Міністерство освіти і науки України
Національний університет «Києво-Могилянська академія»
Кафедра мережних технологій факультету інформатики

Кваліфікаційна робота

освітній ступінь - бакалавр

на тему: **«Розробка Android застосунку системи управління університетом»**

за спеціальністю «Інженерія програмного забезпечення» - 121

Керівник кваліфікаційної роботи

Доктор техн. наук, доцент

Глибовець Андрій Миколайович

_____/підпис/

“_____” _____ 2023 р.

Виконав студент ІПЗ-4:

Козаченко Пилип Максимович

_____/підпис/

“_____” _____ 2023 р.

Київ 2023

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мережних технологій факультету інформатики

ЗАТВЕРДЖУЮ
Зав. кафедри мережних технологій,
проф., д.ф.-м.н.
_____Г. І. Малашонок
(підпис)
„____”_____2023 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на кваліфікаційну роботу

студенту Козаченку Пилипу Максимовичу факультету інформатики 4-го курсу ТЕМА: Розробка Android застосунку системи управління університетом Вихідні дані: Порівняння використання популярних архітектурних підходів при розробці під ОС Андроїд. Детальніший розгляд MVVM підходу та інструментів, які використовуються супутно. Розробка Android застосунку системи управління університетом Зміст ТЧ до кваліфікаційної роботи:
Індивідуальне завдання

Вступ

Розділ 1: Розгляд і порівняння архітектурних підходів в розробці застосунків під ОС Android

Розділ 2: Розгляд структури і підходів в існуючій версії застосунку

Розділ 3: Опис використаних технологій

Розділ 4: Процес впровадження нового функціоналу та вирішення проблем

Розділ 5: Пропозиції для подальшої розробки

Висновки

Список літератури

Дата видачі „___” _____ 2023 р. Керівник _____

(підпис)

Завдання отримав _____

(підпис)

Тема: Розробка Android застосунку системи управління університетом.

Календарний план виконання роботи:

№	Назва етапу	Термін виконання	Примітка
1.	Отримання теми кваліфікаційної роботи	15.07.2022	
2.	Пошук матеріалів про архітектуру	20.01.2023	
3.	Пошук матеріалів про підходи в розробці	21.02.2023	
4.	Ознайомлення з бібліотеками та інструментами для розробки під ОС Android	10.03.2023	
5.	Ознайомленням з наявною версією «KMA Smart»	13.03.2023	
6.	Написання першого розділу кваліфікаційної роботи	05.04.2023	
7.	Написання другого розділу кваліфікаційної роботи	7.04.2023	
8.	Розгляд аспектів використання бібліотек та підходів у наявній версії «KMA Smart»	10.04.2023	
9.	Написання третього розділу кваліфікаційної роботи	11.04.2023	
10.	Написання четвертого розділу кваліфікаційної роботи	15.04.2023	
11.	Написання п’ятого розділу кваліфікаційної роботи	16.04.2023	
12.	Внесення змін до роботи відповідно до коментарів керівника	01.05.2023	
13.	Створення презентації роботи	08.05.2023	
14.	Подача кваліфікаційної роботи на перевірку	24.05.2023	

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

MVC – Model-View-Controller

MVP – Model-View-Presenter

MVVM – Model-View-ViewModel

DI – dependency injection

ОС – операційна система

АНОТАЦІЯ

Розробка Android застосунка системи управління університетом має на меті надати доступ до електронних сервісів та інформації від університету через застосунок на ОС Android. Зокрема, головним завданням є впровадження авторизації в додатку за допомогою університетського акаунту Microsoft та створення функціоналу для взаємодії з веб сервісом, що відповідає за курсові роботи.

В процесі розробки програмного забезпечення було приділено увагу таким аспектам, як чистота та простота коду, коментарі та документація, простота подальшої розробки і масштабування застосунку.

ВСТУП

Заголовок: Розробка Android застосунку системи управління університетом

Актуальність теми: На сьогоднішній день існує тенденція створення онлайн сервісів, на основі та за принципами дійсної офлайн моделі. Переведення або дублювання сервісів у онлайн режим значно спрощує певні процедури для клієнтів сервісу, а також, як правило, така практика економить багато часу та/або ресурсів, як для провайдерів послуг, так і для клієнтської аудиторії. Також, електронні сервіси вважаються більш прозорими, по відношенню до офлайн сервісів, тому що автоматизація певних процесів і переведення їх у онлайн форму, зменшує або взагалі виключає можливість корупційних схем, або можливість вчинення інших не зовсім добропорядних речей.

Мета дослідження: Метою даної роботи є імплементація функціоналу в Android застосунку, для взаємодії з університетським веб сервісом по взаємодії з курсовими роботами, а також створення та налаштування авторизації за акаунтом Microsoft університетської організації, необхідна для аутентифікації при запитах до вищезгаданого веб сервісу.

Також, варто зазначити, що дана тема була обрана, через принцип обов’язку перед університетом та загалом перед могилянською спільнотою. Розробка функцій авторизації та взаємодії з курсовими роботами трішки спростить могилянцям, які використовують Android пристрої, взаємодію з сервісами по курсовим роботам.

Об’єкт, предмет: Об’єкта дослідження – підходи та інструменти при розробці застосунків під операційну систему Android. предмет дослідження – розробка існуючого застосунку «KMA Smart»: впровадження авторизації та функціоналу роботи з курсовими.

Структура роботи: Розділ 1 - розгляд і порівняння архітектурних підходів.

Розділи 2-3 Розгляд структури і підходів в існуючій версії застосунку та опис використовуваних технологій.

Розділи 4 - процес і проблеми впровадження нового функціоналу.

Розділ 5 - пропозиції для подальшої розробки.

Методологія дослідження: Аналіз і аналітика – було проаналізовано найпоширеніші архітектурні підходи та інструменти при розробці під операційну систему Android

Кейс-стаді: було проаналізовано більш детально підходи та проблеми при розробці клієнтського Android застосунку з використанням архітектурного патерну Model-View-ViewModel.

Практична цінність: Для читача, ця кваліфікаційна робота може вважатись лаконічним вступом, як почати розробляти сучасні клієнтські застосунки для комунікації з сервером під операційну систему Android, маючи розуміння про базові речі в розробці під Android та в програмуванні загалом. Тут розглянуті та порівняні деякі часто використовувані архітектурні патерни, також присутній короткий огляд популярних інструментів та бібліотек для розробки під операційну систему Android. В розрізі, також продемонстровано як в Android застосунку налаштувати авторизацію через Microsoft акаунт та використовуючи отриманий токен, взаємодіяти з веб сервісом.

Очікувані результати: Розгляд підходів та інструментів у розробці застосунків під операційну систему Android. Імплементація авторизації та функціоналу для роботи з курсовими в існуючий Android застосунок. Продуктом зможуть користуватись студенти та викладачі, як більш зручною альтернативою веб версії.

1. РОЗГЛЯД І ПОРІВНЯННЯ АРХІТЕКТУРНИХ ПІДХОДІВ В РОЗРОБЦІ ЗАСТОСУНКІВ ПІД ОС ANDROID

Вибір архітектури для Android додатку є дуже важливим кроком у розробці, оскільки архітектурний підхід визначає спосіб організації та взаємодії між компонентами розроблюваного додатку. Існує кілька популярних архітектурних підходів для розробки Android застосунків, і вибір зазвичай залежить від факторів, таких як розмір проекту, кількість членів команди розробників, вимоги до масштабованості та інші. Перед впровадженням use cases згідно з бізнес логікою, було проаналізовано декілька популярних підходів у архітектурі Android застосунків. А також переглянуто архітектурні рішення і підходи існуючої версії розроблюваного застосунку.

1.1 Найпоширеніші підходи

Model-View-Controller (MVC) – одна з найпростіших архітектур для Android додатків. Вона розділяє додаток на модель (логіку даних), представлення (відображення інтерфейсу користувача) і контролер (управління взаємодією між моделлю та представленням). Використання MVC дозволяє вирішити проблему розділення відповідальностей, але може призвести до проблем зі зростанням розміру додатку та складності управління залежностями.

Model-View-Presenter (MVP) - покращена версія MVC, де презентер виступає посередником між моделлю та представленням. Презентер відповідає за обробку даних та управління взаємодією з користувачем. MVP полегшує тестування та розширення функціональності, але вимагає додаткового зусилля для підтримки і підтримки залежностей.

Model-View-ViewModel (MVVM): MVVM є одним з найпоширеніших архітектурних підходів в Android розробці. Вона використовує модель (логіку даних), представлення (відображення інтерфейсу користувача) та модель-представлення-вигляд (ViewModel), який зберігає стан представлення та надає дані для відображення. ViewModel взаємодіє з моделлю для отримання та обробки даних, а також надає методи та зв'язки для забезпечення взаємодії з

представленням. MVVM дозволяє розділити логіку даних від їх відображення, полегшує тестування, покращує модульність та дозволяє легко змінювати візуальну частину без впливу на логіку даних.

Clean Architecture (Чиста архітектура) - це концепція розробки, яка покликана забезпечити високу залежність від модулів та розділити їх на рівні залежностей. Вона включає в себе різні шари, такі як презентаційний шар, доменний шар та шар джерел даних. Цей підхід дозволяє забезпечити високу модульність, залежність від інтерфейсів та простоту тестування, але може бути більш складним у впровадженні та підтримці.

1.2 Порівняння підходів MVC, MVP та MVVM

В наступній таблиці розглянуто основні відмінності в розподілі обов'язків між компонентами застосунку.

Таблиця 1.1 – Розподіл обов'язків компонентів при різних підходах

	MVC	MVP	MVVM
Розподіл обов'язків компонентів	**Model**: управління даними та бізнес логіка **View**: відображення даних та обробка дій користувача **Controller**: керування взаємодією між моделлю та представленням (з Model та View)	**Model**: управління даними та бізнес логіка **View**: відображення даних та передання дій користувача презентеру **Presenter**: оброблення ді1 користувача та взаємодія з моделлю та представленням. (з Model та View)	**Model**: управління даними та бізнес логіка **View**: відображення даних та передання дій користувача до ViewModel **ViewModel**: керування взаємодією між моделлю та представленням (з Model та View)

Таблиця 1.2 демонструє залежність компонентів від платформи при різних архітектурних підходах:

Таблиця 1.2 – Залежність компонентів від платформи при різних підходах

	MVC	MVP	MVVM
Залежність від платформи	Model може бути повторно використана, але View та Controller зазвичай залежать від платформи.	Model може бути повторно використана, але View та Presenter зазвичай залежить від платформи.	Model та ViewModel можуть бути повторно використані, View, зазвичай залежить від платформи

Таблиця 1.3 показує загальні висновки про тестування застосунків з різними підходами:

Таблиця 1.3 – Тестування при різних підходах

	MVC	MVP	MVVM
Тестування	З тестуванням можуть виникати певні складнощі, через пряму залежність між компонентами	Презентер може бути замінений мок-об’єктом, тому тестування проходить простіше	Тестування проходить легше, через те, що ViewModel не залежить від UI компонентів

Вище було наведено та порівняно кілька найчастіше використовуваних архітектурних підходів, доступних для розробки Android додатків. Вибір архітектури залежить від цілей та потреб, командної співпраці, рівня складності проекту, наявних знань у команди та досвіду в розробці, а також відповідності конкретним вимогам додатку. Перед початком проектування важливо ознайомитися з кожним з підходів, вивчити їх переваги та недоліки, і обрати той, який найоптимальніше підходить для вирішення поставленого завдання.

Остаточний вибір архітектури залежить від вподобань команди розробників та від конкретних вимог до проекту. Важливо ретельно зважити на переваги та недоліки кожного підходу, а також забезпечити зрозумілість, читабельність та підтримку коду в майбутньому.

У процесі проектування та розробки Android додатку, власне після формування вимог до продукту, важливо ознайомитись з архітектурними підходами, та інструментами. Наприклад, певні інструменти і бібліотеки, властиво та оптимально використовувати при певних архітектурних підходах.

2. РОЗГЛЯД СТРУКТУРИ І ПІДХОДІВ В ІСНУЮЧІЙ ВЕРСІЇ ЗАСТОСУНКУ

Існуюча версія «KMA Smart» передбачає можливість переглянути поточні етапи вступної кампанії, останні новини, інформацію про університет.

В навігаційному меню є три вкладки: «Головна», «Спеціальності», «Налаштування». Додаток працює з мережевими запитами для оновлення інформації з віддаленої бази даних а також для отримання медіа з онлайн сховища.

2.1 Структура застосунку

Застосунок розбитий на декілька директорій, які групують компоненти за певною логікою.

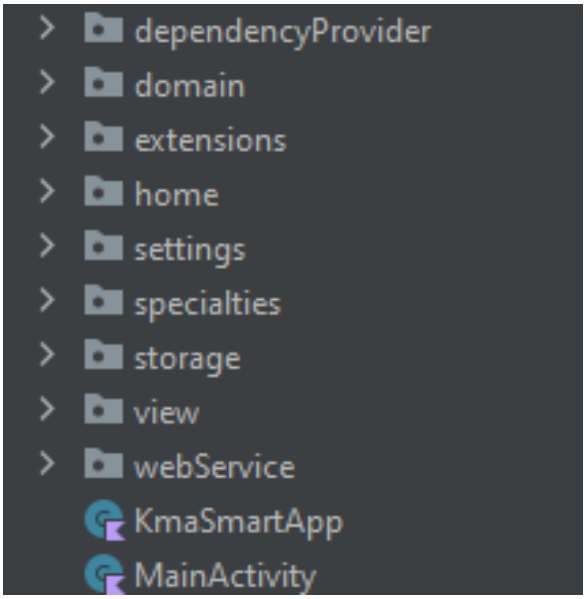


Рис 2.1 – Приклад структури source-директорій

Пакет dependencyProvider містить компоненти необхідні для впровадження DI.

Пакет domain містить data класи, які описують предметну область.

Пакет extensions містить компоненти утиліти та розширення.

Пакет view включає компоненти представлення, які можуть бути перевикористані.

Пакет storage зберігає компоненти, для роботи зі сховищем даних.

Пакет webService - для роботи з мережевими запитами.

Функціонал застосунку розбитий на 3 директорії відповідно до розділів в навігаційному меню:

- home
- settings
- specialities

Власне, кожна з них містить піддиректорії «View» та «ViewModel». В вкладених директоріях View знаходяться компоненти, специфічні власне для даного розділу застосунка. ViewModel містить логіку роботи з даними та необхідні для цього інтерфейси.

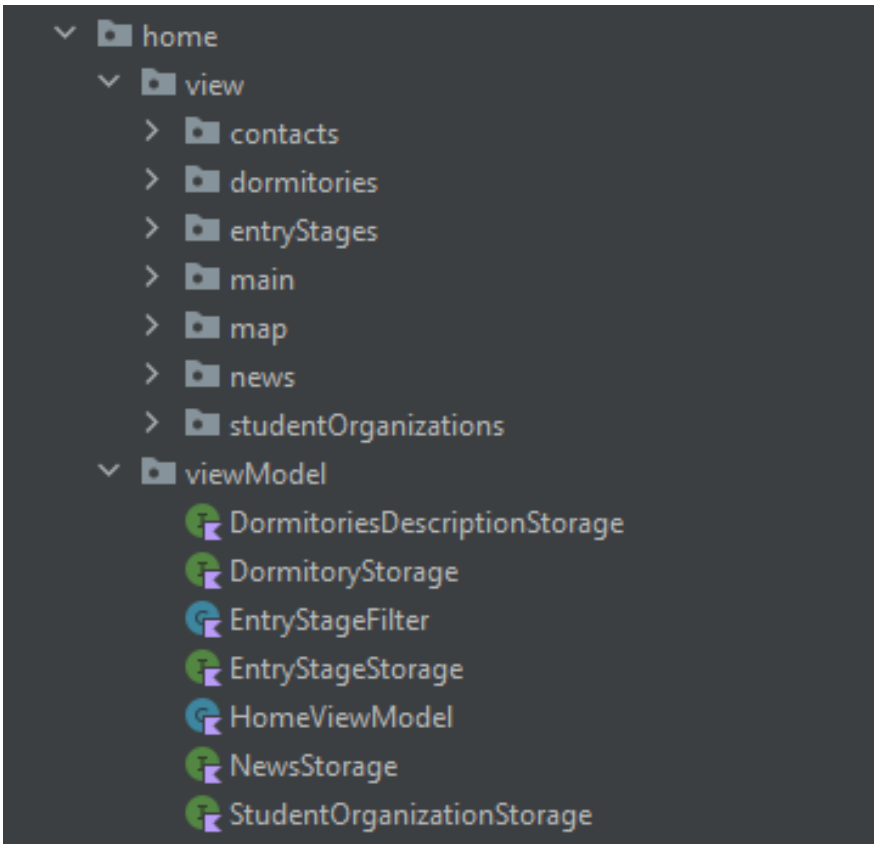


Рис 2.2 – Приклад структури директорії, яка відповідає за певний функціональний розділ програми

2.2 Розгляд підходів в розробці

- В розробці застосунку «KMA Smart» використовується архітектурний патерн MVVM з компонентами, які надає Android. Даний підхід спрощує масштабованість, роботу в команді та читабельність коду.

- Для мережевих запитів використовується бібліотека Retrofit.

- Для кешування даних в локальну базу використовується бібліотека Room.

- Для роботи зі зображеннями – бібліотека Glide.

- Використовується бібліотека Koin як DI провайдер.

- Для локальних налаштувань використовується бібліотека datastore.

Детальний огляд цих інструментів проведений в наступному розділі – «Опис використаних технологій»

3. ОПИС ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

3.1 MVVM в Android розробці

Для розробки «KMA Smart» використовувався архітектурний патерн MVVM. На момент написання роботи - це один з найбільш часто використовуваних підходів при розробці для Android. Це зумовлено тим, що при використанні MVVM, спрощується тестування, зменшується залежність певної частини коду від платформи. Спрощення тестування відбувається за рахунок зменшення залежності між компонентами та в свою чергу, через відсутність залежності певних компонентів від платформи Android.

Також, що дуже важливо, якщо існує задача випустити додаток під різні платформи, з MVVM доволі значу частину коду можна перевикористати при розробці під іншу платформу.

В сучасному світі досить важко знайти популярний сервіс, який розробляє клієнтський застосунок лише під одну платформу. Пам’ятаючи, що MVVM один із способів максимізувати платформо-незалежну частину коду при розробці – розуміємо, що цей архітектурний патерн відчутно спрощує завдання для команд розробників, що планують створити додаток під різні платформи.

Можливо, це одна із основних причин такої популярності підходу MVVM.

3.2 Основні принципи архітектурного підходу MVVM в Android розробці

Далі підкреслено основні принципи архітектурного підходу MVVM в Android розробці.

- Розділення логіки

  MVVM дотримується принципу розділення логіки бізнесу від представлення даних та їх відображення на екрані. Model відповідає за управління даними та бізнес-логікою. View відображає дані та обробляє користувацький ввід. ViewModel виступає як посередник між Моделлю та Представленням, надаючи дані, необхідні для відображення та обробки.

- Двосторонній зв’язок даних

  Одним із ключових принципів MVVM є двосторонній зв’язок даних між Представленням та ViewModel. ViewModel надає дані, необхідні для відображення на Представленні, і відслідковує зміни введення користувача. У свою чергу, Представлення оновлюється залежно від змін у ViewModel, що забезпечує синхронізацію даних між компонентами.

- Використання даних з усіх джерел

  ViewModel може отримувати дані з різних джерел, таких як база даних, веб-сервіси або кеш. Це дозволяє ізолювати Представлення від джерела даних та спрощує тестування.

- Одиничність відображення

  У MVVM кожен компонент Представлення має свій власний ViewModel. Це дозволяє легко відокремити логіку Представлення від даних та зберігати їх стан незалежно.

- Використання реактивних бібліотек

  Для реалізації двостороннього зв’язку даних та спрощення управління станом використовуються реактивні бібліотеки, такі як LiveData або RxJava. Вони дозволяють автоматично оновлювати View.

- Інверсія залежності

  MVVM використовує принцип інверсії залежності, що означає, що Представлення не залежить від конкретної реалізації ViewModel, а залежить лише від її інтерфейсу. Це дозволяє замінювати різні реалізації ViewModel без впливу на Представлення.

- Тестованість

  MVVM сприяє легкості тестування, оскільки логіка бізнесу міститься в окремому класі ViewModel, який можна тестувати без залежності від

Android-компонентів. Можна створити модульні тести для перевірки правильності роботи ViewModel.

- Розширюваність

  MVVM забезпечує хорошу розширюваність, оскільки можна додавати нові функціональності, додавати нові ViewModel і змінювати Представлення без впливу на інші компоненти.

- Чистота коду

  За допомогою MVVM можна зберігати код чистим і добре організованим. Логіка бізнесу розміщується в ViewModel, що дозволяє відокремити її від Представлення та даних.

- Підтримка Jetpack

  MVVM є рекомендованою архітектурою для розробки Android-додатків з використанням Jetpack компонентів, таких як LiveData, ViewModel, Room, Data Binding тощо. Використання MVVM спрощує інтеграцію з цими компонентами та забезпечує сумісність з іншими Jetpack функціональностями.

Загалом, MVVM пропонує структурований та модульний підхід до розробки Android додатків, забезпечуючи розділення логіки бізнесу, легкість тестування та високу розширюваність. Він є популярним вибором для розробки Android-додатків, особливо при використанні Jetpack функціональностей, таких як LiveData, ViewModel, Room, Data Binding та інших.

3.3 Dependency Injection

Dependency Injection (DI) є шаблоном проєктування в розробці програмного забезпечення (включаючи і розробку Android додатків), де залежності між класами вводяться зовні, замість того, щоб бути створеними всередині самого класу. DI дозволяє полегшити керування залежностями, збільшує зв'язаність та покращує тестування та перевикористання коду.

В наступній таблиці наведені основні компоненти DI.

Таблиця 3.1 – Основні компонентами DI

Залежності (Dependencies)	Це об’єкти, з якими клас взаємодіє або від яких він залежить. Наприклад, можуть бути залежності від служб, репозиторіїв або інших класів.
Контейнер ін’єкції залежностей (Dependency Injection Container)	Це механізм, який керує створенням та постачанням залежностей. Контейнер DI дозволяє вказати, які залежності повинні бути створені та введені в класи.
Конфігурація (Configuration)	Це місце, де визначаються правила для створення та постачання залежностей. Це може бути файл конфігурації або код, який налаштовує контейнер DI.
Ін’єкція залежностей (Dependency Injection)	Це процес постачання залежностей в класи. Ін’єкція може бути здійснена за допомогою конструктора, методів або властивостей класу.

DI дозволяє досягнути таких переваг:

- Зменшення зв’язності (Decoupling)

  Класи стають менш залежними один від одного, оскільки їх залежності вводяться ззовні. Це полегшує зміну або заміну залежностей без впливу на весь код.

- Покращення тестування

Ізольоване тестування стає простішим, оскільки можна легко підставляти мок-об’єкти.

- Перевикористання коду

  Залежності можуть бути використані у різних класах та модулях додатку. Це забезпечує більшу гнучкість та зменшує дублювання коду.

- Легка конфігурація

  Контейнери DI надають зручні механізми для конфігурації залежностей. Можливість налаштувати залежності за допомогою файлів конфігурації або анотацій спрощує процес налаштування.

- Розширюваність

  Зміна або додавання нових залежностей може бути легко здійснена без модифікації вже існуючого коду. Це робить додаток більш гнучким та легким для розширення.

У контексті розробки Android-додатків, DI можна використовувати для управління залежностями між класами, такими як репозиторії, служби, мережеві клієнти тощо. Використання DI-контейнера, такого як Koin, дозволяє зручно визначати та постачати залежності в класи Android-додатків. Також є інші відомі DI бібліотеки для розробки Android застосунку, зокрема, такі як Dagger, Hilt.

Загалом, використання DI в Android додатках сприяє покращенню модульності, тестування, перевикористанню та керуванню залежностями. Цей шаблон проєктування дозволяє розробникам ефективно використовувати компоненти, такі як Room, Retrofit, DataBinding та Glide, у своїх додатках, спрощуючи розробку та підтримку коду.

3.4 Шаблон репозиторію

Шаблон репозиторію (Repository Pattern) є підходом до організації доступу до даних у програмному забезпеченні. Він дозволяє абстрагувати доступ до даних

від рівня логіки додатку, що забезпечує відокремлення логіки бізнесу від деталей роботи з даними.

Основна ідея репозиторію полягає в тому, що він надає єдиний інтерфейс для взаємодії з даними незалежно від того, яке джерело даних використовується (наприклад, база даних, веб-сервіс або кеш). Репозиторій включає методи для отримання, збереження, оновлення та видалення даних.

Далі, наведено основні переваги використання репозиторію.

- Відокремлення логіки додатку від деталей роботи з даними

  Репозиторій дозволяє розділити логіку бізнесу від логіки доступу до даних. Це дозволяє зберігати логіку бізнесу незалежною від конкретного джерела даних і спрощує підтримку коду.

- Покращена тестованість

  Використання репозиторію полегшує тестування логіки додатку. Можна створити підроблену (mock) реалізацію репозиторію для забезпечення тестування без прив’язки до реальних джерел даних.

- Легкість зміни джерела даних

  Репозиторій дозволяє змінювати джерело даних, не впливаючи на логіку додатку. Наприклад, можна легко переключитися з бази даних на веб-сервіс або навпаки, змінивши реалізацію репозиторія.

- Повторне використання

  Репозиторій може бути використаний повторно в інших частинах додатку або в інших проектах.

- Централізоване керування

  Використання репозиторія дозволяє централізувати логіку доступу до даних. Це означає, що всі запити до даних пройдуть через репозиторій, що спрощує керування та моніторинг цих операцій.

- Стандартизація

Репозиторій встановлює стандартизований набір методів для взаємодії з даними. Це полегшує спільну роботу над проектом між розробниками і сприяє чистоті та чіткості коду.

- Збереження історії

  Репозиторій може включати функціональність збереження історії змін даних. Це дозволяє відстежувати, коли та які зміни були внесені до даних, що може бути корисним для аудиту та відновлення.

Варто враховувати, що репозиторій - це шаблон, і його реалізація може відрізнятися в залежності від конкретних потреб проекту. Можна створити власний репозиторій або скористатися готовими бібліотеками, які вже містять реалізацію репозиторію, наприклад, Room Persistence Library для роботи з базою даних в Android.

3.5 Retrofit

Retrofit є однією з найпопулярніших бібліотек для роботи з мережевими запитами в Android-розробці. Вона надає простий і зручний спосіб взаємодії з веб-службами, зокрема з API на основі REST.

Далі наведено проблеми, які Retrofit допомагає вирішити в Android-розробці.

- Управління мережевими запитами

  Retrofit дозволяє визначати інтерфейси, які відображають структуру веб-служби. Користувачі бібліотеки можуть описати методи, шляхи, параметри та типи даних в цих інтерфейсах. Цей інструмент надає зручну абстракцію над підключенням до сервера і обробкою мережевих запитів.

- Серіалізація і десеріалізація даних

  Retrofit вбудовує у себе бібліотеку конвертерів, яка дозволяє автоматично перетворювати дані, передані через мережу, у відповідні об’єкти у розроблюваній програмі і навпаки. Залежно від потреби можна

використовувати різні формати серіалізації, такі як JSON, XML або Protobuf.

- Обробка помилок

  Retrofit надає можливість визначати власні обробники помилок для мережевих запитів. Користувачу бібліотеки можна визначити, як обробляти різні коди статусу HTTP і відповіді з сервера, щоб забезпечити гнучку логіку обробки помилок у розроблюваному додатку.

- Планування мережевих запитів

  Retrofit дозволяє розподіляти запити на різні потоки і виконувати їх асинхронно. Можна використовувати RxJava або Kotlin Coroutines, щоб керувати асинхронним виконанням запитів і отримувати результати в зручний спосіб.

- Кешування даних

  Retrofit підтримує можливість кешування мережевих відповідей, що дозволяє зберігати копії відповідей і повторно використовувати їх при подальших запитах. Це допомагає зменшити навантаження на сервер і покращити швидкодію додатку.

- Інтерактивна документація

  Retrofit надає засоби для автоматичної генерації інтерактивної документації для заданої API. Можна використовувати бібліотеки, такі як Swagger або RAML, щоб описати API, а потім згенерувати документацію, яка надає приклади коду та описи різних запитів.

- Підтримка різних аутентифікаційних механізмів

  Retrofit дозволяє встановлювати заголовки аутентифікації для запитів і підтримує різні механізми аутентифікації, такі як базова аутентифікація, OAuth і Bearer токени.

- Масштабованість

Retrofit дозволяє легко налаштовувати розширення і інтерсептори, які додають додаткову функціональність до клієнта. Можна використовувати ці можливості для додавання логування, моніторингу, перехоплення запитів.

- Підтримка різних платформ

  Retrofit не обмежується лише Android розробкою. Його можна використовувати із Java, Kotlin або будь-якою іншою платформою, яка підтримує Java Virtual Machine (JVM). Це дозволяє перевикористовувати код і спільно використовувати логіку мережевого взаємодії між різними проектами і платформами.

Приклад створення запитів за допомогою retrofit в застосунку «KMA Smart»:

```
import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory
import retrofit2.http.GET
import retrofit2.http.Query


interface UserInfoServerClient {
    @GET("/open-api/news/list")
    suspend fun getAllNews(
        @Query("fields") fields: Set<String> = NewsDto.Fields.all,
        @Query("restrict") restrict: String? = null
    ): ResponseMapDto<NewsDto>

    @GET("/open-api/entry-step/list")
    suspend fun getAllEntryStages(
        @Query("fields") fields: Set<String> = EntryStageDto.Fields.all,
        @Query("restrict") restrict: String? = null
    ): ResponseMapDto<EntryStageDto>
```

Рисунок 3.1 – Приклад створення мережевих запитів з використанням retrofit

Наступний рисунок показує структуру класу WebService, який відповідає за роботу з мережевими запитами.

В коді з рисунку продемонтсровано створення класу WebService, який має три приватні властивості (userInfoServer, courseWorkServer і linkProvider). Ці властивості є залежностями, переданими через конструктор класу.

userInfoServer - це об’єкт класу UserInfoServerClient, який представляє клієнт для отримання інформації про користувача з сервера.

courseWorkServer - це об’єкт класу CourseWorkServerClient, який представляє клієнт для взаємодії з сервером курсових робіт.

linkProvider - це об’єкт класу StorageWebServiceMediaProvider, який представляє постачальника медіафайлів з веб-сервісу зберігання.

Ці залежності ін’єктуються (передаються) в конструктор WebService ззовні. Це означає, що при створенні об’єкту WebService, потрібно надати об'єкти userInfoServer, courseWorkServer і linkProvider. Це реалізовується за допомогою Dependency Injection, де залежності надаються зовні, що дозволяє легко замінювати реалізації залежностей і полегшує тестування та розширення класу WebService, наприклад, коли потрібно додати роботу з іншим сервером.

```
class WebService(
    private val userInfoServer: UserInfoServerClient,
    private val courseWorkServer: CourseWorkServerClient,
    private val linkProvider: StorageWebServiceMediaProvider
) {
```

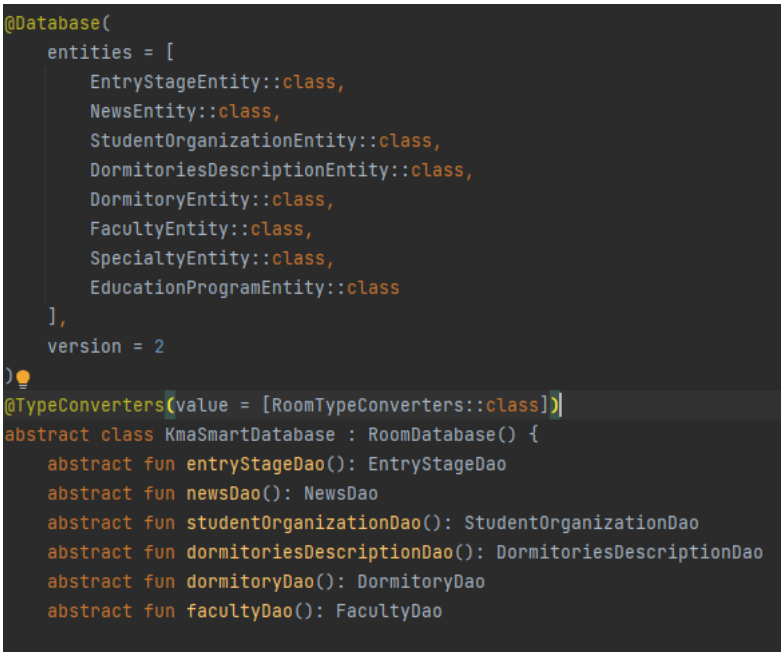
Рисунок 3.2 – WebService – клас для роботи з усіма мережевими запитами

В цілому, Retrofit спрощує роботу з Android розробкою, забезпечуючи зручний та ефективний спосіб взаємодії з веб-службами. Бібліотека допомагає зменшити кількість коду, що потрібно написати, спрощує управління мережевими запитами, автоматизує серіалізацію та десеріалізацію даних, надає можливості для обробки помилок та планування запитів. Крім того, Retrofit має ряд додаткових функціональностей, таких як кешування даних, генерація інтерактивної документації, підтримка різних аутентифікаційних механізмів і розширюваність.

Загалом, Retrofit є потужним інструментом для розробників застосунків під операційну систему Android, який допомагає зосередитися на логіці додатку, забезпечує ефективну взаємодію з веб-службами і прискорює процес розробки.

3.6 Room

Room є бібліотекою для роботи з базами даних в Android-розробці, яка надає рівень абстракції над SQLite і спрощує взаємодію з базою даних. На наступному рисунку показано приклад використання бібліотеки Room.



```
@Database(
    entities = [
        EntryStageEntity::class,
        NewsEntity::class,
        StudentOrganizationEntity::class,
        DormitoriesDescriptionEntity::class,
        DormitoryEntity::class,
        FacultyEntity::class,
        SpecialtyEntity::class,
        EducationProgramEntity::class
    ],
    version = 2
)
@TypeConverters(value = [RoomTypeConverters::class])
abstract class KmaSmartDatabase : RoomDatabase() {
    abstract fun entryStageDao(): EntryStageDao
    abstract fun newsDao(): NewsDao
    abstract fun studentOrganizationDao(): StudentOrganizationDao
    abstract fun dormitoriesDescriptionDao(): DormitoriesDescriptionDao
    abstract fun dormitoryDao(): DormitoryDao
    abstract fun facultyDao(): FacultyDao
```

Рисунок 3.3 – Приклад створення БД за допомогою засобі Room

Цей код визначає абстрактний клас KmaSmartDatabase, який є підкласом RoomDatabase. Він використовується для визначення та отримання різних DAO (Data Access Object) для доступу до відповідних даних у базі даних.

Анотація @TypeConverters вказує, що потрібно використовувати певний клас конвертерів типів для цієї бази даних. У даному випадку, значення анотації value вказує на клас RoomTypeConverters, який містить реалізацію конвертерів типів.

Клас KmaSmartDatabase має кілька абстрактних функцій, які представляють DAO для доступу до різних таблиць бази даних. Кожна функція повертає об'єкт DAO, який може бути використаний для виконання операцій з базою даних для відповідної таблиці.

Наприклад, функція entryStageDao() повертає об’єкт EntryStageDao, який є DAO для доступу до таблиці EntryStage в базі даних. Аналогічно, інші функції повертають DAO для відповідних таблиць, такі як newsDao(), studentOrganizationDao(), dormitoriesDescriptionDao(), dormitoryDao() і facultyDao(). Анотація @Database визначає структуру та версію ьд.

Далі буде наведено переваги використання Room.

- Спрощена робота з базою даних

  Room забезпечує простий і зрозумілий інтерфейс для роботи з базою даних, включаючи створення таблиць, вставку, оновлення та вибірку даних. Користувачам бібліотеки можна використовувати анотації для позначення моделей даних, запитів і асоціацій між таблицями, що дозволяє швидко і зручно визначати структуру бази даних.

- Обробка міграцій

  Room надає механізми для міграції бази даних при зміні її структури. Можна визначити версії бази даних і надати міграційні скрипти, які виконуються автоматично при оновленні додатка. Це дозволяє зберегти існуючі дані після оновлення схеми бази даних і запобігає втраті даних.

- Валідація даних

  Room дозволяє використовувати анотації для валідації даних, перед тим як вони будуть збережені в базу даних. Можна визначити правила перевірки, такі як обов’язкові поля, обмеження значень або унікальність даних, що допомагає забезпечити цілісність даних і запобігає некоректним записам у базу.

- Запити з компіляцією на етапі компіляції

Room надає підтримку запитів SQL, які компілюються на етапі компіляції. Це дозволяє виявляти синтаксичні помилки і проблеми з виконанням запитів на ранньому етапі розробки, що допомагає уникнути помилок під час виконання програми. Крім того, компіляція запитів на етапі компіляції може забезпечити покращену швидкодію виконання запитів.

- Підтримка LiveData та RxJava

  Room інтегрується з архітектурним компонентом LiveData та бібліотекою RxJava, що дозволяє отримувати зміни даних в реальному часі. Як наслідок, можна спостерігати базу даних на зміни і автоматично оновлювати інтерфейс користувача відповідно до змін.

- Асинхронні запити

  Room дозволяє виконувати запити до бази даних асинхронно, уникнути блокування основного потоку та покращити загальну швидкодію додатка. Розробник, що працює з цією бібліотекою, може використовувати асинхронні методи для виконання запитів і отримання результатів за допомогою Kotlin Coroutines, RxJava або використовувати об’єкти LiveData для асинхронного отримання даних.

- Тестування

  Room надає підтримку для тестування бази даних. Розробник може створювати тестові бази даних у пам’яті, заповнювати їх тестовими даними і перевіряти результати запитів і маніпуляцій з даними без реального зв’язку з базою даних.

Загалом, Room спрощує роботу з базами даних в Android розробці, надаючи зручний інтерфейс, підтримку міграцій, валідацію даних, компіляцію запитів на етапі компіляції та інтеграцію з архітектурними компонентами Android. Вона допомагає забезпечити ефективне взаємодію з базою даних, зменшити кількість коду та спростити тестування.

3.7 Koin

Koin - це легка бібліотека для управління залежностями (dependency injection) в Android розробці.

Наступний рисунок демонструє використання Koin в «KMA Smart»:

Рисунок 3.4 – Управління залежностями за допомогою Koin

Наведений код використовує фреймворк Koin для визначення модуля залежностей (appModule). В цьому модулі визначаються різні компоненти, такі як класи ViewModel, клієнти серверів, провайдери зберігання та база даних.

Далі наведено переваги використання Koin в розробці додатків під операційну систему Android.

- Управління залежностями

  Koin дозволяє легко визначати та впроваджувати залежності в розроблюваному Android додатку. Замість того, щоб вручну створювати об’єкти та встановлювати їх залежності, можна використовувати Koin для автоматичного створення та впровадження залежностей відповідно до визначених правил.

- Легке налаштування

  Koin пропонує простий та декларативний підхід до конфігурації залежностей. Потрібно визначити модулі, в яких описуються залежності, та використовувати ін’єкцію залежностей (dependency injection) для отримання цих залежностей в потрібних місцях коду застосунку.

- Легка інтеграція

  Koin надає простий та зрозумілий API для інтеграції залежностей в Android додатки. Вона інтегрується з природними компонентами Android, такими як Activity та Fragment, і пропонує прості методи для отримання залежностей в цих компонентах.

- Модульність

  Koin підтримує модульну структуру, що дозволяє визначати залежності для окремих модулів додатку. Це допомагає зберігати код впорядкованим та полегшує тестування і обслуговування, та зачасту використовується при підході з використанням clean architecture.

- Тестування

  Koin допомагає полегшити тестування Android коду, надаючи можливість замінювати реальні залежності на фальшиві або мок-об’єкти. Це дозволяє легко створювати тести, що ізолюють окремі компоненти додатку та перевіряють їх коректність роботи.

- Легкість підтримки

  Koin пропонує простий синтаксис та мінімальну кількість конфігурації, що робить його легким у використанні та підтримці. Він має невеликий розмір і мінімальну кількість залежностей, що спрощує його включення до проекту та уникнення непотрібних накладних витрат.

Загалом, Koin вирішує проблеми управління залежностями в Android-розробці, допомагаючи спростити конфігурацію та впровадження залежностей, покращити

модульність та тестованість коду. Використання Koin допомогає зробити Android додаток більш організованим, гнучким та легким для підтримки у майбутньому.

3.8 Glide

Glide - це бібліотека для роботи зі зображеннями в Android-розробці. Вона допомагає вирішити наведені далі проблеми.

Glide використовується в розробці «KMA Smart» для роботи з зображеннями, наприклад, на наступному рисунку приклад налаштування зображень з новин.

```
private fun setupNewsImage(news: News) {
    val photo = news.photo ?: Media.LocalPhoto(R.drawable.ic_no_image_rectangle_10x18)
    Glide
        .with(requireContext())
        .loadMedia(news.photo)
        .placeholder(R.drawable.blurred_placeholder)
        .error(R.drawable.ic_no_image_rectangle_10x18)
        .into(binding.newsImage)
    binding.newsImage.setOnClickListener { navigateToImagePreviewFragment(photo) }
}
```

Рисунок 3.5 – *Приклад використання бібліотеки Glide*

Наприклад, виклик функції loadMedia(news.photo) вказує, яке зображення потрібно завантажити за допомогою Glide.

Виклик функції placeholder(R.drawable.blurred_placeholder) встановлює ресурс R.drawable.blurred_placeholder як зображення-заповнювача, яке показується поки основне зображення завантажується.

Виклик функції error(…) встановлює ресурс, переданий в якості аргументу, як зображення, яке буде відображатися у випадку помилки завантаження основного зображення.

Виклик функції into(binding.newsImage) вказує, куди необхідно завантажити зображення. В даному випадку, binding.newsImage вказує на ImageView, де буде відображатися зображення.

Отже, функція setupNewsImage використовує бібліотеку Glide для завантаження зображення з використанням вказаних ресурсів заповнювача та помилки.

Далі згруповано переваги роботи з Glide.

- Завантаження та кешування зображень

  Glide дозволяє легко завантажувати зображення з різних джерел, таких як локальні ресурси, мережа, контент провайдери або URL-адреси. Вона автоматично кешує завантажені зображення, що допомагає уникнути повторного завантаження та зайвого навантаження на мережу.

- Зменшення розміру зображень

  Glide надає можливість зменшити розмір зображень перед їх відображенням. Це допомагає економити пропускну здатність мережі та зменшує час завантаження зображень, особливо при роботі з великими або важкими файлами зображень.

- Обробка зображень

  Glide має потужні функції обробки зображень, такі як обрізка, масштабування, обертання, налаштування кольорів, прозорості та інші. Це дозволяє зручно змінювати зображення перед його відображенням, щоб вони відповідали потребам додатку.

- Асинхронне завантаження зображень

  Glide працює асинхронно, що дозволяє завантажувати зображення в фоновому режимі без блокування головного потоку. Це покращує продуктивність додатку та запобігає затримкам під час завантаження зображень.

- Контроль над життєвим циклом зображень

  Glide автоматично управляє життєвим циклом завантажених зображень, дозволяючи зручно вирішувати проблеми пам’яті та уникати витоків пам’яті.

- Коректне відображення зображень

  Glide забезпечує оптимізоване відображення зображень з урахуванням розмірів і характеристик відображаючого компонента, такого як ImageView. Вона автоматично масштабує зображення до необхідних розмірів та здійснює оптимізоване відображення, щоб забезпечити гладке та ефективне відтворення зображень.

- Підтримка анімацій

  Glide дозволяє використовувати анімації при завантаженні та відображенні зображень. Вона надає можливість налаштувати анімаційні ефекти для плавного переходу між зображеннями або відображенням прогресу завантаження.

- Легкість інтеграції

  Glide легко інтегрується з Android проектами, надаючи простий API та можливість налаштування. Вона підтримує інтеграцію з різними бібліотеками та фреймворками, такими як AndroidX, Retrofit, OkHttp та інші.

Загалом, Glide вирішує проблеми роботи зі зображеннями в Android розробці, забезпечуючи легке завантаження, кешування, обробку та відображення зображень з оптимізованою продуктивністю. Використання Glide допомагає забезпечити коректне відображення зображень у розроблюваному додатку та покращити його продуктивність.

4. ПРОЦЕС ВПРОВАДЖЕННЯ НОВОГО ФУНКЦІОНАЛУ ТА ВИРІШЕННЯ ПРОБЛЕМ

Першим етапом у процесі впровадження нового функціоналу до застосунку «KMA Smart» було створення плану. Пункти цього плану 1,2, 6 були розглянуті в попередніх розділах даної роботи .

4.1 План впровадження нового функціоналу

1. Ознайомлення з середовищем розробки Android Studio, технологіями та підходами до розробки програмного забезпечення для ОС Андроїд, а також огляд тенденцій в Android розробці.
2. Ознайомлення з наявною версією «KMA Smart», визначення використовуваних інструментів та підходів та їх детальніший огляд.
3. Ознайомлення з API сервісу, а також з логікою ролей та permissions.
4. Схематична побудова бізнес логіки (створення use cases на основі інформації про API та про ролі і їх дозволи).
5. Ознайомлення зі способом реалізації аутентифікації Microsoft OAuth за допомогою Azure AD.
6. Детальніший розгляд початкового вихідного коду застосунку, його структури та використаних підходів, використаних бібліотек та інструментів.
7. Планування структури впроваджуваного коду
8. Реалізація необхідного функціоналу
9. Нотування ідей та функцій, які можуть бути корисними для подальшої розробки.
10. Огляд результатів роботи та висновок

4.2 Схематичне представлення сценаріїв використання

Відповідно до завдання, необхідно імплементувати доступ до API для ролей «викладач» та «студент». Основною задачею є імплементація бізнес логіки для взаємодії з веб-сервісом по курсовим роботам.

Для ролі викладача, відповідно до бізнес логіки сервісу, необхідно реалізувати такі сценарії використання:

- Створити курсову роботу
- Редагувати курсову роботу
- Переглянути заявки факультету
- Видалити курсову роботу
- Покласти оцінку викладача
- Вказати відсоток плагіату

(Схематично зображено на наступному рисунку).

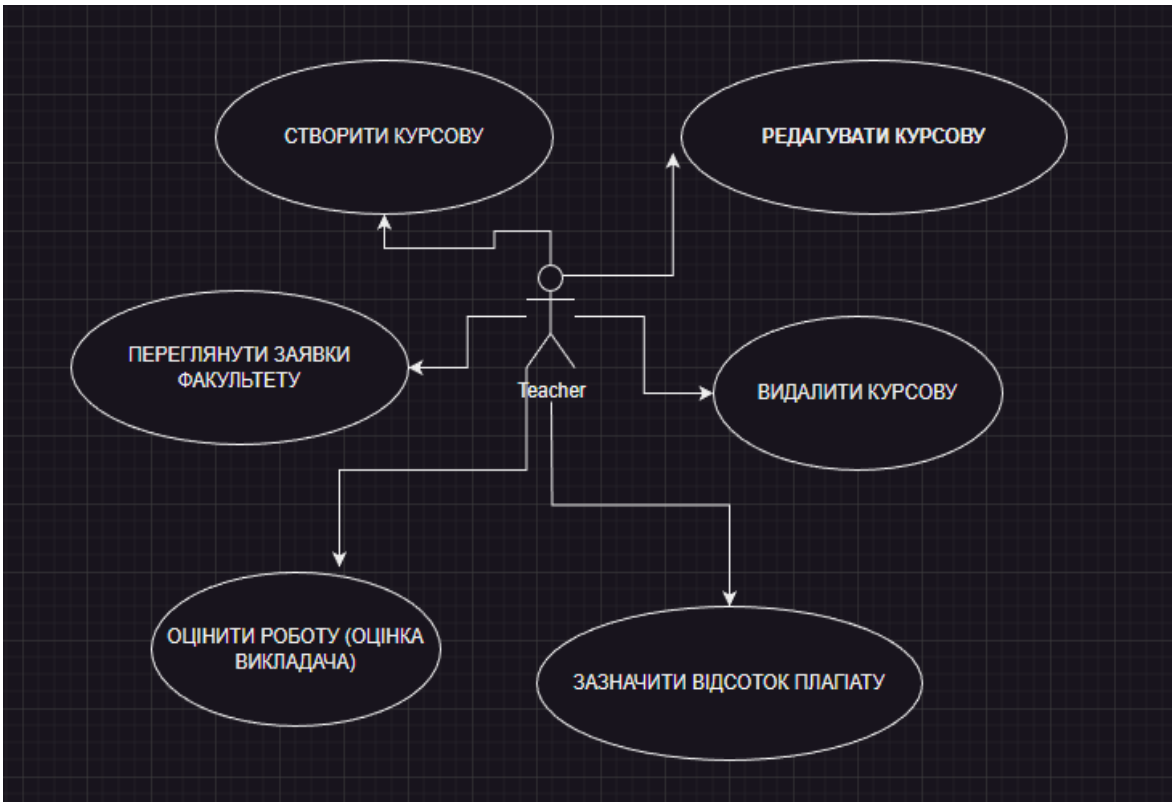


Рисунок 4.1 – use cases для ролі викладача

Для ролі студента, відповідно до бізнес логіки сервісу, необхідно реалізувати такі сценарії використання:

- Створити заявку
- Редагувати заявку
- Переглянути заявку

- Видалити заявку

(Схематично зображено на наступному рисунку).

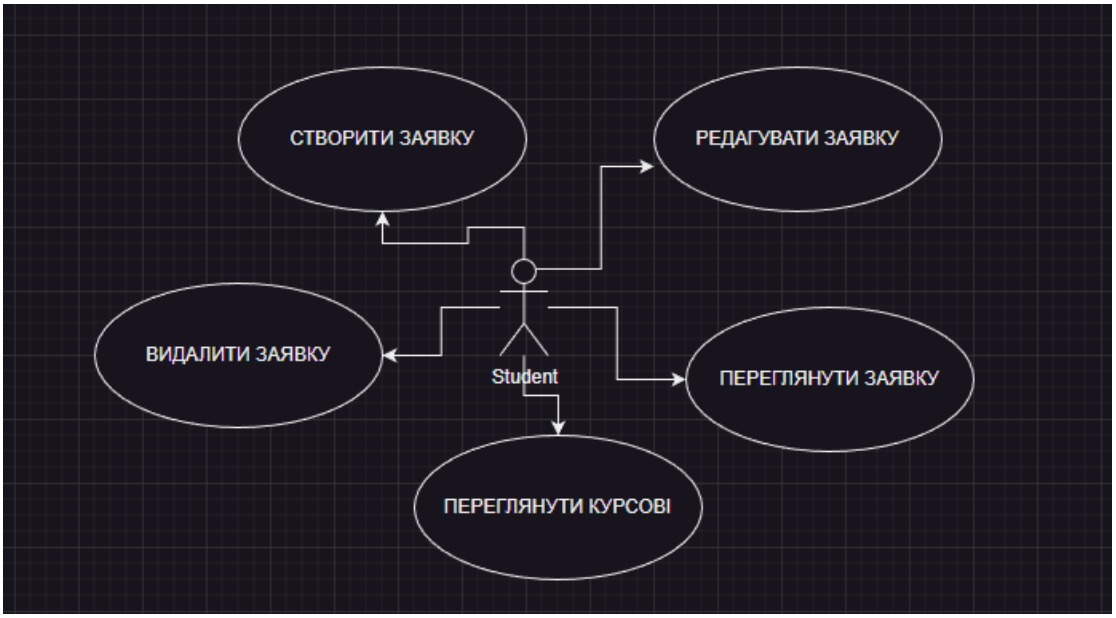


Рисунок 4.2 – use cases для ролі студента

4.3 Ознайомлення зі способом реалізації аутентифікації Microsoft OAuth за допомогою Azure AD.

Для реалізації Microsoft Azure Active Directory (Azure AD) в Android додатку з використанням архітектурного підходу MVVM можна дотримуватись подальшої інструкції.

Першим кроком буде налаштування Azure AD. Потрібно увійти до порталу Azure (portal.azure.com). Далі необхідно створити новий локатор Azure AD або використати вже наявний. Наступним кроком буде реєстрація своєї програми в Azure AD і отримання необхідного клієнтського ідентифікатора та URI перенаправлення.

Далі необхідно додати потрібні залежності до файлу build.gradle Android проєкту:

- Azure AD Authentication Library (ADAL)
- Azure AD MSAL (Microsoft Authentication Library)

Наступний крок - створення служби автентифікації в класі під назвою, наприклад AuthenticationService та реалізація необхідних методів для обробки автентифікації Azure AD, отримання токенів та оновлення токенів, використовуючи бібліотеки ADAL або MSAL. AuthenticationService можна використовувати як сінглтон або впровадити його в конкретний ViewModel.

Далі буде розглянуто, як впровадити AuthenticationService в певний ViewModel, робота з яким згідно до бізнес логіки вимагає автентифікації.

В даному компоненті ViewModel можна реалізувати методи та визначити властивості для обробки автентифікації і керування токенами. За допомогою LiveData об’єктів можна спостерігати за станом автентифікації в інтерфейсі користувача.

Далі у Fragment компоненті можна налаштувати обробники подій користувача, такі як кнопки входу/виходу з системи, і викликати відповідні методи в ViewModel для запуску процесу автентифікації.

Також можна відображати елементи інтерфейсу користувача на основі стану автентифікації.

Останній крок - реалізація потоку автентифікації Azure AD.

Коли користувач розпочинає потік входу в систему, необхідно викликати відповідний метод в AuthenticationService для початку процесу автентифікації.

Після цього отриманий токен можна використовувати згідно до бізнес логіки застосунку, який треба розробити.

В цій інструкції був описаний узагальнений підхід для реалізації Microsoft Azure AD в Android додатку з використанням архітектурного підходу MVVM. У випадку необхідності, можна звернутись до документації бібліотеки ADAL або MSAL для отримання докладних інструкцій щодо інтеграції автентифікації Azure AD в Android додатку.

4.4 Проблема розуміння предметної області

Проблема розуміння предметної області є дуже важливою в контексті розробки клієнтських Android MVVM застосунків. Далі наведено причин, чому розуміння предметної області є одним із ключових аспектів в розробці.

- Адекватне відображення функціональності

  Розуміння предметної області допомагає визначити основні функції і вимоги до застосунку. Це дозволяє створити адекватну модель даних та визначити потрібні компоненти, які забезпечать виконання цих функцій.

- Дизайн архітектури

  Розуміння предметної області сприяє вибору відповідної архітектури для розроблюваного застосунку. Воно дозволяє правильно організувати компоненти, модулі та залежності, щоб забезпечити ефективну та масштабовану структуру коду.

- Тестування та розробка функціональності

  Якщо добре розуміти предметну область, можна легше визначити, які тести потрібні для перевірки правильності реалізації функцій. Тоді можна буде переконатись, що розроблюваний застосунок працює згідно з очікуваннями користувачів та відповідає потребам предметної області.

Отже, розуміння предметної області є необхідним для успішної розробки застосунків. Воно визначає які задачі потрібно вирішити, які дані треба обробляти та які функціональні можливості повинен мати застосунок. Без належного розуміння предметної області можуть виникнути далі перераховані проблеми.

- Неправильна архітектура

  Якщо не розуміти основні концепції і потреби предметної області, можна неправильно спроектувати архітектуру розроблюваного застосунку. Це може призвести до незручної структури коду, надмірної складності або недосяжності певних функціональних можливостей.

- Неправильне моделювання даних

  Якщо не розуміти структуру та взаємозв’язки даних в предметній області, можна неправильно змоделювати дані в розроблюваному застосунку. Це може призвести до некоректного відображення даних або незручного доступу до них.

- Неправильні функціональні можливості

  Якщо не розуміти потреби користувачів і основні завдання, які треба вирішити в предметній області, можна пропустити важливі функціональні можливості у розроблюваному застосунку. Це може призвести до незадоволення кінцевих користувачів та невиконання основних вимог проєкту.

- Неправильне тестування

  Якщо не розуміти, які функції та функціональність потрібно перевірити, можна провести неправильне тестування розроблюваного застосунку. Це може призвести до незавершених або неналежно випробуваних частин застосунку, що в свою чергу може призвести до появи помилок та некоректної роботи програми.

4.5 Планування структури впроваджуваного коду

Для впровадження функціоналу згідно з раніше описаними сценаріями використання, необхідно створити data класи, які описують сутності з заданої предметної області, на рівні логіки застосунку, а також на рівні роботи бази даних, описати відповідні репозиторії, DAO, DTO.

В директорію ресурсів необхідно додати xml layout файли, які описуватимуть користувацький інтерфейс.

Для впровадження авторизації, необхідно створити декілька компонентів згідно інструкції. Після того необхідно впровадити бізнес логіку згідно ролі користувача у два відповідні компоненти ViewModel.

Для з’єднання з сервером по курсовим роботам, необхідно описати репозиторій за допомогою retrofit, і потім впровадити його в WebService.

Структура розбиття директорії на піддиректорії та логіка зберігання файлів, має відповідати структурній логіці у вже існуючій частині коду застосунку.

4.6 Проблема створення різних інтерфейсів доступу до функціоналу залежно від ролі користувача

Один із елегантних варіантів вирішення цієї проблеми – використання Android компонента навігації. Для реалізації переходу до різних інтерфейсів після автентифікації в Android MVVM з використанням компонента навігації (Navigation), можна виконати наступні кроки.

- Налаштування компонента навігації

  Спочатку необхідно налаштувати компонент навігації в розроблюваному проекті. Також треба визначити потрібні навігаційні графи, фрагменти та дії для переходів між екранами.

- Використання автентифікаційного фрагменту

  Необхідно використати фрагмент, який буде відповідати за автентифікацію користувача. У цьому фрагменті можна виконати перевірку інформації для автентифікації, отримання токену та збереження його.

  Використовуючи токен, на цьому кроці необхідно зробити запит до сервера, і визначити роль користувача.

- Реалізація логіки переходу після автентифікації

  У ViewModel, яка відповідає за автентифікацію, треба додати код для переходу до відповідного інтерфейсу після успішної автентифікації і отримання інформації про роль від серверу. У даному ViewModel можна використати Navigation.findNavController() для отримання об’єкта NavController та переходу до певного напрямку в навігаційному графі.

- Використання навігаційного графа для переходу

  Далі необхідно відредагувати навігаційний граф, додавши необхідні напрямки та асоціювавши їх з фрагментами для різних інтерфейсів, які необхідно показати після автентифікації, залежно від ролі.

Ці кроки дозволять впровадити перехід до різних інтерфейсів після автентифікації використовуючи компонент навігації в Android MVVM, забезпечивши зрозумілість, простоту коду та не суперечитимуть наявних підходам у розробці застосунку.

5. ПРОПОЗИЦІЇ ДЛЯ ПОДАЛЬШОЇ РОЗРОБКИ

Розробка Android застосунку система управління університету має важливе значення для всіх учасників навчального процесу, тому що такий застосунок для кінцевого користувача спрощує роботу з певними сервісами, за рахунок мобільності, швидкого доступу. Далі буде запропоновати декілька ідеї, які можуть бути продовженням розробки цього застосунку.

5.1 Сповіщення по роботі з курсовими

Для швидкого інформування щодо змін у системі курсових робіт, можна впровадити систему сповіщень в розроблюваний Android додаток. Ця система буде сповіщати учасників навчального процесу при певних подіях, наприклад, таких як надходження заявки на роботу, виставлення оцінки і так далі.

5.2 Реалізація функціоналу для ролей працівника факультету та адміністратора системи

По аналогії до впроваджуваного функціоналу для ролей викладача та студента, можна додати ще роль працівника факультету та адміністратора системи, так як вони теж є учасниками процесу роботи з курсовими.

Для цього необхідно зробити ширший аналіз предметної області, відповідно до нього побудувати сценарії використання для цих ролей, створити необхідні data класи на різних рівнях застосунку, допоміжні класи та інтерфейси, необхідну xml розмітку, необхідні компоненти ViewModel з описаною бізнес логікою, а також внести зміни до компонента навігації та логіки переходів.

5.3 Впровадження функціоналу САЗ в застосунок «KMA Smart»

Продовжуючи розгляд розвитку застосунку, імплементація сервісів по роботі з вибором дисциплін, груп, перегляду індивідуальних планів та розкладу може значно покращити його функціональність та користувацький досвід. Однак,

перед реалізацією цих сервісів необхідно провести глибший аналіз предметної області та розуміння логіки всіх необхідних процесів в системі.

Одна з ключових проблем, з якою можна зіткнутися під час імплементації цих сервісів, полягає в розумінні різних сутностей та взаємозв'язків між ними. Наприклад, для роботи з вибором дисциплін необхідно мати доступ до списку доступних дисциплін, відповідних груп студентів, а також розкладу занять. Розуміння взаємозв'язків цих елементів та їх правильна імплементація є важливим кроком у розробці цих сервісів.

Крім того, під час розробки сервісів необхідно вирішувати різноманітні проблеми, зокрема актуальною проблемою буде зменшення навантаження на сервер. Запити до сервера для отримання даних можуть призводити до великої кількості запитів і витрати ресурсів. Для зменшення навантаження можна розглянути такі підходи, як кешування даних на клієнтській стороні, оптимізація запитів до сервера та використання асинхронних запитів.

Успішна імплементація цих сервісів може мати значний вплив на користувачів університету. Вони зможуть зручно вибирати дисципліни, переглядати свій індивідуальний план, отримувати розклад занять та здійснювати інші необхідні операції. Такі сервіси допоможуть поліпшити організацію та ефективність навчального процесу в університеті.

У процесі розробки варто враховувати різноманітні проблеми, які можуть виникнути, і шукати оптимальні рішення для їх вирішення. Також розуміння предметної області та співпраця з фахівцями з цієї галузі можуть допомогти уникнути помилок та забезпечити ефективний розвиток застосунку.

ВИСНОВКИ

В розрізі даної роботи було оглянуто та порівняно різні архітектурні підходи в розробці під операційну систему Android, також оглянуто детальніше MVVM підхід та бібліотеки й інструменти, які зручно використовувати при даному підході. В ході виконання роботи, було лаконічно визначено переваги всіх розглянутих підходів та інструментів, такі як перевикористання компонентів, спрощення процесу розробки, спрощення тестування, зменшення залежності між компонентами. Також для бібліотек наведено приклади використання та зазначено головні проблеми, які вони вирішують.

Дана робота може слугувати вступом в розробку клієнтських застосунків під ОС Android, з вимогами збереження зрозумілості коду та збереження легкої масштабованості при ймовірних подальших сценаріях розробки застосунку. Зроблено акцент на важливості певних моментів в розробці, безпосередньо не пов’язаних з написанням коду, наприклад на розумінні предметної області, схематичного опису бізнес логіки, а також на плануванні процесу розробки, на проблемних моментах, та пошуку елегантних рішень.

В цілому, розробка Android-додатків вимагає уваги до деталей і розуміння ширшого контексту, а не просто написання коду. Використання відповідних архітектурних підходів, бібліотек та інструментів може значно полегшити розробку та забезпечити якість та розширюваність додатку.

Крім того, в роботі було розглянуто можливості подальшого розвитку застосунку. Одним з таких напрямків розвитку може бути імплементація сервісів, пов’язаних з вибором дисциплін, груп, переглядом індивідуальних планів та розкладом. Реалізація цих сервісів потребуватиме глибшого аналізу предметної області та розуміння логіки всіх необхідних процесів у системі.

При розробці застосунку в цьому напрямку можуть виникнути різноманітні проблеми, наприклад, необхідність зменшення навантаження на сервер. Важливо проаналізувати та врахувати всі можливі складнощі, що можуть виникнути, і знайти ефективні рішення для їх вирішення.

Імплементація таких сервісів може мати значний вплив на спільноту університету. Наприклад, студенти зможуть легко обирати дисципліни, формувати свої індивідуальні навчальні плани та отримувати доступ до розкладу занять безпосередньо з мобільного пристрою Android. Це сприятиме покращенню процесу навчання та забезпеченню зручності для студентів.

Ураховуючи потенційні переваги і користь для спільноти університету, імплементація цих сервісів може бути значним кроком у поліпшенні функціональності та користувацького досвіду застосунку. Важливо продовжувати глибокий аналіз предметної області, шукати оптимальні рішення та працювати над подальшим розвитком застосунку, щоб забезпечити його успішне використання в університетській спільноті.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. retrofit - https://square.github.io/retrofit/
2. room - https://developer.android.com/training/data-storage/room
3. koin - https://insert-koin.io/docs/reference/koin-android/start
4. glide - https://bumptech.github.io/glide/