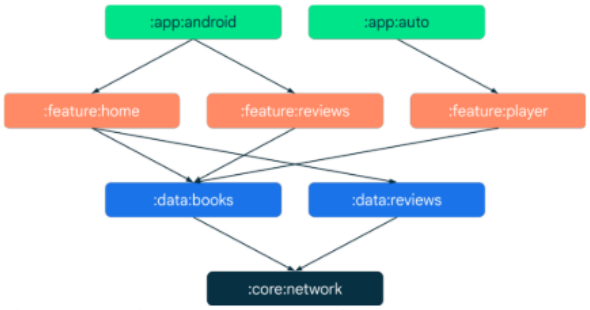

Рис. 1. **Приклад багатомодульної архітектури андроїд застосунку**

Також варто уваги розглянути суттєву перевагу багатомодульної архітектури, таку як зменшення часу на збірку проекту. В той час як при звичайній архітектурі, під час збірки проекту, у вас починає перекомпілюватися весь проект, при багатомодульній цей процес пришвидшується за допомогою паралельної збірки різних модулів, а також зникає потреба перекомпілювати всі модулі, бо якщо в модулях не робили змін, то можна використати вже існуючу збірку модуля з кешу.

Також слід зазначити, що модульність і принципи SOLID [2] взаємодоповнюють один одного, сприяючи створенню програмних систем, які є гнучкими, легко розширюються і підтримуються. Модульність допомагає розділити функціональність програми на окремі модулі з чіткими межами відповідальності, відповідно до принципу єдиної відповідальності. Кожен модуль може бути легко розширений без зміни існуючого коду, як вимагає принцип відкритості/закритості. Розділення функціональності на окремі модулі допомагає досягти принципу заміщення Лісков та принципу розділення інтерфейсу, забезпечуючи заміннність компонентів та уникнення непотрібних залежностей. Крім того, за допомогою модульності можна створювати абстракції для взаємодії між модулями, що відповідає принципу інверсії залежності.

Хоча модульність як інструмент розробки має дуже багато переваг, але все ж таки воно має і ряд деяких недоліків [3]:

- Складність управління залежностями – із зростанням кількості модулів може зростати складність управління залежностями між ними. Це може призвести до проблем з версіями, конфліктами залежностей та іншими аспектами управління проектом.
- Збільшення часу збірки – якщо неправильно використовувати модульність, ви можете не пришвидшити, а навпаки сповільнити збірку проекту через потребу управління залежностями різних модулів.
- Високий поріг входу для нових розробників – хоча модульність полегшує новим розробникам поступове знайомство з проектом, при ізольованому виконанні різних задач, але якщо розробнику потрібно розуміти весь контекст проекту, то модульність навпаки збільшує кількість часу необхідного для цього.

Також хотілося б поділитися певними рекомендаціями щодо застосування

Рисунок А.3 – Матеріал тез (сторінка 2)

модульної архітектури до свого проекту:

- При створенні нових модулів, потрібно знайти певний баланс, бо якщо створювати нові модулі для маленького функціоналу, це може збільшити час на збірку проекту, а якщо в модулі писати забагато функціоналу, він може перетворитися на моноліт, що позбавить переваг багатомодульності.
- Можливо не варто розділяти проект на модулі, якщо над ним працює одна людина і застосунок сам по собі маленький, бо домінуючим фактором все ж таки є розмір кодової бази застосунку.
- Розділяючи проект на модулі не слід забувати про використання основних принципів розробки, таких як SOLID, бо як раз при їх поєднанні модульність дає найбільшу ефективність.

Правильно використовуючи усі переваги модульності, знаючи його недоліки та застосовуючи загальні рекомендації, можна значно оптимізувати розробку та підтримку андроїд застосунку.

Список використаних джерел:

1. Посібник із модуляції застосунків під Android. [Електронний ресурс]. URL: https://developer.android.com/topic/modularization (Дата звернення 31.03.2024).
2. Роберт Мартін. "Чиста архітектура". Фабула, 2019. – 368 с.
3. «Справжня» модульність в Android. [Електронний ресурс]. URL: https://betterprogramming.pub/the-real-clean-architecture-in-android-modularization-e26940fd0a23 (Дата звернення 31.03.2024).

Рисунок А.4 – Матеріал тез (сторінка 3)

ДОДАТОК Г

Фрагменти коду програми

```
@Composable
fun CraftTextField(
    value: String,
    placeholder: String,
    onValueChange: (String) -> Unit,
    modifier: Modifier = Modifier,
    keyboardType: KeyboardType = KeyboardType.Text,
    isError: Boolean = false,
    errorText: String = "",
    enabled: Boolean = true,
) {
    TextField(
        value = value,
        shape = RoundedCornerShape(TextFieldRadius),
        colors = TextFieldDefaults.colors(
            focusedContainerColor = TextFieldBackground,
            unfocusedContainerColor = TextFieldBackground,
            unfocusedPlaceholderColor = LabelTextColor,
            disabledPlaceholderColor = LabelTextColor,
            disabledContainerColor = TextFieldBackground,
            cursorColor = MainColor,
            focusedIndicatorColor = MainColor,
            disabledIndicatorColor = Color.Transparent,
            selectionColors = TextSelectionColors(MainColor,
MainColor.copy(alpha = 0.4f)),
        ),
        enabled = enabled,
        textStyle = CraftTextFieldStyle,
        keyboardOptions = KeyboardOptions(keyboardType = keyboardType),
        visualTransformation = if (keyboardType == KeyboardType.Password)
PasswordVisualTransformation() else VisualTransformation.None,
        onValueChange = onValueChange,
        modifier = modifier.height(if (isError) TextFieldErrorHeight else
TextFieldHeight),
        isError = isError,
        placeholder = {
            Text(text = placeholder)
        },
        supportingText = supportingTextError(isError, errorText),
        singleLine = true,
    )
}

interface AuthenticationApi {

    @POST("auth/register")
    suspend fun signUp(
        @Body model: RegisterApiModel,
    ): Response<AuthenticationResponse>

    @POST("auth/login")
```

```
    suspend fun signIn(
        @Body model: LoginApiModel,
    ): Response<AuthenticationResponse>
}

class AuthenticationRepositoryImpl @Inject constructor(
    private val authApi: AuthenticationApi,
    private val tokenInterceptor: TokenInterceptor,
    private val dsRepository: DataStoreRepository,
) : AuthenticationRepository {

    override suspend fun signUp(model: RegisterModel): RequestState<String>
{
        val response = authApi.signUp(model.mapToRegisterApiModel())

        val token = response.body()?.token

        return if (response.isSuccessful && token != null) {
            tokenInterceptor.updateToken(token)
            dsRepository.saveToken(token)
            RequestState.Success(token)
        } else {
            handleError(response.errorBody()?.string() ?: "")
        }
    }

    override suspend fun signIn(model: LoginModel): RequestState<String> {
        val response = authApi.signIn(model.mapToLoginApiModel())

        val token = response.body()?.token

        return if (response.isSuccessful && token != null) {
            tokenInterceptor.updateToken(token)
            dsRepository.saveToken(token)
            RequestState.Success(token)
        } else {
            handleError(response.errorBody()?.string() ?: "")
        }
    }
}

@HiltViewModel
class SignInViewModel @Inject constructor(
    private val authRepository: AuthenticationRepositoryImpl,
) : ViewModel() {

    private val _signInUiState =
        MutableStateFlow(SignInUiModel("", ""))
    val signInUiState = _signInUiState.asStateFlow()

    private val _requestState: MutableStateFlow<RequestState<String>> =
        MutableStateFlow(RequestState.Idle)
    val requestState = _requestState.asStateFlow()

    fun updateEmail(email: String) {
        _signInUiState.value = signInUiState.value.copy(email = email)
    }
```

```
    fun updatePassword(password: String) {
        _signInUiState.value = signInUiState.value.copy(password =
password)
    }

    fun signIn() {
        viewModelScope.launch(Dispatchers.IO) {
            _requestState.value = RequestState.Loading
            _requestState.value = authRepository.signIn(
                LoginModel(
                    email = signInUiState.value.email,
                    password = signInUiState.value.password,
                )
            )
        }
    }
}
```