Державний вищий навчальний заклад
“Прикарпатський національний університет імені Василя Стефаника”
Кафедра інформаційних технологій

УДК 004

ДИПЛОМНИЙ ПРОЕКТ

Тема: Розробка програмного забезпечення для моніторингу медіа-продукції

Спеціальність: 121 Інженерія програмного забезпечення

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДП.ІПЗ-30.2.ПЗ

Рецензент	Студент
ст.викл. Савка І.Я.	ІПЗ-41 Шевлюк Л.В.
(посада) (підпис) (дата) (розшифровка підпису)	(шифр групи) (підпис) (дата) (розшифровка підпису)
Нормоконтролер	Керівник дипломного проекту
ст.викл. Савка І.Я.	проф. Кузь М.В.
(посада) (підпис) (дата) (розшифровка підпису)	(посада) (підпис) (дата) (розшифровка підпису)

Допускається до захисту

Завідувач кафедри

доц. Козленко М.І.
(посада) (підпис) (дата) (розшифровка підпису)

2020

Державний вищий навчальний заклад
«Прикарпатський національний університет імені Василя Стефаника»
Факультет математики та інформатики Кафедра інформаційних технологій
Спеціальність 121 Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Завідувач кафедри Козленко М.І.
_____
„_____”_____ 20___р.

ЗАВДАННЯ
НА ВИКОНАННЯ ДИПЛОМНОГО ПРОЕКТУ

Студенту Шевлюк Людмилі Вікторівні
(прізвище, ім’я, по батькові студента)

1. Тема проекту Розробка програмного забезпечення для моніторингу медіа-продукції
затверджена розпорядженням по факультету математики та інформатики від 25 жовтня 2019 р., №7
2. Термін здачі студентом закінченого проекту 22 травня 2020 р.
3. Вихідні дані до дипломного проекту категорії медіа-продукції, статистичні дані використання мобільних пристроїв та часу на перегляд медіа продуктів, технології розробки – Android, Java, Firebase.

4. Зміст пояснювальної записки (перелік питань, що їх належить опрацювати)
1. Аналіз області розробки та постановка задачі
2. Огляд технологій та проектування продукту
3. Розробка демонстраційного програмного забезпечення
4. Бізнес-план розробки додатку «Mark It»

5. Перелік графічного матеріалу (з точним забезпеченням обов’язкових креслень) 1.Титульний лист; 2. Мета та новизна проекту; 3. Огляд аналогів; 4. Ключовий функціонал; 5. Діаграма прецедентів; 6. Створення дизайну; 7. Модель «сутність-зв’язок»; 8. Технології розробки; 9. Архітектура проекту; 10. Демонстрація роботи додатку; 11. Бізнес-план; 12. Висновки.

6. Дата видачі завдання 11.09.2019

Керівник _____ Кузь М.В.
(підпис) (розшифровка підпису)

Завдання прийняв до виконання _____ Шевлюк Л.В.
(підпис) (розшифровка підпису)

КАЛЕНДАРНИЙ ПЛАН

Номер і назва етапів дипломного проекту	Термін виконання етапів проекту	Примітка
1. Аналіз області розробки та постановка задачі	02.12.2019	
2. Огляд технологій та проектування продукту	15.02.2020	
3. Розробка демонстраційного програмного забезпечення	17.04.2020	
4. Бізнес-план розробки додатку «Mark It»	11.05.2020	
5. Оформлення пояснювальної записки	18.05.2020	

Студент _____ Шевлюк Л.В.
(підпис) (розшифровка підпису)

Керівник проекту _____ Кузь М.В.
(підпис) (розшифровка підпису)

РЕФЕРАТ

Пояснювальна записка: 74 сторінки (без додатків), 69 рисунків, 27 джерел, 1 додаток на 15 сторінках.

Ключові слова: МЕДІА ПРОДУКЦІЯ, МОБІЛЬНИЙ ДОДАТОК, OS ANDROID, JAVA, DAGGER, RETROFIT, FIREBASE, RXJAVA, BUTTERKNIFE.

Об`єктом дослідження є мобільний додаток на базі OS Android для моніторингу медіа продукції.

Мета роботи: спроектувати та розробити програмний продукт, що відслідковуватиме прогрес користувача у перегляді медіа продуктів та проходженні навчальних курсів.

Стислий опис тексту пояснювальної записки:

Під час виконання дипломного проекту проведено огляд та аналіз існуючих програмних продуктів, виділено їх основні переваги та недоліки. Розроблено дизайн, визначено основні функції та можливості користувача, архітектуру додатку та логіку обробки дій. Реалізовано клієнт-серверне програмне забезпечення та здійснено економічне обгрунтування розробки.

ABSTRACT

Explanatory note: 74 pages (without appendix), 69 figures, 27 references, 1 appendix on 15 pages.

Key words: MEDIA PRODUCTS, MOBILE APPLICATION, OS ANDROID, JAVA, DAGGER, RETROFIT, FIREBASE, RXJAVA, BUTTERKNIFE.

Object of study: a mobile application for media products monitoring based on Android OS.

Brief description of the text of the explanatory note:

During the implementation of the diploma project, a review and analysis of existing software products was done along with highlighting their main advantages and disadvantages. The design is created, the main functions and possibilities of the user, application architecture and logic of action processing are defined. The client-server software is implemented and the economic substantiation of development is carried out.

ЗМІСТ

ВСТУП

Із розвитком сучасних технологій люди все більше часу проводять онлайн. Проте це не означає, що вони витрачають його даремно. Однією з переваг модернізованого світу є наявність доступу до здобутків світової медіа культури, що дозволяє у будь-який час переглядати улюблені чи нові для себе серіали та фільми. Ще одним плюсом є можливість здобувати необхідні чи бажані знання, не змінюючи при цьому власної локації та у зручний для цього час.

Зі збільшенням можливостей ускладнюється також і завдання відслідковування прогресу. Таким чином, переглядаючи одночасно декілька серіалів чи мультфільмів та проходячи кілька курсів, можна легко забути, на якій серії зупинилися чи який урок переглянули. Саме тому зручно одразу відмітити свій прогрес та не витрачати час на знаходження необхідного відео. У такому випадку стає актуальною розробка мобільного додатку «Mark It», що дозволить користувачеві одразу бачити усі свої активності, з легкістю знаходити, на чому він зупинився, рухатися далі, а також аналізувати статистику витраченого часу на дозвілля та набуття нових чи закріплення існуючих знань.

Предмет розробки – багатокористувацький додаток на базі OS Android для відслідковування прогресу перегляду медіа продуктів.

Мета роботи – проектування, дизайн та розробка програмного забезпечення на базі OS Android для моніторингу медіа продукції.

Для досягнення мети поставлено наступні завдання:

- провести аналіз досліджуваної області;
- здійснити огляд аналогів програмного забезпечення;
- визначити особливості функціоналу та вимоги до додатку;
- спроектувати базу даних;
- створити зручний дизайн у вигляді мокапів;
- розробити архітектуру та логіку роботи додатку;
- реалізувати програмне забезпечення;

– здійснити тестування ПЗ на коректність роботи та відповідність вимогам.

Результат роботи – програмне забезпечення для OS Android, що дозволяє користувачам зберігати інформацію щодо прогресу перегляду тої чи іншої медіа продукції.

1 АНАЛІЗ ОБЛАСТІ РОЗРОБКИ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Дослідження області медіа продукції

Модернізоване суспільство прагне використовувати мобільні пристрої, які частково роблять життя простішим та комфортнішим. Саме тому виробники сучасних смартфонів поєднали в них функції одночасно декількох пристроїв. Вони замінюють не лише стаціонарні телефони, а й mp3-плеєри, цифрові фотоапарати, навігатори і навіть частково перебирають функції ПК. А ще декілька років тому всі ці пристрої були надзвичайно популярними.

Існує достатнє різноманіття мобільних пристроїв з безліччю функцій, характеристик та розмаїттям цінової політики. Відповідно є і достатня кількість операційних систем, на базі яких смартфони функціонують, такі як: Android, iOS, Fire OS, Flyme OS, Kai OS, Lineage OS, Sailfish OS. За даними IDC (International Data Corporation) частка Android пристроїв на ринку у 2019 р. становила 87%, що говорить про актуальність розробки мобільного додатку саме на базі OS Android [1].

Більшість людей використовують мобільні пристрої для спілкування в соціальних мережах, виходу в інтернет, перевірки електронної пошти, прослуховування музики, перегляду фото та відео, управління зустрічами та нотуванням, перегляду новин та майже в останню чергу для дзвінків [2]. Таким чином, суспільство все більше переходить на використання усіх доступних онлайн ресурсів.

Одним з найпопулярніших занять в мережі є перегляд різноманітної медіа продукції, а саме фільмів, серіалів та мультсеріалів. Наразі існує безліч ресурсів, що дозволяють безкоштовно переглядати здобутки вітчизняної та світової медіа сфери, маючи лише хороше інтернет-з’єднання та смартфон, планшет або ПК. Лише в Україні середньо статистично людина витрачає більше 6-ти годин на тиждень на перегляд фільмів, серіалів та розважальних програм [3]. Відповідно

не завжди можна запам’ятати, на якій серії зупинилися, особливо якщо їх є більше 100 чи одночасно переглядаються декілька. У такому випадку можна записати у нотатки номер серії та назву медіа продукту [4]. Проте це не завжди зручно та доцільно. Виходячи з цього, даний процес можна спростити та організувати, створивши відповідний додаток, що завжди знаходиться разом із людиною.

Ще одним популярним способом проведення часу в мережі є онлайн освіта [5]. Спочатку до неї ставилися дещо зі зневагою та обережністю, але зараз навчання в інтернеті є майже невід’ємною складовою життя кожного. Існує достатня кількість платформ, де можна знайти щось цікаве саме для себе. Вони містять чимало курсів з різних спеціалізацій та для різних рівнів знань, від початкового з відсутністю знань з даної теми до прогресивного, де глибоко розглядаються усі деталі. Прикладами таких платформ є Prometheus, Coursera, Udemy, edX та інші. Частково існує думка, що найближчим часом різноманітними вебінарами буде замінено частину аудиторних занять. Мало не всі провідні навчальні заклади світу розробили свої онлайн-курси та надали до них масовий доступ. Тепер кожен прямо з дому може навчатися у Прінстоні, Стенфорді чи Кембриджі на безлічі спеціальностей. Онлайн-освіта має чимало переваг, а саме:

- доступність;
- економія коштів;
- зручність;
- мобільність;
- економія часу;
- чітка спрямованість навчання;
- навчання самодисципліни та організованості;
- можливість вибору способу подачі матеріалу.

У даному випадку також є можливість проходження декількох курсів одночасно. Відповідно до цього, прогрес кожного з них також потрібно

контролювати та запам’ятовувати номер або ж тему останнього пройденого уроку [4]. Спростити цей процес можна за допомогою додатку, який зберігатиме дані про усі курси та медіа продукти у одному місці.

Ще одним значним плюсом буде можливість переглядати щоденно, щотижнево та помісячно витрачений час на навчання та дозвілля та проводити аналіз розподілу сил на основі цих даних.

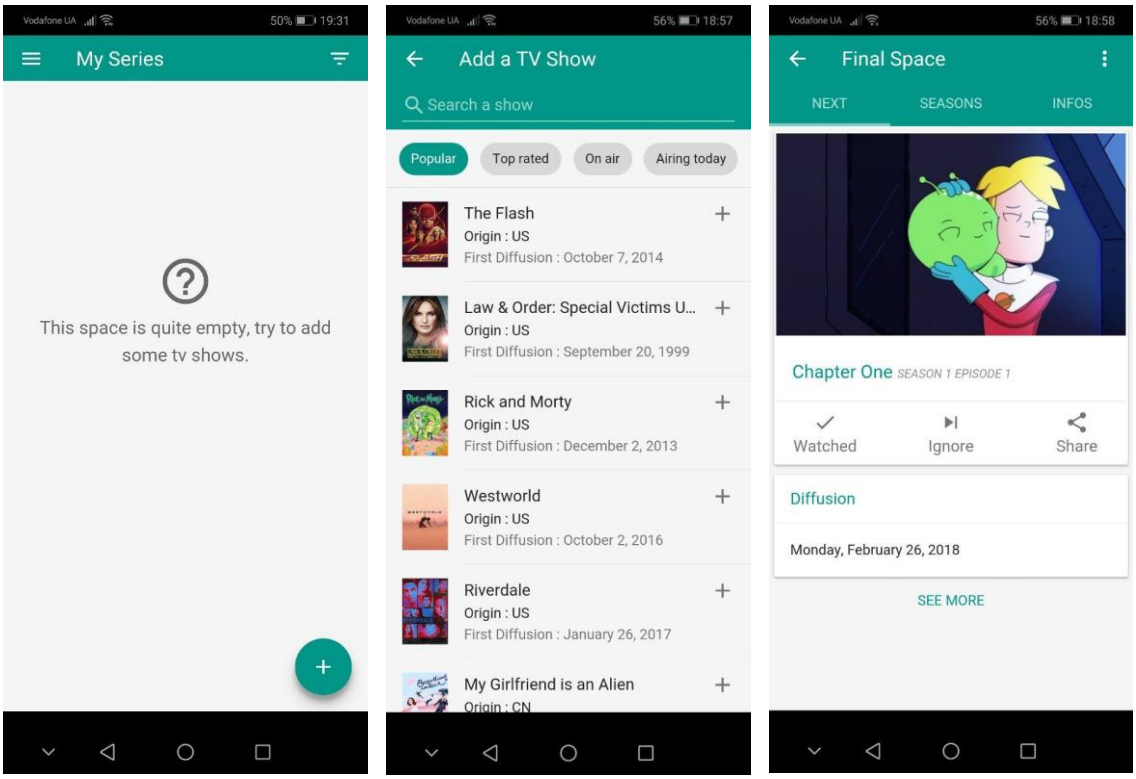
1.2 Огляд існуючих аналогів продукту

Перед початком будь-якої розробки необхідно не лише проаналізувати ринок на потребу такої розробки, а і проглянути існуючі системи, перетворивши їхні недоліки у власні переваги та визначивши унікальність проекту, що реалізовуватиметься.

У межах предметної області даної роботи до уваги беруться три додатки, що допомагають користувачам відслідковувати їх прогрес перегляду різноманітних медіа продуктів, а саме серіалів та мультсеріалів. Цими продуктами є SeriesTracker, SeriesGuide та myShows, що розміщені у Google Play Store. Розглянемо кожен з них детальніше.

На рисунку 1.1 зображено вигляд та функціонал додатку SeriesTracker. При першому відкритті програми одразу доступний увесь функціонал і відсутня реєстрація. Це означає, що увесь прогрес зберігається локально і при видаленні програми чи даних з певних технічних неполадок усі перегляди зникнуть та не підлягатимуть відновленню. При вході у додаток одразу пропонується додати перший серіал (рис. 1.1, а) за допомогою пошуку (рис. 1.1, б). Тобто у програмі міститься база доступних продуктів та їх опис. Для відзначення прогресу необхідно перейти на екран серіалу та з нього до останнього переглянутого епізоду, де і можна позначити, що його було переглянуто або ж пропустити (рис. 1.1, в). Також до кожної серії є фото моменту з даного епізоду. На жаль, неможливо додавати навчальні курси або ж продукти, яких у базі немає. Поточна

версія – 1.2.0, версія Android – 4.0 і вище, розмір програми без даних – 72,23 Мб, оцінка – 3,4.

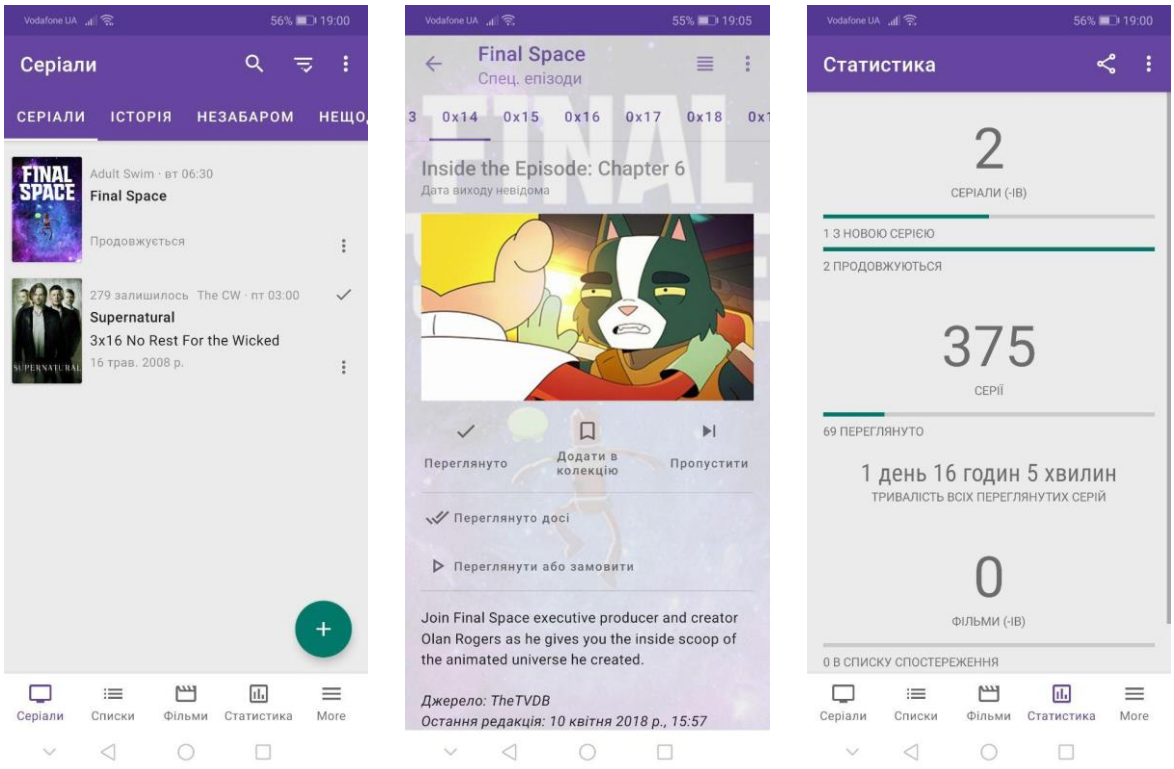


а) б) в)

Рисунок 1.1 – Приклад роботи програми SeriesTracker,
а – перший вхід у програму; б – пошук медіа продукції; в – екран епізоду

На рисунку 1.2 можна побачити декілька екранів додатку SeriesGuide. Для доступу до функціоналу реєстрація не потрібна. Проте є платна можливість створити власний обліковий запис для збереження даних не лише локально, а й на сервері. Додані серіали відображаються на основному екрані із зазначенням останнього переглянутого епізоду (рис. 1.2, а). Відзначення переглянутої серії здійснюється аналогічно до SeriesTracker (рис. 1.2, б), проте є ще можливість позначити переглянутими усі серії, до цієї включно. Також є статистика переглядів, де показано кількість переглянутих серіалів, серій та тривалість всіх переглянутих епізодів (рис. 1.2, в) і можна створювати списки серіалів та

фільмів. Можливість додати навчальний курс відсутня. Поточна версія – 54, версія Android – 5.0 і вище, розмір програми без даних – 16,41 Мб, оцінка – 4,3.

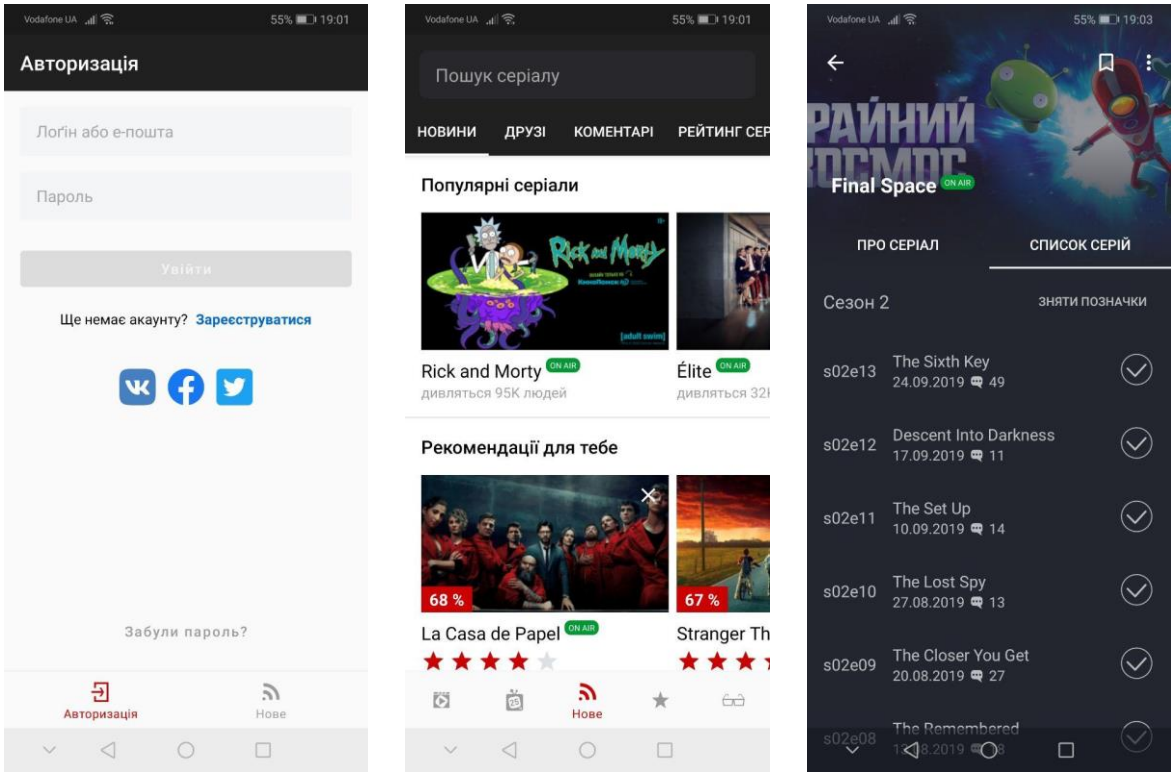


а) б) в)

Рисунок 1.2 – Приклад роботи програми SeriesGuide,
а – головний екран; б – екран епізоду; в – екран статистики

На рисунку 1.3 показано функціонал та вигляд додатку myShows. Для відкриття повного функціоналу потрібно зареєструватися (можна через соціальні мережі, а саме Facebook, Twitter або VK) (рис. 1.3, а). Далі додаток дозволяє перейти до основного екрану (рис. 1.3, б), де відображаються популярні та рекомендовані серіали, а також сортування за рейтингом. Можна знаходити друзів, а також коментувати епізоди. Щоб відмітити переглянуті серії необхідно перейти до серіалу, обрати список серій та галочками відмічати переглянуті серії або ж одразу весь сезон (рис. 1.3, в). У профілі можна переглянути статистику по кількості переглянутих серій, витрачених годин та днів. Також у додатку є блог з останніми новинами про медіа продукти та можливість створити списки

улюблених серіалів та епізодів. Додати навчальний курс неможливо. Поточна версія – 1.3.4, версія Android – 5.0 і вище, розмір програми без даних – 19,26 Мб, оцінка – 4,5.



а) б) в)

Рисунок 1.3 – Приклад роботи програми myShows,
а – екран входу у додаток; б – основний кран; в – екран епізоду

Розглянувши аналоги та проаналізувавши їх сильні та слабкі сторони, можна переходити до постановки задачі на розробку власного додатку.

1.3 Постановка задачі на розробку

Вимоги до програмного продукту, що розроблятиметься в межах даної роботи, звучать наступним чином:

1.3.1 Призначення

«Mark It» є мобільним додатком на базі OS Android для моніторингу медіа продукції, версії 1.0.

1.3.2 Прийняті угоди

Для використання додатку необхідно завантажити на мобільний пристрій з операційною системою Android apk-файл та встановити його. Запуск додатку відбувається з меню або ж пошуку, натисненням на іконку аплікації. Користуватися функціоналом додатку можуть лише зареєстровані користувачі. Реєстрація здійснюється за допомогою облікового запису Google або ж створення аккаунту за допомогою адреси актуальної поштової скриньки. Усі відступи, розміри та розміщення елементів у додатку відповідають розробленому дизайну. Кольори, що використовуються: #091731, #000000, #FFFFFF, #8098C6, #EB5757 (для позначення помилок). Шрифт тексту – «Roboto», розмір – відповідно до дизайну. Існують дві категорії медіа продукції – «Courses» та «Serials».

1.3.3 Передбачувана аудиторія

Програмний продукт передбачений для користувачів операційної системи Android. Вікові обмеження відповідають обмеженням створення облікового запису Google або ж іншої поштової скриньки.

1.3.4 Границі проекту

Програмний продукт призначений для відслідковування прогресу перегляду медіа продуктів, а саме серіалів та мультсеріалів, а також прогресу навчання та проходження тих чи інших онлайн курсів. А також дає можливість проаналізувати щотижневу статистику витраченого часу на дозвілля та навчання. Даний продукт є новим та розрахований на користувачів мобільної операційної системи Android. Розробка програмного забезпечення призначення для демонстрації отриманих знань протягом здобуття освітнього рівня Бакалавр та задоволення власних та ринкових потреб.

1.3.5 Особливості продукту

Основними функціями додатку є:

– реєстрація / авторизація;

– створення нової активності у вигляді курсу чи серіалу зі вказуванням кількості уроків чи епізодів;
– заповнення прогресу перегляду декількох активностей одночасно;
– перегляд щоденної за поточний тиждень статистики з поділом на категорії (дозвілля та навчання), а також загальної кількості годин перегляду медіа продукції за тиждень, місяць та весь період користування.

1.3.6 Класи і характеристики користувачів

У програмному забезпеченні наявні лише два класи користувачів:

– гість – має доступ лише до функцій реєстрації та входу у додаток;
– користувач – зареєстрована особа, що має повний доступ до функціоналу програмного продукту.

1.3.7 Операційне середовище

Апаратними засобами для використання додатку є мобільні пристрої на базі OS Android версії 5.0 та вище, наявність інтернет-підключення та облікового запису Google або іншої поштової скриньки. Усі дані користувача прив’язані до його аккаунту у додатку та зберігаються у хмарі Firebase.

1.3.8 Обмеження дизайну та реалізації

При розробці додатку використовуються найпростіші елементи анімації для мінімальних затрат ресурсів пристрою. Усі значки, іконки та зображення до даються у векторному форматі svg для зберігання якості зображення при різних розширеннях екранів та мінімального навантаження на пристрій. Розробка додатку виконується з використанням Java SE8, Retrofit, Dagger, RxJava та ButterKnife. Дані зберігаються у Firebase Realtime Database. При реалізації використовувати мобільні пристрої на базі OS Android версії 5.0 та вище. При випуску наступних версій оновлення поточної відбуватиметься автоматично через систему Google Play Market.

1.3.9 Припущення і залежності

Можливі конфлікти програмного забезпечення при встановленні на мобільні пристрої з OS Android версії нижче 5.0, а також незначні зміни вигляду, залежно від розширення екрану.

1.3.10 Функції системи

Програмний продукт розробляється з використанням основних принципів об’єктно-орієнтованого програмування та хмарних сервісів для збереження основних даних користувача. Першим етапом реалізації є розробка реєстрація користувача за допомогою облікового запису Google чи іншої поштової скриньки та забезпечення можливості входу до програми (зі збереженням даних) та доступу до основного функціоналу. Наступним є етап розробки основного функціоналу, що містить у собі можливість створення активностей, відслідковування прогресу, коректного відображення та збереження даних. Останнім етапом є розрахунок та відображення статистики перегляду тих чи інших категорій активностей.

1.3.11 Послідовність «вплив реакції»

У функціоналі програми присутня чітка послідовність дій при переході з одного екрану до іншого зі зручним інтерфейсом, що дозволяє на інтуїтивному рівні зрозуміти наступні кроки. Додаток реагує на доторки до екрану та нижньої панелі кнопок (залежно від моделі мобільного пристрою можуть відрізнятися). При взаємодії з певною областю відбувається виконання відповідної їй дії.

1.3.12 Функціональні вимоги

Програмний продукт повинен чітко виконувати зазначені дії при взаємодії з відповідною областю. Наявність двох основних категорії медіа продукції та відповідного поділу статистики. Реєстрація лише через обліковий запис Google чи іншу поштову скриньку. Для одної поштової адреси можливий лише один обліковий запис у додатку. Статистика обробляється на основі введених користувачем даних. Усі дані зберігаються у Firebase Realtime Database. Відображення даних здійснюється відповідно до дизайну.

1.3.13 Інтерфейси користувача

Інтерфейс користувача розроблятиметься відповідно до дизайну та з дотриманням міжнародних вимог та стандартів, а також кольорових схем, що забезпечують адекватне сприйняття усіх кольорів, відтінків та їх поєднання. Дотримуватимуться правила зручного і легкого використання зі зрозумілим для користувача інтерфейсом та UX-сценаріями.

1.3.14. Програмні інтерфейси

Програмні інтерфейси, що використовуватимуться при реалізації додатку:

- Android.
- REST;
- XML;
- Firebase;
- Retrofit;
- Dagger;
- RxJava.

1.3.15 Інтерфейси обладнання

Інтерфейси обладнання відповідають інтерфейсам мобільних пристроїв на базі OS Android версії 5.0 та вище.

1.3.16 Інтерфейси зв'язку і комунікацій

Комунікація із користувачем здійснюється шляхом надписів, що знаходять на кожному екрані та полі, інтуїтивно зрозумілих зображень кнопок та виводу додаткових повідомлень при некоректному введенні даних чи застереженнях. Також після публікації додатку у Google Play Market користувачі матимуть змогу залишати відгуки та пропозиції для програмного продукту.

1.3.17 Вимоги до продуктивності

Мінімальним значенням оперативної пам’яті та пропускної здатності процесора, що не повинні використовуватися в умовах найбільшого навантаження, є 30%. Відповідь на запит повинна здійснюватися у межах 600 мс при хорошому інтернет-з’єднанні та у межах 2 с при поганому інтернет-

з’єднанні. apk-файл для встановлення додатку повинен займати не більше 12 Мб, усі дані програми можуть займати у пам’яті пристрою не більше 30 Мб.

1.3.18 Вимоги збереженості даних

Дані користувача та його облікового запису зберігаються у Firebase Realtime Database. Перевірка коректності введених даних перед занесенням до бази. Аналіз унікальності даних про користувача. Наявність захисту від виникнення помилок при отриманні даних з БД та запису нових у БД. Унеможливлення видалення одного елементу, якщо він пов’язаний з іншим. Коректність відображення даних відповідного юзера при багато користувальницькому режимі.

1.3.19 Критерії якості ПЗ

Програмне забезпечення повинне відповідати вимогам, зазначеним у даній специфікації, розробленому дизайну, а також таким критеріям якості [6], відповідно до стандарту ДСТУ ISO/IEC TR 9126, як:

- надійність;
- ефективність;
- зручність;
- функційність;
- супроводжуваність;
- переносність.

Особливо важливими є динамічні характеристики програмних продуктів, що визначаються часовими параметрами, та відповідають, в основному, за швидкодію та ефективність програмного забезпечення. Для визначення даних характеристик зручно використовувати математичну модель із семи аналітичних залежностей [7].

1.3.20 Вимоги до безпеки системи

Програмний код не повинен бути доступним користувачеві. Усі дані облікового запису користувача не повинні бути у відкритому доступі та не можуть бути переглянуті сторонніми системами. При отриманні дані

користувача хешуються для уникнення витоку приватної інформації власників облікових записів. Усі змінні та методи повинні бути захищені від стороннього доступу та модифікації.

1.3.21 Інші вимоги

Реалізація даних вимог, тобто розробка програмного продукту, відбуватиметься у декілька кроків, відповідно до життєвого циклу програмного забезпечення (SDLC), зображеного на рисунку 1.4:

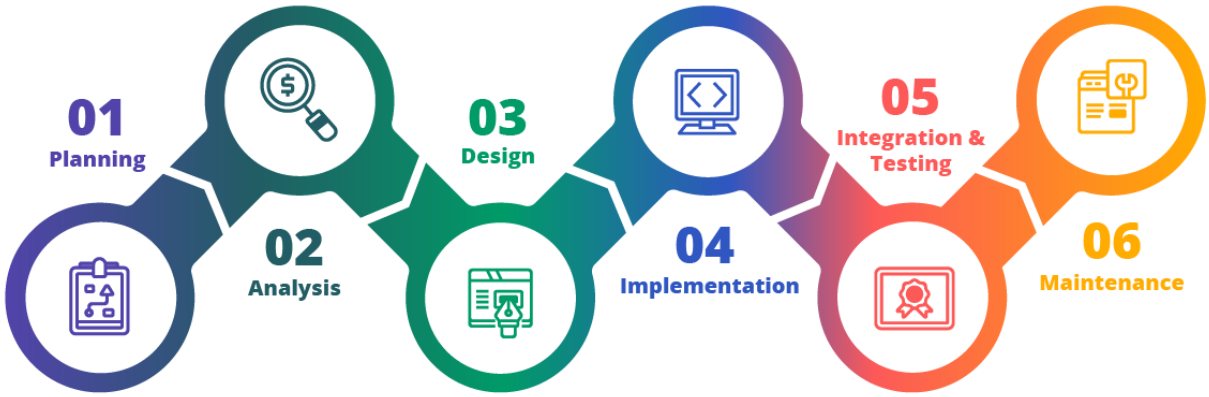


Рисунок 1.4 – Життєвий цикл програмного забезпечення [8]

Враховуючи це, перед розробкою самого продукту необхідно чітко розуміти функціонал, користувацькі сценарії, структуру збереження необхідних даних, обрати технології і засоби реалізації та спроектувати вигляд самого додатку.

2 ОГЛЯД ТЕХНОЛОГІЙ ТА ПРОЕКТУВАННЯ ПРОДУКТУ

2.1 Технології розробки додатку

Перед початком будь-якої роботи необхідно не лише визначитися з тим, що потрібно виконати, але і за допомогою чого можна реалізувати систему та вирішити те чи інше завдання.

Середовищем розробки для виконання даного проекту було обрано Android Studio [9], що є одним з основних для реалізації мобільних додатків на базі OS Android [10]. Воно пристосоване для виконання різноманітних типових завдань при розробці застосунків, у ньому присутні засоби тестування, а саме відображення вигляду інтерфейсу на пристроях з різними розширеннями (рис. 2.1) та сумісності тих чи інших компонентів. У середовище вбудовано підсистему тестування, розгортання та автоматичного складання проекту Gradle та систему контролю версій Git, що дозволяє зекономити час на використання сторонніх сервісів введенням лише декількох команд у вбудованій консолі та збільшити ефективність розробки [11].

Рисунок 2.1 – Перевірка вигляду компонентів на різних девайсах в Android Studio

Оскільки платформою для розробки мобільного додатку було обрано OS Android, базовою мовою реалізації є Java [12], що являється однією з провідних та популярних мов програмування у світі. За результатами опитування DOU Java посідає друге місце у рейтингу мов програмування 2020 року для комерційного використання [13]. На дану мову посилається більшість документації Google, а знайти необхідну бібліотеку не складає труднощів через їх різноманіття. За допомогою величезної спільноти можна вирішити найскладніші завдання, адже завжди можна знайти того, кому цікаво буде його вирішити, або ж того, хто уже стикався з такою ж проблемою. Java здатна вирішувати майже будь-які завдання розробки, є легкою для розуміння завдяки зрозумілому синтаксису та відрізняється з-поміж інших мов своєю швидкодією, надійністю та високим рівнем захисту.

RxJava - бібліотека, що реалізовує реактивний підхід до програмування та застосовується для складання та обробки послідовностей подій [14]. Її зручно застосовувати при обробці таких UI подій, як натиснення кнопок, дотиків до екрану, змінення налаштувань, завершення реєстрації, системних повідомлень та іншого. Реалізаціє здійснюється на базі такого принципу як Observe та Observable, що полягає в очікуванні виконання користувачем тої чи іншої дії та виконання на основі цього ряду наступних подій.

Retrofit 2 – бібліотека для мережевої взаємодії, що в певному роді серед розробників вважається стандартом. Значними перевагами є те, що бібліотека відмінно підтримує REST API, легко тестується і налаштовується, а запити по мережі з її допомогою виконуються зовсім просто [15]. Retrofit дозволяє легко отримати і завантажити JSON (або інші структуровані дані) через вебсервіс та при цьому не використовувати зворотні виклики та інші низько рівневі засоби [16].

Dagger 2 – ще один важливий елемент для роботи, що представляє собою бібліотеку для Dependency Injection (впровадження залежностей) в Java та Android і працює він під час компіляції [17]. Завдяки Dagger зменшуються

залежності об’єктів один від одного, існування яких знижують можливості повторного використання коду, що є однією з основних ідей об’єктно-орієнтованого програмування, ускладнює процес тестування та погіршує підтримуваність при розвитку проекту та збільшенні кількості коду.

ButterKnife – бібліотека, що значно скорочує використання однотипного коду при роботі з View елементами у Activity, фрагментах, холдерах і т.п. Також замінює громіздкі внутрішні анонімні класи однією анотацією (наприклад, для натискань - @OnClick) та дозволяє формувати списки або масиви з View і здійснювати з ними ті чи інші маніпуляції [18].

Firebase [19] є однією з найкращих та найпопулярніших систем BaaS (Backend-as-a-Service). Вона служить базою даних, яка змінюється в реальному часі і зберігає дані у форматі JSON, який зручно обробляти. Будь-які зміни в базі даних одразу синхронізуються між усіма клієнтами або девайсами, які поєднуються однією і тією ж базою даних. Інакше кажучи, оновлення у Firebase відбуваються миттєво. Окрім бази даних та сховища для різноманітних файлів користувачів, Firebase також надає можливість аутентифікації, де всі дані передаються через захищене з'єднання SSL [20]. Для реєстрації для входу підходять такі сервіси, як: Facebook, Twitter, GitHub, Google або ж звичайна комбінація поштової адреси та паролю. Також включений сервіс Crashlytics, що дозволяє відслідковувати збої у роботі додатку, відображає помилку, що стала причиною даної проблеми та кількість користувачів, які стикнулися з цим.

При розробці на Java під OS Android використовуються не тільки Java-класи, що містять програмний код, але також і файли маніфесту на мові XML, що надають системі основну інформацію про програму та підсистему автоматичного складання Gradle, де вказуються необхідні бібліотеки та їх версії [21]. XML використовується також для верстки UI-частини додатку, що дозволяє використовувати повторно фрагменти та не налаштовувати все вручну.

Ще одним важливим інструментом є Android Annotation. Анотації дозволяють в коді розставити ті чи інші смислові мітки, які говорять або

компілятору, або редактору, або віртуальній машині, що потрібно виконати ту чи іншу дію [22]. Саме це і дозволяє спростити код та очистити від повторів елементів, а також розділити пріоритети та функціонал на відповідні модулі, які з'єднуються та генеруються у фрагменті.

2.2 Побудова архітектури

Побудова правильної архітектури мобільного додатку має першочергове значення для всього проекту. Приділивши належну увагу масштабованості системи, якості коду, вибору API сервера та надійних технологій, можна уникнути більшості проблем при подальшій розробці.

Закладена архітектура безпосередньо впливає на те, чи знадобиться постійні підтримка та «ремонт» додатку або ж все буде працювати (і продовжувати працювати) як треба.

Існує ряд шаблонів для проектування архітектури додатку. Одними з найпопулярніших є MVC, MVP та MVVM. При розробці даного програмного продукту використовуватиметься патерн Model-View-Presenter (MVP). Розглянемо його детальніше. Шаблон має три складові (рис. 2.2):

- Model - являє собою інтерфейс, який відповідає за управління даними (включаючи їх кешування та управління базами даних) додатку та зберігання його бізнес-логіки;
- Presenter - виступає посередником між Model і View та відповідає за оновлення представлення, реагуючи на взаємодію користувачів з оновленням моделі. Уся логіка представлень знаходиться в Presenter, який також контролює Model і спілкується з View, що дозволяє оновлювати певний View, коли це необхідно;
- View - відповідає тільки за подання даних у вигляді, що визначаються Presenter. Подання може бути реалізовано за допомогою будь-якого Android віджета або всього, що може виконувати такі операції, як показ ProgressBar, оновлення TextView і заповнення RecyclerView [23].

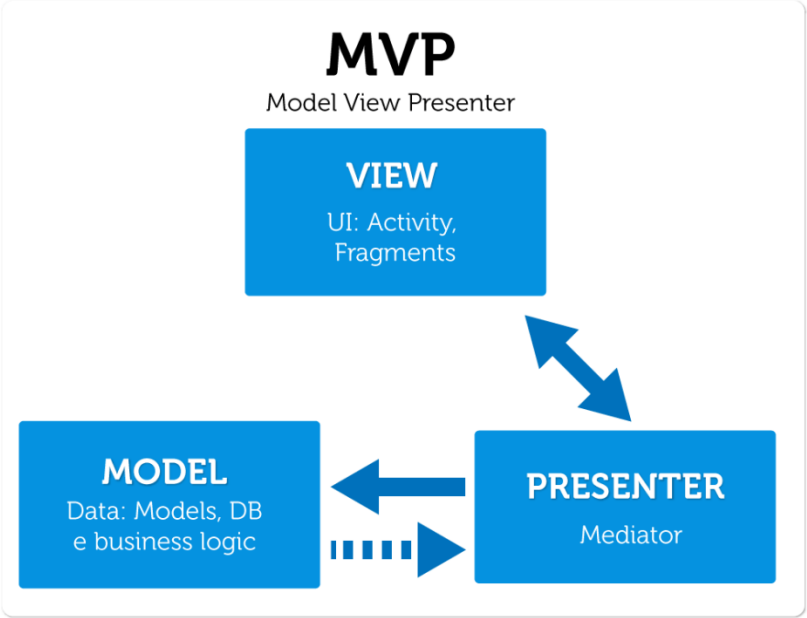


Рисунок 2.2 – Шаблон MVP [23]

Даний шаблон являє собою ефективну архітектурну модель для розробки Android додатків. Основними перевагами патерну є:

– більш просте налагодження проекту - простіше робити налагодження в додатку, так як MVP використовує три різних шари абстракцій, проводити модульне тестування також простіше, оскільки бізнес-логіка повністю відокремлена від View;
– дозволяє повторне використання коду - при побудові MVP архітектури розробники можуть багаторазово застосовувати вже написаний код, що стає можливим завдяки безлічі представників, які контролюють View;
– ефективний поділ функціональностей додатку - під цим розуміється відділення бізнес-логіки й інших частин від класів активностей і фрагментів, які, у свою чергу, краще забезпечують поділ функціональностей [23].

Firebase є достатньо потужним сервісом, щоб використовувати його як бекенд частину додатку. Даний патерн ідеально підходить, якщо:

– розробляється абсолютно нова програма або переписується існуюча з нуля;

– програма потребує мінімальної інтеграції зі застарілими системами чи іншими сторонніми сервісами;
– додаток не містить у собі обчислень та інших затратних операцій, які повинні виконуватися у бекенд частині;
– у додатку немає великих потреб у обробці даних або складних вимог аутентифікації користувачів [24].

У даній архітектурі додаток складається лише зі статичного вмісту та ресурсів, а весь динамічний вміст та дані користувача зберігаються та за необхідності отримуються з Firebase, включаючи і відновлення даних користувача (рис. 2.3).

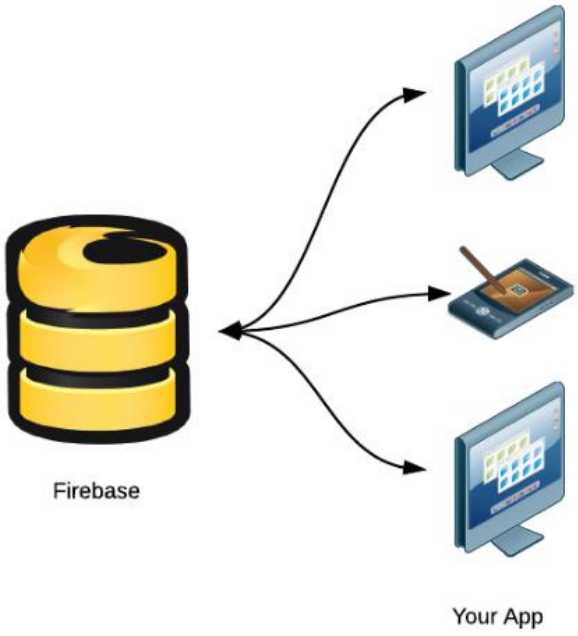


Рисунок 2.3 – Архітектура Firebase програми [24]

2.3 Проектування бази даних програмного забезпечення

Firebase Realtime Database – хмарна система управління базою даних класу NoSQL, що дозволяє зберігати та синхронізувати дані між користувачами у реальному часі [25]. ER-діаграма бази даних, необхідної для даного програмного

продукту, складатиметься з 5 таблиць (сутностей) та виглядатиме наступним чином (рис. 2.4):

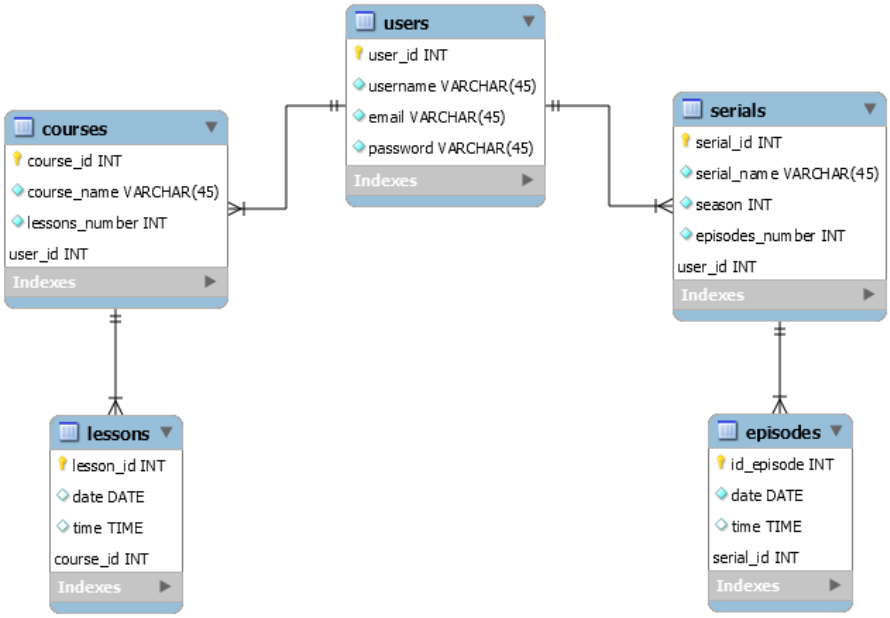


Рисунок 2.4 – Схема структури збереження даних у додатку «Mark It»

Розглянемо кожну таблицю детальніше. Таблиця users призначена для зберігання власне інформації про користувача та містить такі поля:

- user_id – INT – є первинним ключем таблиці, що однозначно визначає користувача;
- username – VARCHAR (45) – містить ім’я користувача, яке він вводить при реєстрації, або яке визначається з облікового запису Google;
- email – VARCHAR (45) – містить поштову адресу користувача;
- password – VARCHAR (45) – містить хешований пароль користувача, щоб забезпечити надійне зберігання даних.

Таблиця serials призначена для зберігання серіалів, які користувачі додають до свого списку, та містить наступні поля:

- serial_id – INT – є первинним ключем таблиці, що однозначно визначає серіал;

– serial_name – VARCHAR (45) – містить назву серіалу, яку задає користувач;
– season – INT – вказує номер сезону даного серіалу, за замовчуванням – 1;
– episodes_number – INT – містить кількість серій у даному сезоні серіалу, за замовчуванням – 1;
– user_id – INT – поле є зовнішнім ключем, що визначає зв’язок один-до-багатьох із таблицею users, оскільки серіал прив’язаний до одного користувача, проте користувач може мати безліч серіалів у своєму списку.

Таблиця episodes призначена для зберігання епізодів певного серіалу та містить наступні поля:

– episode_id – INT – є первинним ключем таблиці, що однозначно визначає серію;
– date – DATE – містить дату перегляду епізоду, що використовується під час складання статистики, за замовчуванням - поточна дата;
– time – TIME – містить тривалість епізоду, що також використовується для статистичних даних, за замовчуванням – 5 хвилин;
– serial_id – INT – поле є зовнішнім ключем, що визначає зв’язок один-до-багатьох із таблицею serials, оскільки кожен епізод прив’язаний до конкретного серіалу, а серіал має певну кількість епізодів.

Таблиця courses призначена для зберігання навчальних курсів, які користувачі додають до свого списку, та містить наступні поля:

– course_id – INT – є первинним ключем таблиці, що однозначно визначає навчальний курс;
– course_name – VARCHAR (45) – містить назву курсу, яку задає користувач;
– lessons_number – INT – містить кількість уроків у даному навчальному курсі, за замовчуванням – 1;

– user_id – INT – поле є зовнішнім ключем, що визначає зв’язок один-до-багатьох із таблицею users, оскільки кожен навчальний прив’язаний до одного користувача, проте користувач може мати безліч курсів у своєму списку.

Таблиця lessons призначена для зберігання уроків певного навчального курсу та містить наступні поля:

– lesson_id – INT – є первинним ключем таблиці, що однозначно визначає урок;
– date – DATE – містить дату проходження уроку, що використовується під час складання статистики, за замовчуванням - поточна дата;
– time – TIME – містить тривалість уроку, що також використовується для статистичних даних, за замовчуванням – 5 хвилин;
– course_id – INT – поле є зовнішнім ключем, що визначає зв’язок один-до-багатьох із таблицею courses, оскільки кожен урок прив’язаний до конкретного навчального курсу, а курс складається з одного і більше уроків.

У Real Time Database дані зберігаються не саме так, як на рисунку 2.4, а більше у вигляді дерева значень (рис. 2.5).

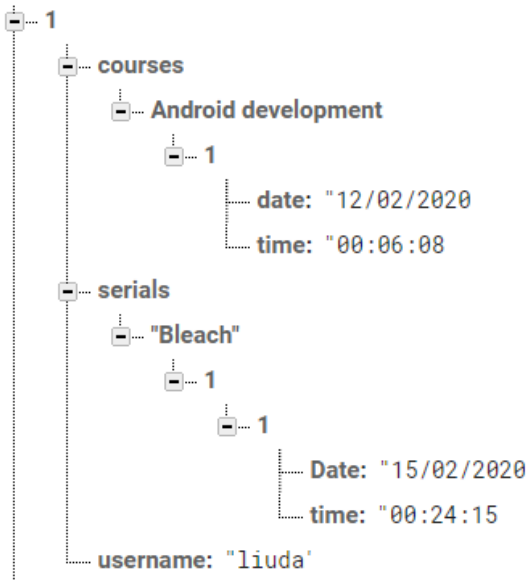


Рисунок 2.5 – Приклад дерева значень у Firebase Realtime Database

2.4 Реалізація use case діаграми функціоналу ПЗ

Use case діаграма або ж діаграма прецедентів складається з множини акторів, варіантів використання (прецедентів), що обмежені границею системи, а також асоціацій між акторами та прецедентами [26]. Дана діаграма є хорошим варіантом представлення основного функціоналу, що необхідно розробити, та взаємодії користувача з ним. Тут зручно показати створення залежностей, що відбуваються внаслідок певних дій, щоб краще зрозуміти основний user flow та можливості користувачів.

У розроблюваному програмному продукті відсутній як такий поділ на ролі. Тобто є гість, який вперше завантажує додаток та ще не має аккаунту, та уже зареєстрований користувач, що має змогу використовувати повний функціонал додатку. Відповідно гість має змогу лише зареєструватися. Тут у нього є вибір: зробити це через обліковий запис Google або ж за допомогою будь-якої існуючої електронної адреси. Користувач має змогу увійти у додаток за допомогою облікового запису Google або ж за зареєстрованою електронною адресою та паролем. Далі користувач може переглянути список власних доданих серіалів та навчальних курсів, з можливістю відобразити лише серіали, або лише курси. Також він може додати новий серіал або курс (activity). У самій activity користувач може бачити кожен епізод або урок (відповідно до категорії) та змінити прогрес, тобто відмітити переглянуті серії чи уроки, задавши дату перегляду та витрачений на це час. Наступний кроком користувач може переглянути статистику переглядів по категоріях та загалом, тобто побачити кількість витрачених годин на навчання та дозвілля за поточний день, тиждень, місяць та увесь час користування додатком. Також користувач має змогу вийти з додатку, щоб увійти з іншого облікового запису або ж уникнути перегляду особистих даних сторонніми людьми. З іншого боку діаграми є розробник, що має право оновити додаток, змінивши вигляд, виправивши помилку чи додавши новий функціонал. Описана вище діаграма прецедентів програмного забезпечення зображена на рисунку 2.6.

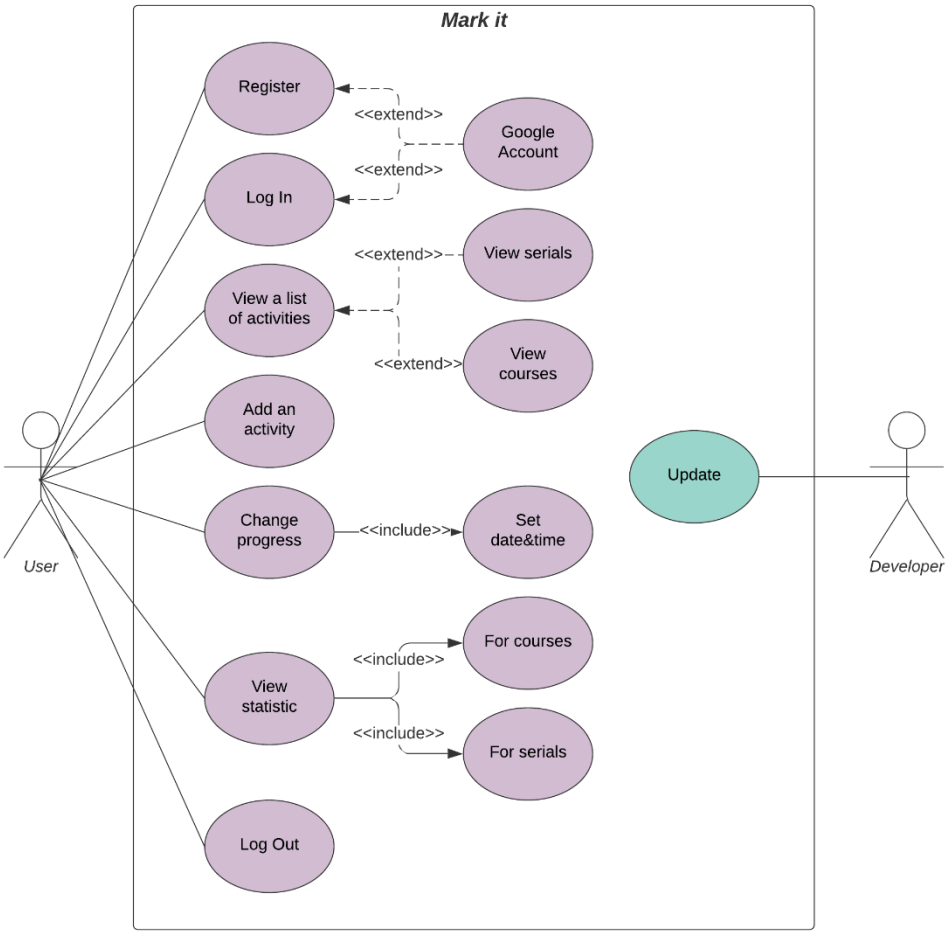


Рисунок 2.6 – Use case діаграма додатку

2.5 Розробка інтерфейсу додатку

Одним з основних етапів у життєвому циклі програмного забезпечення є розробка дизайну. Адже відповідно до дизайну і відбувається реалізація додатку. Проте варто зазначити те, що під час розробки продукту можуть здійснюватися корективи до початкового дизайну, у зв'язку із знаходженням кращого шляху реалізації та зміни інтерфейсу на зручніший.

Для дизайну даного додатку було використано векторний сервіс розробки інтерфейсів та прототипування Figma. Він дозволяє безкоштовно розробляти два проекти та зберігати онлайн-версії, що вберігає від втрати даних. Сервіс зручний тим, що має багато вбудованих та розроблених бібліотек, які спрощують процес

дизайну, зручний інтерфейс та достатню кількість уроків для навчання. Також має десктопну та браузерну версії.

Створення дизайну відбувається на основі use case діаграми та основних правил UX технологій для простоти, зрозумілості та зручності використання. Кожні малюнок та іконка є у векторному форматі svg, що дозволяє зберегти якість зображення при різних розширеннях екрану.

Перше, що бачить користувач при запуску програмного продукту, є splash screen (рис. 2.7), або екран заставки, час відображення якого напряму залежить від тривалості завантаження даних у додатку.

Рисунок 2.7 – Дизайн splash screen

Наступним є екран з можливістю входу у додаток або ж реєстрації (рис. 2.8). Тобто підходить, як для користувачів, так і для гостей додатку. Оскільки є два варіанти входу у додаток, то відповідно є дві кнопки з надписами, що чітко дають зрозуміти користувачеві, які дії виконуватимуться при взаємодії з ними. При виборі реєстрації чи входу за допомогою облікового запису Google

з’являтиметься стандартне системне вікно для вибору або додавання нового аккаунту.

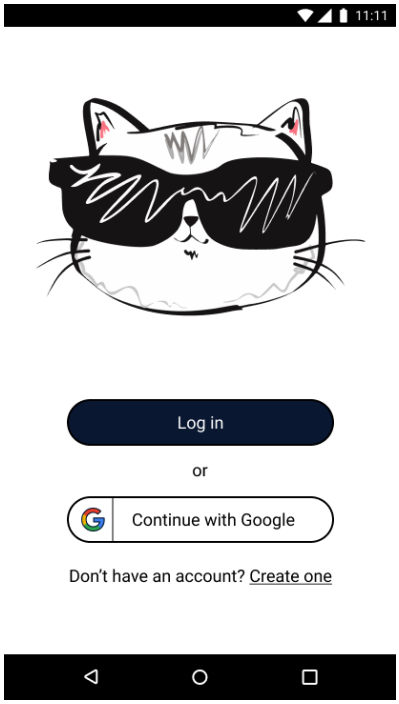
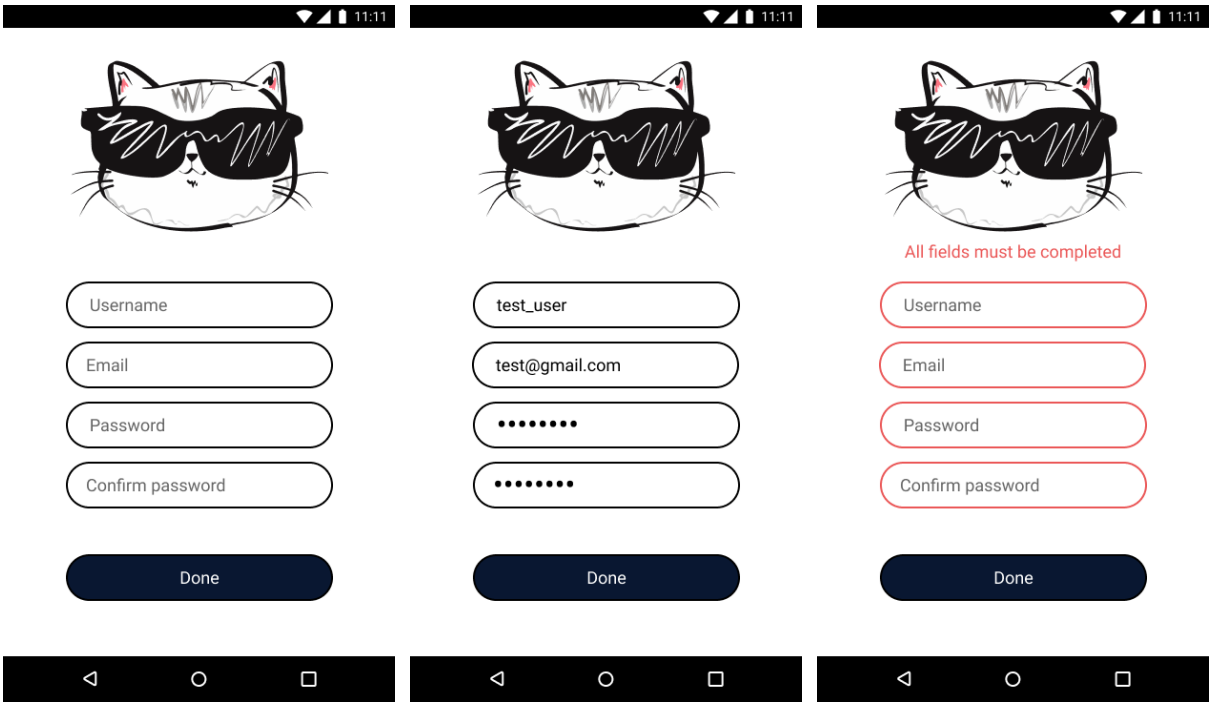


Рисунок 2.8 – Дизайн початкового екрану

При створенні нового облікового запису (рис. 2.9, а), користувачу необхідно заповнити такі дані, як ім’я, адресу електронної скриньки, пароль та підтвердження паролю (рис. 2.9, б). Усі поля повинні бути заповнені, при відсутності тих чи інших даних або їх невідповідності вимогах, користувач побачить відповідне повідомлення (рис. 2.9, в). Вимоги до полів виглядають наступним чином:

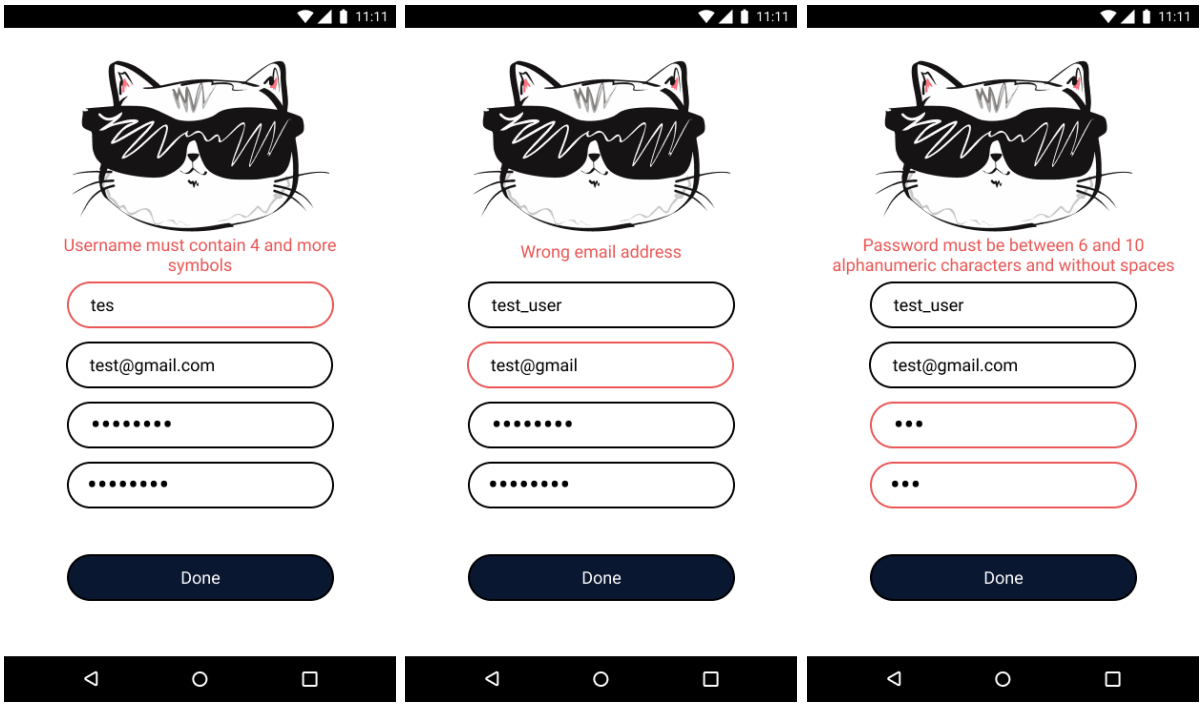
- username – повинен складатися із 4 та більше символів (рис. 2.10, а);
- email – повинен бути невикористаним у додатку та коректним (рис. 2.10, б);
- password та confirm password – повинні співпадати та складатися з 6-10 цифр або літер, або ж їх комбінації без пропусків. (рис. 2.10, в)



а) б) в)

Рисунок 2.9 – Дизайн реєстрації,

а – екран реєстрації; б – заповнені дані; в – помилка при пустих полях

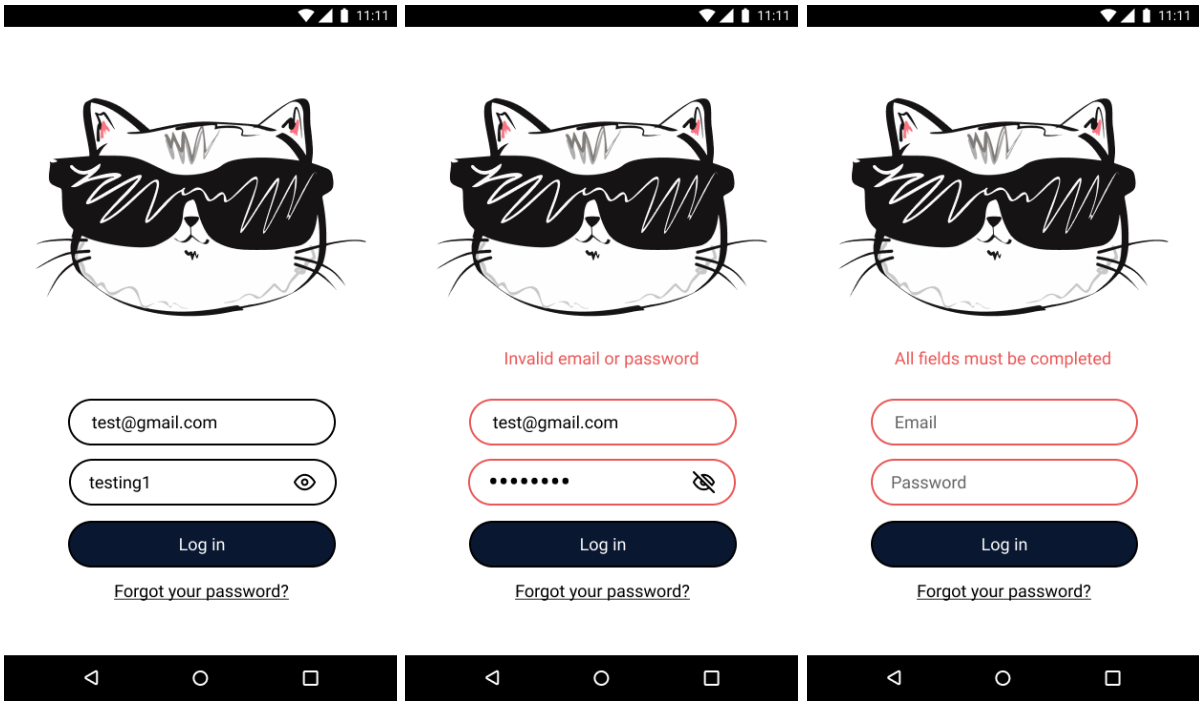


а) б) в)

Рисунок 2.10 – Дизайн реєстрації,

а – помилка username; б – помилка email; в – помилка password

При вході у додаток за допомогою адреси поштової скриньки користувачу необхідно ввести електронну адресу, за якою він реєструвався, та відповідний пароль, з можливістю його показати та приховати (рис. 2.11, а). Якщо пароль не підходить до даного облікового запису (рис. 2.11, б) або поля не заповнені (рис. 2.11, в), користувач побачить відповідну помилку.

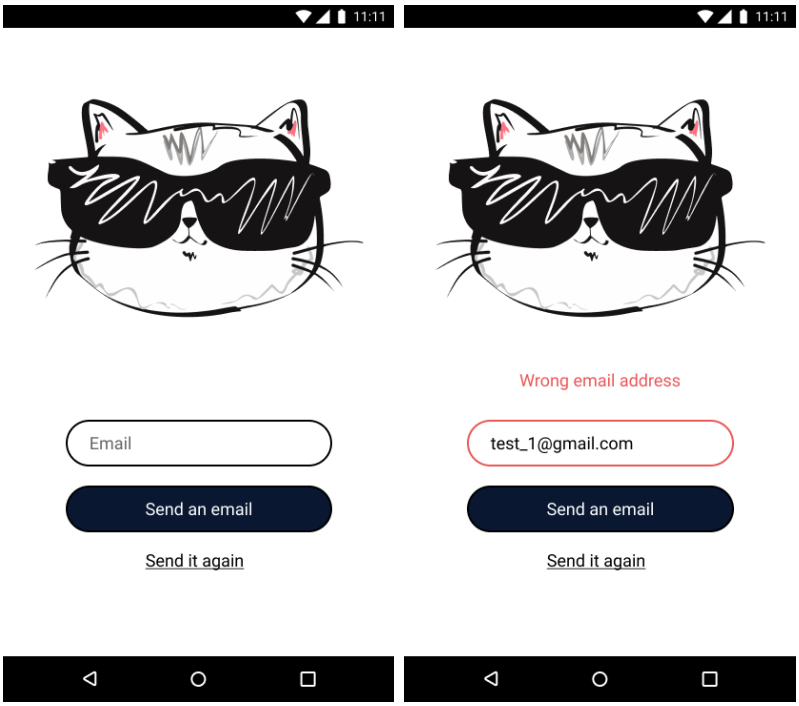


а) б) в)

Рисунок 2.11 – Дизайн входу у додаток,
а – екран входу; б – помилка при не співпадінні даних; в – помилка при порожніх полях

Якщо користувач не пам’ятає пароль до свого облікового запису, то є можливість його відновити за допомогою електронного листа. Тобто, натиснувши на рядок «Forgot your password?», користувач перейде на новий екран, де є поле вводу адреси електронної скриньки (рис. 2.12, а), до якої прив’язаний обліковий запис. Якщо адреса є некоректною або ж такої немає в базі, буде виведено відповідну помилку (рис. 2.12, б). При відсутності листа на

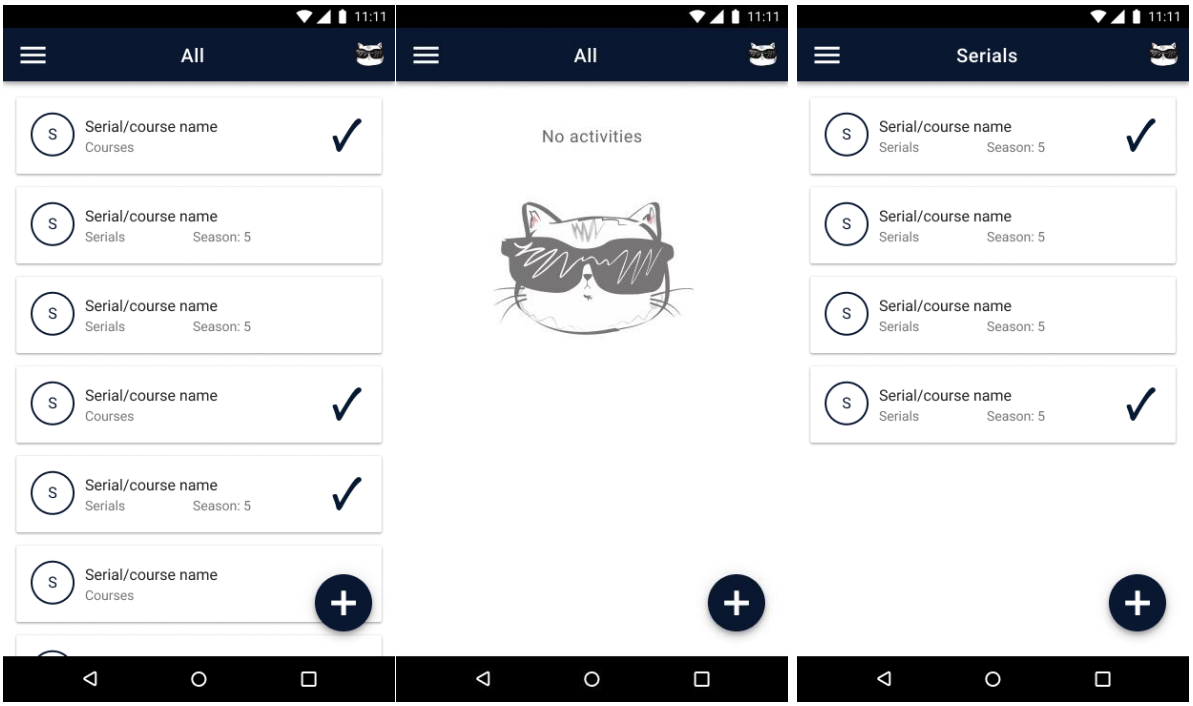
електронній скриньці, можна його відправити знову, натиснувши «Send it again». Подальше відновлення паролю здійснюватиметься у системних вікнах.



а) б)

Рисунок 2.12 – Дизайн відновлення паролю,
а – екран відновлення паролю; б – помилка при неправильній адресі скриньки

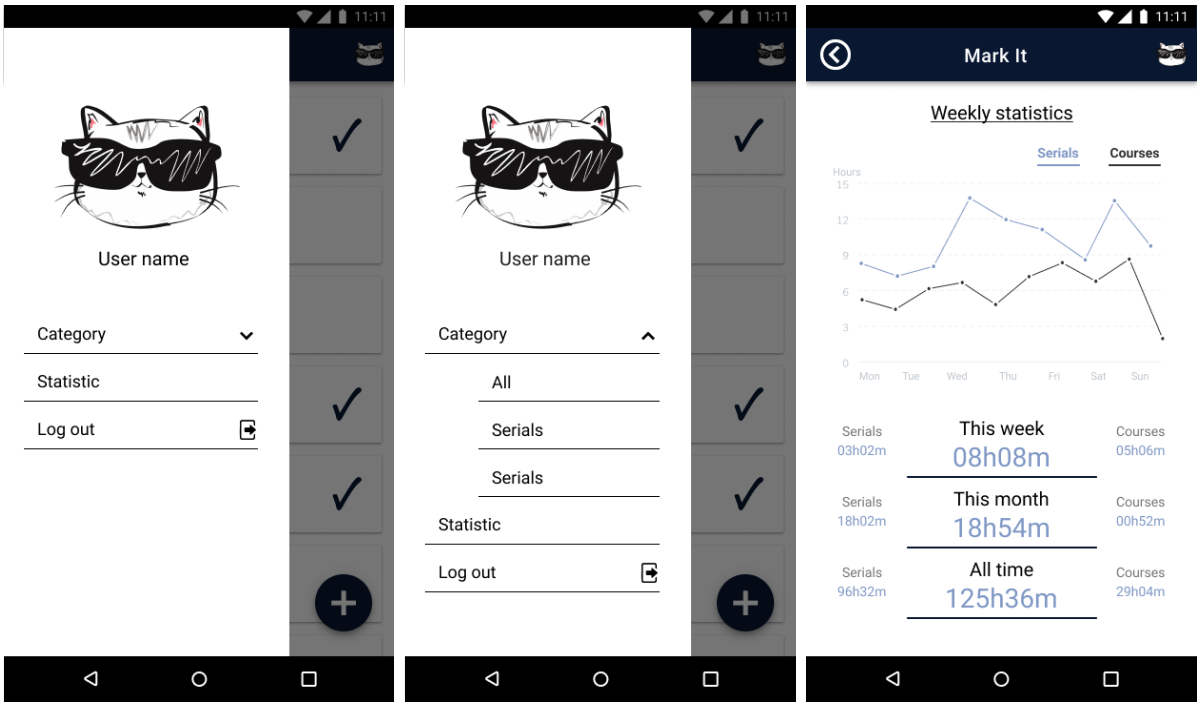
Після успішного входу у додаток користувач бачитиме основну сторінку, де відображатиметься список доданих серіалів та навчальних курсів, а завершені курси та серіали відмічені галочкою (рис 2.13, а). Новий користувач бачитиме порожній екран (рис 2.13, б) та матиме можливість додати записи. Користувач, у якого уже є дані, зможе створити новий запис та обрати, який список він хоче побачити: з усіма активностями, лише навчальними курсами чи лише серіалами (рис 2.13, в). Обрати відображення можна з бокового меню (рис 2.14, а), розкривши запис «Categories» (рис 2.14, б). У ньому ж можна вийти з додатку, а також перейти до екрану статистики (рис 2.14, в), де відображено графік тижневого затраченого часу на дозвілля та навчання, а також суму годин за поточний день, тиждень та увесь час користуванням додатком.



а) б) в)

Рисунок 2.13 – Дизайн основного екрану,

а – із доданими записами; б – нового користувача; в – лише із серіалами

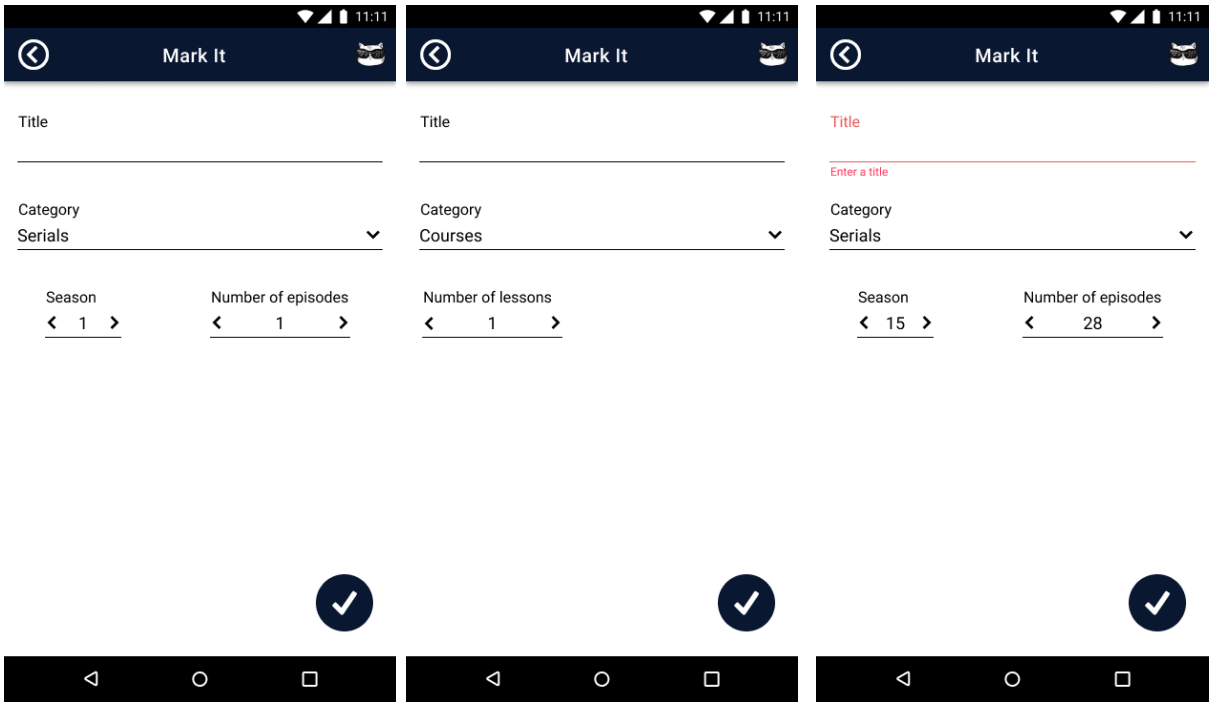


а) б) в)

Рисунок 2.14 – Дизайн бокового меню та екрану статистика,

а – бокове меню; б – вибір категорії для відображення; в – статистика

Натиснувши на плюсик у нижньому правому куті екрану, користувач може додати новий серіал або навчальний курс. Для цього відкривається новий екран, де необхідно вказати назву, обрати категорію та для серіалу – номер сезону та кількість епізодів (рис. 2.15, а), для курсу – кількість уроків (рис. 2.15, б). Додати запис можна натиснувши на галочку у правому нижньому куті. Якщо активність не матиме назви, то користувач побачить повідомлення про відповідну помилку та не зможе зберегти запис допоки не заповнить поле (рис. 2.15, в).

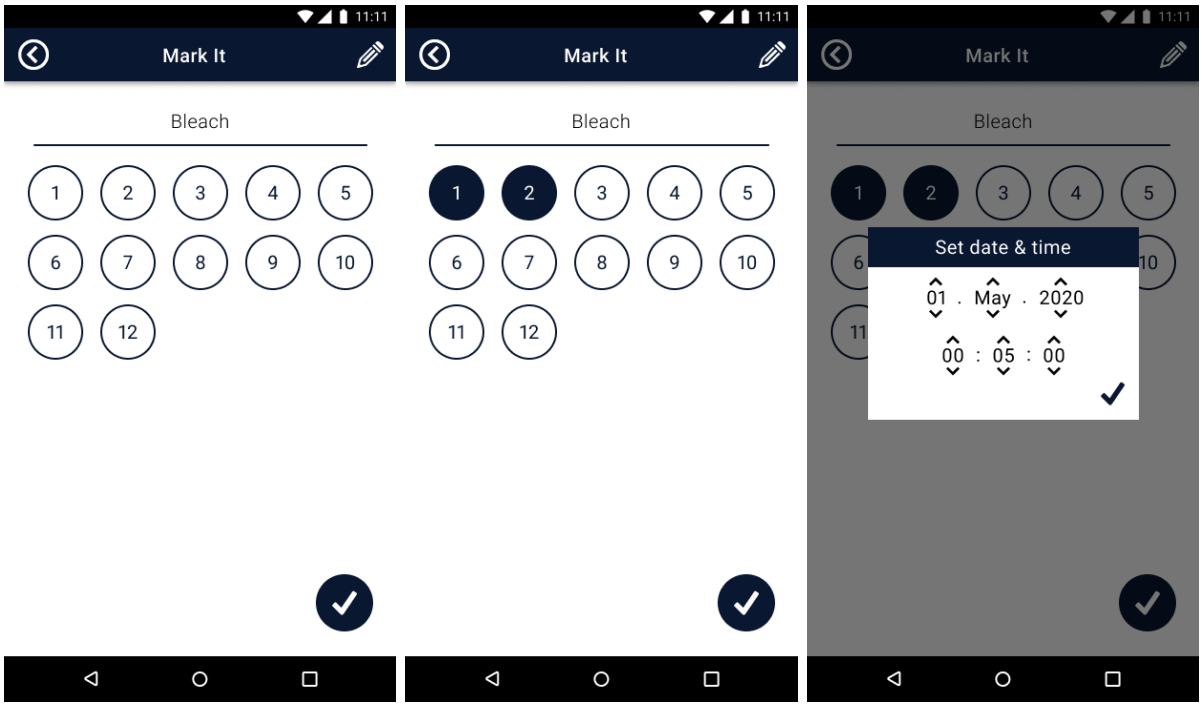


а) б) в)

Рисунок 2.15 – Дизайн створення нового запису,
а – додавання серіалу; б – додавання курсу; в – помилка при порожньому полі

Для перегляду детальної інформації про ту чи іншу активність необхідно на основному екрані натиснути на потрібний серіал чи курс. Після цього відкриється вікно, де буде відображено кількість епізодів чи уроків (рис. 2.16, а). Відповідно до прогресу, переглянуті будуть зафарбованими (рис. 2.16, б). Щоб відмітити серію чи урок завершеним, необхідно натиснути на нього та вказати

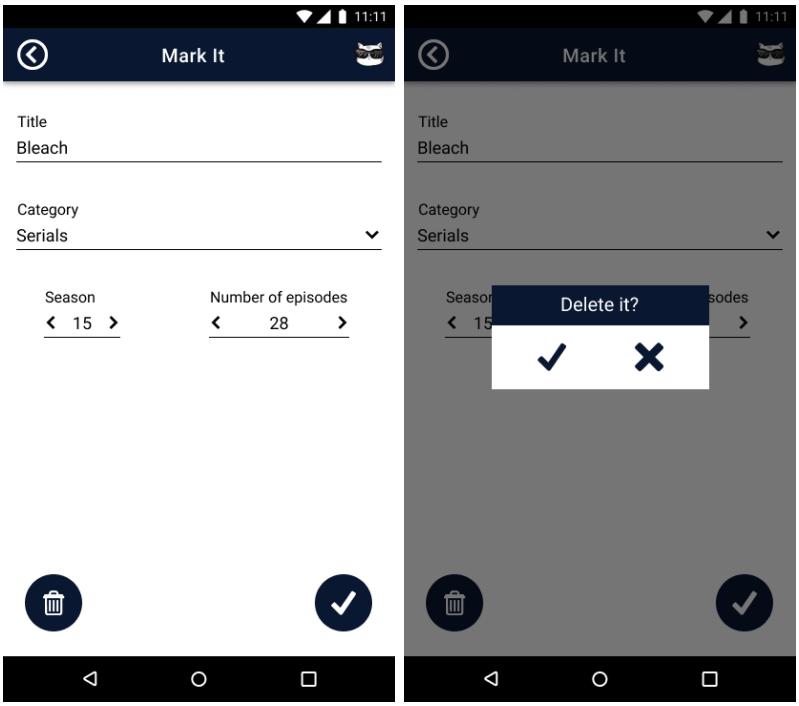
дату перегляду і витрачений на це час у діалоговому вікні, що виникне при натисканні (рис. 2.16, в), що використовуватимуться для статистики.



а) б) в)

Рисунок 2.16 – Дизайн екрану запису,
а – запис з нульовим прогресом; б – запис із переглянутими серіями; в – зазначення дати перегляду та часу

Кожну активність можна редагувати. Для здійснення даної операції потрібно у вікні запису натиснути на олівець у правому верхньому куті. Тоді відкриється вікно, аналогічне до додавання активностей, проте із заповненими даними (рис. 2.17, а), та можливістю видалити запис, натиснувши на відповідну кнопку у лівому нижньому куті екрану та підтвердивши здійснення даної операції у діалоговому вікні (рис. 2.17, б).



а) б)

Рисунок 2.17 – Дизайн редагування запису,
а – екран редагування; б – підтвердження видалення

Обравши технології, спроектувавши базу даних, описавши функціонал та розробивши дизайн можна з легкістю переходити до реалізації задуманого програмного продукту.

3 РОЗРОБКА ДЕМОНСТРАЦІЙНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Налаштування проекту

Перед початком розробки необхідно чітко вказати усі дані про програму системі Android, оскільки без них неможливо буде виконати код додатку. Саме цей крок є одним з найголовніших при створенні проекту у середовищі розробки. Основним файлом, що задає інформацію про програму, є AndroidManifest.xml. У ньому визначається простір імен Android (xmlns:android) та унікальне ім'я пакету (package), що задається при створенні проекту (краще вказувати прізвище, оскільки при виставленні додатку Google Play це перевіряється на унікальність) (рис. 3.1). Також вказуються дозволи (user-permission), які необхідні для роботи додатку, наприклад, доступ до камери, контактів, локації, Інтернету тощо (рис. 3.1).

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.shevliuk.markit">

    <uses-permission android:name="android.permission.INTERNET" />
```

Рисунок 3.1 – Початкове налаштування та надання дозволів

У тезі application, що є унікальним для проекту, необхідно задати опис наступних компонентів (рис. 3.2):

- name – вказуємо сетап якого підкласу необхідно зробити, оскільки без вказання створюється екземпляр базового класу (тобто activity), а ініціалізація класу AppApplication необхідна для коректної роботи з Dagger;
- allowBackup – дозволяє створювати резервні копії даних та відновлювати при повторному встановленні додатку;
- icon – задається розташування іконки додатку;

– hardwareAccelerated – вказується для ефективнішого використання ресурсів мобільного пристрою;
– label – вказується назва додатку, що відображатиметься користувачу на мобільному пристрої у меню додатків;
– roundIcon – для задання адаптивної іконки застосунку;
– supportRtl – задається для використання векторної графіки, тобто зображень та іконок у форматі svg;
– theme – задає основний стиль додатку.

```
<application
    android:name=".AppApplication"
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:hardwareAccelerated="true"
    android:label="Mark It"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
```

Рисунок 3.2 – Конфігурації додатку

Кожна активність повинна бути оголошена у файлі маніфесту, оскільки в іншому випадку система не зможе її побачити, що призведе до помилок роботи додатку. Для оголошення необхідно вказати клас активності, за необхідності, тему та параметри запуску (рис. 3.3).

```
<activity
    android:name=".presentation.splash.SplashActivity"
    android:theme="@style/SplashTheme">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Рисунок 3.3 – Приклад оголошення activity

У файлі маніфесту також оголошують сервіси, що будуть доступні додатку, як-от сповіщення за допомогою Firebase (рис. 3.4), що нагадуватимуть користувачу проходити курси вдень та ближче до вечора про дозвілля.

```
<service
    android:name=".data.notification.fcm.NotificationFirebaseMessagingServiceImpl"
    android:enabled="true"
    android:exported="true">
    <intent-filter>
        <action android:name="com.google.firebase.MESSAGING_EVENT" />
    </intent-filter>
</service>
```

Рисунок 3.4 –Оголошення сервісу сповіщень

Ще одним важливим файлом для коректного запуску та роботи проекту є файл системи автоматичної проекту Gradle. У ньому вказуємо основні плагіни для збирання додатку та додаткових сервісів (рис. 3.5)

```
apply plugin: 'com.android.application'
apply plugin: 'com.google.gms.google-services'
apply plugin: 'com.google.firebase.crashlytics'
```

Рисунок 3.5 – Встановлення плагінів

Gradle-файл визначає основні дані про сам додаток, тобто вказується основна та мінімальна підтримувана версії SDK, що рівноцінно підтримуваним версіям Android, версія застосунку, підтримка векторів, інструменти тестування та інше (рис. 3.6).

```
android {
    compileSdkVersion 29
    buildToolsVersion "29.0.2"

    defaultConfig {
        applicationId "com.shevliuk.markit"
        minSdkVersion 24
        targetSdkVersion 29
        versionCode 1
        versionName "1.0"
        vectorDrawables.useSupportLibrary = true
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }
```

Рисунок 3.6 – Конфігурація додатку

Необхідно також підключити потрібну версію Java (у даному випадку 8) у compileOptions та задати конфігурації для кожної побудови застосунку (release, debug та інші), вказавши необхідні налаштування, як-от url для зв'язку з бекенд частиною, налаштування кешу та його ліміту, необхідність відладки тощо (рис. 3.7).

```
compileOptions {
    targetCompatibility 1.8
    sourceCompatibility 1.8
}

release {
    debuggable false
    minifyEnabled false
    proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
    buildConfigField String, BASE_URL, "\"https://mark-it-9148f.firebaseio.com/\""
    buildConfigField String, CACHE_CONTROL_HEADER, CACHE_CONTROL_HEADER_VAL
    buildConfigField "int", LIMIT, "1000"
    buildConfigField "int", CACHETIME, "432000"
}
```

Рисунок 3.7 – Підключення Java та налаштування відповідних побудов

У файлі Gradle обов'язково прописуються усі біблотеки, що використовуватимуться у проекті з вказанням версій. Для даного додатку необхідно підключити Firebase (рис. 3.8), Dagger 2 (рис. 3.9), RxJava (рис. 3.10), ButterKnife (рис. 3.11).

```
implementation 'com.google.firebase:firebase-crashlytics:17.0.0'
implementation 'com.google.firebase:firebase-messaging:20.1.7'
implementation 'com.google.firebase:firebase-core:17.4.1'
implementation 'com.google.firebase:firebase-analytics:17.4.1'
implementation 'com.google.firebase:firebase-auth:19.3.1'
implementation 'com.google.android.gms:play-services-auth:18.0.0'
```

Рисунок 3.8 – Підключення Firebase

```
implementation 'com.google.dagger:dagger-android:2.26'
implementation 'com.google.dagger:dagger-android-support:2.26'
annotationProcessor 'com.google.dagger:dagger-compiler:2.26'
annotationProcessor 'com.google.dagger:dagger-android-processor:2.26'
```

Рисунок 3.9 – Підключення Dagger 2

```
implementation 'io.reactivex.rxjava2:rxandroid:2.1.1'
implementation 'com.jakewharton.rxbinding:rxbinding:0.4.0'
```

Рисунок 3.10 – Підключення RxJava

```
implementation'com.jakewharton:butterknife:10.2.1'
annotationProcessor 'com.jakewharton:butterknife-compiler:10.2.1'
```

Рисунок 3.11 – Підключення ButterKnife

Також важливо вказати основні репозиторії, звідки і братимуться пакети бібліотек, та залежності (рис. 3.12).

```
repositories {
    google()
    jcenter()
    maven { url "https://jitpack.io" }

}
dependencies {
    classpath 'com.google.gms:google-services:4.3.3'
    classpath 'com.android.tools.build:gradle:3.6.3'
    classpath 'com.google.firebase:firebase-crashlytics-gradle:2.1.0'
```

Рисунок 3.12 – Зазначення репозиторіїв та залежностей проекту

Звісно, що з розширенням проекту файли маніфесту та автономного складання можуть змінюватися та розширюватися, проте на початку роботи завжди потрібно вказати усі відомі та необхідні дані, без яких запуск додатку не відбудеться.

3.2 Огляд основних пакетів проекту

Ще одним важливим етапом розробки є побудова архітектури додатку, поділ класів на пакети для їх ефективної та зручної взаємодії, а також уникнення повторів однаковий частин коду, що значно знижує якість коду та самого продукту. Тож даний програмний продукт складається із наступних пакетів (рис. 3.13):

– base – опис та налаштування усіх базових класів та інтерфейсів, від яких наслідуватимуться всі інші, наприклад BaseActivity, BaseFragment, BaseDialog, BaseModel та інші;
– common – містить клас, у якому знаходяться загальні константні значення, як-от часові значення для анімації, ключі для реєстрації Google, дані для сповіщень та помилок, а також маски для regex-виразів;
– data – пакет для роботи з мережею, обробки запитів, налаштувань кешу, запису даних у кеш (shared preferences), керування дозволами, обробок помилок мережі та ssl, виклику RxJava та модуля сповіщень, що відслідковує прийняття нотифікаційних даних з бекенду, параметри їх відображення та обробки;
– di (dependency injection) – пакет для підключення і авторизації модулів та їх впровадження у застосунок;
– dialog – містить класи, що встановлюють роботу вспливаючих діалогових вікон;
– domain – пакет для модуля репозиторію, що поєднує сервіси з моделлю для витягування даних з бази;
– models – пакет для роботи з моделями;
– presentation – пакет, у якому відбувається імплементація активностей та фрагментів (скрінів), а також імплементація необхідних адаптерів, що прив’язані до скрінів, ідентифікація UI-елементів та їх взаємодія з моделями через презентер (бізнес-логіка);
– util – містить допоміжні функції, такі як декоратори, допоміжні view, google sign in client та google sign in options;

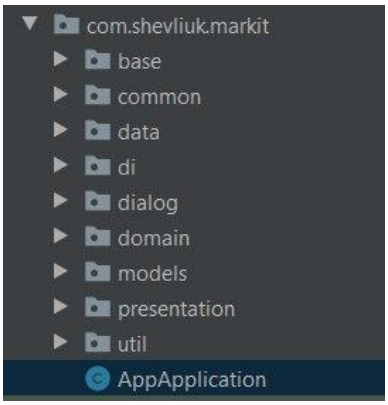


Рисунок 3.13 – Організація пакетів аплікації

3.3 Реалізація навігації у межах застосунку

Навігація є важливою складовою кожного програмного забезпечення. Оскільки завжди існують переходи з одного вікна на інше, будь то вебсайт чи мобільний додаток.

Перш за все для навігації потрібно створити інтерфейс (рис. 3.14), що керуватиме переходами на активіті, фрагменти чи меню. Відповідно до цього є функції navigateBack() для переходу при натиску на кнопку back, dispose() для закриття усіх вікон, відкриття та закриття меню, а також декілька варіантів відкриття екранів, відповідно до їх складу та необхідних даних.

```
public interface INavigator {
    boolean navigateBack();
    void closeMenu();
    void openMenu();
    <T extends Fragment> void openMenu(final Class<T> _fragmentToShow,
                                       final @Nullable Bundle _arguments);
    boolean navigateBack(final boolean _closeAll);
    void dispose();
    <T extends Fragment> void openScreen(final Class<T> _fragmentToShow,
                                         final @Nullable Bundle _arguments);

    <T extends Fragment> void openScreen(final Class<T> _fragmentToShow,
                                         final TypeBackStack _addToBackStack,
                                         final @Nullable Bundle _arguments);

    <T extends Fragment> void openScreen(final Class<T> _fragmentToShow,
                                         final TypeBackStack _addToBackStack,
                                         final @Nullable Bundle _arguments,
                                         boolean isAnimation,
                                         int enterAnimation,
                                         int exitAnimation);
```

Рисунок 3.14 – Інтерфейс навігації

Кожне активіті може відкриватися з різними умовами. Найпростіше відкриття зображено на рисунку 3.15. Проте до цього можна ще додати такі функції, як: обробка необхідних даних та відображення анімації за її наявності (рис. 3.16).

```
public final <T extends Activity> void openScreen(final Class<T> _activityToOpen) {
    openScreen(_activityToOpen,  _finishCurrent: false);
}
```

Рисунок 3.15 – Приклад імплементації відкриття активіті

```
    final Intent launchIntent = new Intent(baseActivity, _activityToOpen);
    if (extras != null) {
        launchIntent.putExtra(extras.first, extras.second);
    }
    ActivityCompat.startActivity(baseActivity, launchIntent,
            _activityOptions != null ? _activityOptions.toBundle() : null);
    if (finishAll)
        baseActivity.finishAffinity();
    else if (finishCurrent) {
        if (isAnimation) {
            baseActivity.overridePendingTransition(enterAnimation, exitAnimation);
        }
        baseActivity.finish();
    }
```

Рисунок 3.16 – Приклад імплементації відкриття активіті з анімацією та необхідними деталями

Для відкриття фрагменту також можливе вказування таких опцій, як встановлення аргументів та задання анімації (рис. 3.17).

```
fragment = baseActivity.getSupportFragmentManager().getFragmentFactory().instantiate(ClassLoader.getSystemClassLoader(), _fragmentToShow.getName());

if (_arguments != null) {
    fragment.setArguments(_arguments);
}

final FragmentTransaction fragmentTransaction = baseActivity.getSupportFragmentManager().beginTransaction();

if (isAnimation) {
    fragmentTransaction.setCustomAnimations(enterAnimation, exitAnimation);
}
```

Рисунок 3.17 – Приклад імплементації відкриття фрагменту

При відкритті нового фрагменту існують три варіанти дій з ним: додати до BackStack, де він стане першим елементом, а попередній зупинить свою діяльність (зручно здійснювати навігацію по backpress, оскільки перший видалятиметься зі стеку, а попередній ставатиме активним), видалити останній фрагмент та не додавати до стеку (рис. 3.18). Наступними операціями є заміна поточного фрагменту в активіті на новий (replace) та затвердження усіх змін (commitAllowingStateLoss) (рис. 3.18).

```
switch (_addToBackStack){
    case ADD_TO_BACK:
        fragmentTransaction.addToBackStack(fragment.getClass().getName());
        break;
    case REMOVE_LAST_FRAGMENT:
        fragmentTransaction.addToBackStack(null);
        baseActivity.getSupportFragmentManager().popBackStack();
        break;
    case NOT_ADD_TO_BACK:
        break;
}

//TODO commitALLOWStateLoss cleaning
fragmentTransaction.replace(baseActivity.getFragmentContainerId(), fragment, fragment.getClass().getName());
fragmentTransaction.commitAllowingStateLoss();
```

Рисунок 3.17 – Можливі операції із фрагментами

Зміна фрагментів відбувається наступним чином: спочатку здійснюється ідентифікація поточного фрагменту та перевірка його на нуль і чи не є він тим самим, що має бути наступним, далі відбувається ініціалізація необхідного фрагменту, перевірка його аргументів, додавання до стеку (add) та підтвердження змін (рис. 3.18).

```
final Fragment currentFragment = baseActivity.getSupportFragmentManager()
        .findFragmentById(baseActivity.getMenuContainerId());

if (currentFragment != null && currentFragment.getClass().equals(_fragmentToShow)) {
    return;
}


final Fragment fragment;

fragment = baseActivity.getSupportFragmentManager().getFragmentFactory().instantiate(ClassLoader.getSystemClassLoader(), _fragmentToShow.getName());

if (_arguments != null) {
    fragment.setArguments(_arguments);
}

final FragmentTransaction fragmentTransaction = baseActivity.getSupportFragmentManager().beginTransaction();

fragmentTransaction.add(baseActivity.getMenuContainerId(), fragment, fragment.getClass().getName());
fragmentTransaction.commitAllowingStateLoss();
```

Рисунок 3.18 – Приклад зміни фрагментів

3.4 Налаштування роботи із сервером та data management

Оскільки дані користувача зберігатимуться не локально на пристрої, а в онлайн сервісах, необхідно налаштувати правильну мережеву взаємодію для обробки та передачі даних.

Для початку потрібно встановити максимальний розмір кешованих даних на пристрої (рис. 3.19), щоб не перевантажуватися пам’ять, проте все ще збільшувати швидкодію та ефективність через стягування даних з локального сховища.

```
@Singleton
public final class NetworkConfiguration {

    private final int CACHE_SIZE = 10 * 1024 * 1024;
```

Рисунок 3.19 – Встановлення максимального значення розміру кешу

Для здійснення запитів необхідно створити такий базовий (Interceptor), що використовуватиметься при кожному запиті і додаватиме до нього необхідний заголовок (header) та URL зв’язку із сервером, що прописаний у Gradle-файлі (рис. 3.20).

```
private Interceptor getAuthRequestInterceptor() {
    return chain -> {
        Request request = chain.request();
        if (request.url().toString().startsWith(BuildConfig.BASE_URL)) {
            request = request.newBuilder()
                    .header( name: "Content-Type",  value: "application/json")
                    .header( name: "Accept",  value: "application/json")
                    .build();
        }
        return chain.proceed(request);
```

Рисунок 3.20 – Задання базового запиту

Оскільки відбувається передача даних через мережу, то обов’язково потрібно забезпечити безпечне з’єднання між сервером та клієнтом. Для цього

потрібно ініціалізувати криптографічний протокол SSL, що шифруватиме дані з використанням асиметричного алгоритму з відкритим ключем (рис. 3.21).

```
public static SSLSocketFactory getSSLSocketFactory() {
    try {
        // Install the all-trusting trust manager
        final SSLContext sslContext = SSLContext.getInstance("SSL");
        sslContext.init( km: null, trustAllCerts, new java.security.SecureRandom());
        // Create an ssl socket factory with our all-trusting manager
        return sslContext.getSocketFactory();
    } catch (KeyManagementException | NoSuchAlgorithmException e) {
        return null;
    }
}
```

Рисунок 3.21 – Ініціалізація SSL протоколу

Також потрібно ініціалізувати та налаштувати OkHttp клієнта, що задає час на запис та читання даних, додає встановлений Interceptor та створює сесію, перевіряючи її на наявність помилок (рис. 3.22).

```
final OkHttpClient.Builder okHttpBuilder = new OkHttpClient.Builder()
        .readTimeout(Keys.NetworkConstants.READ_TIMEOUT, TimeUnit.MINUTES)
        .writeTimeout(Keys.NetworkConstants.WRITE_TIMEOUT, TimeUnit.MINUTES);
for (Interceptor interceptor : defaultNetworkConfig.getInterceptors()) {
    okHttpBuilder.addInterceptor(interceptor);
}

okHttpBuilder.hostnameVerifier((hostname, session) -> true);
```

Рисунок 3.22 – Налаштування OkHttpClient

Відповідно OkHttp клієнту необхідно налаштувати Interceptor з імплементованим логером для перегляду отриманих даних, запитів, статусу запитів та іншого (рис. 3.23). Саме налаштування є схожим до заданого на рисунку 3.20, проте у даному випадку важливішим компонентом є саме логер або, іншими, словами журнал даних.

```
HttpLoggingInterceptor interceptor = new HttpLoggingInterceptor(s -> Log.i( tag: "REST_LOGGER", s));
interceptor.setLevel(HttpLoggingInterceptor.Level.BODY);
okHttpBuilder.addInterceptor(chain -> {
    Request original = chain.request();
    Request request = original.newBuilder()
            .header( name: "Content-Type",  value: "application/json")
            .header( name: "Accept",  value: "application/json")
            .header( name: "Accept-Language", getLanguage())
            .build();
    okhttp3.Response response = chain.proceed(request);
    response.cacheResponse();
    return response;
});
```

Рисунок 3.23 – Налаштування Interceptor для OkHttpClient

Для створення запитів до бази даних зручно користуватися Retrofit. Відповідно для цього необхідно налаштувати Retrofit Builder, вказавши base url, додавши OkHttpClient та здійснивши певні модифікації, також додавши конвертер json формату, конвертер, що заповнюватиме null, якщо інформація відсутня, та RxJava адаптера, як обгортку (рис. 3.24).

```
@Singleton
@Provides
static Retrofit provideBaseRetrofit(Context context, final OkHttpClient httpClient) {
    return new Retrofit.Builder()
            .baseUrl(BuildConfig.BASE_URL)
            .client(httpClient)
            .addConverterFactory(ScalarsConverterFactory.create())
            .addConverterFactory(new NullOnEmptyConverterFactory())
            .addConverterFactory(GsonConverterFactory.create())
            .addCallAdapterFactory(RxJava2CallAdapterFactoryWrapper.create(RxJava2CallAdapterFactory.create()))
            .build();
}
```

Рисунок 3.24 – Налаштування Retrofit Builder

У модулі мережі необхідно створити провайдерів, щоб отримувати дані, причому для кожного запиту повинен бути свій провайдер (рис. 3.25).

```
@Singleton
@Provides
static GetUsernameService provideUsernameService(final Retrofit _retrofit) {
    return _retrofit.create(GetUsernameService.class);
}
```

Рисунок 3.25 – Приклад створення провайдера сервісу

Також для кожного запиту повинен створюватися окремий сервіс, що міститиме шлях (endpoint), до якого звертатися по дані, та вказавши, які саме дані необхідно отримати (рис. 3.26).

```
public interface GetUsernameService {
    @GET("users/{id}/username.json/")
    Single<String> getUsername(@Path("id")String id);
}
```

Рисунок 3.26 – Приклад створення сервісу

Щоб зв'язати сервіс із моделлю необхідно створити репозиторій (рис. 3.27), що відповідатиме даним, які отримуватиме сервіс, оскільки при виклику репозиторію, викликатиметься і сервіс.

```
public interface GetUsernameRepository {
    Single<String>getUsername(String id);
}
```

Рисунок 3.27 – Приклад створення репозиторію

Для запуску усіх елементів необхідно створити клас імплементації репозиторію, де створюємо екземпляр сервісу та перевизначаємо його та підписуємо на основний потік, відслідковуючи помилки чи успішне завершення (рис. 3.28).

```
@Singleton
public class GetUsernameRepositoryImpl implements GetUsernameRepository {

    private GetUsernameService getUsernameService;

    @Inject
    GetUsernameRepositoryImpl(GetUsernameService _getUsernameService){
        getUsernameService = _getUsernameService;
    }

    @Override
    public Single<String> getUsername(String id) {
        return getUsernameService.getUsername(id)
                .subscribeOn(Schedulers.io())
                .observeOn(AndroidSchedulers.mainThread());
    }
}
```

Рисунок 3.28 – Приклад імплементації репозиторію

Варто зазначити, що усі репозиторії повинні бути прописані в модулі репозиторіїв, що є провайдером для них всіх (аналогічно до провайдера сервісу) (рис. 3.29).

```
@Module
public abstract class RepositoryModule {

    @Singleton
    @Binds
    abstract GetUsernameRepository provideGetUsernameRepository (final GetUsernameRepositoryImpl username);
```

Рисунок 3.29 – Приклад додавання репозиторію і Repository Module

Виклик репозиторію відбувається тоді, коли користувач переходить на екран, де ці дані відображаються. Відповідно тоді презентер звертається до моделі, щоб отримати дані, у відповідь модель передає необхідну інформацію або ж помилку, яка через презентер передається відображенню (рис. 3.30).

.

```
@Override
void getUsername(String id, String name) {
    addDisposable(getUsernameRepository.getUsername(id)
            .subscribe(response -> safeDelegate(() -> requirePresenter().showUser(response, name, id)),
                    throwable -> safeDelegate(() -> requirePresenter().showError( text: "username Error"))));
}
```

Рисунок 3.30 – Приклад виклику даних

У презентері відповідно викликаємо метод з моделі, який і виконує запит з рисунку 3.30. Запит запускається відносно результату реєстрації firebase, тобто якщо результат успішний і користувач не дорівнює null, передаємо в метод ідентифікатор користувача та ім’я (рис. 3.31).

```
@Override
void registerCallback(Task<AuthResult> task, FirebaseUser user) {
    if(task.isSuccessful() && user != null){
        requireModel().getUsername(user.getUid(), user.getDisplayName());
    }
```

Рисунок 3.31 – Одержання даних користувача

Сама ж реєстрація, виконується при кліку на відповідну кнопку, у даному випадку через Google. Також реалізовано реєстрацію через email і пароль, але, фактично, результатом того чи іншого є створення запису в базі даних firebase. При кліку на кнопку google_holder через анотацію кліку, яку реалізовує бібліотека ButterKnife, запускаємо intent (намірення), яке відкриває системне вікно та чекає допоки користувач зробить вибір. Передаємо для старту цього вікна сам intent і код-специфікатор, який ідентифкуватиме intent в подальшому (рис. 3.32).

```
@OnClick(R.id.google_holder)
void signGoogle(){
    Intent signInIntent = UtilFunction.getGoogleSignInClient(requireContext()).getSignInIntent();
    startActivityForResult(signInIntent, Keys.Google.GOOGLE_REGISTER_CODE);
}
```

Рисунок 3.32 – Реєстрація через Google

При певній взаємодії з системним вікном перевизначаємо слухач андроїду, що відповідно до дій користувача присилає два коди підтвердження, а також дані, які вже передаються наступним intent. Оскільки його викликати можна тільки у фрагменті або актівіті, ці дані (а саме requestCode, intent і актівіті, яке потрібне буде в подальшому через контекст) відправляємо у презентер, який і займається бізнес-логікою (рис. 3.33).

```
@Override
public void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    requirePresenter().handleGoogleResponse(requestCode, data, requireActivity());
}
```

Рисунок 3.33 – Передача даних до презентера

У презентері збираємо дані, що були передані з фрагменту і перевіряємо requestCode, також перевіряємо, чи intent був отриманий у результаті вибору акаунту у ході реєстрації, чи це системний шум у вигляді системних підпрограм.

У разі підтвердження коду відсилаємо дані і контекст на подальшу обробку у модель, у разі негативного результату - відображаємо користувачу системне повідомлення з помилкою (рис. 3.34).

```
@Override
void handleGoogleResponse(int requestCode, Intent data, Activity activity) {
    if (requestCode == Keys.Google.GOOGLE_REGISTER_CODE) {
        requireModel().getGoogleToken(data, activity);
    }
    else {
        requireView().showBaseError( text: "Google auth failed: wrong request code");
    }
}
```

Рисунок 3.34 – Перевірка відповіді Google

При проходженні intent до наступного етапу, вже точно відомо, що у ньому є вся необхідна інформація про користувача. Для подальшої успішної реєстрації необхідно взяти акаунт користувача, який можна створити відштовхуючись від intent, що надійшов із презентера. Створивши Task треба з нього витягнути акаунт, попередньо здійснивши перевірку на API проблеми з боку сервера. При знаходженні помилки, повідомляємо про це користувача, з відповідним кодом помилки, а при успішному проходженні витягуємо акаунт юзера і надсилаємо в подальший етап реєстрації (рис. 3.35).

```
@Override
void getGoogleToken(Intent data, Activity activity) {
    Task<GoogleSignInAccount> task = GoogleSignIn.getSignedInAccountFromIntent(data);

    try {
        GoogleSignInAccount account = task.getResult(ApiException.class);
        if (account != null){
            firebaseGoogleAuth(account, activity);
        }
    } catch (ApiException e) {
        requirePresenter().showError(CommonStatusCodes.getStatusCodeString(e.getStatusCode()));
    }
}
```

Рисунок 3.35 – Отримання даних акаунту

Наступним етапом є безпосередня реєстрація користувача, відповідно до тих даних, які пройшли фільтрацію попередніх етапів. Спочатку створюємо

об’єкт даних, які реалізуються через Google провайдер, що доступний через бібліотеку, а також використовуючи токен, який беремо з об’єкту акаунта, що пройшов попередню обробку. Наступним кроком є логін. За допомогою об’єкта авторизації підписуємся відповідно нашими даними і налаштовуємо слухачі на успішність і помилку. За наявності помилки – показуємо її, а саме повідомлення виводимо з об’єкту помилки. При успішному завершенні - передаємо дані, а саме екземпляр таску і зареєстрованого користувач, презентеру (рис. 3.36).

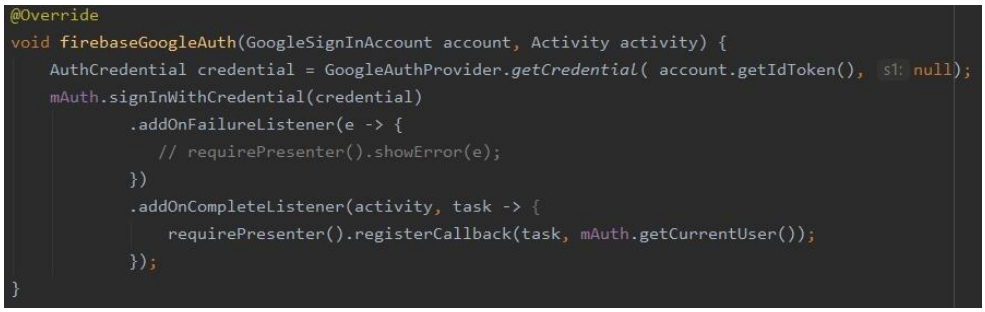


Рисунок 3.36 – Реєстрація користувача

Наступним етапом іде перевірка на успішність реєстрації. Із попереднього кроку приймаємо таск, у якому перевіряємо успішність виконання реєстрації, а також, чи користувач не є null. При похибці відображаємо системне повідомлення з текстом похибки, яке витягуємо із самого об’єкту таску. При успішному виконанні - даємо запит на перевірку, чи користувач вже існує в базі даних. Для цього передаємо у модель id користувача і username, що витягується з firebase об’єкту (рис. 3.37).

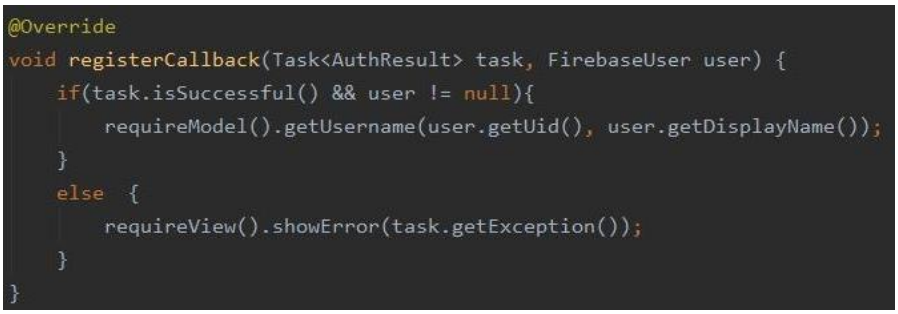


Рисунок 3.37 – Перевірка успішності реєстрації

При успішності запиту виконуємо наступні дії. Для початку в презентері перевіряємо response на null. Якщо підтверджується, значить у базі немає запису з таким користувачем, тому викликаємо метод на запис даних, для ідентифікатора даємо id користувача і відповідне ім’я, що відповідно реєстрації з email акаунтом береться від користувача firebase, а якщо через пароль та email, то від поля username (рис. 3.38).

```
@Override
void showUser(String userResponse, String name, String id) {

    if (userResponse.equals("null")){
        requireModel().patchUsername(id, new Username(name));
    }
    else {
        requireView().startMainActivity();
    }
}
```

Рисунок 3.38 – Перевірка наявності користувача у базі

Безпосередня реалізація запиту на реєстрацію користувача в базу даних відбувається шляхом приймання id та username, тоді викликається екземпляр репозиторію і через нього викликаємо функцію сервісу, за допомогою якого і виконується запит. У даному випадку використовується patch-запит, оскільки треба при кожному зверненні оновити базу і при наявності нового запису вставляти його. Запускається функція за допомогою RxJava для кращого виконання клієнт-серверного спілкування. Підписуємо це в основний потік, відловлюємо відповідь за допомогою делегату і відповідно при успішності викликаємо метод, який переносить на наступне вікно через презентер, а при невдачі - відображаємо системну похибку з відповідним текстом (рис.3.39).

```
@Override
void patchUsername(String id, Username name) {
    addDisposable(patchUsernameRepository.patchUsername(id, name)
            .subscribe(response -> safeDelegate(() -> requirePresenter().onSuccess()),
                    throwable -> safeDelegate(() -> requirePresenter().showError( text: "username Error"))));
}
```

Рисунок 3.39 – Запис користувача у базу

3.5 Організація інтерфейсу

Для звернення з інших компонентів до фрагменту потрібні специфічні конструкції – інтерфейси (рис. 3.40), щоб мати змогу звертатись з презентера, а також з моделі і представлення до презентера (для презентера і моделі використовуємо абстрактні класи, які в подальшому наслідують екземпляри моделі і презентера). Інтерфейси імплементуються у фрагменті або активіті і перевизначаються, так можна організувати імплементацію методів за участі необхідних ui компонентів. Також за парадигмою MVP звертаємося з view до презентера через абстрактні класи, які потім також перевизначаються і записується імплементація. Аналогічно і з моделлю.

```
interface View extends IView {
    void startLoginScreen();
    void startRegistrationScreen();
    void startResetScreen();
    void startMainActivity();
    void showBaseError(String text);
}
```

Рисунок 3.40 – Приклад інтерфейсу

Для того щоб ідентифікувати ui елемент з фрагменту або активіті, використовуємо анотації, які підключаються за допомогою ButterKnife. Надалі анотації ButterKnife використовуються для різних типів колбеків по взаємодії з ui елементами, як-от: кліки, колапси та різного типу активності. У даному випадку ідентифікуємо редагування тексту (EditText) для поштової адреси за заданим id, який ми записуємо у xml-файл. id - це ідентифікатор, який використовується для ідентифікації ui елементів для подальшого використання або оновлення. Зберігається він у файлі ресурсів проекту. В андроїді є різні типи ресурсів, але групи локального типу зберігаються у R файлі самого проекту.

```
@BindView(R.id.email)
EditText email;
```

Рисунок 3.41 – Приклад елементу

Реалізація ui елементу EditText для email відбувається через теги (рис. 3.42). Спочатку ідентифікуємо view через id, який є унікальним у всьому файлі, встановлюємо ширину за шириною екрану, а висоту таку, щоб охоплювала висоту контенту. Фон робимо прозорим і підтягуємо з андроїд сховища, виставляємо потрібні відступи у 22 dp та верхні і нижні відступи в 10 dp (пікселі). Ідентифікуємо тип даних, які будуть введені, тобто email текстові дані, і встановлюємо розмір тексту у 14 sp відповідно до макету. На останок, встановлюємо підказку відповідно до макету, значення підказки зберігаємо у файлі ресурсів рядків. Це необхідно зробити для хорошого тону, а також для спрощення імплементації різних локалізацій. Даний компонент входить до складу layout, який в свою чергу є складовою головного екрану. Всі набори layout і формують вид нашого додатку.

```
<EditText
    android:id="@+id/email"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@android:color/transparent"
    android:paddingStart="22dp"
    android:paddingEnd="22dp"
    android:paddingTop="10dp"
    android:paddingBottom="10dp"
    android:textSize="14sp"
    android:inputType="textEmailAddress"
    android:hint="Email"/>
```

Рисунок 3.42 – Приклад елемента view

Аналогічно до прикладу даного елементу описуються і інші відповідно до розроблених мокапів.

4 БІЗНЕС-ПЛАН РОЗРОБКИ ДОДАТКУ «Mark It»

4.1 Резюме проекту

Мобільний додаток займатиметься наданням послуг організаційного характеру, зокрема допомагатиме користувачам впорядковувати переглянуту та заплановану до перегляду медіа продукцію, маючи на увазі навчальні курси та серіали.

Цільовою аудиторією, тобто майбутніми користувачами додатку, є люди, що надають перевагу онлайн освіті або ж просто застосовують її поряд із стаціонарною та проводять частину власного дозвілля за переглядом різноманітної медіа продукції, що складається із одного та більше епізодів чи частин.

Для реалізації проекту, а саме розробки та релізу продукту для загального доступу, необхідно залучити 2320$.

Фінансування буде здійснюватися за рахунок коштів інвесторів та особистих вкладень.

Плановий об’єм завантажень додатку за перший рік роботи становить 153 тисячі, що відповідає в середньому 12,75 тисяч завантажень на місяць.

Організаційно-правовою формою майбутнього бізнесу буде діяльність фізичної особи підприємця (ФОП), оскільки створення та реалізація здійснюватиметься одноосібно.

4.2 Маркетингова діяльність

Користувачам програмного продукту буде запропоновано мобільний додаток на базі OS Android, що дозволятиме відслідковувати прогрес перегляду медіа продуктів та проходження навчальних курсів за допомогою одного лише програмного забезпечення. Цінність пропозиції полягає у наступному:

– усі переглянуті, заплановані та у процесі медіа продукти знаходяться у одному місці;
– безкоштовний продукт;
– ненав’язлива реклама;
– кооперація із навчальними платформами;
– економія часу;
– ефективність та зручність використання;
– визначення кількості витраченого часу на дозвілля та навчання;
– простота та витриманість інтерфейсу.

Розроблений продукт призначений для використання у повсякденному житті, не потребує специфічних знань та не потребує визначеної сфери застосування. Він не прив’язаний до конкретної області та є загальнодоступним для користувачів різних категорій.

Унікальна відмінність додатку полягає у поєднанні прогресу навчання та проведення дозвілля, що дає змогу аналізувати та коригувати час, відведений для кожного категорії. Також користувач власноруч створює курс чи серіал, оскільки у базах додатків не завжди можна знайти необхідний серіал, а прогрес курсу можна відмітити лише на платформі, де він знаходиться, що є не зручним при використанні декількох ресурсів одночасно.

Додаток потраплятиме «до рук» користувачів через Google Play Market, що є доступним для усіх власників мобільних пристроїв на базі OS Android. Там необхідно буде лише завантажити та встановити продукт. Перехід до сторінки додатку здійснюватиметься з реклами, розміщеної на сторонніх ресурсах, пошуку в мережі та самому Play Store.

Застосунок буде постійно підтримуватися та оновлюватися, базуючись на відгуках користувачів, подальшим розвитком технологій та постійним аналізом ринку. При виявленні тих чи інших недоліків будуть прийняті усі міри для їх усунення та удосконалення застосунку.

Ринок збуту додатку не має як таких обмежень, адже користувачем може бути будь-хто, не залежно від віку, статі, соціального стану, місця проживання, роду занять та рівня доходів. Все, що потрібно для користування – з'єднання з мережею та мобільний телефон з OS Android.

Потенційними клієнтами є не лише користувачі застосунку, а й платформи онлайн навчання та сервіси перегляду серіалів та мультсеріалів у мережі інтернет, оскільки у додатку буде можливість додати їх рекламу. Використання застосунку буде безкоштовним, а монетизація здійснюватиметься якраз завдяки рекламі, кооперація із сторонніми сервісами, а також добровільними вкладеннями користувачів.

На ринку існує декілька схожих додатків, проте всі вони мають свої відмінності. Основними перевагами додатку «Mark It» є поєднання прогресу навчання та дозвілля, а також безкоштовність використання повного функціоналу та можливість відновлення даних після повторного встановлення застосунку, що відсутнє у аналогах або є платними функціями.

Попит на даний продукт можна відслідкувати проаналізувавши пошукові запити та їх популярність за допомогою Google Trends, де цифри показують популярність пошукового запиту відносно найвищої точки на графіку для певного періоду:

- 100 – пік популярності;
- 50 – популярність вдвічі менша;
- 0 – замало даних про цей запит.

За останній рік інфографіка запитів «courses tracker» (рис. 4.1) та «serial tracker» (рис. 4.2) досить коливалася, проте не втрачала своєї популярності. З цього випливає актуальність розробки даного продукту, виходу на ринок у даний момент часу та отримання прибутку саме за рахунок кількості завантажень та актуальної кількості користувачів. Також постійна актуальність пошуку продукту такого типу дозволяє залучити інвесторів, що будуть зацікавлені розвитком додатку.

Рисунок 4.1 – Інфографіка запиту «courses tracker»

Рисунок 4.2 – Інфографіка запиту «serial tracker»

Враховуючи дані інфографіки, можна зробити оптимістичний (рис. 4.3), реалістичний (рис. 4.4) та песимістичний (рис. 4.5) прогнози завантажень, що прямо впливатимуть на прибуток від програмного забезпечення.

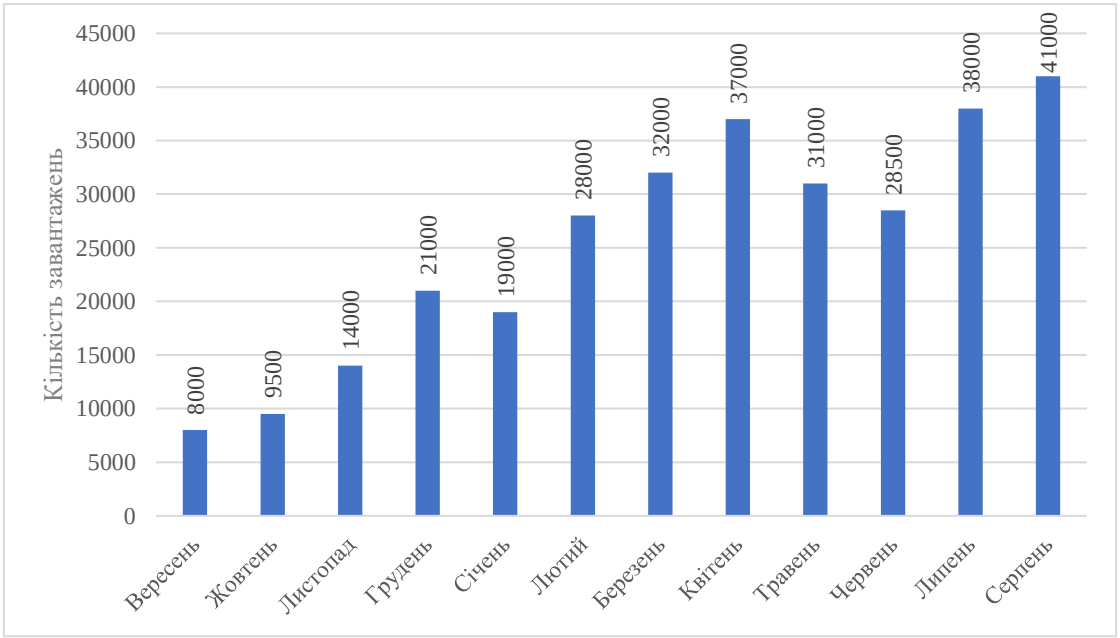


Рисунок 4.3 – Оптимістичний прогноз завантажень

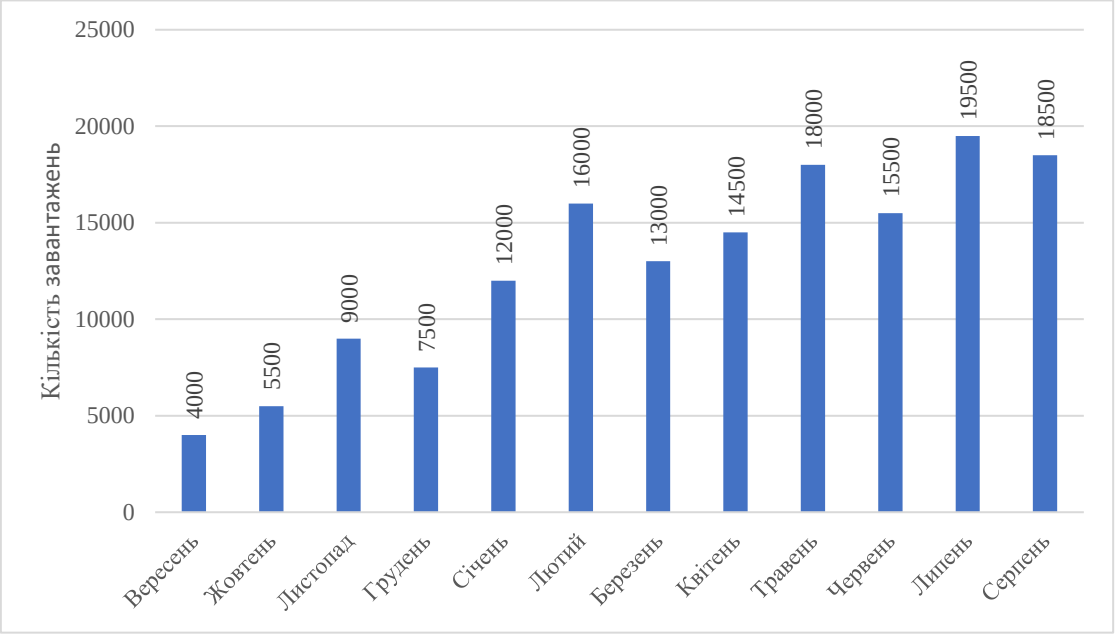


Рисунок 4.4 – Реалістичний прогноз завантажень

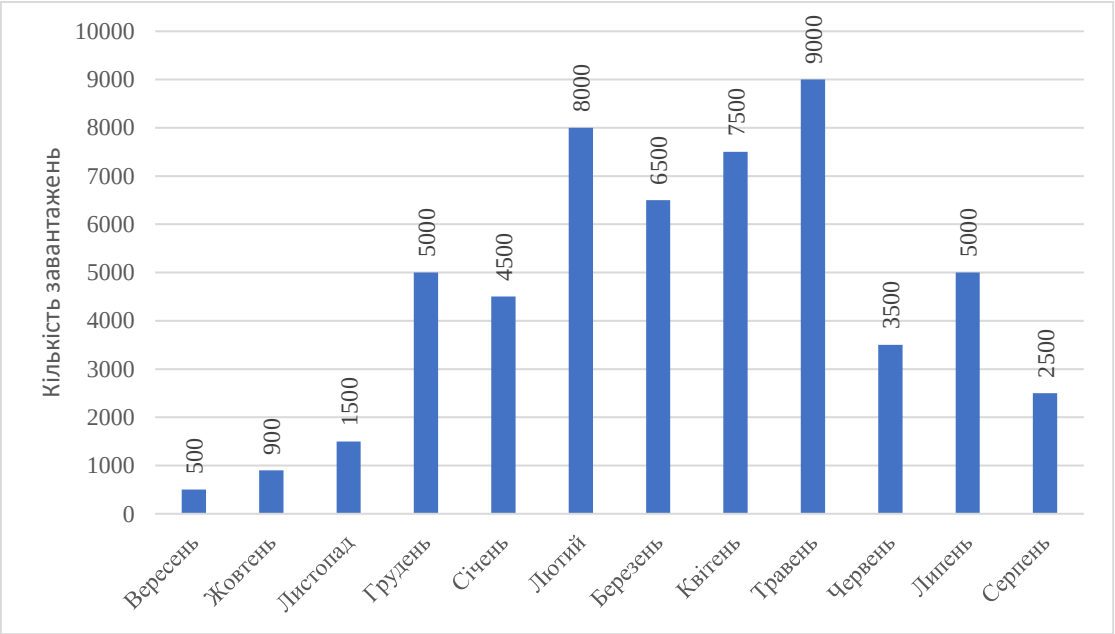


Рисунок 4.5 – Песимістичний прогноз завантажень

Потенційні користувачі дізнаються про запуск додатку за допомогою реклами у соціальних мережах, інших додатках та пов’язаних тематикою вебсайтах. На початкове просування додатку від дня запуску до отримання перших прибутків планується витратити 400$, що становить 17% від загального бюджету.

Рекламуючи товар необхідно показати основний функціонал за допомогою декількох екранів додатку, а також перелічивши основні переваги, що виділяють його серед аналогів. Щомісячно на рекламу додатку виділятиметься 5-12% від прибутку.

4.3 Нормативно-правові нюанси діяльності

Для початку роботи найкраще обрати діяльність фізичної особи підприємця (ФОП), оскільки керування проектом відбуватиметься одноосібно, з поступовим розширенням штату, та даний статус надає можливість спрощеної системи оподаткування.

Реєстрація та отримання статусу ФОПа є нескладною процедурою, що вимагає лише подати декілька документів, таких як:

- паспорт громадянина України;
- заповнену реєстраційну картку на проведення державної реєстрації фізичної особи – підприємця, за формою № 10, що затверджена наказом Міністерства юстиції України від 14.10.2011 р № 3178/5;
- копію картки платника податків;
- електронний підпис за подачі документів у електронному форматі [27].

За наявності та відповідності усіх необхідних паперів реєстрація відбуваються протягом 48 годин та не потребує жодних фінансових вкладень.

Враховуючи подальший розвиток продукту, варто зазначити, що форма ведення бізнесу може бути змінена у майбутньому.

За необхідності професійної підтримки буде звернено за послугами бухгалтера, юриста чи банківського працівника.

4.4 Обгрунтування необхідних фінансових вкладень та бюджету

Для розробки програмного забезпечення необхідні такі виробничі потужності, як приміщення з доступом до електроенергії та мережі Інтернет

(може бути житловим приміщенням), а також один ноутбук/ПК для аналізу, проектування, дизайну та розробки програмного продукту, ведення необхідної маркетингової та бізнес діяльності. Необхідне програмне забезпечення є безкоштовним для одного продукту, платна підтримка здійснюватиметься поступово та відповідно до розширення.

Трудомісткість розробки програмного забезпечення визначатиметься через суму наступних значень:

- дослідження області розробки та ринкової ситуації – 10 люд/год;
- підготовка й опис поставленої задачі – 15 люд/год;
- проектування програмного продукту – 9 люд/год;
- розробка дизайну – 30 люд/год;
- реалізація backend частини – 95 люд/год;
- реалізація frontend частини – 80 люд/год;
- тестування програмного забезпечення – 30 люд/год.

Загалом, трудомісткість розробки даного продукту становить 269 людино-годин. Година роботи програміста рівня Junior становить 5$/год, тож собівартість розробки додатку становитиме 1345$.

Для впровадження програмного продукту необхідні будуть такі додаткові витрати:

- витрати на електроенергію – 10$ на місяць – 20$ на період розробки;
- оплата аккаунту розробника у Google Play Market – 25$;
- витрати на маркетингову діяльність (сума встановлюється з власних міркувань, оскільки Google Ads самостійно підбирає план відповідно до бюджету) – 400$;
- амортизаційні відрахування, що становлять 15% від витрат на розробку – 200$;
- оплата податків (для фізичної особи підприємця ІІ типу становить на місяць не більше 20% від мінімальної заробітної плати) – 40$ на місяць – 80$ на період розробки;

– непередбачувані витрати (ризики) – 250$.

Загальна сума, що необхідна для розробки додатку становить 2320$, що у відсотковому співвідношенні витрат на ті чи інші категорії виглядає наступним чином (рис. 4.6):

Рисунок 4.6 – Діаграма витрат на розробку програмного продукту

Для отримання прибутку від використання користувачами додатку прийнято рішення використовувати внутрішню рекламу, що полягає у відображенні реклами сторонніх сервісів у попередньо визначеному місці інтерфейсу додатку. Прибуток вираховується виходячи з кількості переглядів та/або переходів по рекламі.

При показі 3 рекламних оголошень на годину та середньому використанні додатку 20 хв на день з одного пристрою в день буде 1 показ, що рівноцінно 30 показам на місяць. Вартість одного показу становить близько 0,1$. Тож з одного пристрою в місяць прибуток становитиме 3$. При реалістичному плані завантажень у перший місяць буде 4000, врахувавши використання додатку не щоденно та не з початку місяця, у середньому лише 1000 користувачів можна враховувати у місячному плані. Таким чином, прибуток становитиме 3000$.

Рентабельність продукту визначаємо як відношення прибутку від реалізації до собівартості, що становитиме 3000$ / 2320$ = 1,29. Оскільки визначена рентабельність більше 1, впровадження продукту вважається економічно ефективним та доцільним.

ВИСНОВКИ

Метою роботи над дипломним проектом була розробка мобільної аплікації для відстежування найпопулярніших активностей користувачів, а саме перегляду серіалів та проходження навчальних курсів, що дозволятиме постійно бачити прогрес діяльності. У ході виконання було здійснено аналіз ринкових потреб сфери та існуючих варіантів їх задоволення. На основі нього, визначено основні недоліки аналогів та перетворено на власні переваги. Відповідно визначено основний функціонал, що підійде користувачам, та створено зручний та інтуїтивно зрозумілий інтерфейс для виконання необхідних дій.

Для реалізації функціоналу здійснено підбір найкращих технологічних рішень розробки, а саме таких бібліотек та сервісів як: Dagger 2, Retrofit 2, Firebase, RxJava, ButterKnife; та побудову ефективної клієнт-серверної архітектури. Інтерфейс програмного забезпечення розроблено у відповідності до створеного дизайну та з урахуванням найкращих UI/UX технологій та практик.

У відповідності до усіх вимог та поставлених завдань, у кінцевому результаті, розроблено мобільний додаток на базі OS Android для моніторингу медіа продукції, а саме відслідковування навчального прогресу та прогресу проводження дозвілля (перегляду тих чи інших серіалів чи мультсеріалів).

Для подальшого розвитку продукту заплановано розширення функціоналу та можливостей користувачів, як-от:

- можливість додавання нотаток до курсів чи уроків;
- створення списків для майбутнього перегляду;
- додавання нагадувань про серію чи курс;
- можливість створення власних категорій;
- реалізація між користувальницької бази медіа продуктів;
- імплементація реєстрації через інші соціальні мережі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

REFERENCES

1. Рынок смартфонов вернется к росту уже в будущем году. URL: https://www.ixbt.com/news/2019/09/10/rynok-smartfonov-vernetsja-k-rostu-uzhe-v-budushem-godu.html
   (дата звернення: 06.11.2019).
2. Як смартфони змінюють життя. URL: http://uc.kr.ua/2019/09/17/kak-smartfony-menyayut-obraz-zhyzny/
   (дата звернення: 11.11.2019).
3. Наскільки українці залежні від телебачення — статистика https://24tv.ua/naskilki_ukrayintsi_zalezhni_vid_telebachennya__statistika_n650731
   (дата звернення: 21.11.2019).
4. Rachel Watts, Ready-to-Use Habit Trackers: Log Daily Actions, Build Healthy Routines, Achieve Goals and Live Your Best Life, Ulysses Press, ISBN 978-1-61243-917-4, 2019.
5. М. Дутчак, "Моделювання процесу автоматизованої побудови індивідуалізованого навчання в інтелектуальних освітніх онлайн системах", Прикладні науково-технічні дослідження: матеріали IV міжнар. наук.-практ. конф., Івано-Франківськ, 2020, т.1., с. 47-51.
6. Кузь М.В., Соловко Я.Т., Андрейко В.М. Методологія формування узагальненого критерію якості програмного забезпечення в умовах невизначеності. Вісник Вінницького політехнічного інституту. Вінниця, 2015. №5. С. 104-107.
7. Kuz M., Shevliuk L., Ostafiichuk T., Mishagin R. Modeling of software time characteristics. Applied scientific and technical research: Proceedings of the IV

International Scientific and Practical Conference, Ivano-Frankivsk, April 1-3, 2020, Ivano-Frankivsk, V.2. 2020. P. 39-42.

8. When Penetration Testing Must Be Part of Your SDLC. URL: https://softjourn.com/blog/article/when-penetration-testing-must-be-part-of-your-sdlc (дата звернення: 28.11.2019).
9. Barbara Hohensee, Getting Started with Android Studio, Barbara Hohensee, ISBN 978-1-301-75433-5, 2013.
10. Ian F. Darwin, Android Cookbook: Problems and Solutions for Android Developers, "O'Reilly Media, Inc.", ISBN 978-1-4493-7449-5, 2017.
11. Android Studio. URL: https://uk.wikipedia.org/wiki/Android_Studio (дата звернення: 06.12.2019).
12. Robert Liguori; Patricia Liguori, Java 8 Pocket Guide: Instant Help for Java Programmers, "O'Reilly Media, Inc.", ISBN 978-1-4919-0111-3, 2014.
13. Рейтинг мов програмування 2020. URL: https://dou.ua/lenta/articles/language-rating-jan-2020/ (дата звернення: 09.12.2019).
14. Tomasz Nurkiewicz; Ben Christensen, Reactive Programming with RxJava: Creating Asynchronous, Event-Based Applications, "O'Reilly Media, Inc.", ISBN 978-1-4919-3162-2, 2016.
15. Используем Retrofit 2 в Android-приложении. URL: https://devcolibri.com/getting-started-with-retrofit-in-android/ (дата звернення: 12.12.2019).
16. Использование Retrofit 2.x в качестве REST клиента — Tutorial. URL: https://habr.com/ru/post/428736/ (дата звернення: 12.12.2019).
17. Dagger 2 для начинающих Android разработчиков — Введение. URL: https://habr.com/ru/post/343248/

(дата звернення: 16.12.2019).

18. Обзор и пример использования библиотеки ButterKnife в Android. URL: https://javadevblog.com/obzor-i-primer-ispol-zovaniya-biblioteki-butterknife-v-android.html
(дата звернення: 18.12.2019).
19. Laurence Moroney, The Definitive Guide to Firebase: Build Android Apps on Google's Mobile Platform, Apress, ISBN 978-1-4842-2943-9, 2017.
20. Введение в Firebase: пишем простое социальное приложение на Swift. URL:
https://habr.com/ru/post/277941/
(дата звернення: 20.12.2019).
21. На чём пишут приложения для Android. URL:
https://livetyping.com/ru/blog/na-chem-pishut-prilozhenija-pod-android
(дата звернення: 23.12.2019).
22. Кратко об аннотациях в Java. URL:
http://blog.harrix.org/article/7231
(дата звернення: 26.12.2019).
23. Построение MVP архитектуры Android приложения: советы и технологии. URL:
https://smartum.pro/ru/blog-ru/mvp-architecture-development-for-android-apps-tips-and-technologies/
(дата звернення: 09.01.2020).
24. The Firebase Blog: Where does Firebase fit in your app? URL:
https://firebase.googleblog.com/2013/03/where-does-firebase-fit-in-your-app.html?m=1
(дата звернення: 14.01.2020).
25. Firebase. URL:
https://ru.bmstu.wiki/Firebase
(дата звернення: 14.01.2020).

26. Діаграма прецедентів. URL:
https://uk.wikipedia.org/wiki/Діаграма_прецедентів
(дата звернення: 22.01.2020).
27. Фізична особа – підприємець: порядок реєстрації. URL:
https://homebiznes.in.ua/fizychna-osoba-pidpryjemets-poryadok-rejestratsiji/
(дата звернення: 28.04.2020).

ДОДАТОК А

Лістинг програми (клас MainActivity.java)

```
package com.shevliuk.markit.presentation.main;

import android.content.Context;
import android.os.Bundle;
import androidx.annotation.Nullable;
import androidx.fragment.app.FragmentManager;

import com.google.android.material.floatingactionbutton.FloatingActionButton;
import com.mxn.soul.flowingdrawer_core.ElasticDrawer;
import com.mxn.soul.flowingdrawer_core.FlowingDrawer;
import com.shevliuk.markit.R;
import com.shevliuk.markit.base.BaseActivity;
import com.shevliuk.markit.base.DefaultNavigator;
import com.shevliuk.markit.base.TypeBackStack;
import com.shevliuk.markit.presentation.main.menu.MenuFragment;
import com.shevliuk.markit.presentation.main.root.RootFragment;

import butterknife.BindView;



public class MainActivity extends BaseActivity<MainPresenter, DefaultNavigator> implements
        MainContract.View {

    @BindView(R.id.drawer_layout)
    FlowingDrawer flowingDrawer;

    @BindView(R.id.fab)
    FloatingActionButton fab;

    @Override
    protected int getLayoutResource() {
        return R.layout.activity_main;
```

```
@Override
    protected void onViewReady(@Nullable Bundle _savedInstanceState) {
        super.onViewReady(_savedInstanceState);
        flowingDrawer.setTouchMode(ElasticDrawer.TOUCH_MODE_BEZEL);
        setFloatingActionButton(fab);
        getNavigator().openScreen(RootFragment.class, TypeBackStack.NOT_ADD_TO_BACK, null);
        setupMenu();
        requirePresenter().getUser();
      }

    @Override
    protected final int getFragmentContainerId() {
        return R.id.frame_layout;
    }

    @Override
    protected int getMenuContainerId() {
        return R.id.id_container_menu;
    }

    @Override
    public Context getContext() {
        return this;
    }

    @Override
    public void setupMenu() {
       getNavigator().openMenu(MenuFragment.class, null);
    }

    @Override
    public void onBackPressed() {
        if (flowingDrawer.isMenuVisible()) {
            flowingDrawer.closeMenu();
        } else {
```

```
            super.onBackPressed();
        }
    }

    @Override
    protected FlowingDrawer getMenu() {
        return flowingDrawer;
    }
}
```

Лістинг програми (клас MainModule.java)

```
package com.shevliuk.markit.presentation.main;

import com.shevliuk.markit.base.BaseActivity;
import com.shevliuk.markit.di.scope.FragmentScope;
import com.shevliuk.markit.presentation.main.action.ActionContract;
import com.shevliuk.markit.presentation.main.action.ActionFragment;
import com.shevliuk.markit.presentation.main.action.ActionModel;
import com.shevliuk.markit.presentation.main.action.ActionModule;
import com.shevliuk.markit.presentation.main.action.add.AddContract;
import com.shevliuk.markit.presentation.main.action.add.AddFragment;
import com.shevliuk.markit.presentation.main.action.add.AddModel;
import com.shevliuk.markit.presentation.main.action.add.AddModule;
import com.shevliuk.markit.presentation.main.action.edit.EditContract;
import com.shevliuk.markit.presentation.main.action.edit.EditFragment;
import com.shevliuk.markit.presentation.main.action.edit.EditModel;
import com.shevliuk.markit.presentation.main.action.edit.EditModule;
import com.shevliuk.markit.presentation.main.menu.MenuContract;
import com.shevliuk.markit.presentation.main.menu.MenuFragment;
import com.shevliuk.markit.presentation.main.menu.MenuModel;
import com.shevliuk.markit.presentation.main.menu.MenuModule;
import com.shevliuk.markit.presentation.main.root.RootContract;
import com.shevliuk.markit.presentation.main.root.RootFragment;
import com.shevliuk.markit.presentation.main.root.RootModel;
import com.shevliuk.markit.presentation.main.root.RootModule;
```

```
import com.shevliuk.markit.presentation.main.statistic.StatisticContract;
import com.shevliuk.markit.presentation.main.statistic.StatisticFragment;
import com.shevliuk.markit.presentation.main.statistic.StatisticModel;
import com.shevliuk.markit.presentation.main.statistic.StatisticModule;

import dagger.Binds;
import dagger.Module;
import dagger.android.ContributesAndroidInjector;

@Module
public abstract class MainModule {

    @MainScope
    @Binds
    abstract MainContract.Presenter providePresenter(final MainPresenter presenter);

    @MainScope
    @Binds
    abstract MainContract.View provideView(final MainActivity activity);

    @MainScope
    @Binds
    abstract BaseActivity provideActivity(final MainActivity activity);

    @FragmentScope
    @ContributesAndroidInjector(modules = RootModule.class)
    abstract RootFragment provideRootFragment();

    @MainScope
    @Binds
    abstract RootContract.AbsModel provideRootModel(final RootModel rootModel);

    @FragmentScope
    @ContributesAndroidInjector(modules = MenuModule.class)
    abstract MenuFragment provideMenuFragment();
```

```
    @MainScope
    @Binds
    abstract MenuContract.AbsModel provideMenuModel(final MenuModel menuModel);

    @FragmentScope
    @ContributesAndroidInjector(modules = ActionModule.class)
    abstract ActionFragment provideActionFragment();

    @MainScope
    @Binds
    abstract ActionContract.AbsModel provideActionModel(final ActionModel actionModel);

    @FragmentScope
    @ContributesAndroidInjector(modules = StatisticModule.class)
    abstract StatisticFragment provideStatisticFragment();

    @MainScope
    @Binds
    abstract StatisticContract.AbsModel provideStatisticModel(final StatisticModel
statisticModel);

    @FragmentScope
    @ContributesAndroidInjector(modules = EditModule.class)
    abstract EditFragment provideEditFragment();

    @MainScope
    @Binds
    abstract EditContract.AbsModel provideEditModel(final EditModel editModel);

    @FragmentScope
    @ContributesAndroidInjector(modules = AddModule.class)
    abstract AddFragment provideAddFragment();

    @MainScope
    @Binds
    abstract AddContract.AbsModel provideAddModel(final AddModel addModel);
```

```
}
```

Лістинг програми (клас StartPresenter.java)

```
package com.shevliuk.markit.presentation.registration.start;

import android.app.Activity;
import android.content.Intent;
import android.util.Log;

import androidx.annotation.NonNull;

import com.google.android.gms.tasks.Task;
import com.google.firebase.auth.AuthResult;
import com.google.firebase.auth.FirebaseUser;
import com.shevliuk.markit.common.Keys;
import com.shevliuk.markit.models.retrofit.username.Username;

import javax.inject.Inject;

public class StartPresenter extends StartContract.AbsPresenter {

    private String name;

    @Inject
    public StartPresenter(@NonNull StartContract.View _view,
                          @NonNull StartContract.AbsModel _model) {
        super(_view, _model);
    }

    @Override
    void handleGoogleResponse(int requestCode, Intent data, Activity activity) {
        if (requestCode == Keys.Google.GOOGLE_REGISTER_CODE) {
            requireModel().getGoogleToken(data, activity);
        }
        else {
```

```
            requireView().showBaseError("Google auth failed: wrong request code");
        }
    }

    @Override
    void showError(String text) {
        requireView().showBaseError(text);
    }

    @Override
    void registerCallback(Task<AuthResult> task, FirebaseUser user) {
        if(task.isSuccessful() && user != null){
            requireModel().getUsername(user.getUid(), user.getDisplayName());
        }
        else  {
            requireView().showError(task.getException());
        }
    }

    @Override
    void onSuccess() {
        requireView().startMainActivity();
    }

    @Override
    void showUser(String userResponse, String name, String id) {

        if (userResponse.equals("null")){
            requireModel().patchUsername(id, new Username(name));
        }
        else {
            requireView().startMainActivity();
        }
    }
```

```
}
```

Лістинг програми (клас StartPresenter.java)

```
package com.shevliuk.markit.presentation.registration.start;

import android.content.Intent;
import android.os.Bundle;
import android.widget.Toast;

import androidx.annotation.Nullable;

import com.shevliuk.markit.R;
import com.shevliuk.markit.base.BaseFragment;
import com.shevliuk.markit.base.DefaultNavigator;
import com.shevliuk.markit.base.TypeBackStack;
import com.shevliuk.markit.common.Keys;
import com.shevliuk.markit.presentation.main.MainActivity;
import com.shevliuk.markit.presentation.registration.login.LoginFragment;
import com.shevliuk.markit.presentation.registration.register.RegisterFragment;
import com.shevliuk.markit.util.UtilFunction;

import java.security.Key;

import butterknife.OnClick;

public class StartFragment extends BaseFragment<StartPresenter, DefaultNavigator>
        implements StartContract.View{
    @Override
    protected int getLayoutResource() {
        return R.layout.fragment_start;
    }

    @Override
    protected void onViewReady(Bundle _savedInstanceState, Boolean fragmentReturnsFromBackStack)
{
```

```
    }


    @Override
    public void startLoginScreen() {
        getNavigator().openScreen(LoginFragment.class, TypeBackStack.ADD_TO_BACK, null,true,
R.anim.anim_exit_in, R.anim.anim_exit);
    }


    @Override
    public void startRegistrationScreen() {
        getNavigator().openScreen(RegisterFragment.class, TypeBackStack.ADD_TO_BACK, null,true,
R.anim.anim_exit_in, R.anim.anim_exit);
    }


    @Override
    public void startResetScreen() {
      //  getNavigator().openScreen(ResetFragment.class,null);
    }


    @Override
    public void startMainActivity() {
        getNavigator().openScreen(MainActivity.class, true);
    }


    @Override
    public void showBaseError(String text) {
        Toast.makeText(requireContext(), text, Toast.LENGTH_LONG).show();
    }


    @OnClick(R.id.log_in)
    void login(){
        startLoginScreen();
    }


    @OnClick(R.id.google_holder)
    void signGoogle(){
```

```
        Intent signInIntent =
UtilFunction.getGoogleSignInClient(requireContext()).getSignInIntent();
        startActivityForResult(signInIntent, Keys.Google.GOOGLE_REGISTER_CODE);
    }

    @OnClick(R.id.creation_acc)
    void registration(){
        startRegistrationScreen();
    }

    @Override
    public void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        requirePresenter().handleGoogleResponse(requestCode, data, requireActivity());
    }
}
```

Лістинг програми (клас RegisterPresenter.java)

```
package com.shevliuk.markit.presentation.registration.register;

import android.app.Activity;
import android.util.Log;
import android.widget.EditText;

import androidx.annotation.NonNull;

import com.google.android.gms.tasks.Task;
import com.google.firebase.auth.AuthResult;
import com.google.firebase.auth.FirebaseUser;
import com.jakewharton.rxbinding.widget.RxTextView;
import com.shevliuk.markit.R;
import com.shevliuk.markit.common.Keys;
import com.shevliuk.markit.models.additional.RegistrationValidation;
import com.shevliuk.markit.models.retrofit.username.Username;

import java.util.concurrent.TimeUnit;
```

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import javax.inject.Inject;

import butterknife.OnFocusChange;
import rx.android.schedulers.AndroidSchedulers;

public class RegisterPresenter extends RegisterContract.AbsPresenter {

    private Username username;

    public void setUsername(String username) {
        this.username = new Username(username);
    }

    private RegistrationValidation registrationValidation = new RegistrationValidation();

    @Inject
    public RegisterPresenter(@NonNull RegisterContract.View _view,
                             @NonNull RegisterContract.AbsModel _model) {
        super(_view, _model);
    }

    @Override
    void registerCallback(Task<AuthResult> task, FirebaseUser user) {
        if(task.isSuccessful() && user != null){
            requireModel().patchUsername(user.getUid(), username);
        }
        else  {

if(task.getException().getMessage().equals(Keys.Error.FIREBASE_ERROR_EMAIL_ALREADY_EXIST)){
                requireView().showEmailAlreadyTaken();
            }
            else {
                requireView().showError(task.getException());
```

```
            }
        }
    }

    @Override
    void checkIfPass(String email, String pass, Activity activity, String username, String
confirmPass) {
        if(username.isEmpty()){
            requireView().showUsernameFieldError();
            requireView().showAllFieldError();
        }
        if(email.isEmpty()){
            requireView().showEmailFieldError();
            requireView().showAllFieldError();
        }
        if(pass.isEmpty()){
            requireView().showPassFieldError();
            requireView().showAllFieldError();
        }
        if(confirmPass.isEmpty()){
            requireView().showPassConfirmFieldError();
            requireView().showAllFieldError();
        }


if(registrationValidation.isEmailPass()&&registrationValidation.isPassPass()&&registrationValidat
ion.isUserNamePass()){
            requireModel().emailRegister(email, pass, activity);
        }


    }


    @Override
    boolean checkRxPattern(String text, String mask) {
        Pattern pattern;
        Matcher matcher;
```

```
    pattern = Pattern.compile(mask);
    matcher = pattern.matcher(text.trim());

    return matcher.matches();
}

@Override
void checkEmailValidation(CharSequence email) {
    if(!android.util.Patterns.EMAIL_ADDRESS.matcher(email).matches()){
        requireView().showEmailError();
        registrationValidation.setEmailPass(false);
    }
    else {
        requireView().hideErrorText();
        requireView().hideEmailErrorField();
        registrationValidation.setEmailPass(true);
    }
}

@Override
void checkUsername(String text) {
    if (!checkRxPattern(text, Keys.RegexMask.VALUE_MASK_USERNAME)){
        requireView().showUserError();
        registrationValidation.setUserNamePass(false);
    }
    else {
        registrationValidation.setUserNamePass(true);
        requireView().hideErrorText();
        requireView().hideUserErrorField();
    }
}

@Override
void checkPasswordValidation(String text) {
    if (!checkRxPattern(text, Keys.RegexMask.VALUE_MASK_PASSWORD)){
```

```
            requireView().showPassCharacterError();
        }
        else {
            requireView().hideErrorText();
            requireView().hidePasswordErrorField();
        }
    }

    @Override
    void checkPasswordMatches(String pass, String confirm) {
        if (!pass.equals(confirm)){
            requireView().showPassMatchError();
            registrationValidation.setPassPass(false);
        }
        else {
            requireView().hideErrorText();
            requireView().hidePasswordErrorField();
            registrationValidation.setPassPass(true);
        }
    }

    @Override
    void setupUsername(EditText username) {
        RxTextView.textChanges(username)
                .debounce(Keys.NumericDat.DEBOUNCE, TimeUnit.SECONDS)
                .observeOn(AndroidSchedulers.mainThread())
                .subscribe(textChanged -> {
                    checkUsername(username.getText().toString());
                });
    }

    @Override
    void setupEmail(EditText email) {
        RxTextView.textChanges(email)
                .debounce(Keys.NumericDat.DEBOUNCE, TimeUnit.SECONDS)
```

```
                .observeOn(AndroidSchedulers.mainThread())
                .subscribe(textChanged -> {
                    checkEmailValidation(email.getText().toString());
                });
    }

    @Override
    void setupPass(EditText pass) {
        RxTextView.textChanges(pass)
                .debounce(Keys.NumericDat.DEBOUNCE, TimeUnit.SECONDS)
                .observeOn(AndroidSchedulers.mainThread())
                .subscribe(textChanged -> {
                    checkPasswordValidation(pass.getText().toString());
                });
    }

    @Override
    void setupConfirmPass(EditText confirm, EditText pass) {
        RxTextView.textChanges(confirm)
                .debounce(Keys.NumericDat.DEBOUNCE, TimeUnit.SECONDS)
                .observeOn(AndroidSchedulers.mainThread())
                .subscribe(textChanged -> {
                    checkPasswordMatches(pass.getText().toString(),
confirm.getText().toString());
                });
    }

    @Override
    void showError(String text) {
        requireView().showError(text);
    }
    @Override
    void onSuccess() {
        requireView().showSuccess();
    }
}
```