Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет _ інформаційних радіотехнологій та технічного захисту інформації_
(повна назва)

Кафедра __ медіаінженерії та інформаційних радіоелектронних систем_____
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти _____другий (магістерський)_____
_Розробка прототипу мобільного застосунку з віддаленим збереженням даних.
_____
(тема)

Виконав:
студент _2__ курсу, групи _СТМм-22-1_
_____Костянтин _КУТЬКО_____
(прізвище, ініціали)

Спеціальність 171 Електроніка_____
_____
(код і повна назва спеціальності)
Тип програми ___освітньо-професійна____
Освітня програма ____Електронні системи
і комп'ютерні засоби мультимедіа____
(повна назва освітньої програми)

Керівник __ст.викл. Костянтин КОЛІСНИК
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ ___Володимир КАРТАШОВ__
(підпис)

2024 р.

Харківський національний університет радіоелектроніки

Факультет інформаційних радіотехнологій та технічного захисту інформації
Кафедра медіаінженерії та інформаційних радіоелектронних систем

Рівень вищої освіти другий (магістерський)

Спеціальність 171 Електроніка
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Електронні системи і комп'ютерні засоби мультимедіа
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Кутько Костянтину Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка прототипу мобільного застосунку з віддаленим збереженням даних.

затверджена наказом по університету від "_20_"_____11_____ 2023 р. № ___1371 Ст____

2. Термін подання студентом роботи до екзаменаційної комісії _____08.01.2024 р._____

3. Вихідні дані до роботи Провести аналітичний огляд методів зберігання даних користувачів в контексті мобільних додатків, розлянувши основні методи збереження даних мобільних додатків та аналогічні застосунки. Розглянути технологічний стек мобільної розробки, а саме процес визначення мови програмування, вибір патерну проектування мобільного додатку, а також вибір бібліотек для розробки мобільного додатку. Написати прототип програми для віддаленого зберігання даних за наступними етапами: налаштування проекту, написання базових класів, створення локальної бд, створення dependency injection, створення main екрану додатку, створення послідовності екранів з додавання елементів до бази даних, створення активності для обробки get запитів.

4. Перелік питань, що потрібно опрацювати в роботі _____
__Вступ_____
__1. Аналітичний огляд методів зберігання даних користувачів в контексті мобільних додатків.
__2. Технологічний стек мобільної розробки_____
__3. Написання прототипу програми для віддаленого зберігання даних_____
__Висновки_____
__Перелік посилань_____
__Додатки_____

5. Перелік графічного матеріалу із зазначенням обов'язкових креслеників, схем, плакатів, комп'ютерних ілюстрацій_____
_____1. Титульний лист (1 аркуш А4).

2. Постановка задачі (1 аркуш А4).
3. Обгрунтування вибору мови програмування (1 аркуш А4).
4. MVVM архітектура (1 аркуш А4).
5. Алгоритм роботи програми (1 аркуш А4).
6. Введення даних для відправлення на сервер (1 аркуш А4).
7. Відправка даних на сервер (1 аркуш А4).
8. Отримання даних з серверу (1 аркуш А4).
9. Висновки (1 аркуш А4).

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналітичний огляд літератури	21.11.23–27.11.23	
2	Огляд методів зберігання даних	28.11.23–02.12.23	
3	Технологічний стек мобільної розробки	04.12.23–09.12.23	
4	Написання прототипу програми	11.12.23–20.12.23	
5	Обробка результатів	21.12.23-23.12.23	
6	Графічна частина роботи	24.12.23-26.12.23	
7	Перевірка керівником	27.12.23-28.12.23	
8	Перевірка на академічний плагіат	29.12.23-02.01.24	
9	Перевірка завідувачем кафедри, рецензування	03.01.24-08.01.24	

Дата видачі завдання _____20.11.2023 р._____

Студент _____ _____Костянтин КУТЬКО_____

(підпис)

Керівник роботи _____ _____Костянтин КОЛІСНИК_____

(підпис)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи: 69 сторінок, 13 рисунків, 1 таблиця, 31 джерела.

БАЗА ДАНИХ, ANDROID, МЕСЕНДЖЕР, TELEGRAM, MONGODB, MVVM, МОБІЛЬНИЙ ДОДАТОК.

Об’єкт дослідження – використання віддалених серверів для зберігання і обробки інформації, яку генерують та обмінюють користувачі мобільних додатків.

Мета роботи – розробити прототип додатку з віддаленим збереженням даних

В роботі вирішувалися ключові завдання, пов'язані з розробкою мобільного додатку для управління рецептами та стравами. Порівнювалися та аналізувалися три основних методи збереження даних: локальне збереження на пристрої, власний сервер та віддалений арендований сервер.

Аналіз ринку та аналогічних додатків (вайбер, телеграм, інстаграм) дозволив визначити найкращі практики та інноваційні рішення в сфері рецептів та кулінарії. Оптимальний вибір мови програмування Kotlin та архітектури MVVM дозволили створити продуктивний та легко розширюваний додаток.

В роботі використано багато бібліотек та фреймворків, таких як Koin для управління залежностями, OkHttp для взаємодії із сервером, та інші, що сприяють покращенню якості та продуктивності коду.

Результатом дослідження та розробки є прототип мобільного додатку, який відповідає сучасним вимогам та надає користувачеві можливість ефективно управляти своїми рецептами, а також взаємодіяти з іншими користувачами через віддалене збереження даних.

ABSTRACT

Explanatory note to the qualification work: 69 pages, 13 figures, 1 table, 31 source.

DATABASE, ANDROID, MESSENGER, TELEGRAM, MONGODB, MVVM, MOBILE APPLICATION.

The object of research is the use of remote servers for storing and processing information generated and exchanged by users of mobile applications.

Purpose - to develop a prototype application with remote data storage

The work solved the key tasks related to the development of a mobile application for managing recipes and dishes. Three main methods of data storage were compared and analyzed: local storage on the device, own server, and remote leased server.

Market analysis and similar applications (Viber, Telegram, Instagram) allowed us to identify best practices and innovative solutions in the field of recipes and cooking. The optimal choice of the Kotlin programming language and MVVM architecture allowed us to create a productive and easily extensible application.

The work uses many libraries and frameworks, such as Koin for dependency management, OkHttp for server interaction, and others that help improve the quality and performance of the code.

The result of the research and development is a prototype mobile application that meets modern requirements and provides the user with the ability to effectively manage their recipes, as well as interact with other users through remote data storage.

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API (Application Programming Interface) — Інтерфейс програмування додатків — набір правил та інструментів для взаємодії між програмами.

Data Base — Система для зберігання та організації даних, що дозволяє їх ефективне використання.

DI (Dependency Injection) — Методологія управління залежностями в програмуванні для полегшення інтеграції та тестування.

Gradle — Інструмент для автоматизації збірки та управління залежностями в проектах програмного забезпечення.

HTML (Hypertext Markup Language) — Мова розмітки для створення структурованих документів та веб-сторінок.

HTTP (Hypertext Transfer Protocol) — Протокол передачі гіпертексту, що визначає, як відбувається обмін інформацією у Всесвітній павутині.

JSON (JavaScript Object Notation) — Легковагоний формат обміну даними, заснований на синтаксисі мови JavaScript.

Model View Controller (MVC) — Архітектурний шаблон програмування, що розділяє додаток на три компоненти: модель, представлення та контролер.

Model View Presenter (MVP) — Архітектурний шаблон програмування, що визначає розділення додатка на три компоненти: модель, представлення та презентер.

Model-View-ViewModel (MVVM) — Архітектурний шаблон програмування, що розділяє додаток на три компоненти: модель, представлення та модель представлення.

URI (Uniform Resource Identifier) — Рідний ідентифікатор ресурсу, який ідентифікує засіб за його ідентифікатором чи його місцезнаходженням

ЗМІСТ

ВСТУП

Сучасний світ, насичений технологіями та інноваціями, надає нам безліч можливостей для зручного та ефективного спілкування, розваг, та роботи. У цьому контексті мобільні додатки стають невід’ємною частиною нашого повсякденного життя, допомагаючи нам вирішувати різноманітні задачі та виконувати функції, які раніше здавались неможливими.

Однією з ключових складових успіху сучасного мобільного додатку є ефективне управління та збереження даних. У цьому контексті, віддалене збереження даних виявляється надзвичайно важливою складовою, надаючи користувачам можливість зберігати та отримувати свої дані з будь-якого пристрою та місця, де є доступ до Інтернету.

Ця атестаційна робота присвячена розробці мобільного додатку, який не лише надає користувачам зручні та потужні інструменти, але й впроваджує ефективний механізм віддаленого збереження даних. Додаток буде спроектований з урахуванням сучасних стандартів дизайну та функціональності, забезпечуючи високу продуктивність та зручний користувацький інтерфейс.

У рамках цього дослідження будуть розглянуті та впроваджені найновіші підходи до розробки мобільних додатків, з використанням передових технологій програмування та архітектурних рішень. Особлива увага буде приділена забезпеченню безпеки даних, високій ефективності та можливостям впровадження нововведень у майбутньому.

Розробка та впровадження мобільних додатків з віддаленим збереженням даних є актуальним завданням в сучасному програмному ринку, спрямованому на мобільність та гнучкість використання. Цей диплом не лише демонструє наше розуміння сучасних технологій, але й ставить за мету вирішення конкретних викликів, пов'язаних із забезпеченням ефективного збереження та обробки даних у мобільних додатках.

1 АНАЛІТИЧНИЙ ОГЛЯД МЕТОДІВ ЗБЕРІГАННЯ ДАНИХ КОРИСТУВАЧІВ В КОНТЕКСТІ МОБІЛЬНИХ ДОДАТКІВ

1.1 Основні методи збереження даних мобільних додатків

В еру сучасних інформаційних технологій, коли мобільні додатки стають невід'ємною частиною нашого повсякденного життя, ефективне та безпечне зберігання даних набуває стратегічної важливості. Оперуючи в різноманітних сферах, від особистих заміток до корпоративних процесів, ці додатки потребують надійних та ефективних методів зберігання, щоб забезпечити безпеку, швидкодію та доступність даних.

Здобуття повноцінного розуміння цих методів стане ключем до розробки додатків, які не лише відповідають вимогам сучасного користувача, але й гарантують надійність та безпеку зберігання конфіденційної інформації.

1.1.1. Внутрішнє зберігання даних

Внутрішнє зберігання - це найпоширеніший метод зберігання даних мобільних додатків [1]. Дані зберігаються на пристрої користувача, як правило, у файлах або базах даних. Цей метод забезпечує швидкий доступ до даних і не вимагає підключення до Інтернету. Однак він також може бути менш безпечним, ніж інші методи, оскільки дані можуть бути втрачені або змінені, якщо пристрій буде пошкоджений або втрачений.

Внутрішнє зберігання мобільних додатків складається з двох основних компонентів:

– Пам'ять - це фізичний пристрій, який використовується для зберігання даних. Пам'ять може бути флеш-пам'яттю, твердотільним накопичувачем (SSD) або традиційним жорстким диском (HDD).

– Файлова система - це програмне забезпечення, яке використовується для організації даних на пристрої. Файлова система визначає, як дані зберігаються на пристрої та як до них можна отримати доступ.

Дані мобільних додатків, які зберігаються на внутрішньому накопичувачі, зазвичай зберігаються у файлах. Файл - це організована колекція даних, яка має ім'я та розташування. Файл може містити будь-який тип даних, включаючи текст, зображення, відео та код.

Дані мобільних додатків також можуть зберігатися в базах даних. База даних - це ретельно структурована колекція даних, яка використовується для зберігання та організації інформації. Бази даних часто використовуються для зберігання великих обсягів даних, таких як контакти, історія повідомлень та фінансова інформація.

Внутрішнє зберігання має ряд переваг, включаючи:

– Швидкий доступ до даних - дані, які зберігаються на внутрішньому накопичувачі, можуть бути доступні негайно, без необхідності підключення до Інтернету.
– Незалежність від Інтернету - дані, які зберігаються на внутрішньому накопичувачі, доступні навіть якщо пристрій не підключений до Інтернету.
– Зручність - дані, які зберігаються на внутрішньому накопичувачі, легко доступні користувачам.

Внутрішнє зберігання також має ряд недоліків, включаючи:

– Небезпека втрати даних - дані, які зберігаються на внутрішньому накопичувачі, можуть бути втрачені або пошкоджені, якщо пристрій буде пошкоджений або втрачений.
– Недостатня безпека - дані, які зберігаються на внутрішньому накопичувачі, можуть бути доступні несанкціонованим користувачам, якщо пристрій буде зламаний.
– Складність у використанні - корпорація Google з останніми оновленнями операційної системи Android все більше ускладнює доступ до файлової

системи пристрою користувача, що створює додаткові проблеми при збереженні даних на пристрої .

– Розмір даних за стосунку – при локальному збережені даних на пристрій, дані займають певне місце у файловій системі.

Безпека даних, які зберігаються на внутрішньому накопичувачі, є важливим фактором, який слід враховувати при виборі цього методу зберігання. Розробники мобільних додатків можуть підвищити безпеку своїх додатків, використовуючи такі методи:

– Шифрування - дані можуть бути зашифровані перед їх зберіганням на внутрішньому накопичувачі. Це робить їх нечитабельними для несанкціонованих користувачів.
– Захист паролем - доступ до даних може бути захищений паролем. Це допомагає запобігти несанкціонованому доступу до даних.
– Резервне копіювання - регулярне створення резервних копій даних може допомогти захистити їх від втрати.

1.1.2. Хмарне зберігання даних.

Хмарне зберігання - це метод зберігання даних на віддалених серверах, які надаються компанією, яка надає хмарні послуги [2]. Хмарні сервери розташовані в дата-центрах, які зазвичай мають високу доступність і безпеку.

Хмарне зберігання має ряд переваг перед локальним зберіганням даних. До цих переваг відносяться:

– Зручність - дані, які зберігаються в хмарі, доступні з будь-якого пристрою, підключеного до Інтернету. Це робить їх зручними для спільного використання та доступу.
– Безпека - дані, які зберігаються в хмарі, захищені від несанкціонованого доступу. Хмарні провайдери використовують різні заходи безпеки, такі як шифрування та брандмауер, для захисту даних своїх клієнтів.

– Економічність - хмарне зберігання може бути більш економічним, ніж традиційні методи зберігання даних. Хмарні провайдери часто пропонують різні тарифні плани, які підходять для різних потреб.

Хмарне зберігання використовується для зберігання різних типів даних, включаючи:

– Файли - хмарні сервіси зберігання файлів, такі як Google Drive, OneDrive і Dropbox, дозволяють користувачам зберігати фотографії, відео, документи та інші файли в хмарі.
– Дані додатків - хмарні сервіси зберігання даних додатків, такі як Amazon S3 і Microsoft Azure, дозволяють розробникам додатків зберігати дані своїх додатків в хмарі.
– Бізнес-дані - хмарні сервіси зберігання бізнес-даних, такі як Salesforce і Oracle Cloud, дозволяють підприємствам зберігати свої бізнес-дані в хмарі.

Існує два основних типи хмарних сервісів зберігання:

– Область зберігання - це віддалений сервер, який надає простір для зберігання даних. Область зберігання може використовуватися для зберігання будь-якого типу даних.
– База даних - це ретельно структурована колекція даних, яка використовується для зберігання та організації інформації. Хмарні бази даних часто використовуються для зберігання великих обсягів даних, таких як контакти, історія повідомлень та фінансова інформація.

При виборі хмарного сервісу зберігання слід враховувати такі фактори:

Тип даних, які ви хочете зберігати - деякі хмарні сервіси зберігання краще підходять для зберігання файлів, а інші - для зберігання даних додатків або бізнес-даних.

Обсяг даних, які ви хочете зберігати - деякі хмарні сервіси зберігання пропонують безкоштовний обсяг зберігання, а інші - платні тарифні плани.

Функції, які вам потрібні - деякі хмарні сервіси зберігання пропонують додаткові функції, такі як шифрування, автоматизоване резервне копіювання та спільний доступ до файлів.

Безпека даних, які зберігаються в хмарі, є важливим фактором, який слід враховувати при виборі хмарного сервісу зберігання. Хмарні провайдери використовують різні заходи безпеки, такі як шифрування та брандмауер, для захисту даних своїх клієнтів. Однак важливо також вжити заходів для захисту своїх даних, таких як використання надійних паролів і двофакторної аутентифікації.

1.1.3. Зберігання даних на власному сервері

Сховище даних на власному сервері - це метод зберігання даних на фізичному сервері, який знаходиться у власності або під управлінням організації. Власні сервери можуть бути розташовані в дата-центрі організації або в іншому місці [3].

Основна відмінність між зберіганням даних на власному сервері та хмарним зберіганням полягає в тому, що власні сервери знаходяться в локальній мережі організації, а хмарні сервери знаходяться в дата-центрах хмарного провайдера. Це означає, що власні сервери мають більший контроль над даними та їх безпекою, ніж хмарні сервери.

Сховище даних на власному сервері має ряд переваг перед хмарним зберіганням [4], включаючи:

- Більший контроль над даними - організації мають повний контроль над своїми даними, коли вони зберігаються на власних серверах. Вони можуть вибрати, де розташувати свої сервери, які типи обладнання використовувати і які заходи безпеки впровадити.
- Більша безпека даних - організації можуть впровадити більш суворі заходи безпеки для захисту своїх даних, коли вони зберігаються на власних серверах. Це включає шифрування даних, фізичну безпеку дата-центру та управління доступом.
- Більша масштабованість - організації можуть легко масштабувати свою інфраструктуру зберігання даних, коли вона розташована на власних

серверах. Це дозволяє їм легко додавати або видаляти сервери, а також збільшувати або зменшувати обсяги зберігання.

Сховище даних на власному сервері також має ряд недоліків, включаючи:

- Вартість - власні сервери можуть бути дорогими в придбанні та обслуговуванні. Це включає вартість обладнання, електроенергії, охолодження та персоналу.
- Комплексність - управління власними серверами може бути складним. Це включає такі завдання, як установка та конфігурація обладнання, підтримка операційної системи та програмного забезпечення та забезпечення безпеки даних.
- Необхідність спеціалістів - для управління власними серверами необхідні спеціалісти з комп'ютерних технологій. Це може бути складно для організацій, які не мають в штаті таких спеціалістів.

Вибір між зберіганням даних на власному сервері та хмарним зберіганням залежить від кількох факторів, включаючи:

- Види даних, які потрібно зберігати - деякі типи даних, такі як чутливі дані або критично важливі дані, можуть вимагати більшого контролю та безпеки, ніж інші типи даних.
- Обсяг даних, які потрібно зберігати - організації з великими обсягами даних можуть виявити, що хмарне зберігання є більш економічним варіантом, ніж власні сервери.
- Бюджет - хмарне зберігання може бути більш економічним варіантом, ніж власні сервери, особливо для малих і середніх підприємств.
- Компетенції - організації, які не мають в штаті спеціалістів з комп'ютерних технологій, можуть виявити, що хмарне зберігання є більш зручним варіантом, ніж власні сервери.

Сховище даних на власному сервері є хорошим вибором для організацій, які потребують повного контролю над своїми даними та їх безпекою. Однак це може бути дорогим і складним варіантом. Хмарне зберігання є хорошим вибором для організацій, які потребують зручного та економічного варіанту зберігання даних.

1.2. Огляд аналогічних застосунків

У світі мобільних додатків, де інновації та зручність визначають користувацький досвід, питання ефективного та безпечного зберігання даних стає критично важливим. У цьому розділі ми ретельно розглянемо та проаналізуємо недоліки та переваги трьох популярних мобільних додатків: Telegram, Viber та Instagram.

Кожен з цих додатків надає користувачам унікальний набір можливостей, але разом із тим вони стикаються з певними обмеженнями та викликами у сфері збереження даних. Аналізуючи ці аспекти, ми отримаємо глибше розуміння того, як кожен додаток оптимізує процеси збереження інформації та які компроміси вони роблять між зручністю використання та безпекою даних.

Враховуючи широкий спектр аспектів, від технічних характеристик до взаємодії з користувачем, ми визначимо, які саме переваги та недоліки визначають досвід використання цих додатків у сфері збереження даних, допомагаючи визначити оптимальний вибір для різних потреб та вимог.

1.2.1. Огляд Telegram

Telegram — це месенджер, який визначається не лише високою швидкістю та зручністю використання, але й розширеними можливостями зберігання та обміну різноманітними типами даних.

Методи Збереження Даних в Telegram:

Текстові Повідомлення зберігаються локально (текстові повідомлення можуть бути локально збережені на пристрої користувача для швидкого доступу та можливості перегляду навіть при відсутності мережі) та віддалено (для забезпечення синхронізації та доступу до повідомлень з інших пристроїв використовуються віддалені сервери Telegram).

Медіафайли зберігаються віддалено (Фотографії, відео та документи, відправлені через Telegram, зберігаються на серверах Telegram. Це забезпечує

зручний доступ до медіафайлів з різних пристроїв) та локально за допомогою кешування (для прискорення відображення при подальших переглядах.)

Всі дані про групові чати та канали зберігаються на серверах Telegram, що дозволяє отримати доступ до повідомлень та медіафайлів з будь-якого пристрою.

End-to-end шифрування застосовується до всіх особистих чатів, що означає, що дані локально шифруються на пристрої відправника та розшифровуються на пристрої отримувача.

Переваги Telegram:

- End-to-end Шифрування: Забезпечує високий рівень конфіденційності та безпеки особистих чатів.
- Зручність Використання: Швидка та ефективна синхронізація даних з мережею та іншими пристроями.
- Багатофункціональність: Можливість обміну різними типами даних майже без обмежень розміру.

Недоліки:

- Залежність від Серверів: Всі дані знаходяться на серверах Telegram, що може викликати проблеми безпеки у разі порушень безпеки серверів.
- Локальні Обмеження: Деякі дані, такі як історія повідомлень у групових чатах, можуть бути обмежені з огляду на великий обсяг інформації.

У цілому, методи збереження даних у Telegram підкреслюють баланс між безпекою, зручністю використання та функціональністю, роблячи цей месенджер популярним серед користувачів по всьому світу.

1.2.2. Огляд Viber

Viber — месенджер, який відомий своєю широкою функціональністю та зручністю використання. Давайте розглянемо його методи збереження даних та особливості.

Методи Збереження Даних в Viber:

Повідомлення і медіа можуть бути локально збережені на пристрої користувача, що дозволяє швидкий доступ до них. В той же час, для забезпечення синхронізації та доступу до повідомлень з інших пристроїв, дані відправляються на Google drive користувача.

Стікери та медіафайли, які користувач завантажив чи придбав у магазині, зберігаються на серверах Viber

На відміну від Telegram, Viber не зберігає листування користувачів на власних серверах, що робить синхронізацію між пристроями більш проблемною. Також користувач повинен сам піклуватися про збереження власного листування.

1.2.3. Огляд Instagram

Instagram — це популярний соціальний мережевий сервіс, спрямований на обмін фотографіями та відео. Розглянемо його методи збереження даних та особливості.

Фотографії та відео, які користувачі завантажують на Instagram, зберігаються на серверах компанії. Це дозволяє отримати доступ до контенту з різних пристроїв та в зручний спосіб поділитися ним з іншими користувачами.

Instagram також зберігає тимчасові копії фотографій та відео на пристроях користувачів для швидкого відображення контенту без повторного завантаження.

Дані про профіль, включаючи інформацію про користувача, публічні пости та підписників, зберігаються на серверах Instagram.

Деяка інформація, така як локальні налаштування та дані для підрахунку взаємодії, може бути збережена локально на пристроях користувачів.

Дані про історії та особисті повідомлення в Директі зберігаються на серверах Instagram для забезпечення доступу та синхронізації між пристроями.

Зображення та відео історій можуть також тимчасово зберігатися в кеші пристрою для швидкого відображення при повторному перегляді.

Переваги:

- Зручний Доступ: Можливість переглядати та завантажувати фотографії та відео з різних пристроїв.
- Широка Функціональність: Можливість взаємодії з різними типами контенту та взаємодія з іншими користувачами.

Недоліки:

- Залежність від Серверів: Дані розміщені на серверах Instagram, що може створювати ризики в разі порушення безпеки.
- Обмежені Локальні Опції: Обмежена можливість локального збереження контенту на пристрої користувача.

Висновки по розділу: у процесі аналізу аналогічних застосунків розглянуто три типи збереження даних: на пристрої, на віддаленому сервері та власному сервері. Кожен з цих методів має свої унікальні переваги та виклики, і їх вибір залежить від конкретного контексту та завдань додатку.

Використання віддаленого арендованого сервера для збереження даних у мобільному додатку може бути обумовлене декількома ключовими факторами, які впливають на ефективність та безпеку додатку. Основні переваги використання віддаленого арендованого сервера: зручність для користувача, безпека персональних даних та економічність. Загалом, використання віддаленого арендованого сервера дозволяє максимально оптимізувати витрати та забезпечити найвищий рівень доступності, безпеки та швидкодії для користувачів мобільного додатку.

Було проведено аналіз аналогічних мобільних застосунків, що функціонують у схожому контексті. В рамках цього дослідження ми ретельно розглянули та порівняли функціонал та особливості зберігання даних відомих додатків, що дозволить нам виявити найкращі практики та можливості для подальшого вдосконалення розроблюваного додатку.

Порівняння мобільних застосунків на ринку допоможе виявити їхні переваги та недоліки, а також визначити ті аспекти, які важливі для кінцевого користувача. Окрім того, аналіз конкурентів дозволить нам зрозуміти тенденції

ринку та визначити ті напрямки, які важливі для подальшого розвитку та вдосконалення нашого додатку.

2 ТЕХНОЛОГІЧНИЙ СТЕК МОБІЛЬНОЇ РОЗРОБКИ

2.1. Процес визначення мови програмування

У сучасному світі розробки програмного забезпечення, вибір мови програмування стає критичним етапом, що визначає якість, продуктивність та успіх розробленого продукту. Цей процес, відомий як "стратегічне визначення мови програмування", є ключовим елементом стратегічного управління в розробці програмного забезпечення.

Стратегічне визначення мови програмування — це не просто технічне рішення; це складний аналітичний процес, що враховує потреби проекту, специфіку завдань, характеристики команди розробників та цільову аудиторію продукту. Вибір мови програмування визначає не лише структуру коду, але і впливає на продуктивність, масштабованість, зручність розробки та інші важливі аспекти.

У даному контексті важливо враховувати різноманітні аспекти, такі як вартість розробки, швидкість виконання, доступність бібліотек та фреймворків, які визначають загальний успіх проекту [5]. Цей процес не лише допомагає забезпечити вибір оптимальної мови програмування для конкретного завдання, але й визначає траєкторію для подальшого успішного розвитку програмного продукту.

2.1.1. Java

Java – це офіційна мова для розробки під Android, яка підтримується Android Studio і є основою для вивчення Kotlin [6].

Слід пам'ятати, що Kotlin - це обгортка над Java. Щоб зрозуміти документацію Kotlin та отримати допомогу в процесі розробки мобільних Android застосунків, слід знати саме Java.

Основні переваги мови:

– Великий обсяг документації;
– Обширна спільнота розробників;
– Середовище Android Studio від самого початку заточене під роботу з Java;
– Наявність компетенцій у Java сприяє ефективному освоєнню мови програмування Kotlin.

2.1.2. Kotlin

Kotlin – ще одна офіційна мова, що останнім часом набирає все більшу популярність [6]. Також підтримується Android Studio, характеризується «синтаксичним цукром» і надає корутини, що спрощує асинхронну роботу.

Синтаксис Java заочно передбачає більш громіздкий код, ніж у випадку з Kotlin. Далі (рис.2.1) [6] зображено порівняння синтаксису мов програмування Java та Kotlin на прикладі створення класів:

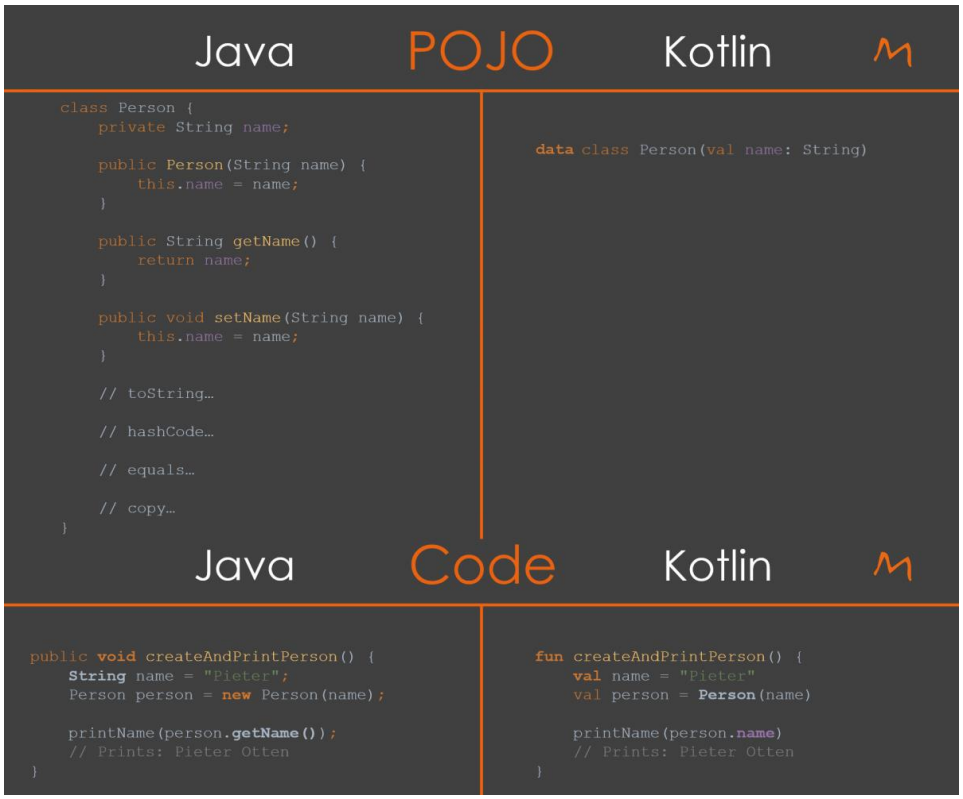

Рисунок 2.1 – порівняння синтаксису Java та Kotlin [6]

Як вбачається, Kotlin дозволяє втілити аналогічний функціонал, проте з меншою затратою часу та застосуванням скороченої кількості рядків коду.

Основні переваги Kotlin у контексті розробки під Android включають:

– Лаконічний код – Kotlin надає можливість написання компактного та зрозумілого коду, що сприяє підвищенню ефективності розробки та обслуговування програмного продукту.
– Підтримка Android Studio – Kotlin інтегрується з Android Studio, що забезпечує зручний та дружній інтерфейс для розробників Android-додатків.
– Корутини – однією з ключових переваг Kotlin є вбудована підтримка корутин, що дозволяє ефективно реалізувати асинхронні операції та полегшити роботу з багатозадачністю.

2.1.3. Python

Необхідно визнати, що розробка мобільних додатків для Android мовою програмування Python залишається високою рідкістю [5]. Здебільшого це відзначається як особистий вибір розробника або як основа для проєктів молодих та амбітних компаній.

Так, ентузіастам вдалося адаптувати одну з найпопулярніших мов програмування для розробки під Android. Це було досягнуто завдяки використанню Kivy та BeeWare. Kivy є відкритою бібліотекою для розробки кросплатформних додатків, зокрема для операційних систем Android та iOS. BeeWare представляє собою комплекс інструментів інтерфейсу користувача для створення нативних додатків для Android.

Слід зазначити, що розробник, який обирає Python для створення мобільних додатків для Android, є винятковим випадком. Це часто пов'язано з індивідуальними уподобаннями чи особливостями проєкту.

2.1.4. C/C++

Мови програмування з роду C позначаються своєю високою продуктивністю, що стає особливо важливим у випадку розробки вагомих програмних продуктів, таких як мобільні 3D-ігри [5]. Профільованість C-мов в області високоефективних операцій з пам'яттю та ресурсами надає їм перевагу у сферах, де критичне значення має оптимізація продуктивності.

Необхідно враховувати, що в контексті розробки мобільних додатків для платформи Android, використання Java є практично необхідним. Android NDK (Native Development Kit) надає можливість розробки частини програми мовами C/C++, проте велика частина додатка, включаючи інтерфейс користувача та основні функціональні елементи, повинна бути написана мовою Java.

Цей гібридний підхід дозволяє поєднати переваги C-мов у високопродуктивних обчисленнях з обов'язковою участю Java для взаємодії з Android API та створення повноцінного мобільного додатка. Такий підхід стає доречним у випадках, коли оптимальна продуктивність та широкі можливості мобільного розробництва є основними пріоритетами.

2.1.5. JavaScript

JavaScript виявляється більш життєздатним в контексті розробки для платформи Android ніж Python [5]. Зокрема, використання JavaScript стає особливо актуальним у зв'язку з фреймворком React Native, що дозволяє створювати мобільні додатки з високофункціональним мобільним інтерфейсом. Важливо відзначити, що програми, розроблені з використанням React Native, є повністю нативними, тобто вони не є мобільними веб-додатками. Це досягається завдяки використанню React Native тими ж компонентами, які використовуються в звичайних Android-додатках.

React Native та його Переваги:

- Використання React Native дозволяє створювати додатки, що повністю взаємодіють з мобільною операційною системою, забезпечуючи нативний досвід користувача.
- Завдяки перевагам JavaScript, React Native дозволяє швидко створювати функціональні та естетично збалансовані додатки.
- Додатки, розроблені з використанням React Native, можуть бути легко адаптовані для різних платформ, таких як Android та iOS, що спрощує утримання та розвиток.
- JavaScript дозволяє реалізувати більш ефективні процеси збірки порівняно із середовищем розробки Android Studio.
- Використання flexbox у JavaScript сприяє створенню високоякісного та гнучкого інтерфейсу користувача.
- JavaScript надає простий спосіб передачі даних через мережу за допомогою API, що полегшує роботу з мережевим взаємодією.

Враховуючи ці аспекти, використання JavaScript та React Native може бути перспективним вибором для розробки мобільних додатків для платформи Android.

2.1.6. Dart

Flutter представляє собою відносно нову технологію від компанії Google, яка була випущена у 2018 році та визначається як повноцінний SDK для розробки кросплатформових мобільних застосунків [5]. У порівнянні з React Native, Flutter вважається більш вдалим рішенням за рядом параметрів. Його потужність полягає в здатності розробляти кросплатформові застосунки, що спрощує ефективну розробку додатків для Android із загальною базою коду, написаного мовою Dart.

Мова Dart позиціонується як альтернатива JavaScript. Однією з його ключових переваг є те, що вона компілюється в бінарний код, що забезпечує високу швидкість виконання операцій. На відміну від використання XML для

опису інтерфейсу, Dart використовує концепцію так званих "дерев макетів" для більш ефективного та зручного визначення вигляду додатків.

Основні Переваги Dart:

- Мова Dart легко освоюється для розробників, які вже володіють мовою програмування Java.
- Завдяки компіляції в бінарний код, Dart демонструє високу продуктивність виконання програм.
- У складі Flutter існує механізм Hot Reload, що забезпечує швидке перезавантаження додатка зі збереженням його поточного стану. Це робить процес розробки більш ефективним та гнучким.

Flutter разом із мовою Dart представляє собою потужний інструментарій для розробки мобільних додатків, і його популярність зростає завдяки його кросплатформенним можливостям та ефективності у вигляді гнучкого та продуктивного SDK.

2.1.7. C#

Програмування для операційної системи Android за допомогою мови програмування C# передбачає використання платформи Xamarin, що дозволяє розробникам створювати загальну логіку застосунку, написаного мовою C#, для обох популярних мобільних платформ – Android [5]. Це забезпечує кросплатформенність та спрощує утримання кодовой бази для обох платформ.

Проте, схоже до ситуації з використанням мови програмування Python для Android-розробки, вибір C# для розробки додатків для Android залишається високою рідкістю та частково визначається індивідуальними уподобаннями розробника. В основному, це рішення може бути привабливим для тих, хто вже володіє мовою програмування C# та бажає розширити свої навички на полі мобільної розробки.

Налаштування для розробки мобільних додатків на C# може виявитися доречним для програмістів, які знаходяться в зоні комфорту з даною мовою програмування та бажають використати її для створення кросплатформових додатків для Android

2.2. Вибір патерну проектування мобільного додатку

З огляду на мету зменшення трудовитрат на розробку складного програмного забезпечення, припустимо, що необхідно використовувати готові уніфіковані рішення. Адже шаблонність дій полегшує комунікацію між розробниками, дає змогу посилатися на відомі конструкції, знижує кількість помилок [7].

За словами Вікіпедії, патерн (англ. design pattern) - повторювана архітектурна конструкція, що являє собою розв'язання проблеми проєктування в межах деякого контексту, що часто виникає.

Почнемо з першого головного - Model-View-Controller. MVC - це фундаментальний патерн, який знайшов застосування в багатьох технологіях, дав розвиток новим технологіям і щодня полегшує життя розробникам.

Уперше патерн MVC з'явився в мові SmallTalk. Розробники мали придумати архітектурне рішення, яке давало б змогу відокремити графічний інтерфейс від бізнес-логіки, а бізнес-логіку від даних. Таким чином, у класичному варіанті, MVC складається з трьох частин, які й дали йому назву.

Під Model зазвичай розуміють частину, що містить у собі функціональну бізнес-логіку програми. Модель має бути повністю незалежна від інших частин продукту. Модельний шар нічого не повинен знати про елементи дизайну і про те, яким чином він буде відображатися. Досягається результат, що дає змогу змінювати представлення даних, те як вони відображаються, не чіпаючи саму Модель.

Model має такі ознаки:

– Model - це бізнес-логіка програми;

– Model володіє знаннями про себе саму і не знає про контролери та View;
– Для деяких проєктів Model - це просто шар даних (DAO, база даних, XML-файл);
– Для інших проєктів Model - це менеджер бази даних, набір об'єктів або просто логіка програми;

В обов'язки View входить відображення даних отриманих від Model. Однак, View не може безпосередньо впливати на модель. Можна говорити, що View має доступ "тільки на читання" до даних.

View має такі ознаки:

– У View реалізується відображення даних, які отримуються від моделі будь-яким способом;
– У деяких випадках, View може мати код, який реалізує деяку бізнес-логіку.

Приклади View: HTML-сторінка, WPF форма, Windows Form.

Найпоширеніші види MVC-паттерна, це:

– Model-View-Controller
– Model-View-Presenter
– Model-View-View Model

Розглянемо та порівняємо кожен із них.

2.2.1. Model-View-Presenter

Цей підхід дає змогу створювати абстракцію View. Для цього необхідно виділити інтерфейс View з певним набором властивостей і методів. Презентер, своєю чергою, отримує посилання на реалізацію інтерфейсу, підписується на події View і за запитом змінює модель [9].

Ознаки презентера:

– Двостороння комунікація з View;
– View взаємодіє безпосередньо з презентером, шляхом виклику відповідних функцій або подій екземпляра презентера;

– Презентер взаємодіє з View шляхом використання спеціального інтерфейсу, реалізованого View;
– Один екземпляр презентера пов'язаний з одним відображенням.

Кожне View має реалізовувати відповідний інтерфейс. Інтерфейс View визначає набір функцій і подій, необхідних для взаємодії з користувачем. Презентер повинен мати посилання на реалізацію відповідного інтерфейсу, яке зазвичай передають у конструкторі.

Логіка View повинна мати посилання на екземпляр презентера. Усі події View передаються для опрацювання в презентер і практично ніколи не опрацьовуються логікою View (у т.ч. створення інших подань).

Схему роботи Model-View-Presenter зображено далі (рис.2.2) [9]

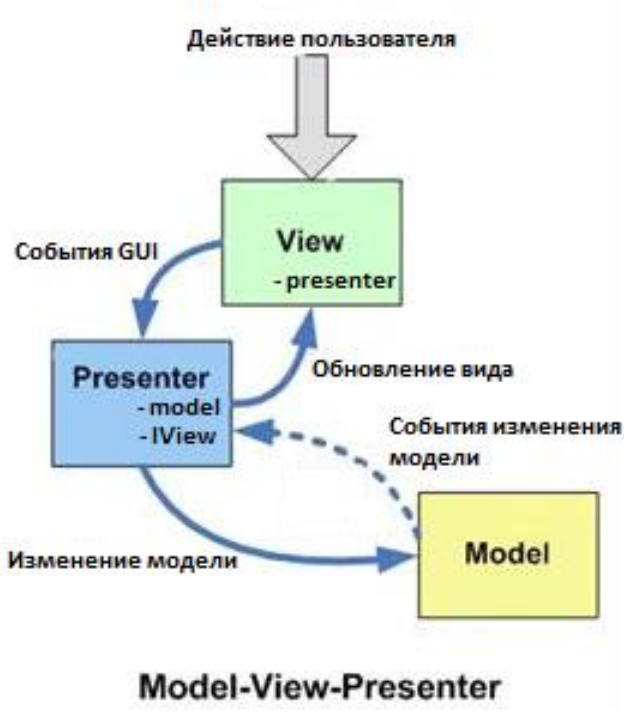

Рисунок 2.2 – Model-View-Presenter [9]

2.2.2. Model-View-View Model

Цей підхід дає змогу пов'язувати елементи View з властивостями і подіями View-моделі [10]. Можна стверджувати, що кожен шар цього патерну не знає про

існування іншого шару. Схему роботи Model-View-Presenter зображено далі (рис.2.3) [11]

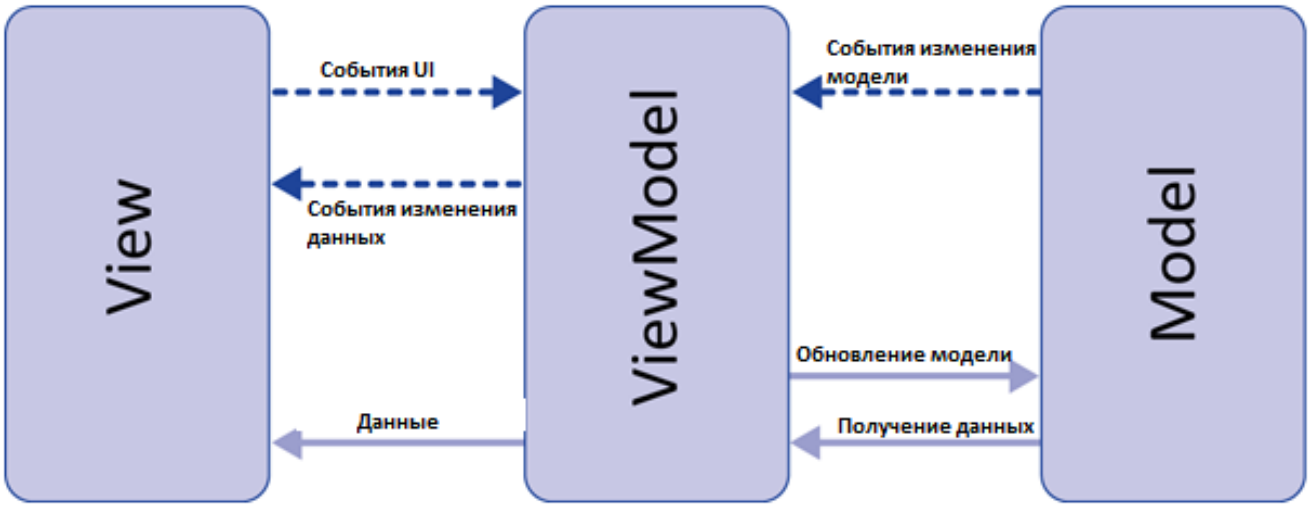

Рисунок 2.3 – Model-View-ViewModel [11]

Ознаки View-моделі:

Двостороння комунікація з View;

- View-модель - це абстракція View. Зазвичай означає, що властивості View збігаються з властивостями View-моделі / моделі
- View-модель не має посилання на інтерфейс View (IView). Зміна стану View-моделі автоматично змінює View і навпаки, оскільки використовується механізм зв'язування даних (Bindings)
- Один екземпляр View-моделі пов'язаний з одним відображенням.

При використанні цього патерну, View не реалізує відповідний інтерфейс (IView).

View повинно мати посилання на джерело даних (DataContex), яким у цьому випадку є View-модель. Елементи View пов'язані (Bind) з відповідними властивостями та подіями View-моделі.

Своєю чергою, View-модель реалізує спеціальний інтерфейс, який використовується для автоматичного оновлення елементів View.

2.2.3. Model-View-Controller

Основна ідея цього патерну в тому, що і контролер, і View залежать від моделі, але модель ніяк не залежить від цих двох компонент [8]. Схему роботи Model-View-Controller зображено далі (рис.2.4).

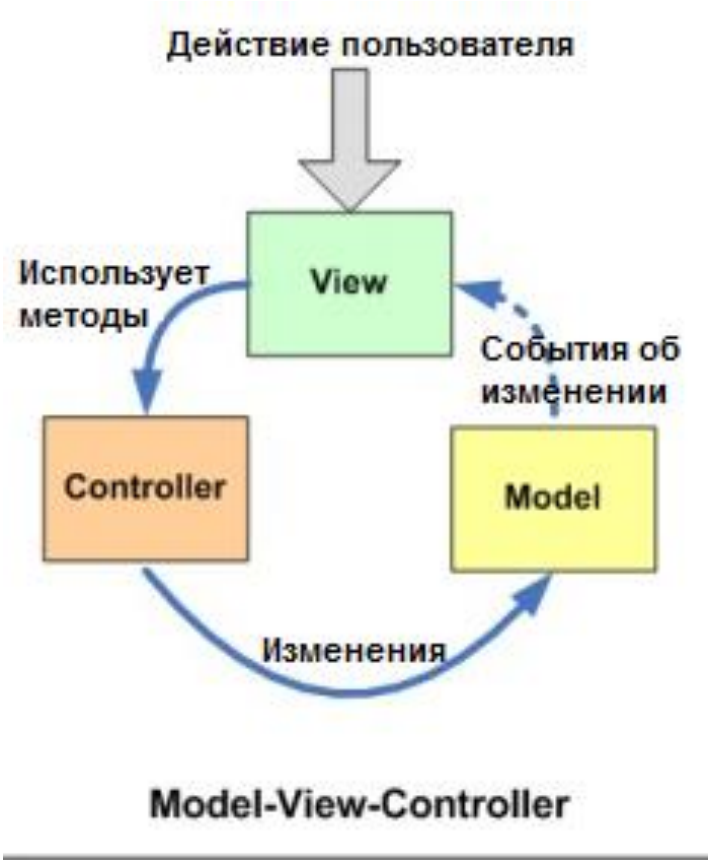

Рисунок 2.4 – Model-View-Controller [8]

Ознаки контролера:

- Контролер визначає, яке View має бути відображено в даний момент;
- Події View можуть вплинути тільки на контролер, контролер може вплинути на модель і визначити інше View.
- Можливо кілька View тільки для одного контролера;

Контролер перехоплює подію ззовні і відповідно до закладеної в нього логіки, реагує на цю подію, змінюючи Модель, за допомогою виклику відповідного методу. Після зміни Модель використовує подію про те, що вона

змінилася, і всі підписані на цю подію View, отримавши її, звертаються до Моделі за оновленими даними, після чого їх і відображають.

Реалізація MVVM і MVP-патернів, на перший погляд, має досить простий схожий вигляд. Однак, для MVVM зв'язування View з View-моделлю здійснюється автоматично, а для MVP - необхідно програмувати

MVC, мабуть, має більше можливостей з управління View.

Загальні правила вибору патерну:

MVVM – використовується в ситуації, коли можливе зв'язування даних без необхідності введення спеціальних інтерфейсів View (тобто відсутня необхідність реалізовувати IView);

MVP – використовується в ситуації, коли неможливе зв'язування даних (не можна використовувати Binding);

MVC – використовується в ситуації, коли зв'язок між View та іншими частинами додатка неможливий (і Ви не можете використовувати MVVM або MVP);

2.3. Вибір бібліотек для розробки мобільного додатку

Вибір бібліотек для мобільної розробки є стратегічно важливим завданням, оскільки це визначає майбутні можливості та характеристики розроблюваного додатку. У даному розділі буде розглянуто різноманіття бібліотек, враховуючи їхні переваги та обмеження, з метою надання обґрунтованої бази для вибору оптимального інструментарію для конкретного мобільного проекту.

2.3.1. MongoDB Realm

Realm - це інтегрована база даних для мобільних додатків, яка забезпечує високу продуктивність, масштабованість і надійність. Realm є нативною базою даних, що означає, що вона реалізована безпосередньо в платформі, для якої вона

призначена [12]. Це дозволяє уникнути додаткових витрат на копіювання даних і забезпечує високу продуктивність.

Realm також є паралельною базою даних, що означає, що вона може обробляти запити від декількох потоків одночасно. Це дозволяє масштабувати базу даних для підтримки великої кількості даних і користувачів.

Realm є реактивною базою даних, що означає, що вона може оновлюватися в режимі реального часу. Це дозволяє додаткам отримувати доступ до актуальних даних без необхідності повторної завантаження даних з бази даних.

Realm підтримує живі об'єкти, які є об'єктами, які відображають дані в базі даних. Це дозволяє додаткам працювати з даними так само, як з локальними об'єктами.

Realm має наступні ключові особливості:

- Zero copy: Realm зберігає дані в пам'яті, що дозволяє уникнути додаткових витрат на копіювання даних.
- MVCC: Realm підтримує багатоверсійну версію контролю (MVCC), яка дозволяє одночасно працювати з різними версіями даних.
- ACID: Realm підтримує атомарність, цілісність, ізольованість і довговічність (ACID), які забезпечують цілісність і надійність даних.
- Немає вбудованого механізму старіння і очищення даних: Realm не має вбудованого механізму старіння і очищення даних. Це означає, що розробникам додатків необхідно самостійно управляти цим процесом.

Realm має хорошу документацію і безліч прикладів, що полегшує розробникам початок роботи з базою даних. Компанія Realm також підтримує спільноту розробників, яка надає допомогу та підтримку.

Realm має наступні переваги:

- Висока продуктивність: Realm забезпечує високу продуктивність за рахунок використання нативних API і паралельної обробки.
- Масштабованість: Realm масштабується для підтримки великої кількості даних і користувачів.

– Реактивність: Realm може оновлюватися в режимі реального часу, що дозволяє додаткам отримувати доступ до актуальних даних.
– Живі об'єкти: Realm підтримує живі об'єкти, які дозволяють додаткам працювати з даними так само, як з локальними об'єктами.

У таблиці 2.1 наведено порівняння Realm з SQL:

Таблиця 2.1 – Порівняння Realm та SQL

Характеристика	Realm	SQL
Тип бази даних	NoSQL	Реляційна
Призначення	Мобільні додатки	Веб-додатки та інші типи додатків
Продуктивність	Висока	Середня
Масштабованість	Висока	Середня
Реактивність	Висока	Середня
Живі об'єкти	Так	Ні
Контроль над даними	Середній	Високи
Складність запитів	Середня	Висока
Документація	Хороша	Дуже хороша
Спільнота розробників	Активна	Дуже активна

2.3.2. Koin

Koin - це бібліотека ін'єкції залежностей для Kotlin, яка забезпечує простий і ефективний спосіб ін'єктувати залежності в компоненти додатка. Koin заснований на принципах інверсії контролю (IoC) і Dependency Injection (DI), які дозволяють розмежувати відповідальність за створення і ін'єкцію залежностей між компонентами додатка.

Основні особливості Koin:

– Koin має простий і лаконічне синтаксис, який полегшує розробникам ін'єктування залежностей.
– Koin забезпечує чітку і читабельну структуру коду, яка полегшує розуміння та підтримку додатка.

- Koin є ефективним способом ін'єктування залежностей, який не знижує продуктивність додатка.
- Розширюваність: Koin є розширюваною бібліотекою, яка дозволяє розробникам налаштовувати її під свої потреби.

Koin працює за допомогою інверсії контролю (IoC), яка дозволяє розмежувати відповідальність за створення і ін'єкцію залежностей між компонентами додатка. У традиційній архітектурі додатка компоненти самі створюють свої залежності. Це може призвести до дублювання коду і ускладнення підтримки додатка.

З інверсією контролю компоненти не створюють свої залежності, а отримують їх від зовнішніх джерел. Ці зовнішні джерела називаються ін'єкторами (injectors). Ін'єктори створюють і зберігають залежності, а потім ін'єктують їх у компоненти, коли це потрібно.

Koin реалізує інверсію контролю за допомогою концепції модулів (modules). Модуль - це набір залежностей, які можна ін'єктувати в компоненти. Модулі можуть бути створені вручну або за допомогою синтаксису Koin.

Для використання Koin необхідно додати бібліотеку в проект. Це можна зробити за допомогою менеджера пакетів, такого як Gradle або Maven.

Після додавання бібліотеки необхідно створити ін'єктор. Ін'єктор - це об'єкт, який відповідає за ін'єкцію залежностей. Ін'єктор можна створити вручну або за допомогою синтаксису Koin.

Наступним кроком є створення модулів. Модулі містять залежності, які можна ін'єктувати в компоненти. Модулі можна створити вручну або за допомогою синтаксису Koin.

Останнім кроком є ін'єкція залежностей у компоненти. Ін'єкція залежностей здійснюється за допомогою синтаксису Koin.

Приклад використання Koin зображено далі (рис.2.5).

```
// Створити ін'єктор
val injector = koin.createAndroidInjector()

// Створити модуль
val module = module {
    // Створити залежність
    single { MyDependency() }
}

// Ін'єкція залежностей
class MyComponent {

    // Ін'єкція залежності
    @Inject
    lateinit var myDependency: MyDependency

    fun doSomething() {
        // Використання залежності
        myDependency.doSomething()
    }
}

// Створити компонент
val myComponent = injector.inject(MyComponent::class.java)

// Використати компонент
myComponent.doSomething()
```

Рисунок 2.5 – приклад використання Koin

У цьому прикладі ми створюємо ін'єктор, модуль і компонент. Ін'єктор відповідає за ін'єкцію залежностей у компоненти. Модуль містить залежність, яка називається MyDependency. Компонент MyComponent ін'єктує залежність MyDependency.

2.3.3. OkHttp

OkHttp - це бібліотека HTTP для Android і Java, яка забезпечує простий і ефективний спосіб здійснення HTTP-запитів. OkHttp заснований на принципах HTTP/2 і реалізує ряд функцій, які підвищують продуктивність і надійність HTTP-запитів.

Основні особливості OkHttp:

– OkHttp має простий і лаконічне синтаксис, який полегшує розробникам здійснення HTTP-запитів.
– OkHttp реалізує ряд функцій, які підвищують швидкість і надійність HTTP-запитів, таких як HTTP/2, QUIC і TLS 1.3.

– OkHttp є розширюваною бібліотекою, яка дозволяє розробникам налаштовувати її під свої потреби.

OkHttp працює за допомогою HTTP/2, який є новим стандартом HTTP, який забезпечує ряд переваг у порівнянні з попередніми версіями HTTP, такими як більш швидка передача даних, кращий контроль потоку і підтримка одночасних запитів.

OkHttp також реалізує ряд функцій, які підвищують швидкість і надійність HTTP-запитів, таких як:

– QUIC - це новий протокол транспортного рівня, який забезпечує більш швидку передачу даних і зниження затримки, ніж TCP.
– TLS 1.3 - це новий стандарт безпеки, який забезпечує більш надійну і ефективну передачу даних.
– Автоматичне повторне підключення: OkHttp автоматично повторює підключення до сервера, якщо підключення було втрачено.
– OkHttp може кешувати відповіді сервера для підвищення продуктивності.

Для використання OkHttp необхідно додати бібліотеку в проект. Це можна зробити за допомогою менеджера пакетів, такого як Gradle або Maven.

Після додавання бібліотеки необхідно створити об'єкт-клієнт. Об'єкт-клієнт - це об'єкт, який відповідає за здійснення HTTP-запитів.

Наступним кроком є здійснення HTTP-запиту. HTTP-запит здійснюється за допомогою методів об'єкта-клієнта.

2.3.4. Kotlinx-serialization

Kotlinx-serialization - це бібліотека для сериалізації і десериалізації об'єктів Kotlin у різні формати, такі як JSON, Protobuf, CBOR і інші. Бібліотека заснована на принципах Reflectionless, що означає, що вона не використовує рефлексию для сериалізації і десериалізації об'єктів. Це дозволяє підвищити продуктивність і надійність бібліотеки.

Основні особливості kotlinx-serialization:

- Бібліотека заснована на принципах Reflectionless, що означає, що вона не використовує рефлексію для сериалізації і десериалізації об'єктів. Це дозволяє підвищити продуктивність і надійність бібліотеки.
- Бібліотека має простий і лаконічне синтаксис, який полегшує розробникам сериалізацію і десериалізацію об'єктів.
- Бібліотека забезпечує високу продуктивність сериалізації і десериалізації об'єктів.
- Бібліотека є розширюваною, що дозволяє розробникам налаштовувати її під свої потреби.

Kotlinx-serialization працює за допомогою компілятора-плагіна, який генерує код для сериалізації і десериалізації об'єктів. Компілятор-плагін аналізує код об'єктів і генерує код, який відповідає обраному формату сериалізації.

Для використання kotlinx-serialization необхідно додати бібліотеку в проект. Це можна зробити за допомогою менеджера пакетів, такого як Gradle або Maven.

Після додавання бібліотеки необхідно позначити об'єкти, які потрібно сериалізувати або десериалізувати, за допомогою анотації @Serializable.

Приклад використання kotlinx-serialization зображено далі (рис.2.6).:

```
// Оголосити об'єкт
@Serializable
data class User(val name: String, val age: Int)

// Створити об'єкт
val user = User("John Doe", 30)

// Сериалізувати об'єкт у JSON
val json = kotlinx.serialization.json.encodeToString(user)

// Десериалізувати об'єкт з JSON
val deserializedUser = kotlinx.serialization.json.decodeFromString(json)
```

Рисунок 2.6 – приклад використання kotlinx-serialization

У цьому прикладі ми оголошуємо об'єкт User, позначаємо його за допомогою анотації @Serializable і створюємо об'єкт. Потім ми сериалізуємо об'єкт у JSON і десериалізуємо його з JSON.

Висновок за розділом: вибір мови програмування – ключовий аспект при розробці мобільного додатку, і використання Kotlin для цієї мети обумовлене декількома важливими факторами:

Kotlin пропонує зручний та читабельний синтаксис, який дозволяє виразно виражати ідеї та покращує зрозумілість коду. Мова є більш компактною порівняно з Java, що сприяє прискоренню розробки та зменшенню кількості написаного коду.

Kotlin повністю сумісний із Java, що дозволяє безболісно інтегрувати Kotlin-код із існуючими Java-бібліотеками та проектами. Це значно полегшує міграцію та покращує переносимість коду.

Kotlin пропонує безпечний і надійний код завдяки усуненню багатьох можливих проблем, таких як null-посилання, завдяки введенню концепції nullable та non-nullable типів.

Мова має багато функціональних конструкцій, таких як лямбда-вирази, короткі функції та інші елементи, які дозволяють писати ефективний та сучасний код.

Вибір MVVM (Model-View-ViewModel) архітектури для розробки мобільного додатку зумовлений низкою вагомих переваг:

MVVM дозволяє чітко розділити логіку додатку на три основні компоненти: Model, View та ViewModel. Це полегшує розуміння коду, утримання та масштабування проекту.

ViewModel, яка містить логіку взаємодії між Model та View, легко тестується. Це дозволяє створювати ефективні тести та забезпечує високий рівень якості коду.

MVVM надає можливість використовувати двунапрямлений зв'язок між View та ViewModel, що дозволяє автоматично оновлювати інтерфейс користувача при зміні даних в ViewModel та навпаки.

Чітке розділення між компонентами робить код більш переносимим. ViewModel може використовуватися на різних платформах без змін, що полегшує розробку для різних мобільних платформ.

MVVM полегшує утримання проекту, оскільки кожен компонент відповідає за свою конкретну область логіки. Це робить код більш зрозумілим та готовим до розширення.

Обрання Kotlin та MVVM архітектури надає платформонезалежність, чистоту та ефективність у розробці мобільного додатку, що відповідає сучасним вимогам до розробки програмного забезпечення.

3 НАПИСАННЯ ПРОТОТИПУ ПРОГРАМИ ДЛЯ ВІДДАЛЕНОГО ЗБЕРІГАННЯ ДАНИХ

3.1. Налаштування проекту

На цьому етапі ми визначаємо залежності, конфігуруємо параметри проекту, встановлюємо бібліотеки, фреймворки та інші інструменти, необхідні для розробки програмного продукту. Алгоритм роботи програми зображено далі (рис.3.1).

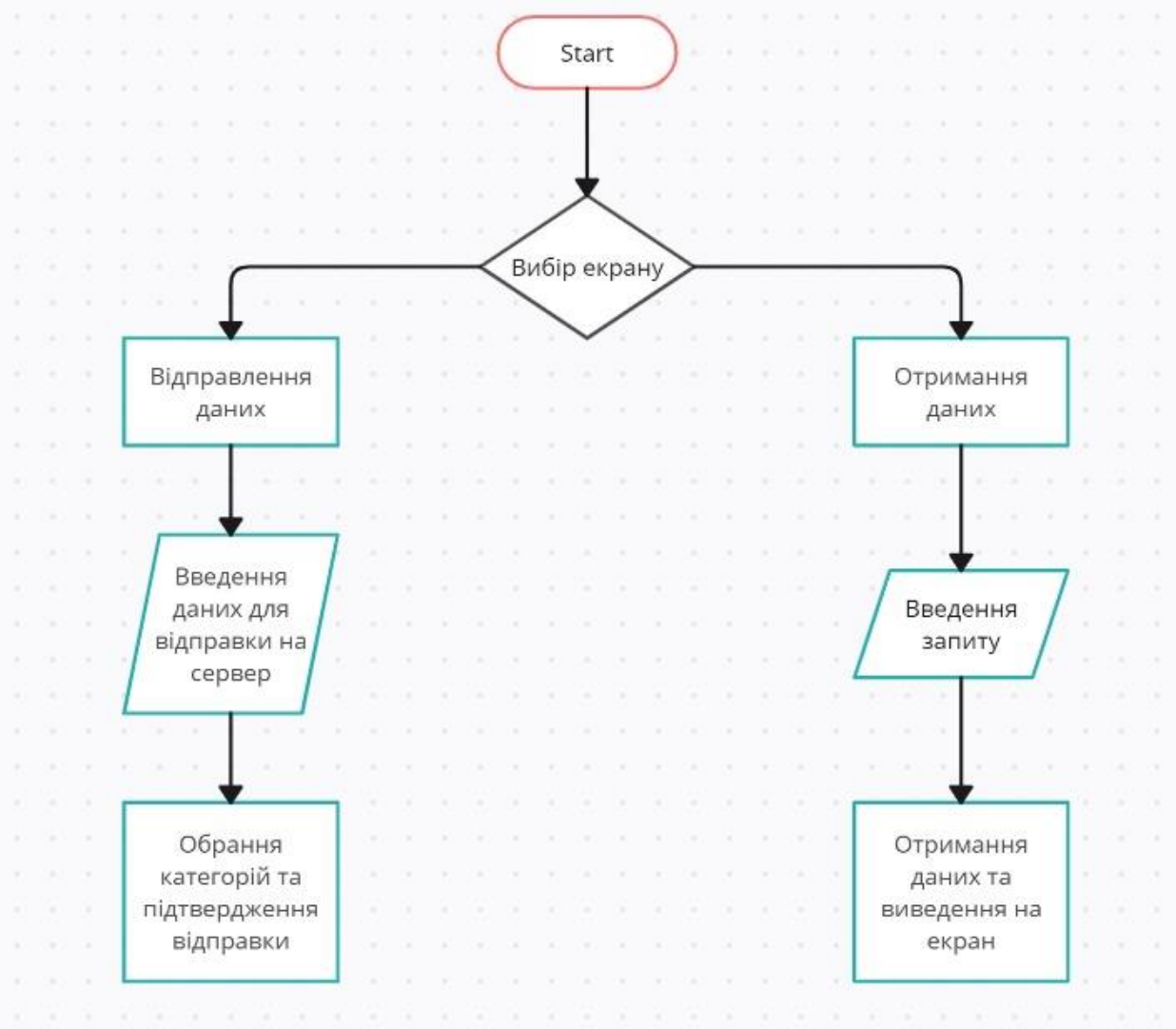

Рисунок 3.1 – Алгоритм роботи програми

Gradle - це система збірки проектів, яка використовується в багатьох Java-проектах, включаючи проекти для платформи Android. Додавання залежностей у Gradle використовується для включення зовнішніх бібліотек, фреймворків чи інших модулів до вашого проекту.

Код файлу build.graddle:

```
plugins {
    id 'com.android.application'
    id 'org.jetbrains.kotlin.android'
    id 'io.realm.kotlin'
    id 'org.jetbrains.kotlin.plugin.parcelize'
}

android {
    namespace 'com.example.organiseeatclone'
    compileSdk 33

    defaultConfig {
        applicationId "com.example.organiseeatclone"
        minSdk 24
        //minSdk 33
        targetSdk 33
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner
"androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles  getDefaultProguardFile('proguard-
android-optimize.txt'), 'proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    kotlinOptions {
```

```
            jvmTarget = '1.8'
        }
        buildFeatures {
            viewBinding true
        }

    }

    dependencies {

        //DI (KOIN)
        implementation "io.insert-koin:koin-core:3.4.2"
        implementation "io.insert-koin:koin-android:3.4.2"

        //Database (Realm)
        implementation 'io.realm.kotlin:library-base:1.10.0' //
Add to use local realm (no sync)
        implementation  'io.realm.kotlin:library-sync:1.10.0'//
Add to use Device Sync
        implementation          'org.jetbrains.kotlinx:kotlinx-
coroutines-core:1.7.0' // Add to use coroutines with the SDK

        //Preferences DataStore
        implementation           'androidx.datastore:datastore-
preferences:1.1.0-alpha04'

        //WEB
        implementation("com.squareup.okhttp3:okhttp:4.10.0")

        //Serialisation
        implementation          "org.jetbrains.kotlinx:kotlinx-
serialization-json:1.5.1"

        //Image work (Glide)
        implementation 'com.github.bumptech.glide:glide:4.14.2'

        implementation 'androidx.core:core-ktx:1.10.1'
        implementation 'androidx.appcompat:appcompat:1.6.1'
        implementation
'com.google.android.material:material:1.9.0'
        implementation
'androidx.constraintlayout:constraintlayout:2.1.4'
        testImplementation 'junit:junit:4.13.2'
```

```
        androidTestImplementation
'androidx.test.ext:junit:1.1.5'
        androidTestImplementation
'androidx.test.espresso:espresso-core:3.5.1'
    }
```

Давайте розглянемо основні елементи цього скрипта

Плагіни:

com.android.application: Плагін для розробки Android-додатків.

org.jetbrains.kotlin.android: Плагін для підтримки Kotlin в Android-проекті.

io.realm.kotlin: Плагін для підтримки Realm в Kotlin.

org.jetbrains.kotlin.plugin.parcelize: Плагін для генерації коду Parcelable в Kotlin.

Блок android:

namespace: Простір імен для пакету додатка.

compileSdk: Версія SDK, яку використовує ваш проект для компіляції.

defaultConfig: Конфігурація за замовчуванням, така як ідентифікатор додатка (applicationId), мінімальна та цільова версії SDK, версійний код та ім'я версії.

testInstrumentationRunner: Клас для виконання тестів.

Блок buildTypes:

release: Конфігурація для релізної збірки, де встановлено minifyEnabled на false (заборона зменшення розміру коду) та вказані файли ProGuard.

Блоки compileOptions, kotlinOptions, buildFeatures:

compileOptions: Налаштування сумісності версій Java.

kotlinOptions: Налаштування версії JVM для Kotlin.

buildFeatures: Включення функцій, таких як View Binding.

Блок dependencies:

io.insert-koin: Koin - фреймворк для внедрення залежностей (DI).

io.realm.kotlin: Realm - база даних для платформ Android.

androidx.datastore:datastore-preferences: DataStore - бібліотека для роботи зі збереженням даних.

com.squareup.okhttp3:okhttp: OkHttp - бібліотека для взаємодії з мережею.

org.jetbrains.kotlinx:kotlinx-serialization-json: kotlinx.serialization - бібліотека для серіалізації JSON.

com.github.bumptech.glide:glide: Glide - бібліотека для роботи з зображеннями.

Інші бібліотеки для UI, тестування та ін.

Цей Gradle-скрипт має за мету забезпечити конфігурацію та додавання необхідних залежностей для розробки Android-додатку, написаного на Kotlin.

3.2. Написання базових класів

При розробці Android-додатків дуже часто виникає потреба у створенні загальних шаблонів та функціоналу, які можна використовувати в різних частинах програми. Один із способів цього досягнення - використання базових класів. Базові класи визначають загальний функціонал, спільний для всіх підкласів, і можуть значно полегшити розробку та підтримку коду.

Використання базових класів у нашому проекті дозволяє забезпечити єдність структури та спростити розробку нових функціональностей. Ці класи стають фундаментом для всього додатку, допомагаючи зробити код більш зрозумілим, підтримуваним та масштабованим.

В нашому проекті ми визначили три ключові базові класи: BaseActivity, BaseFragment та BaseHTTPRequest, які відіграють ключову роль у структурі та функціональності нашого Android-додатка.

3.2.1 Base Activity та Base Fragment

BaseActivity відображає базовий клас для всіх активностей в нашому додатку. Він використовує View Binding для зручного доступу до елементів інтерфейсу користувача та забезпечує загальний цикл життя для всіх активностей.

Цей клас встановлює стандартний підхід до створення макетів та ініціалізації елементів інтерфейсу, що спрощує процес розробки та забезпечує консистентність у вигляді та поведінці різних екранів.

Код файлу BaseActivity.kt:

```
    abstract class BaseActivity<T : ViewBinding> : AppCompatActivity() {
        lateinit var binding: T
        override fun onCreate(savedInstanceState: Bundle?) {
            super.onCreate(savedInstanceState)
            binding = setupBinding(LayoutInflater.from(this), null)
            setContentView(binding.root)
            binding.initializeLayout()
        }
        abstract fun setupBinding(inflater: LayoutInflater, container: ViewGroup?): T
        abstract fun T.initializeLayout()
    }
```

Цей код визначає базовий клас BaseActivity, який є узагальненим класом для всіх ваших діючих додатків (Activity) і використовує View Binding для прив'язки макетів.

Клас є узагальненим і наслідується від AppCompatActivity, що дозволяє вам використовувати всі можливості бібліотеки AndroidX.

Має поле binding, яке буде використовуватися для прив'язки до елементів макету за допомогою View Binding.

Метод onCreate викликається при створенні активності, ініціалізує поле binding, викликаючи абстрактний метод setupBinding, який повинен бути реалізований в підкласах і встановлює контент відображенням кореневого елемента зв'язаного макету.

Абстрактний метод setupBinding оголошується як абстрактний, щоб вимагати його реалізацію в підкласах. Приймає LayoutInflater та ViewGroup? як параметри і повертає екземпляр ViewBinding. Викликається в методі onCreate для створення екземпляру binding.

Абстрактний метод initializeLayout оголошується як абстрактний, щоб вимагати його реалізацію в підкласах, викликається в методі onCreate після ініціалізації binding та призначений для налаштування елементів інтерфейсу користувача (UI).

BaseFragment визначає базовий клас для всіх фрагментів, що використовуються у нашому додатку. Він забезпечує спільний функціонал для управління UI елементами та взаємодії фрагментів.

Крім того, він може містити загальний код для обробки подій життя фрагменту, таких як створення, призупинення та відновлення, спрощуючи управління цими подіями в різних частинах додатка.

Важливо зазначити, що базовий фрагмент та базова активність дійсно можуть бути досить схожими в деяких випадках, особливо якщо вони спроектовані для вирішення загальних задач у Android-додатку. Часто це відбувається через те, що обидва вони взаємодіють із життєвим циклом Android і мають подібний набір завдань.

Основні схожості включають:

– Життєвий цикл: Обидва мають схожий життєвий цикл, такий як onCreate, onStart, onResume, onPause, onStop і onDestroy. Це вказує на те, що вони обидва можуть реагувати на події, пов'язані зі станом екрану та додатку.

– UI-прив'язка: Обидва можуть використовувати View Binding або інші методи для зручної роботи із UI-елементами та їхніми подіями.
– Організація коду: У вас може бути схожа структура коду, така як ініціалізація UI, налаштування подій та реакція на їх зміни.

Враховуючи ці схожості, детальний розгляд базового фрагменту не є необхідним.

3.2.2. Base HTTP request

BaseHTTPRequest є базовим класом для роботи з HTTP-запитами в нашому додатку. Він може містити загальні методи для виконання GET, POST, PUT, DELETE запитів та обробки відповідей сервера.

Цей клас спрощує комунікацію з сервером, надаючи єдиний інтерфейс для всіх HTTP-запитів у додатку і забезпечуючи обробку помилок та винятків.

Код файлу `BaseHTTPRequest`.kt:

```
abstract class BaseHTTPRequest {
    abstract val apikey: String
    abstract val url: String
    abstract val gettingFieldName: String
    val client = OkHttpClient()
    open fun getResponse(callback: (String) -> Unit) {
        val request = Request.Builder()
            .url(this.url)
            .addHeader("X-Api-Key", this.apikey)
            .build()
        client.newCall(request).enqueue(
            object : Callback {
                override fun onFailure(call: Call, e:
IOException) {
```

```
                    Log.e("error", "API failed", e)
                    callback("Something went wrong")
                }
                override fun onResponse(call: Call,
response: Response) {
                    val body = response.body?.string()
                    if (body != null) {
                        Log.v("data", body)
                    } else {
                        Log.v("data", "empty")
                    }
                    val jsonObject = JSONArray(body)
                    //val textResult =
jsonObject.getJSONObject(0).getString(gettingFieldName)
                    val textResult =
jsonObject.getJSONObject(0)
                    callback(textResult.toString())
                }
            }
        )
    }
}
```

Цей код представляє базовий клас BaseHTTPRequest для роботи з HTTP-запитами у вашому Android-додатку. Давайте розглянемо його основні елементи:

– Apikey, url і gettingFieldName є абстрактними властивостями, які повинні бути реалізовані в підкласах. Ці властивості визначають ключ API, URL для запиту та поле, з якого буде отримано результат.
– Client є екземпляром OkHttpClient, який використовується для виконання HTTP-запитів.

– Метод getResponse приймає замикання (callback), яке буде викликано з отриманим результатом, створює об'єкт Request за допомогою OkHttpClient, додає заголовок з API-ключем та викликає вказаний URL. Викликається асинхронно, використовуючи enqueue для обробки відповіді асинхронно.
– У методі onFailure обробляється випадок невдалого запиту, а в методі onResponse обробляється успішна відповідь, отримує тіло відповіді, перетворює його у JSONArray та викликає зазначене замикання з отриманим результатом (першим об'єктом JSON з масиву).

3.3. Створення локальної БД

У світі сучасної розробки Android-додатків, де зручність, швидкість та надійність грають ключову роль, обрання правильної системи управління базами даних є вирішальною. Одним із потужних інструментів, які дозволяють легко та ефективно працювати з локальними базами даних, є Realm.

Створення локальної бази даних на основі Realm в Android-додатках відкриває нові перспективи для розробників, дозволяючи швидко та безпечно зберігати та управляти даними прямо на пристрої користувача. Realm надає простий та ефективний спосіб робити операції з базою даних, забезпечуючи високий рівень продуктивності та зменшуючи обсяг коду, необхідного для роботи з базовими операціями CRUD (створення, читання, оновлення, видалення).

По-перше, розглянемо файл Database.kt:

```
class Database {
    val config = RealmConfiguration.create(schema = setOf(DishType::class, Dish::class))
    val realm: Realm = Realm.open(config)

    fun addDishType(
        name: String,
```

```
            icon: Int
        ) {
            realm.writeBlocking {
                copyToRealm(DishType().apply {
                    this.name = name
                    this.icon = icon
                })
            }
        }

        fun setDefaultDishTypes() {
            realm.writeBlocking {
                copyToRealm(DishType().apply {
                    this.name = "Salads"
                    this.icon = R.drawable.icon_typedish_salads
                })
                copyToRealm(DishType().apply {
                    this.name = "Soups"
                    this.icon = R.drawable.icon_typedish_soup
                })
                copyToRealm(DishType().apply {
                    this.name = "Breakfasts"
                    this.icon                                    =
R.drawable.item_typedish_breakfasts
                })
                copyToRealm(DishType().apply {
                    this.name = "Bird"
                    this.icon = R.drawable.item_typedish_bird
                })
```

```
                copyToRealm(DishType().apply {
                    this.name = "Fish"
                    this.icon = R.drawable.item_typedish_fish
                })
                copyToRealm(DishType().apply {
                    this.name = "Meat"
                    this.icon = R.drawable.icon_typedish_meat
                })
            }
        }

        fun addDish(
            types: List<DishTypeLocalModel>,
            name: String,
            recipe: String,
            ingridients: List<String>,
            image: Uri?,
            icon: Uri?,
        ) = CoroutineScope(Dispatchers.IO).launch {

            realm.writeBlocking {

                this.copyToRealm(Dish().apply {
                    this.name = name
                    this.icon = icon.toString()
                    this.image = image.toString()
                    this.recipe = recipe
                    this.ingridients                                    =
ingridients.toRealmList()
```

```
                })
            }

            realm.writeBlocking {
                val currentDish = this.query<Dish>("name ==
$0", name)
                    .find()
                    .first()

                types.forEach {
                    val dishType = this.query<DishType>("name
== $0", it.name)
                        .find()
                        .first()
                    var newDishes = dishType.dishes
                    newDishes.add(currentDish)
                    dishType
                    dishType.dishes = newDishes
                }
            }
        }
        fun getAllDishes(): List<DishLocalModel> {
            return       realm.query<Dish>().find().map       {
it.toLocalModel() }
        }

        fun getDishesByType(type:String): List<Dish> {
            val dishType = realm.query<DishType>("name == $0",
type).first().find()
```

```
            return dishType?.dishes?.toList() ?: listOf()
        }

        fun getDishTypes(): List<DishTypeLocalModel> {
            return     realm.query<DishType>().find().map     {
it.toLocalModel() }
        }

    }
```

Цей код представляє клас Database, який відповідає за взаємодію з локальною базою даних Realm в Android-додатку. Давайте розглянемо його ключові елементи:

- Конфігурація та ініціалізація Realm. У конструкторі створюється конфігурація Realm з вказаною схемою, яка включає класи DishType і Dish. Після цього створюється екземпляр Realm з цією конфігурацією.
- Додавання типів страв та встановлення типів за замовчуванням: addDishType додає новий тип страви до бази даних, setDefaultDishTypes встановлює типи страв за замовчуванням.
- AddDish додає нову страву до бази даних за допомогою введених параметрів. Використовує вираз CoroutineScope(Dispatchers.IO).launch для асинхронного виконання операції.
- GetAllDishes повертає список всіх страв у форматі DishLocalModel., getDishesByType отримує страви за вказаним типом.
- Використовуються Realm-запити для отримання даних з бази даних. Змінні, такі як currentDish та dishType, використовуються для отримання об'єктів з бази даних.
- Використовуються URI для представлення шляхів до зображень страви та її іконки.

– Введені корутини для асинхронного виконання операцій запису у базі даних.

Загалом, код здійснює важливі операції взаємодії з локальною базою даних Realm, включаючи додавання та отримання даних. У реальному додатку це може бути основою для розширення та оптимізації роботи з локальними даними.

Також було створено класи Dish та DishType. Класи Dish та DishType розширюють RealmObject, що дозволяє їм зберігатися в базі даних Realm.

Локальні моделі, такі як DishLocalModel та DishTypeLocalModel, використовуються для зручності роботи з даними в рівні представлення та можуть містити лише необхідну інформацію для відображення на користувацькому інтерфейсі або в інших частинах програми.

3.4. Dependency injection

У сучасному світі розробки програмного забезпечення навколо терміну "Dependency Injection" (DI) складається значна аура, що об'єднує комфорт та покращення в управлінні залежностями. У світі Android, де швидкість розробки та підтримка великих проектів мають ключове значення, DI стає необхідністю.

Вміст файлу modules.kt:

```
val singleModules = module {
    singleOf(::Database)
    singleOf(::AppPreferences)
    singleOf(::provideSharedPreferences)
}
val viewModelModules = module {
    viewModelOf(::MainViewModel)
    viewModelOf(::AddDishViewModel)
    viewModelOf(::ChooseCategoryViewModel)
    viewModelOf(::DishTypeViewModel)
```

```
    }

    fun provideSharedPreferences(context: Context):
SharedPreferences =

context.getSharedPreferences("organise_eat_clone_preferences",
Activity.MODE_PRIVATE)

    fun provideModules() = listOf(singleModules,
viewModelModules)
```

SingleModules та viewModelModules представляють два модулі для конфігурації Koin. SingleModules використовує singleOf, щоб оголосити залежності як single (одиничні), що означає, що Koin створює один об'єкт та повертає його кожен раз, коли йому потрібно цю залежність.

ViewModelModules використовує viewModelOf, щоб оголосити залежності, які пов'язані із життєвим циклом ViewModel.

ProvideSharedPreferences - це функція, яка визначає, як створити об'єкт SharedPreferences для використання в додатку. Ця функція використовує контекст для створення SharedPreferences.

ProvideModules - це функція, яка об'єднує два модулі (singleModules та viewModelModules) і повертає їх у вигляді списку. Це дозволяє легко використовувати ці модулі при конфігурації Koin у вашому додатку.

3.5. Main екран додатку

В даному розділі ми розглянемо ключові елементи основного екрану нашого Android-додатку, які відіграють важливу роль у взаємодії користувача з програмою. Головний екран, представлений класом MainActivity, є входовою точкою в програму, де користувач отримує можливість взаємодіяти з різними функціональними можливостями.

В класі MainActivity реалізовано відображення списку типів страв, які користувач може вибрати. Це створює основу для введення рецептів та роботи з категоріями страв.

Використання Intent дозволяє нам впроваджувати логіку переходів між екранами, зокрема, запуск екрану вибору типу страв (DishTypeActivity) та екрану додавання рецептів (AddDishActivity).

Цей модуль є ключовим у відкритті доступу до різних можливостей нашого додатку. Відправна точка нашої програми дозволяє користувачеві здійснювати вибір та взаємодію з основними функціональними можливостями, відображаючи зручний та інтуїтивно зрозумілий інтерфейс.

Код MainActivity.kt:

```
class MainActivity : BaseActivity<ActivityMainBinding>() {
    private lateinit var permissionLauncher:
ActivityResultLauncher<String>
    val viewModel: MainViewModel by inject()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        viewModel.dishesTypeLiveData.observe(this,
::handleDishesType)
        viewModel.getDishTypes()
    }

    private fun handleDishesType(dishTypeLocalModels:
List<DishTypeLocalModel>?) {
        val adapter =

DishTypeRecyclerViewAdapter(dishTypeLocalModels,
::categoryChooseListenerCallback)
```

```
            binding.recyclerView.adapter = adapter
        }

        private fun categoryChooseListenerCallback(dishType:
String) {
            if(checkPermission()) {
                startDishTypeActivity(dishType)
            }
        }

        private fun startDishTypeActivity(dishType: String) {
            val intent = Intent(this@MainActivity,
DishTypeActivity::class.java)
            intent.putExtra(DISH_TYPE_NAME, dishType)

//intent.addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION)
            this@MainActivity.startActivity(intent)
        }

        override fun setupBinding(
            inflater: LayoutInflater,
            container: ViewGroup?
        ) = ActivityMainBinding.inflate(inflater, container,
false)


        override fun ActivityMainBinding.initializeLayout() {
            registerPermissionListener()
            addRecipeButton.setOnClickListener {
```

```
                val    intent    =    Intent(this@MainActivity,
AddDishActivity::class.java)
                this@MainActivity.startActivity(intent)
            }
            searchButton.setOnClickListener {
                val    intent    =    Intent(this@MainActivity,
SearchActivity::class.java)
                this@MainActivity.startActivity(intent)
            }
        }

        private fun registerPermissionListener(){
            permissionLauncher                                    =
registerForActivityResult(ActivityResultContracts.RequestPermiss
ion()){
                if(!it)Toast.makeText(this,"Permission  denied",
Toast.LENGTH_SHORT)
            }
        }

        private fun checkPermission(): Boolean {
            when {
                ContextCompat.checkSelfPermission(
                    this,

android.Manifest.permission.READ_EXTERNAL_STORAGE
                )
                        == PackageManager.PERMISSION_GRANTED ->
{
```

```
                        return true
                }

shouldShowRequestPermissionRationale(Manifest.permission.READ_EX
TERNAL_STORAGE) -> {
                    Toast.makeText(this,"We need your
permission", Toast.LENGTH_SHORT)
                    return false
                }

                else -> {

permissionLauncher.launch(Manifest.permission.READ_EXTERNAL_STOR
AGE)
                    return false
                }

            }
        }

        companion object {
            const val DISH_TYPE_NAME = "Dish type name"
        }
    }
```

Цей код представляє діяльність (Activity) Android-додатку, яка є головним екраном додатку.

Метод onCreate викликається при створенні діяльності.

Підписка на dishesTypeLiveData служить для отримання списку типів страв з viewModel, яка використовує Dependency Injection (DI) для отримання екземпляру MainViewModel.

Метод getDishTypes() ініціалізує запит для отримання типів страв.

Метод handleDishesType викликається при отриманні даних про типи страв з viewModel.

Створюється і встановлюється адаптер DishTypeRecyclerViewAdapter для відображення типів страв у recyclerView. Також вказується callback для обробки подій вибору типу страви.

Метод categoryChooseListenerCallback, який викликається при виборі типу страви в адаптері перевіряє наявність необхідних дозволів перед відкриттям DishTypeActivity. Якщо дозволів немає, вони запитуються у користувача.

StartDishTypeActivity метод це метод для старту DishTypeActivity з передачею імені вибраного типу страви як додаткової інформації.

Методи setupBinding та initializeLayout це методи, які служать для ініціалізації Binding та встановлення розмітки активності.

Методи registerPermissionListener та checkPermission це метди, які відповідають за перевірку дозволів на читання зовнішньої пам'яті та їхню обробку. Якщо дозволу немає, вони виводять повідомлення або запитують дозвіл.

Об'єкт-компаніон із константою DISH_TYPE_NAME, яка використовується для передачі імені типу страви між активностями.

Цей код демонструє використання архітектурних підходів у створенні Android-додатку, таких як використання ViewModel, адаптера для RecyclerView, робота з дозволами та взаємодія між різними екранами додатку.

Також розглянемо MainVIewModel.kt:

```
class MainViewModel() : ViewModel(), KoinComponent {

    val database: Database by inject()
    val sharedPreferences: AppPreferences by inject()
```

```
        val dishesTypeLiveData =
MutableLiveData<List<DishTypeLocalModel>>()

        fun getDishTypes() {
            CoroutineScope(Dispatchers.IO).launch {
                if (sharedPreferences.isFirstRun()) {
                    database.setDefaultDishTypes()
                    sharedPreferences.setAnyRun()
                }
                val list = database.getDishTypes()
                dishesTypeLiveData.postValue(list)
            }
        }
```

Цей код представляє клас MainViewModel, який є частиною архітектури MVVM (Model-View-ViewModel). Нижче наведено роз'яснення ключових елементів цього коду:

Клас використовує Koin для внедрення залежностей. Класи Database та AppPreferences вибираються та ініціалізуються через Koin.

Властивості:

- database: Об'єкт класу Database, який використовується для взаємодії з базою даних.
- sharedPreferences: Об'єкт класу AppPreferences, який використовується для роботи зі спільними налаштуваннями додатку.
- dishesTypeLiveData: Об'єкт MutableLiveData, який буде оновлюватися при отриманні нового списку типів страв.

Метод getDishTypes викликається для отримання списку типів страв, перевіряє, чи це перший запуск додатку. Якщо так, то встановлюється типовий набір страв у базу даних. Також цей метод отримує список типів страв з бази даних та оновлює dishesTypeLiveData.

Операції з базою даних виконуються в окремому потоці за допомогою корутин, щоб уникнути блокування головного потоку та підвищити ефективність додатку.

Цей клас відповідає за логіку взаємодії з базою даних та обробки даних для головного екрану додатку.

3.6. Послідовність екранів з додавання елементів до бази даних

У розробці нашого мобільного додатку створення функціоналу для додавання нових страв є ключовим етапом. Цей процес включає в себе не лише локальне збереження інформації, а й її відправлення на віддалений сервер для подальшого обміну та синхронізації з іншими пристроями. У цьому контексті виникає необхідність створення екранів, які забезпечують користувача інтуїтивно зрозумілим та ефективним інтерфейсом для введення даних про нові страви.

На цьому етапі розробки важливо розглянути два аспекти: локальне збереження даних та взаємодію із віддаленим сервером. Локальне збереження зазвичай використовує місцеву базу даних для забезпечення ефективного доступу та обробки інформації, що вводиться користувачем. У цьому випадку, крім введення текстової інформації, можливе завантаження зображень страви для подальшого відображення. З іншого боку, для забезпечення синхронізації та обміну даними із віддаленим сервером, використовуються механізми відправки HTTP-запитів.

У цьому контексті створення екранів для додавання нових страв включає в себе розробку інтерфейсу для зручного введення інформації, вибору категорії, завантаження зображень та інших деталей страви. Паралельно з цим, код для взаємодії із локальною базою даних та відправки цих даних на віддалений сервер має забезпечувати ефективність та надійність обробки інформації.

У цьому контексті, давайте розглянемо ключові етапи та інструменти, які можуть бути використані для реалізації екранів додавання нових страв та їх подальшої синхронізації із віддаленим сервером.

В першу чергу, розглянемо файл AddDishActivity.kt:

```
class AddDishActivity : BaseActivity<ActivityNewDishBinding>() {

    private val viewModel: AddDishViewModel by inject()
    var imgUri: Uri? = null
    var icoUri: Uri? = null
    override fun setupBinding(
        inflater: LayoutInflater,
        container: ViewGroup?
    ) = ActivityNewDishBinding.inflate(inflater, container,
false)

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        viewModel.ingredientsLiveData.observe(this,
::handleIngredients)
    }

    private fun handleIngredients(strings:
ArrayList<String>?) {
        val adapter =
IngredientsRecyclerViewAdapter(strings,
::deleteIngredientButtonCallback)
        binding.ingredientRecyclerView2.adapter = adapter
    }

    private fun deleteIngredientButtonCallback(position:
Int) {
        viewModel.deleteIngredient(position)
```

```
        }

        override  fun  ActivityNewDishBinding.initializeLayout()
{

            addIngridientButton.setOnClickListener {

viewModel.addIngredient(newIngridient.text.toString())
                newIngridient.text = null
            }

            val                  imageActivity                  =
createAddImageActivityForResult(receiptImage, URI_IMG)
            addImageButton.setOnClickListener {
                imageActivity.launch()
            }

            val                   iconActivity                  =
createAddImageActivityForResult(dishIcon, URI_ICO)
            changeIconButton.setOnClickListener {
                iconActivity.launch()
            }

            saveReceiptButton.setOnClickListener {
                val   intent   =   Intent(this@AddDishActivity,
ChooseCategoryActivity::class.java)
                val model = DishParcelizeLocalModel(
                    id = ObjectId().toString(),
                    name = dishTitle.text.toString(),
```

```
                    recipe = receipt.text.toString(),
                    ingridients                                    =
viewModel.ingredientsLiveData.value,
                    image = imgUri,
                    icon = icoUri
                )
                intent.putExtra(DISH_MODEL, model)
                this@AddDishActivity.startActivity(intent)
            }

        }

        private fun createAddImageActivityForResult(
            imageView: ImageView,
            uriIdentify: String
        ): ActivityResultLauncher<Unit> {
            val                activityLauncher                =
registerForActivityResult(ActivityImageContract()) { result ->
                if (result != null) {
                    imageView
                        .setImageBitmap(
                            MediaStore
                                .Images
                                .Media
                                .getBitmap(
                                    this@AddDishActivity
                                        .contentResolver,
                                    result
                                )
```

```
                        )
                }
                if (uriIdentify == URI_ICO) imgUri = result
else icoUri = result
            }
            return activityLauncher
        }

        companion object {
            const val DISH_MODEL = "Dish Model"
            const val URI_IMG = "Uri image"
            const val URI_ICO = "Uri icon"
        }
    }
```

Цей код представляє собою активність (Activity) у додатку Android для додавання нового рецепту страви. Давайте розглянемо деякі ключові аспекти цього коду.

Клас AddDishActivity успадковує базовий клас BaseActivity, який має спільний код для всіх активностей в додатку.

Використання Koin для внедрення (injection) залежностей дозволяє отримати доступ до AddDishViewModel, який відповідає за логіку даної активності.

Використовуючи ViewBinding, ми ініціалізуємо зв'язку між кодом та макетом XML (ActivityNewDishBinding). Також встановлюєте спостереження за ingredientsLiveData, яке відслідковує зміни інгредієнтів.

У методі handleIngredients встановлюємо адаптер для RecyclerView, який відображатиме список інгредієнтів.

Для роботи з зображеннями використовуються дві різні активності. Метод createAddImageActivityForResult створює лаунчер для отримання результату вибору зображення та відображення його у відповідному ImageView.

При кліку на кнопку збереження, створюється об'єкт моделі рецепту (DishParcelizeLocalModel), а потім запускається інша активність (ChooseCategoryActivity) із передачею цього об'єкта.

Метод registerForActivityResult – метод для створення лаунчера для отримання результату вибору зображення та встановлення його в ImageView.

Цей код взаємодіє з різними компонентами Android, відображаючи екран для додавання нового рецепту страви та оброблюючи подальші кроки користувача.

Також розглянемо AddDishViewModel.kt:

```
class AddDishViewModel() : ViewModel(), KoinComponent {

    var ingredientsLiveData = MutableLiveData<ArrayList<String>?>()

    fun addIngredient(ingredient: String) {
        var currentList = ingredientsLiveData.value
        if (currentList == null) {
            currentList = arrayListOf()
        }
        currentList.add(ingredient)
        ingredientsLiveData.value = currentList
    }

    fun deleteIngredient(position: Int){
        var currentList = ingredientsLiveData.value
```

```
currentList!!.asReversed().remove(currentList[position])
            ingredientsLiveData.value = currentList
        }

        fun createDishModel(){

        }

    }
```

Клас успадковує базовий клас ViewModel, який надає можливості для збереження та управління даними, пов'язаними з інтерфейсом користувача. Також використовується KoinComponent для інтеграції з фреймворком Koin.

LiveData використовується для спостереження за змінами у списку інгредієнтів. Коли дані змінюються, будуть автоматично сповіщені всі слухачі.

Метод addIngredient додає новий інгредієнт до списку. Якщо список не існує, він створюється.

Метод deleteIngredient видаляє інгредієнт за певною позицією у списку.

Цей ViewModel призначений для управління даними та логікою взаємодії для екрану додавання нової страви. Він використовує LiveData для спрощення сповіщення про зміни в даних та надає методи для додавання та видалення інгредієнтів у списку.

Далі в розробці екранів для додавання нових страв важливо передбачити механізми збереження моделі даних у локальній базі даних та відправлення цієї моделі на віддалений сервер. Для цього нам знадобляться методи та функціонал для взаємодії з базою даних та відправки HTTP-запитів.

Розглянемо клас ChooseCategoryActivity:

```
    class ChooseCategoryActivity :
BaseActivity<ActivityNewDishCategoryBinding>() {

        val viewModel: ChooseCategoryViewModel by inject()
        override fun setupBinding(
            inflater: LayoutInflater,
            container: ViewGroup?
        ) = ActivityNewDishCategoryBinding.inflate(inflater,
container, false)

        override fun onCreate(savedInstanceState: Bundle?) {
            super.onCreate(savedInstanceState)
            viewModel.dishesTypeLiveData.observe(this,
::handleDishesType)
            viewModel.getDishTypes()
        }

        private fun handleDishesType(dishTypeLocalModels:
List<DishTypeLocalModel>?) {
            val adapter =
DishTypeChooseRecyclerViewAdapter(dishTypeLocalModels,::dishType
Choose)
            binding.recyclerView2.adapter = adapter
        }

        private fun dishTypeChoose(dishTypeLocalModel:
DishTypeLocalModel) {

viewModel.chosedDishTypesListAction(dishTypeLocalModel)
```

```
        }

        override fun
ActivityNewDishCategoryBinding.initializeLayout() {
            val model =
intent.getParcelableExtra<DishParcelizeLocalModel>(DISH_MODEL)
            saveReceiptButton2.setOnClickListener {
                viewModel.saveReceipt(model!!)
                viewModel.post(model,baseContext)
                val intent =
Intent(this@ChooseCategoryActivity,MainActivity::class.java)
                intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK
or              Intent.FLAG_ACTIVITY_CLEAR_TASK or
Intent.FLAG_ACTIVITY_CLEAR_TOP)

this@ChooseCategoryActivity.startActivity(intent)
                this@ChooseCategoryActivity.finish()
            }
        }
    }
```

Давайте розглянемо кожен його елемент окремо:

Метод setupBinding це метод, який повертає об'єкт ActivityNewDishCategoryBinding, який використовується для прив'язки інтерфейсу користувача (XML-файлу розмітки) до коду активності.

В методі onCreate встановлюється спостереження за dishesTypeLiveData в ViewModel, і викликається метод getDishTypes(), який, ймовірно, отримує список категорій страв.

В методі handleDishesType встановлюється адаптер для RecyclerView, який відображає список категорій страв (DishTypeLocalModel). Кожен елемент списку має кнопку або дію, пов'язану з вибором цієї категорії.

Метод dishTypeChoose викликається при виборі певної категорії страви. Можливо, він реалізований у ViewModel (ChooseCategoryViewModel) і виконує дії, пов'язані із вибором конкретної категорії.

В методі initializeLayout міститься ініціалізація елементів інтерфейсу користувача та встановлення обробників подій, таких як кнопка saveReceiptButton2.

В даному випадку, при кліку на кнопку, викликається метод saveReceipt в ViewModel, а також відправляється модель на сервер за допомогою viewModel.post(model, baseContext).

Також відбувається перехід на головний екран програми.

Отже, активність ChooseCategoryActivity відображає категорії страв, дозволяє користувачеві вибрати одну з них, і виконує деякі дії при збереженні та відправленні моделі страви.

Також необхідно розглянути view model до цієї активності:

```
    class ChooseCategoryViewModel() : ViewModel(), KoinComponent {

        val database: Database by inject()
        val dishesTypeLiveData = MutableLiveData<List<DishTypeLocalModel>>()
        var dishTypeSelected = mutableListOf<DishTypeLocalModel>()
        //var saveStateLiveData = MutableLiveData<String>(CATEGORY_CHOOSE) //Состояние

        fun getDishTypes() {
```

```
            CoroutineScope(Dispatchers.IO).launch {
                val list = database.getDishTypes()
                dishesTypeLiveData.postValue(list)
            }
        }

        fun chosedDishTypesListAction(dishTypeLocalModel:
DishTypeLocalModel) {
            if (!dishTypeSelected.contains(dishTypeLocalModel))
                dishTypeSelected.add(dishTypeLocalModel) else
                dishTypeSelected.remove(dishTypeLocalModel)
        }

        private fun getTmpImageFileFromUri(
            context: Context,
            uri: Uri,
            fileSuffix: String = ".jpg",
            filePrefix: String = "logo_"
        ): File {
            val file = File.createTempFile(filePrefix,
fileSuffix)
            file.deleteOnExit()
            context.contentResolver.openInputStream(uri)?.use {
input ->
                file.outputStream().use { output ->
                    input.copyTo(output)
                }
            }
            return file
```

```
        }

        private fun getImg(file: File): Bitmap? {
            return BitmapFactory.decodeFile(file.toString())
        }

        fun     post(model:DishParcelizeLocalModel,     context:
Context){

                CoroutineScope(Dispatchers.IO).launch {
                    val client = OkHttpClient()

                    val formBody = FormBody.Builder()
                        .add("id", model.id)
                        .add("name", model.name)
                        .add("recipe", model.recipe)
                        .add("ingridients",
model.ingridients.toString())
                        .add("image", model.image.toString())
                        .add("icon", model.icon.toString())
                        .build();

                    val request = Request.Builder()

.url("https://eodnc0ppu6e1uii.m.pipedream.net")
                        .post(formBody)
                        .build();

                    val call = client.newCall(request)
```

```
                call.execute()
            }

        }
        fun saveReceipt(dish: DishParcelizeLocalModel) {

            database.addDish(
                dishTypeSelected.toList(),
                dish.name,
                dish.recipe,
                dish.ingridients ?: listOf(),
                dish.image,
                dish.icon
            )

        }

        companion object {
            const val CATEGORY_CHOOSE = "Category choose"
            const val SAVING_RECEIPT = "Saving receipt"
        }

    }
```

Цей клас є ViewModel для екрану вибору категорій для нової страви. Розглянемо його основні складові частини:

– Об'єкт Database використовується для взаємодії з базою даних, а LiveData для спостереження за списком категорій страв (DishTypeLocalModel).
– MutableList відображає вибрані користувачем категорії страв

- Метод getDishTypes для асинхронного отримання списку категорій страв з бази даних та оновлення dishesTypeLiveData.
- Метод chosedDishTypesListAction для додавання або видалення обраної категорії від користувача.
- Метод post для відправки моделі страви на віддалений сервер за допомогою OkHttp бібліотеки.
- Метод saveReceipt для збереження нової страви в базу даних з вибраними категоріями.
- Компанійський об'єкт для збереження різних станів ViewModel.

Ця ViewModel відповідає за управління вибором категорій для нової страви, а також за відправку та збереження цієї страви.

3.7 Створення активності для обробки GET запитів

Розглянемо активіті, яка взаємодіє з віддаленим сервером та отримує дані за допомогою HTTP GET-запитів. Дізнаємося, як ця активіті взаємодіє з сервером, обробляє отримані дані та відображає їх у користувацькому інтерфейсі.

Активіті відіграє ключову роль в процесі забезпечення комунікації між Android-додатком та віддаленим сервером. У цьому контексті HTTP GET-запити є одним з основних механізмів для отримання інформації з сервера.

Розглянемо клас SearchActivity:

```
class SearchActivity: BaseActivity<ActivitySearchBinding>() {
    override fun setupBinding(
        inflater: LayoutInflater,
        container: ViewGroup?
    ) = ActivitySearchBinding.inflate(inflater,container,false)

    override fun ActivitySearchBinding.initializeLayout() {
        activitySearchButton.setOnClickListener {
```

```
            CoroutineScope(Dispatchers.IO).launch {
                val dish = activitySearchText.text
                val recipeAPI = RecipeAPI()
                recipeAPI.url += dish
                recipeAPI.getResponse {
                    val response = JSONObject(it)
                    lifecycleScope.launch {

                        fun    ingredientsResponseToString(list:
List<String>): String {
                            var result = ""
                            list.forEach { result += it + "\n" }
                            return result
                        }

                        activitySearchReceiptNameText
                            .text = response
                            .getString("title")
                        val ingredientsArray = response
                            .getString("ingredients")
                            .split("|")
                        activitySearchIngredientsText
                            .text                                =
ingredientsResponseToString(ingredientsArray)
                        activitySearchServingsText
                            .text                                =
response.getString("servings")
                        activitySearchInstructionsText
```

```
                                .text                                =
response.getString("instructions")
                        }
                    }
                }
            }
        }
}
```

При натисканні кнопки activitySearchButton створюється корутин в IO-потоці, отримується текст з activitySearchText, створюється екземпляр RecipeAPI, до якого додається текст пошуку для створення URL, викликається метод getResponse, який виконує HTTP-запит до сервера за допомогою RecipeAPI. При отриманні відповіді викликається блок коду.

Відповідь від сервера обробляється як об'єкт JSONObject, здійснюється оновлення UI за допомогою lifecycleScope.launch, створюється функція ingredientsResponseToString, яка перетворює список інгредієнтів у рядок з роздільниками, здійснюється оновлення різних текстових полів в інтерфейсі за допомогою отриманих даних з відповіді сервера.

Цей код демонструє взаємодію активіті з віддаленим сервером, використовуючи HTTP-запити, і оновлення інтерфейсу користувача на основі отриманих даних.

3.8 Огляд прототипу проекту

Протягом розробки мобільного додатку, був реалізований головний екран додатку, що зображено далі (рис.3.2).

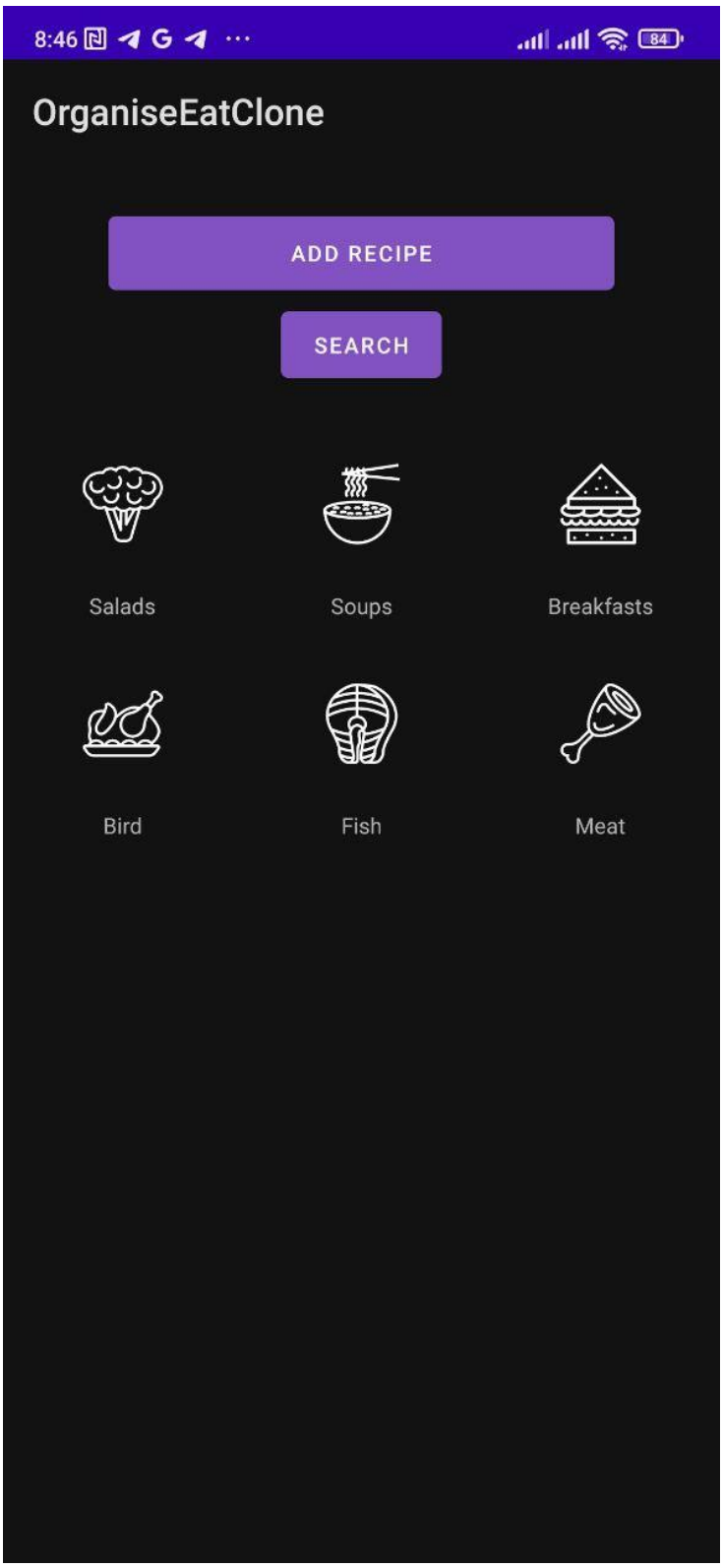

Рисунок 3.2 – Головний екран додатку

При натисканні кнопки “Add recipe” користувач переходить на екран, що визначається наявністю інтерфейсних елементів, спрямованих на відправлення

введених користувачем даних на віддалений сервер. Екран з введенням даних зображено далі (рис. 3.3).

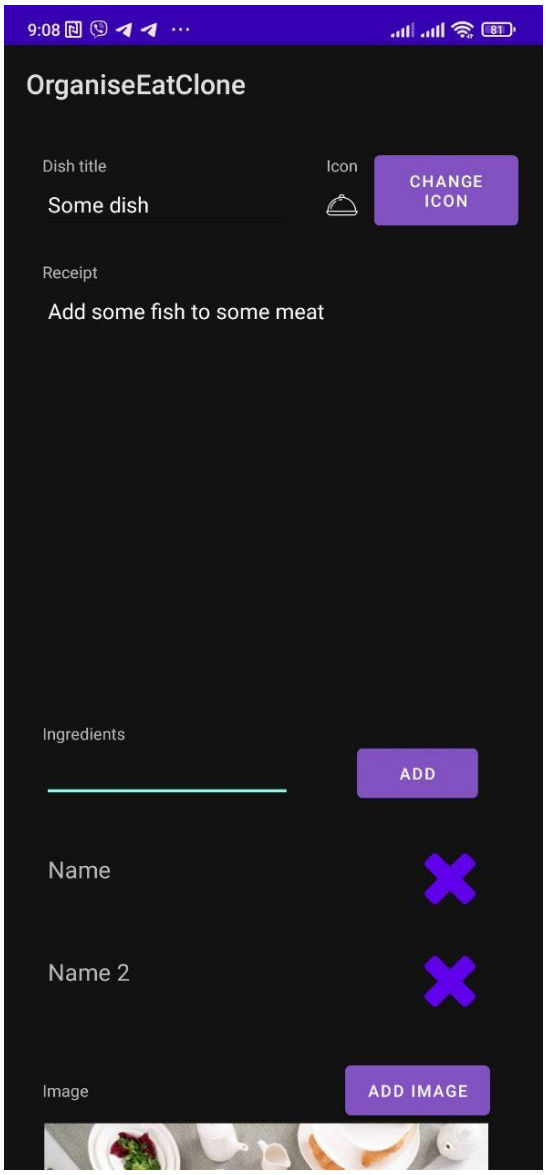

Рисунок 3.3 – Екран введення даних

Цей етап реалізації додатку передбачає взаємодію із зовнішнім сервером для обміну необхідною інформацією, що відіграє ключову роль у подальших процесах роботи додатку.

Під час взаємодії з додатком користувач, переходить на наступний етап, де відбувається підтвердження відправлення інформації. Цей процес забезпечує взаємодію користувача з системою та гарантує точність та обробку введених

даних перед подальшим використанням. Екран підтвердження відправки запиту зображено далі (рис.3.4).

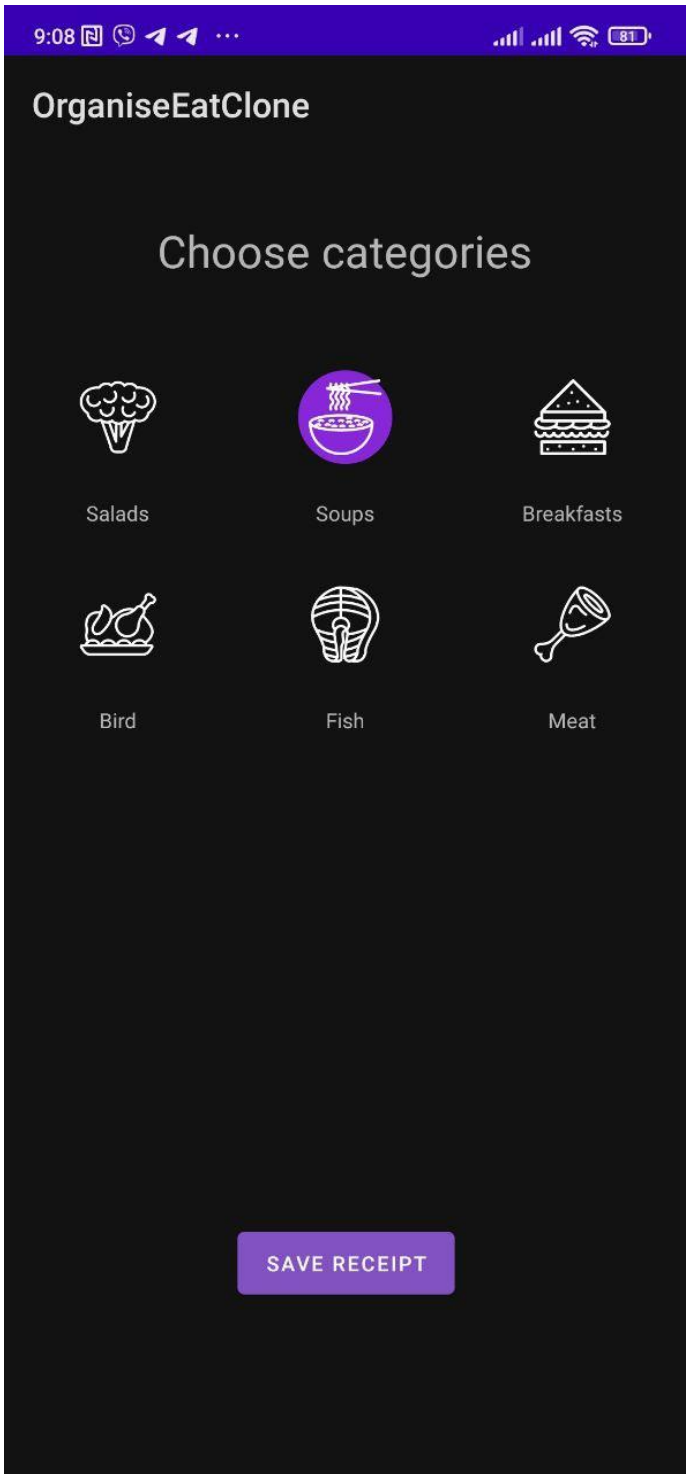

Рисунок 3.4 – Екран підтвердження відправки даних

Після вибору категорії та підтвердження дій на екрані підтвердження, додаток забезпечує повернення користувача на головний екран, де відображено

ключові функції та опції програми. Цей крок відображає зручність та інтуїтивність використання додатку, надаючи користувачеві зручний та швидкий доступ до основних функціональних можливостей застосунку.

Результат що приходить на сервер зображено на далі (рис.3.5).

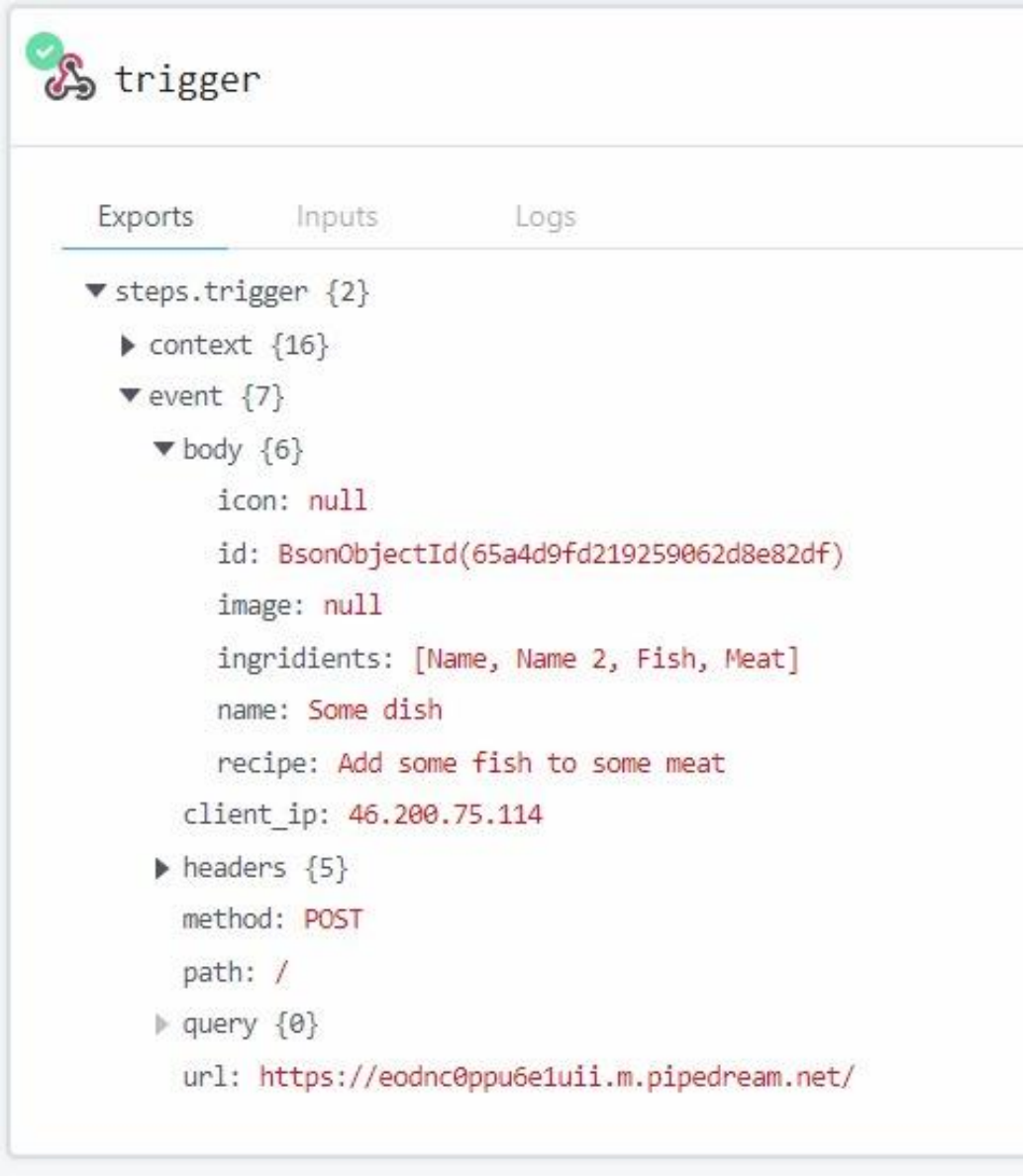

Рисунок 3.5 – Результат отриманий сервером

На головному екрані додатку забезпечено інтуїтивно зрозумілу функціональність, яка дозволяє користувачеві легко та швидко перейти на екран для отримання даних з серверу. Екран для отримання даних зображено на далі (рис.3.6).

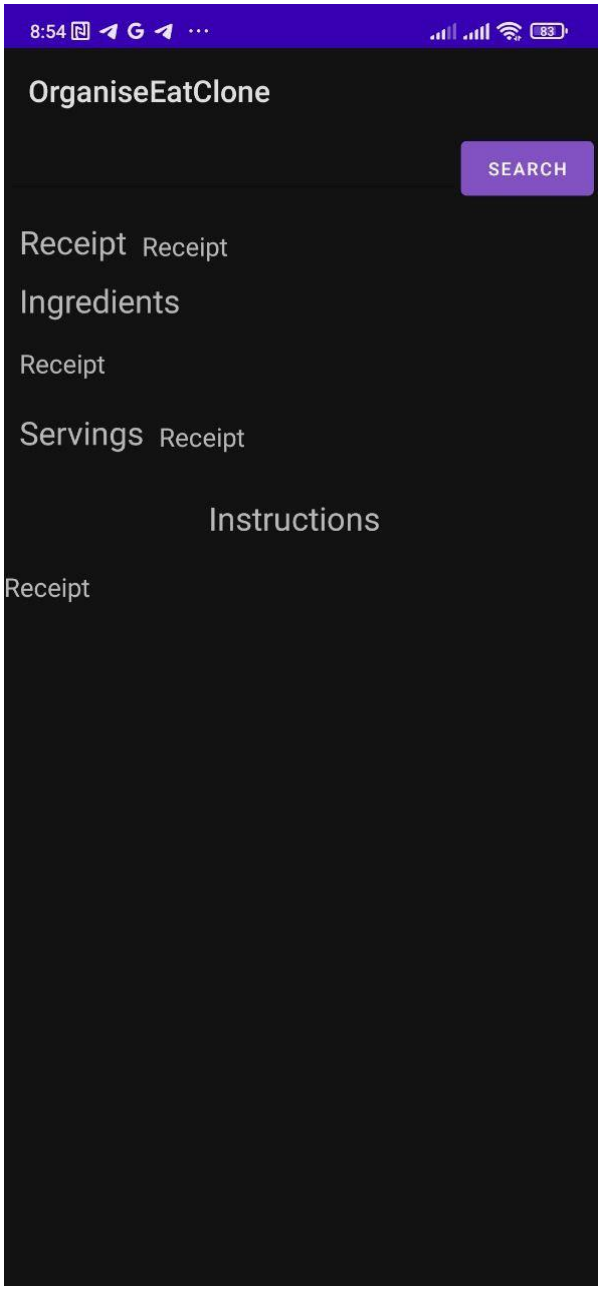

Рисунок 3.6 – Екран для отримання даних з сервера

Після введення основних даних на екрані входу користувач має можливість легко ініціювати відправлення запиту на сервер, натисканням нативної клавіші "Search".

Після натискання клавіші, система відправляє запит на сервер, і забезпечує отримання результатів. Екран після отримання результатів зображено далі (рис.3.7).

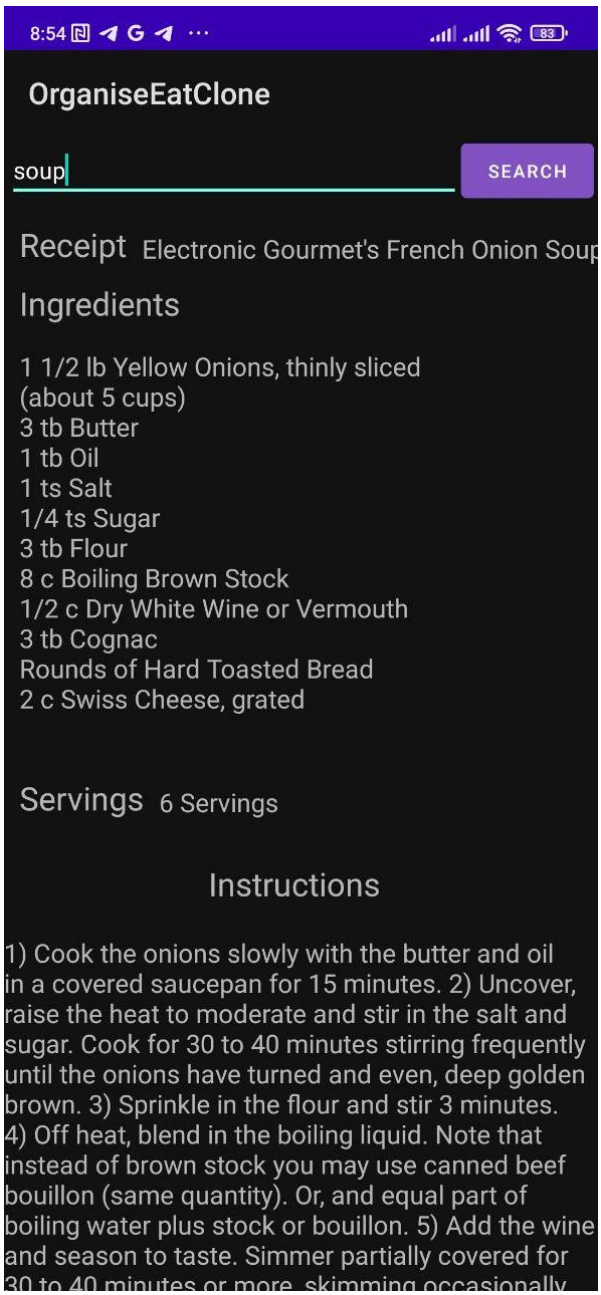

Рисунок 3.7 – Екран для отримання даних після відправки запиту

Висновок за розділом 3: на етапі налаштування проекту визначено залежності, конфігуровано параметри проекту, встановлено бібліотеки, фреймворки та інші інструменти, необхідні для розробки програмного продукту

Створено три ключові базові класи: BaseActivity, BaseFragment та BaseHTTPRequest, які відіграють ключову роль у структурі та функціональності нашого Android-додатка.

BaseActivity відображає базовий клас для всіх активностей в нашому додатку. Він використовує View Binding для зручного доступу до елементів інтерфейсу користувача та забезпечує загальний цикл життя для всіх активностей.

BaseFragment визначає базовий клас для всіх фрагментів, що використовуються у нашому додатку. Він забезпечує спільний функціонал для управління UI елементами та взаємодії фрагментів.

BaseHTTPRequest є базовим класом для роботи з HTTP-запитами в нашому додатку. Він може містити загальні методи для виконання GET, POST, PUT, DELETE запитів та обробки відповідей сервера.

Створено локальну БД на основі Realm, що надає простий та ефективний спосіб робити операції з базою даних, забезпечуючи високий рівень продуктивності та зменшуючи обсяг коду, необхідного для роботи з базовими операціями CRUD (створення, читання, оновлення, видалення).

Розроблено головний екран, представлений класом MainActivity, є входовою точкою в програму, де користувач отримує можливість взаємодіяти з різними функціональними можливостями. Також розроблено екран для відправлення на віддалений сервер для подальшого обміну та синхронізації з іншими пристроями і екран для прийому інформації з віддаленого серверу.

ВИСНОВКИ

У процесі аналізу аналогічних застосунків розглянуто три типи збереження даних: на пристрої, на віддаленому сервері та власному сервері. Кожен з цих методів має свої унікальні переваги та виклики, і їх вибір залежить від конкретного контексту та завдань додатку.

Використання віддаленого арендованого сервера для збереження даних у мобільному додатку може бути обумовлене декількома ключовими факторами, які впливають на ефективність та безпеку додатку. Основні переваги використання віддаленого арендованого сервера: зручність для користувача, безпека персональних даних та економічність. Загалом, використання віддаленого арендованого сервера дозволяє максимально оптимізувати витрати та забезпечити найвищий рівень доступності, безпеки та швидкодії для користувачів мобільного додатку.

Було проведено аналіз аналогічних мобільних застосунків, що функціонують у схожому контексті. В рамках цього дослідження ми ретельно розглянули та порівняли функціонал та особливості зберігання даних відомих додатків, що дозволить нам виявити найкращі практики та можливості для подальшого вдосконалення розроблюваного додатку.

Порівняння мобільних застосунків на ринку допоможе виявити їхні переваги та недоліки, а також визначити ті аспекти, які важливі для кінцевого користувача. Окрім того, аналіз конкурентів дозволить нам зрозуміти тенденції ринку та визначити ті напрямки, які важливі для подальшого розвитку та вдосконалення нашого додатку.

Вибір мови програмування – ключовий аспект при розробці мобільного додатку, і використання Kotlin для цієї мети обумовлене декількома важливими факторами:

1) Kotlin пропонує зручний та читабельний синтаксис, який дозволяє виразно виражати ідеї та покращує зрозумілість коду. Мова є більш компактною

порівняно з Java, що сприяє прискоренню розробки та зменшенню кількості написаного коду.

2) Kotlin повністю сумісний із Java, що дозволяє безболісно інтегрувати Kotlin-код із існуючими Java-бібліотеками та проектами. Це значно полегшує міграцію та покращує переносимість коду.

3) Kotlin пропонує безпечний і надійний код завдяки усуненню багатьох можливих проблем, таких як null-посилання, завдяки введенню концепції nullable та non-nullable типів.

4) Мова має багато функціональних конструкцій, таких як лямбда-вирази, короткі функції та інші елементи, які дозволяють писати ефективний та сучасний код.

Вибір MVVM (Model-View-ViewModel) архітектури для розробки мобільного додатку зумовлений низкою вагомих переваг:

1) MVVM дозволяє чітко розділити логіку додатку на три основні компоненти: Model, View та ViewModel. Це полегшує розуміння коду, утримання та масштабування проекту.

2) ViewModel, яка містить логіку взаємодії між Model та View, легко тестується. Це дозволяє створювати ефективні тести та забезпечує високий рівень якості коду.

3) MVVM надає можливість використовувати двунапрямлений зв'язок між View та ViewModel, що дозволяє автоматично оновлювати інтерфейс користувача при зміні даних в ViewModel та навпаки.

4) Чітке розділення між компонентами робить код більш переносимим. ViewModel може використовуватися на різних платформах без змін, що полегшує розробку для різних мобільних платформ.

5) MVVM полегшує утримання проекту, оскільки кожен компонент відповідає за свою конкретну область логіки. Це робить код більш зрозумілим та готовим до розширення.

Обрання Kotlin та MVVM архітектури надає платформонезалежність, чистоту та ефективність у розробці мобільного додатку, що відповідає сучасним вимогам до розробки програмного забезпечення.

На етапі налаштування проекту визначено залежності, конфігуровано параметри проекту, встановлено бібліотеки, фреймворки та інші інструменти, необхідні для розробки програмного продукту

Створено три ключові базові класи: BaseActivity, BaseFragment та BaseHTTPRequest, які відіграють ключову роль у структурі та функціональності нашого Android-додатка.

BaseActivity відображає базовий клас для всіх активностей в нашому додатку. Він використовує View Binding для зручного доступу до елементів інтерфейсу користувача та забезпечує загальний цикл життя для всіх активностей.

BaseFragment визначає базовий клас для всіх фрагментів, що використовуються у нашому додатку. Він забезпечує спільний функціонал для управління UI елементами та взаємодії фрагментів.

BaseHTTPRequest є базовим класом для роботи з HTTP-запитами в нашому додатку. Він може містити загальні методи для виконання GET, POST, PUT, DELETE запитів та обробки відповідей сервера.

Створено локальну БД на основі Realm, що надає простий та ефективний спосіб робити операції з базою даних, забезпечуючи високий рівень продуктивності та зменшуючи обсяг коду, необхідного для роботи з базовими операціями CRUD (створення, читання, оновлення, видалення).

Розроблено головний екран, представлений класом MainActivity, є входовою точкою в програму, де користувач отримує можливість взаємодіяти з різними функціональними можливостями. Також розроблено екран для відправлення на віддалений сервер для подальшого обміну та синхронізації з іншими пристроями і екран для прийому інформації з віддаленого серверу.

Отже, мета і задачі кваліфікаційної роботи виконані у повному обсязі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Способи організації інфраструктури з базами даних: від простого до складного та ефективного. Url: https://habr.com/ru/companies/selectel/articles/725234/
2. Що таке хмарна база даних? Url: https://www.oracle.com/cis/database/what-is-a-cloud-database/
3. Домашній веб-сервер для чайників. Url: https://habr.com/ru/articles/650117/
4. Локальний чи хмарний сервер: плюси кожного вибору. Url: https://habr.com/ru/companies/timeweb/articles/665620/
5. 8 мов програмування для Android-розробника. Url: https://tproger.com/articles/8-jazykov-programmirovanija-dlja-android-razrabotchika
6. Kotlin vs Java. Url: https://habr.com/ru/companies/otus/articles/508060/
7. Патерни для новачків: MVC vs MVP vs MVVM. Url: https://habr.com/ru/articles/215605/
8. Model-View-Controller. Url: https://uk.wikipedia.org/wiki/Model-View-Controller
9. Model-View-Presenter. Url: https://uk.wikipedia.org/wiki/Model-View-Presenter
10. Model-View-ViewModel. Url: https://uk.wikipedia.org/wiki/Model-View-ViewModel
11. MVVM architecture, ViewModel and LiveData (Part 1). Url: https://www.linkedin.com/pulse/mvvm-architecture-viewmodel-livedata-part-1-hazem-saleh
12. Реалістичний Realm. 1 рік досвіду. Url: https://habr.com/ru/articles/328418/

13. V. M. Kartashov, V. N. Oleynikov, S. A. Sheyko, I. V. Koryttsev, S. I. Babkin, O. V. Zubkov, "Peculiarities of small unmanned aerial vehicles detection and recognition," Telecommunications and Radio Engineering, 2019, V. 78, Iss. 9, pp. 771–781. DOI: 10.1615/TelecomRadEng.v78.i9.30.

14. V. N. Oleynikov, O. V. Zubkov, V. M. Kartashov, I. V. Korytsev, S. I. Babkin, S.A. Sheiko, "Investigation of detection and recognition efficiency of small unmanned aerial vehicles on their acoustic radiation," Telecommunications and Radio Engineering, 2019, V. 78, Iss. 9, pp. 759–770. DOI: 10.1615/TelecomRadEng.v78.i9.20.

15. Kartashov, V.M., Sidorov, G.I., Sheiko, S.A., Kolendovska, M.M., Sergienko O.Yu. Principles of Construction and Assessment of technical Characteristics of multi-Frequency atmospheric Sodar in the Humidity Measurement Mode / Telecommunications and Radio Engineering.- New York. - 2020.- Vol. 79, №4.- P.323-333. (стаття). DOI: 10.1615/TelecomRadEng.v79.i4.50.

16. Kartashov, V.M., Oleynikov V.N, Zubkov, O.V., Korytsev I.V., Babkin, S. I., Sheiko, S.A.,. Kolendovskaya, M.M. Spatial-temporal Processing of acoustic Signals of Unmanned Aerial Vehicles; Telecommunications and Radio Engineering, 2020. Vol. 79, Iss, 9, pp.769-780.

17. V.M. Semenets, V.M. Kartashov, V.I. Leonidov. Features of Acoustic Noise of Small Unmanned Aerial Vehicles // Telecommunications and Radio Engineering.- New York. - 2020.- Vol. 79, №11.- P. 985-995. DOI: 10.1615/TelecomRadEng.v79.i11.80 (стаття).

18. Oleynikov V.N., Kartashov, V.M., Babkin, S. I., Zubkov, O.V., Korytsev I.V., Sheiko, S.A.,. Seleznov I.S. Structure and Parameter Unmanned Aerial Vehicles Sound Fields/ Telecommunications and Radio Engineering.- New York. - 2020.- Vol. 79, №17.- P.1539-1550. DOI: 10.1615/TelecomRadEng.v79.i17.50 (стаття).

19. В.А. Тихонов, В.М. Карташов, В.М. Олейніков, В.И. Леонидов, Л.П. Тимошенко, И.С. Селезнеов, Н.В. Рыбников. Обнаружение-распознавание

беспилотных летательных аппаратов с использованием составной модели авторегрессии их акустического излучения// Вісник НТУУ «КПІ». Радіотехніка. Радіоапаратобудування. - 2020.- Вип. №81. – С.38-46. (Web of Science)

20. Карташов В.М., Корытцев И.В., Олейников В.Н., Зубков О.В., Шейко С.А., Бабкин С.И., Левский Н.А., Селезнев И.С. Алгоритмы пеленгации беспилотных летательных аппаратов по их акустическому излучению// Радиотехника. (Харьков). — 2019. — Вып. 196. — С. 22-31.
21. Карташов В.М., Харченко О.И., Чумаков В.И. Использование эффекта стохастического резонанса для анализа спектров акустического излучения малых беспилотных летательных аппаратов // Радиотехника. (Харьков). — 2019. — Вып. 197. — С. 100-106.
22. Карташов В.М., Сидоров Г.І., Толстих Є.Г., Шаповалов С.В. Акустичний вимірювач швидкості вітру в атмосферному прикордонному шарі// Радиотехника. (Харьков). — 2019. — Вып. 199. – С. 54-58.
23. Карташов В.М., Посошенко В.А., Цехмистро Р.И., Тимошенко Л.П., Колендовская М.М. Методы ориентации, навигации и контроля мобильных робототехнических платформ// Радиотехника. (Харьков). — 2019. — Вып. 199. – С. 38-44.
24. Олейников В.Н., Зубков О.В, Карташов В.М., Корытцев И.В., Бабкин С.И., Шейко С.А, Селезнев И.С. Экспериментальная оценка эффективности аглоритмов пеленгования беспилотных летательных аппаратов по акустическому излучению// Радиотехника. (Харьков). — 2019. — Вып. 199. – С. 29-37.
25. Карташов В.М., Олейников В.Н., Колендовская М.М., Тимошенко Л.П., Капуста А.И., Рыбников Н.В. Комплексирование изображений при обнаружении беспилотных летательных аппаратов // Радиотехника. (Харьков). — 2020. — Вып. 201. — С. -120-129.
26. Карташов В.М., Олейников В.Н., Воронин В.В., Рябуха В.П., Капуста А.И., Рыбников Н.В., Селезнев И.С. Методы комплексной обработки и

интерпретации радиолокационных, акустических, оптических и инфракрасных сигналов беспилотных летательных аппаратов // Радиотехника. (Харьков). — 2020. — Вып. 202. — С. 173-182-. DOI:10.30837/rt.2020.3.202.19

27. Карташов В.М., Корытцев И.В., Олейников В.Н., Зубков О.В., Шейко С.А., Бабкин С.И., Обработка сигналов при пеленгации и определении дальности до малоразмерных БПЛА в оптическом и инфракрасном диапазонАХ // Радиотехника. (Харьков). — 2020. — Вып. 202. — С. 125-135. DOI:10.30837/rt.2020.3.202.13
28. Карташов В.М., Корытцев И.В., Олейников В.Н., Зубков О.В., Шейко С.А., Бабкин С.И. ОПТИКО-ЭЛЕКТРОННЫЕ МЕТОДЫ ОБНАРУЖЕНИЯ ВОЗДУШНЫХ ОБЪЕКТОВ И ИЗМЕРЕНИЯ ИХ КООРДИНАТ// Радиотехника. (Харьков). — 2020. — Вып. 202. — С. 153-59. DOI:10.30837/rt.2020.3.202.16
29. Карташов В.М., Корытцев И.В., Олейников В.Н., Зубков О.В., Шейко С.А., Бабкин С.И.. Эффективность детектирования и распознавания изображений дронов по видеопотоку стационарной видеокамеры // Радиотехника. (Харьков). — 2020. — Вып. 202. — С. 136-146. DOI:10.30837/rt.2020.3.202.14
30. Рябуха В.П., Карташов В.М. Методы обнаружения-распознавания радиолокационных, акустических, оптических и инфракрасных сигналов беспилотных летательных аппаратов / В.П. Рябуха, В.М. Карташов// Известия высших учебных заведений. Радиоэлектроника. — 2020. — Т. 63, № 11. — С. 1–35.
31. В.М. Карташов, В.Н. Олейников, В.И. Леонидов, канд. Техн. Наук, В.В. Воронин, А.И. Капуста, И.С. Селезнев, Е.В. Першин/ Комплексная обработка сигналов интегрированной системы наблюдения беспилотных летательных аппаратов с использованием целеуказания//Радиотехника. (Харьков). - 2020. - Вып. 203. - С. 1-13.