Київський національний торговельно-економічний університет
Кафедра програмної інженерії та кібербезпеки

ВИПУСКНИЙ КВАЛІФІКАЦІЙНИЙ ПРОЕКТ

на тему:

Розробка програмного забезпечення «Особистий кабінет студента КНТЕУ»

Студента 4 курсу, 7 групи,
Напрям підготовки 6. 050103
«Програмна інженерія» _____ Маркевич Богдан Степанович
підпис студента

Науковий керівник кандидат педагогічний наук, старший викладач _____ Жирова Тетяна Олександрівна
підпис керівника

Гарант освітньої програми
кандидат технічних наук,
доцент _____ Цензура Микола Олександрович
підпис керівника

КИЇВ – 2019

Зміст

ВСТУП 3
РОЗДІЛ 1. ОСНОВНІ ПОНЯТТЯ МОБІЛЬНОЇ РОЗРОБКИ 5
1.1. Основні відомості про операційну систему Android 5
1.2. Дослідження проблеми 7
1.3. Use Case 8
1.4. Технічне завдання 9
1.5. Висновки до Розділу 1 11
РОЗДІЛ 2. ПРОЕКТУВАННЯ ІНОФРМАЦІЙНОЇ СИСТЕМИ «ОСОБИСТИЙ КАБІНЕТ СТУДЕНТА КНТЕУ» 12
2.1. MVP 12
2.2. Retrofit 2 13
2.3. RxJava (RxKotlin), RxAndroid 14
2.4. Dagger2 16
2.5. Висновки до Розділу 2 18
РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ДОДАТКУ «ОСОБИСТИЙ КАБІНЕТ СТУДЕНТА КНТЕУ» 19
3.1. Екран логіну 19
3.2. Реєстрація 19
3.3. Головний екран 20
3.4. Екрани інформації 21
3.5. Моя група 22
3.6. Висновки до Розділу 3 23
ВИСНОВКИ ТА ПРОПОЗИЦІЇ 24
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 25
ДОДАТКИ 26

ВСТУП

Мобільні телефони давно перестали бути чимось незвичайним і чудово справляються зі своєю функцією – бути засобом комунікації між людьми. При цьому, недавно з'явилися, але вже міцно ввійшли в наше життя смартфони настільки функціональні, що важко сказати, чого вони не вміють: це і плеєр, і фотоапарат, і можливість використання Інтернет-ресурсів, та інше. По суті, всі смартфони стали невеликою копією комп'ютера, яку постійно можна мати при собі.

У наш час все більше і більше смартфонів, комунікаторів, планшетних ПК і інших видів пристроїв, зручних для використання як в повсякденному житті, так і в закордонних поїздках зокрема, випускаються на базі ОС Android. Які ж причини поширення даної операційної системи?

По-перше, Android підтримує велику кількість пристроїв різних виробників. По-друге, Android характеризується високою доступністю засобів розробки. Засоби розробки для платформи Android безкоштовні, в той час як розробка, наприклад, під iPhone (від компанії Apple) вимагає чималих початкових фінансових вкладень. Крім усього перерахованого вище, перевагою ОС Android є наявність безкоштовних бібліотек для роботи зі сторонніми ресурсами (Yandex MapKit, Google Map API, ін.), в той час як для Windows Phone Mobile такі бібліотеки не поширені.

Зазначені вище переваги обумовлюють масовість і широке поширення сучасних пристроїв на базі Android, оснащених різними функціями і додатками, що роблять повсякденне життя простішим.

Орієнтованість на мобільні додатки пояснюється тим, що студенти в переважній більшості є власниками і користувачами сучасних смартфонів, планшетних ПК і інших кишенькових пристроїв. Таким чином, розроблений проект допоможе студентам у будь-якому місці, та в будь-який момент часу отримати доступ до даних своєї успішності, заборгованостей в бібліотеці, навчального плану та списку групи.

Об'єкт дослідження: освітній процес КНТЕУ.

Предмет дослідження: створення додатку Android.

Мета роботи: дослідження алгоритмів, інструментів та засобів для створення додатку.

Задача дослідження: створення додатку з функцією перегляду та онайомлення студентом його успішності, заборгованостей та новин на базі мобільної операційної системи Android.

Методи і технології розробки: візуальне і об'єктно-орієнтоване програмування, порівняння, спостереження, побудова програмного продукту на основі проаналізованих данних, використання сучасних технологій та фреймворків, аналіз алгоритмів пошуку та сортування та структур даних, ознайомлення із принципома клієнт-серврноих додатків.

Результат роботи: створено програмне забезпечення «Особистий кабінет студента КНТЕУ».

РОЗДІЛ 1.
ОСНОВНІ ПОНЯТТЯ МОБІЛЬНОЇ РОЗРОБКИ

1.1. Основні відомості про операційну систему Android

Що таке операційна система Android?

Операційна система Android - це мобільна операційна система, розроблена Google, яка використовується в основному для сенсорних пристроїв, мобільних телефонів і планшетів. Його дизайн дозволяє користувачам маніпулювати мобільними пристроями інтуїтивно, за допомогою рухів пальцями, які відображають загальні рухи, такі як затискання, зчитування та натискання.

На додаток до мобільних пристроїв, Google використовує програмне забезпечення для Android в телевізорах, автомобілях і наручних годинниках, кожен з яких обладнано унікальним інтерфейсом користувача.

Розуміння операційної системи Android

Операційна система Android вперше була розроблена компанією Android, Inc., компанією з програмного забезпечення, розташованою в Силіконовій долині, перш ніж компанія Google придбала її в 2005 році. Але в будь-якому випадку, скоро після цього, Google оголосив про майбутнє розгортання свого першого комерційно доступного Android-пристрою в 2007 році, хоча цей продукт фактично потрапив на ринок у 2008 році. З тих пір, розробники програмного забезпечення та додатків були в змозі використовувати технології Android, щоб розробляти мобільні програми, які продаються через магазини програм, наприклад,

Google Play. І оскільки вона розроблена як продукт Google, користувачам Android надається можливість пов'язувати свої мобільні пристрої з іншими продуктами Google, такими як хмара, електронні платформи та відеопослуги.

Вихідний код Android випускається у форматі з відкритим вихідним кодом, щоб сприяти розширенню відкритих стандартів для мобільних пристроїв. Однак, незважаючи на те, що Android випущений як «відкритий», він все ще упакований з фірмовим програмним забезпеченням, проданим на пристроях пристроїв.

Поява Android створило нове суперництво між виробниками смартфонів, а Apple (AAPL) - головним конкурентом Google. Для деяких, це конкурентне динамічне відображення «війни» між Coca-Cola (KO) і Pepsi (PEP) протягом останніх 30 років, де не було чіткого переможця чи програвача. Але на початку 2017 року Android був найпопулярнішою операційною системою для використання на мобільних пристроях з більш ніж двома мільярдами активних користувачів.

Зростаюча популярність системи також призвела до ряду патентних позовів. Останній вийшов з Oracle (ORCL), який стверджує, що Google незаконно використовував Java API для розробки програмного забезпечення для Android.

Недоліки операційної системи Android

Хоча Android пропонує користувачам життєздатну альтернативу іншим мобільним операційним системам, деякі обмеження залишаються. На стороні розробника, кодування складного досвіду користувача та інтерфейсів є часто важким завданням, яке вимагає більшої залежності від Java, ніж Objective-C. Для користувачів програми на Android Market, як правило, мають нижчі стандарти, ніж у порівнянних магазинах програм. Іншими словами, програми мають більш низькі профілі безпеки і роблять

користувачів більш сприйнятливими до порушень даних. Між тим, відсутність у Android голосового асистента і його велика залежність від реклами можуть відбивати деяких користувачів коментарями і підказками при першому запуску апарату. Операційна система швидко реагує на натискання користувача і виробляє установку і скачування потрібних програм і файлів з швидкістю, яка не програє іншим сучасним мобільним ОС.

1.2. Дослідження проблеми

Мобільний додаток (англ. «Mobile app») — програмне забезпечення, призначене для роботи на смартфонах, планшетах та інших мобільних пристроях. Багато мобільних додатків встановлені на самому пристрої або можуть бути завантажені на нього з онлайн магазинів мобільних додатків, таких як App Store, Google Play, Windows Phone Store та інших, безкоштовно або за плату. Спочатку мобільні додатки використовувалися для швидкої перевірки електронної пошти, але їх високий попит призвів до розширення їх призначень і в інших областях, таких як ігри для мобільних телефонів, GPS, спілкування, перегляд відео та користування Інтернетом. Ринок мобільних додатків сьогодні дуже розвинений і неухильно зростає.

В цілому додатки можна розділити на програми для внутрішніх потреб компанії (додатки на пристроях співробітників або ж на пристроях компанії), і додатки для маркетингу, брендингу та збільшення продажів (додатки на пристроях клієнтів бізнесу).

В контексті даної роботи, додаток розроблявся як мобільна програма для внутрішніх потреб університету, а саме доступу студентів до даних їх успішності, заборгованостей в бібліотеці, навчального плану та списку групи.

Оскільки виникла проблема, що студент не завжди має доступ до WEB-версії особистого кабінету, а його мобільний «друг» завжди під рукою, було прийнято рішення розробити мобільний клієнт для зручності використання ресурсів університету.

1.3. Use Case

1. Характеристична інформація

Мета в контексті: Клієнт надсилає запит, очікує дані оцінок, книжок, навчального плану та списку групи.

Сфера застосування: Університет

Передумови: наявність Інтернету

2. Успішна стадія завершення: Користувач отримав доступ до оцінок, книжок, навчального плану та списку групи.
3. Невдале завершення стану:
   - під час процесу роботи трапився збій додатку, відбулося аварійне завершення.
   - під час запиту даних трапився збій Інтернету, клієнт не отримав дані оцінок, книжок, навчального плану та списку групи.
   - користувач ввів недостовірні дані для входу
   - користувачу відобразились невідповідні дані оцінок, книжок, навчального плану та списку групи

Основний актор: Користувач.

Тригер: надходить запит на успішний перегляд оцінок, книжок, навчального плану та списку групи.

4. Генеральний сценарій успіху
   - 1.Користувач запустив додаток.
   - 2.Користувач ввійшов в обліковий запис.

- 3.Користувач надсилає запит на отримання даних оцінок, книжок, навчального плану та списку групи.
- 5.Користувач переглянув дані оцінок, книжок, навчального плану та списку групи.
- 6.Користувач вийшов із облікового запису.

5. Sub-варіації

Користувач може використовувати:

- Телефон
- Планшет

6. Відповідна інформація

Пріоритет: ТОП

Цільова ефективність: 10 хвилин на перегляд даних.

Частота: 5 / день

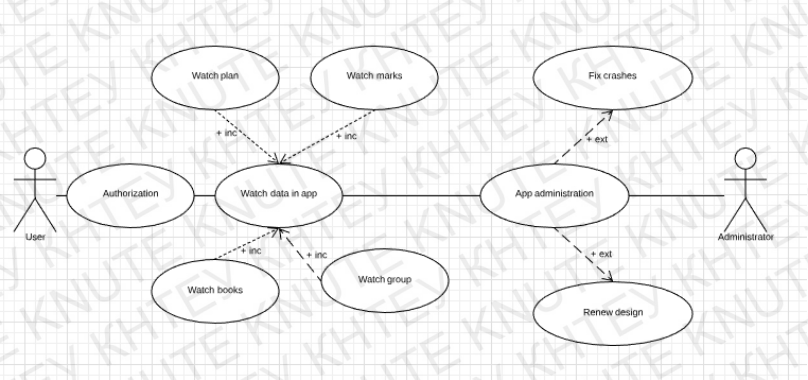

Рис. 1.1 Use Case Diagram

1.4. Технічне завдання

- Розділи додатку
- Оцінки
- Навчальний план
- Бібліотека
- Моя група

1. Типові екрани
    - Перший рівень – вхідний екран
    - Другий рівень (універсальний) - це екрани будь-якого рівня для виведення даних, результатів пошуку і т.д.
2. Дизайн частина проекту

Дизайн екранів виконується в суворій відповідності з наданими ескізами.

3. Програмна частина проекту

Клієнтські технології:

- Kotlin
- XML
- JSON
- Android SDK
- Google JVM (Dalvik)
- Java Core

4. Структура проекту
    - Login Screen
    - Registration Screen
    - Main Screen
        - ✓ Dialog “Menu”
    - Marks Screen
        - ✓ Dialog “Show more”
        - ✓ Dialog “Sort”Search
    - Plan Screen
        - ✓ Dialog “Show more”
        - ✓ Dialog “Sort”
        - ✓ Search

- Library Screen
  - ✓ Dialog “Sort”
  - ✓ Search
- My Group Screen
  - ✓ Dialog “Sort”
  - ✓ Search
- Options Screen
- Dialog “My Account”

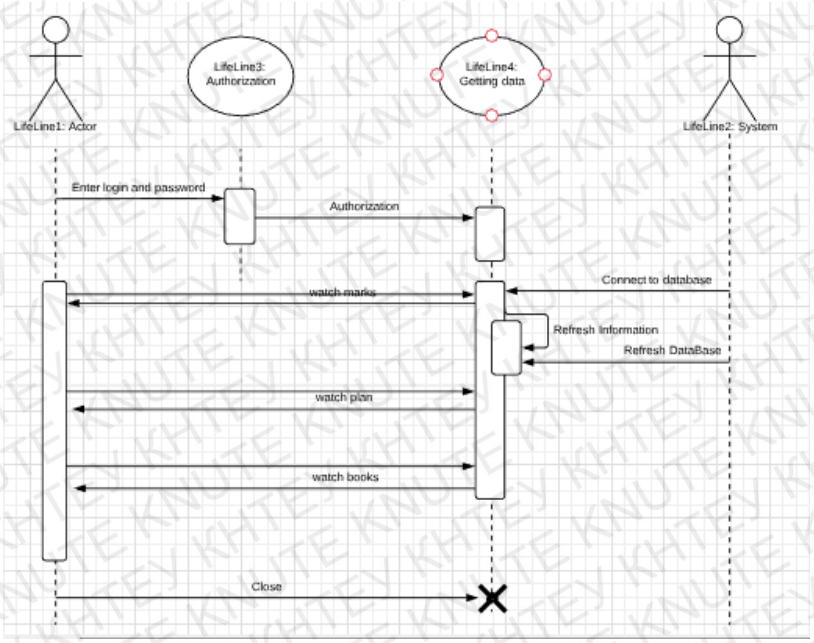

Рис 1.2 Sequence Diagram

1.5. Висновки до Розділу 1

Було досліджено ринок мобільних телефонів, проаналізовані дані за останні роки, з яких було прийнято вирішення вибрати ОС Android як операційну систему для розробки даного програмного рішення.

Розглянуто ОС Android зі сторони Користувача, Розробника та її архітектуру.

Також сформульовано проблему, розроблено Use Case та технічне завдання для подальшої роботи та розробки.

РОЗДІЛ 2.
ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ «ОСОБИСТИЙ КАБІНЕТ СТУДЕНТА КНТЕУ»

2.1. MVP

MVP – це спосіб поділу відповідальності в коді програми. Model надає дані для Presenter. View виконує дві функції: реагує на команди від користувача (або від елементів UI), передаючи ці події в Presenter і змінює gui на вимогу Presenter. Presenter виступає як сполучна ланка між View і Model. Presenter отримує події з View, обробляє їх (використовуючи або не використовуючи Model), і командує View про те, як вона повинна себе змінити.

У такого підходу до поділу відповідальності є ряд плюсів:

- Сильно спрощується написання тестів до коду
- Легко міняти якусь частину, не ламаючи при цьому іншу
- Код розбивається на дрібні шматочки, за рахунок чого він стає більш зрозумілим і читабельним

У той же час, звичайно, є і мінуси:

- Коду стає більше
- До цього підходу потрібно звикати
- На даний момент не сильно поширений (але відомий) підхід, тому доводиться всім розповідати про нього

MVP в Android. Activity в Android є God object. На ній зазвичай лежить така відповідальність:

- Повне управління GUI
- Обробка взаємодії з користувачем.

- Запуск асинхронних завдань.
- Обробка результату асинхронної завдання.

Найсумніше, God Object не безсмертний - Activity ще й вмирає при зміні конфігурації.

MVP знімає частину відповідальності з Activity. Вся робота з асинхронними завданнями йде в Presenter. Вся бізнес-логіка - в Presenter і Model. Activity, в свою чергу, стає View. Вона починає просто відображати те, що скаже Presenter і передає події в Presenter, щоб той вирішував, як бути далі.

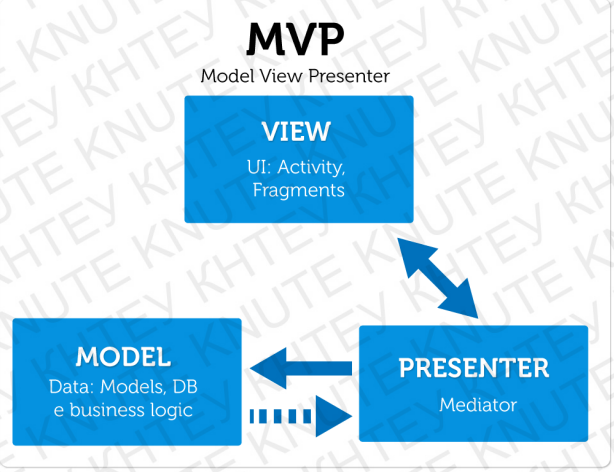

Рис. 2.2 Схема роботи MV

Retrofit 2

Retrofit (згідно з офіційним сайтом) - HTTP-клієнт для Android і Java. Він є незамінним інструментом для роботи з API в клієнт-серверних додатках.

Retrofit дозволяє зробити повноцінний REST-клієнт, який може виконувати POST, GET, PUT, DELETE. Для позначення типу та інших аспектів запиту використовуються анотації. Наприклад, для того, щоб позначити, що потрібно GET запит, нам потрібно написати перед

методом GET, для POST запиту POST, і так далі. В дужках до типу запиту ставиться цільової адресу.

2.2. RxJava (RxKotlin), RxAndroid

Рис. 2.4 RxJava + Android

Rx – це потужний інструмент, який дозволяє вирішити проблеми в елегантному декларативному стилі, який притаманний функціональному програмуванню, та який використовує парадигму реактивного програмування.

Реактивне програмування - програмування з асинхронними потоками даних. Ви можете створювати потоки даних не тільки з подій наведення або клікання мишкою. Потоком може бути що завгодно: змінні, користувацьке введення, властивості, кеш, структури даних і т.д.

Потік - це послідовність подій, упорядкована за часом. Він може викидати три типи даних: значення (певного типу), помилку або сигнал завершення. Сигнал завершення поширюється, коли поточне вікно або вікно, що містить кнопку, закривається. Ми перехоплюємо ці події асинхронно, вказуючи одну функцію, яка буде викликатися, коли викинуто значення, іншу для помилок і третю для обробки сигналу завершення. У деяких випадках можна опустити останні дві і сфокусуватися на оголошенні функції для перехоплення значень. Прослуховування потоку називається підпискою (subscribing). Функції, які ми оголошуємо, називаються спостерігачами (observer). Потік - це об'єкт наших спостережень (observable, спостережуваний об'єкт). Це в точності патерн проектування, так званий «Спостерігач».

Реактивний підхід підвищує рівень абстракції вашого коду і ви можете сконцентруватися на взаємозв'язку подій, які визначають бізнес-логіку, замість того, щоб постійно підтримувати код з великою кількістю деталей реалізації. Код в реактивному програмуванні, ймовірно, буде коротшим. У реальному додатку все працює асинхронно. У нас є мережа, в яку ми відправляємо запити і з якої через тривалий час отримуємо відповіді. Ми не можемо блокувати основний потік виконання, так що робота з мережею повинна виконуватися у фоновому потоці. Ми не можемо блокувати основний потік при роботі з файловою системою, базою даних, записи в сховище або навіть в shared preferences, так що доводиться виконувати ці операції в фонових потоках.

Користувачі - теж щось на кшталт асинхронних джерел даних. Ми даємо їм інформацію через UI, і вони реагують на неї натисканням кнопок і внесенням даних в поля.

Користувач може повертатися в додаток в різний час. І програма має бути готова до прийому даних, має бути реактивною, щоб не виникло стану, при якому блокується основний потік виконання; щоб не було ситуації, коли частина даних надходить асинхронно, а додаток цього не очікує і в результаті не враховує отримані дані або взагалі аварійно завершується. В цьому і полягає складність: ви повинні підтримувати всі ці стани в ваших Activity / Fragment. Потрібно примиритися з тим, що численні асинхронні джерела генерують і споживають дані, можливо, з різною швидкістю. І це ми ще не враховуємо роботу самої Android, яка є асинхронної платформою.

Поки ви не можете забезпечити синхронність всієї архітектури вашого застосування, наявність єдиного асинхронного ресурсу буде ламати традиційний імперативний стиль програмування. Замість того щоб намагатися координувати всі асинхронні елементи архітектури, ми

можемо зняти з себе цей обов'язок, з'єднавши їх безпосередньо. Можна безпосередньо підписати UI на базу даних, щоб він реагував на зміни даних. Можна так змінити базу даних та Інтернет-дзвінки, щоб вони реагували на натискання кнопки, а не намагатися отримати клік і потім його відправити.

В цілому такий підхід дозволяє нам не писати багато коду, необхідного для підтримки станів: замість того щоб управляти станами, ми просто з'єднуємо компоненти один з одним.

Рис. 2.5 Схема роботи реактивного програмування

2.3. Dagger2

Dagger 2 – це Android бібліотека, яка дозволяє реалізувати патерн впровадження залежностей (Dependency Injection), який в свою чергу є формою інверсії управління (IoC - Inversion of Control). Інверсія управління (англ. Inversion of Control, IoC) - важливий принцип об'єктно-орієнтованого програмування, що використовується для зменшення зачеплення в комп'ютерних програмах.

Інверсія управління буває не тільки в фреймворками, але і в деяких бібліотеках (але зазвичай бібліотеки не створюють інверсії управління - вони надають набір функцій, які повинен викликати програміст).

Однією з реалізацій інверсії управління в застосуванні до управління залежностями є впровадження залежностей (англ. Dependency injection).

Умовно, якщо об'єкту потрібно отримати доступ до певного сервісу, об'єкт бере на себе відповідальність за доступ до цього сервісу: він або

отримує пряме посилання на місцезнаходження сервісу, або звертається до відомого «сервіс-локатор» і запитує посилання на реалізацію певного типу сервісу. Використовуючи ж впровадження залежності, об'єкт просто надає властивість, яке в змозі зберігати посилання на потрібний тип сервісу.

Впровадження залежності більш гнучке, тому що стає легше створювати альтернативні реалізації даного типу сервісу, а потім вказувати, яка саме реалізація повинна бути використана в, наприклад, файлі конфігурації, без змін в об'єктах, які цей сервіс використовують. Це особливо корисно в юніт-тестування, тому що вставити реалізацію «заглушки» сервісу в тестований об'єкт дуже просто.

З іншого боку, зайве використання впровадження залежностей може зробити програми більш складними і важкими в супроводі: так як для розуміння поведінки програми програмісту необхідно дивитися не тільки в вихідний код, а ще й в конфігурацію, а конфігурація, як правило, невидима для IDE, які підтримують аналіз посилань і рефакторинг, якщо явно не зазначена підтримка фреймворків з впровадженнями залежностей. Dagger 2 аналізує ці залежності для вас і генерує код, який допомагає з'єднати їх.

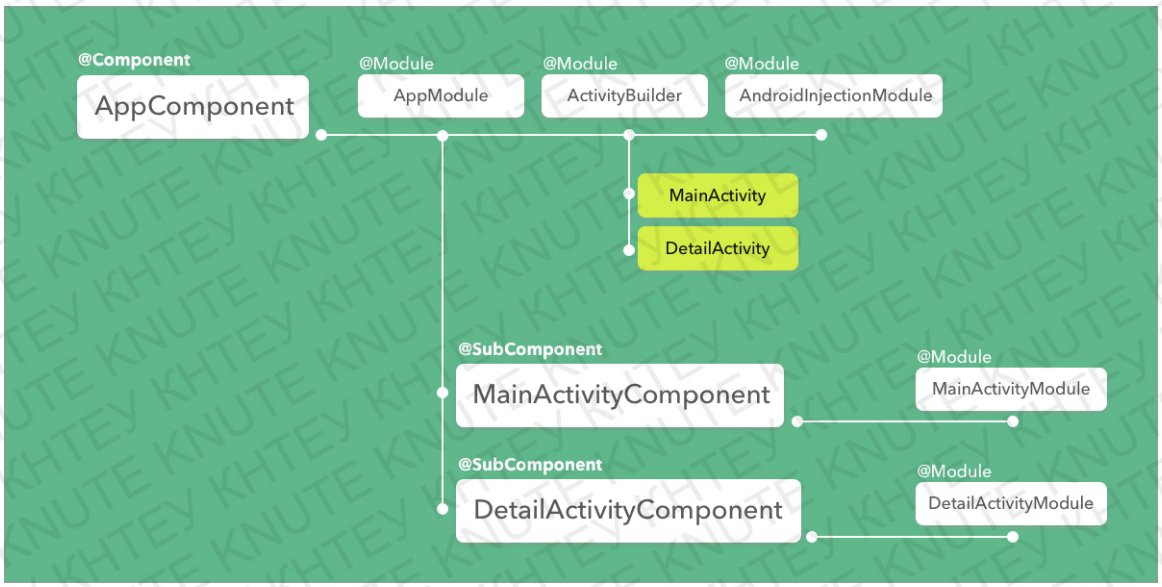

Рис. 2.6 Схема-приклад роботи із Dagger2

2.4. Висновки до Розділу 2

Було проаналізовано та описано основні характеристики технологій та інструментів для розробки програмного рішення.

Отже, для розробки мобільного додатку будуть використані архітектурне рішення MVP (Model – View - Presenter), технології реактивного програмування Rx, а саме RxJava(RxKotlin), RxAndroid, впровадження залежностей (Dependency Injection) за допомогою бібліотеки Dagger2, та бібліотека Retrofit2 для комунікації із сервером за допомогою REST API.

РОЗДІЛ 3.
ПРОГРАМНА РЕАЛІЗАЦІЯ ДОДАТКУ «ОСОБИСТИЙ КАБІНЕТ СТУДЕНТА КНТЕУ»

3.1. Екран логіну

Логін – це головний екран, який користувач першим бачить при запуску додатка.

На ньому розташовані емблема університету, елементи полів вводу, кнопка входу та посилання на екран реєстрації.

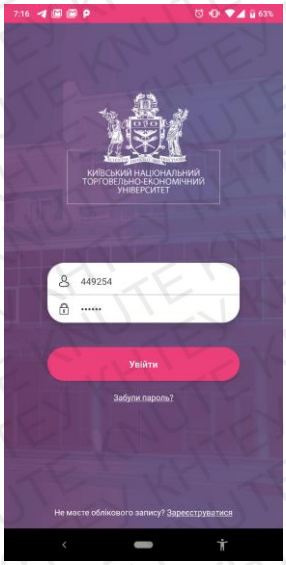

Рис 3.1 Логін

3.2. Реєстрація

Через екран реєстрації користувач реєструється в додатку і далі має змогу ним користуватись.

Він складається з трьох частин:

1. Введення email
2. Введення паролю
3. Введення особистих даних

Рис 3.2 Реєстрація

3.3. Головний екран

Головний екран – це екран, на якому знаходяться елементи навігації та меню користувача.

З елементів навігації можна перейти на екрани з інформацією тої чи іншої категорії.

З меню користувача можна перейти на:

1. Налаштування
2. Інформацію про аккаунт
3. Закінчення сесії.

Рис 3.3 Головний екран та меню користувача

3.4. Екрани інформації

В додатку є чотири екрани з інформацією різних категорій. Всі вони складаються зі списку даних, які можна сортування за допомогою меню сортування.

Кожний елемент має кнопку «Додатково», при натисканні якої з'являється діалогове вікно з інформацію про конкретний об'єкт.

Присутній функціонал пошуку по даних, який доступний по натисканню на кнопку «Пошук».

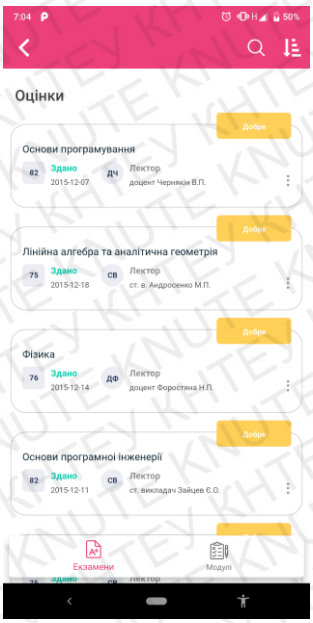

Рис 3.4 Один із екранів інформації

3.5. Моя група

На екрані «Моя група» є список одногрупників користувача, який успішно залогінився. На даному екрані студент має змогу переглянути дані своїх одногрупників та, за допомогою кнопки «Написати», відкрити вікно чату.

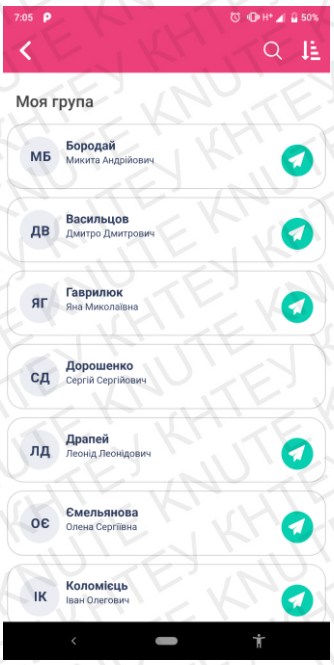

Рис 3.5 Моя група

3.6. Висновки до Розділу 3

Розроблений додаток побудований на основі всі схем та діаграм, відповідно до технічного завдання, використовуючи новітні технології, ресурси та інструменти.

Був ретельно протестований як в реальному середовищі, так і за допомогою Unit-тестів. Додаток готовий до реалізації через Google Play Store.

В майбутньому планується доробити функціонал месенджингу, щоб студенти могли вести особисте листування через додаток.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

Під час створення програмної системи у дипломній роботі було пройдено усі етапи, а саме:

1. На етапі аналізу предметної області було досліджено предметну область – рейтингову систему КНТЕУ. Проведено огляд та аналіз кількох веб-систем аналогів, що реалізують схожі функції предметної області.

2. На етапі проектування програмної системи розроблено її архітектуру та представлено у вигляді класів програми. Вивчено інтерфейс серверної частини та способи інтеграції.

3. На етапі програмної реалізації, з використанням об’єктно-орієнтованого підходу в інтегрованому середовищі розробки програмного забезпечення Android Studio, реалізовано функції системи, які написані на мові програмування Kotlin з імплементацією запитів до серверу. Розроблено інтерфейс програми.

4. На етапі тестування було проведено функціональне та тестування безпеки.

Також розроблено інструкцію користувача, де вказується правила використання програми. Програмна система дає можливість швидкого та зручного доступу до рейтингової системи КНТЕУ студентами. Після затвердження її керівництвом університету, дану програму буде введено в подальшу експлуатацію.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

Основний:

1. Programming Android / Zigurd Mednieks, G. Blake Meike, Лейрд Дорнін, Masumi Nakamura. – O`Reilly, 2011. - 736 с.
2. Head First Android Development / Thomas Asbridge, Anthony J.F. Griffiths. – O`Reilly, 2015. – 738 с.

Інтернет ресурси:

1. Android [Електронний ресурс]. – Режим доступу: URL: uk.wikipedia.org/wiki/Android.
2. История Android - от стартапа до лидера рынка [Електронний ресурс]. – Режим доступу: URL: magicmobile.com.ua/novosti/istorija_android__ot_startapa_do_lidera_rynka.
3. Все про андроїд [Електронний ресурс]. – Режим доступу: URL: ka4an.at.ua/publ/obzori/vse_pro_androjid/2-1-0-7.
4. Про ОС Андроїд [Електронний ресурс]. – Режим доступу: URL: www.proandroid.com.ua/pro-os-androjid.html.
5. Перспективи розробки під Android (EN). [Електронний ресурс]. – Режим доступу:URL:http://armikael.com/mobile/prospects_of_development_on_android.html

ДОДАТОК А

Екран реєстрації

```
package com.office.knute.screen.get_started

import android.content.Context
import android.content.Intent
import android.os.Bundle
import android.support.v4.app.Fragment
import com.arellomobile.mvp.presenter.InjectPresenter
import com.arellomobile.mvp.presenter.ProvidePresenter
import com.office.knute.App
import com.office.knute.R
import com.office.knute.base.BaseActivity
import com.office.knute.base.BaseFragment
import com.office.knute.screen.get_started.adapter.RegistrPagerAdapter
import com.office.knute.screen.get_started.email.EmailFragment
import com.office.knute.screen.get_started.info.InfoFragment
import com.office.knute.screen.get_started.pass.PassFragment
import com.office.knute.utils.KeyBoardUtils
import kotlinx.android.synthetic.main.activity_get_started.*
import javax.inject.Inject

class GetStartedActivity : BaseActivity(), GetStartedView {

    companion object {
        fun start(context: Context) {
            context.startActivity(Intent(context, GetStartedActivity::class.java))
        }
    }

    @Inject
    @InjectPresenter
    lateinit var registrPresenter: GetStartedPresenter

    private var counter = 1
    private var adapter: RegistrPagerAdapter? = null

    @ProvidePresenter
    fun getPresenter(): GetStartedPresenter {
```

```
        val getStartedComponent =
DaggerGetStartedComponent.builder().appComponent((application as
App).appComponent).build()
        getStartedComponent.inject(this)
        return registrPresenter
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_get_started)
        val fragments = listOf(EmailFragment(), PassFragment(), InfoFragment())
        adapter = RegistrPagerAdapter(supportFragmentManager, fragments)
        vp_container.adapter = adapter
        circleIndicator.setViewPager(vp_container)
        btn_continue.setOnClickListener {
            registrPresenter.checkFragment(adapter?.getItem(counter - 1)!!)
        }
        tv_login.setOnClickListener { onBackPressed() }
        KeyBoardUtils.hideKeyboardAndClearFocus(this, parent_container,
request_focus)
    }

    fun setNextFragment() {
        if (counter > 3)
            return
        vp_container.currentItem = counter
        counter++
        btn_continue.isEnabled = false
    }

    override fun checkFragment(item: Fragment) {
        when (item) {
            is EmailFragment -> item.checkUserEmail()
            is PassFragment -> item.setPass()
            is InfoFragment -> item.createUser()
        }
    }
```

ДОДАТОК Б

Екран «Моя група»

```
package com.office.knute.screen.my_group

import android.content.Context
import android.content.Intent
import android.content.pm.PackageManager
import android.net.Uri
import android.os.Bundle
import android.support.v4.app.ActivityCompat
import android.support.v4.content.ContextCompat
import android.support.v7.view.ActionMode
import android.support.v7.widget.GridLayoutManager
import android.support.v7.widget.SearchView
import android.view.Menu
import android.view.MenuItem
import android.view.View
import com.arellomobile.mvp.presenter.InjectPresenter
import com.arellomobile.mvp.presenter.ProvidePresenter
import com.office.knute.App
import com.office.knute.R
import com.office.knute.base.BaseActivity
import com.office.knute.dialog.NAME_REVERSE_TAG
import com.office.knute.dialog.NAME_TAG
import com.office.knute.dialog.SortDialog
import com.office.knute.model.data.GroupItem
import com.office.knute.view.GROUP
import com.office.knute.view.ShimmerAdapter
import kotlinx.android.synthetic.main.activity_group.*
import javax.inject.Inject

class GroupActivity : BaseActivity(), GroupView {

    companion object {

        fun start(context: Context) {
            context.startActivity(Intent(context, GroupActivity::class.java))
        }
```

```
    }

    private val sortListener = object : SortDialog.SortClickListener {
        override fun sort(TAG: String) {
            when (TAG) {
                NAME_TAG -> groupPresenter.sortByName()
                NAME_REVERSE_TAG -> groupPresenter.sortByNameReverse()
            }
        }

        override fun hide() {
            groupPresenter.hideSortDialog()
        }

    }

    @Inject
    @InjectPresenter
    lateinit var groupPresenter: GroupPresenter

    private var actionMode: ActionMode? = null
    private var searchView: SearchView? = null

    @ProvidePresenter
    fun getPresenter(): GroupPresenter {
        val loginComponent =
DaggerGroupComponent.builder().appComponent((application as
App).appComponent).build()
        loginComponent.inject(this)
        return groupPresenter
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_group)
        toolbar.title = getString(R.string.simple_group)
        setSupportActionBar(toolbar)
        toolbar.setNavigationOnClickListener { onBackPressed() }
        if (resources.getBoolean(R.bool.is_land)) {
            rv_group.layoutManager = GridLayoutManager(this, 3)
```

```
            rv_shimmer.layoutManager = GridLayoutManager(this, 3)
        } else {
            rv_group.layoutManager = GridLayoutManager(this, 2)
            rv_shimmer.layoutManager = GridLayoutManager(this, 2)
        }
        rv_shimmer.adapter = ShimmerAdapter(GROUP)
        groupPresenter.getGroup(savedInstanceState)
    }

    override fun onCreateOptionsMenu(menu: Menu?): Boolean {
        menuInflater.inflate(R.menu.options_menu, menu)
        return true
    }

    override fun onOptionsItemSelected(item: MenuItem?): Boolean {
        when (item?.itemId) {
            R.id.nav_search -> groupPresenter.startActionMode()
            R.id.nav_sort -> groupPresenter.showSortDialog()
        }
        return false
    }

    override fun startSearchActionMode() {
        actionMode = startSupportActionMode(actionModeCallback)!!
        if (searchView != null && groupPresenter.lastQuery != null) {
            searchView!!.setQuery(groupPresenter.lastQuery, true)
        }
        toolbar.visibility = View.INVISIBLE
    }

    private val actionModeCallback = object : ActionMode.Callback {
        override fun onActionItemClicked(p0: ActionMode?, p1: MenuItem?):
Boolean {
            return true
        }

        override fun onCreateActionMode(p0: ActionMode?, p1: Menu?):
Boolean {
            searchView = createSearch(baseContext)
            p0?.customView = searchView.also { it?.onActionViewExpanded() }
```

```
            searchView?.setOnQueryTextListener(object :
SearchView.OnQueryTextListener {
                override fun onQueryTextSubmit(query: String): Boolean {
                    return false
                }

                override fun onQueryTextChange(newText: String): Boolean {
                    groupPresenter.search(newText)
                    return true
                }
            })
            return true
        }

        override fun onPrepareActionMode(p0: ActionMode?, p1: Menu?):
Boolean {
            return true
        }

        override fun onDestroyActionMode(p0: ActionMode?) {
            groupPresenter.destroyActionMode()
        }

    }

    override fun destroySearchActionMode() {
        actionMode = null
        searchView = null
        toolbar.visibility = View.VISIBLE
    }

    override fun populateGroup(group: List<GroupItem>) {
        if (group.isEmpty()) {
            groupPresenter.showEmptySearch()
        } else {
            groupPresenter.hideEmptySearch()
        }
        val adapter = GroupAdapter { startToCall
```