АНОТАЦІЯ

Кваліфікаційна робота бакалавра за спеціальністю 121 – Інженерія програмного забезпечення. Тернопільський національний технічний університет імені Івана Пулюя, факультет комп’ютерно-інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СП-42, 2023 рік. . Пояснювальна записка до кваліфікаційної роботи на здобуття освітнього ступеню «бакалавр» містить: 60 с., 30 рис., 2 додатки, презентація.

Тема: Розробка інтерактивного додатку для подорожей з використанням Java та Android Studio.

В атестаційній роботі бакалавра висвітлено ключові моменти розробки Android додатку, створеного за допомогою мови програмування Java та середовища розробки Android Studio. В процесі було створено застосунок для подорожей з інтуїтивно зрозумілим інтерфейсом, основною метою якого є добавлення власних подорожей користувачами і перегляд вже списку існуючих. За допомогою Java, добавлено функціонал для взаємодії з користувачем. Створено активності що представляють сторінки екрану для зрозумілого пересування між розділами додатку.

Ключові слова: андроїд додаток, Java, Android Studio, Android, Nox, Activity, Android SDK, JVM, MVC.

ANNOTATION

Bachelor's qualification work in the specialty of 121 - Software Engineering. Ivan Puluj Ternopil National Technical University, Faculty of Computer and Information Systems and Software Engineering, Department of Software Engineering, SP-42 group, 2023. The explanatory note for the bachelor's qualification work includes: 60 pages, 30 figures, 2 appendices, presentation.

Topic: Development of an interactive travel application using Java and Android Studio.

The bachelor's qualification work highlights key aspects of developing an Android application created using the Java programming language and the Android Studio development environment. During the process, an application for travel was created with an intuitive interface, with the main goal being the addition of user-generated trips and the viewing of an existing list. Java was used to add functionality for user interaction. Activities representing screen pages were created to facilitate navigation between different sections of the application.

Keywords: Android application, Java, Android Studio, Android, Nox, Activity, Android SDK, JVM, MVC.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Android — це операційна система з відкритим кодом, яка працює на ядрі Linux. Випущена у 2003 році, зараз є найпоширенішою мобільною операційною системою у світі. Android – це, перш за все, проект з відкритим кодом під назвою AOSP (Android Open Source Project), який потім був використаним Google для розробки Android і повторно використовується іншими виробниками, такими як Samsung, LG та іншими. Талісманом Android є маленький зелений робот на ім'я Bugdroid або Andy.

Активність (Activity) - основна складова у фреймворку розробки Android, наданому Android Studio. Activity представляє собою окремий екран із користувацьким інтерфейсом, де користувачі можуть взаємодіяти.

Android SDK (Software Development Kit) - це набір інструментів, документації та ресурсів, який надається розробникам для створення мобільних додатків для операційної системи Android.

JVM (Java Virtual Machine) - віртуальна машина, що дозволяє виконувати програми, написані на мові програмування Java. Вона створює абстрактне середовище для запуску та виконання Java-коду незалежно від конкретної апаратної платформи. JVM виконує компіляцію вихідного коду Java в проміжний байт-код, який потім інтерпретується та виконується на специфічній операційній системі, дозволяючи розробникам писати один раз і виконувати програми на різних платформах.

MVC (Model-View-Controller) - популярна архітектурна концепція, що використовується в розробці додатків для платформи Android у середовищі Android Studio. Вона дозволяє розділити функціональність додатку на три основних компонента: модель (Model), представлення (View) та контролер (Controller).

ВСТУП

Сучасний світ переживає постійний розвиток технологій, що має безпосередній вплив на всі аспекти нашого життя, включаючи сферу подорожей. З появою смартфонів та мобільних додатків, шляхи планування та відкриття нових місць стали набагато доступнішими та зручнішими для кожного користувача. У цьому контексті розробка інтерактивних додатків для подорожей стає дедалі більш актуальною.

Метою даної роботи є розробка інтерактивного додатку для подорожей з використанням Java та Android Studio. Основною метою є створення зручного та функціонального інструменту, який дозволить користувачам додавати свої дописи про подорожі та переглядати рекомендовані і найочікуваніші. Всі додані подорожі, будуть збережені в базі даних і відображатися в профілі користувача у вигляді списку. Допис буде складатися з: заголовку, міста відправлення, міста прибуття, дистанції, фотографії та опису подорожі. Користувач матиме можливість взаємодіяти з кожним своїм доданим дописом, тобто його редагувати або змінювати. Для навігації по сайту буде розроблена навігаційна панель, що дасть можливість вільно пересуватися додатком. Вона буде складатися з кнопок: профілю користувача, добавити допис та перегляду рекомендованих дописів. Для того щоб ввійти в додаток буде розроблена форма реєстрації та входу. Не зареєстрований користувач не зможе добавляти свої подорожі і взаємодіяти з застосунком.

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Основні технології і мови програмування для створення додатків

У сьогоднішньому цифровому світі мобільні додатки виконують важливу роль у житті суспільства. Вони стали невід'ємною частиною повсякденного функціонування людей, надаючи їм доступ до різноманітних сервісів і можливостей.

Створення мобільних програм - це досить нова та швидкозростаюча сфера, що стала популярною завдяки поширенню смартфонів серед користувачів. В останні роки спостерігається значний ріст популярності мобільних програм, розроблених для операційних систем Android та iOS. Кожна з цих платформ має власну інфраструктуру розробки, яка впливає на процес створення програм[1].

Знизу перелічено різні підходи до розробки, що використовуються для створення таких програм:

1) Нативні додатки - це програми, розроблені для конкретної платформи (наприклад, iOS або Android) з використанням мов програмування та інструментарію, специфічного для цієї платформи. Вони розробляються з використанням нативного SDK (Software Development Kit), що надає розробникам доступ до функцій та можливостей операційної системи[2].

З переваг нативних додатків можна зазначити:

- продуктивність - нативні додатки мають безпосередній доступ до функцій та можливостей пристрою, що дозволяє досягти високої швидкодії та ефективності роботи;
- інтерфейс - розробники можуть використовувати специфічні елементи керування та інтерфейсу, що забезпечують нативний вигляд та поведінку додатка;
- доступ до пристроєвих функцій - нативні додатки мають повний доступ до функцій пристрою, таких як камера, геолокація, датчики, мікрофон та інші, що дозволяє створювати додатки з розширеними можливостями.

Розробка нативних застосунків є ефективним підходом для створення мобільних додатків з високою продуктивністю та нативним інтерфейсом. Використання мов програмування, специфічних для кожної платформи, дозволяє розробникам максимально використовувати можливості пристрою та забезпечує високу якість та надійність додатків.

2) Мобільні веб-додатки – додатки, що запускаються безпосередньо з будь-якого браузера, встановленого на пристрої. Оскільки веб-додатки розміщуються на сервері, необхідне активне підключення до Інтернету. Процес розробки аналогічний стандартному процесу розробки з використанням HTML, CSS і JavaScript як основних технологій. Коли відбуваються зміни, вони негайно набувають чинності і користувачі завжди використовують останню версію програми.Веб-додатки складаються з трьох основних компонентів:

- клієнтської частини;
- серверної частини;
- бази даних.

Клієнтська частина включає інтерфейс користувача, який відображається в браузері. Серверна частина обробляє запити користувачів та забезпечує доступ до необхідних ресурсів. База даних зберігає інформацію, яку можна використовувати в додатку. Основні технології для написання веб додатків які вже були згадані, включають в себе HTML (Hypertext Markup Language), CSS (Cascading Style Sheets) та JavaScript - складові, що забезпечують функціональність та взаємодію з користувачем.

HTML є основою веб-сторінок і застосунків, де створюється структура та розмітка контенту. Він дозволяє розміщувати текст, зображення, відео, посилання та інші елементи на веб-сторінці, надаючи їй візуальну структуру.

CSS використовується для стилізації та оформлення веб-додатків. За допомогою CSS можна змінювати кольори, розміри, шрифти та макети елементів на сторінці, надаючи їм привабливий та зручний зовнішній вигляд.

JavaScript дозволяє надати динамічність та функціональність веб-додаткам. За допомогою нього відбувається взаємодія з користувачем, оброблення подій,

перевірка даних, створення анімацій та виконання інших дій на стороні клієнта. JavaScript також дозволяє взаємодіяти з сервером за допомогою технології AJAX, що забезпечує асинхронний обмін даними без перезавантаження сторінки.

Окрім основних технологій, існує широкий вибір фреймворків та бібліотек, які полегшують розробку веб-додатків. Наприклад, React, Angular та Vue.js - це популярні фреймворки, що дозволяють створювати потужні та масштабовані додатки зі спрощеною розробкою компонентів.

3) Гібридні веб-додатки - використовують інструменти веб-розробки, вони не виконуються з браузера. Програми встановлюються на пристрої через внутрішній веб-контейнер, вони виконуються локально та мають доступ до деяких специфічних функцій пристрою через API. Гібридні програми пропонують великі переваги, оскільки вони дозволяють повторно використовувати код для різних платформ, доступ до апаратного забезпечення пристрою та розповсюдження через магазини програм.

Для розробки гібридних веб-додатків використовуються різні технології та фреймворки. Наприклад, Cordova (раніше відомий як PhoneGap) є однією з популярних технологій, що дозволяє упаковувати веб-додатки у нативні контейнери та отримувати доступ до функціональних можливостей пристрою. Іншим популярним фреймворком є React Native, який дозволяє розробляти гібридні додатки з використанням JavaScript та компонентів, що наближаються до нативних.

4) Кросплатформені додатки - оперують на різних платформах одночасно, забезпечуючи зручний інтерфейс програмування додатків (API) і можливість використовувати загальні кодові бази для різних платформ. При розробці кросплатформних додатків використовуються спеціальні технології, що орієнтовані на підтримку обох платформ. Два найперспективніші фреймворки в цьому напрямку - React Native, розроблений компанією Facebook, і Flutter - Google на базі мови програмування Dart.

Окрім основних переваги кросплатформних додатків - спроможність працювати на різних платформах одночасно, є наступні:

- зручний інтерфейс;
- повторне використання коду;
- дешевша ціна розробки;
- гнучкість використання функцій програмного забезпечення.

Важливо зазначити, що хоча кросплатформні додатки призначені для роботи на різних платформах, деякі компанії використовують їх і для розробки додатків для однієї платформи.

Отже, використання кросплатформних фреймворків у розробці мобільних додатків не обов'язково пов'язане з головною метою забезпечення доступності додатка на різних платформах. Рішення про використання таких фреймворків може бути прийняте з урахуванням інших факторів, таких як наявність технічних компетенцій в організації.

1.2 Обґрунтування вибору мови програмування Java

Java - універсальна, паралельна, об’єктно-орієнтована мова програмування, побудована на основі класів. Мова програмування має строгу типізацію. Це означає, що специфікація мови чітко розрізняє помилки, які можуть бути виявлені під час компіляції або під час виконання програми.

Під час виконання програми відбуваються такі дії, як завантаження та зв'язування класів, генерація машинного коду, динамічна оптимізація програми та фактичне виконання програми. Компіляція зазвичай включає перетворення програм в машинно-незалежне представлення, відоме як байт-код.

Байт-код є оптимізованим набором інструкцій, який виконується віртуальною машиною Java (JVM). JVM була розроблена як інтерпретатор байт-коду, оскільки багато сучасних мов компілюються у виконуваний код через проблеми з продуктивністю. Трансляція програми Java у байт-код робить запуск програми в різних середовищах легшим, оскільки потрібно створити лише JVM

для кожної платформи, що додає унікальності і відкриває безліч можливостей.

Мова Java також використовує певні обмеження для підвищення безпеки. Коли програма компілюється в проміжну форму і потім інтерпретується віртуальною машиною, вона працює трохи повільніше, ніж у випадку компіляції в виконуваний код. Однак високооптимізований байт-код дозволяє JVM виконувати програми швидше, ніж можна було б очікувати.

Незважаючи на те, що Java була розроблена як інтерпретована мова, вона може бути компільована в нативний код для покращення продуктивності. Тому компанія Sun використуває свою технологію HotSpot після першого випуску Java.

HotSpot, який входить до складу JVM, надає компілятор Just-In-Time (JIT) для байт-коду. Цей JIT-компілятор компілює вибрані частини байт-коду у виконуваний код у реальному часі, по частинах, в залежності від потреби. Варто зрозуміти, що компілювати весь код програми Java відразу в виконуваний код не є практично можливим, оскільки Java виконує різні перевірки, які відбуваються лише під час виконання програми. Замість цього, компілятор JIT компілює код по мірі необхідності під час виконання.

1.3 Обґрунтування вибору середовища розробки Android Studio

Android Studio - не єдиний спосіб розробки програм для Android, існують також інші IDE, такі як Eclipse та NetBeans. Розробити програму можна навіть використовуючи лише блокнот і командний рядок, але цей метод буде дуже повільним і громіздким.

Мною було вибрано Android Studio, тому що — це потужне та складне середовище, виготовлене спеціально для розробки, тестування та пакування додатків Android. Його можна завантажити разом із Android SDK як єдиний пакет, але також можливо встановити і оновлювати не залежно одне від одного. Створено з певною метою. Android Studio приваблює все більше сторонніх

плагінів, які надають великий набір цінних функцій, недоступних безпосередньо через IDE. До них входять плагіни для пришвидшення часу створення, налагодження проекту через Wi-Fi та багато іншого.

Android Studio пропонує ряд функціональних особливостей, які роблять його унікальним середовищем розробки, дозволяє розробникам швидко вносити зміни, надсилаючи код і сприяючи швидким змінам без перезапуску програми. Це забезпечує надзвичайну гнучкість для внесення невеликих змін у програму, яка на той момент все ще працює. Знизу наведено список переваг середовища розробки:

- швидкий і багатофункціональний емулятор - Android Studio має вбудований емулятор, задопомогою якого програми запускаються швидше, ніж на фізичних пристроях. Цей емулятор надає можливість тестувати програми на різних пристроях, таких як телефони, планшети, та інших пристроях розроблених задопомогою операційної системи Android. Крім того, емулятор може імітувати різні апаратні функції, включаючи GPS, різні сенсорні вводи, датчики руху, прискорення та багато інших, що додає унікальність та гнучкість при тестуванні програм;
- надійні механізми тестування - Android Studio пропонує широкий набір інструментів та фреймворків для тестування програм Android, зокрема функціональні інструменти для тестування користувацького інтерфейсу. Унікальність Android Studio виявляється у різноманітті покращених інструментів та фреймворків для тестування, які задовольнять будь-які потреби. Ці інструменти дозволяють проводити тести на реальних пристроях, емуляторах або надійних інтеграційних середовищах, а також використовувати Firebase Test Lab, що додає додаткову унікальність та можливості для тестування програм;
- підтримка Firebase та інтегрована хмара - Android Studio включає в себе Firebase Assistant, що дає змогу зручно підключати будь-яку програму до Firebase-серверу. Цей інструмент дозволяє не лише додати основні служби, такі як програмна аналітика, автентифікація та сповіщення, але й надає доступ до розмаїтих інших функцій. Крім того, Android Studio забезпечує

зручну інтеграцію програми з платформою Google Cloud, що розширює можливості розробки та розгортання програм на цій потужній хмарній платформі;

- потужна система розробки - кілька розширених функцій, які допомагають автоматизувати процеси, керувати залежностями та налаштовувати один раз конфігурацію. У проект можна включити як локальні, так і розміщені бібліотеки, є величезні можливості для налаштування проектів програми;

- підхід командної роботи - Android Studio надає вбудовані засоби для управління версіями, такі як GitHub і Subversion, що сприяють ефективному контролю над роботою команди та можливості вносити зміни у програму вчасно.

2 ПРОЄКТУВАННЯ ДОДАТКУ

2.1 Розробка моделі предметної області

Розробка моделі предметної області для додатку "Розробка інтерактивного додатку з використанням Java та Android Studio” - є важливим етапом проекту, що дозволяє чітко визначити основні сутності та їх взаємозв'язки. Ця модель визначає структуру та функціональні можливості додатку, допомагає розуміти, як користувачі будуть взаємодіяти з ним та які дані будуть зберігатися.

Основною метою даного додатку є надання користувачам можливості зберігання та організації інформації про їх подорожі в одному місці. Додаток надає зручний інтерфейс для додавання та керування подорожами, а також для перегляду інформації про уже здійснені та рекомендовані подорожі.

Модель предметної області цього додатку описує структуру та основні сутності, що використовуються для збереження та управління інформацією про подорожі користувачів. Основними сутностями моделі предметної області є: користувач, подорож, навігаційна панель, автентифікація, список подорожей та форма додавання подорожей.

Користувач: Представляє користувача додатку. Містить дані про обліковий запис користувача, такі як унікальний ідентифікатор, електронна пошта та пароль.

Подорож: Представляє окрему подорож. Містить інформацію про подорож, таку як заголовок, фото, місто відправлення, місто відвідання, дистанцію та опис. Кожна подорож пов'язана з конкретним користувачем, який її створив.

Навігаційна панель: Представляє нижню навігаційну панель додатку. Містить кнопки для переходу на різні сторінки додатку, такі як головна сторінка, сторінка додавання подорожей та сторінка перегляду подорожей.

Аутентифікація: Представляє процес реєстрації та входу в систему. Містить функції для створення облікових записів користувачів, перевірки введених даних та забезпечення безпеки.

Список подорожей: Представляє список подорожей, які належать конкретному користувачеві. Він містить посилання на окремі подорожі та надає можливість перегляду, редагування та видалення подорожей.

Форма додавання подорожі: Представляє форму, в якій користувач може ввести дані про нову подорож. Вона містить поля для введення заголовку, фото, міста відправлення, міста відвідання, дистанції та опису.

Щоб пройти аутентифікацію користувач повинен зареєструватися в додатку. Форма реєстрації є важливим елементом додатку і дозволяє користувачам створювати свої облікові записи для отримання повного доступу до всіх функціональних можливостей. Цей процес забезпечує ідентифікацію користувача та забезпечує безпеку та конфіденційність їх особистої інформації. Форма реєстрації має наступні поля:

- ім'я користувача - користувач вводить своє ім'я, яке буде використовуватися в додатку;

- електронна пошта - користувач вказує свою дійсну електронну пошту, яка буде використовуватися для входу в обліковий запис;

- пароль - користувач створює пароль для свого облікового запису, який повинен бути відповідати вимогам та містити мінімум символів;

- підтвердження пароля - користувач ще раз вводить пароль для його перевірки;

- кнопка "Зареєструватися" - користувач натискає цю кнопку для завершення процесу реєстрації. Виконується перевірка введених даних та створення облікового запису.

Після успішної реєстрації користувач направлений на форма входу, яка складається з наступних елементів:

- електронна пошта: Користувач вводить зареєстровану електронну пошту;

- пароль: Користувач вводить пароль, пов'язаний з обліковим записом;

- кнопка "Увійти": Користувач натискає цю кнопку для входу в свій обліковий запис. Виконується перевірка введених даних та перехід на сторінку профілю користувача у разі успішного процесу входу.

Профіль користувача складається з списку добавлених подорожей, або є пустим, якщо подорожі ще не добавлено. Список подорожей містить в собі окремі пости про подорожі, здійснені користувачем. Подорожі представляють окремі поїздки зі своїми характеристиками, такими як:

- фото;
- заголовок;
- місто відправлення;
- місто відвідування;
- дистанція;
- опис.

Кожна подорож користувача є унікальною, ідентифікатором подорожі служить ідентифікаційний номер. Цей номер є унікальним і не повторюється. Також неявним ідентифікатором подорожі є заголовок. Читаючи заголовок користувач може легко здогадатися про яку саме подорож йдеться. Наприклад: “Підземелля Тернополя”, “Кінбурнська коса”, “Український Хогвартс”. Відвідані міста, фото, дистанція можуть повторюватися необмежену кількість разів.

Навігаційна панель додатку є важливою частиною інтерфейсу, яка забезпечує зручну навігацію між різними розділами та функціями, тому розміщена внизу екрану і складається з трьох кнопок:

1) Головна сторінка: Ця кнопка відображається як початкова сторінка додатку після входу користувача. При натисканні на неї користувач переходить до головної сторінки, де він бачить список своїх доданих подорожей.

2) Додати подорож: Ця кнопка дозволяє користувачу додавати нову подорож. При натисканні на неї відкривається форма, в якій користувач може ввести інформацію про подорож, а саме, фото, заголовок, місто відправлення, місто відвідання, дистанцію та опис. Після заповнення форми користувач може зберегти подорож і перейти назад до головної сторінки.

3) Переглянути рекомендовані подорожі: Ця кнопка дозволяє користувачу відкрити нове вікно, зі списком рекомендованих та найочікуваніших подорожей, містами України.

Навігаційна панель забезпечує зручну навігацію між розділами додатку та дозволяє користувачу швидко переходити до потрібних функцій. Вона створює зрозумілу структуру додатку і полегшує користування ним.

Ще одним важливим елементом інтерфейсу є верхня панель додатку, яка розташована вгорі екрану. Вона містить інформацію та функції, пов'язані з обліковим записом користувача.

1. Кнопка "Вийти" - розташована з лівого боку верхньої панелі і служить для виходу з облікового запису користувача. При натисканні на неї користувач зможе вийти з акаунту і повернеться на сторінку входу.
2. Логін користувача - розташований з правого боку верхньої панелі розташований логін користувача або його ім'я, що використовується для позначення користувача в додатку. Це може бути його ім'я або псевдонім.
3. Картинка користувача – розташована з правої сторони від логіну.

Верхня панель додатку забезпечує швидкий доступ до функції виходу з облікового запису та відображення імені або логіну користувача. Вона допомагає забезпечити персоналізований досвід користування та зручність управління обліковим записом.

2.2. Розробка бізнес моделі

Бізнес-модель є важливим моментом розробки проєкту. Вона описує, яку цінність буде створювати та отримувати додаток для користувачів і бізнесу, основну аудиторію додатку і його монетизацію.

Додаток спрямований на широкий сегмент цільової аудиторії. Основний сегмент людей, на яких спрямований додаток це люди які мають інтерес до подорожей, та бажають зберегти свої спогади та досвід про поїздки. Нижче наведено можливі сегменти цільової аудиторії:

- молоді мандрівники любителі;
- професійні мандрівники;
- туристи та паломники;
- люди, що планують подорож.

Для молодих людей, які активно шукають нові враження і місця для відвідування, додаток буде цікавий і приверне їх увагу. Нові найочікуваніші подорожі будуть розташовані в спеціальному розділі, чим і зможуть зацікавити молодих ентузіастів.

Професійні мандрівники, тобто люди які багато подорожують і це їх основний вид діяльності, можуть використовувати додаток для зручного та легкого способу зберігання своїх подорожей і відвіданих місць.

Туристів та паломників додаток приверне увагу, можливістю зберігати святині та туристичні атракції в яких вони побували. Вони можуть бути зацікавлені в плануванні наступних подорожей.

Для людей що планують подорожі, додаток приверне увагу через великий вибір рекомендованих або найочікуваніших подорожей які є актуальні на даний час.

Основна цінність додатку полягає в наданні користувачам зручного та функціонального інструменту для організації та зберігання інформації про їх подорожі. Додаток надає можливість користувачам створювати та свої подорожі в зручній формі. Вони можуть вказувати місця відправлення та призначення, дистанцію, опис та інші деталі, що стосуються подорожі. Такий підхід робить легкий доступ до інформації про подорожі та їх систематизації.

Додаток надає користувачам зручну навігаційну панель, яка дозволяє легко переміщатися між різними розділами додатку. Візуальне представлення подорожей з фотографіями та додатковою інформацією, описом допомагає користувачам легко орієнтуватися та знаходити потрібну інформацію. Також додаток дозволяє користувачам зберігати спогади про подорож в цифровій формі і знижує ризики втрати цієї інформації коли-небудь. Це робить можливим зберегти емоції та повертатися до перегляду відвіданих міст у майбутньому.

Додаток надає користувачам актуальну інформацію про найочікуваніші та рекомендовані подорожі.

Отже цінність додатку полягає в зберігання інформації про подорожі, та перегляді нових подорожей, а також наданні зручного та візуально привабливого інтерфейсу для користувачів.

Заробіток додатку може складатися з кількох моделей монетизації. Знизу наведені деякі способи заробітку:

- реклама;
- внутрішні купівлі;
- партнерські програми;
- платна підписка.

В додатку може бути розміщена реклама, рекламні банери, тексти або відео під час перемикання розділів або перегляду вмісту. За кліки по рекламі або її відображення, розробник може отримувати плату від рекламних мереж або прямих рекламодавців.

Додаток може мати можливість внутрішньої купівлі, де користувачі можуть купувати додаткові функції, розширення або вміст в додатку. Наприклад, користувачі зможуть придбати додаткові шаблони для створення постів, або фільтри для фотографій.

Додаток може підписати партнерську програму з туристичними агенціями або певними організаціями, яка буде відповідати вимогам додатку. Компанія підпише контракт з розробником, в якому будуть описані опиши умови та терміни співпраці. Це може реклама подорожей, або їх продаж через додаток, за що розробник отримує виплату, або комісію.

Платна підписка в додатком, може містити розширення функціоналу, та добавлення певних опцій, що дозволить користувачу використовувати ці преміум-послуги.

Отже, модель монетизації в першу чергу повинна відповідати цілям додатку. Спрямування монетизації на розширення можливостей та потреб користувачів, буде нести позитивні наслідки.

Наведено діаграму варіантів використання для актора “Користувач”(рис. 2.1).

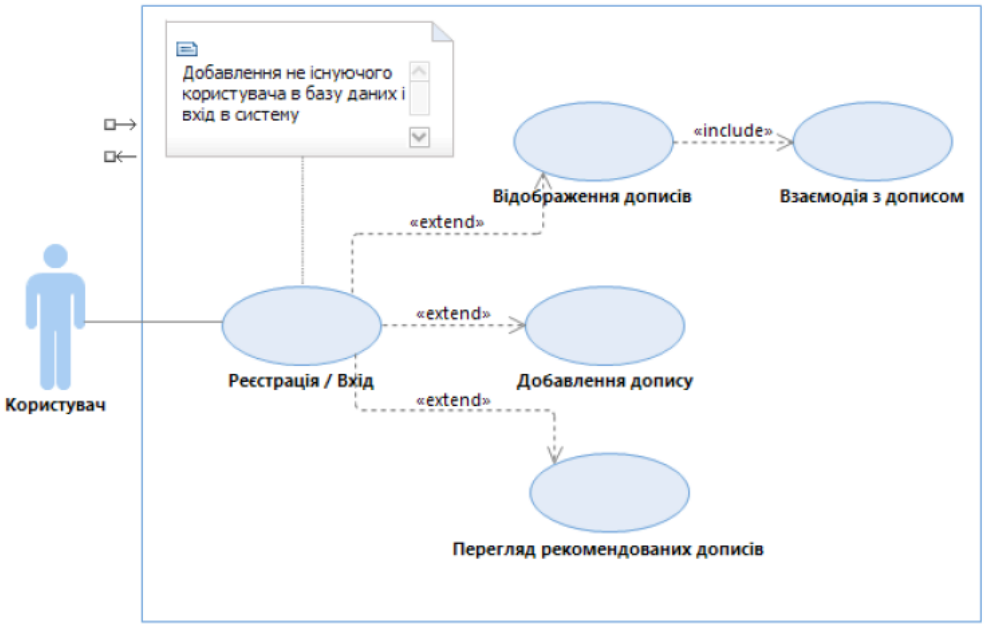

Рисунок 2.1 - Діаграма варіантів використання для актора “Користувач”

Опис варіантів використання:

Варіант використання “Реєстрація / Вхід”

Короткий опис:

Цей варіант використання дозволяє створити новий обліковий запис користувачу, для входу в додаток, використання його функціоналу і взаємодії з ним.

Основний потік подій:

1. Користувач відкриває додаток;
2. Вибирає кнопку “Зареєструватися”, або якщо має акаунт - “Увійти”;
3. В запропонованій формі, вводить персональні дані;
4. Натискає кнопку “Зареєструватися” або “Увійти”;
5. Заходить в додаток.

Альтернативний потік подій:

1. Користувач вводить неправильні персональні дані;

2. З'являється повідомлення про помилку при введенні даних;
3. Користувач продовжує заповнювати форму.

Передумови:

Користувач має створений акаунт, додаток працює стабільно.

Післяумови:

При виконанні цього варіанту користувач заходить в свій профіль, отримує доступ до функціоналу додатку.

Варіант використання “Відображення дописів”

Короткий опис:

Цей варіант використання дозволяє відобразити дописи про подорожі, після успішного входу в додаток.

Основний потік подій:

1. Користувач успішно заходить в додаток;
2. В навігаційній панелі вибирає розділ профіль;
3. Користувачу відображається список дописів.

Альтернативний потік подій:

1. Користувач зайшов в додаток, але втратив втратив підключення до інтернету;
2. Список дописів не відображається.

Передумови:

Додаток працює стабільно, наявне підключення до інтернету.

Післяумови:

Можливість переглядати дописи про подорожі.

Варіант використання “Взаємодія з дописом”

Короткий опис:

Цей варіант використання дозволяє взаємодіяти з дописом, а саме його редагувати або видаляти.

Основний потік подій:

1. Користувач вибирає допис зі списку;
2. Користувач натискає кнопку “Редагувати”;

3. Відкривається форма для редагування допису;
4. Користувач вводить зміни;
5. Користувач натискає кнопку підтвердити;
6. Допис відображається з добавленими змінами;
7. Користувач натискає кнопку “Видалити”;
8. Користувач підтверджує видалення допису;
9. Допис видалений зі списку.

Альтернативний потік подій:

1. Допис користувача не змінюється, через відсутнє підключення до інтернету.

Передумови:

Додаток працює стабільно, наявне підключення до інтернету.

Післяумови:

Можливість переглянути допис зі змінами, або його відсутність в списку.

Варіант використання “Добавлення допису про подорожі”

Короткий опис:

Цей варіант використання дозволяє добавити допис про подорож до списку подорожей.

Основний потік подій:

1. Користувач успішно заходить в додаток;
2. В навігаційній панелі вибирає розділ добавити допис;
3. Заповнює форму інформацією про подорож, а саме, назву подорожі, місто відправлення, місто відвідання, дистанцію, опис та вибирає фото;
4. Користувач натискає кнопку підтвердити;
5. Допис про подорожі успішно добавляється до списку подорожей користувача.

Альтернативний потік подій:

1. Користувач зайшов в додаток, але підключення до інтернету відсутнє;
2. В навігаційній панелі вибирає розділ “Добавити допис”;

3. Заповнює відповідну форму;
4. Натискає кнопку підтвердити;
5. Допис про подорож не створюється, через відсутнє з'єднання з інтернетом.

Передумови:

Додаток працює стабільно, наявне підключення до інтернету.

Післяумови:

Можливість переглянути новий щойно добавлений допис про подорожі.

Варіант використання “Перегляд рекомендованих дописів”

Короткий опис:

Цей варіант використання дозволяє переглянути рекомендовані дописи

Основний потік подій:

1. Користувач успішно заходить в додаток;
2. В навігаційній панелі вибирає розділ “Рекомендовані дописи”;
3. Переглядає дописи про рекомендовані подорожі;
4. Переглядає дописи про найочікуваніші подорожі;

Альтернативний потік подій:

1. Користувач зайшов в додаток, але підключення до інтернету відсутнє;
2. Розділ “Рекомендовані дописи” не загружається;

Передумови:

Додаток працює стабільно, наявне підключення до інтернету.

Післяумови:

Можливість переглянути допис про рекомендовані та найочікуваніші подорожі.

Описана вище діаграма варіантів використання ілюструє та відображає функціональність та взаємодію між актором (користувачем) та системою у рамках додатку. Описані сценарії діаграми варіантів використання дозволяють детально ознайомитися з послідовністю подій, які відбуваються між актором та системою під час взаємодії.

2.3 Проєктування архітектури

Проєктування архітектури є важливим етапом, для створенння будь якого програмного-проєкту. Проєктування визначає фундамент та структуру додатку, а також забезпечує його ефективність та масштабованість.

У контексті мого дипломного проекту, якому присвячена дана робота, метою є розробка інтерактивного додатку з використанням Java та Android Studio. Цей додаток надасть користувачам можливість додавати свої дописи про подорожі, що включатимуть заголовок, фото, місто відправлення, місто відвідування, дистанцію та опис. Крім того, користувачі матимуть змогу переглядати актуальні подорожі.

Дизайн клієнт-серверної архітектури програми для Android зазвичай включає клієнтську програму на пристрої Android, яка спілкується із сервером через Інтернет. Клієнтська програма надсилає запити на сервер, який обробляє запити та надсилає відповідь. Ця архітектура дозволяє розділити проблеми між клієнтом і сервером і надає програмі доступ до даних і функцій, які можуть бути недоступні на самому пристрої.

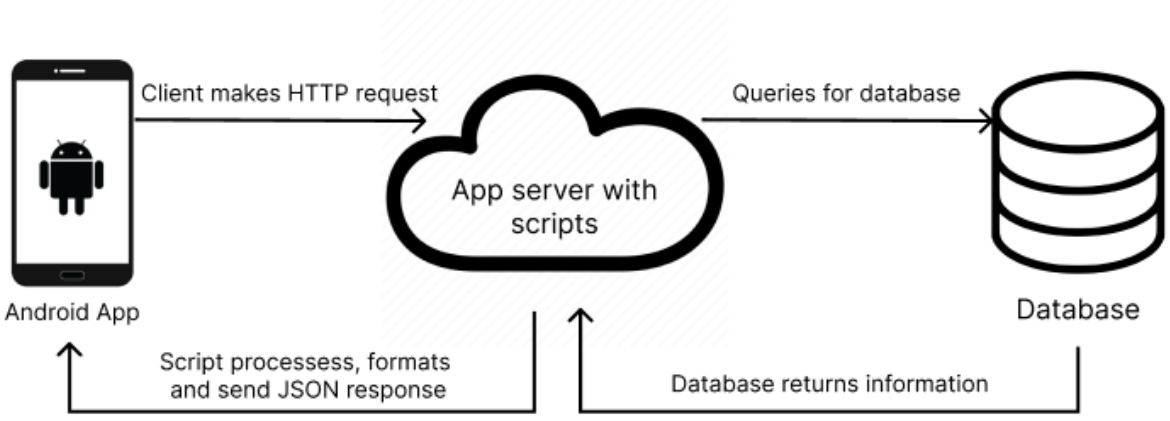

Рисунок 2.2 - Клієнт-серверна модель для андроїд додатків

Наведено приклад клієнт-серверної моделі для андроїд додатків(рис. 2.2).

При проектуванні загальної архітектурної моделі використано MVC (Model-View-Controller) підхід(рис. 2.3)

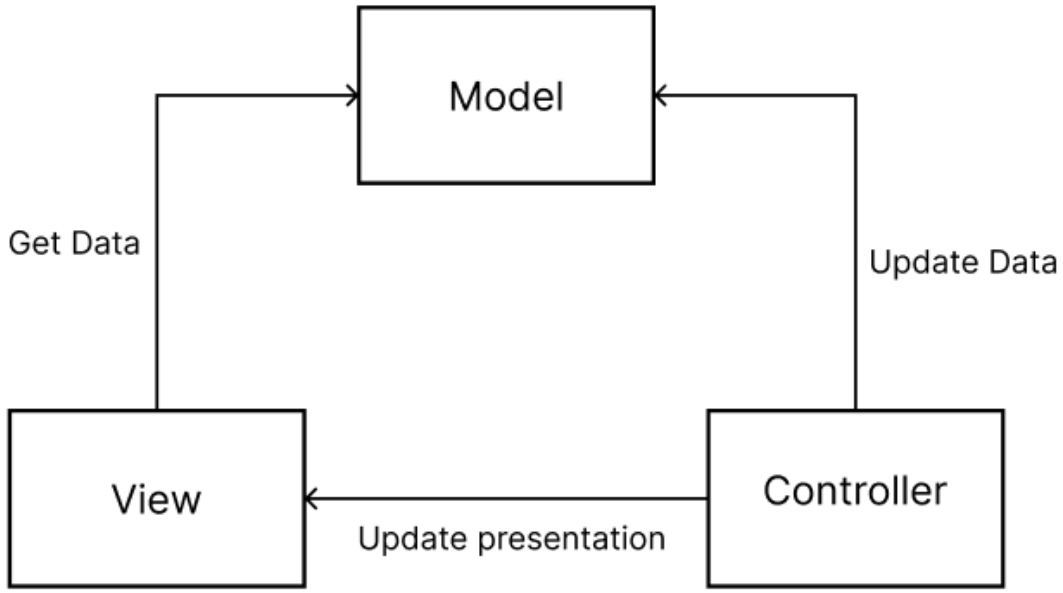

Рисунок 2.3 - Принцип роботи MVC підходу

1. View - відповідає за представлення даних користувачу та взаємодію з ним. Відображає інформацію, яку отримує від моделі, та реагує на взаємодію користувача.
2. Model - представляє дані та бізнес-логіку додатку. Вона отримує, зберігає та маніпулює даними, а також надає методи для доступу до цих даних.
3. Controller - обробляє взаємодію користувача та виконує дії відповідно до цієї взаємодії. Він приймає вхідні дані від користувача, оновлює модель та відправляє оновлені дані до вигляду.

Основним принципом цього архітектурним шаблону є розділення ролей та відповідальностей між моделлю, видом та контролером. Він допомагає організувати логіку, відображення та взаємодію компонентів програми, забезпечуючи розділення обов'язків і полегшуючи підтримку та розширення додатків. Також дозволяє забезпечити гнучкість і розширюваність додатків, оскільки зміни в одному компоненті не впливають на інші.

Наведено діаграму класів додатку(рис. 2.4). Основною метою діаграми класів є представлення загального огляду класів, їх атрибутів та методів, а також визначення взаємозв'язків між класами.

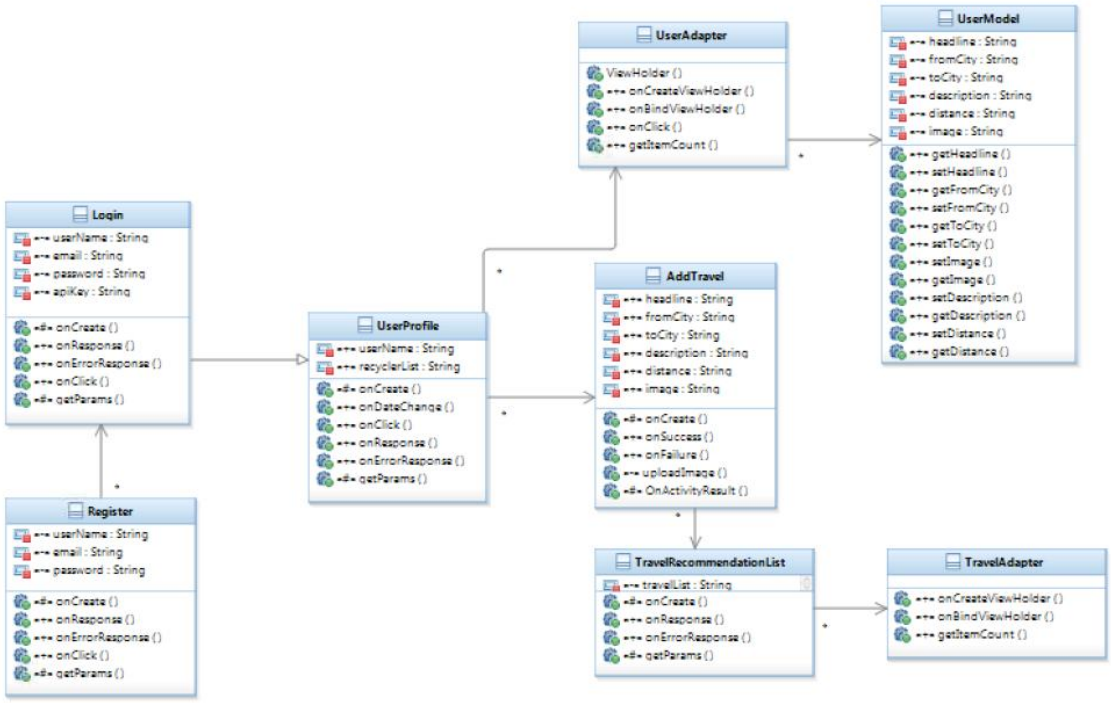

Рисунок 2.4 - Діаграма класів

На основі діаграми, наведено список основних класів:

- Login відповідає за вхід користувача в систему;

- Register відповідає за реєстрацію користувача в системі;

- UserProfile відповідає профіль користувача, відображення списку його подорожей;

- UserAdapter відповідає за відображення списку дописів про подорожі користувача і взаємодію з ними;

- UserModel відповідає за присвоєння значень змінним, кожного допису про подорож;

- AddTravel відповідає за добавлення нового допису про подорож;

- TravelRecommendationList відповідає за відображення рекомендованого списку дописів про подорожі;

- TravelAdapter відповідає за відображення спику рекомендованих дописів про подорожі.

Список основних методів класів:

- onCreate() є метод життєвого циклу активності (Activity) в Android Studio. Викликається під час створення активності і використовується для початкового налаштування інтерфейсу користувача;

- onClick є методом обробки подій кліків на елементах інтерфейсу Android додатку. Він є частиною підхіду Model-View-Controller (MVC) у розробці Android додатків;

- onResponse () є методом зворотного виклику, який автоматично викликається при отриманні відповіді на асинхронний запит до сервера;

- onErrorResponse () є методом зворотного виклику, який автоматично викликається при отриманні помилки у відповіді, після асинхронного запиту на сервер;

- getParams () є методом, який використовується для отримання параметрів запиту в контексті мережевих викликів на сервер у форматі ключ-значення, дозволяє вказати параметри запиту, такі як заголовки, параметри URL-адреси.

- ViewHolder () є методом в архітектурі спискових адаптерів для збереження посилань на елементи інтерфейсу в кожному елементі списку, і уникати повторного звернення до них під час прокручування списку.;

- onBindViewHolder() є методом для зв'язування даних з елементами списку. Він викликається для кожного елемента, який потребує оновлення даних;

- uploadImage () є методом, який використовується для завантаження зображення з пристрою користувача на сервер або інше мережове сховище;

- onSuccess () є методом використання при роботі з асинхронними операціями для обробки успішної операції або відповіді під час виконання певної операції;

- onFailure () - є також асинхронним методом, для обробки не успішного результату або відповіді під час виконання певної операції.

Наведено діаграму послідовностей входу користувача в систему(рис. 2.5).

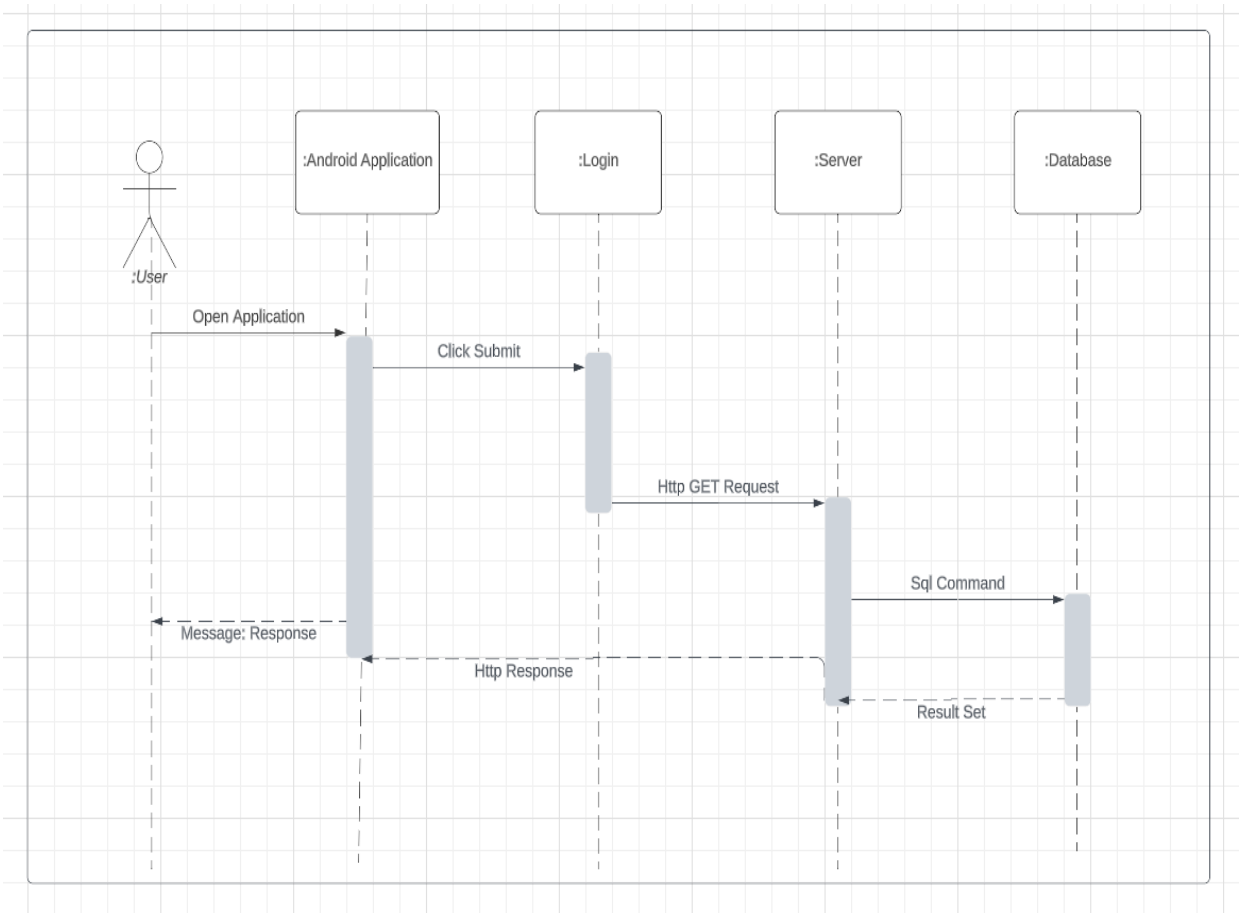

Рисунок 2.5 - Діаграма послідовностей для входу в систему

На діаграмі зображено 5 об’єктів:

- User;
- Android Application;
- Login;
- Server;
- Database.

Дії, що зображені на діаграмі послідовностей:

- користувач заходить в додаток;
- натиснення кнопки ввійти;
- надсилання запиту GET на сервер;
- надсилання сервером Sql запиту в базу даних;
- користувач отримує відповідь.

Наведено діаграму послідовностей для добавлення нового допису(рис. 2.6).

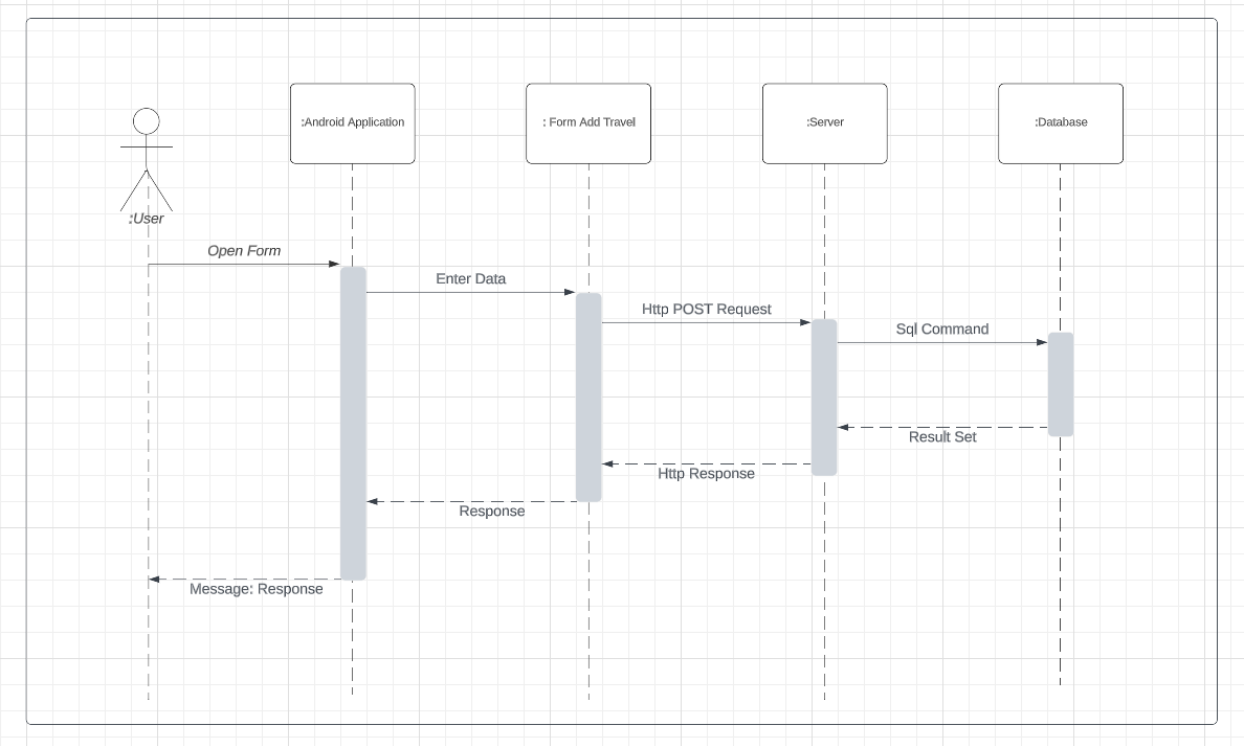

Рисунок 2.6 - Діаграма послідовностей для добавлення нового допису

На діаграмі зображено 5 об’єктів:

- User;
- Android Application;
- Add Travel Form;
- Server;
- Database.

Дії, що зображені на діаграмі послідовностей:

- Користувач відкриває форму для добавлення нового допису;
- Надсилання заповненої форми користувачем;
- Надсилання запиту POST на сервер;
- Надсилання сервером Sql запиту в базу даних;
- Користувач отримує відповідь;

3. КОНСТРОЮВАННЯ ДОДАТКУ

3.1 Реалізація ключових класів

У цьому розділі буде розглянуто детальну реалізацію ключових класів, які використовуються в додатку. Будуть описані класи, відповідальні за збереження та обробку даних про подорожі, а також класи для взаємодії з користувачем, включаючи форми вводу та відображення інформації.

Клас Login відповідає за вхід користувача в додаток. Він містить необхідні методи та функціонал, для перевірки введених даних в полях форми та авторизації користувача. Також в класі оголошуються змінні, які потім використовуються в методах класу. Для збереження користувачів створено базу даних MySql. Наведемо основні елементи класу Login:

Метод onCreate() - є важливим методом життєвого циклу в активності (Activity) в Android(рис. 3.1). Він викликається автоматично, коли створюється новий екземпляр активності. В методі зазвичай описаний весь функціонал класу. Основна функція методу, це налаштування графічного інтерфейсу користувача(GUI) активності. Де описуються початкові значення змінних, встановлюється макет активності, який визначає позицію і вигляд елементів інтерфейсу.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_registration);
```

Рисунок 3.1 – Код методу onCreate()

Для входу користувача в систему, використано бібліотеку Http запитів Volley. Бібліотека надсилає запити в базу даних і обробляє відповіді.

Для переходу в форму реєстрації, користувач натискає на посилання

“textViewRegisterNow”, в якого є подія натискання(рис. 3.2). Після натискання відображається сторінка реєстрації з класом Register.

```
// Register Page
textViewRegisterNow.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(getApplicationContext(), Registration.class);
        startActivity(intent);
        finish();
    }
});
```

Рисунок 3.2 – Код посилання на форму реєстрації

Клас UserProfile відповідає за відображення сторінки користувача з його дописами. Він містить функціонал для відображення списку дописів, навігаційну панель, верхню панель керування. Для збереження і завантаження списку дописів використовується база даних Firebase. Наведемо основні елементи класу UserProfile:

Для збереження і розпізнавання стану користувача використано інтерфейс SharedPreferences (рис. 3.3).

```
sharedPreferences = getSharedPreferences( name: "MyAppName", MODE_PRIVATE);
// Checking if the user is authenticated
if(sharedPreferences.getString( key: "logged",  defValue: "false").equals("true")){
    Intent intent = new Intent(getApplicationContext(), UserProfile.class);
    startActivity(intent);
    finish();
}
```

Рисунок 3.3 – Код інтерфейсу SharedPreferences

Наведемо програмну реалізацію відображення списку дописів з бази даних Firebase:

```
firebaseDatabase = FirebaseDatabase.getInstance();
firebaseDatabase.getReference().child("post").addListenerForSingleValueEvent
(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot snapshot) {
        for(DataSnapshot dataSnapshot : snapshot.getChildren()){
            UserModel userModel = dataSnapshot.getValue(UserModel.class);
            recyclerList.add(userModel);
```

```
            }
            recyclerAdapter.notifyDataSetChanged();
        }
    });
```

Програмна реалізація навігаційної панелі (рис. 3.3).

```
// Bottom Navigation items
bottomNavigationView.setOnItemSelectedListener(item -> {
    switch(item.getItemId()){
        case R.id.bottom_profile:
            return true;
        case R.id.bottom_add:
            startActivity(new Intent(getApplicationContext(), AddTravel.class));
            overridePendingTransition(R.anim.slide_in_right, R.anim.slide_out_left);
            finish();
            return true;
        case R.id.bottom_travel_list:
            startActivity(new Intent(getApplicationContext(), TravelRecommendationList.class));
            overridePendingTransition(R.anim.slide_in_right, R.anim.slide_out_left);
            finish();
            return true;
    }
    return false;
});
```

Рисунок 3.3 – Код навігаційної панелі

Клас AddTravel відповідає за добавлення нового допису подорожі. Він містить форму для заповнення, яка складається з: заголовка, міста відправлення, міста прибуття, дистанції, опису і фото, та кнопки надсилання. Кнопка відповідає за відправлення заповненої форми користувачем на сервер, та добавлення нового допису в базу даних.

```
// Upload Image Method
private void uploadImage() {
    Dexter.withContext(this) DexterBuilder.Permission
            .withPermission(Manifest.permission.READ_EXTERNAL_STORAGE) DexterBuilder.SinglePermissionListener
            .withListener(new PermissionListener() {
                @Override
                public void onPermissionGranted(PermissionGrantedResponse permissionGrantedResponse) {
                    Intent intent = new Intent();
                    intent.setType("image/*");
                    intent.setAction(Intent.ACTION_GET_CONTENT);
                    startActivityForResult(intent,  requestCode: 101);
                }
                @Override
                public void onPermissionDenied(PermissionDeniedResponse permissionDeniedResponse) {
                    Toast.makeText( context: AddTravel.this,  text: "Permission Denied", Toast.LENGTH_SHORT);
                }
                @Override
                public void onPermissionRationaleShouldBeShown(PermissionRequest permissionRequest, PermissionToken permissionToken) {
                    permissionToken.continuePermissionRequest();
                }
            }).check();
}
```

Рисунок 3.4 – Код методу завантаження фото допису

Програмна реалізація завантаження фото допису користувачем (рис. 3.4).

Програмна реалізація функціоналу добавлення нового допису в базу даних (рис. 3.5).

```
database.getReference().child( pathString: "post").child(postId).setValue(model)
        .addOnSuccessListener(new OnSuccessListener<Void>() {
            @Override
            public void onSuccess(Void unused) {
                Toast.makeText( context: AddTravel.this,  text: "Post upload Successfully"
                        +database.getReference( path: "Courses"), Toast.LENGTH_SHORT).show();
                dialog.dismiss();
                Intent intent = new Intent(getApplicationContext(), UserProfile.class);
                startActivity(intent);
                finish();
            }
        }).addOnFailureListener(new OnFailureListener() {
            @Override
            public void onFailure(@NonNull Exception e) {
                dialog.dismiss();
                Toast.makeText( context: AddTravel.this,  text: "Error Occured", Toast.LENGTH_SHORT).show();
            }
        });
```

Рисунок 3.5 – Код добавлення нового допису

Клас UserAdapter відповідає за відображення і взаємодію з дописами. В класі описані методи і класи, що дозволяють користувачу редагувати і видалити обраний допис.

Клас ViewHolder, що відповідає за зберігання посилань на візуальні елементи, що відображають дані в кожному елементі списку (рис. 3.6).

```
static public class ViewHolder extends RecyclerView.ViewHolder{
    TextView itemHeadLine, itemCityFrom, itemCityTo, itemDescription, itemDistance;
    ImageView itemImage;
    Button btnEdit, btnDelete;
    public ViewHolder(@NonNull View itemView) {
        super(itemView);
        // Initialization of post parameters
        itemHeadLine = itemView.findViewById(R.id.itemHeadLine);
        itemCityFrom = itemView.findViewById(R.id.itemCityFrom);
        itemCityTo = itemView.findViewById(R.id.itemCityTo);
        itemDescription = itemView.findViewById(R.id.itemDescription);
        itemDistance = itemView.findViewById(R.id.itemDistance);
        itemImage = itemView.findViewById(R.id.itemImage);
        btnEdit = itemView.findViewById(R.id.btnEdit);
        btnDelete = itemView.findViewById(R.id.btnDelete);
    }
}
```

Рисунок 3.6 – Код добавлення нового допису

Наведено програмну реалізацію методу onCreateViewHolder(), що відповідає

за створення нового екземпляра ViewHolder:

```
public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int
viewType) {
    View view = LayoutInflater.from(parent.getContext())
.inflate(R.layout.activity_user_item, parent, false);
    return new ViewHolder(view);
}
```

Метод onBindViewHolder(), що відповідає за кожен допис в списку, та взаємодію з ним (рис. 3.7).

```
// Selected Element
@Override
public void onBindViewHolder(@NonNull ViewHolder holder, int position) {
    // Selected post
    UserModel model = list.get(position);
    // Selected post id
    String postId = list.get(position).getTravelId();

    Picasso.get().load(model.getTravelImage()).placeholder(R.drawable.travelphoto).into(holder.itemImage);
    holder.itemHeadLine.setText(model.getHeadline());
    holder.itemCityFrom.setText(model.getFromcity());
    holder.itemCityTo.setText(model.getTocity());
    holder.itemDistance.setText(model.getDistance());
    holder.itemDescription.setText(model.getDescription());
}
```

Рисунок 3.7 – Код методу onBindViewHolder()

3.2 Розробка GUI додатку

У цьому розділі буде описано процес розробки графічного інтерфейсу користувача, представлено фото інтерфейсу додатку. Буде детально пояснено, які компоненти і елементи використовуються для відображення інформації дописів про подорожі.

Для розробки графічного інтерфейсу в Android Studio використано інструмент дизайну Layout Editor. Він дозволяє візуально проектувати та редагувати графічний інтерфейс додатків, створювати та розташовувати елементи інтерфейсу, встановлювати їх розміри, кольори та інші атрибути. За допомогою Layout Editor створено макети у форматі XML.

Форма входу(Login) забезпечує безпечний спосіб авторизації користувачів в додатку. Графічний інтерфейс форми входу складається з полів для введення електронної пошти, паролю та кнопки "Увійти”. Під формою розташоване посилання на форму реєстрації (рис. 3.8). Створено поля введення, за допомогою елементу інтерфейсу “<EditText/>”. Для текстових полів використано елемент інтерфейсу “<TextView/>”. Кнопка надсилання була створена за допомогою “<Button/>”.

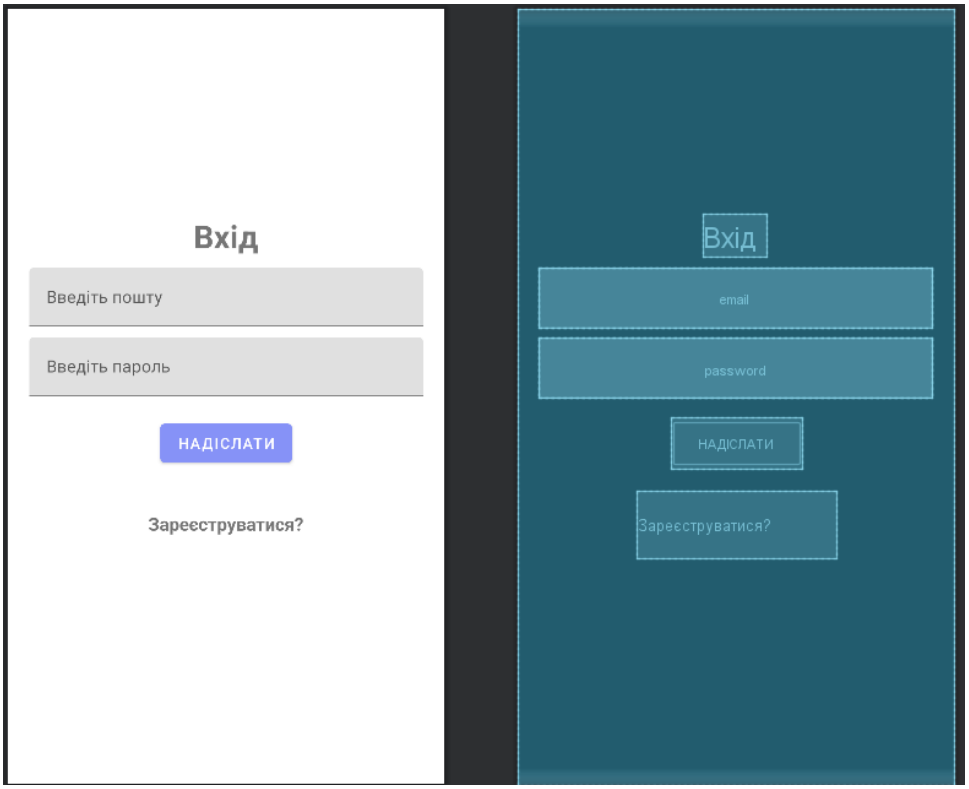

Рисунок 3.8 - Графічний інтерфейс форми входу

Наведено програмну реалізацію відображення графічного інтерфейсу поля email форми входу:

```
<com.google.android.material.textfield.TextInputLayout
  android:layout_width="match_parent"
  android:layout_marginStart="20dp"
  android:layout_marginEnd="20dp"
  android:layout_marginBottom="10dp"
  android:layout_height="wrap_content">
<com.google.android.material.textfield.TextInputEditText
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:id="@+id/email"
      android:hint="@string/enter_email"/>
</com.google.android.material.textfield.TextInputLayout>
```

Графічний інтерфейс профілю користувача складається зі списку дописів про подорож, нижньої навігаційної панелі та верхньої панелі керування (рис. 3.9). Кожний елемент є окремою частиною коду. Відображення списку дописів про подорож відбувається за допомогою елемента “<RecyclerView/>”.

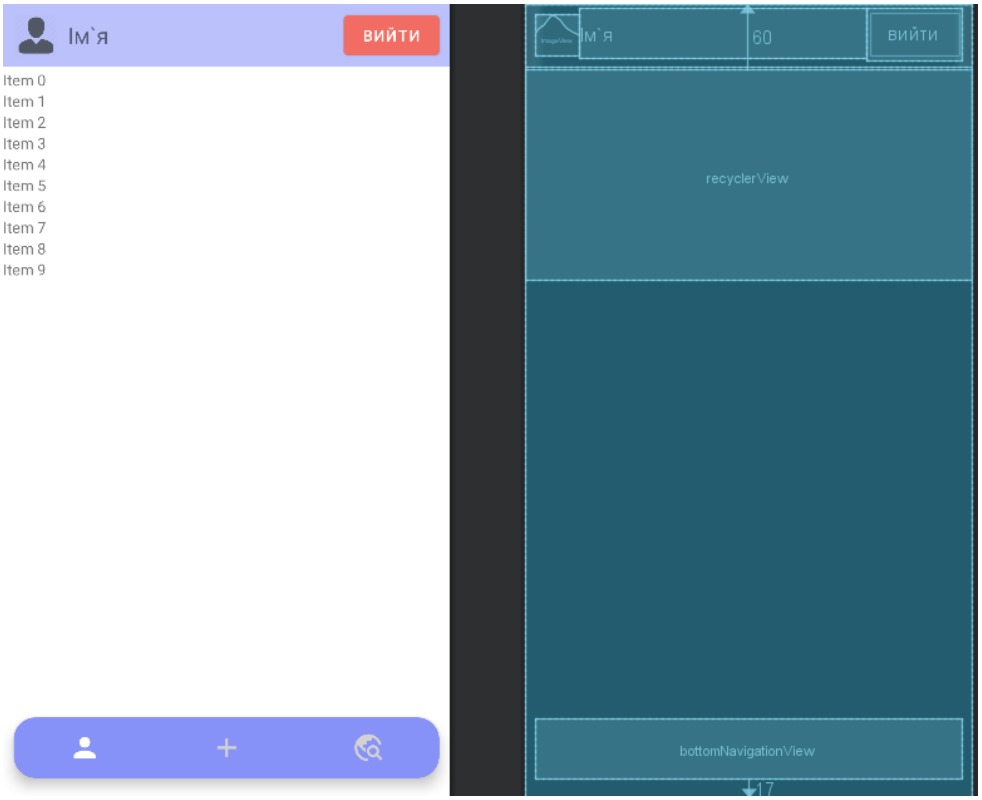

Рисунок 3.9 – Графічний інтерфейс профілю користувача

Наведемо програмну реалізацію відображення графічного інтерфейсу списку дописів:

```
<androidx.recyclerview.widget.RecyclerView
 android:id="@+id/recyclerView"
 android:layout_width="match_parent"
 android:layout_alignParentTop="true"
 android:layout_marginTop="60dp"
 android:layout_marginBottom="75dp"
 android:layout_height="wrap_content" />
```

Навігаційна панель відображається за допомогою елементу “<BottomNavigationView/>.

Наведемо програмну реалізацію відображення графічного інтерфейсу нижньої навігаційної панелі:

```
<com.google.android.material.bottomnavigation.BottomNavigationView
 android:id="@+id/bottomNavigationView"
```

```
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_alignParentBottom="true"
android:layout_marginStart="10dp"
android:layout_marginTop="30dp"
android:layout_marginEnd="10dp"
android:layout_marginBottom="17dp"
android:background="@drawable/bottom_background"
android:elevation="2dp"
   app:itemIconSize="30dp"
   app:itemIconTint="@drawable/item_selector"
   app:itemRippleColor="@android:color/transparent"
   app:labelVisibilityMode="unlabeled"
   app:menu="@menu/bottom_menu" />
```

Відображення верхньої панелі керування відбувається за допомогою елементу View “<LinearLoyaut>”. Всередині розміщені інші елементи, такі як: “ImageView”, “TextView”, “Button”.

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:paddingTop="5dp"
    android:paddingBottom="5dp"
    android:layout_marginBottom="5dp"
    android:background="#BBC2FB"
    android:orientation="horizontal">
    <ImageView...>
    <TextView
        android:id="@+id/name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="10dp"
        android:text="Ім`я"
        android:layout_weight="1"
        android:layout_gravity="center"
        android:textSize="20sp" />
    <Button
        android:id="@+id/logout"
        android:backgroundTint="#F16C62"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginRight="10dp"
        android:layout_gravity="center"
        android:text="Вийти" />
</LinearLayout>
```

Рисунок 3.10 – Код верхньої панелі керування

Наведемо програмну реалізацію відображення графічного інтерфейсу верхньої панелі керування (рис. 3.10).

Відображення інтерфейсу профілю користувача, при успішному вході в додаток (рис. 3.11).

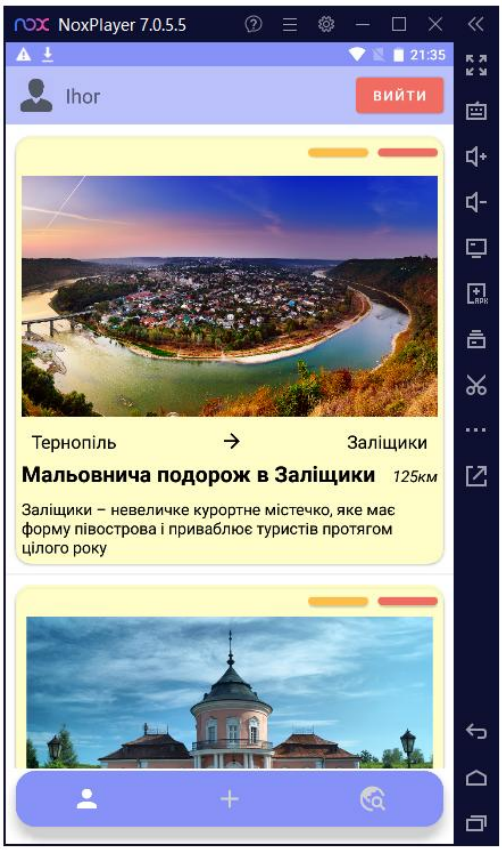

Рисунок 3.11 – Інтерфейс профілю користувача

Графічний інтерфейс форми добавлення допису про подорож складається з елементів: заголовку, заголовку подорожі, міста відправлення, міста прибуття, дистанції, опису, фото, кнопки “Опублікувати” (рис. 3.12). За відображення інтерфейсу відповідає елемент “<ScrollView>”, всередині нього розміщені всі інші елементи.

ScrollView - це віджет в Android Studio, який дозволяє створювати прокрутні контейнери для вмісту, що не поміщається на одну сторінку екрану. Він може мати лише один прямий дочірній елемент. Тобто, можна розмістити всі вмістові віджети в одному дочірньому контейнері (наприклад, LinearLayout або RelativeLayout), який потім буде прокручуватись всередині ScrollView.

В цьому віджеті розміщені елементи: TextView, TextInputLayout, ImageView та TextInputEditText.

ImageView відповідає за відображення фото, тому після натиснення на синю область – завантажити фото, користувач може вибрати фото зі свого пристрою і воно з’явиться в формі.

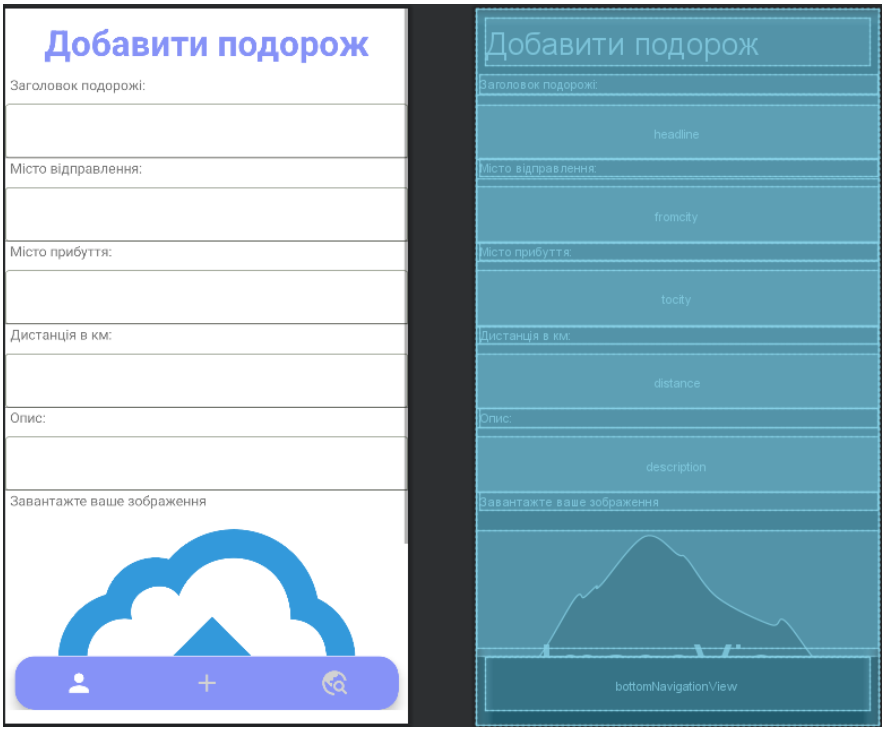

Рисунок 3.12 – Форма добавлення нового допису про подорож

Наведемо програмну реалізацію графічного інтерфейсу контейнера ScrollView:

```
<ScrollView
    android:layout_width="match_parent"
    android:layout_height="680dp"
    android:layout_above="@+id/bottomNavigationView"
    android:layout_marginBottom="70dp">
```

Графічний інтерфейс редагування допису про подорож складається з таких елементів: заголовку, заголовку подорожі, міста відправлення, міста прибуття, дистанції, опису, кнопки “Редагувати”. Відображення інтерфейсу відбувається за допомогою елементу RelativeLayout.

RelativeLayout - контейнер розмітки в Android Studio, який дозволяє розміщувати віджети на екрані відносно інших віджетів. Всередині нього розміщені елементи: TextView, TextInputLayout та TextInputEditText, для кожного поля вводу.

Відображення інтерфейсу редагування допису про подорож (рис. 3.13).

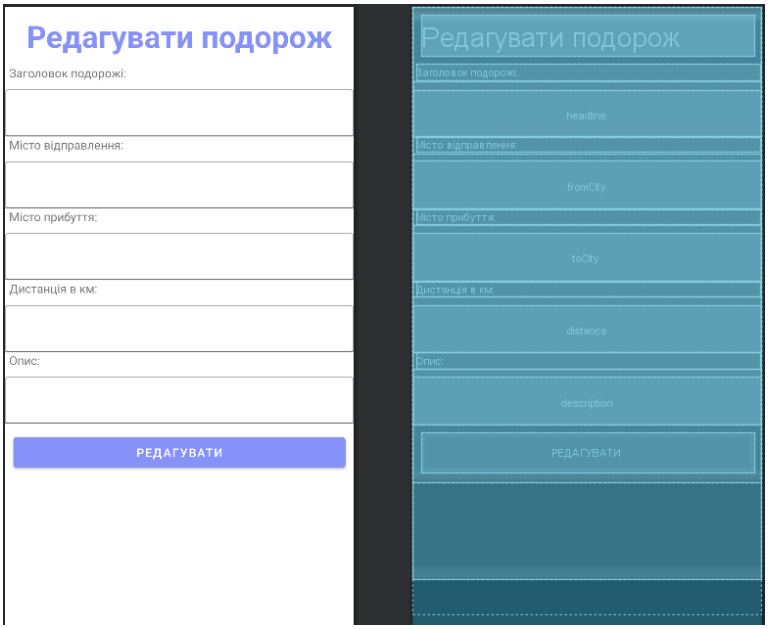

Рисунок 3.13 – Графічний інтерфейс редагування допису про подорож

Наведено програмну реалізацію графічного інтерфейсу редагування допису про подорож одного поля вводу:

```
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginLeft="5dp"
    android:text="Місто відправлення:" />
<com.google.android.material.textfield.TextInputLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
    <com.google.android.material.textfield.TextInputEditText
        android:id="@+id/fromCity"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="10dp"
        android:background="@drawable/edit_background"
        android:text="" />
</com.google.android.material.textfield.TextInputLayout>
```

Графічний інтерфейс розділу рекомендованих подорожей складається з верхньої панелі керування, кнопок “Рекомендовані” та “Найочікуваніші”, вибраного списку подорожей та нижньої навігаційної панелі. При натисканні кнопки “Найочікуваніші”, список зміниться на найочікуваніші подорожі. Таким чином користувач може переміщатися між цими розділами, натисненням бажаної кнопки.

Відображення графічного інтерфейсу списку рекомендованих подорожей користувача (рис. 3.14)

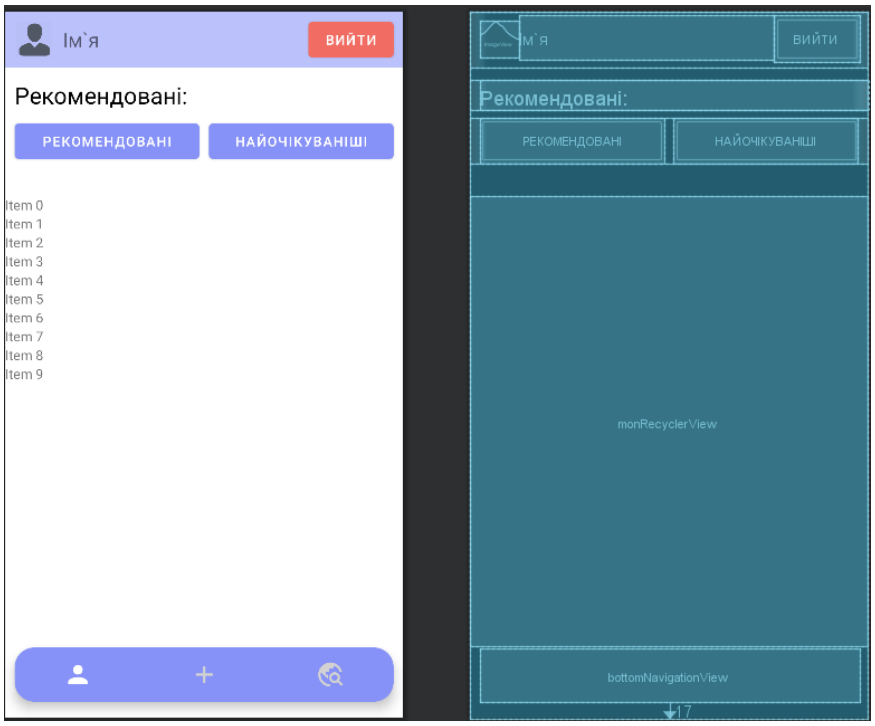

Рисунок 3.14 – Графічний інтерфейс розділу рекомендовані подорожі

Наведено програмну реалізацію інтерфейсу кнопок “Рекомендовані” та “Найочікуваніші” :

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="110dp">
    <Button
        android:id="@+id/recommendationListBtn"
        android:layout_width="171dp"
        android:layout_height="wrap_content"
        android:layout_marginLeft="10dp"
        android:layout_marginRight="10dp"
        android:layout_weight="1"
        android:text="Рекомендовані" />
    <Button
        android:id="@+id/expectationListBtn"
        android:layout_width="171dp"
        android:layout_height="wrap_content"
        android:layout_marginRight="10dp"
        android:layout_weight="1"
        android:text="Найочікуваніші" />
</LinearLayout>
```

3.3 Тестування програмного забезпечення та оцінка якості

Тестування програмного забезпечення є невід'ємною частиною процесу розробки мобільних застосунків і програмного забезпечення, оскільки воно дозволяє виявити й виправити помилки та недоліки до того, як програма буде випущена для користувачів. Враховуючи особливості мобільних додатків, тестування мобільного програмного забезпечення має свої характеристики та вимоги. Дослідження зосереджені на методах тестування загального призначення, не завжди відповідають потребам мобільних програм. Тому важливо враховувати специфічні вимоги та особливості мобільного тестування для досягнення високої якості та надійності мобільного програмного забезпечення[8].

Під час тестування буде перевірено спроможність функцій та аспект додатку, враховуючи різні сценарії взаємодії з користувачем. Буде використано ручні тести і платформу Postman для тестування API, щоб забезпечити функціональність та якість.

Postman – це потужний інструмент для розробки, тестування та документування API. Він надає можливість розробникам взаємодіяти з різними ресурсами шляхом відправки HTTP-запитів таких як GET, POST, PUT, DELETE та інших. Використання Postman спростить процес тестування API шляхом надання зручного графічного інтерфейсу користувача та багатофункціональних можливостей[9].

За допомогою Postman проведемо тестування форми логін і реєстрації за допомогою http запитів в базу даних MySQL.

Тестування складається з наступних етапів:

- відкриття Postman і створення нового запиту;
- введення URL-адреси бази даних MySQL;
- добавлення заголовків, необхідних для взаємодії з базою даних;
- налаштування параметрів запиту;
- виконання запиту, натиснення кнопки "Send" для виконання запиту до

бази даних MySQL;

- проведення аналізу результатів.

Для перевірки форми реєстрації, створено новий запит в Postman. Вибрано тип запиту POST та добавлено значення для ключів: name, email, password, в розділі “Body” (рис. 3.15). Після натискання кнопки “Send”, отримано відповідь “success” , що свідчить про успішне створення нового користувача в базі даних (рис. 3.16).

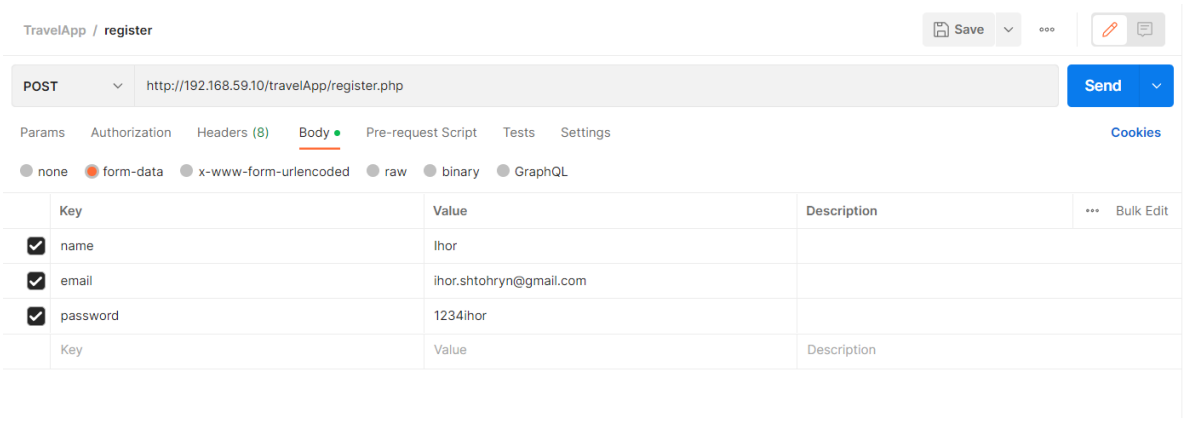

Рисунок 3.15 – Створення нового запиту в Postman форми реєстрації

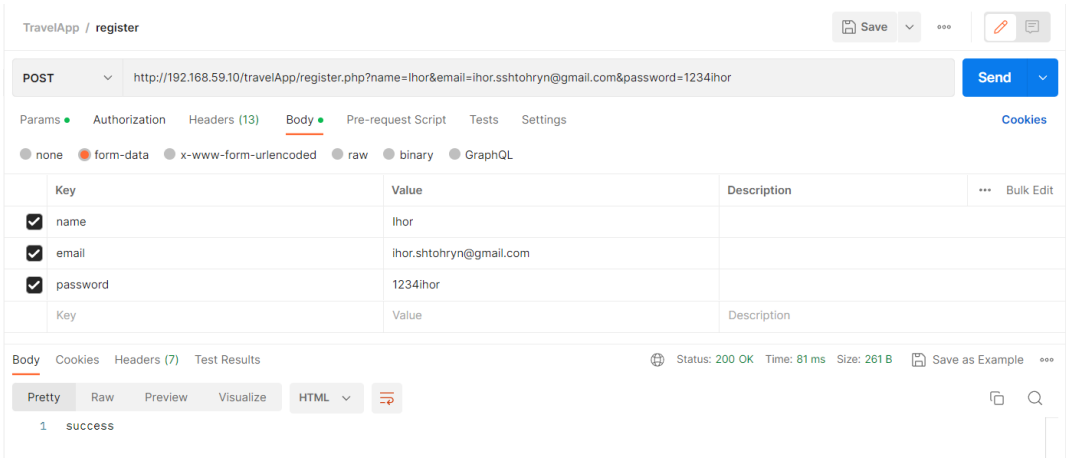

Рисунок 3.16 – Успішне створення користувача

Для перевірки форми входу, створено новий запит в Postman. Вибрано тип запиту POST, в розділі “Body” добавлено значення що були введені при реєстрації для ключів: email, password(рис. 3.17).

Після натискання кнопки “Send”, отримано відповідь в форматі JSON, що

свідчить про успішний вхід в додаток(рис. 3.18).

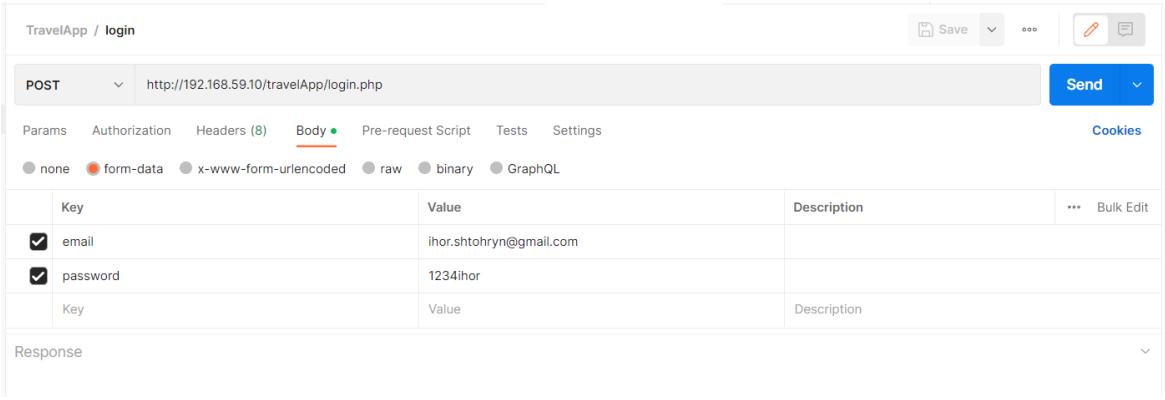

Рисунок 3.17 – Створення нового запиту в Postman форми входу

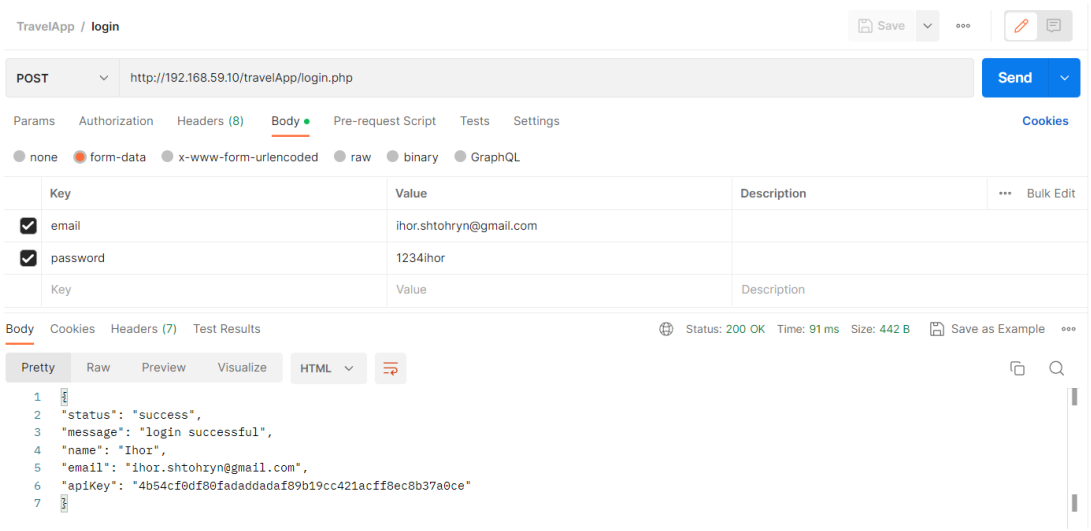

Рисунок 3.18 – Повідомлення про успішний вхід в додаток

Наступним етапом тестування додатку, є тестування за допомогою емулятора Nox. Емулятор є корисним для перевірки функціональності та сумісності застосунку з різними версіями Android та різними пристроями.

За допомогою Nox проведно перевірку функціональності добавлення нових дописів подорожі, їх редагування і видалення з бази даних.

Відображення профілю користувача після успішного завантаження емулятору і запуску в ньому додатку (рис. 3.19). Попередньо в профілі користувача добавлено декілька дописів подорожей.

Після натискання кнопки “Добавити допис”, відкривається форма для

внесення даних про нову подорож (рис. 3.20).

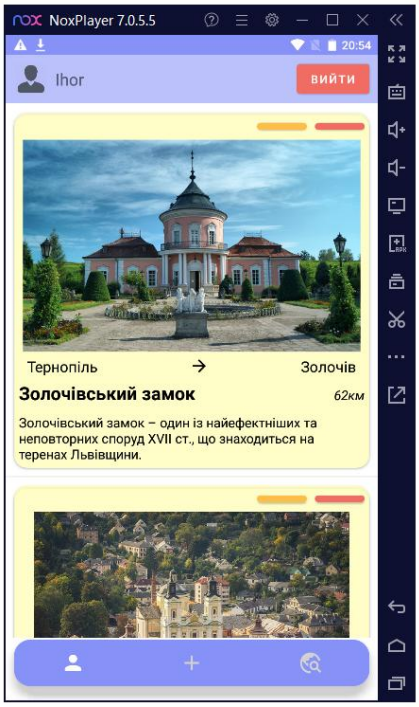

Рисунок 3.19 – Профіль користувача в емуляторі Nox

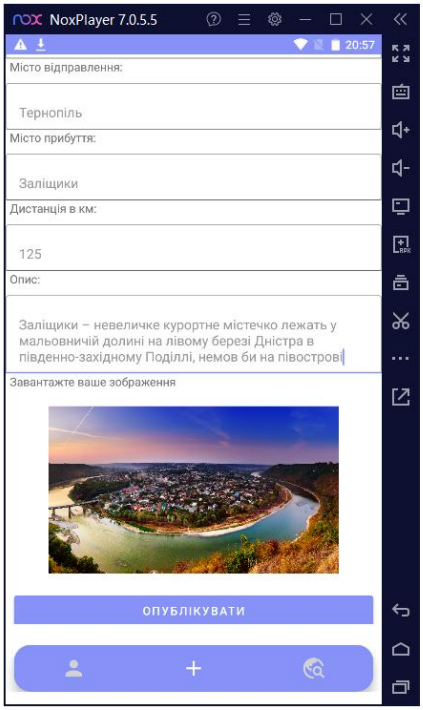

Рисунок 3.20 – Форма добавлення нового допису про подорож

Після успішного заповнення форми і натиснення кнопки “Опублікувати”,

новий допис добавляється в профіль користувача (рис.3.21).

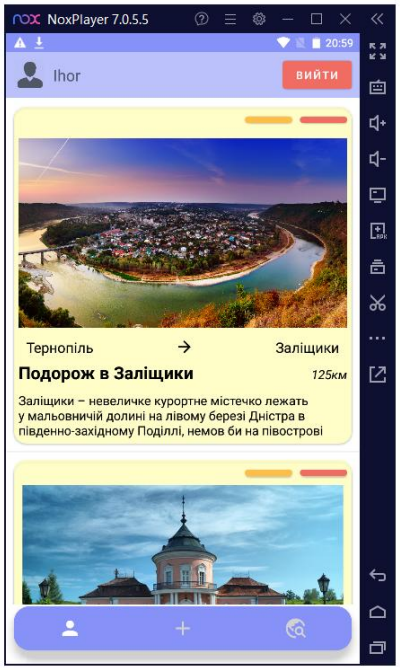

Рисунок 3.21 – Відображення нового допису в профілі користувача

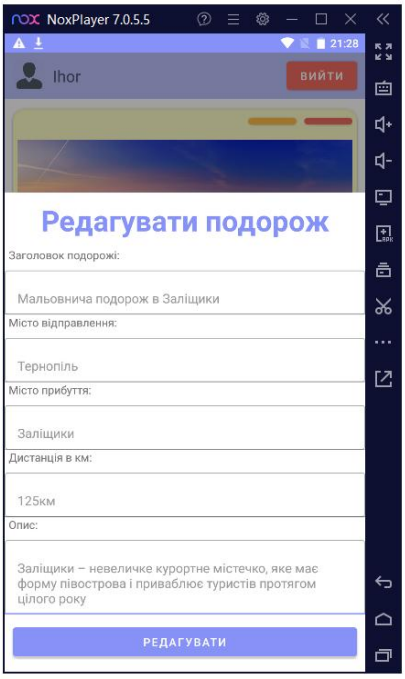

Рисунок 3.22 – Форма редагування допису про подорож

Наведено редагування вибраного допису про подорож(рис. 3.22).

Натиснення кнопки редагувати(жовта кнопка) в правому верхньому куті

допису дозволяє відкрити форму редагування.

Натиснення кнопки “Редагувати”, дозволяє оновити допис і відобразити внесені зміни (рис. 3.23).

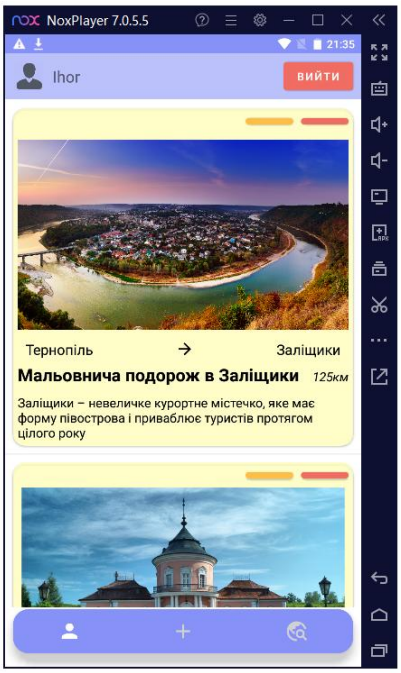

Рисунок 3.23 – Оновлений допис про подорож

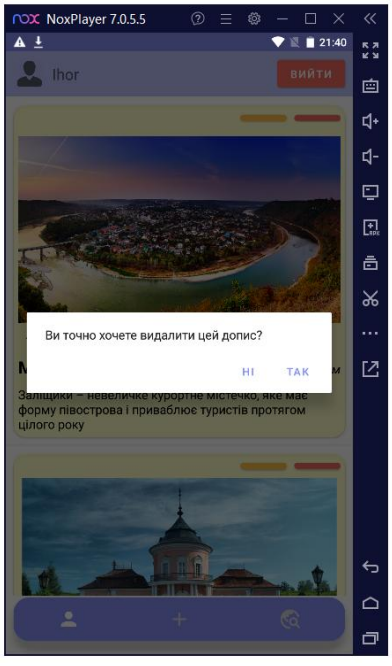

Рисунок 3.24 – Вікно підтвердження дій видалення допису

Наведено вікно для видалення вибраного допису про подорож(рис. 3.24).

Видалення вибраної подорожі, відбувається натисненням кнопки

видалити(червона кнопка) в правому верхньому куті допису. Після натискання кнопки відображається вікно для підтвердження дій.

Після натисканні кнопки “Так”, допис про подорож видалиться, при “Ні” - дія скасовується.

За допомогою тестування емулятором Nox, перевірено основні функції застосунку, а саме відображення списку дописів про подорож, добавлення, редагування, видалення окремої поїздки.

3.4 Результати розробки

В ході виконання кваліфікаційної роботи, на тему “Розробка інтерактивного додатку для подорожей з використанням Java та Android Studio”, розроблено застосунок для операційної системи Android.

В розділі огляду предметної області, проаналізовано основні технології і мови програмування для створення додатків, обгрунтовано вибір мови програмування Java та середовища розробки Android Studio.

В розділі проєктування додатку, розроблено модель предметної області та бізнес модель. Спроєктовано архітектуру застосунку, добавлено діаграми класів, послідовностей та варіантів використання.

В розділі констроювання додатку, реалізовано ключові класи, розроблено графічний інтерфейс користувача, проведено тестування і огляд функціоналу застосунку.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Ергономічні проблеми безпеки життєдіяльності

При написанні, кваліфікаційної роботи на здобуття освітнього ступеня “бакалавр” на тему: “Розробка інтерактивного додатку для подорожей з використанням Java і Android Studio”, я дійшов висновку, що працюючи над проєктами будь-яких систем, дуже важливим є уникнення ергономічних проблем.

В ході застосування мого додатку, користувач повинен дотримуватися ергономічних вимог щодо робочого місця, часу та відпочинку. А використання функціоналу цілої системи, а саме: перегляд дописів, додавання фотографій та опису подорожей, їх редагування та видалення, передбачає перебування споживача в одній позиції впродовж тривалого часу , що може погано впливати на його здоров’я та життєдіяльність. Щоб уникнути цих негативних наслідків від перегляду застосунків та користування мобільними пристроями чи ПК, потрібно проаналізувати і вивчити ергономічні проблеми та вимоги безпеки життєдіяльності.

Під ергономікою розуміють галузь знань, що комплексно вивчає трудову діяльність людини у системах людина-машина-середовище (ЛМС) з метою забезпечення її ефективності, безпеки та комфорту[11].

Мета ергономіки - підвищення ефективності системи ЛМС, забезпечення безпеки праці.

Ергономічні проблеми безпеки життєдіяльності – це проблеми, пов’язані з проектуванням та організацією праці та середовища, які впливають на безпеку та здоров’я людей та включають:

– тривалість і інтервали роботи і відпочинку;

– організація та планування робочого місця;

– фізичні і розумові навантаження і стреси.

Продуктивність праці, працездатність людини в багатьох випадках визначаються правильним встановленням режиму праці та відпочинку, що

означає зміну періодів праці та відпочинку протягом доби, тижня та довшого терміну.

Реалізація основних ергономічних вимог до режимів праці та відпочинку дає змогу забезпечити необхідний рівень працездатності, зменшити втому, зберегти здоров’я людей.

При розробленні режимів праці та відпочинку необхідно встановити[13]:

– тривалість періодів безперервної праці протягом доби (тривалість робочої зміни);

– інтервали між періодами безперервної праці (між змінами);

– кількість змін, які забезпечують чергування;

– тривалість та форму відпочинку.

У системі ЛМС завжди є 3 елементи: предмет праці, засоби праці та суб'єкт праці[11]. Найменшою цільною одиницею, де наявні вказані елементи, є місце праці. Місце праці — це зона, де є необхідні технічні засоби, де відбувається трудова діяльність людини. Місце праці обладнане засобами відображення інформації, органами керування та допоміжним обладнанням. Організацією місця праці називається проведення системи заходів щодо його обладнання засобами та предметами праці і їх розташуванням у визначеному порядку з метою досягнення:

– оптимізації умов трудової діяльності;

– безпеки праці;

– максимальної ефективності;

– комфортності роботи людини.

До робочого місця ставляться такі вимоги:

– достатній робочий простір, який дає змогу працюючій людині здійснювати необхідні рухи та переміщення;

– достатні фізичні, зорові та слухові зв'язки між людиною та обладнанням, а також між людьми Під час виконання спільного трудового завдання;

– необхідний рівень освітлення;

– допустимий рівень шуму, і вібрацій та інших шкідливих чинників, які генерує обладнання місця праці та інші джерела;

– наявність необхідних засобів захисту;

– оптимальне розташування робочих місць, а також безпечні та достатні проходи для працюючих людей.

Засоби відображення інформації забезпечують своєчасність отримання людиною потрібної інформації для її аналізу, логічної обробки та прийняття погрібного рішення.

Використання застосунків вимагає концентрації, аналізу та вирішення складних завдань. Це призводить до психологічного напруження та втоми. Для зменшення розумових навантажень рекомендується дотримуватися правил раціональної організації робочого процесу, встановлення чітких цілей та пріоритетів, а також надання користувачам необхідних інструментів та підтримки.

Високі очікування, термінові завдання та обмежений час створюють стресові ситуації для користувачів. Стрес може негативно впливати на якість роботи та здоров'я. Для зменшення стресу важливо забезпечити раціональне планування роботи, розподіл завдань та належну комунікацію між учасниками проєкту. Звернення уваги на фізичні і розумові навантаження, а також на стресові ситуації під час розробки та використання інтерактивного додатку допоможе забезпечити безпечні умови праці та покращити ефективність та задоволення користувачів.

Щоб вирішити ці проблеми, слід скористатися ергономічними принципами та стандартами для оптимізації взаємодії між людьми, машинами та навколишнім середовищем.

Ергономічні рекомендації до роботи за комп'ютером[12]:

- перед увімкненням живлення користувач ПК має переконатися у наявності заземлення, передбаченого інструкцією, а також перевірити справність шнура живлення і шнура зв'язку клавіатури із блоком живлення;
- положення тулуба користувача ПК є таким, щоб його погляд був спрямований прямо на монітор;
- нижній край екрана монітора знаходиться на 20 см нижче від рівня очей користувача ПК;

- верхній край екрана монітора є на рівні чола користувача ПК;

- екран монітора розташований на відстані не менше 75—120 см від очей користувача ПК;

- робоче крісло користувача ПК відрегульоване так, щоб кут між його стегнами і тулубом становив 90°;

- при восьмигодинному робочому дні кількість опрацьованих (шляхом введення даних або їх зчитуванням з екрана монітора) символів (знаків) не повинна перевищувати 30 000 за 4 години роботи.

Надання належної уваги ергономічним аспектам роботи за комп'ютером є важливим завданням для користувача. Інвестиція в налагодження комфортних умов праці та дотримання ергономічних принципів розглядається, як важлива складова частина створення здорового та продуктивного робочого середовища.

4.2 Гігієнічні вимоги до організації та обладнання робочих місць з ВДТ

Гігієнічні вимоги до організації обладнання робочих місць з ВДТ стають необхідністю для забезпечення оптимальних умов праці. Правильно організоване робоче місце забезпечує ефективність та продуктивність працівника, а також допомагає уникнути можливих проблем зі здоров'ям, пов'язаних з тривалим використанням ВДТ.

Метою даного розділу є дослідження та опис гігієнічних вимог до організації обладнання робочих місць з ВДТ. Розглянемо такі аспекти, як розташування екрану, ергономічність робочого місця, освітлення, перерви та відпочинок. Дослідження цих вимог допомагає встановити оптимальні умови роботи з ВДТ та забезпечити здоров'я та комфорт працівників.

1. Екран ВДТ має розташовуватися на оптимальній відстані від очей користувача, що становить 600...700 мм, але не ближче ніж за 600 мм з урахуванням розміру літеро-цифрових знаків і символів[14].

Розташування екрана ВДТ повинен забезпечувати зручність зорового спостереження у вертикальній площині під кутом +30º до нормальної лінії погляду працюючого.

Монітори потрібно розміщувати на висоті рівня очей (висота від підлоги до нижнього краю екрана має становити 95…100 см) на відстані 60…70 см від оператора (відстань від краю столу – 50…70 см). Кут зору працюючого щодо екрана має дорівнювати 10…20°, але не більше 40°, кут між верхнім краєм монітора і рівнем очей користувача має становити менш як 10°. Найдоцільніше розміщувати екран перпендикулярно до лінії погляду користувача. Кут нахилу екрана по вертикалі має становити 0…30°. З цією метою сучасні монітори комплектують підставкою з поворотним кронштейном, що дає змогу регулювати кут нахилу монітора і горизонтально обертати його навколо вертикальної осі. Висоту екрана від поверхні підлоги регулюють змінюючи висоту робочої поверхні столу. Іноді монітори встановлюють на спеціальні підставки, що уможливлює його переміщення у просторі у вертикальному та горизонтальному напрямах.

З метою зменшення напруження очей потрібно, щоб відстань між краями сусідніх точок зображення на моніторі не перевищувала Г. Оптимальний розмір літеро-цифрових знаків – 16…20, складних знаків – 35…40. Оптимальні співвідношення параметрів літер і цифр такі: ширина значка – 0,75 їх висоти, товщина ліній при зворотному контрасті – 1/6-1/8, відстань між знаками – 0,25…0,5 висоти значка, між словами – 0,75…1, між рядками – 0,5…1.

2. Приміщення для роботи з ВДТ повинні мати природне та штучне освітлення відповідно до СНиП II-4-79[16].

Природне освітлення має здійснюватися через світлові прорізи, орієнтовані переважно на північ чи північний схід і забезпечувати коефіцієнт природної освітленості (КПО) не нижче ніж 1,5 %. Розраховується КПО за методикою, викладеною в СНиП II-4-79.

Штучне освітлення в приміщеннях з робочими місцями, обладнаними ВДТ ЕОМ та ПЕОМ, має здійснюватися системою загального рівномірного освітлення.

У виробничих та адміністративно-громадських приміщеннях, у разі переважної роботи з документами, допускається застосування системи комбінованого освітлення.

Зазначення освітлення освітленості на поверхні робочого столу в зоні розміщення документів має становити 300 – 500 лк. Якщо ці значення освітленості неможливо забезпечити системою загального освітлення, допускається використовувати місцеве освітлення. При цьому світильники місцевого освітлення слід встановлювати таким чином, щоб не створювати блиск на поверхні екрана, а освітленість екрана має не перевищувати 300 лк.

Джерела світла в разі штучного освітлення мають застосовуватись переважно люмінесцентні лампи типу ЛБ. У разі влаштування відбитого освітлення у виробничих та адміністративно-громадських приміщеннях допускається застосування метало галогенних ламп потужністю 250 Вт. Допускається застосування ламп розжарювання у світильниках місцевого освітлення.

Система загального освітлення має становити суцільні або переривчасті лінії світильників, розташовані збоку від робочих місць (переважно ліворуч), паралельно лінії зору працюючих.

Допускається використання світильників таких класів світлорозподілу:

– прямого світла – П;

– переважно прямого світла – Н;

– переважно відбитого світла – В.

Для загального освітлення слід застосовувати світильники серії ЛПО 3б із дзеркальними гратами, укомплектовані високочастотними пускорегулювальними апаратами (ВЧ ПРА). Допускається застосовувати світильники цієї серії без ВЧ ПРА тільки в модифікації “Косо світло”. Застосування світильників без розсіювачів та екрануючих грат заборонено.

3. При організації праці, що пов'язана з використанням ВДТ ЕОМ і ПЕОМ, для збереження здоров'я працюючих, запобігання професійним захворювання і

підтримки працездатності слід передбачити внутрішньо змінні регламентовані перерви для відпочинку[15].

Внутрішньо змінні режими праці і відпочинку мають передбачати додаткові нетривалі перерви в періоди, що передують появі об'єктивних і суб'єктивних ознак стомлення і зниження працездатності.

При виконанні протягом дня робіт, що належать до різних видів трудової діяльності, за основну роботу з ВДТ ЕОМ і ПЕОМ слід вважати таку, що займає не менше 50 % часу впродовж робочої зміни мають передбачатися:

– перерви для відпочинку і вживання їжі (обідні перерви);

– перерви для відпочинку і особистих потреб (згідно з трудовими нормами);

– додаткові перерви, що вводяться для окремих професій з урахуванням особливостей трудової діяльності.

Тривалість обідньої перерви визначається чинним законодавством про працю і Правилами внутрішнього трудового розпорядку підприємства (Організації, установи).

ВИСНОВКИ

У даній кваліфікаційній роботі було представлено процес розробки інтерактивного додатку для подорожей з використанням мови програмування Java та середовища розробки Android Studio. Досліджено основні аспекти розробки мобільних додатків, зокрема створення інтерфейсу користувача, роботу з базою даних та реалізацію функцій, необхідних для ефективного використання додатку під час подорожей. У процесі розробки були використані сучасні інструменти, які забезпечують швидкість та ефективність розробки мобільних додатків. Android Studio надає розширені можливості для створення інтерфейсу користувача за допомогою графічного редактора, а також вбудовану систему симуляції, яка дозволяє перевірити роботу додатку на різних пристроях та роздільних здатностях екрану.

В підсумку було розроблено інтерактивний додаток для подорожей, який містить такий функціонал:

- реєстрація та авторизація користувачів. Користувач має можливість створити свій обліковий запис ввівши логін, пошту і пароль. На дані поля стоять деякі вимоги, щоб вберегти користувача від випадкового створення запису;
- перегляд дописів про подорожі які користувач добавив в профіль. Користувач має можливість добавити подорож, яка буде відображена в його профілю, можливість редагувати або видаляти подорожі;
- перегляд рекомендованих та найочікуваніших подорожей.

Дана робота також містить презентацію графічного інтерфейсу програми з її докладним описом та графічним зображення результатів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Joorabchi, M.E., Mesbah, A., Kruchten, P.: Real challenges in mobile app development. In: ACM / IEEE International Symposium on Empirical Software Engineering and Measurement, Baltimore, Maryland, US, October 2013
2. Android Application Development: A Brief Overview of Android Platforms and Evolution of Security Systems / A. Sarkar et al. Palladam : IEEE, 2019. 79 p.
3. Iversen J., Eierman M. Learning Mobile App Development. Boston : Addison-Wesley, 2014. 441 p.
4. Shevtsiv N. A., Striuk A. M. Cross platform development vs native development. Kryvyi Rih : Kryvyi Rih National University, 2020. 83 p.
5. The Java Language Specification, Java Se 7 Edition / J. Gosling et al. 7th ed. California, 2013. 644 p.
6. Cotroneo D., Orlando S., Russo S. Characterizing Aging Phenomena of the Java Virtual Machine. Beijing, 2007. 136 p.
7. Command-line tools | Android Studio | Android Developers. Android Developers. URL: https://developer.android.com/tools
8. Hagos T. Learn Android Studio 4. Berkeley, CA : Apress, 2020. URL: https://doi.org/10.1007/978-1-4842-5937-5
9. Ranta J. Testing AWS hosted Restful APIs with Postman. Vantaa, 2023. 34 p.
10. Working with the ScrollView | CodePath Android Cliffnotes. CodePath Cliffnotes. URL: https://guides.codepath.com/android/Working-with-the-ScrollView
11. Желібо Є.П., Зацарний В.В. Безпека життєдіяльності. Підручник. – К.: Каравела, 2006. – 288 с.
12. Лекція на тему «Ергономічні обґрунтування й оцінки у безпеці життєдіяльності» [Електронний ресурс] URL: http://opcb.kpi.ua/wp-

content/uploads/2014/09/%D0%9B1.pdf

13. Я. І. Бедрій. Безпека життєдіяльності: Навчальний посібник. Київ Кондор 2009 286с
14. Закон України «Про охорону праці». [Електронний ресурс] URL: https://zakon.rada.gov.ua/laws/show/2694-12
15. К.Н. Ткачук, О.Л. Гуменюк, Т.П. Бивойно, Н.М. Денисова, В.М. Челябієва, К.К. Ткачук, Н.П. Буяльська Безпека праці та промислова санітарія. ЧДТУ, 2010. – 368 с.
16. Гігієнічні вимоги до організації роботи з візуальними дисплейними терміналами електронно-обчислювальних машин [Електронний ресурс] URL: https://zakon.rada.gov.ua/rada/show/v0007282-98

ДОДАТКИ

ДОДАТОК А

Лістинг коду розробленого додатку

Лістинг 1а – Файл UserProfile.java

```
// Application package
package edu.example.travelapp;
// Importing necessary modules and components
import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.recyclerview.widget.DividerItemDecoration;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;
import com.android.volley.Request;
import com.android.volley.RequestQueue;
import com.android.volley.Response;
import com.android.volley.VolleyError;
import com.android.volley.toolbox.StringRequest;
import com.android.volley.toolbox.Volley;
import com.google.android.material.bottomnavigation.BottomNavigationView;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
// Java util
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

public class UserProfile extends AppCompatActivity {

  public TextView userName;
  public SharedPreferences sharedPreferences;
  public Button buttonLogout;
  public RecyclerView recyclerView;
  public ArrayList<UserModel> recyclerList;

  FirebaseDatabase firebaseDatabase;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
```

```
setContentView(R.layout.activity_main);

// Bottom Navigation
BottomNavigationView bottomNavigationView =
findViewById(R.id.bottomNavigationView);
bottomNavigationView.setSelectedItemId(R.id.bottom_profile);
Assigning values to variables
tonLogout = findViewById(R.id.logout);
yclerView = findViewById(R.id.recyclerView);
yclerList = new ArrayList<>();

Connection with DB Firebase
ebaseDatabase = FirebaseDatabase.getInstance();

Display travel list
rAdapter recyclerAdapter = new UserAdapter(recyclerList,
 getApplicationContext());
earLayoutManager linearLayoutManager = new LinearLayoutManager(this);
yclerView.setLayoutManager(linearLayoutManager);
yclerView.addItemDecoration(new DividerItemDecoration(recyclerView
.getContext(), DividerItemDecoration.VERTICAL));
yclerView.setNestedScrollingEnabled(false);
yclerView.setAdapter(recyclerAdapter);

 // Assigning values to sharedPreferences
 sharedPreferences = getSharedPreferences("MyAppName",
 MODE_PRIVATE);
 userName = findViewById(R.id.name);
 // Checking if the user is authenticated
 if (sharedPreferences.getString("logged", "false")
 .equals("false")) {
     Intent intent = new Intent(getApplicationContext(),
     Login.class);
     startActivity(intent);
     finish();
 }
 userName.setText(sharedPreferences.getString("name", ""));

 // Show List of Travel
 firebaseDatabase.getReference().child("post")
 .addListenerForSingleValueEvent(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot snapshot) {
      for(DataSnapshot dataSnapshot : snapshot.getChildren()){
        UserModel userModel = dataSnapshot
        .getValue(UserModel.class);
          recyclerList.add(userModel);
      }
      recyclerAdapter.notifyDataSetChanged();
    }
    @Override
```

```
public void onCancelled(@NonNull DatabaseError error) {}
});
// Bottom Navigation items
bottomNavigationView.setOnItemSelectedListener(item -> {
  switch(item.getItemId()){
    case R.id.bottom_profile:
      return true;
    case R.id.bottom_add:
      startActivity(new Intent(getApplicationContext(),
       AddTravel.class));
      overridePendingTransition(R.anim.slide_in_right,
       R.anim.slide_out_left);
      finish();
      return true;
    case R.id.bottom_travel_list:
      startActivity(new Intent(getApplicationContext(),
      TravelRecommendationList.class));
      overridePendingTransition(R.anim.slide_in_right,
      R.anim.slide_out_left);
      finish();
     return true;
  }
  return false;
});

// Button Log out
buttonLogout.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View v) {
    RequestQueue queue = Volley
    .newRequestQueue(getApplicationContext());
      // Log out Request
      String url = "http://192.168.59.10/travelApp/logout.php";
      StringRequest stringRequest =
      new StringRequest(Request.Method.POST,
      url, new Response.Listener<String>() {
        @Override
          public void onResponse(String response) {
            if(response.equals("success")){
              SharedPreferences.Editor editor =
              sharedPreferences.edit();
              editor.putString("logged", "");
              editor.putString("name", "");
              editor.putString("email", "");
              editor.putString("apiKey", "");
              editor.apply();
              Intent intent = new Intent(getApplicationContext(),
              Login.class);
              startActivity(intent);
              finish();
            }
```

```
                    else
                    Toast.makeText(UserProfile.this, response,
                     Toast.LENGTH_SHORT).show();
                  }
                }, new Response.ErrorListener() {
                  @Override
                  public void onErrorResponse(VolleyError error) {
                    error.printStackTrace();
                  }
                }) {
                  protected Map<String, String> getParams() {
                    paramV.put("email", sharedPreferences.
                    getString("email", ""));
                    Map<String, String> paramV = new HashMap<>();
                    paramV.put("apiKey", sharedPreferences.
                    getString("apiKey", ""));
                  return paramV;
                  }
                };
                queue.add(stringRequest);
            }
        });
    }
}
```

Лістинг 1а – Файл AddTravel.java

```
// Application package
package edu.example.travelapp;
// Importing necessary modules and components
import android.Manifest;
import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.appcompat.app.AppCompatActivity;
import android.app.ProgressDialog;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.RelativeLayout;
import android.widget.TextView;
import android.widget.Toast;
import com.google.android.gms.tasks.OnFailureListener;
import com.google.android.gms.tasks.OnSuccessListener;
import com.google.android.material.bottomnavigation.BottomNavigationView;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.storage.FirebaseStorage;
import com.google.firebase.storage.StorageReference;
import com.google.firebase.storage.UploadTask;
import com.karumi.dexter.Dexter;
import com.karumi.dexter.PermissionToken;
import com.karumi.dexter.listener.PermissionDeniedResponse;
import com.karumi.dexter.listener.PermissionGrantedResponse;
```

```
import com.karumi.dexter.listener.PermissionRequest;
import com.karumi.dexter.listener.single.PermissionListener;

public class AddTravel extends AppCompatActivity {
   // Declaration of variables
  private TextView headline, fromCity, toCity, description, distance;
  private String postId;
  ImageView uploadBtn, travelImage;
  Button submitBtn;
  Uri ImageUri;
  RelativeLayout relativeLayout;

  private FirebaseDatabase database;
  private FirebaseStorage firebaseStorage;
  DatabaseReference databaseReference;
  ProgressDialog dialog;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_add)
    dialog = new ProgressDialog(this);
    dialog.setProgressStyle(ProgressDialog.STYLE_SPINNER);
    dialog.setMessage("Будь ласка зачекайте...");
    dialog.setTitle("Допис завантажується");
    dialog.setCanceledOnTouchOutside(false)
    firebaseStorage = FirebaseStorage.getInstance();
    database = FirebaseDatabase.getInstance();
    databaseReference = database.getReference("post")
    // Bottom Navigation View
    BottomNavigationView bottomNavigationView = findViewById(
      R.id.bottomNavigationView);
    bottomNavigationView.setSelectedItemId(R.id.bottom_add);
    // Assigning values to variables
    headline = findViewById(R.id.headline);
    fromCity = findViewById(R.id.fromcity);
    toCity = findViewById(R.id.tocity);
    description = findViewById(R.id.description);
    relativeLayout = findViewById(R.id.relative);
    distance = findViewById(R.id.distance);
    travelImage = findViewById(R.id.travelImage)
    uploadBtn = findViewById(R.id.uploadBtn);
    submitBtn = findViewById(R.id.submit)
    // Upload Image
    uploadBtn.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        uploadImage();
        relativeLayout.setVisibility(View.VISIBLE);
        uploadBtn.setVisibility(View.GONE);
      }
    });

    // Submit Form
    submitBtn.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        dialog.show();
        final StorageReference reference = firebaseStorage
        .getReference().child("posts");
        reference.putFile(ImageUri).addOnSuccessListener(
          new OnSuccessListener<UploadTask.TaskSnapshot>() {
        @Override
        public void onSuccess(UploadTask.TaskSnapshot taskSnapshot) {
           reference.getDownloadUrl().addOnSuccessListener(
```

```
            new OnSuccessListener<Uri>() {
               @Override
               public void onSuccess(Uri uri) {
                UserModel model = new UserModel();
                model.setTravelImage(uri.toString());

                String travelHeadline = headline
                .getText().toString();
                  String travelFromCity= fromCity
                  .getText().toString();
                  String travelToCity = toCity
                  .getText().toString();
                  String travelDescription = description
                  .getText().toString();
                  String travelDistance = distance
                  .getText().toString()+"KM";
                  postId = travelDistance + travelHeadline;
                    // Assignment of new post values
                   model.setHeadline(travelHeadline);
                   model.setFromcity(travelFromCity);
                   model.setTocity(travelToCity);
                   model.setDescription(travelDescription);
                   model.setDistance(travelDistance);
                   model.setTravelId(postId);

                    // Add post by id
                  database.getReference().child("post").child(postId)
                  .setValue(model)
                  .addOnSuccessListener(new OnSuccessListener<Void>() {
                  @Override
                  public void onSuccess(Void unused) {
                    Toast.makeText(AddTravel
                    .this, "Post upload Successfully"
                            +database.getReference("Courses"),
                            Toast.LENGTH_SHORT).show();
                     dialog.dismiss();
                     Intent intent = new Intent(getApplicationContext(),
                     UserProfile.class);
                     startActivity(intent);
                     finish();
                  }
                }).addOnFailureListener(new OnFailureListener() {
                @Override
                public void onFailure(@NonNull Exception e) {
                  dialog.dismiss();
                  Toast.makeText(AddTravel.this, "Error Occured",
                  Toast.LENGTH_SHORT).show();
                }
                });
              }
            });
          }
        });
      }
    });

// Bottom Navigation
bottomNavigationView.setOnItemSelectedListener(item -> {
  switch(item.getItemId()){
    startActivity(new Intent(getApplicationContext(),
    UserProfile.class));
    case R.id.bottom_profile:
      finish();
      overridePendingTransition(R.anim.slide_in_right,
```

```
            R.anim.slide_out_left);
          case R.id.bottom_add:
            return true;
          case R.id.bottom_travel_list:
            return true;
          overridePendingTransition(R.anim.slide_in_right,
          R.anim.slide_out_left);
          startActivity(new Intent(getApplicationContext(),
          TravelRecommendationList.class));
            return true;
          finish();
            return false;
        }
      });
    }
      // Upload Image Method
      private void uploadImage() {
        Dexter.withContext(this)
       .withPermission(Manifest.permission.READ_EXTERNAL_STORAGE)
       .withListener(new PermissionListener() {
        @Override
        public void onPermissionGranted(
          PermissionGrantedResponse permissionGrantedResponse) {
          Intent intent = new Intent();
          intent.setType("image/*");
          intent.setAction(Intent.ACTION_GET_CONTENT);
          startActivityForResult(intent, 101);
        }
        @Override
        public void onPermissionDenied(
          PermissionDeniedResponse permissionDeniedResponse) {
          Toast.makeText(AddTravel.this, "Permission Denied",
          Toast.LENGTH_SHORT);
        }
        @Override
        public void onPermissionRationaleShouldBeShown(
          PermissionRequest permissionRequest,
        PermissionToken permissionToken) {
          permissionToken.continuePermissionRequest();
        }
      }).check();
    }
  // Activity result
    @Override
    protected void onActivityResult(int requestCode, int resultCode,
    @Nullable Intent data) {
      super.onActivityResult(requestCode, resultCode, data);
      if(requestCode== 101 && resultCode == RESULT_OK){
        ImageUri = data.getData();
        travelImage.setImageURI(ImageUri);
      }
    }
  }
```

ДОДАТОК Б

Диск з розробленою програмою