Хмельницький національний університет
Факультет програмування
та комп'ютерних і телекомунікаційних систем
Кафедра інженерії програмного забезпечення

ДИПЛОМНИЙ ПРОЕКТ

Довідково-інформаційна система «Easy Trip» для пошуку місця відпочинку у вигляді мобільного додатку

Назва теми

Рівень вищої освіти Перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр ДППЗ.180110.01.07.ПЗ

Виконав студент IV курсу група ІПЗ-18-1 _____ П.А.Олійник
Підпис Ініціали, прізвище

Керівник д-р фіз.-мат. наук, професор _____ Л. П. Бедратюк
Науковий ступінь, звання Підпис Ініціали, прізвище

Нормоконтролер канд. пед. наук, доцент _____ Н. І. Праворська
Підпис Ініціали, прізвище

До захисту допускаю:
Завідувач кафедри інженерії
програмного забезпечення _____ Л. П. Бедратюк
Підпис Ініціали, прізвище

1 червня 2022 р.

Хмельницький 2022

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Програмування та комп'ютерних і телекомунікаційних систем
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ІТТЗ
Л. П. Бедратюк
01 03 2022 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ)

Олійнику Павлу Анатолійовичу

Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Довідково-інформаційна система «Easy Trip» для пошуку місця відпочинку у вигляді мобільного додатку

Керівник проекту (роботи) Бедратюк Леонід Петрович

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

д-р фіз.-мат. наук, професор

Затверджена наказом ректора університету від 01.03.2022 р. № 18

2. Строк подання студентом проекту (роботи) на кафедру 01.06.2022 р.

3. Вихідні дані до проекту (роботи) Матеріали переддипломної практики

4.Зміст пояснювальної записки (перелік питань, які потрібно розробити) Дослідження предметної області, та постановка задачі, проектування програмного забезпечення, програмна реалізація, тестування мобільного додатку

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень) Презентаційні матеріали (слайди, 15 шт.)

6. Консультанти розділів дипломного проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Праворська Н.І., доцент кафедри ІПЗ		
Антиплагіат	Гурман І.В., доцент кафедри ІПЗ		

7. Дата видачі завдання «_01_» березня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1 Ознайомлення з тематикою дипломного проектування (ДП), визначення та узгодження індивідуальних тем ДП	01.12– 30.12.2021	
2 Дослідження предметної області, в якій планується використання програмного засобу (ПЗ), визначення задач та вимог, розробка технічного завдання	02.01 – 31.01.2022	
3 Проектування програмного забезпечення	01.02 – 28.02 2022	
4 Програмна реалізація	01.03 – 10.04.2022	
5 Тестування програмного забезпечення	11.04 – 30.04.2022	
6 Написання вступу, загальних висновків, оформлення джерел посилання та додатків. Оформлення пояснювальної записки ДП згідно вимог стандартів	01.05 – 25.05.2022	
7 Попередній захист ДП	травень 2021 (згідно графіка)	
8 Перевірка ДП на плагіат, нормоконтроль, отримання відгуків та рецензій. Брошурування (зшиття) пояснювальної записки	26.05 – 30.05.2022	
9 Підготовка до захисту та захист ДП	з 01.06.2022	

Студент _____ Олійник Х.А.
Підпис Ініціали, прізвище

Керівник проекту (роботи) _____ Бездольний І.П.
Підпис Ініціали, прізвище

АНОТАЦІЯ

Тема дипломного проекту: Довідково-інформаційна система «Easy Trip» для пошуку місця відпочинку у вигляді мобільного додатку.

Автор проекту: Олійник Павло Анатолійович.

Керівник проекту: Бедратюк Леонід Петрович.

Пояснювальна записка: 180 с., 29 рис., 16 табл., 3 дод., 17 джерел.

Графічна частина: 15 презентаційних слайдів.

МІСЦЯ ВІДПОЧИНКУ, МОБІЛЬНИЙ ДОДАТОК, КЛІЄНТСЬКА ЧАСТИНА, СЕРВЕРНА ЧАСТИНА, ІДЕНТИФІКАТОР РЕСУРСУ, ANDROID, HTTP, REST API, MYSQL, COMPOSER, PHP, JAVA.

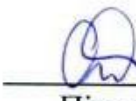
Метою проекту є розробка мобільного додатку на операційній системі Android, що дозволяє користувачам здійснювати пошук готелів та ресторанів відповідно до обраного міста.

У дипломному проекті проведено аналіз предметної області, а також виділені основні поняття, що використовуються протягом роботи над програмним продуктом.

Наведено опис та характеристики програм, що пропонують схожий функціонал. Визначено основні вимоги до програми, а також складено технічне завдання на основі яких була спроектована архітектура додатку, а також проведена його повна реалізація. Було проведено функціональне та конфігураційне тестування, а також тестування методикою «чорний ящик».

Розроблений мобільний застосунок забезпечує швидке та зручне виконання пошуку та отримання інформації у стислому вигляді про ресторани та готелі.

30.05.2022
Дата


Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	ДПІПЗ.180110.01.07.ПЗ	Пояснювальна записка	180		
2	А4		Завдання на дипломний	1		
			проект			
3	А4		Анотація	1		
			Графічні документи			
4	А4		Презентаційні матеріали	15		

Змн.	Арк.	№ докум.	Підпис	Дата	ДПІПЗ.180110.01.07.ПЗ		
Виконав		Олійник П.А.			Довідково-інформаційна система «Easy Trip» для пошуку місця відпочинку у вигляді мобільного додатку Відомість документів	Літ.	Арк. 1 / Акрушів 1
Керівник		Бедратюк Л.П.				ХНУ, ІПЗ-18-1	
Н. Контр.		Праворська Н.І.					
Зав. каф.		Бедратюк Л.П.					

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ.....6
ВСТУП.....7
1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ЗА ТЕМОЮ ДП ТА ПОСТАНОВКА ЗАДАЧІ.....9
1.1 Аналіз предметної області.....9
1.2 Огляд існуючого програмного забезпечення.....14
1.2.1 TripAdvisor.....14
1.2.2 Yelp.....16
1.2.3 Foursquare.....18
1.3 Визначення вимог до ПЗ та постановка задачі.....19
1.3.1 Клієнтська частина.....19
1.3.2 Серверна частина.....20
1.3.3 Моделі варіантів використання.....21
2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....27
2.1 Аналіз та проектування архітектури системи.....27
2.2 Проектування структури бази даних.....30
2.3. Проектування архітектури серверної частини додатку.....37
2.4. Проектування архітектури клієнтської частини додатку.....40
2.5 Проектування інтерфейсу користувача.....42
2.6 Аналіз та вибір технологій і методів реалізації додатку.....55

3 ПРОГРАМНА РЕАЛІЗАЦІЯ.....60
3.1 Реалізація логіки серверної частини додатку.....60
3.2 Реалізація логіки клієнтської частини додатку.....77
3.3 Реалізація розмітки мобільного додатку 86
3.4 Розробка бази даних.....94
3.5 Керівництво користувача 98
3.6 Технічні характеристики мобільного додатку 100
4 ТЕСТУВАННЯ МОБІЛЬНОГО ДОДАТКУ 101
4.1 Аналіз та вибір методів тестування 101
4.2 Тестування серверної частини.....102
4.3 Тестування клієнтської частини.....104
ВИСНОВКИ.....110
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 112
ДОДАТОК А.....114
ДОДАТОК Б 119
ДОДАТОК В.....173

ПЕРЕЛІК СКОРОЧЕНЬ

СКБД	–	система керування базами даних
CEO	–	Chief Executive Officer
FR	–	Foreign Key
HTTP	–	HyperText Transfer Protocol
iOS	–	iPhone Operation System
JSON	–	JavaScript Object Notation
PK	–	Primary Key
PHP	–	Hypertext Preprocessor
REST	–	REpresentational State Transfer
SQL	–	Structured Query Language
XML	–	extensible markup language
XSD	–	XML Schema Definition

ВСТУП

Сьогодні, подорожі є невід'ємною частиною нашого життя. Люди по усьому світу відвідують десятки, якщо не сотні різноманітних місць в пошуках нових вражень та незабутніх емоцій. Подорожі є хорошими ліками для зняття стресу та депресії. Нові країни дозволяють ширше глянути на різноманітні культури та звичаї, що допомагає збільшити кругозір та поглибити знання про світ в цілому. Важливість та необхідність подорожей важко недооцінити.

Відкриття нових країн та міст не обходиться без відвідин місць відпочинку, які допомагають перепочити перед майбутніми подорожами тією чи іншою країною. Серед найбільш популярних місць відпочинку можна виділити готелі та ресторани. Ресторани дають змогу краще зрозуміти колорит, а також традиції країни, що прослідковуються у стравах, які пропонуються у меню, хоча найголовніше для туриста є сам факт того, що можна смачно перекусити, не витрачаючи багато часу та сил. Готелі, з іншої сторони, дозволяють провести ніч в комфортних умовах задля кращого продовження відвідин галерей, музеїв, а також інших визначних місць в межах міста поточної країни. Готелі та ресторани збагачують бюджет країни, а також дозволяють надавати якісний та необхідний сервіс туристам, тому зазвичай кількість таких місць сильно вирізняється з поміж інших місць, які здатні надавати різноманітні послуги задля задоволення потреб відвідувачів, а також туристів.

Актуальність теми дипломного проекту полягає у створенні простого у використанні андроїд додатку, який буде дозволяти отримувати інформацію про готелі та ресторани відповідно до обраної країни чи міста. На сьогоднішній час, смартфонами користується більша частина планети, тому кишеньковий додаток є доволі актуальною річчю, що дозволяє отримувати необхідну інформацію швидко та у зручному вигляді для користувачів.

Мета розробки дипломного проекту полягає у створенні зручного андроїд додатку, який буде включати клієнтську та сервер ну частини, що дозволить

відокремити дві логіки, задля того, щоб мати змогу зручно вносити необхідні зміни та модифікації до кожної з частин.

Мета розробки деталізується на наступні задачі, які необхідно виконати в процесі розробки:

– провести аналіз предметної області, виділити базові поняття;

– проаналізувати існуючі програмні рішення, що пропонують схожий функціонал у порівнянні із розроблюваним мобільним застосунком;

– зробити проектування, яке буде наглядно представляти структуру додатку та його окремих частин;

– реалізувати клієнтську частину з інтерфейсом користувача;

– реалізувати серверну частину, яка буде отримувати вхідні запити та видавати відповідні відповіді;

– провести тестування додатку, щоб виявити несправності у роботі та вирішити їх.

Додаток націлений для надання зручного у користуванні та ефективного андроїд додатку, який буде надавати доступ до необхідної інформації у зручному вигляді з використанням інтуїтивно-зрозумілого інтерфейсу.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ЗА ТЕМОЮ ДП ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної області

Мобільний додаток(застосунок) – це тип програми, призначений для роботи на мобільному пристрої, яким може бути смартфон або планшет. Кожен мобільний додаток надає ізольовану й обмежену функціональність, яка сконцентрована в основному на вирішення конкретної задачі. Мобільні додатки мають обмежені апаратні ресурси, тому вони не є багатофункціональними в порівнянні із персональними комп'ютерами. Також, смартфони є портативними девайсами, тобто користуватись мобільними додатками досить зручно.

Існує дві найбільш відомі операційні системи для мобільних застосунків, які використовуються у більшій половині девайсів світу. Ними є iOS та Android, які постійно конкурують за лідерство на ринку. Відповідно програми для Android пишуться з використанням програмних мов Java або Kotlin, а для iOS існує лише мова Swift.

Операційна система – це програмне забезпечення, яке діє як інтерфейс між компонентами комп’ютерного обладнання та користувачем. Вона займається управлінням робочими процесами та ефективним розподілом системних ресурсів.

iOS є операційною системою для мобільних застосунків iPhone, iPad та інших мобільних пристроїв компанії Apple. Дана операційна система заснована на операційній системі macOS, яка була розроблена для десктопних рішень. Вихід на ринок операційної системи відбувся 9 січня 2007. Розробник і творець системи Стів Джобс, засновник компанії Apple, першим реалізував концепцію сенсорного екрану. Варто відмітити, що до цього корпорація взагалі не займалася виробництвом телефонів.

Можна виділити наступні переваги у використанні iOS у порівнянні із системою Android:

– високий рівень захисту проти різноманітних типів вірусів;

– усі додатки перед публікацією проходять ретельну перевірку від розробників компанії;

– велика увага приділяється вдосконаленню візуального інтерфейсу, задля покращення вражень користувачів протягом користування продуктами Apple;

– висока автономність роботи, що постійно збільшується за рахунок ретельної роботи над оптимізацією енергоресурсів батареї;

– висока якість фотографій;

– чудова загальна швидкодія;

– краща система охолодження.

До недоліків iOS відносять:

– висока вартість девайсів, що робить їх важкодоступними для людей з порівняно не великим бюджетом;

– неможливість модифікації апаратного забезпечення, такого як карти пам’яті або батареї;

– мала кількість додатків у магазині, а також платні рішення коштують набагато дорожче ніж у прямого конкурента.

Android – це операційна система з відкритим вихідним кодом на базі Linux для мобільних пристроїв, таких як смартфони та планшетні комп’ютери. Android був розроблений Google та іншими компаніями. Android пропонує уніфікований підхід до розробки додатків для мобільних пристроїв. Це означає, що розробникам потрібно розробляти лише для даної операційної системи, а їхні програми повинні мати можливість працювати на різних пристроях на базі Android. Дана мобільна операційна система характеризується наявністю відкритого вихідного коду в порівнянні із iOS. Це означає, що будь-хто, хто має досвід розробки, може завантажити Android і гнучко його налаштувати. Дани принцип дозволяє будь-кому завантажувати вихідний код або переглядати документацію, пов’язану з новими випусками, що дозволяє програмістам створювати велику кількість нових рішень, а також виправляти існуючі помилки.

Android має наступні переваги у порівнянні із iOS:

– відкритий вихідний код;

– велика кількість додатків у магазині Google Play;

– додавання нових додатків є доволі легкою процедурою;

– велика доступність, адже більша половина мобільних девайсів створені на основі даної операційної системи;

– порівняно низька ціна.

До недоліків можна віднести наступне:

– повільна швидкість роботи на девайсах зі старим апаратним забезпеченням;

– важче розробляти програмні рішення;

– низький рівень захисту від вірусів;

– малий час роботи батареї;

– характерна реклама у інтерфейсі користувача;

– більша кількість помилок у додатках в порівнянні із iOS.

Для обох вищеописаних операційних систем існують програмні застосунки, які дають можливість завантажувати додатки, написані на конкретній системі. Для Android таким застосунком є Google Play, а для iOS – Apple App store.

Існує три загальні категорії для мобільних додатків, якими є веб додатки, гібридні додатки та нативні додатки. Дана класифікація була створена задля відокремлення різних підходів для створення мобільних застосунків, а також для визначення технологій розробки.

Більшість мобільних програм, які знаходяться в магазині додатків є саме нативними мобільними додатками. Нативні програми, розроблені для Android, написані на мові програмування Java, тоді як програми, розроблені для iOS, написані на Swift. Відомо, що нативні мобільні додатки швидші та надійніші, ніж гібридні чи веб-додатки, що дозволяє їм забезпечувати кращий досвід користувачів. Вони також дозволяють взаємодіяти з інтерфейсами пристрою та внутрішнім обладнанням, таким як камера, список контактів, мікрофон та локація. Можливість взаємодіяти з операційною системою пристрою є причиною, чому багато компаній вирішують розробляти власні мобільні додатки. Реалізація таких додатків вимагає

доволі великі початкові витрати на розробку, що часто заважає організаціям з меншим бюджетом створювати їх. Вищі витрати на розробку є гарантією, що гарантує безперешкодне використовувати програму на кількох пристроях, оскільки для цього потрібно створити два мобільних застосунки: один для Apple – App Store, а інший для Android – Google Play.

Веб-додатки — це додатки, які подібно до нативних мобільних застосунків працюють на мобільних пристроях, однак існують суттєві відмінності між ними. Вони запускаються браузером і зазвичай написані на HTML, CSS та JavaScript. Користувачі спочатку отримують доступ до них так само, як і до будь-якої веб-сторінки при переходять по спеціальній адресі. Насправді це адаптивні веб-сайти, які підлаштовують свій інтерфейс до пристрою користувача. Можна вибрати опцію, яка встановить веб-додаток, але це просто створює закладки для адреси веб-сайту на пристрої користувача. Основний недолік мобільних веб-додатків – це можливість доступу лише через мережу Інтернет.

Гібридні мобільні додатки поєднують елементи як веб, так і нативного застосунку. Гібридні мобільні програми можна встановити на пристрої та запускати через веб-браузер, тому вони є чимось середнім між попередніми двома категоріями. Часто компанії створюють гібридні програми як обгортки для наявної веб-сторінки. Гібридні додатки популярні через те, що вони дозволяють розробляти кросплатформенні рішення і таким чином значно знижують витрати на розробку, тобто одні і ті ж самі компоненти HTML-коду можна повторно використовувати в різних мобільних операційних системах.

Кожна мобільна програма вирізняється своїм функціоналом, а також своєрідними задачами, які вона повинна вирішувати, тому існує поділ на різноманітні типи за призначенням, задля полегшення вибору кінцевим користувачам. Серед мобільних додатків вирізняють такі типи як ігрові, застосунки для бізнесу чи продуктивності, освітні, а також застосунки для подорожей.

Ігрові додатки. Ігри є однією з найбільш популярних категорій мобільних додатків. Ці типи програм популярні серед розробників, оскільки вони дозволяють

збільшувати трафік відвідувань користувачами, повертаючи їх кожного дня до продовження захоплюючої гри. Компанії зазвичай вкладають більше ресурсів і часу в розробку ігор, оскільки це надзвичайно прибутковий ринок.

Застосунки для бізнесу та продуктивності. Сьогодні ці програми займають значну частину ринку, оскільки люди все частіше використовують свої смартфони та планшети для виконання багатьох складних завдань на ходу. Наприклад, програми можуть допомогти їм забронювати квитки, надсилати електронні листи чи відстежувати хід роботи. Бізнес-додатки спрямовані на підвищення продуктивності та мінімізацію затрат часу, оскільки вони дозволяють користувачам виконувати широкий спектр завдань.

Освітні програми. Ця категорія включає портативні програми, які допомагають клієнтам отримувати нову інформацію в процесі навчання. Наприклад, програми для вивчення мови стали надзвичайно популярними, оскільки вони дають клієнтам саме ті функції, які необхідні для якісного вивчення обраної мови. Численні освітні програми користуються популярністю і серед педагогів, які використовують їх, щоб покращити свій навчальний процес. Важливість та цінність даних додатків важко переоцінити.

Програмні рішення для подорожей. Основна ідея цієї класифікації полягає в тому, щоб допомогти клієнтам подорожувати без будь-яких проблем. Додатки для подорожей перетворюють смартфон на путівник, який допомагає користувачам дізнатися все, що їм потрібно знати про конкретне місце, країну чи місто яке вони відвідують. Дані рішення націлені на мандрівники, які подорожують та дозволяють дізнатись про наявність певного місця.

Об'єктом розробки дипломного проекту є мобільний застосунок на основі операційної системі Android, який забезпечує доступ до інформації про готелі та ресторани в залежності від вибраної країни та міста. Даний додаток буде містити зручну форму логіну та реєстрації, що дозволить користувачам створювати власні аккаунти за допомогою яких у них буде можливість доступу до вищезгаданої інформації про конкретний вид місць.

Зберігання інформації буде виконуватись за допомогою бази даних на віддаленому сервері. Доступ до неї буде здійснюватися через мережу Інтернет шляхом відсилання HTTP запитів, на які будуть відправлятись HTTP відповіді.

Розробка додатку планується у вигляді двох частин: клієнтської та серверної.

В ході опису предметної області були згадані базових понять, які будуть фігурувати впродовж процесу розробки. Даний опис є лише частковим, оскільки основні поняття та способи реалізації окремих компонентів будуть уточнюватись більш детально та деталізуватись в ході роботи на дипломним проектом.

1.2 Огляд існуючого програмного забезпечення

Мобільні програмні засоби, що призначені для пошуку місць відпочинку, які виконують функції своєрідних інформаційних путівників набувають великої популярності, оскільки подорожі є широко поширеними серед людей в усьому світі. Вони допомагають отримувати інформацію у швидкому та зручному вигляді для користувачів в будь-якому місці планети, оскільки їх можна брати з собою.

На сьогодні існує величезна кількість застосунків, що мають на меті полегшення доступу до інформації про місця відпочинку, що цікавлять користувачів. Як правило, дані додатки мають не лише мобільну версію, а й десктопну задля охоплення великої частини користувачів.

1.2.1 TripAdvisor

TripAdvisor – найбільший у світі туристична платформа. Додаток містить мільйони відгуків мандрівників, фотографій та інформації про різноманітні місця. Завдяки цьому він став надійним супутником у подорожах. Крім того, завдяки TripAdvisor мандрівникам буде легше знаходити авіакомпанії, які пропонують найнижчі авіаквитки, найкращі готелі та курорти, чудові ресторани, а також цікаві розваги. Більше того, забронювати улюблені готелі, ресторани та авіаквитки можна

лише одним дотиком. Споживачі все частіше покладаються на додатки, які містять відгуки, оскільки це допомагає визначати напрямки для подорожей, та допомагають зорієнтуватись у якому саме готелі зупинитися, та де смачно поїсти тощо. Користувачам корисно знати про досвід інших людей і на основі відгуків інших людей планувати подорожі.

Зовнішній вигляд мобільного додатку подано на рисунку 1.1.

Рисунок 1.1 – Інтерфейс мобільного додатку TripAdvisor

Основними характерними особливостями TripAdvisor є:

– можливість бронювати готелі та ресторани, а також оглядати велику кількість різноманітних місць;

– безкоштовний доступ;

– існує журнал для додавання основних моментів з подорожей, таких як

записи та фотографії;

– функція порівняння цін на авіаквитки;

– безкоштовний доступ;

– функція оренди автомобілів;

– здатність до тримання інформацію про місця які оточують користувача в даний момент часу за допомогою функцій аналізу геолокації.

На тлі великої кількості функціоналу існують також певні недоліки, які слід враховувати при користуванні:

– не зовсім досконала система бронювання;

– можливість залишати відгуки користувачам, які не були зареєстровані в системі, а також відсутність перевірки та відбору написаних відгуків;

– є однією з найбільш популярних платформ на тлі інших додатків, що не дозволяє користувачам ширше використовувати інші рішення, що пропонують схожий функціонал.

1.2.2 Yelp

Yelp – є популярним сервісом для пошуку місцевих підприємств, починаючи від барів, ресторанів та кафе до перукарень, салонів та заправних станцій. Спочатку Yelp створювався як веб-сайт, який дозволяв людям взаємодіяти з кулінарними закладами, ставлячи їм оцінку від 1 до 5 зірок та мати можливість писати відгуки, які включали обслуговування і якість їжі. З моменту свого запуску в 2004 році Yelp перетворився на програму для мобільного пристрою, а в даний час пропонує підтримку навіть для Apple Watch. За допомогою додатку є можливість не лише оцінювати ресторани та інші заклади, такі як перукарні, заправні станції, аптеки тощо, але також робити замовлення і створювати закладки для збереження місць. Кожен обліковий запис Yelp має список друзів, який можна заповнити, підключивши додаток до Facebook і адресну книгу смартфона.

Основні особливості, якими володіє Yelp:

– величезна кількість відгуків;

– безкоштовний доступ до базових функцій пошуку;

– хороша загальна швидкодія застосунку;

– доволі велика кількість постійних відвідувачів.

До визначних недоліків Yelp входять:

– приховування правдивих відгуків задля забезпечення хорошого статусу компаніям, які купують послуги реклами;

– укладення контракту з іншими компаніями на строк мінімум в один рік;

– високі ціни на платні функції, які дозволяють просувати конкретні місця та послуги вище за інші в рейтингу результатів видачі.

Інтерфейс мобільного додатку подано на рисунку 1.2.

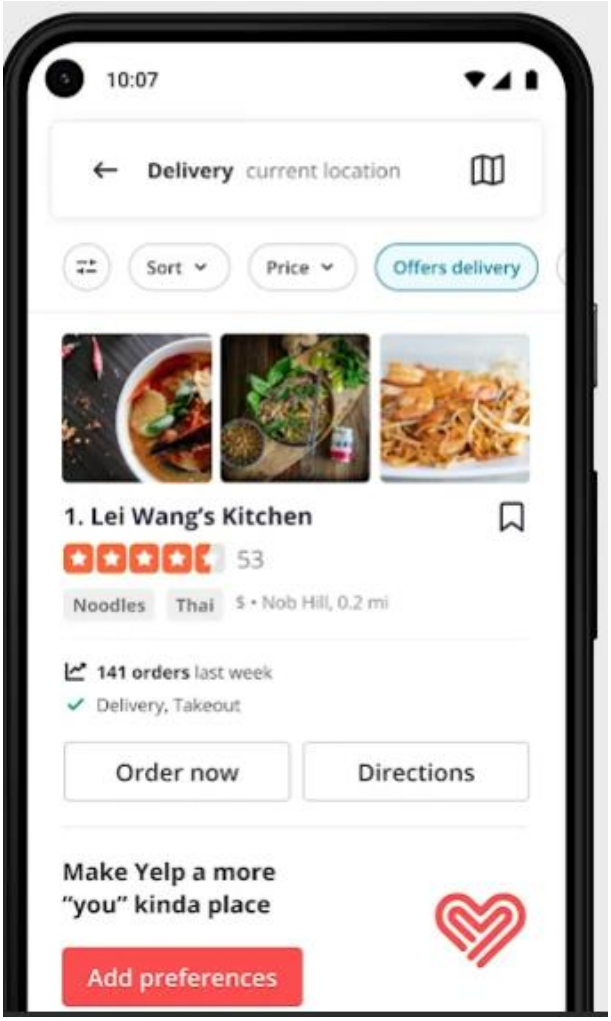


Рисунок 1.2 – Інтерфейс мобільного додатку Yelp

1.2.3 Foursquare

Foursquare — це веб-сайт і мобільний додаток, який дозволяє шукати інформацію та оглядати об’єкти у поточному куточку планети. Також додаток має функцію запам’ятовування вподобань користувачів задля передбачення місць в яких хоче побувати юзер. Foursquare дозволяє користувачам вводити як місто, так і ключові слова, пов’язані з пошуком. Користувачі можуть відфільтрувати результати пошуку та прочитати відгуки про них, щоб обрати місце для відвідування. Додаток також почне рекомендувати вам місця на основі попередніх пошуків. Foursquare був випущений у 2009 році та представлений у Нью-Йорку. Він планувався як додаток для реєстрації, де користувачі можуть повідомляти іншим, де вони перебувають і що вони роблять. Однак у багатьох інших популярних соціальних мережах таких як наприклад Facebook дана функція уже вбудована, тому Foursquare довелося розширити та вдосконалити свій функціонал, що і було зроблено у 2014 році.

Інтерфейс мобільного додатку подано на рисунку 1.3.

Рисунок 1.3 – Інтерфейс мобільного додатку Foursquare

Характерні особливості Foursquare:

– можливість запам’ятовування попередньо обраних місць користувачами та рекомендування до відвідування схожих місць;

– безкоштовний доступ до функцій пошуку;

– система відгуків, що дозволяє користувачам краще розуміти основні недоліки та переваги конкретного місця;

– система подарунків, яка досягається через фізичні відвідуванням певних місць користувачами додатку.

1.3 Визначення вимог до ПЗ та постановка задачі

1.3.1 Клієнтська частина

На основі аналізу предметної області необхідно виділити основні вимоги до розроблюваного програмного забезпечення.

Для клієнтської сторони додатку, що відповідає за візуальний інтерфейс користувача можна виділити наступні функціональні вимоги:

– забезпечення інтуїтивно-зрозумілого інтерфейсу користувача;

– наявність форми реєстрації;

– наявність форми логіну;

– забезпечення відображення інформації за вибором про готелі та ресторани;

– безперебійна робота у нормально режимі функціонування;

– потужна валідація введених даних користувачами на формах реєстрації та логіну, задля уникнення некоректно введених даних;

– інтерфейс додатку, який побудований з використанням англійської мови;

– наявність акаунту для зареєстрованих користувачів;

– функція видалення аккаунту;

можливість зміни даних, які були введені користувачем при реєстрації.

Основною вимогою для роботи месенджера є наявність стійкого доступу до

ресурсів мережі Інтернет.

Написання клієнтської частини планується з використанням мови програмування високого рівня Java, яка є доволі популярною у розробці мобільних застосунків для операційної системи Android. Важливо дотримуватись вищенаведених вимог, оскільки будь-яке відхилення від них може негативно вплинути на загальний досвід користувача при використанні мобільного додатку. Програмне забезпечення призначене для зручного та ефективного відображення потрібної інформації користувачам без зайвих функцій. В майбутньому планується розширення додатку за рахунок реалізації нових функцій, що націлені на покращення загального вигляду, швидкодії, а також на залученні більшої кількості користувачів.

1.3.2 Серверна частина

Для того, щоб забезпечити зберігання даних, а також реалізацію мережевого архітектурного підходу REST, було прийняте рішення для розробки серверної частини, яка має на меті забезпечувати та ефективно поєднувати даний функціонал.

Слід виділити основні функціональні вимоги, якими повинна володіти серверна частина розроблюваного Android додатку:

– забезпечення роботи з базою даних;

– перевірка коректності сформованої URL-адреси при її отриманні задля запобігання помилок у роботі;

– забезпечення базової авторизації, що забороняє відсилати запити не довіреним користувачам;

– відправка HTTP відповідей у форматі JSON про помилку або успіх клієнту на його вхідний HTTP запит;

– якісне зберігання інформації шляхом використання логічної структури таблиць бази даних та правильно побудованих зв’язків між ними.

Написання серверної частини планується з використанням мови

програмування високого рівня PHP. Розробка, тестування, а також демонстрація додатку на етапі написання дипломної роботи відбувається з використанням локального серверу. Подальша робота над додатком планується після завершення захисту дипломного проекту, тому зараз є не доцільним викладення програмного продукту на сервіс хостингу, що надає доступ до серверів, які дозволяють розміщувати програмні продукти в мережі Інтернет. Сервер на хостингу повинен працювати цілодобово, а також бути захищеним від несподіваних поломок, оскільки проблеми серверної частино будуть прямо відображені на клієнтській, що призведе до неможливості реєстрації та авторизації юзерів.

Гадаю, що були згадані усі головні функціональні вимоги для обох незалежних частин мобільного додатку, що дозволить якісно проводити проектування та програмну реалізацію програми.

1.3.3 Моделі варіантів використання

Діаграми варіантів використання моделюють функціональність системи за допомогою акторів та елементів що мають назву варіанти використання. Варіанти використання — це набір певних дій, послуг і функцій, які система повинна виконувати. Актори — це люди або сутності, які виконують певні ролі в системі. Діаграми варіантів використання є цінними для візуалізації функціональних вимог системи, що впливає на пріоритети розробки. Вони також допомагають визначити будь-які внутрішні або зовнішні фактори, які можуть вплинути на систему та повинні бути взяті до уваги. Даний тип діаграм забезпечує хороший аналіз системи, що розробляється. Діаграми варіантів використання визначають, як система взаємодіє з акторами, не концентруючись на деталях того, як ця функціональність реалізована всередині.

Задля кращого унаочнення необхідно побудувати діаграму варіантів використання на якій будуть відображені основні актори, а також варіанти використання, які призначені для відображення основних функцій додатку.

Діаграма варіантів використання зображена на рисунку 1.4.

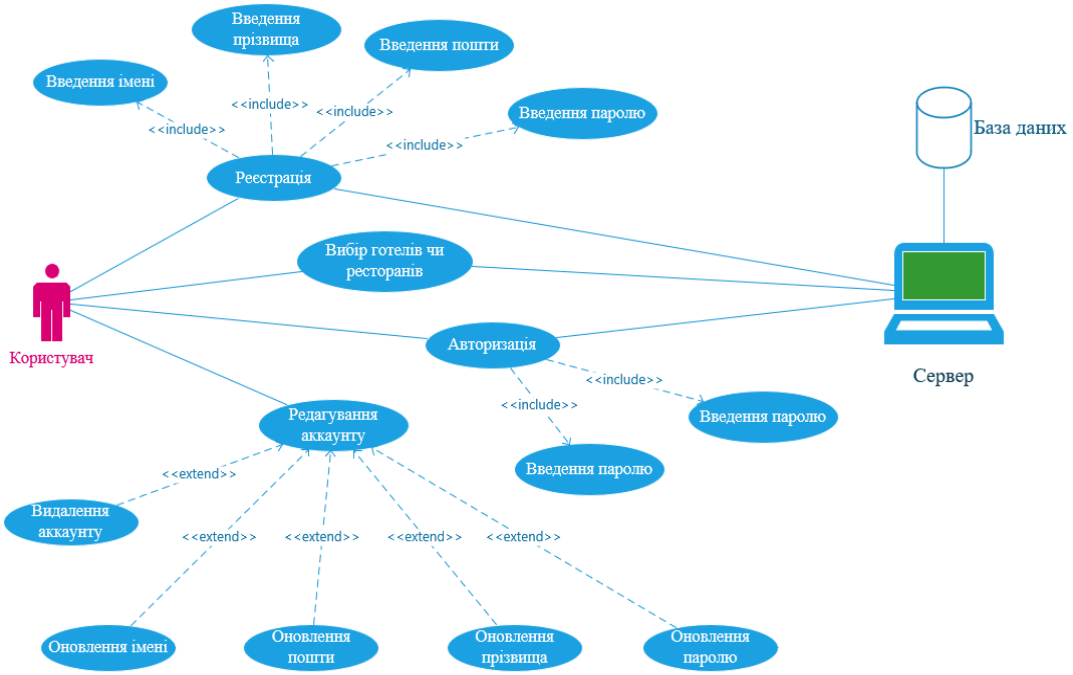


Рисунок 1.4 – Діаграма варіантів використання

Необхідно провести детальний опис основних елементів що зображені у вищенаведеній діаграмі, щоб краще показати наявні зв'язки між варіантами використання, а також подати короткий опис. Потрібно почати опис із основних акторів. Опис акторів приведений у таблиці 1.1.

Таблиця 1.1 Опис головних акторів

Актор	Короткий опис
Сервер	Забезпечує авторизацію та реєстрацію користувачів, займається оновленням даних зареєстрованих користувачів, а також виводить результати про готелів та ресторани, отримує та обробляє запити від користувачів і відсилає відповідні відповіді.
Користувач	Здійснює реєстрацію, авторизацію, оновлення даних всередині акаунту

Слід навести усі варіантів використання, а також вказати зв’язки, які містять у кожного з них з іншими компонентами системи. Існує три типи зв’язків, які широко використовуються при розробці діаграм варіантів використання, а саме: асоціація, включення та розширення. Усі варіанти використання, а також їх зв’язки подані у таблиці 1.2.

Таблиця 1.2 – Перелік варіантів використання та їх зв’язків

Варіант використання	Тип зв’язку
Реєстрація	Асоціація, Включення
Редагування аккаунту	Асоціація, Розширення
Авторизація	Асоціація
Вибір готелів чи ресторанів	Асоціація
Оновлення імені	Розширення
Оновлення прізвища	Розширення
Оновлення пошти	Розширення
Оновлення паролю	Розширення
Введення імені	Включення
Введення прізвища	Включення
Введення пошти	Включення
Введення паролю	Включення

Далі потрібно провести загальний опис поточних варіантів використання, задля кращого опису їх функцій. У кожній з наступних таблиць будуть вказуватись зв’язки поточного варіанту використання з іншими елементами діаграми, а також короткий опис конкретного варіанту використання. Ці характеристики є головними в розумінні варіантів використання.

Детальний опис варіанту використання «Реєстрація» поданий у таблиці під номером 1.3 знизу.

Таблиця 1.3 Варіант використання «Реєстрація»

Назва	Реєстрація
Короткий опис	Користувач проводить реєстрацію у системі, після чого відбувається запит до сервера, де відбувається збереження даних, якщо вони були введені вірно.
Зв’язок з іншими варіантами використання та акторами	«Реєстрація» містить зв’язок включення з наступними варіантами використання: «Введення імені», «Введення прізвища», «Введення пошти», «Введення паролю». «Реєстрація» містить зв’язок асоціації з актором «Користувач».

Детальний опис варіанту використання «Авторизація» поданий у таблиці під номером 1.4 знизу.

Таблиця 1.4 Варіант використання «Авторизація»

Назва	Авторизація
Короткий опис	Користувач проводить авторизацію у системі, після чого відбувається запит до сервера, де відбувається перевірка на існування користувача, якщо вони були введені коректно.
Зв’язок з іншими варіантами використання	«Авторизація» містить зв’язок включення з наступними варіантами використання: «Введення пошти», «Введення паролю». «Авторизація» містить зв’язок асоціації з актором «Користувач».

Наступний варіант використання представляє подію, при якій користувач виконує пошук. Пошук може проводитись з метою отримання даних про готелі чи

ресторани. Детальний опис варіанту використання «Вибір готелів чи ресторанів» поданий у таблиці під номером 1.5 знизу.

Таблиця 1.5 Варіант використання «Вибір готелів чи ресторанів»

Назва	Вибір готелів чи ресторанів
Короткий опис	Користувач проводить вибір готелів чи ресторанів, після чого відбувається запит на сервер, який повертає результати пошуку.
Зв'язок з іншими варіантами використання	«Вибір готелів чи ресторанів» містить зв'язок асоціації з актором «Користувач».

Детальний опис варіанту використання «Редагування аккаунту» поданий у таблиці під номером 1.6 знизу.

Таблиця 1.6 Варіант використання «Редагування аккаунту»

Назва	Редагування аккаунту
Короткий опис	Зареєстрований користувач проводить зміну даних, які були введені на етапі реєстрації аккаунту, після чого відбувається запит на сервер.
Зв'язок з іншими варіантами використання	«Редагування аккаунту» містить зв'язок розширення з наступними варіантами використання: «Оновлення імені», «Оновлення прізвища», «Оновлення пошти», «Оновлення паролю», «Видалення аккаунту». «Редагування аккаунту» містить зв'язок асоціації з актором «Користувач».

В даному розділі був проведений аналіз предметної області, а також наведено ряд базових понять, що будуть використовуватись впродовж подальшої роботи над програмним продуктом. Були наведені існуючі програмні рішення, що пропонують

схожу функціональність у порівнянні із розроблюваним мобільним застосунком, виділені їх основні характеристики. Вказано перелік вимог, яким повинні задовільняти клієнтська та серверна частини, а також складено діаграму варіантів використання задля кращої візуалізації процесів, що протікають у програмі. На основі виокремлених вимог, було розроблене технічне завдання, яке подано у додатку А.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Аналіз та проектування архітектури системи

Мобільний додаток розробляється із використанням двох незалежних частин, якими є клієнт, що допомагає відображати візуальний інтерфейс користувача, а також сервер, що призначений для надання інтерфейсу клієнту через який проводиться спілкування між ними. Дані між клієнтом та сервером будуть передаватись через веб-мережу у форматі JSON. Також, на стороні серверу міститься база даних, яка виступає сховищем даних, дані якої повертаються клієнтам або модифікуються відповідно до певних HTTP запитів. Задля реалізації даної поведінки, буде використана саме клієнт-серверна архітектура.

Клієнт-серверна архітектура – це один із видів архітектурних підходів, що використовується при проектуванні програмного забезпечення. В даній архітектурі виділяють два основних поняття якими є клієнт та сервер. Сервер - це комп'ютер, який приймає HTTP-запити від клієнтів і видає HTTP відповіді. Клієнт – це певний програмний застосунок, який надсилає запити на сервер для виконання конкретних завдань або надання конкретної інформації. Найчастіше клієнтом є браузер. Отримана відповідь у вигляді розмітки сайту та іншої інформації дозволяє браузеру відображати елементи інтерфейсу користувача. Спілкування базується на використанні мережевого протоколу HTTP.

HTTP – це формально визначений набір правил для зв’язку між клієнтом (мережевим ресурсом, який запитує дані або послуги) і сервером (ресурсом, який отримує запит і відповідає на нього). Можна сказати, що Інтернет – це інфраструктура, яка призначена для з’єднання веб-клієнтів та серверів, а HTTP є у свою чергу «мовою», на якій вони спілкуються один з одним через це з’єднання. Стандартизовані протоколи комп’ютерної мережі гарантують, що обладнання та програмне забезпечення, вироблене різними постачальниками, можуть надійно працювати разом. Протокол HTTP визначає базові правила для запитів ресурсів і відповідей між веб-клієнтами та серверами. Тіло містить інформацію, яка буде

передана на сервер задля її додавання або видалення. Такою інформацією може бути ім’я та прізвище користувача, пароль, або будь-які інші дані.

Кожне HTTP повідомлення складається із заголовків, а також тіла. Заголовки містять текстову інформацію, що зберігається у вигляді ключ-значення. Вони включаються в кожен запит і відповідь. Ці заголовки передають основну інформацію, наприклад, який браузер використовує клієнт, які дані запитуються та у якому форматі.

HTTP протокол визначає ряд методів, які використовуються клієнтами для спілкування із серверами. Слід навести найбільш часто використовувані з них:

– GET;

– HEAD;

– POST;

– PUT;

– DELETE.

GET метод використовується для отримання інформації з даного сервера за допомогою конкретної URI адреси. Даний тип запитів повинен лише отримувати дані, але не модифікувати їх.

HEAD метод є ідентичний по структурі та призначенню із методом GET. Єдина різниця полягає у тому, що у HTTP відповіді на даний метод не міститься тіла з інформацією.

POST метод використовується для надсилання даних на сервер. Найчастіше, даний метод використовується для створення нового ресурсу. Прикладом може слугувати створення нового юзера при реєстрації, додавання відгуку під статтею.

DELETE метод використовується задля видалення ресурсу. Зазвичай доступ до ресурсів здійснюється через вказування конкретного ідентифікатору.

PUT метод використовується для оновлення ресурсу.

Також, за допомогою HTTP протоколу можна відправляти та отримувати дані у таких популярних форматах як JSON і XML.

XML є мовою розмітки, подібною до HTML, але без попередньо визначених тегів. Потрібно визначати власні теги. Даний формат є доволі зручним в плані зберігання та пошуку інформації. Оскільки формат XML стандартизований, то будь-який отримувач чи сервіс зможе з легкістю розібрати його вміст. Для перевірки правильності формування XML використовуються XSD файли, які додатково підключаються до файлів даного формату.

JSON є форматом, який широко використовується при передачі різноманітних даних. Даний формат був побудований на основі синтаксису мови JavaScript. Це дозволило використовувати такі добре відомі структурні елементи як базові скалярні типи, масиви та об’єкти. Усі ці елементи є ключовими поняттями для більшості мов програмування, тому задача швидкого орієнтування в JSON досягається дуже легким шляхом. Через це, даний формат має кращий синтаксис в порівнянні із XML, що дозволяє його легко читати та швидко розуміти. Дані зберігаються у вигляді ключ-значення. Даний формат є доволі простим у розумінні.

Необхідно визначити спосіб взаємодії клієнтської та серверної частин. Це можна зробити з використанням відомого архітектурного підходу, який саме призначений для стандартизації, або простіше кажучи для опису взаємодії клієнта із сервером, що має назву REST API.

REST API – це архітектурний підхід, що описує правила взаємодії між клієнтом та сервером. REST використовує протокол HTTP для передачі даних через мережу. Це дозволяє використовувати такі найбільш поширені HTTP методи як POST, PUT, DELETE та GET. Дані методи відповідають базовим операціями по створенню, оновленню, видалення та доступу до даних у базі даних. У REST архітектурі використовуються такі формати для передачі даних як XML та JSON.

Для взаємодії надаються інтерфейси, а саме URI адреси, через які відбувається доступ клієнта до даних на віддаленому сервері. Це дозволяє надзвичайно спростити налагодження комунікації, оскільки потрібно лише знати шлях, тобто адресу куди потрібно відсилати дані, а також формат даних, що відсилаються.

Базова структура мобільного додатку базується на клієнт-серверній архітектурі з використанням архітектурного підходу REST API, що описує спосіб взаємодії клієнтської та серверної частин. Схема спілкування основних модулів програми зображена на рисунку 2.1.

Рисунок 2.1 – Схема спілкування клієнта та сервера

2.2 Проектування структури бази даних

База даних являє собою організовану колекцію структурованих даних, що робить її легко доступною, керованою та оновленою. Простими словами, можна сказати, що база даних є місцем де зберігаються дані. Прикладом може слугувати бібліотека. Бібліотека містить величезну колекцію книг різних жанрів, тут бібліотека – це база даних, а книги – це дані. Існують різноманітні типи баз даних, але виділяють три основних, а саме:

– об’єктно орієнтовані бази даних;

– NoSQL бази даних;

– реляційні бази даних.

Об’єктно орієнтовані бази даних забезпечують зберігання даних у вигляді об’єктів та класів. Об’єкти є реальними сутностями, а класи є колекціями об’єктів. Даний тип баз даних є поєднанням особливостей реляційної моделі з об’єктно-орієнтованими принципами. Це альтернативна реалізація реляційної моделі. Кожна таблиця є відображенням реального класу певної сутності у програмному коді, що володіє певними атрибутами. Існує рівень об’єктно-реляційного відображення,

який відображає схеми бази даних з об’єктами в коді. Зчитування та зіставлення даних бази даних з об’єктами у коді відбувається напряму, без сторонніх засобів. Це допомагає отримувати швидший доступ до даних і кращу продуктивність роботи в порівнянні з іншими типами баз даних.

NoSQL є не табличними базами даних і зберігають дані інакше, ніж реляційні бази даних. Взагалі термін NoSQL означає «не тільки SQL», характеризуючи відгалуження від традиційного підходу до проектування баз даних. Бази даних NoSQL бувають різних типів на основі конкретної моделі даних. Основними типами є документ, ключ-значення, широкий стовпець і граф. Даний тип баз даних забезпечує гнучкі схеми і легке масштабування з великими обсягами даних та оптимізацію високих навантажень з боку клієнтів. Спеціально призначені для додатків, які мають швидко, з низькою часовою затримкою обробляти великий обсяг даних із різною структурою.

Третім типом є реляційні бази даних, які були обрані в якості розробки бази даних для мобільного додатку, що містяться на віддаленому сервері.

Реляційна база даних представляє та зберігає дані у вигляді набору таблиць і надає способи для маніпулювання цими даними. Зберігання даних в таблицях забезпечує ефективний, інтуїтивно зрозумілий і гнучкий спосіб доступу до структурованої інформації.

Таблиці, також відомі як відношення, складаються зі стовпців, які називають атрибутами, що відповідають певним характеристикам або категоріям даних сутності, і рядків, також відомих як записи таблиці, що містять набір даних, які визначені атрибутами. Програми отримують доступ до даних виконуючи відповідні запити.

Спілкування із сутностями бази даних відбувається за допомогою структурованої мови запитів SQL, що широко використовується у базах даних, що базуються на реляційному підході.

Існують три види зв’язку, які використовуються для формування залежностей між сутностями реляційної бази даних. Виділяють три види:

– один-до-одного, що використовується для позначення зв’язку між одним записом із таблиці А з іншим, який міститься у таблиці Б;

– один-до-багатьох, який використовується коли одному запису із таблиці А повинен відповідати деякий набір записів із таблиці Б;

– багато-до-багатьох, що використовується коли деякому набору записів таблиці А повинен відповідати деякий набір записів із таблиці Б.

Виділяють такі важливі поняття, що відносяться до атрибутів таблиці як первинний та зовнішній ключ.

Унікальний ключ є атрибутом або колекцією атрибутів, що однозначно ідентифікують кожен запис відношення. Позначаються на схемах вони у вигляді PK. Первинні ключі можуть бути простими або складеними. В першому випадку до первинного ключа входить лише один атрибут, а у другому випадку первинний ключ включає більше ніж один атрибут.

Зовнішній ключ використовується для зв’язку сутностей в межах однієї бази даних. За допомогою нього відбується формування зв’язків між таблицями. На діаграмах та схемах його позначають у вигляді FK.

Реляційні бази даних використовують такий підхід до проектування відношень як нормалізація.

Нормалізація є процесом організації даних у базі даних. Це включає створення таблиць і встановлення зв’язків між цими таблицями згідно з правилами, які розроблені для захисту даних, та для забезпечення більшої гнучкості бази даних, усуваючи надмірність і непостійну залежність. Нормалізація допомагає уникнути різноманітних аномалій оновлення, видалення та вставки. Ці аномалії виникають в результаті зберігання різнорідної інформації в межах однієї таблиці.

Всього існує більше шести нормальних форм на основі яких проектуються відношення у базі даних. При проектуванні були використані лише три перших, оскільки їх є достатньо для ефективного усунення вищезгаданих аномалій.

Відношення знаходиться у першій нормальній формі, якщо кожен запис є унікальним та усі значення є атомарними.

Відношення знаходиться у другій нормальній формі, якщо відношення знаходиться у першій нормальній формі, а також усі не ключові атрибути залежать від усього первинного ключа, а не від його частини.

Відношення знаходиться у третій нормальній формі, якщо воно знаходиться у другій нормальній формі і усі не ключові атрибути залежать від первинного ключа, а не від інших не ключових атрибутів.

Усі відношення містять атомарні значення та записи, які унікально ідентифікуються первинним ключем. Між не ключовими атрибутами не має зв’язків. На основі даних висновків, можна вважати, що відношення знаходяться у третій нормальній формі.

Для розроблюваного додатку було виділено п’ять основних відношень, що виступають у ролі таблиць, а саме «Користувач», «Країна», «Місто», «Тип місця», а також «Місце». Слід провести короткий опис кожного з них, вказати їх атрибути та виділити типи атрибутів. Я наведу поділ на такі типи як первинний ключ, зовнішній ключ та звичайний атрибут, що буде позначений прочерком.

Відношення «Користувач» використовується для опису користувачів системи, а саме їх основних характеристик. Дане відношення містить такі атрибути як «Ідентифікатор», «Ім’я», «Прізвище», «Пошта», «Пароль», «Дата оновлення запису», а також «Дата створення запису». Таблиця «Користувач» не має зв’язків з іншими таблицями. Атрибути даного відношення окремо подані у таблиці під номером 2.1.

Таблиця 2.1 – Відношення «Користувач»

Назва відношення	Назва атрибуту	Тип атрибуту
Користувач	Ідентифікатор	Первинний ключ
	Ім’я	-
	Прізвище	-
	Пошта	-
	Пароль	-

Закінчення таблиці 2.1

	Дата оновлення запису	-
	Дата створення запису	-

Відношення «Країна» використовується для визначення країн, які будуть використовуватись при пошуку місць відпочинку. У кожній країні міститься певний набір міст, які зберігаються у відношенні «Місто». Дане відношення складається із таких атрибутів як «Ідентифікатор» та «Назва». Атрибути даного відношення подаються у таблиці під номером 2.2.

Таблиця 2.2 – Відношення «Країна»

Назва відношення	Назва атрибуту	Тип атрибуту
Країна	Ідентифікатор	Первинний ключ
	Назва	-

Відношення «Місто» позначає міста, які будуть використовуватись при пошуку місць відпочинку. Відношення місить такі атрибутів як «Ідентифікатор», «Назва» та «Ідентифікатор країни». «Ідентифікатор країни» використовується для позначення зв’язку один-до-багатьох між відношеннями «Країна» та «Місто». Атрибути даного відношення окремо наведені у таблиці під номером 2.3.

Таблиця 2.3 – Відношення «Місто»

Назва відношення	Назва атрибуту	Тип атрибуту
Місто	Ідентифікатор	Первинний ключ
	Назва	-

Відношення «Тип місця» призначене для виділення основних типів місць, які використовуються у додатку, якими на даний момент є готелі та ресторани. Дане

відношення містить мінімальну інформацію, а саме назву типу місця. Таблиця «Тип місця» має такі атрибути як «Ідентифікатор» та «Назва». Атрибути даного відношення окремо подані у таблиці під номером 2.4.

Таблиця 2.4 – Відношення «Тип місця»

Назва відношення	Назва атрибуту	Тип атрибуту
Тип місця	Ідентифікатор	Первинний ключ
	Назва	-

Останнє відношення «Місце» описує основні характеристики, якими повинні володіти готелі та ресторани. Даними характеристиками є «Ідентифікатор», «Ідентифікатор міста», «Ідентифікатор типу місця», «Назва», «Опис», «Телефон», «Робочі години», «Адреса», «Картинки», «Дата оновлення запису», а також «Дата створення запису». Атрибути «Ідентифікатор міста» та «Ідентифікатор типу місця» використовуються для встановлення зв’язків один-до-багатьох між відношеннями «Місто» та «Тип місця» відповідно. Атрибути даного відношення окремо подані у таблиці під номером 2.5.

Таблиця 2.5 – Відношення «Місце»

Назва відношення	Назва атрибуту	Тип атрибуту
Місце	Ідентифікатор	Первинний ключ
	Ідентифікатор міста	Зовнішній ключ
	Ідентифікатор типу місця	Зовнішній ключ
	Назва	-
	Опис	-
	Телефон	-
	Робочі години	-
	Адреса	-
	Картинки	-

Закінчення таблиці 2.5

	Дата оновлення запису	-
	Дата створення запису	-

Задля кращого зображення відношень слід створити фізичну модель бази даних, що допомагає зображати реальний вигляд відношень у базі даних, їх зв'язки, а також внутрішню структуру. Дана модель була побудована за допомогою інструментів середовища PhpStorm в якому відбувалась розробка серверної частини додатку. Дана модель подана на рисунку 2.2.

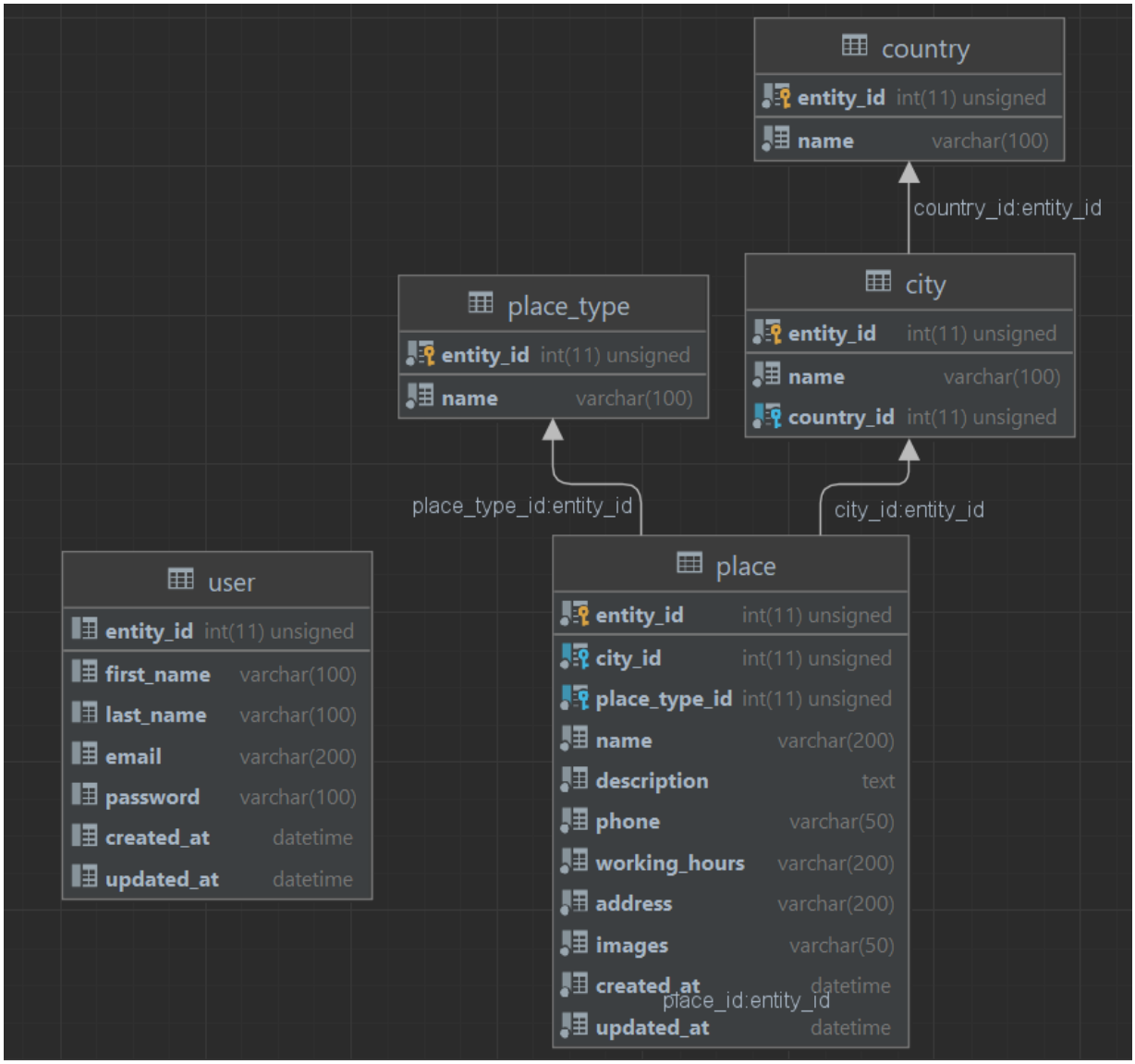


Рисунок 2.2 – Фізична модель бази даних

2.3. Проектування архітектури серверної частини додатку

Серверна частина розробляється з використанням мови програмування високого рівня PHP. Дана мова спеціально призначена для розробки програмного забезпечення на сервері.

Як уже зазначалось, задля опису спілкування між клієнтом та сервером використовується архітектурний підхід REST API. Дані у мобільному додатку передаються у форматі JSON. У даному підході ключову роль відіграють ідентифікатори ресурсу, які визначають інтерфейси для взаємодії. Ідентифікаторами ресурсу виступають частини URL адрес, які йдуть після домену. Увесь трафік надходить до файлу «index.php». У моєму програмному продукті він слугує початковою точкою через який проходить кожен запит зі сторони клієнта.

У додатку використовується одинадцять ідентифікаторів ресурсу, які закріплені до конкретного HTTP методу. Для кожного ідентифікатору ресурсу визначений клас репозиторію та метод даного репозиторію, який буде викликаний при звертанні до даного ресурсу. Кожен ідентифікатор ресурсу відповідає конкретній сутності у базі даних, тому перша частина ресурсу містить ім’я відношення, що відноситься до даного ресурсу. Для деяких ресурсів, вкінці можна побачити позначення «:id». Це означає, що на цьому місці повинене бути ідентифікатор ресурсу, у загальному даним ідентифікатором виступає ключовий атрибут або зовнішній ключ відношення.

Для HTTP методу POST існує один ресурс, який має назву «/user». Він призначений для створення нових користувачів при реєстрації аккаунту. Тобто при створені нового користувача буде відбуватись HTTP запит, який буде додавати нові дані до відношення «Користувач».

Для HTTP методу GET існує вісім ідентифікаторів ресурсу. Для відношення «Користувач» є такі ресурси як «/user/:id», «/user», «/user/by-email». Вони відповідно використовуються для отримання конкретного користувача, усіх користувачів, а також користувача, який має конкретну пошту. Для останнього

ресурсу, дані про пошту передаються через параметри HTTP запиту. Для відношення «Країна» існує один ресурс під назвою «/country» за допомогою якого можна отримати список країн. Відношення «Місто» також містить один ресурс «/city», що використовується для отримання списку міст. Відношення «Тип місця» має один ресурс «/place-type» за допомогою якого можна отримати список усіх типів місць. Відношення «Місце» має такі ресурси як «/place/hotel/:id» і «/place/restaurant/:id». Вони використовуються відповідно для отримання конкретного готелю та ресторану за допомогою ідентифікатору.

HTTP метод DELETE має лише один ресурс «/user/:id». Він слугує для видалення користувача за його ідентифікатором.

HTTP метод PUT містить ресурс «/user/:id». Він використовується для оновлення користувача за його унікальним ідентифікатором.

Задля кращого представлення усіх ресурсів, які використовуються при спілкуванні клієнта та сервера, варто створити окрему таблицю. Дані про HTTP методи та ідентифікатори ресурсів, що закріплені за ними подані у таблиці 2.6.

Таблиця 2.6 – Інформація про ідентифікатори ресурсів

Назва HTTP методу	Назва ідентифікатору ресурсу
POST	/user
GET	/user/:id
	/user
	/user/by-email
	/country
	/city
	/place/hotel/:id
	/place/restaurant/:id
	/place-type
DELETE	/user/:id
PUT	/user/:id

Головна задача серверної частини – це якісна обробка вхідних запитів та виконання операцій з даними. Операції з даними виконуються шляхом постійних звернень до бази даних. Існує чотири основних види операцій, які можна проводити із даними. Ними є операції створення, видалення, оновлення, а також операції тримання даних. Саме на основі цих принципів проектувалась архітектура серверної частини. Також, були створені окремі класи моделей, які закріплені за певними відношеннями у базі даних. Саме за допомогою цих класів відбувається взаємодія із базою даних. Також, за кожним класом моделі закріплений відповідний клас репозиторію. Репозиторії допомагають абстрагуватись від того як працює система зсередини. Вони надають прозорі інтерфейси у вигляді методів, за допомогою яких виконуються відповідні операції, що так чи інакше взаємодіють з даними всередині бази даних. Усі інші класи виконують допоміжні функції, такі як наприклад операції над файлами чи надання можливості доступу до бази даних. Більш детальний опис класів буде приведений у наступному розділі.

Щоб відобразити основні компоненти серверної частини, якими виступають класи, слід побудувати діаграму класів. На даній діаграмі будуть відображені лише компоненти та зв’язки між ними, без вказання детальної структури, яка складається з полів класу та його методів. Діаграма класів подана на рисунку 2.3.

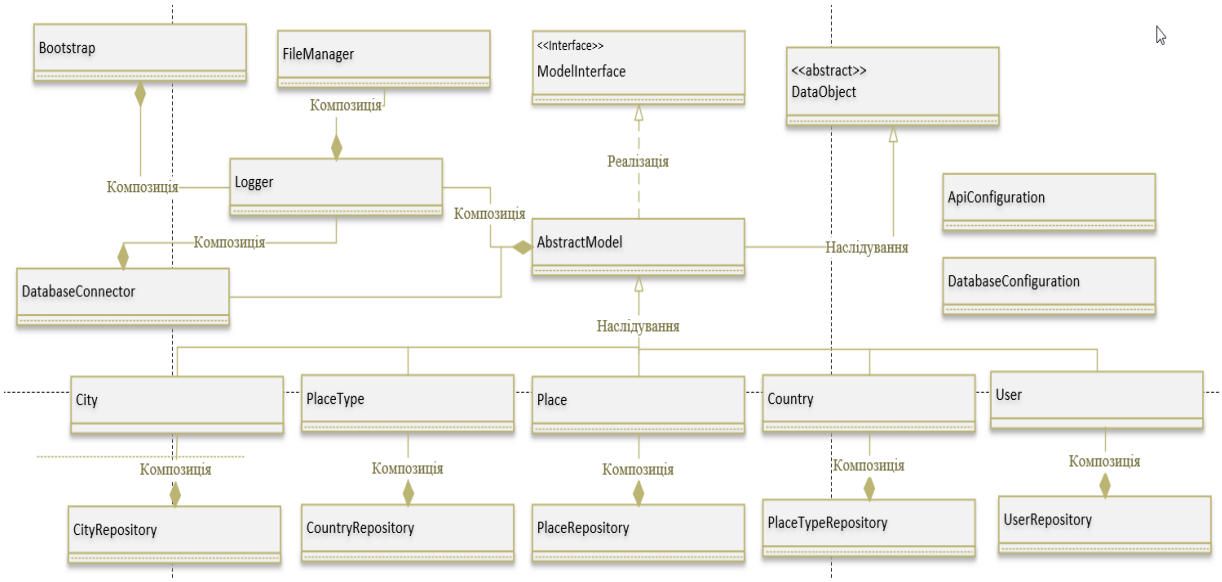


Рисунок 2.3 – Діаграма класів серверної частини

На діаграмі видно, що існує п’ять класів репозиторіїв, які дозволяють взаємодіяти з методами моделей через власні інтерфейси у вигляді методів. Ними є такі класи як CityRepository, CountryRepository, PlaceRepository, PlaceTypeRepository, UserRepository. Кожен репозиторій містить у собі об’єкт відповідного класу моделі, без якого репозиторій не може існувати. Моделей є також п’ять: City, PlaceType, Place, Country, User. Класи ApiConfiguration та DatabaseConfiguration не мають зв’язків з іншими класами. Абстрактний клас AbstractModel містить основні методи по роботі із даними, які визиваються із класів моделей. Усі класи моделей наслідують даний клас, оскільки операції оновлення, видалення, додавання та доступу до даних для моделей є однаковими. Лише деякі моделі додають необхідні методи, якщо в цьому є необхідність.

Інтерфейс ModelInterface визначає основні методи, що використовуються у роботі з базою даних. Клас DatabaseConnector використовується для створення об’єкту з’єднання із базою даних, через який можливе виконання операцій по модифікації даних.

Клас Logger використовується для запису інформації про помилки у спеціально відведений для цього файл. Клас FileManager забезпечує роботу із файловою системою. За допомогою класу Bootstrap виконується запуск виконання основного коду.

2.4. Проектування архітектури клієнтської частини додатку

Клієнтська частина створюється з використанням мови програмування високого рівня Java. Основна логіка роботи клієнтської частини як і серверної виражається через взаємодію класів, а точніше їх об’єктів.

Слід розпочати проектувати архітектури із визначення основних модулів, тобто класів клієнтської частини, а також зв’язків між ними. Необхідно побудувати діаграму класів, що призначена саме для зручного представлення структури класів. Перша частина діаграма класів клієнтської частини подана на рисунку 2.4.

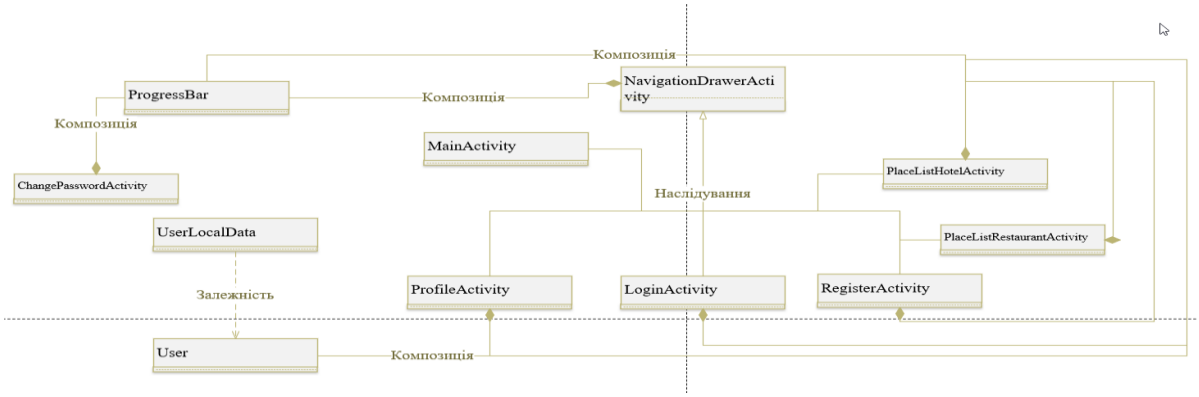


Рисунок 2.4 – Діаграма класів клієнтської частини (перша частина)

Друга частина діаграма класів клієнтської частини подана на рисунку 2.5.

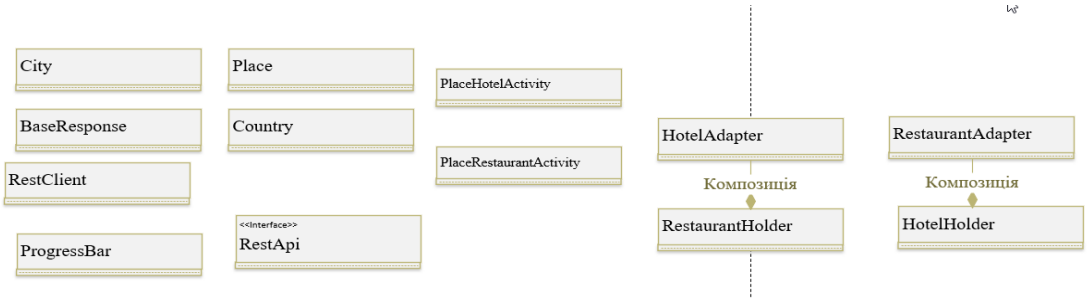


Рисунок 2.5 – Діаграма класів клієнтської частини (друга частина)

На вищенаведених діаграмах були наведені основні компоненти мобільного додатку, які є класами. Класи City, Place, BaseResponse, Country та User є класами моделей і призначені лише для опису структури відношень бази даних. Інтерфейс RestApi містить опис усіх ідентифікаторів ресурсу, за допомогою яких здійснюється спілкування із сервером. Клас RestClient використовується для мережевої взаємодії використовуючи ідентифікатори ресурсу для доступу до серверу. ProgressBar містить функціонал для показу спливаючого повідомлення завантаження, що блокує графічний інтерфейс користувача. Класи PlaceHotelActivity та PlaceRestaurantActivity забезпечують відображення конкретного готелі чи ресторану, який був обраний користувачем із списку. PlaceListHotelActivity та PlaceListRestaurantActivity використовуються для відображення екрану пошуку готелів та ресторанів відповідно, а також

відповідають за виведення місць у вигляді списку. Клас HotelAdapter містить внутрішній клас HotelHolder, а також клас RestaurantAdapter містить внутрішній клас RestaurantHolder. Дані класи адаптерів використовуються для зберігання списку готелів та ресторанів, що були отримані від сервера та підготовці їх до виведення. UserLocalData призначений для тимчасового зберігання даних про зареєстрованих юзерів, тобто виступає у ролі системи кешування. Класи ChangePasswordActivity, ProfileActivity, LoginActivity, RegisterActivity та MainActivity відповідають за виведення екранів зміни паролю, акаунту користувача, логіну, реєстрації та початкового меню. Клас NavigationDrawerActivity забезпечує відображення головного меню, яке доступне усім класам активностей, які наслідують даний клас.

Діаграма класів в повній мірі відображає модулі клієнтської частини та зв'язки між ними. Програмні модулі будуть детальніше розглянуті у наступному розділі дипломної роботи.

2.5 Проектування інтерфейсу користувача

Важливу роль грає грамотна побудова графічного інтерфейсу користувача, де вдало поєднуються елементи та дизайн. Саме зовнішній вигляд є тим, що бачить користувач в перший момент запуску додатка, тому важливо підійти до розробки візуальної частини відповідально.

Інтерфейс додатку розроблений з використанням оранжевого кольору, що використовується у верхній та нижній частині панелі інструментів. Синій колір був обраний для відображення шрифту. Інтерфейс базується в основному на світлій та теплій гаммі. У дизайні не має зайвих деталей, оскільки надмірне перевантаження графічного інтерфейсу різними елементами може справити погане враження на користувача мобільного додатку, оскільки орієнтуватись у функціоналі буде важко.

Однією із найважливіших речей є логотип мобільного додатку. Він допомагає користувачам вирізняти даний програмний продукт з поміж інших. На

логотипі зображені перші букви з яких починаються обидва слова, що використовуються у назві додатку. Він поданий у вигляді своєрідної абревіатури, що допомагає краще асоціювати логотип із назвою додатку. Іконка додатку зображена на рисунку 2.6.

Рисунок 2.6 – Логотип додатку

Головна сторінка відкривається для усіх користувачів по замовчуванню при кожному запуску додатку. Вона містить логотип додатку, а також короткий текст, який призначений для ознайомлення юзерів із функціоналом мобільного застосунку. Вигляд головного екрану поданий на рисунку 2.7.

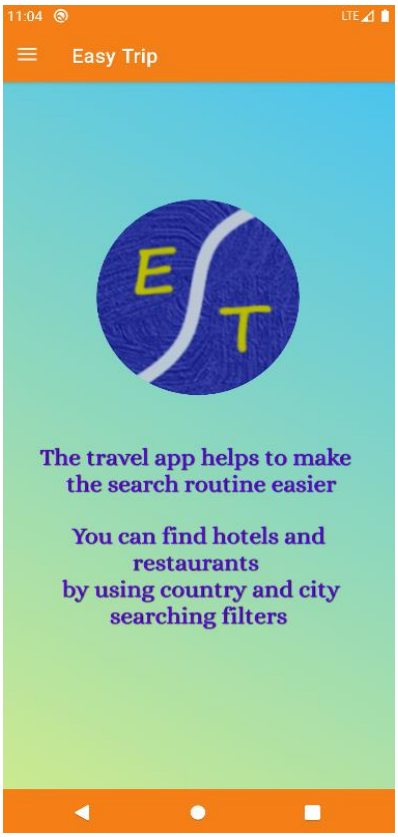


Рисунок 2.7 – Головний екран додатку

Для переходу до інших екранів використовується головне меню, яке можна побачити у верхній лівій частині екрану, що подане у вигляді значку з трьома горизонтальними лініями. Після кліку по навігаційному меню, відкривається меню навігації, що допомагає зручно переходити на інші екрани. Вигляд меню навігації подане на рисунку 2.8.

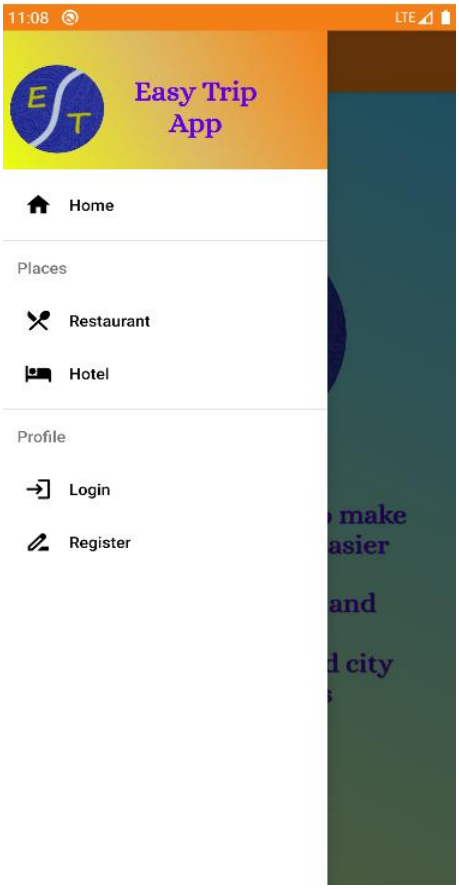


Рисунок 2.8 – Вигляд меню навігації

З меню навігації не зареєстровані користувачі мають змогу відкрити форму логіну чи реєстрації. Фон обох екранів був виконаний за допомогою змішування двох кольорів, а саме світло зеленого та блакитного, що дозволило досягнути доволі приємного візуального ефекту. На екрані логіну зображені елементи для вводу пошти та паролю, а також кнопки підтвердження логіну та переходу на форму реєстрації, якщо користувач бажає створити новий акаунт. Поля для введення даних володіють хорошою валідацією введених даних користувачем.

Паролі повинні містити мінімум одну цифру, букву верхнього та нижнього регістрів. Усі помилки які виникають при не коректному введені даних подаються у лівій частині полів введення. Також, з правої частини містяться діапазони цифр, що показують кількість уже введених символів та максимально дозволену кількість для введення. Екрани логіну та реєстрації подані на рисунках 2.9 та 2.10.

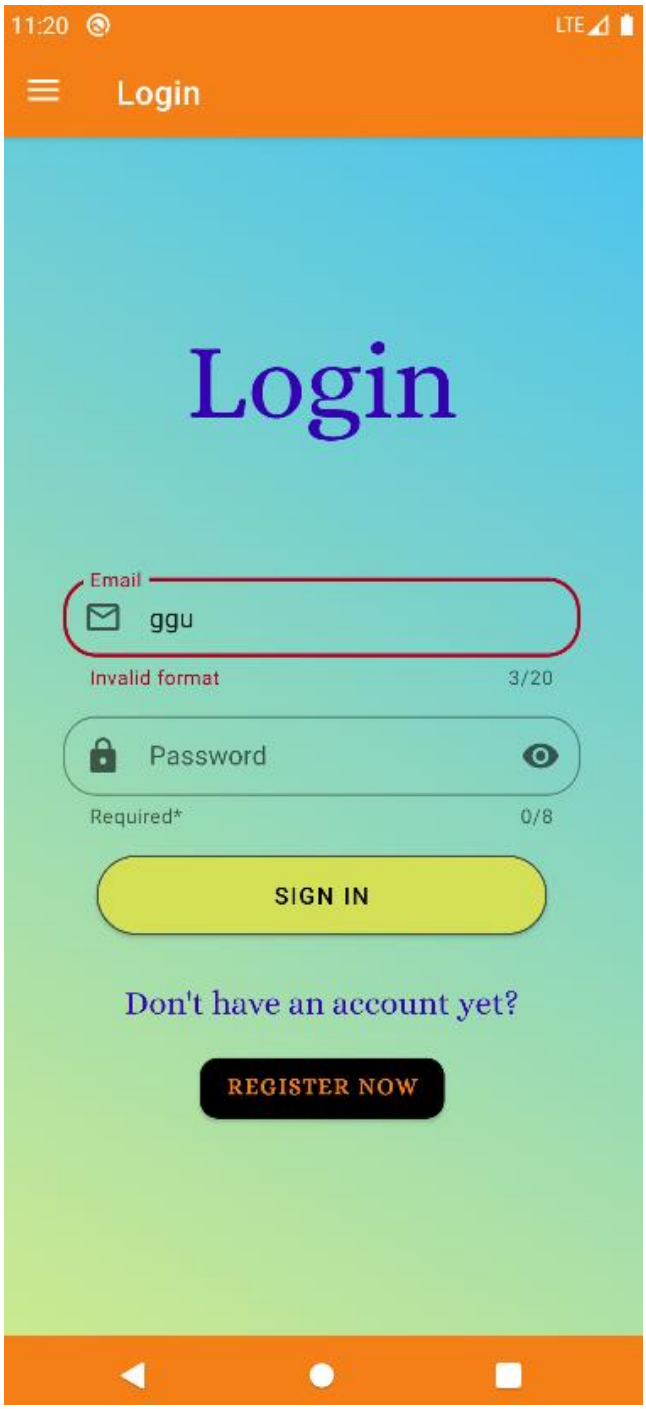


Рисунок 2.9 – Вигляд екрану логіну

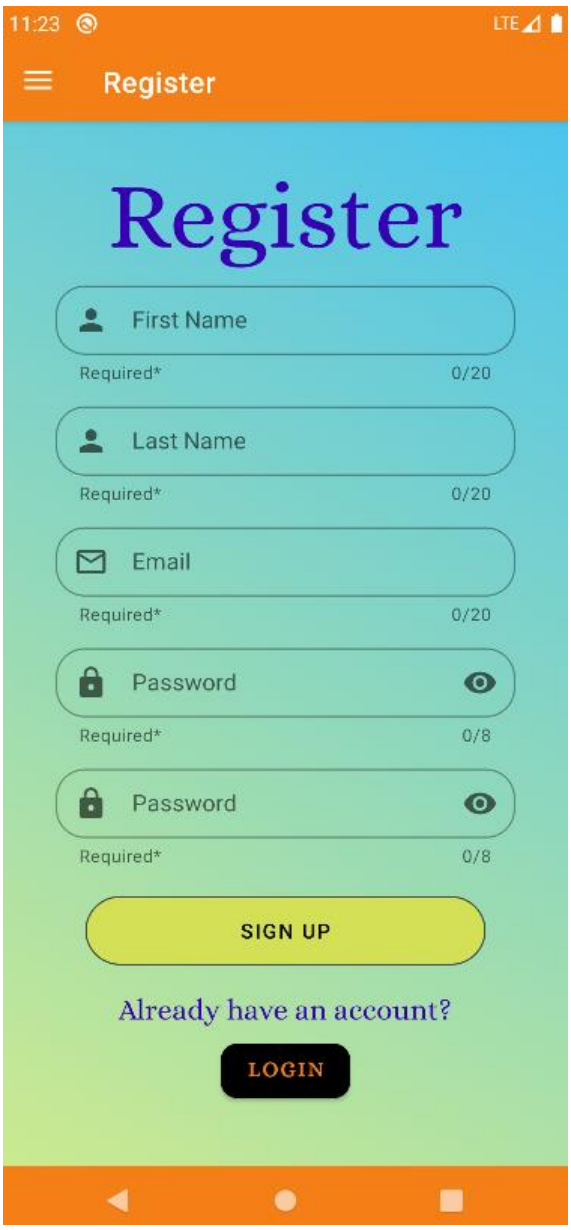


Рисунок 2.10 – Вигляд екрану реєстрації

Зареєстровані юзери мають доступ до спеціально відведених для них екранів. Першим таким екраном є профіль користувача. Він відкривається одразу після логіну чи реєстрації користувача. Тут є можливість оновити данні, що були вказані при реєстрації, а саме ім’я, прізвище та пошту. Також міститься значок аватару, але наразі не має функціоналу для зміни фотографії користувача, але це планується зробити в подальшому, оскільки планується продовження роботи над даним додатком після захисту даної роботи. В даному екрані реалізований один цікавий функціонал, який дозволяє показувати кнопку оновлення даних, лише якщо

початкові дані користувача були змінені. Тобто якщо до ім’я користувача дописати одну букву, то з’явиться кнопка UPDATE DATA. Якщо ж останній символ буде прибраний, то кнопка буде прихована. Вигляд екрану профіля користувача до зміни показаний на рисунку 2.11.

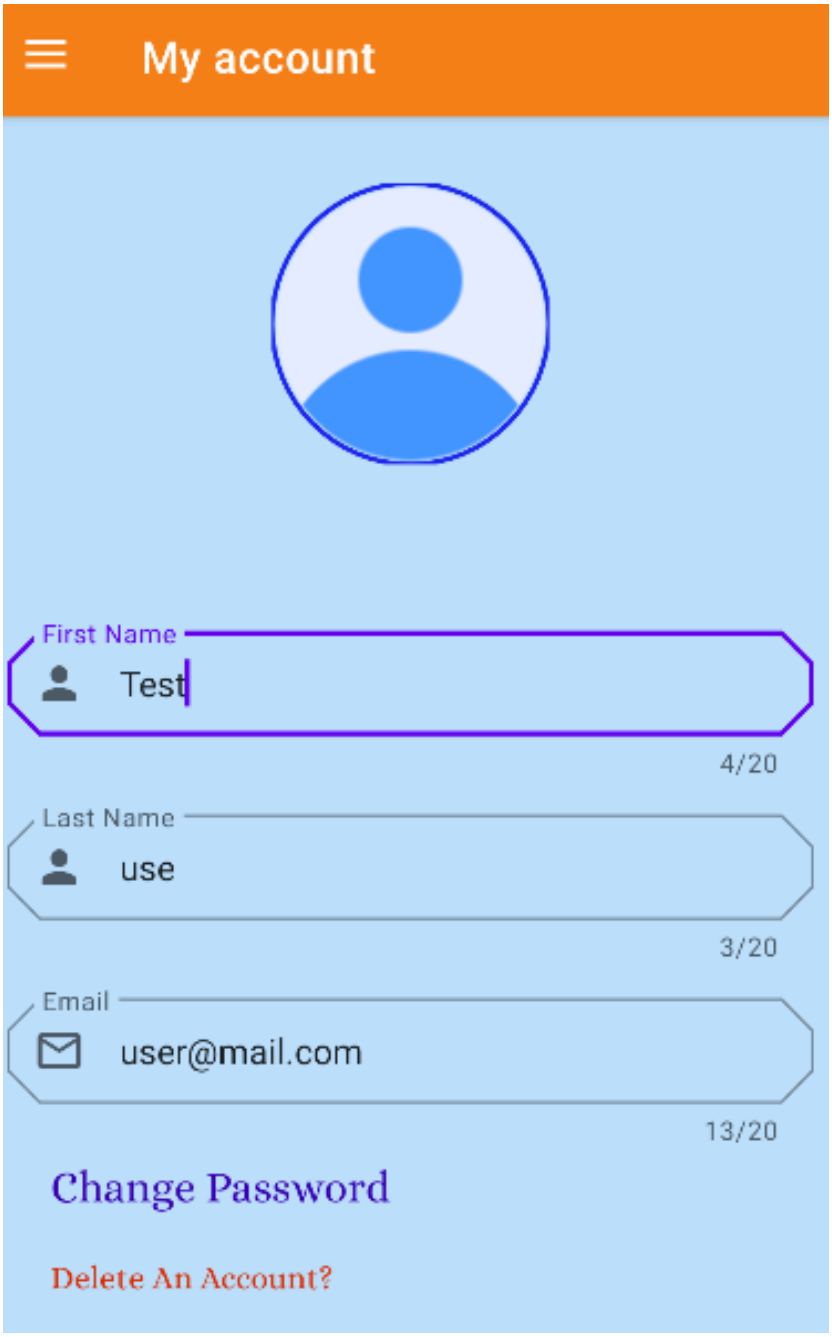


Рисунок 2.11 – Вигляд профілю до зміни даних

Вигляд профілю після зміни початкових даних поданий на рисунку 2.12.

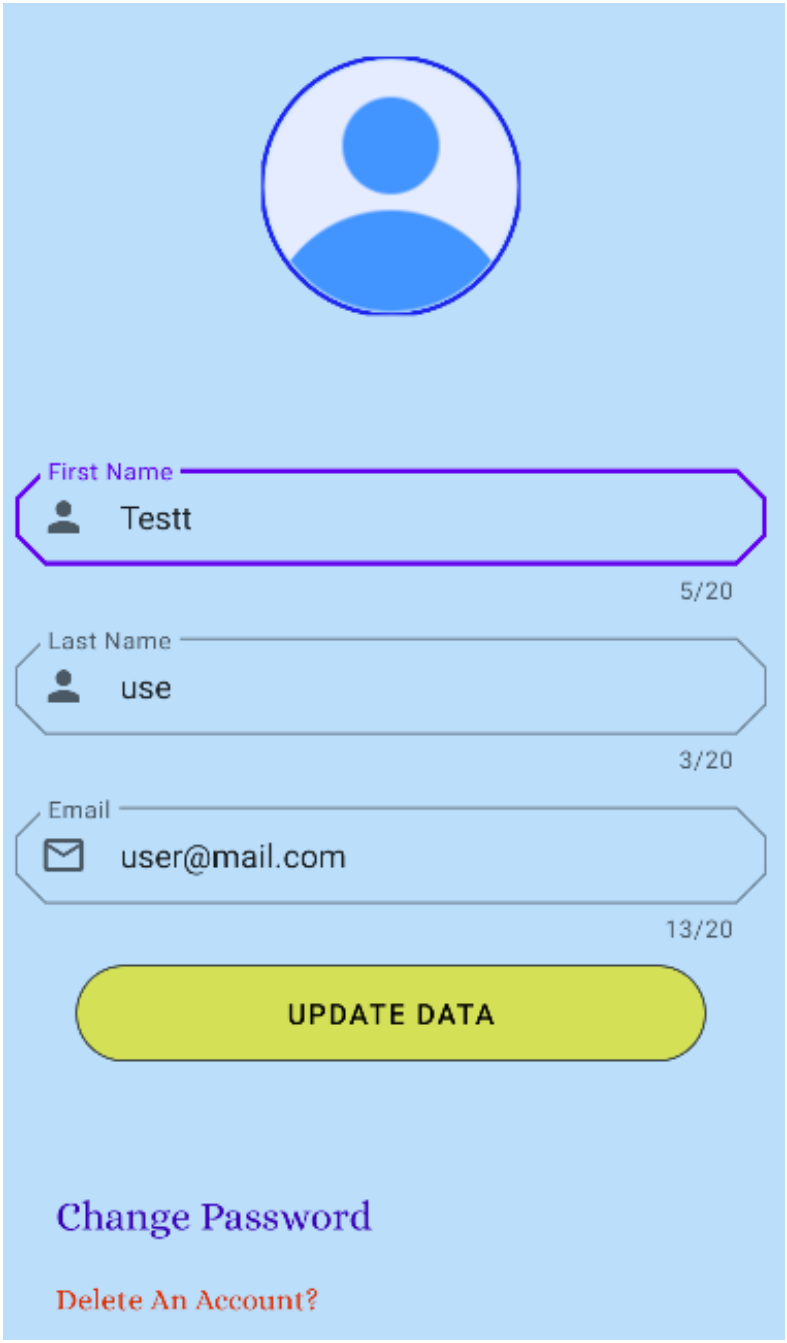


Рисунок 2.12 – Вигляд профілю користувача після зміни початкових даних

На екрані профілю є можливість зміни паролю, що відкриває новий екран. На цьому екрані зображені три поля для введення паролю та кнопка оновлення паролю. У першому полі потрібно ввести старий пароль, а у двох наступних вказати новий пароль та повторити його. У верхній частині замість меню навігації міститься стрілка, що повертає користувача на попередній екран. Екран зміни паролю показаний на рисунку 2.13.

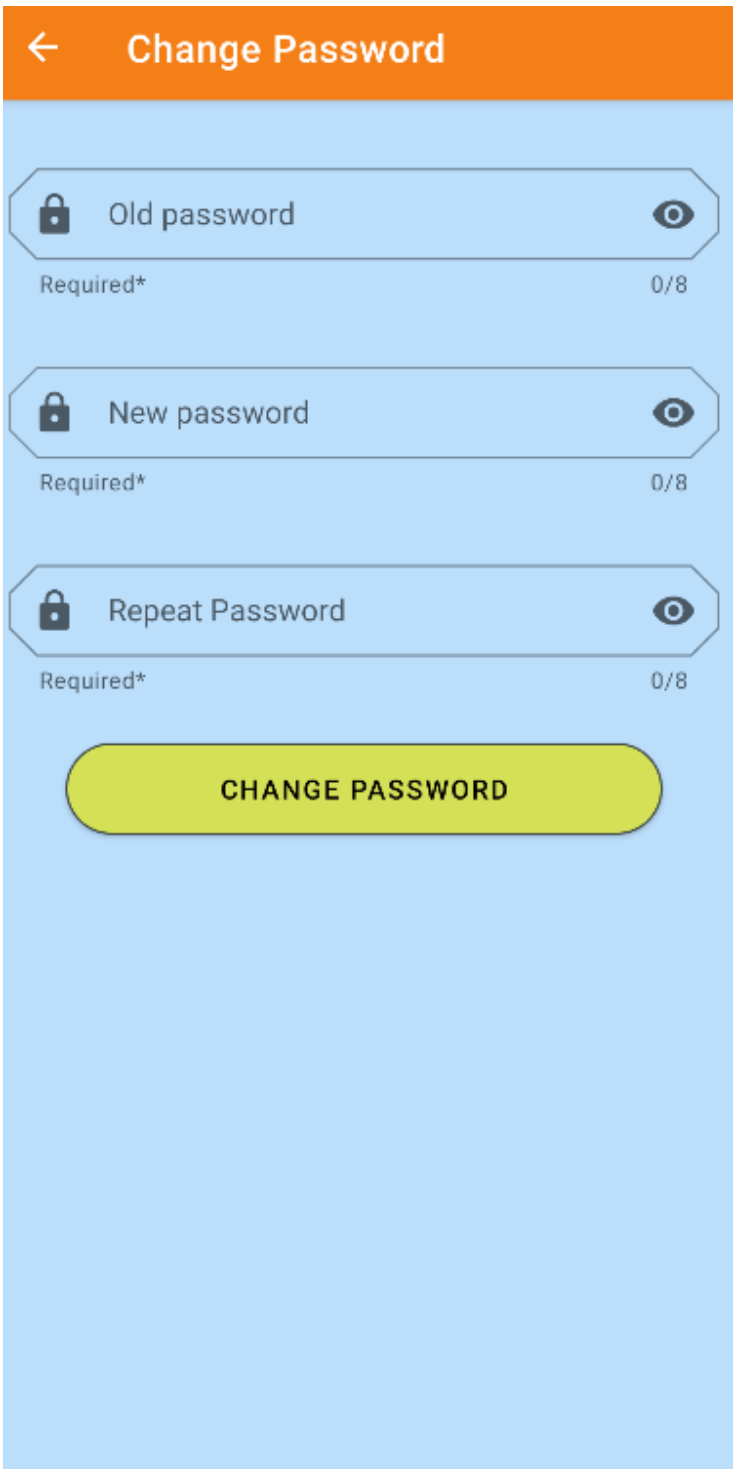


Рисунок 2.13 – Екран зміни паролю

Також, з екрану профіля можна здійснити видалення акаунту. Після кліку відкривається діалогове повідомлення, яке дозволяє здійснити операції видалення, після чого користувач перенаправляється на екран логіну. Діалогове повідомлення подане на рисунку 2.14.

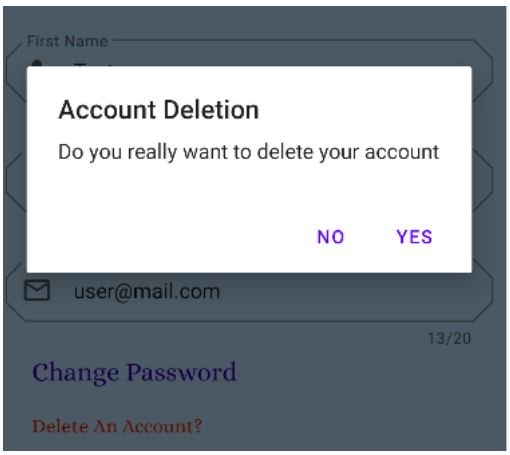


Рисунок 2.14 – Діалогове повідомлення для видалення акаунту

Після реєстрації акаунту або логіну, у навігаційному меню елементи логіну та реєстрації замінюються елементами профілю та елементом, що дозволяє здійснити вихід із акаунту. Вигляд навігаційного меню після логіну/реєстрації поданий на рисунку 2.15.

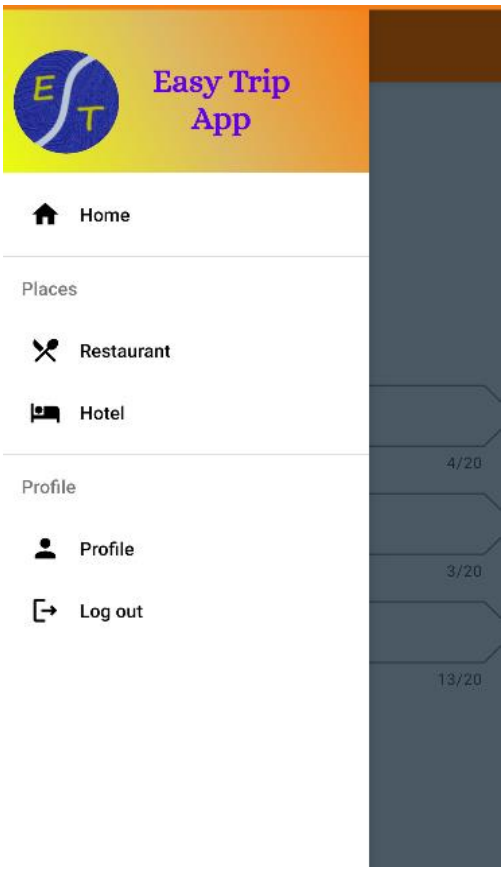


Рисунок 2.15 – Навігаційне меню після реєстрації/логіну

Екрани для пошуку готелів та ресторанів є ідентичними, тому малюнки зовнішнього виду будуть подані для готелів. Після вибору одного з пунктів меню виводиться екран на якому зображені елементи випадаючого списку для країн та міст. Пошук здійснюється лише після вибору конкретного міста. Після вибору країни, список міст звужується у відповідності до обраної країни. Пошук країн та міст можна проводити шляхом набору потрібної назви з клавіатури. При написанні перших букв, що співпадають з назвами країни чи міста у випадаючому списку будуть показані лише ті елементи, які співпадають із введеними даними. Початковий екран пошуку готелів зображений на рисунку 2.16.

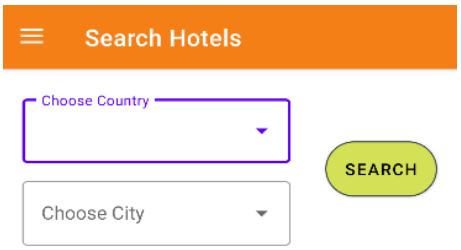


Рисунок 2.16 – Початковий екран пошуку готелів

Вигляд списку міст після вибору конкретної країни поданий на рисунку 2.17.

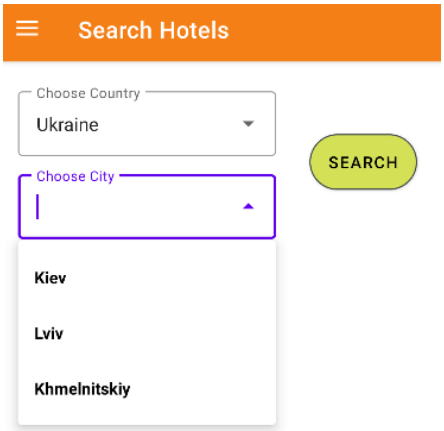


Рисунок 2.17 – Вигляд списку міст для пошуку

Після натискання кнопки пошуку на даному екрані знизу відображаються місця у вигляді списку, що відповідають категорії для якої здійснюється пошук. Кожен елемент має картинку, ім’я, та початковий опис. Якщо для готелю чи ресторану не існує картинки у базі даних, то виводиться чорно-біла картинка як в останньому елементі списку, що поданий на рисунку нижче. Вигляд виведених елементів після натискання кнопки пошуку зображений на рисунку 2.18.

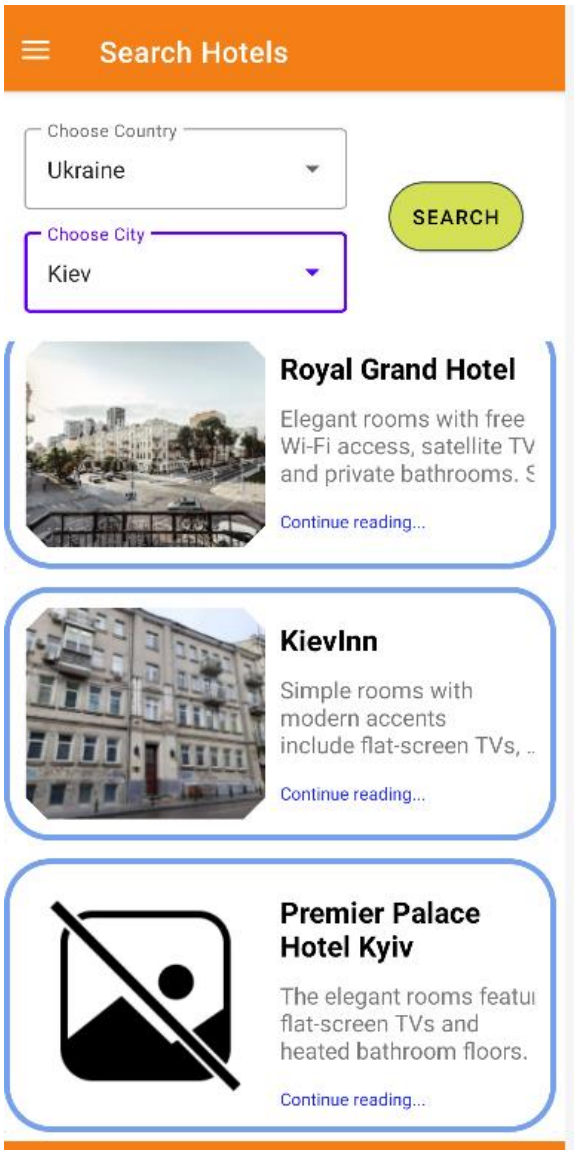


Рисунок 2.18 – Вигляд готелів після натискання клавіші пошуку

Після вибору конкретного елементу відображається екран з більш детальною інформацію про дане місце. Тут міститься картинка, ім’я, телефон для зв’язку,

адреса, робочі години, а також повний опис. Присутній елемент прокрутки сторінки, якщо контент не влізає на початковий екран. Екран з детальною інформацію про обраний готель поданий на рисунку 2.19.

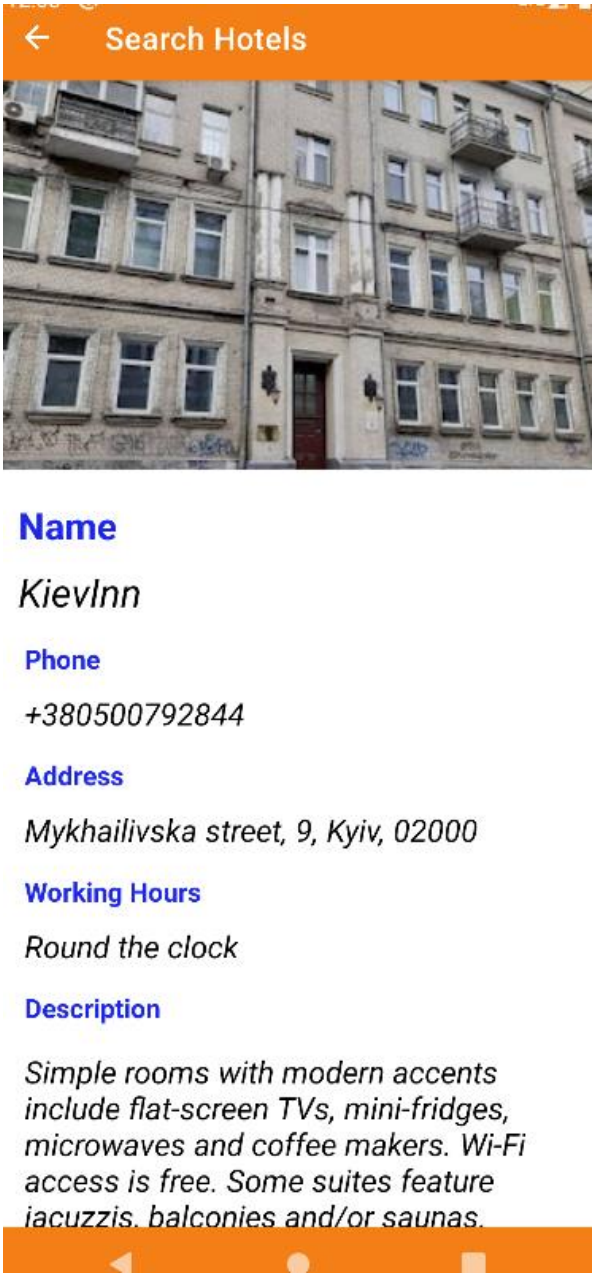


Рисунок 2.19 – Детальна інформація про обраний готель

Крім показу графічного інтерфейсу, також потрібно подати відповідні схеми, на яких зображені послідовності переходів між екранами. Схема переходу екранів для не зареєстрованого користувача приведена на рисунку 2.20.

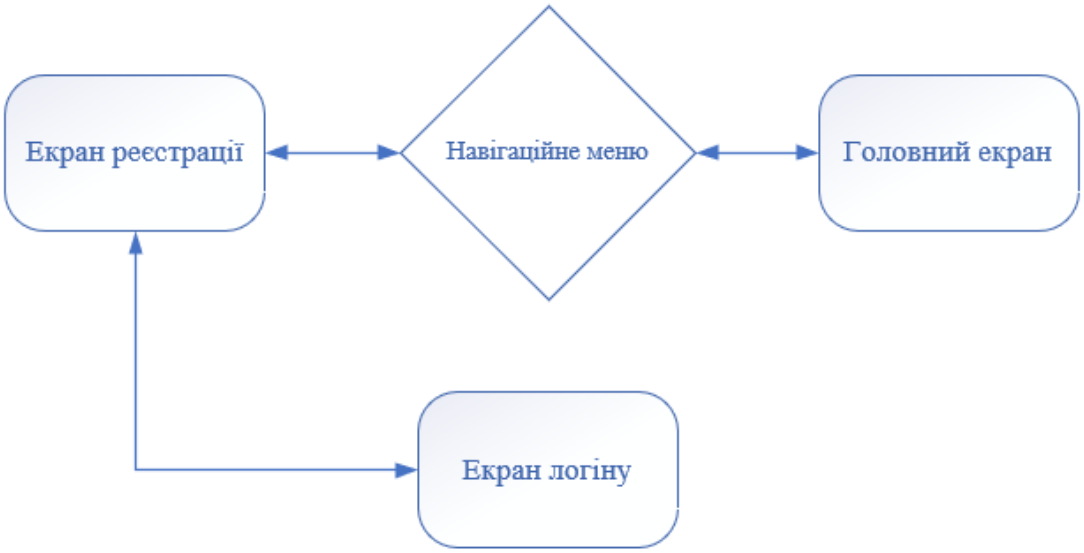


Рисунок 2.20 – Схема переходів екранів для не зареєстрованого юзера

Схема переходу екранів для зареєстрованого користувача приведена на рисунку 2.21 нижче.

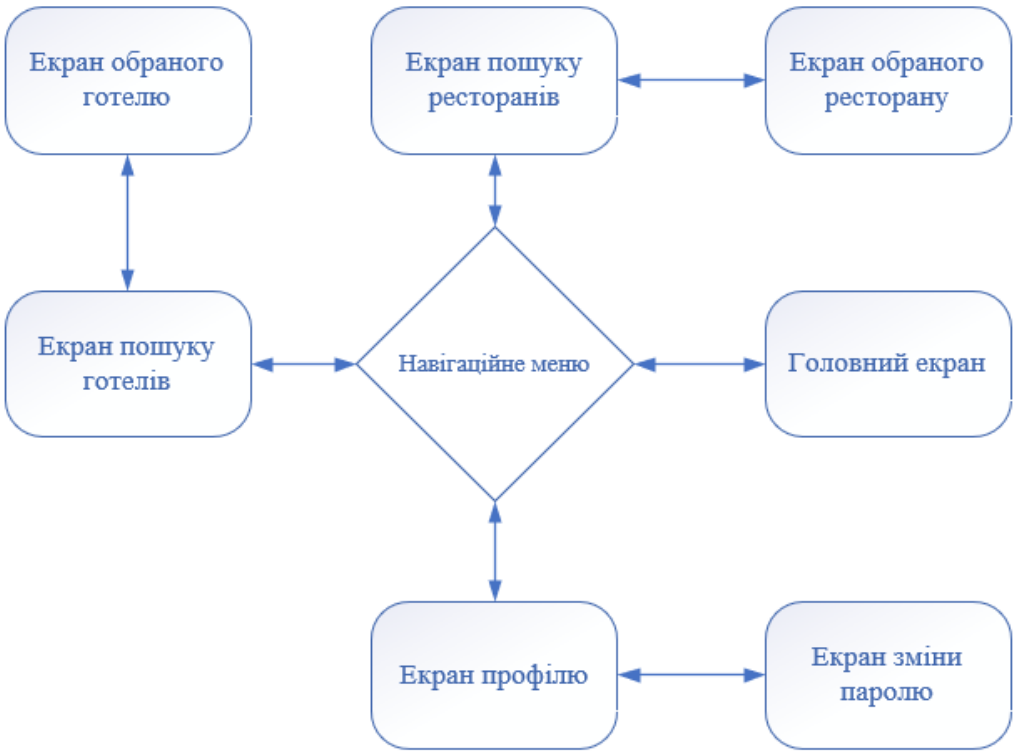


Рисунок 2.21 – Схема переходів екранів для зареєстрованого юзера

2.6 Аналіз та вибір технологій і методів реалізації додатку

В якості веб-серверу, який дозволяє розміщувати базу даних, а також виконувати код, написаний на мові високого рівня PHP було обрано веб-сервер Apache, який є одним із найпопулярніших рішень серед серверів.

Apache є кросплатформним веб-сервером з відкритим вихідним кодом. Він був розроблений Тімом Бернерсом Лі та випущений у 1995 році. Apache швидко став популярним. Веб-сервер був розроблений для доставки веб-контенту, доступ до якого здійснюється через Інтернет. Він відомий тим, що відігравав ключову роль у початковій еволюції та росту популярності інтернету. Apache розроблене та підтримуване відкритим співтовариством розробників і працює в різних операційних системах. Архітектура включає ядро Apache і модулі. Основний компонент надає базову серверну функцію, тому він приймає з'єднання та керує паралелізмом. Різні модулі відповідають різним функціям. Конкретне розгортання Apache може бути налаштовано для включення різних модулів, таких як функції безпеки, керування динамічним контентом або базової обробки HTTP-запитів.

Задля зберігання даних на сервері, було прийняте рішення використовувати систему керування базами даних MariaDB, як прямого конкурента MySQL. Щоб краще зрозуміти різницю між цими двома системами управління базами даних, далі СКБД, варто навести їх короткий опис та вказати їх визначні особливості.

MySQL є найпопулярнішою та першою СКБД реляційними базами даних з відкритим програмним кодом. Сьогодні, незважаючи на безліч альтернатив даного програмного забезпечення, MySQL зумів зберегти свою репутацію. Дана СКБД була випущена у далекому 1995 році і була першим у своєму роді програмним забезпеченням, що дозволяло зручно керувати базами даних. Важливо зауважити, що всі варіації даної СКБД мають подібний синтаксис, оскільки MySQL була першою системою керування базами даних. MySQL широко використовує мову запитів SQL для роботи із даними. Дана СКБД була написана з використанням таких мов програмування як C та C++.

Ключові особливості MySQL:

– перша СКБД;

– наявність відкритого коду, але деякі модулі мають обмежений доступ;

– надійна підтримка транзакцій і веб-розробки в цілому;

– підтримується багатьма серверними операційними системами, такими як Linux, Windows, Solaris;

– пропонує маршрутизацію, але не підтримує аналіз даних.

СКБД MariaDB є розширеною та покращеною версією MySQL в якій були враховані недоліки її попередниці. Є можливість замінити на MariaDB і повністю скористатися перевагами вдосконалень MariaDB без необхідності змінювати код програми, оскільки дані СКБД є повністю сумісними між собою. Дана СКБД є швидкою, масштабованою і надійною. Вона підтримує більше механізмів зберігання, ніж MySQL. MariaDB також включає багато плагінів та інструментів, які роблять її універсальною. Випуск даної систему керуванням базами даних відбувся у 2009 році. Її розробкою займалась група розробників, які звільнились із компанії, що займалась розробкою прямого конкуренту MySQL, оскільки їх погляди стосовно майбутнього розвитку MySQL відрізнялись від поглядів компанії. MariaDB володіє усіма характеристиками MySQL, але також додає ряд покращень, що робить її більш привабливим вибором для розробки.

Ключові особливості MariaDB:

– повністю відкритий програмний код, без окремих модулів, які мають обмежений доступ;

– краща швидкість роботи у порівнянні із MySQL;

– розроблена на основі СКБД MySQL;

– є повністю сумісною із MySQL;

– більша кількість підтримуваних движків зберігання даних;

– підтримка більшої кількості одночасних з’єднань в порівнянні із MySQL.

Вищезгадані веб-сервер Apache та СКБД MariaDB були включені в комплект локального серверу XAMPP. Саме за допомогою локального серверу була змога

побудувати структуру серверної частини та проводити її тестування. Розробка мобільного додатку велась з використанням комплекту програмного забезпечення ХАМРР на локальному комп'ютері. Також, локальний сервер включав підтримку роботи із мовою високого рівня PHP, а саме інтерпретатор даної мови, що використовується для трансляції написаного коду у мову низького рівня, яку розуміє операційна система.

Уся логіка серверної частини була написана за допомогою серверної мови PHP. PHP є це мовою високого рівня, яка виконується на стороні сервера та яка має відкритий вихідний код. Дана мова є популярною серед розробників та широко використовується у веб-розробці. Це також мова загального призначення, яку можна використовувати для створення різноманітних проектів, включаючи графічні інтерфейси користувача. Головним призначенням мови є обробка вхідних запитів, що надходять на сервер та видача результатів клієнту.

Програмний код може вбудовуватись у розмітку HTML документів. PHP має широку підтримку у роботі із великою кількістю СКБД. Слід також відмітити кросплатформеність мови, оскільки вона не залежить від конкретного середовища або операційної системи. Також, мова є легко доступною, оскільки її синтаксис та базові принципи є доволі простими у розумінні. Написання програмного коду PHP відбувалось у програмному забезпеченні PHPStorm.

У парі з PHP було використане програмне забезпечення Composer. Composer є пакетним менеджером залежностей. Програмне забезпечення написане на мові PHP та призначене для використання лише у межах даної мови. Менеджер залежностей дозволяє у зручній формі встановлювати різноманітні PHP фреймворки та бібліотеки враховуючи залежності з іншими бібліотеками чи пакетами. Він допомагає автоматизувати та спростити процеси управління бібліотеками та їх залежностями.

Робота із Composr здійснюється із використанням інтерфейсу терміналу. У розробці мобільного додатку, Composer використовувався не за прямим призначення, а саме не як пакетний менеджер. Дане програмне забезпечення

додатково володіє функціями автоматичного підключення таких компонентів PHP як класи та файли.

Задля підключення PHP модулів використовуються загальновідомі ключові слова «include» та «require», але це погіршує загальну структуру коду, а також потребує додаткової уваги зі сторони розробників, але Composer вирішує дану проблему. Він дає можливість створювати відповідність неймспейсу класа до директорії у якій повинен знаходитись даний клас, що дозволяє у динамічному режимі підключати потрібні класи у моменти створення їх об’єктів у програмному коді, що значно спрощує роботу.

Для розробки клієнтської частини була використана мова програмування високого рівня Java, яка на мою суб`єктивну думку є найкращим рішення для створення мобільних застосунків, які працюють на операційній системі Android. Написання коду проводилось у Android Studio.

Java є об’єктно орієнтованою мовою високого рівня та однією із найпопулярніших мов програмування. Дана мова має широкий спектр застосування, а саме веб-розробка, ігрова індустрія, створення мобільних додатків, а також розробка десктопних програмних рішень. У парі із Java використовується програмне забезпечення Gradle.

Gradle є інструментом автоматизації збірки проекту і відомий своєю гнучкістю у створенні програмного забезпечення.

Інструмент автоматизації збірки використовується для автоматизації створення додатків. Процес побудови включає компіляцію, зв’язування та пакування коду. Gradle дозволяє підключати різноманітні пакети та бібліотеки автоматизуючи процес розробки.

У ході проведення детального аналізу та проектування мобільного додатку була визначена основна архітектура, а саме клієнт-сервер, що є основою при розробці додатку, а також наведений архітектурний підхід REST API, що допомагає встановлювати чітке та прозоре спілкування між двома незалежними частинами мобільного додатку, якими виступають клієнт та сервер. Окрім цього, були

приведені основні поняття, що використовуються при проектуванні мобільного додатку, а також їх детальний опис.

Було проведене проектування структури бази даних із вказанням основних відношень, що є таблицями у вигляді окремих таблиць із вказанням усіх атрибутів. Також, для кращої візуалізації була створена фізична модель бази даних на якій зображені усі відношення, їх внутрішня структура, а також зв’язки між ними. Було здійснене проектування серверної та клієнтської частин для яких були створені діаграми класів, що відображають їх основні модулі, а також зв’язки між ними. У серверній частині додатково був наведений архітектурний підхід REST API та пояснення його вирішальної ролі у додатку над яким ведеться розробка.

Був наведений візуальний вигляд спроектованого інтерфейсу користувача у якому вказані основні екрани додатку разом із наповненням та їх зовнішній вигляд. Були спроектовані схеми переходів екранів додатку окремо для зареєстрованих та не зареєстрованих користувачів. Вони відображають екрани до яких мають доступ відповідні групи користувачів, а також

В останню чергу були визначені технології, програмні засоби та компоненти, що будуть використовуватись при розробці мобільного застосунку.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Реалізація логіки серверної частини додатку

Пояснення реалізації серверної частини слід почати із подання загальної структури папок та файлів, що допоможе краще зрозуміти структуру серверної частини над якою ведеться розробка. Вигляд структури файлів та папок серверної частини поданий на рисунку 3.1.

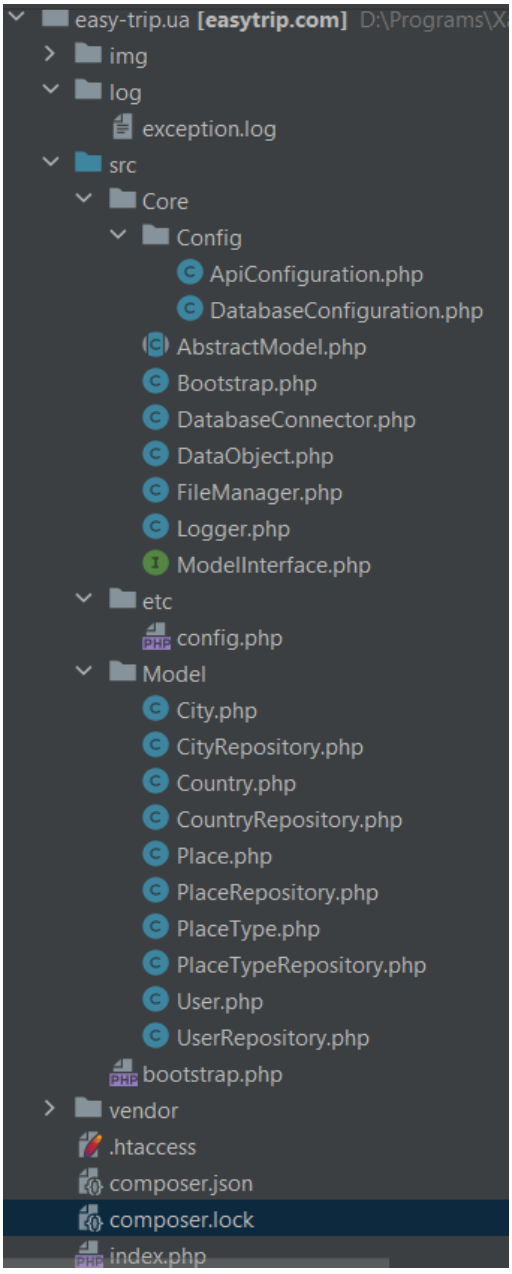


Рисунок 3.1 – Файлова структура серверної частини додатку

Папка img містить картинки, які використовуються при пошуку готелів та ресторанів у мобільному додатку. Всередині даної папки містяться такі папки як hotel та restaurant, які вміщують картинки для готелів та ресторанів відповідно. Всередині кожної папки є список папок і мені яких складаються із цифр, які відповідають назвам ключового атрибуту entity_id таблиці place, що містить інформацію про готелі та ресторани. Вигляд вмісту папки img поданий на рисунку під номером 3.2.

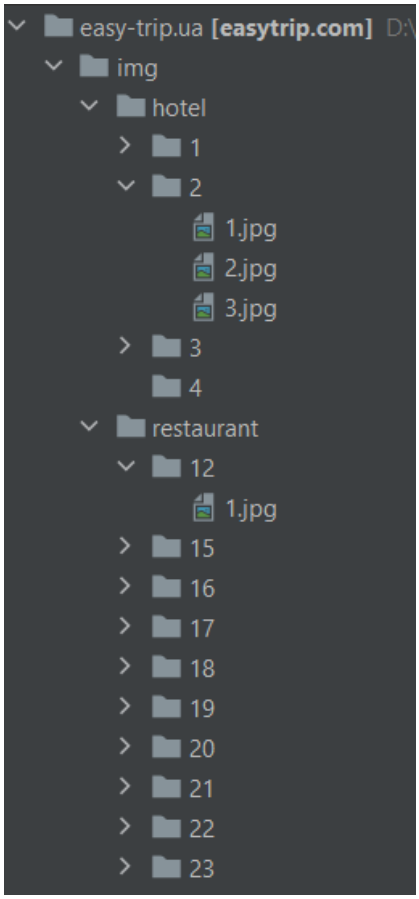


Рисунок 3.2 – Внутрішня структура папки img

У корені проекту міститься ряд важливих файлів. Слід розпочати опис із файлів пакетного менеджера залежностей під назвою Composer. Йому відповідає папка «vendor» у якій міститься програмний код, що забезпечує роботу Composer, а також два файли, а саме composer.json та «composer.lock». Файл composer.json призначений для вказання бібліотек, пакетів та фреймворків, від яких залежить

проект, а у файлі composer.lock міститься їх детальний опис, а також наводяться залежності основних модулів, які були вказані у попередньому файлі. Як уже зазначалось, Composer використовується у даному проекті не з метою автоматичного завантаження бібліотек, а з метою використання його функції автопідключення класів та файлів PHP. Для роботи автопідключення слід додати блок коду до файлу composer.json, оскільки уся взаємодія при роботі із Composer здійснюється саме з цим файлом, а файл composer.lock автоматично генерується на його основі. Частина коду, яка відповідає за автоматичне підключення усіх класів проекту наведена нижче:

```
"autoload": {
        "psr-0": {
            "": "src/"
        }
    },
```

Структура обох вищезгаданих конфігураційних файлів Composer побудована у форматі JSON. На зображені вище подається елемент autoload, у тілі якого вказуються різноманітні правила автоматичного підключення класів. Виділяють ряд правил, що визначають способи підключення PHP модулів, але у даному проекті використовується лише одне правило, а саме psr-0 у якому вказується неймспейс класів для яких потрібно проводити пошук при створенні об’єктів, а також вказується назва папки, тобто місце у якому має проводитись перевірка на існування певного класу.

Пустий неймспейс, означає те, що будь-який клас, який задовільняє правилам визначеним у psr-0 та який міститься у папці src буде динамічно підключатись у програмному коді.

Файл .htaccess є спеціальним конфігураційним файлом веб-серверу Apache. В даному випадку, він використовується з метою перенаправлення вхідних запитів на файл index.php задля їх подальшого аналізу та обробки. Структура файлу є

доволі проста. У ньому вказується, що потрібно включити функцію перенаправлення вхідного трафіку, а також вказується умова, при якій усі запити, що містять у назві частину img не повинні бути перенаправлені до файлу index.php, оскільки такі запити призначені не для спеціальної обробки, а для звертання до картинок напряму через вказання точного шляху до них. Усі інші запити, що не містять частку img перенаправляються до файлу index.php. Структура конфігураційного файлу веб-серверу Apache:

```
## Enable rewrites
RewriteEngine on
## Rewrite everything else to index.php
RewriteCond %{REQUEST_URI} !^/img/
RewriteRule .* index.php [L]
```

Файл index.php, як уже було сказано, призначений для обробки усіх вхідних запитів, що надходять до серверу. Всередині нього викликається функція header, що призначена для вказування інформації, що буде міститись у заголовку HTTP відповіді, а саме вказує ресурси, які можуть мати доступ до серверу, формат відповіді, що буде надісланий клієнту, а також дозволені HTTP методи. Нижче йде підключення файлу bootstrap.php у якому виконується додаткові перевірки. Вкінці файлу index.php викликається метод run класу Bootstrap, якщо початкові запит пройшов початкові перевірки. Код index.php поданий нижче:

```
header("Access-Control-Allow-Origin: *");
header("Content-Type: application/json; charset=utf-8");
header("Access-Control-Allow-Methods: GET,POST,PUT,DELETE");

require 'src/bootstrap.php';

$bootstrap = new Bootstrap();
$bootstrap->run($_SERVER);
```

Файл bootstrap містить базові перевірки. У першій стрічці міститься нативний метод error_reporting, що призначений для включення відображення широко спектру помилок у програмі. Далі йде перевірка на коректність PHP версії. Якщо сервер містить версію нижчу за восьму, то буде викинута помилка. Для викидання помилок та закінчення дострокового закінчення виконання програми, у програмі застосовуються дві нативних функції http_response_code та exit. Перша вказує код HTTP відповіді, а друга завершує головний потік виконання програми та містить функцію json_encode, що призначена для кодування повідомлення про помилку яке вказано у її тілі. Загалом у додатку використовується три типи кодів для HTTP відповідей, а саме 200 для позначення успішного виконання запиту, 500 для позначення помилок на стороні сервера, а також 400 код, що показує помилки зі сторони клієнта. Далі йде перевірка на коректність введених значень логіну та паролю. Для доступу до серверу, а саме для можливості виконання запитів на визначені ідентифікатори ресурсу використовуєтья базова верифікація кожного запиту. Це допомагає відкрити доступ лише для обмеженої групи користувачів задля уникнення різноманітних перевантажень серверу спричинених націленими атаками з боку хакерів. У самому низу вказується часовий пояс Europe/Kiev з використанням нативної функції date_default_timezone_set. Також, у файлы bootstrap.php підключається автолоадер Composer з використанням ключового слова require_once. Програмний код файлу bootstrap.php:

```
require_once 'vendor/autoload.php';
error_reporting(E_ALL);
// Check PHP version availability
if (!defined('PHP_VERSION_ID') || PHP_VERSION_ID < 80000) {
    http_response_code(500);
    exit(json_encode(['success' => 0, 'message' => "The API doesn't
support the PHP version lower than 8.0.0"]));
}
// Basic authorization check
```

```
if (empty($_SERVER['PHP_AUTH_USER']) || $_SERVER['PHP_AUTH_USER']
!== ApiConfiguration::API_LOGIN ||
    empty($_SERVER['PHP_AUTH_PW']) || $_SERVER['PHP_AUTH_PW'] !==
ApiConfiguration::API_PASSWORD) {
    http_response_code(401);
    exit(json_encode(['success' => 0, 'message' => 'Provided
credentials are wrong']));
}
date_default_timezone_set('Europe/Kiev');
```

Увесь програмний код міститься у папці src, що виступає кореневою папкою для серверної частни проекту.

Задля зручного відслідковування помилок, які виникають при різноманітних збоях на сервері, було прийняте рішення розробити клас, що буде вести записи про них. Саме клас під назвою Logger призначений для виконання запису помилок у файл exception.log, що міститься у папці log. Даний клас містить лише один метод log, який приймає на вхід повідомлення про помилку, а також не обов'язковий параметр, що дозволяє включати системні повідомлення великих обсягів. Частина програмного коду класу Logger:

```
/**
     * Adding records to the log file
     *
     * @param array $errorTrace
     */
    public function log(string $message, array $errorTrace = [])
    {
        $dateTime = new DateTime("now");
        $dateTime->setTimestamp(time());
        $dateTime = $dateTime->format('d.m.Y, H:i:s');
        $encodedErrorTrace = json_encode($errorTrace);
        try {
```

```
            $this->fileManager->write(self::LOG_FILE_PATH,
"[$dateTime] $message: $encodedErrorTrace" . PHP_EOL, FILE_APPEND);
        } catch (\Exception $error) {
            http_response_code(500);
            exit(json_encode(['success' => 0, 'message' => $error-
>getMessage()]));
        }
    }
```

Всередині вищенаведеного методу задається формат виведення дати у повідомленні, а також відбувається виклик методу write класу FileManager, який був розроблений для маніпулювання елементами файлової структури. У класі FileManger містяться методи, що дозволяють виконувати читання файлу, запис у файл, перевіряти чи існує файл чи директорія, а також перевіряти доступність файлів на читання та запис. Також всередині класу існує метод setReadWritePermissions, що дозволяє змінювати доступ вказаним файлам. Метод write, який нас цікавить виконує запис інформації у вказаний файл. У методі проводиться перевірка на існування файлу та права на його запис, а також проводиться безпосередній запис шляхом виклику нативної функції file_put_contents. Код методу write класу FileManager:

```
/**
     * @param string $path
     * @param $data
     * @return false|int
     * @throws \Exception
     */
    public function write(string $path, $data, int $flag =
FILE_APPEND): bool|int
    {
        if ($this->isFileExist($path)) {
            if (!$this->isFileWritable($path)) {
```

```
                if (!$this->setReadWritePermissions($path)) {
                    throw new \Exception("Write permissions can't be
changed for requested file $path");
                }
            }
            $result = file_put_contents($path, $data, $flag);
        } else {
            throw new \Exception("Requested file $path doesn't
exist");
        }
        return $result;
    }
```

Класи ApiConfiguration та DatabaseConfiguration використовуються для зберігання інформації про логін, пароль, назви HTTP методів, а також даних, які використовуються для доступу до бази даних.

Для доступу до бази даних був розроблений клас DatabaseConnector, що використовує вбудований клас PDO за допомогою якого створюєтья об'єкт з'єднання, що надає методи для виконання операцій з даними у базі даних. Клас DatabaseConnector містить три методи, а саме метод initConnection, що призначений для створення об’єкту класу PDO у конструктор якого передаються базові данні, що слугують для встановлення з’єднання, метод getConnection, що повертає щойно створений об’єкт методом initConnection та метод closeConnection який знищує об’єкт з’єднання.

Після надходження запиту на сервер та успішної базової перевірки у файлі bootstrap.php, відбувається виклик методу run класу Bootstrap у файлі index.php. Всередині методу відбувається створення об’єкту класу Logger, а також виклик методу processRequest у якому проводиться перевірка вхідного ідентифікатору ресурсу на коректність. Програма достроково завершить свою роботу, якщо ресурс не заданий, HTTP метод порожній або вхідний запит складається більше ніж з трьох частин. Вхідний ресурс розбивається на окремі елементи або частини що

містяться між знаками слешу та записується у масив. Під ідентифікатором ресурсу, слід розуміти частину запиту, що слідує після доменного імені серверу. У методі processRequest міститься умовна конструкція switch що займається перенаправленням вхідного запиту у відповідності з його типом HTTP методу. Серверна частина підтримує чотири типи HTTP методів, а саме GET, POST, PUT і DELETE. Для кожного з наведених HTTP методів викликаються два методи match і validateRepositoryCall. Метод match виконує пошук та співставлення вхідного ресурсу із списком статично заданих, що беруться із файлу config.php.

У файлі config.php міститься асоціативний масив, що описує доступні ідентифікатори ресурсу до яких можна звертатись клієнтам. Масив розбивається на чотири типи попередньо визначених HTTP методів, всередині яких містяться назви сутностей, що співпадають із першою частиною ресурсу, завдяки яким здійснюється групування запитів кожної із сутностей у межах одного HTTP методу. Всередині назви сутності вказується повна назва адреси ресурсу, яка вказує на масив у якому міститься клас та метод, який буде викликаний для обробки даного ресурсу. Код файлу config.php:

```
<?php
return [
    'POST' => [
        'user' => [
            '/user' => [
                'class' => 'Model\UserRepository',
                'method' => 'save'
            ]
        ],
    ], 'GET' => [
    'user' => [
            '/user/:id' => [
                'class' => 'Model\UserRepository',
                'method' => 'getById'
```

```
        ],
        '/user' => [
            'class' => 'Model\UserRepository',
            'method' => 'getList'
        ],
        '/user/by-email' => [
            'params' => true,
            'class' => 'Model\UserRepository',
            'method' => 'getByEmail'
        ]
    ],
    'country' => [
        '/country' => [
            'class' => 'Model\CountryRepository',
            'method' => 'getList'
        ],
    ],
    'city' => [
        '/city' => [
            'class' => 'Model\CityRepository',
            'method' => 'getList'
        ],
    ], 'place' => [
        '/place/hotel/:id' => [
            'class' => 'Model\PlaceRepository',
            'method' => 'getHotelByCityId'
        ],
        '/place/restaurant/:id' => [
            'class' => 'Model\PlaceRepository',
            'method' => 'getRestaurantByCityId'
        ],
    ], 'place-type' => [
                '/place-type' => [
```

```
'class' => 'Model\PlaceTypeRepository',
                'method' => 'getList'
            ],
        ]
    ],
    'DELETE' => [
        'user' => [
            '/user/:id' => [
                'class' => 'Model\UserRepository',
                'method' => 'delete'
            ]
        ],
    ],
    'PUT' => [
        'user' => [
            '/user/:id' => [
                'class' => 'Model\UserRepository',
                'method' => 'update'
            ]
        ],
    ]
];
```

Деякі ресурси містять :id у своєму закінченні. Це означає, що на цьому місці повинен вказуватись числовий ідентифікатор ресурсу.

Слід також зазначити, що ресурс /user/by-email містить ключ params, що вимагає передачі параметру, який містить значення пошти задля отримування даних про користувача, що має дану пошту. Даний файл використовується лише у методі match класу Bootstrap.

Метод match містить логіку, що допомагає знайти відповідність вхідного ресурсу до ресурсу, що заданий у файлі config.php. Якщо відповідність не знайдено,

то програма буде завершена і повідомлення про відповідну помилку буде відправлене клієнту у форматі JSON.

Після успішного співставлення вхідного запиту відбувається виклик методу validateRepositoryCall у якому проводиться перевірка даних, що використовуються для виклику відповідного методу класу, що були визначені у файлі конфігурації config.php. У методі проводиться три перевірки. Перша перевіряє чи дані не є пустими, друга перевіряє чи заданий клас існує у проекті і третя перевіряє чи існує такий метод у заданому класі. Код методу validateRepositoryCall класа Bootstrap:

```
/**
     * Validate data for calling repository method
     *
     * @param $callData
     * @return array|void
     */
    public function validateRepositoryCall($callData)
    {
        // data existence check
        if (!is_array($callData) || empty($callData)) {
            $this->logger->log(sprintf('Call data is incorrect %s',
$callData));
            http_response_code(500);
            exit(json_encode(['success' => 0, 'message' => 'Internal
server error. Try again later']));
        }
        // class existence check
        if (empty($callData['class']) ||
!class_exists($callData['class'])) {
            $this->logger->log("Requested class " .
$callData['class'] . " doesn't exist");
            http_response_code(500);
            exit(json_encode(['success' => 0, 'message' => 'Internal
server error. Try again later']));
```

```
        }

        // method existence check
        if (empty($callData['method']) ||
!method_exists($callData['class'], $callData['method'])) {
            $this->logger->log("Requested method " .
$callData['method'] . " doesn't exist");
            http_response_code(500);
            exit(json_encode(['success' => 0, 'message' => 'Internal
server error. Try again later']));
        }
        return $callData;
    }
```

Якщо вхідний запит був знайдений у файлі config.php і не було виявлено жодних помилок при його обробці, починається виклик відповідного методу класу репозиторію. Існує п’ять класів репозиторіїв, які закріплюються за конкретними класами моделей. Репозиторії надають інтерфейси, що являють собою методи, які взаємодіють із методами абстрактного класу AbstractModel. Існує п’ять класів репозиторіїв, а саме CityRepository, CountryRepository, PlaceRepository, PlaceTypeRepository, UserRepository.

Існує п’ять класів моделей: City, Place, Country, PlaceType та User. Обов’язковим для кожного методу є наявність конструктору у якому викликається метод _init абстрактного класу AbstractModel. У якості параметрів передається назва відношення у базі даних, а також його ключове поле. Ці дані будуть використовуватись у методах, що реалізовані у абстрактному класі. Опис структури методів, що використовуються для доступу до даних слід почати з інтерфейсу ModelInterface. Даний інтерфейс визначає чотири основних методи, а саме load, save, update та delete. Реалізації даних методів містяться у абстрактному класі AbstractModel. Кожен з методів має схожу структуру, тому доцільним є розбір лише одного з них, зважаючи на їх об’єм. Метод delete складається із частини у

якій перевіряється чи були передані початкові дані відповідним класом моделі, що визвав даний метод, а також проводиться перевірка того, чи існує вказана таблиця шляхом виклику методу isTableExist. Далі відбувається формування SQL запиту відповідного типу, що відповідає функціям конкретного методу. Сформований метод перевіряється на коректність шляхом виклику методу prepare у об'єкта вбудованого класу PDO, який був створений за допомогою розробленого клас DatabaseConnector. Якщо запит сформований успішно, то відбувається виклик методу beginTransaction у об'єкта класу PDO і виконання самого запиту шляхом виклику методу execute. Якщо впродовж запиту не сталось системної помилки, то об'єкт PDO викликає метод commit, що підтверджує виконання запиту і дані відповідним чином модифікуються чи отримуються. Якщо ж стається помилка впродовж виконання запиту, то викликається метод rollBack, що призводить до відміни даного SQL запиту. Після виконання запиту з'єднання із базою даних розривається та йде повернення результатів у класі Bootstrap. Код методу delete:

```
/**
     * Delete model
     *
     * @param int $entityId
     * @return bool
     */
    public function delete(int $entityId): bool
    {
        $success = false;

        $this->isInitialized();
        $this->isTableExist();
        $connection = $this->getConnection();

        $sql = "DELETE FROM $this->tableName WHERE $this-
>idFieldName = $entityId";
```

```
        $statement = $connection->prepare($sql);
        if ($statement) {
            try {
                $connection->beginTransaction();
                $success = $statement->execute();
                $connection->commit();
            } catch (\Exception $error) {
                $connection->rollBack();
                $this->logger->log('Error during deleting model
data' . __CLASS__, [$error->getMessage()]);
                http_response_code(500);
                exit(json_encode(['success' => 0, 'message' =>
$error->getMessage()]));
            }
        }
        $this->databaseConnector->closeConnection();
        return $success;
    }
```

Клас AbstractModel наслідується від класу DataObject. Клас DataObject містить різноманітні методи, що призначені для роботи із масивом, що оголошений у ньому як публічне поле. Даний клас надає можливість записувати та видаляти дані із масиву у зручному вигляді, для об’єктів моделей, оскільки кожна модель наслідується від абстрактного класу AbstractModel. Це буває корисним при оновленні та створенні нових записів, оскільки масив класу DataObject виступає у ролі своєрідного контейнеру для даних і дозволяє маніпулювати ними надаючи велику кількість різноманітних інструментів у вигляді методів.

Частина коду методу setData класу DataObject, що виконує запис даних у масив в залежності від вхідного параметру:

```
/**
     * Overwrite or add data to the object
```

```
     * if $key is an array data will be overwritten
     * if $key is a string data will be added
     * @param $key
     * @param $value
     * @return $this
     */
    public function setData($key, $value = null): static
    {
        if ($key === (array)$key) {
            $this->_data = $key;
        } else {
            $this->_data[$key] = $value;
        }
        return $this;
    }
```

Також, клас DataObject містить цікавий магічний метод __call, який викликається кожного разу, коли відбувається звертання до не існуючого методу у даному класі.

В даному випадку метод __call дозволяє викликати такі методи як set, get, has та uns після яких слідує ім’я одного із елементів масиву, що дозволяє відповідно додавати, отримувати, робити перевірку на існування та видаляти окремі елементи публічного масиву, тобто здійснювати роботу над окремими елементами. Код магічного методу __call класу DataObject:

```
/**
     * Set/Get attribute wrapper
     * @param string $method
     */
    public function __call(string $method, array $args)
    {
        switch (substr($method, 0, 3)) {
            case 'get':
```

```
                $key = $this->_underscore(substr($method, 3));
                return $this->getData($key);
            case 'set':
                $key = $this->_underscore(substr($method, 3));
                $value = $args[0] ?? null;
                return $this->setData($key, $value);
            case 'uns':
                $key = $this->_underscore(substr($method, 3));
                return $this->unsetData($key);
            case 'has':
                $key = $this->_underscore(substr($method, 3));
                return isset($this->_data[$key]);
        }
        throw new \Exception('Invalid method %1::%2(%3)',
[get_class($this), $method, print_r($args, 1)]);
    }
```

Якщо співпадіння з методами set, get, uns, has не знайдене, то буде показана помилка.Код методу getData, що забезпечує отримання даних із публічного масиву:

```
    public function getData(string $key = ''): mixed
    {
        if ('' === $key) {
            return $this->_data;
        }

        if (isset($this->_data[$key])) {
            return $this->_data[$key];
        }
        return null;
    }
```

Повний код серверної частини подано у додатку Б.1.

3.2 Реалізація логіки клієнтської частини додатку

Опис реалізації клієнтської частини як і у випадку із серверною слід почати із наведеннями структури основних модулів у файловій системі. Організація структури основних модулів клієнтської частини подана на рисунку 3.3.

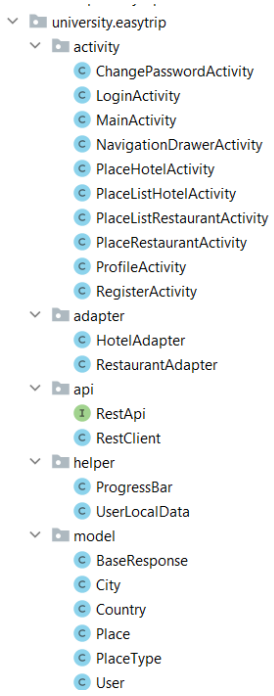


Рисунок 3.3 – Структура модулів клієнтської частини

Основні бібліотеки, що використовувались для розробки клієнтської частини містяться у файлі Gradle. Першою бібліотекою є Retrofit. Вона виступає REST клієнтом і забезпечує опис та реалізацію спілкування із сервером. Другою бібліотекою є Gson, що дозволяє на льоту перетворювати відповіді у форматі JSON, що приходять із серверу у відповідні об’єкти. Третьою бібліотекою є ReactiveJava, що також включає підтримку для Android. Дана бібліотека використовується для управління потоками. Також, бібліотека реактивного програмування реалізує паттерн Observer-Observable. Біблітека OkHTTP виступає HTTP клієнтом, за допомогою якої і відбувається здійснення запитів. Бібліотека Retrofit побудована на бібліотеці OkHTTP. Для можливості використання графічних елементів, що

входять до пакету Material була додана відповідна бібліотека. І останньою бібліотекою є Glide, що дозволяє завантажувати та відображати картинки. Частина коду файлу build.gradle з підключенням усіх бібліотек, що використовуються при розробці клієнтської частини:

```
// Rest client
implementation 'com.squareup.retrofit2:retrofit:2.9.0'
implementation 'com.squareup.retrofit2:adapter-rxjava3:2.9.0'
// HTTP client
implementation "com.squareup.okhttp3:okhttp:4.9.3"
implementation "com.squareup.okhttp3:logging-interceptor:4.9.3"
// Converters for retrofit
implementation 'com.google.code.gson:gson:2.9.0'
implementation 'com.squareup.retrofit2:converter-gson:2.9.0'
// Reactive Java
implementation "io.reactivex.rxjava3:rxjava:3.1.1"
implementation 'io.reactivex.rxjava3:rxandroid:3.0.0'
// For some UI elements
implementation "com.google.android.material:material:1.7.0-alpha01"
// Glide for image loading
implementation 'com.github.bumptech.glide:glide:4.13.0'
```

Можна почати опис із файлів у папці helper. У даній папці міститься два класи. Перший клас ProgressBar використовується майже у всіх класах активностей і призначений для показу вікна завантаження сторінки. Даний клас містить два методи, що призначені для показу вікна завантаження – startProgressBar, а також для його закриття – stopProgressBar. Частина програмного коду класу ProgressBar:

```
private Activity activity;
    private AlertDialog alertDialog;
    public ProgressBar(Activity activity) {
        this.activity = activity;
```

```
    }
    public void startProgressBar() {
        AlertDialog.Builder builder = new
AlertDialog.Builder(this.activity);
LayoutInflater layoutInflater =this.activity.getLayoutInflater();
builder.setView(layoutInflater.inflate(R.layout.progress_bar,null));
        builder.setCancelable(false);
        alertDialog = builder.create();
        alertDialog.show();
    }
    public void stopProgressBar() {alertDialog.dismiss();}
```

Вигляд вікна завантаження при виконанні запиту приведений на рисунку 3.4:

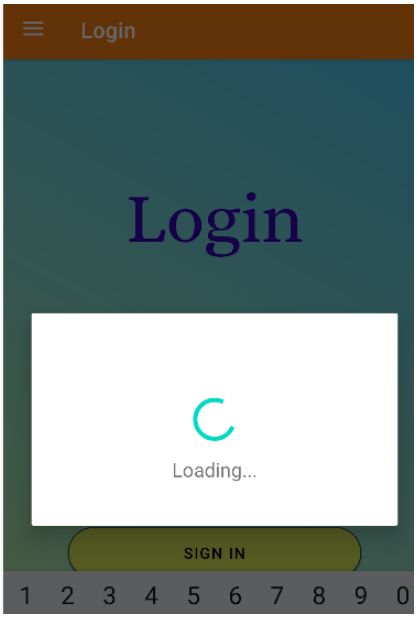


Рисунок 3.4 – Вікно завантаження при відправці запиту на сервер

Інший клас UserLocalData призначений для тимчасового зберігання даних про користувача, а саме його імені, прізвище пошти, паролю, та інформації про те, чи реєстрований користувач. Даний клас, так само як і попередній використовується у великій кількості класів активностей. Усі члени класу UserLocalData є статичними, що дозволяє користуватись функціоналом без

створення додаткових об’єктів. Він виступає своєрідним кешем. Клас містить методи, що допомагають зберігати, видаляти та очищувати дані про користувачів. Якщо користувач реєструє акаунт або входить у нього, здійснюється виклик відповідних методів, що призначені для встановлення відповідних значень. Якщо користувач здійснює вихід із додатку, то дані очищаються. Також існує метод, що перевіряє чи користувач зареєстрований.

Для функціонування бібліотеки Retrofit, що допомагає спілкуватись із сервером, було створено ряд класів. До цього списку входить інтерфейс RestApi та клас RestClient, що містяться у папці api. Всередині інтерфейсу RestApi визначаються методи, що відповідають ідентифікаторам ресурсу, про які знає сервер. Бібліотека Retrofit динамічно створює реалізацію даного інтерфейсу при виклику одного з його методів. Кожен метод містить анотацію, що складається з імені HTTP методу та назви ресурсу за яким буде виконуватись запит. У назві ресурсу можуть вказуватись значення у фігурних дужках, які динамічно підставляються значеннями, що передаються у даний метод у вигляді параметрів, за умови, що дані параметри мають відповідну анотацію @Path у якій вказується значення у фігурних дужках. Також, параметри можуть мати анотацію @Body, що дозволяє передавати відповідні об’єкти класів моделей та поміщати їх у тіло запиту. Дана анотація використовується у таких HTTP методах як PUT і POST. Частина коду класу RestApi:

```
// User's endpoints
@GET("user/{id}")
Observable<BaseResponse<User>> getUser(@Path("id") int userId);
@GET("user/by-email")
Observable<BaseResponse<User>> getUserByEmail(@Query("value") String
email);
@POST("user")
Observable<BaseResponse<User>> saveUser(@Body User user);
@PUT("user/{id}")
```

```
Observable<BaseResponse<User>> updateUser(@Path("id") int userId,
@Body User user);
@DELETE("user/{id}")
Observable<BaseResponse<String>> deleteUser(@Path("id") int userId);
```

Клас RestClient містить логіку для ініціалізації об’єкту бібліотеки Retrofit, а також створення об’єкту HTTP клієнта бібліотеки OkHTTP. Для ініціалізації об’єкту Retrofit потрібно вказати базову адресу, у даному випадку базовою адресою, що відповідає локальному комп’ютеру є адреса 10.0.2.2, також необхідно вказати клас конвертер, у даному випадку використовується бібліотека Gson, також необхідно визначити адаптер за допомогою якого буде відбуватись відправка запитів, у даному випадку адаптером виступає бібліотека RxJava, а також слід вказати HTTP клієнта, яким у нашому випадку буде об’єкт OkHTTP. Для створення об’єкту Retrofit призначений метод createRetrofitClient. Даний метод є статичним, що дозволяє викликати його не створюючи об’єкти класу RestClient. У класі RestClient міститься метод createOkHttpClient, який дозволяє додати заголовок Authorization у об’єкт OkHTTP шляхом виклику інтерсептора. Також, через інтерсептор була додана можливість логування вхідних та вихідних результатів HTTP запитів, що дозволяє краще спостерігати за процесом спілкування клієнту та серверу. Інтерсептори у класі OkHTTP використовуються для зміни запитів перед та після відправки HTTP запитів. Частина коду класу RestClient:

```
private static Retrofit retrofitClient;
public static String credentials = Credentials.basic("test_api",
"123");
public static final String baseUrl = "http://10.0.2.2/";
private RestClient() { }
public static Retrofit createRetrofitClient() {
    if (retrofitClient == null) {
        retrofitClient = new Retrofit.Builder()
                .baseUrl(baseUrl)
```

```
        .addConverterFactory(GsonConverterFactory.create())
.addCallAdapterFactory(RxJava3CallAdapterFactory.create())
                .client(RestClient.createOkHttpClient())
                .build();
    }

    return retrofitClient;
}
private static OkHttpClient createOkHttpClient() {
    OkHttpClient.Builder httpClient = new OkHttpClient.Builder();
    httpClient.addInterceptor(new Interceptor() {
        @NonNull
        @Override
        public Response intercept(@NonNull Chain chain) throws
IOException {
            Request request = chain.request().newBuilder()
                    .addHeader("Authorization", credentials)
                    .build();
            return chain.proceed(request);
        }
    });
    HttpLoggingInterceptor logging = new HttpLoggingInterceptor();
    logging.level(HttpLoggingInterceptor.Level.BODY);
    httpClient.addInterceptor(logging);
    return httpClient.build();}
```

Бібліотека Retrofit широко використовує класи моделей, які призначені для опису структури HTTP відповідей. Вдалий опис кожної відповіді дозволяє автоматично проводити конвертацію відповіді у відповідний об’єкт моделі. У проекті було створено шість класів моделей. Даними класами є BaseResponse, City, Country, Place, PlaceType та User. Структура в усіх класів моделей однакова. Вона складається із опису полів, що відповідають атрибутам відношення, а також публічних методів гетерів та сетерів. Над кожним полем класу, через анотацію

@SerializedName бібліотеки Retrofit вказується реальна назва атрибутів у відношенні. Не важливо, що назва самого поля класа співпадала із назвою атрибута таблиця, важливо лише те, щоб назва атрибуту таблиці співпадала із назвою, що вказана у дужках анотації @SerializedName.

Серед класів моделей, потрібно відмітити параметризований клас BaseResponse, який описує загальну структуру HTTP відповіді. Загальна структура відповіді складається із ключа succees, що містить значення нуль у разі помилки та одиницю при успіху, ключа message, що містить інформацію про запит, а також ключа response, що має динамічно визначену структуру, і може представляти як один об’єкт так і колекцію об’єктів. Частина коду класу BaseResponse:

```
public class BaseResponse <T> {
    @SerializedName("success")
    private int success;
    @SerializedName("response")
    private T response;
    @SerializedName("message")
    private String message;
    public int getSuccess() {
        return success;
    }
    public String getMessage() {
        return message;
    }
    public T getResponse() {
        return response;
    }
}
```

Наостанок, залишились класи активностей, що відповідають за логіку пов’язану із візуальним інтерфейсом, а саме за виклик та обробку різноманітних подій введення з клавіатури та клацання курсором миші по елементам графічного

інтерфейсу користувача. Уся логіка даних класів сконцентрована у методі onCreate, який викликається при рендері екрану.

Клас LoginActivity містить логіку пов’язану із роботою сторінки логіну. Всередині даного класу містяться події, що викликаються після кожного введення користувача. Метод обробки нажаття на клавіші addTextChangedListener викликається для необхідних елементів екрану та ловить введені дані у перевизначеному класом TextWatcher методі onTextChanged. Всередині методу onTextChanged проводиться перевірка введених даних на правильність. У разі не правильного формату введеня, користувачу буде показане повідомлення про помилку у нижній лівій частині елементу введення тексту. Якщо усі дані були введені вірно, то відбувається нажаття на кнопку логіну і відповідний HTTP запит, що провіряє користувача за його поштою та паролем починає своє виконання. Як уже зазначалось, кожен запит виконується за допомогою бібліотеки Retrofit. При створенні об'єкту Retrofit, у метод create передається назва класу інтерфейсу RestApi, що повертає об’єкт реалізації даного інтерфейсу. Через об’єкт, що реалізує інтерфейс, можна здійснювати виклик методів, які визначені у інтерфейсі та прив’язані до конкретного ресурсу. Перед здійсненням запиту викликаються такі методи як subscribeOn, observeOn та subsribe. Ці методи належать бібліотеці RxJava. Принцип виконання запиту полягає у використанні паттерну Observer-Observable. Тобто Observable є деякою подією, що ініціює запит, а Observer є елементом, що підписується на обробку події конкретним Observable. Метод subscribeOn дозволяє визначити потік у якому буде виконуватись Observable. В даному випадку буде створений окремий потік із пулу потоків Schedulers.io(). Метод observeOn визначає потік на якому буде працювати Observer прослуховуючи події, які виконує елемент Observable. Потоком на якому буде відбуватись прослуховування є основний потік Android який відповідає за роботу графічного інтерфейсу. І нарешті метод subscribe виконує зв’язування елементу Observable із елементом Observer. Всередині методу subscribe визначається об’єкт Observer із вказанням типу, що буде повернений після виконання запиту. Також, всередині методу subscribe можна побачити три методи,

які перевизначаються, а саме onSubscribe, onNext, onError та onComplete. Вони викликаються коли Observer підписується на Observable, коли елемент Observable ініціює події, коли виникає помилка при відправці запиту і коли завершується прослуховування подій, які були надіслані від Observable. Головна логіка сконцентрована у методі onNext, який містить перевірку на те, чи успішно був виконаний запит і відповідно до цього показує повідомлення про помилку чи успіх і виконує відповідні дії. В даному випадку, після успішного запиту, данні у класі UserLocalData будуть заповнені інформацією про користувача та буде здійснений перехід до профілю користувача. У класі LoginActivity також міститься метод onDestroy, що викликається при знищенні екрану активності. Даний метод містить виклик методу dispose класу CompositeDisposable, що видаляє ресурси, які були створені в процесі виконання запиту.

Такі класи як RegisterActivity, ChangePasswordActivity, PlaceHotelListActivity та PlaceRestaurantListActivity побудовані за схожим принципом із вищеописаним класом LoginActivity. Слід виділити класи PlaceHotelListActivity та PlaceRestaurantListActivity, що призначені для показу екранів пошуку готелів та ресторанів відповідно та виведення елементів у вигляді списку. Для виведення списку елементів використовуються класи HotelAdapter та RestaurantAdapter відповідно. Вони наслідують клас RecyclerView.Adapter. Вони зберігають та виводять інформацію про масиви об’єктів моделей Country та City.

Для відображення детальної інформації про елементи використовуються класи PlaceRestaurantActivity та PlaceHotelActivity, які отримують данні від класів PlaceRestaurantListActivity та PlaceHotelListActivity і відображають їх у відповідних елементах графічного інтерфейсу.

Задля відображення навігаційного меню та обробки клацання на пункти меню використовувався клас NavigationDrawerActivity. Від даного класу, наслідується ряд класів активностей, яким потрібно мати доступ до навігаційного меню у лівій верхній частині екрану.

Ключові частини коду клієнтської частини подано у додатку Б.2.

3.3 Реалізація розмітки мобільного додатку

Задля реалізації розмітки мобільного додатку, використовуються файли розмітки у форматі XML, які призначені для опису структури та елементів активності у Android Studio. Уся структура була побудована із використанням ConstraintLayout, що дозволяє зручно розміщувати елементи встановлюючи горизонтальні та вертикальні обмеження для них.

Для показу екрану зміни паролю використовується файл розмітки activity_change_password.xml. Він складається із елементів, які призначені для введення тексту, кнопки та верхнього меню. Поля для введення реалізовані за допомогою сторонньої бібліотеки Google яка має назву Material. До неї входить велика кількість елементів, що використовуються при створенні розмітки для екранів мобільного Android додатку. Елемент, що відображає верхню панель має назву MaterialToolbar, який у свою чергу є частиною елементу AppBarLayout. Елемент MaterialToolbar також входить до бібліотеки Material, що можна легко прослідкувати через його назву. Фрагмент коду, який описує елемент верхнього меню MaterialToolbar:

```
<com.google.android.material.appbar.AppBarLayout
    android:id="@+id/appBarLayout"
style="@style/Widget.MaterialComponents.AppBarLayout.PrimarySurface"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:fitsSystemWindows="true"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent">
    <com.google.android.material.appbar.MaterialToolbar
        android:id="@+id/topAppBar"
style="@style/Widget.MaterialComponents.Toolbar.PrimarySurface"
        android:layout_width="match_parent"
```

```
        android:layout_height="?attr/actionBarSize"
        android:background="@color/main_interface_color"
        android:elevation="0dp" />
</com.google.android.material.appbar.AppBarLayout>
```

Фрагмент коду елемента TextInputEditText, що призначений для введення тексту користувачами:

```
<com.google.android.material.textfield.TextInputLayout
        android:id="@+id/old_password_input_layout"
style='@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox'
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginHorizontal="10dp"
        android:layout_marginVertical="30dp"
        android:hint="@string/old_password_label"
        app:counterEnabled="true"
        app:counterMaxLength="8"
        app:endIconMode="password_toggle"
        app:errorEnabled="true"
        app:errorIconDrawable="@null"
        app:helperText="@string/require_label"
        app:hintEnabled="true"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/appBarLayout"
        app:shapeAppearance="@style/CutEdges.InputMaterial"
        app:startIconDrawable="@drawable/custom_lock_icon">
        <com.google.android.material.textfield.TextInputEditText
            android:id="@+id/old_password_input"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:inputType="textPassword"
            android:maxLength="8"
```

```
            android:padding="10dp" />
    </com.google.android.material.textfield.TextInputLayout>
```

Фрагмент коду, що описує елемент кнопки Button:

```
<Button
    android:id="@+id/change_password_btn"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginHorizontal="40dp"
    android:layout_marginTop="25dp"
    android:background="@drawable/base_btn"
    android:text="@string/change_password_label"
    android:textColor="@color/black"
    app:backgroundTint="@null"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toBottomOf="@+id/repeat_new_password_input_
layout" />
```

Екран активності, що використовується для показу сторінки логіну має використовує файл розмітки activity_login.xml. У даному файлі містяться аналогічні елементи, що були представлені у попередньому файлі розмітки, лише додається новий елемент TextView, що призначений для виведення тексту. Фрагмент коду, що описує елемент TextView:

```
<TextView
    android:id="@+id/textView3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="30dp"
    android:textSize="20sp"
    android:textColor="@color/purple_700"
    android:fontFamily="@font/alice"
```

```
    android:text="@string/ask_register_label"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/login_btn" />
```

Для опису головної сторінки активності використовується файл розмітки activity_main.xml, який містить елементи TextView, а також елемент, що призначений для виведення зображень ShapeableImageView, який належить до бібліотеки Material. Фрагмент коду, що наводить елемент ShapeableImageView:

```
<com.google.android.material.imageview.ShapeableImageView
    android:id="@+id/avatar_img"
    android:layout_width="200dp"
    android:layout_height="200dp"
    android:src="@mipmap/logo_main_round"
    android:layout_marginTop="120dp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:shapeAppearanceOverlay="@style/roundedImageView"
    tools:srcCompat="@tools:sample/avatars" />
```

Файл розмітки activity_profile.xml призначений для опису елементів, які містяться на сторінці профілю користувача, activity_register.xml призначений для опису елементів, що містяться на сторінці реєстрації, dropdown_item.xml використовується для опису елементів випадаючого списку, що показує наявні країни та міста. Усі елементи даних файлів, такі як TextView, TextInputEditText та Button, уже були приведені вище, тому додаткових пояснень не потребують.

Файл progress_bar.xml призначений для опису вікна, що з’являється завантаження, що з’являється під час ініціювання запиту на сервер і триває до його повного закінчення. Даний файл містить новий елемент, а саме ProgressBar, який

призначений для відображення кільця, що крутиться. Розмітка елементу ProgressBar подана нижче:

```
<ProgressBar
    android:id="@+id/progressBar"
    style="?android:attr/progressBarStyle"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

Файли розміток activity_place_hotel.xml та activity_place_restaurant.xml мають ідентичну структуру і використовуються для опису розмітки конкретних елементів зі списку, екран яких з’являється після їхнього вибору на сторінці пошуку. У файлах міститься ряд елементів TextView та елемент MaterialToolbar, що забезпечує відображення верхнього меню. Також, присутній елемент ImageView, що є нативним елементом інтерфейсу та використовується для показу картинок. Усі елементи даних файлів розмітки огорнуті елементом ScrollView, що забезпечує додавання можливості прокрутки екрану. Код елементу ImageView:

```
<ImageView
    android:id="@+id/place_image"
    android:layout_width="match_parent"
    android:layout_height="250dp"
    android:scaleType="centerCrop"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.5"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/appBarLayout"
 />
```

Файли activity_place_list_restaurant.xml та activity_place_list_hotel.xml мають однакову структуру і використовуються для опису екрану пошуку, на якому містяться два елементи з випадаючими списками країн та міст, і списки готелів чи ресторанів, які є результатами пошуку пошуку за конкретним містом. Елемент розмітки AutoCompleteTextView відповідає за виведення елементу, що містить випадаючий список, а також дозволяє вводити значення за якими відбувається фільтрація елементів списку. Елемент AutoCompleteTextView загорнений у елемент лейауту TextInputLayout. Розмітка елементу AutoCompleteTextView:

```
<com.google.android.material.textfield.TextInputLayout
    android:id="@+id/dropdown_country_layout"
style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox.
ExposedDropdownMenu"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="20dp"
    android:layout_marginStart="20dp"
    android:layout_marginEnd="150dp"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent">

    <AutoCompleteTextView
        android:id="@+id/dropdown_country_view"
        android:layout_width="match_parent"
        android:verticalScrollbarPosition="right"
        android:maxHeight="40dp"
        android:inputType="text"
        android:scrollbars="vertical"
        android:fadeScrollbars="false"
        android:layout_height="match_parent"
        android:hint="@string/country_choose_label" />
</com.google.android.material.textfield.TextInputLayout>
```

За виведення списку елементів, а також за наявність можливість їх прокрутки відповідає елемент RecyclerView. Код елементу RecyclerView:

```
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/place_list"
    android:layout_width="0dp"
    android:layout_height="0dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    android:layout_marginTop="20dp"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/dropdown_city_layout"
/>
```

activity_place_list_hotel_item.xml та activity_place_list_restaurant_item.xml файли використовуються для опису елементів списку і є ідентичними у своїй структурі. Вони містять новий елемент CardView, що виступає контейнером, у якому зберігаються інформація елементів. Найчастіше даний елемент використовується у списках.

Файл navigation_header.xml забезпечує показ навігаційного меню. Він містить два елементи. Перший елемент include використовується для підключення сторонніх файлів розміток. У даному випадку include підключає файл розмітки content_layout.xml, що містить елемент верхнього меню MaterialToolbar, а також елемент FrameLayout у який вбудовуються інші екрани активностей, що наслідуються від класу активності навігаційного меню. Код елементу FrameLayout:

```
<FrameLayout
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:id="@+id/activityContainer"
    app:layout_constraintBottom_toBottomOf="parent"
```

```
    app:layout_constraintTop_toBottomOf="@id/appBarLayout"/>
```

Іншим елементом файлу navigation_header.xml є NavigationView, що безпосередньо відповідальний за вивід списку елементів на екрані пошуку. Код елементу NavigationView:

```
<com.google.android.material.navigation.NavigationView
        android:id="@+id/nav_view"
        app:headerLayout="@layout/navigation_header"
        app:menu="@menu/navigation_drawer_menu"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_gravity="start" />
```

Елемент NavigationView підключає ще два файли, такі як navigation_header.xml та navigation_drawer_menu.xml. Файл navigation_header.xml забезпечує опис структури верхньої частини навігаційного меню, де зображений логотип та текст. Інший файл під назвою navigation_drawer_menu.xml забезпечує відображення елементів навігаційного меню, які об’єднані у окремі групи.

У папці drawable містяться різноманітні елементи, що допомагають змінювати зовнішній вигляд деяких елементів, а також картинки.

Варто привести код файлу bgr_light.xml, що забезпечує створення фонового малюнку на сторінках логіну, реєстрації та домашній сторінці. Для цього використовується градієнт у якому змішуються два кольори. Код файлу bgr_light.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<shape
xmlns:android="http://schemas.android.com/apk/res/android"
android:shape="rectangle">
    <gradient
```

```
        android:startColor="#4CC4EF"
        android:endColor="#CCEA8F"
        android:angle="225" />
</shape>
```

Файл base_btn.xml використовується для створення фону для деяких кнопок у додатку. Код файлу base_btn.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<selector
xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_enabled="true">
        <shape android:shape="rectangle">
   <stroke android:color="@color/dark" android:width="1dp"/>
            <corners android:radius="50dp" />
            <solid android:color="#D4E157" />
        </shape>
    </item>
</selector>
```

Вище були наведені лише ключові частини інтерфейсу користувача.

3.4 Розробка бази даних

Розробку бази даних полягає у приведенні та розборі SQL команд, які призначені для створення таблиць.

Для створення усіх таблиць бази даних використовується команда CREATE TABLE IF NOT EXISTS після якої вказується ім’я відношення, всередині якого приводяться його атрибути та вказуються їх імена, тип та обмеження. Дана команда виконується лише тоді, коли такого відношення ще не існує в базі даних.

Команда SQL, що створює відношення city:

```
CREATE TABLE IF NOT EXISTS `city`(
     `entity_id` INT(11) UNSIGNED NOT NULL AUTO_INCREMENT,
     `country_id` INT(11) UNSIGNED NOT NULL,
     `name` VARCHAR(100) NOT NULL,
     PRIMARY KEY (`entity_id`),
     CONSTRAINT `fk_city_country_id_country_entity_id`
     FOREIGN KEY (`country_id`) REFERENCES `country` (`entity_id`)
ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

У коді вище, видно, що вказується ряд обмежень. Обмеження NOT NULL означає, що значення даного атрибуту не можуть мати значення null, UNSIGNED використовується з числовими атрибутами та означає те, що значення атрибуту є беззнаковими числами, тобто можуть бути лише додатними, AUTO_INCREMENT зазвичай вказується навпроти ключового атрибуту і забезпечує автоматичне збільшення значення даного атрибуту на одиницю.

Для ключових атрибутів вказується обмеження PRIMARY KEY, яке в даному випадку вказане на рівні таблиці, але може також вказуватись на рівні атрибуту. Обмеження FOREIGN KEY визначає зовнішній ключ у відношенні. В даному випадку зовнішнім ключем виступає атрибут country_id, який посилається на атрибут entity_id таблиці country. Також, вкінці вказуються дії, що потрібно зробити при видаленні чи оновленні значення у головній таблиці, якою у даному випадку виступає таблиця country.

Значення CASCADE забезпечує синхронне видалення та оновлення значень, тобто якщо значення головної таблиці змінюється то значення залежної прямо відображають дані зміни. Вкінці також приводиться ключове слово ENGINE, що вказує движок InnoDB, що використовується для зберігання даних, а також вказується кодування utf8 через ключове слово CHARSET. Таблиця city пов’язана із таблицею country зв’язком один один-до-багатьох, оскільки одній країні може відповідати багато міст.

Команда SQL, що створює відношення country:

```
CREATE TABLE IF NOT EXISTS `country`(
     `entity_id` INT(11) UNSIGNED NOT NULL AUTO_INCREMENT,
     `name` VARCHAR(100) NOT NULL,
     PRIMARY KEY (`entity_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Відношення country не має зв'язків з іншими відношеннями і призначене для зберігання інформації про країни.

Команда SQL, що створює відношення place_type:

```
CREATE TABLE IF NOT EXISTS `place_type`(
     `entity_id` INT(11) UNSIGNED NOT NULL AUTO_INCREMENT,
     `name` VARCHAR(100) NOT NULL,
     PRIMARY KEY (`entity_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Відношення place_type призначене для опису типу місць, яких у даному випадку є всього два, це ресторани та готелі. Відношення не має зв'язків з іншими таблицями у базі даних.

Команда SQL, що створює відношення user:

```
CREATE TABLE IF NOT EXISTS `user`(
     `entity_id` INT(11) UNSIGNED NOT NULL AUTO_INCREMENT,
     `first_name` VARCHAR(100) NOT NULL,
     `last_name` VARCHAR(100) NULL,
     `age` TINYINT(3) UNSIGNED NULL,
     `sex` TINYINT(1) NULL,
     `email` VARCHAR(200) NOT NULL,
     `password` VARCHAR(100) NOT NULL,
     `created_at` DATETIME NULL DEFAULT CURRENT_TIMESTAMP,
```

```
    `updated_at` DATETIME ON UPDATE CURRENT_TIMESTAMP NULL DEFAULT
CURRENT_TIMESTAMP,
    PRIMARY KEY (`entity_id`),
    CONSTRAINT uq_user_email UNIQUE (`email`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Відношення user описує характеристики, якими володіє користувач додатку та не має зв'язків з іншими відношеннями. У даному відношенні присутній унікальний атрибут email навпроти якого стоїть обмеження UNIQUE, що оголошене на рівні таблиці. Кожен юзер повинен мати унікальне значення даного атрибуту, тому саме обмеження UNIQUE гарантує таку унікальність. Якщо буде спроба додавання записів із однаковими значеннями даного поля, то буде видане повідомлення про помилку, оскільки така поведінка суперечить тому, що визначає дане обмеження.

Команда SQL, що створює відношення place:

```
CREATE TABLE IF NOT EXISTS `place`(
    `entity_id` INT(11) UNSIGNED NOT NULL AUTO_INCREMENT,
    `city_id` INT(11) UNSIGNED NOT NULL,
    `place_type_id` INT(11) UNSIGNED NOT NULL,
    `name` VARCHAR(200) NOT NULL,
    `description` TEXT NOT NULL,
    `phone` VARCHAR(50) NOT NULL,
    `working_hours` VARCHAR(200) NOT NULL,
    `address` VARCHAR(200) NOT NULL,
    `images` VARCHAR(50) NOT NULL,
    `created_at` DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    `updated_at` DATETIME ON UPDATE CURRENT_TIMESTAMP NOT NULL
DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (`entity_id`),
    CONSTRAINT `fk_place_city_id_city_entity_id`
    FOREIGN KEY (`city_id`) REFERENCES `city` (`entity_id`) ON
```

```
DELETE CASCADE ON UPDATE CASCADE,
     CONSTRAINT `fk_place_place_type_id_place_type__entity_id`
     FOREIGN KEY (`place_type_id`) REFERENCES `place_type`
(`entity_id`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Відношення place описує загальну інформацію якою володіють готелі та ресторани. Дане відношення має два зв'язки один-до-багатьох. Перший зв'язок встановлюється із відношенням city, оскільки одне місто може належати багатьом ресторанам чи готелям. Другий зв'язок встановлюється із відношенням place_type, оскільки один тип місця можуть мати декілька місць.

3.5 Керівництво користувача

Розроблюваний мобільний застосунок на момент роботи дипломної роботи не є опублікований у PlayMarket, оскільки планується подальша робота над даним програмним продуктом, але у майбутньому додаток буде виставлений у PlayMarket, щоб користувачі мали змогу завантажити та почати користуватись ним.

Після входу у мобільний застосунок перед користувачем відкривається головний екран додатку на якому зображений логотип та коротка інформація, що націлена на ознайомлення користувача із функціоналом програми.

Здійснення переходу до інших екранів здійснюється через навігаційне меню, що знаходиться у верхньому лівому куті вікна програми. Для не зареєстрованих користувачів доступ буде відкритий лише до головного меню програми, сторінки логіну та реєстрації. На сторінці логіну користувач має змогу ввести пошту та пароль, що дозволить йому зайти в акаунт. Якщо дані введені не вірно, то користувачу буде виведене повідомлення про помилку. Помилка може бути пов'язана із не вірним паролем або не існуючою поштою. На екрані логіну є можливість переходу на сторінку реєстрації акаунту. Екран реєстрації містить ряд елементів інтерфейсу, що призначені для введення даних. Таким даними є ім'я,

прізвище, пошта та пароль. Також, на сторінці реєстрації є кнопка, що дозволяє перейти на екран логіну. Екрани логіну та реєстрації містять правила перевірки на не коректно введені значення, повідомлення про які з'являються у нижній лівій частині кожного елементу та підсвічуються червоним кольором. Введені значення полів не повинні бути пустими, пароль повинен складатись із однієї цифри, однієї букви нижнього та верхнього регістру. При реєстрації введені паролі повинні співпадати. Загальна довжина усіх полів введення інформації складає двадцять символів, а для паролів дозволена довжина складає вісім символів. Інформація про максимальну кількість та про кількість уже введених символів користувачами показується у нижній частині полів введення тексту.

Після успішної реєстрації чи логіну, користувач потрапляє на екран профілю, у якому є можливість зміни імені, прізвища та пошти. При зміні одного із згаданих значень, з'являється кнопка UPDATE, що дозволяє оновити дані та продовжити роботу. Також, тут міститься кнопка, що дозволяє видалити аккаунт. Після її натискання буде виведене діалогове вікно, після підтвердження якого буде здійснено видалення акаунту користувача та його перенаправлення до екрану логіну. Також, на сторінці профілю є можливість змінити пароль.

У зареєстрованого користувача є можливість доступу до функцій пошуку готелів та ресторанів. Дані опції можна вибрати у навігаційному меню. Після входу в аккаунт, навігаційне меню трохи змінює свої елементи. На заміну елементам логіну та реєстрації з'являються елементи, що забезпечують перехід до профілю користувача або вихід з аккаунту.

Екран пошуку містить два елементи з випадаючим списком країн та міст. Пошук здійснюється виключно по містам, тому важливо вибрати якесь місто. Після вибору країни, у іншому елементі, що показує список міст, їх кількість зменшується та відбувається показ лише тих міст, що належать до обраної країни, це допомагає спростити пошук необхідного міста. Міста та країни не обов'язково обирати із випадаючого списку вручну, можна провести набір перших символів, після чого будуть виведені варіанти, що задовільняють набраному набору. Після натискання

кнопки пошуку, знизу відображається список готелів чи ресторанів. У кожному елементі списку відображена картинка, назва, а також початковий опис. Після вибору елементу зі списку, відкривається екран, звідки можна отримати детальну інформацію про обране місце. На даному екрані відображається картинка місця, назва, телефон для зв'язку, адреса, робочі години, а також наводиться повний опис. Якщо контент не влазить на усю сторінку, то додається можливість прокрутки екрану, що забезпечує показ усієї інформації.

3.6 Технічні характеристики мобільного додатку

Мобільний додаток здатен працювати лише за наявності стабільного підключення до мережі Інтернет, а також в повній мірі відповідати наступним мінімальним системним вимогам:

– Android версії 10.0;

– 1.5 гб ОЗП;

– 100 Мб пам'яті пристрою;

Рекомендовані системні вимоги:

– Android версії 12.0;

– 2 гб ОЗП;

– 250 Мб пам'яті пристрою.

Отже, результатом написання даного розділу стало розкриття способів програмної реалізації двох незалежних частин додатку, а саме клієнтської та серверної сторін. Була розроблена архітектура інтерфейсу користувача та наведені її основні компоненти. Також, була розроблена структура бази даних, шляхом подання лістингу SQL команд.

Окрім цього, була розроблена детальне керівництво користувача, що описує основні функції додатку та пояснює як взаємодіяти з ними, а також були наведені технічні характеристики мобільного додатку.

4 ТЕСТУВАННЯ МОБІЛЬНОГО ДОДАТКУ

4.1 Аналіз та вибір методів тестування

Існує велика різноманітність методів тестування програмного коду, але не завжди усі з них підходять для перевірки конкретного функціоналу на працездатність.

Опираючись на вищенаведений факт, слід виділити лише ті види тестування, які будуть використані у даному розділі.

Для тестування програмного продукту буде використано три типи тестування, якими є:

– функціональне тестування;

– конфігураційне тестування;

– тестування за принципом «чорного ящику».

Функціональне тестування – це тип тестування програмного забезпечення, яке перевіряє програмну систему на відповідність функціональним вимогам.

Мета функціональних тестів полягає в тому, щоб перевірити кожну функцію додатку на відповідність функціям, що наводяться у інструкції користувача. Даний вид тестування призначений для перевірки вихідного коду програми. Це тестування перевіряє такі елементи як інтерфейс користувача, зв’язок між клієнтом і сервером та інші функції.

Конфігураційне тестування є видом тестування, при якому програмне забезпечення перевіряється за допомогою кількох комбінацій програмного та апаратного забезпечення, щоб оцінити функціональні вимоги та знайти оптимальні конфігурації, за яких програма працює без будь-яких дефектів чи недоліків.

Дана методика тестування проводиться для визначення оптимальних конфігурацій, за яких система чи програма можуть працювати без будь-яких помилок. Таким чином, за допомогою цього тестування виявляється найбільш ефективна конфігурація, яка забезпечить необхідні працездатні умови.

Конфігураційне тестування є актуальним у розробці мобільних застосунків.

Тестування за принципом «чорного ящику» – це метод тестування програмного забезпечення, при якому перевіряються функціональні можливості програмних додатків без знання їх внутрішньої структури та деталей реалізації. Даний вид тестування в основному зосереджується на введені початкових даних та аналізі отриманих результатів і повністю базується на вимогах та специфікаціях програмного забезпечення. Основним джерелом тестування «чорного ящика» є специфікація вимог, що заявлена замовником.

Мобільний застосунок над яким ведеться розробка складається із двох частина, а саме клієнтської та серверної. Беручи це до уваги, необхідно провести тестування для даних частин окремо. Для клієнтської частини доцільно провести функціональне та конфігураційне тестування, а для серверної лише тестування за принципом «чорного ящику».

4.2 Тестування серверної частини

В якості методики тестування серверної частини був обраний принцип тестування «чорний ящик».

При тестуванні не важливо, що як працює сервер зсередини, нам важливо знати відповіді, отримані від сервера, на основі тих запитів, які були відіслані зі сторони клієнта.

Головною функцією серверної частини є надання прозорих інтерфейсів клієнту для спілкування, якими є ідентифікатори ресурсів, що заздалегідь визначені. Тестування полягає у звертанні до даних ресурсів та аналізі результатів відповідей, що повертаються зі сторони серверу.

Для тестування серверної частини протягом усієї розробки використовувалось доволі відоме програмне забезпечення під назвою Postman, що призначене саме для вищеописаних цілей. Дане програмне забезпечення дозволяє легко тестувати різноманітні види API включаючи REST API.

Тестування проводиться для кожного ідентифікатору ресурсу окремо.

Проведемо тестування за принципом «чорний ящик». Визначимо основні сценарії тестування та наведемо їх у вигляді таблиці.

Тестування проводилось шляхом звертання до серверу через кожен ідентифікатор ресурсу. Сценарії тестування, що відповідають методиці «чорний ящик» подані у таблиці 4.1.

Таблиця 4.1 – Тестові сценарії методики тестування «чорний ящик»

Ідентифікатор	Назва ресурсу	HTTP метод	Вхідні дані	Очікуваний результат
В-1	/country	GET	-	Повернення списку усіх країн
В-2	/user/:id		Ідентифікатор користувача за яким повинна здійснюватись вибірка із таблиці	Отримання користувача за вказаним ідентифікатором
В-3	/user		-	Отримання списку усіх користувачів
В-4	/city		-	Повернення списку усіх міст
В-5	/place/hotel/:id		Ідентифікатор міста за яким здійнюється вибірка даних	Повернення списку готелів, що належать до міста, яке відповідає вказаному ідентифікатору
В-6	/place/restaurant/:id		Ідентифікатор міста за яким здійнюється вибірка даних	Повернення списку ресторанів, що належать до міста, яке відповідає вказаному ідентифікатору

Закінчення таблиці 4.1

B-7	/place-type		-	Повернення списку усіх типів місць
B-8	/user/by-email		GET параметр email, щомістить назву пошти	Повернення користувача, якому відповідає введена пошта
B-9	/user/:id	DELETE	Ідентифікатор користувача за яким повинне здійснитись видалення	Повернення результату про успішне видалення вказаного користувача
B-10	/user/:id	PUT	Ідентифікатор користувача за яким повинне здійснитись оновлення даних; також вказуються данні, які необхідно оновити у тілі повідомлення	Повернення результату про успішне оновлення даних вказаного користувача, а також його оновлених даних
B-11	/user	POST	У тілі повідомлення вказується інформація про користувача, що буде додана до бази даних	Повернення результату про успішне додавання даних вказаного користувача, а також його доданих даних

4.3 Тестування клієнтської частини

Задля тестування клієнтської частини було прийняте рішення використовувати такі методики тестування як функціональне та конфігураційне.

Для конфігураційного тестування слід створити емулятори, що дозволять імітувати поведінку реальних пристроїв. Емулятори що були використані відповідають наступним девайсам: Pixel 5, Pixel 3, Pixel 4.

Задля проведення функціонального тестування були виділені деякі сценарії тестування на основі функціональних можливостей, які пропонуються мобільним застосунком. Для прикладу приводиться лише частина тестових сценаріїв, але тестуються усі можливі. Тестові сценарії функціонального тестування для клієнтської частини подані у таблиці 4.2.

Таблиця 4.2 – Тестові сценарії функціонального тестування

Ідентифікатор	Функціональна вимога	Вихідні дані	Очікуваний результат
F-1	Перехід до екрану реєстрації після натиснення на кнопку реєстрації, що знаходиться на екрані логіну	Користувач натискає на кнопку реєстрації, що знаходиться на екрані логіну	Відкривається екран реєстрації
F-2	Перехід до екрану логіну після натиснення на кнопку логіну, що знаходиться на екрані реєстрації	Користувач натискає на кнопку логіну, що знаходиться на екрані реєстрації	Відкривається екран логіну
F-3	Перехід до екрану логіну при виборі елементу логіну із навігаційного меню	Користувач обирає елемент логіну із навігаційного меню	Відкривається екран логіну
F-4	Перехід до екрану реєстрації при виборі елементу реєстрації із навігаційного меню	Користувач обирає елемент реєстрації із навігаційного меню	Відкривається екран реєстрації

Закінчення таблиці 4.2

F-5	Введення не коректних даних на екрані логіну	Користувач вводить не коректні дані на екрані логіну	Після введення не коректних даних на екрані логіну відображаються повідомлення про помилку
F-6	Введення не коректних даних на екрані реєстрації	Користувач вводить не коректні дані на екрані реєстрації	Після введення не коректних даних на екрані реєстрації відображаються повідомлення про помилку
F-7	Перехід до головного екрану після вибору відповідного пункту із навігаційного меню	Користувач вибирає елемент головного екрану із навігаційного меню	Відкривається головний екран
F-8	Перехід до екрану профілю після успішного проходження етапу логіну	Користувач успішно проходить етап логіну	Відкривається екран профілю
F-9	Перехід до екрану профілю після успішного проходження етапу реєстрації	Користувач успішно проходить етап реєстрації	Відкривається екран профілю
F-10	Поява кнопки оновлення даних, після зміни даних на екрані профілю	Користувач змінює дані на екрані профілю	З’являється кнопка для оновлення змінених даних

Оскільки конфігураційне тестування проводиться на різноманітних девайсах, а функціональне тестування призначене для перевірки коректності роботи конкретного функціоналу на різних девайсах, було прийняте рішення сумістити результати функціонального та конфігураційного тестувань у вигляді однієї

таблиці. Результати проведених тестувань серверної та клієнтської частини подані у наступному підрозділі.

4.4 Аналіз результатів тестування клієнтської та серверної частини

Під час виконання тестування методикою «чорного ящику», функціонального та конфігураційного тестування жодних помилок у роботі мобільного додатку не було виявлено.

Результати тестування з використанням методики «чорний ящик» подані у таблиці під номером 4.3.

Таблиця 4.3 – Результати тестування за методикою «чорний ящик»

Ідентифікатор	Назва ресурсу	HTTP метод	Отриманий результат	Результат
B-1	/country	GET	Повернення списку усіх країн	Правильно
B-2	/user/:id		Отримання користувача за вказаним ідентифікатором	Правильно
B-3	/user		Отримання списку усіх користувачів	Правильно
B-4	/city		Повернення списку усіх міст	Правильно
B-5	/place/hotel/:id		Повернення списку готелів, що належать до міста, яке відповідає вказаному ідентифікатору	Правильно

Закінчення таблиці 4.3

В-6	/place/restaurant/:id		Повернення списку ресторанів, що належать до міста, яке відповідає вказаному ідентифікатору	Правильно
В-7	/place-type		Повернення списку усіх типів місць	Правильно
В-8	/user/by-email		Повернення користувача, якому відповідає введена пошта	Правильно
В-9	/user/:id	DELETE	Повернення результату про успішне видалення вказаного користувача	Правильно
В-10	/user/:id	PUT	Повернення результату про успішне оновлення даних вказаного користувача, а також його оновлених даних	Правильно
В-11	/user	POST	Повернення результату про успішне додавання даних вказаного користувача, а також його доданих даних	Правильно

Результати функціональних тестів, що проводились на створених емуляторах подані у таблиці 4.4

Таблиця 4.4 – Результати конфігураційного та функціонального тестування

Назва емулятору	Ідентифікатор тесту									
	F-1	F-2	F-3	F-4	F-5	F-6	F-7	F-8	F-9	F-10
Pixel 5	+	+	+	+	+	+	+	+	+	+
Pixel 4	+	+	+	+	+	+	+	+	+	+
Pixel 3	+	+	+	+	+	+	+	+	+	+

В даному розділі були наведені основні типи тестувань, що використовувались у ході перевірки клієнтської та серверної частин на коректну роботу. Результати тестувань є позитивними, тобто помилок не було виявлено. Компоненти програми та програма в цілому виявились повністю працездатними, оскільки успішно пройшли такі типи тестувань як методика «чорний ящик», функціональне та конфігураційне тестування. Задля кращого візуального подання результатів тестування були створені відповідні таблиці.

ВИСНОВКИ

Під час виконання дипломної роботи було проведено детальний аналіз предметної області, а також огляд існуючих програмних засобів, що мають схожий функціонал із додатком над яким ведеться розробка. Були виділені основні особливості програм аналогів, що допомогло з'ясувати принцип їх роботи та зробити відповідні висновки при написанні власного програмного продукту. Також, були виділені основні вимоги, яким повинен відповідати мобільний застосунок, після чого було розроблене технічне завдання.

Здійснено проектування додатку, завдяки якому була підібрана клієнт-серверна архітектура, а також проведена декомпозиція на дві незалежні частини додатку, а саме клієнт та сервер. Створена архітектура бази даних, яка подана у вигляді фізичної моделі. Також були створені діаграми класів для серверної та клієнтської частин, що дозволило зрозуміти способи їх подальшої реалізації. Відбулось проектування інтерфейсу користувача у якому наводяться зовнішній вигляд екранів додатку, а також показані схеми переходів між ними для зареєстрованих та не зареєстрованих користувачів. Були наведені та визначені основні технології та інструменти, що використовуються протягом усієї розробки програмного забезпечення.

На основі зробленого проектування здійснено реалізацію серверної та клієнтської частин, а також структури бази даних. Було розроблене керівництво користувача, а також неведені технічні характеристики додатку. Для розробленого мобільного застосунку проведене тестування з використанням методики «чорний ящик», функціонального та конфігураційного тестування.

В результаті роботи над дипломним проектом був розроблений мобільний додаток, що працює на операційній системі Android, який допомагає шукати інформацію про готелі та ресторани та виводити її у зручному для користувача вигляді. Додаток оснащений доволі простим, але привабливим дизайном. Завдяки дипломному проекту, вдалось покращити та набути нові навички при розробці

програмного забезпечення, що допоможе мені у майбутніх проектах, а також у подальшій роботі по спеціальності. Після захисту дипломного проекту, планується продовження роботи над програмним продуктом задля покращення продуктивності його роботи, а також для додавання нового функціоналу, що дозволить привернути увагу користувачів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Л. П. Бедратюк. Дипломний проект : методичні вказівки щодо його виконання для студентів спеціальності 121 «Інженерія програмного забезпечення» / Л. П. Бедратюк, Г. І. Радельчук, Ю. В. Форкун, О. М. Яшина. – Хмельницький: ХНУ, 2020. – 77 с.

2. What Is A Mobile Application? [Електронний ресурс] – Режим доступу до ресурсу: https://www.emizentech.com/blog/what-is-a-mobile-application.html

3. Android vs iOS: Which mobile platform is best for app development? [Електронний ресурс] – Режим доступу до ресурсу: https://www.techaheadcorp.com/blog/android-vs-ios/#:~:text=Both%20operating%20systems%20come%20with,tighter%20privacy%20and%20security%20controls

4. Difference Between App Store vs Google Play Store [Електронний ресурс] – Режим доступу до ресурсу: https://citrusbits.com/difference-app-store-vs-google-play-store-2/

5. 5 things you need to know about TripAdvisor [Електронний ресурс] – Режим доступу до ресурсу: https://www.trekksoft.com/en/blog/5-things-to-know-about-tripadvisor-in-2018

6. 15 Things You May Not Know About Yelp [Електронний ресурс] – Режим доступу до ресурсу: https://www.searchenginejournal.com/yelp-facts/355044/#close

7. What Is Foursquare and How Does It Work? [Електронний ресурс] – Режим доступу до ресурсу: https://techboomers.com/t/what-is-foursquare-how-it-works

8. Why use a UML diagram? [Електронний ресурс] – Режим доступу до ресурсу: https://www.lucidchart.com/pages/uml-use-case-diagram

9. 10 Common Software Architectural Patterns in a nutshell [Електронний ресурс] – Режим доступу до ресурсу: https://towardsdatascience.com/10-common-software-architectural-patterns-in-a-nutshell-a0b47a1e9013

10. REST APIs [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/cloud/learn/rest-apis

11. What is Class Diagram? [Електронний ресурс] – Режим доступу до ресурсу:https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/

12. Different Types of Testing in Software [Електронний ресурс] – Режим доступу до ресурсу: https://www.perfecto.io/resources/types-of-testing

13. Software Testing Methodologies [Електронний ресурс] – Режим доступу до ресурсу: https://smartbear.com/learn/automated-testing/software-testing-methodologies/

14. MariaDB vs MySQL: A Database Technologies Rundown [Електронний ресурс] – Режим доступу до ресурсу: https://kinsta.com/blog/mariadb-vs-mysql/

15. Apache Vs NGINX – Which Is The Best Web Server for You? [Електронний ресурс] – Режим доступу до ресурсу: https://serverguy.com/comparison/apache-vs-nginx/

16. Client/Server Architecture [Електронний ресурс] – Режим доступу до ресурсу: https://www.techopedia.com/definition/438/clientserver-architecture

17. 5 essential patterns of software architecture [Електронний ресурс] – Режим доступу до ресурсу: https://www.redhat.com/architect/5-essential-patterns-software-architecture

ДОДАТОК А

(Обов’язковий)

ТЕХНІЧНЕ ЗАВДАННЯ

1. Найменування та область застосування

Найменування довідково-інформаційна система «Easy Trip» для пошуку місця відпочинку у вигляді мобільного додатку. Програмний продукт націлений на забезпечені інформації про ресторани та готелі відповідно до вибраної країни чи міста у зручному вигляді. При виконанні пошуку, повинен виводитись список готелів чи ресторанів. Після вибору елементу зі списку повинен відображатись екран на якому міститиметься детальна інформація про обраний елемент. Також, повинна надаватись можливість логіну та реєстрації.

2. Підстави для розробки

Підставою для розробки є “Завдання на дипломний проект”, яке було затверджене завідуючим кафедри інженерії програмного забезпечення.

3. Призначення розробки

Функціональне призначення: безперебійна робота, забезпечення користувачів даними про ресторани та готелі, можливість реєстрації та авторизації у системі, функція зміни початково введених даних для зареєстрованих користувачів. Експлуатаційне призначення: встановлення додатку може бути здійснене лише на операційну систему Android. Робота додатку здійснюється за допомогою доступу до мережі Інтернет.

4. Вимоги до програми

4.1 Вимоги до функціональних характеристик

Клієнтська та серверні частини повинні забезпечувати:

– інтуїтивно-зрозумілий інтерфейс користувача;

– функції авторизації та реєстрації;

– валідацію введених даних;

– модифікацію даних для зареєстрованих юзерів;

– робота із базою даних;

– перевірка вхідних запитів на коректність із сторони веб-сервера;

– відображення основної інформації про готелі та ресторани.

4.2 Вимоги до надійності:

– обмеження доступу до даних, шляхом впровадження базової перевірки логіну та паролю на стороні серверу;

– надійність даних які передаються мережею;

– можливість відміни ланцюжка операцій пов’язаних із модифікацією інформації у базі даних при непередбачуваних ситуаціях.

4.3 Умови експлуатації

Розроблюваний програмний продукт здатен працювати лише на смартфонах, які оснащені операційною системою Android. Доступ до усіх функцій додатку здійснюється лише за наявності мережі.

4.4 Вимоги до складу та параметрів технічних засобів

Мінімальні системні вимоги додатку:

– Android версії 10.0;

– 1.5 гб ОЗП;

– 100 Мб пам’яті пристрою;

Рекомендовані системні вимоги:

– Android версії 12.0;

– 2 гб ОЗП;

– 250 Мб пам’яті пристрою;

4.5 Вимоги до інформаційної та програмної сумісності

Мова програмування, яка застосовується при розробці мобільного додатку є мовою високого рівня Java, що є нативною мовою для розробки мобільних застосунків для операційної системи Android. Тому додаток повинен працювати на усіх смартфонах, які оснащені операційною системою Android.

4.6 Вимоги до програмної документації

При передачі мобільного додатку замовнику, слід передати наступні матеріали, які входять у програмну документацію:

– схематичне представлення структури програми та її компонентів;

– програмна система;

– попередження про можливі помилки в роботі;

– інструкція користувача;

– технічне завдання;

– детальний опис усіх функцій програмного продукту;

– текст програми, який містить її повну реалізацію.

4.7 Стадії та етапи розробки

В ході роботи над мобільним додатком будуть пройдені такі стадії та етапи розробки як показано на таблиці А.1 нижче.

Таблиця А.1 – Стадії та етапи розробки

Стадії розробки	Етапи робіт	Зміст робіт
Технічне завдання 02.01.22 – 31.01.22	Розробка детального технічного завдання	Загальний опис програмного забезпечення; підстава і призначення розробки; розробка вимог до ПЗ
Детальний проект 01.02.22 – 28.02.22	Планування та створення детального проекту	Детальне проектування усіх програмних модуль; розробка візуального інтерфейсу додатку; проектування структури бази даних; використання діаграм для опису основних компонентів додатку

Закінчення таблиці А.1

Програмна реалізація 01.03.22 – 10.04.22	Розробка програмного продукту	Реалізація програмного продукту з використанням усіх особливостей , які були описані в технічному завданні та детальному проекті ПЗ; розробка допоміжної документації
Тестування програмного застосунку 11.04.22 – 30.04.22	Проведення тестування програми	Проведення тестування реалізованого додатку; виправлення помилок виявлених в ході тестування
Впровадження	Завершення роботи над проектом	Випуск готового мобільного додатку

4.8 Порядок контролю і приймання

Процедура контролю якості та приймання кінцевого програмного продукту здійснюється за допомогою тестування його функцій задля виявлення недоліків у роботі додатку.

ДОДАТОК Б

(Обов’язковий)

КОД (ЛІСТИНГ) ПРОГРАМИ

Б.1 Програний код серверної частини додатку

Вміст файлу index.php:

```
<?php
use Core\Bootstrap;
header("Access-Control-Allow-Origin: *");
header("Content-Type: application/json; charset=utf-8");
header("Access-Control-Allow-Methods: GET,POST,PUT,DELETE");
require 'src/bootstrap.php';
$bootstrap = new Bootstrap();
$bootstrap->run($_SERVER);
```

Вміст файлу config.json:

```
<?php
return [
    'POST' => [
        'user' => [
            '/user' => [
                'class' => 'Model\UserRepository',
                'method' => 'save'
            ]
        ],
    ],
    'GET' => [
        'user' => [
            '/user/:id' => [
                'class' => 'Model\UserRepository',
                'method' => 'getById'
            ],
            '/user' => [
                'class' => 'Model\UserRepository',
                'method' => 'getList'
            ],
            '/user/by-email' => [
                'params' => true,
                'class' => 'Model\UserRepository',
                'method' => 'getByEmail'
            ]
        ],

        'country' => [
            '/country' => [
```

```
                'class' => 'Model\CountryRepository',
                'method' => 'getList'
            ],
        ],
        'city' => [
            '/city' => [
                'class' => 'Model\CityRepository',
                'method' => 'getList'
            ],
        ],
        'place' => [
            '/place/hotel/:id' => [
                'class' => 'Model\PlaceRepository',
                'method' => 'getHotelByCityId'
            ],
            '/place/restaurant/:id' => [
                'class' => 'Model\PlaceRepository',
                'method' => 'getRestaurantByCityId'
            ],
        ],
        'place-type' => [
            '/place-type' => [
                'class' => 'Model\PlaceTypeRepository',
                'method' => 'getList'
            ],
        ]
    ],
    'DELETE' => [
        'user' => [
            '/user/:id' => [
                'class' => 'Model\UserRepository',
                'method' => 'delete'
            ]
        ],
    ],
    'PUT' => [
        'user' => [
            '/user/:id' => [
                'class' => 'Model\UserRepository',
                'method' => 'update'
            ]
        ],
    ]
];
```

Вміст файлу composer.json:

```
{
    "name": "anonym/easytrip",
    "description": "REST Api for EasyTrip Android project",
    "type": "project",
    "authors": [
        {
            "name": "Oliinyk Pavlo",
            "email": "test@mail.com"
        }
    ],
    "minimum-stability": "stable",
    "autoload": {
```

```
            "psr-0": {
                "": "src/"
            }
        },
        "require": {
            "ext-pdo": "*"
        }
    }
```

Вміст файлу .htaccess:

```
## Enable rewrites
 RewriteEngine on
 ## Rewrite everything else to index.php
 RewriteCond %{REQUEST_URI} !^/img/
 RewriteRule .* index.php [L]
```

Вміст файлу bootstrap.php:

```
<?php

use Core\Config\ApiConfiguration;

require_once 'vendor/autoload.php';

// Enable all levels of reporting exceptions
error_reporting(E_ALL);

// Check PHP version availability
if (!defined('PHP_VERSION_ID') || PHP_VERSION_ID < 80000) {
    http_response_code(500);
    exit(json_encode(['success' => 0, 'message' => "The API doesn't support
the PHP version lower than 8.0.0"]));
}
// Basic authorization check
if (empty($_SERVER['PHP_AUTH_USER']) || $_SERVER['PHP_AUTH_USER'] !==
ApiConfiguration::API_LOGIN ||
    empty($_SERVER['PHP_AUTH_PW']) || $_SERVER['PHP_AUTH_PW'] !==
ApiConfiguration::API_PASSWORD) {
    http_response_code(401);
    exit(json_encode(['success' => 0, 'message' => 'Provided credentials are
wrong']));
}

date_default_timezone_set('Europe/Kiev');
```

Клас ApiConfiguration.php:

```
<?php

namespace Core\Config;
```

```
class ApiConfiguration
{
    public const API_LOGIN = 'test_api';
    public const API_PASSWORD = '123';

    public const HTTP_METHOD_POST = 'POST';
    public const HTTP_METHOD_PUT = 'PUT';
    public const HTTP_METHOD_GET = 'GET';
    public const HTTP_METHOD_DELETE = 'DELETE';
}
```

Клас DatabaseConfiguration.php:

```
<?php

namespace Core\Config;

class DatabaseConfiguration
{
    public const HOST = 'localhost';
    public const DB_NAME = 'easytrip';
    public const DB_USERNAME = 'root';
    public const DB_PASSWORD = '';
}
```

Клас AbstractModel.php:

```
<?php
declare(strict_types=1);
namespace Core;
use PDO;
abstract class AbstractModel extends DataObject implements ModelInterface
{
    /**
     * @var string
     */
    protected $tableName;
    /**
     * @var Logger
     */
    protected $logger;
    /**
     * @var string
     */
    protected $idFieldName;
    /**
     * @var DatabaseConnector
     */
    protected $databaseConnector;
    public function __construct()
    {
        $this->logger = new Logger();
        $this->databaseConnector = new DatabaseConnector();
    }
    /**
     * Basic model initialization data
```

```
             *
             * @param $tableName
             * @param $idFieldName
             * @return void
             */
            public function _init($tableName, $idFieldName) {
                $this->tableName = $tableName;
                $this->idFieldName = $idFieldName;
            }
            /**
             * Check whether model is initialized with basic data
             *
             * @return bool
             */
            public function isInitialized(): bool
            {
                if (empty($this->tableName) || empty($this->idFieldName)) {
                    http_response_code(500);
                    $this->logger->log( "The data for entity $this->tableName hasn't
been initialized properly");
                    exit(json_encode(['success' => 0, 'message' => 'Internal server
error. Try again later']));
                }
                return true;
            }
            /**
             * Get database connection
             *
             * @return PDO
             */
            public function getConnection(): PDO
            {
                return $this->databaseConnector->getConnection();
            }
            /**
             * Get model
             *
             * @param int|null $entityId
             * @param string $custom
             * @return array|false
             */
            public function load(int $entityId = null, string $custom = ''):
bool|array
            {
                $success = false;
                $this->isInitialized();
                $this->isTableExist();
                $connection = $this->getConnection();
                if (!$custom) {
                    if ($entityId) {
                        $sql = "SELECT * FROM $this->tableName WHERE $this-
>idFieldName = $entityId";
                    } else {
                        $sql = "SELECT * FROM $this->tableName";
                    }
                } else {
                    $sql = $custom;
                }
                $statement = $connection->prepare($sql);
                if ($statement) {
                    try {
                        $connection->beginTransaction();
                        $statement->execute();
```

```
                    if ($entityId) {
                        $success = $statement->fetch(PDO::FETCH_ASSOC);
                    } else {
                        $success = $statement->fetchAll(PDO::FETCH_ASSOC);
                    }
                    $connection->commit();
                } catch (\Exception $error) {
                    $connection->rollBack();
                    $this->logger->log('Error during selecting model data ' .
__CLASS__, [$error->getMessage()]);
                    http_response_code(500);
                    exit(json_encode(['success' => 0, 'message' => $error-
>getMessage()]));
                }
            }
            $this->databaseConnector->closeConnection();
            return $success;
        }
        /**
         * @param int $entityId
         * @return array
         */
        public function update(int $entityId): array
        {
            $success = [];
            $this->isInitialized();
            $this->isTableExist();
            if ($entityId) {
                $isEntityExist = $this->load($entityId);
                if (!$isEntityExist) {
                    http_response_code(400);
                    exit(json_encode(['success' => 0, 'message' =>
sprintf('Requested entity with %s value doesn\'t exist',$this->idFieldName)]));
                }
            }
            $connection = $this->getConnection();
            $data = $this->prepareDataForTable($this);
            if (!$data) {
                return $success;
            }
            unset($data[$this->idFieldName]);
            $updatedValues = implode(',', array_map(
                function ($v, $k) {
                    $v = !is_string($v) ?: "'$v'";
                    return "$k=$v";
                },
                $data,
                array_keys($data)
            ));
            $sql = "UPDATE $this->tableName SET {$updatedValues} WHERE $this-
>idFieldName = $entityId";
            $statement = $connection->prepare($sql);
            if ($statement) {
                try {
                    $connection->beginTransaction();
                    $this->beforeUpdate();
                    $success = $statement->execute();
                    $this->afterUpdate();
                    $connection->commit();
                    if ($success) {
                        $success = $this->load($entityId);
                    } else {
                        $success = [];
```

```
                    }
                } catch (\Exception $error) {
                    $this->databaseConnector->closeConnection();
                    $connection->rollBack();
                    $this->logger->log('Error during saving model data ' .
__CLASS__, [$error->getMessage()]);
                    http_response_code(500);
                    exit(json_encode(['success' => 0, 'message' => $error-
>getMessage()]));
                }
            }
            $this->databaseConnector->closeConnection();
            return $success;
        }
        /**
         * Create new model
         *
         * @return array
         */
        public function save(): array
        {
            $success = [];
            $this->isInitialized();
            $this->isTableExist();
            $connection = $this->getConnection();
            $data = $this->prepareDataForTable($this);
            if (!$data || !is_array($data)) {
                return $success;
            }
            $attributes = implode(',', array_keys($data));
            $values =  implode(',', array_fill(0, count($data), '?'));
            $bind = array_values($data);
    //        $sql = "INSERT INTO  $this->tableName ({$attributes}) VALUES
({$values}) ON DUPLICATE KEY UPDATE {$updatedValues}";
            $sql = "INSERT INTO  $this->tableName ({$attributes}) VALUES
({$values})";
            $statement = $connection->prepare($sql);
            if ($statement) {
                try {
                    $connection->beginTransaction();
                    $this->beforeSave();
                    $statement = $connection->prepare($sql);
                    $success = $statement->execute($bind);
                    $this->afterSave();
                    $lastInsertedId = (int)$connection->lastInsertId();
                    $connection->commit();
                    if ($success && $lastInsertedId) {
                        $success = $this->load($lastInsertedId);
                    } else {
                        $success = [];
                    }
                } catch (\Exception $error) {
                    $connection->rollBack();
                    $this->logger->log('Error during saving model data ' .
__CLASS__, [$error->getMessage()]);
                    http_response_code(500);
                    exit(json_encode(['success' => 0, 'message' => $error-
>getMessage()]));
                }
            }
            $this->databaseConnector->closeConnection();
            return $success;
        }
```

```
        /**
         * Delete model
         *
         * @param int $entityId
         * @return bool
         */
        public function delete(int $entityId): bool
        {
            $success = false;
            $this->isInitialized();
            $this->isTableExist();
            $connection = $this->getConnection();
            $sql = "DELETE FROM $this->tableName WHERE $this->idFieldName =
$entityId";
            $statement = $connection->prepare($sql);
            if ($statement) {
                try {
                    $connection->beginTransaction();
                    $success = $statement->execute();
                    $connection->commit();
                } catch (\Exception $error) {
                    $connection->rollBack();
                    $this->logger->log('Error during deleting model data' .
__CLASS__, [$error->getMessage()]);
                    http_response_code(500);
                    exit(json_encode(['success' => 0, 'message' => $error-
>getMessage()]));
                }
            }
            $this->databaseConnector->closeConnection();
            return $success;
        }
        /**
         * Retrieving table attributes and mapping them to model's values
         *
         * @param DataObject $object
         * @return array|null
         */
        public function prepareDataForTable(DataObject $object): ?array
        {
            $data = [];
            $connection = $this->getConnection();
            $this->isTableExist();
            $sql = "DESCRIBE $this->tableName";
            try {
                $connection->beginTransaction();
                $statement = $this->getConnection()->query($sql);
                if (!$statement) {
                    $connection->rollBack();
                    $this->logger->log('Error during describing table ' . $this-
>tableName);
                    http_response_code(500);
                    exit(json_encode(['success' => 0, 'message' => 'Internal
server error. Try again later']));
                }
                $fields = $statement->fetchAll(PDO::FETCH_ASSOC);
                $connection->commit();
            } catch (\Exception $error) {
                $connection->rollBack();
                $this->logger->log('Error during preparing data for ' .
__CLASS__, [$error->getMessage()]);
                http_response_code(500);
```

```
                exit(json_encode(['success' => 0, 'message' => $error-
>getMessage()]));
            }
            foreach ($fields as $field) {
                if ($object->hasData($field['Field'])) {
                    $fieldValue = $object->getData($field['Field']);
                    if (!is_null($fieldValue)) {
                        $data[$field['Field']] = $fieldValue;
                    }
                }
            }
            return $data;
        }
        /**
         * @return $this
         */
        protected function beforeSave(): static
        {
            return $this;
        }
        /**
         * @return $this
         */
        protected function afterSave(): static
        {
            return $this;
        }
        /**
         * @return $this
         */
        protected function beforeUpdate()
        {
            return $this;
        }
        /**
         * @return $this
         */
        protected function afterUpdate()
        {
            return $this;
        }
        protected function isTableExist()
        {
            try {
                $this->getConnection()->query('SELECT 1 FROM ' . $this-
>tableName);
            } catch (\Exception $error) {
                $this->logger->log(sprintf("The table %s doesn't exist", $this-
>tableName), [$error->getMessage()]);
                http_response_code(500);
                exit(json_encode(['success' => 0, 'message' => $error-
>getMessage()]));
            }
        }
    }
```

Клас Bootstrap.php:

```
<?php
```

```
declare(strict_types=1);
namespace Core;
use Core\Config\ApiConfiguration;
class Bootstrap
{
    public string $requestUri;
    /**
     * @var string
     */
    private $requestMethod;
    /**
     * @var mixed
     */
    private $requestHasParams = false;
    private $endpointHasParams = false;
    /**
     * @var mixed
     */
    private $requestParamValue = null;
    /**
     * @var int
     */
    private $requestedId;
    /**
     * @var Logger $logger
     */
    private $logger;
    public function run($requestData)
    {
        $this->initLogger();
        $this->processRequest($requestData);
    }
    /**
     * @param $requestData
     * @return void
     */
    public function processRequest($requestData)
    {
        $this->requestUri =
rtrim(parse_url($requestData['REQUEST_URI'],PHP_URL_PATH), '/');
        if (!empty($_GET)) {
            if (!empty($_GET['value'])) {
                $this->requestHasParams = true;
                $this->requestParamValue = $_GET['value'];
            }
        }
        if (!$this->requestUri) {
            http_response_code(400);
            exit(json_encode(['success' => 0, 'message' => "You didn't enter
any endpoint"]));
        }
        if (empty($requestData["REQUEST_METHOD"])) {
            http_response_code(500);
            exit(json_encode(['success' => 0, 'message' => 'The HTTP method
is undefined']));
        } else {
            $this->requestMethod = $requestData["REQUEST_METHOD"];
        }
        $uri = array_values(array_filter(explode('/', $this->requestUri)));
        if (count($uri) < 1 || count($uri) > 3) {
            http_response_code(400);
            exit(json_encode(['success' => 0, 'message' => 'The endpoint is
incorrect']));
```

```
            }
            switch ($this->requestMethod) {
                case ApiConfiguration::HTTP_METHOD_PUT:
                case ApiConfiguration::HTTP_METHOD_POST:
                    $action = $this->requestMethod ===
ApiConfiguration::HTTP_METHOD_PUT ? 'update' : 'save';
                    $jsonData = json_decode(file_get_contents("php://input"),
true);
                    if (empty($jsonData)) {
                        http_response_code(400);
                        exit(json_encode(['success' => 0, 'message' => 'Request
body is empty']));
                    }
                    $callData = $this->match($uri);
                    $callData = $this->validateRepositoryCall($callData);
                    if ($this->requestMethod ===
ApiConfiguration::HTTP_METHOD_POST) {
                        $response = (new $callData['class']())-
>{$callData['method']}($jsonData);
                    } else {
                        if (empty($this->requestedId)) {
                            http_response_code(400);
                            exit(json_encode(['success' => 0, 'message' => 'You
should specify the entity id in the end of URL']));
                        }
                        $response = (new $callData['class']())-
>{$callData['method']}($jsonData, $this->requestedId);
                    }
                    if (empty($response)) {
                        http_response_code(400);
                        exit(json_encode(['success' => 0, 'message' => "The
error during $action operation has happened"]));
                    }
                    break;
                case ApiConfiguration::HTTP_METHOD_DELETE:
                    $response = null;
                    $callData = $this->match($uri);
                    $callData = $this->validateRepositoryCall($callData);
                    if (count($uri) == 2) {
                        if (empty($this->requestedId)) {
                            http_response_code(400);
                            exit(json_encode(['success' => 0, 'message' => 'You
should specify the entity id in the end of URL']));
                        }
                        $response = (new $callData['class']())-
>{$callData['method']}($this->requestedId);
                    }
                    if (empty($response)) {
                        http_response_code(400);
                        exit(json_encode(['success' => 0, 'message' => "The
requested data doesn't exist, so it wasn't deleted"]));
                    }
                    break;
                case ApiConfiguration::HTTP_METHOD_GET:
                    $callData = $this->match($uri);
                    $callData = $this->validateRepositoryCall($callData);
                    if (!empty($this->requestedId)) {
                        $response = (new $callData['class']())-
>{$callData['method']}($this->requestedId);
                    } elseif (!empty($this->requestParamValue)) {
                        $response = (new $callData['class']())-
>{$callData['method']}($this->requestParamValue);
                    } elseif (empty($this->endpointHasParams)) {
```

```
                        $response = (new $callData['class']())-
>{$callData['method']}();
                    }
                    if (empty($response)) {
                        http_response_code(400);
                        exit(json_encode(['success' => 0, 'message' => "The
requested entity doesn't exist"]));
                    }
                    break;
                default:
                    http_response_code(405);
                    exit(json_encode(['success' => 0, 'message' => 'Requested
HTTP method are not allowed. You can use only GET, PUT, POST or DELETE']));
            }
            http_response_code(200);
            echo json_encode(['success' => 1, 'message' => 'Success', 'response'
=> $response], JSON_UNESCAPED_SLASHES);
        }
        /**
         * Validate data for calling repository method
         *
         * @param $callData
         * @return array|void
         */
        public function validateRepositoryCall($callData)
        {
            // data existence check
            if (!is_array($callData) || empty($callData)) {
                $this->logger->log(sprintf('Call data is incorrect %s',
$callData));
                http_response_code(500);
                exit(json_encode(['success' => 0, 'message' => 'Internal server
error. Try again later']));
            }
            // class existence check
            if (empty($callData['class']) || !class_exists($callData['class']))
{
                $this->logger->log("Requested class " . $callData['class'] . "
doesn't exist");
                http_response_code(500);
                exit(json_encode(['success' => 0, 'message' => 'Internal server
error. Try again later']));
            }

            // method existence check
            if (empty($callData['method']) || !method_exists($callData['class'],
$callData['method'])) {
                $this->logger->log("Requested method " . $callData['method'] . "
doesn't exist");
                http_response_code(500);
                exit(json_encode(['success' => 0, 'message' => 'Internal server
error. Try again later']));
            }
            return $callData;
        }
        /**
         * Match endpoint to model's repository method
         *
         * @param $uri
         * @return mixed|string|void
         */
        public function match($uri) {
            $apiConfiguration = include 'src/etc/config.php';
```

```
            foreach ($apiConfiguration[$this->requestMethod] as
$endpointFirstPart => $routes) {
                // requested uri
                $requestedUriFirstPart = $uri[array_key_first($uri)];
                $requestedUriLastPart = $uri[array_key_last($uri)];
                if ($endpointFirstPart === $requestedUriFirstPart) {
                    foreach($routes as $endpoint => $content) {
                        $this->endpointHasParams = false;
                        // endpoint defined in the configuration
                        $currentEndpoint =
array_values(array_filter(explode('/', $endpoint)));
                        if (count($currentEndpoint) === count($uri)) {
                            $this->endpointHasParams =
!empty($content['params']);
                            $endpointHasId = in_array(':id', $currentEndpoint);
                            // Endpoints can have the same size, but be
different
                            if ($endpointHasId) {
                                if (!(str_replace(':id', $requestedUriLastPart,
$endpoint) === $this->requestUri)) {
                                    continue;
                                }
                            }
                            // if endpoint has :id and the last part of the
request is numeric it is a match
                            if (is_numeric($requestedUriLastPart)) {
                                if ($endpointHasId) {
                                    $this->requestedId =
(int)$requestedUriLastPart;
                                    return $content;
                                } else {
                                    continue;
                                }
                            }
                            if ($this->endpointHasParams && $this->requestUri
=== $endpoint) {
                                if (!empty($this->requestParamValue)) {
                                    return $content;
                                } else {
                                    http_response_code(400);
                                    $errorMessage = "You should have the 'value'
parameter in your url with appropriate value";
                                    exit(json_encode(['success' => 0, 'message'
=> $errorMessage]));
                                }
                            }
                            if (!$this->requestHasParams && $this->requestUri
=== $endpoint) {
                                return $content;
                            }
                        }
                    }
                    break;
                }
            }
            http_response_code(400);
            $errorMessage = sprintf("Requested endpoint %s for HTTP method %s
doesn't exist. Try to change endpoint", $this->requestUri, $this->requestMethod);
            exit(json_encode(['success' => 0, 'message' => $errorMessage]));
        }
        /**
         * @return void
         */
```

```
    public function initLogger()
    {
        $this->logger = new Logger();
    }
}
```

Клас DatabaseConnector.php:

```
<?php
declare(strict_types=1);
namespace Core;
use Core\Config\DatabaseConfiguration;
use PDO;
use PDOException;
class DatabaseConnector
{
    private $connection;
    /**
     * @var string
     */
    private string $host;
    /**
     * @var string
     */
    private $db_name;
    /**
     * @var string
     */
    private $username;
    /**
     * @var string
     */
    private string $password;
    /**
     * @var Logger
     */
    private $logger;
    /**
     * DatabaseConnector constructor
     */
    public function __construct()
    {
        $this->host = DatabaseConfiguration::HOST;
        $this->db_name = DatabaseConfiguration::DB_NAME;
        $this->username = DatabaseConfiguration::DB_USERNAME;
        $this->password = DatabaseConfiguration::DB_PASSWORD;
        $this->logger = new Logger();
        $this->initConnection();
    }
    /**
     * @return void
     */
    private function initConnection()
    {
        try {
            $this->connection = new PDO("mysql:host=" . $this->host .
";dbname=" . $this->db_name, $this->username, $this->password);
            $this->connection->exec("set names utf8");
        } catch (PDOException $error) {
```

```
                $this->closeConnection();
                http_response_code(500);
                $this->logger->log('Database connection exception has been
thrown', [$error->getMessage()]);
                exit(json_encode(['success' => 0, 'message' => 'Internal server
error. Try again later']));
            }
        }
        /**
         * @return PDO
         */
        public function getConnection(): PDO
        {
            if (!$this->connection) {
                $this->initConnection();
            }
            return $this->connection;
        }
        /**
         * @return void
         */
        public function closeConnection()
        {
            $this->connection = null;
        }
    }
```

Клас DataObject.php:

```
<?php
namespace Core;
class DataObject
{
    /**
     * Object attributes
     *
     * @var array
     */
    protected $_data = [];
    /**
     * Overwrite or add data to the object
     *
     * if $key is an array data will be overwritten
     * if $key is a string data will be added
     *
     * @param $key
     * @param $value
     * @return $this
     */
    public function setData($key, $value = null): static
    {
        if ($key === (array)$key) {
            $this->_data = $key;
        } else {
            $this->_data[$key] = $value;
        }
        return $this;
    }
```

```
        /**
         * Unset data from the object
         *
         * If $key is empty all attributes will be removed
         * If $key isn't empty the specific attribute will be removed
         *
         * @param $key
         * @return $this
         */
        public function unsetData($key = null): static
        {
            if ($key === null) {
                $this->setData([]);
            } elseif (is_string($key)) {
                if (isset($this->_data[$key]) || array_key_exists($key, $this-
>_data)) {
                    unset($this->_data[$key]);
                }
            } elseif ($key === (array)$key) {
                foreach ($key as $element) {
                    $this->unsetData($element);
                }
            }
            return $this;
        }
        /**
         * Get object data
         *
         * If $key isn't defined retrieve all attributes
         * If $key defined retrieve a specific attribute
         *
         * @param string $key
         * @return array|mixed|string|null
         */
        public function getData(string $key = ''): mixed
        {
            if ('' === $key) {
                return $this->_data;
            }
            if (isset($this->_data[$key])) {
                return $this->_data[$key];
            }
            return null;
        }
        /**
         * Check attributes availability
         *
         * If $key is empty or has non-string type the whole data is checked
         * If $key isn't empty the specific attribute is checked
         *
         * @param string $key
         * @return bool
         */
        public function hasData(string $key = ''): bool
        {
            if (empty($key) || !is_string($key)) {
                return !empty($this->_data);
            }
            return array_key_exists($key, $this->_data);
        }
        /**
         * Set/Get attribute wrapper
         *
```

```
            * @param string $method
            * @param array $args
            * @return mixed
            * @throws \Exception
            */
           public function __call(string $method, array $args)
           {
               switch (substr($method, 0, 3)) {
                   case 'get':
                       $key = $this->_underscore(substr($method, 3));
                       return $this->getData($key);
                   case 'set':
                       $key = $this->_underscore(substr($method, 3));
                       $value = $args[0] ?? null;
                       return $this->setData($key, $value);
                   case 'uns':
                       $key = $this->_underscore(substr($method, 3));
                       return $this->unsetData($key);
                   case 'has':
                       $key = $this->_underscore(substr($method, 3));
                       return isset($this->_data[$key]);
               }
               throw new \Exception('Invalid method %1::%2(%3)', [get_class($this),
$method, print_r($args, 1)]);
           }
           /**
            * Converts field names for setters and getters
            *
            * @param string $name
            * @return string
            */
           protected function _underscore(string $name): string
           {
               return  strtolower(trim(preg_replace('/([A-Z]|[0-9]+)/', "_$1",
$name), '_'));
           }
       }
```

Клас FileManager.php:

```
       <?php
       namespace Core;

       class FileManager
       {
           /**
            * @param string $path
            * @return false|string
            * @throws \Exception
            */
           public function read(string $path): bool|string
           {
               if ($this->isFileExist($path)) {
                   if (!$this->isFileReadable($path)) {
                       if (!$this->setReadWritePermissions($path)) {
                           throw new \Exception("Read permissions can't be changed
for requested file $path");
                       }
                   }
```

```
                $result = file_get_contents($path);
            } else {
                throw new \Exception("Requested file $path doesn't exist");
            }
            return $result;
        }
        /**
         * @param string $path
         * @param $data
         * @param int $flag
         * @return false|int
         * @throws \Exception
         */
        public function write(string $path, $data, int $flag = FILE_APPEND):
bool|int
        {
            if ($this->isFileExist($path)) {
                if (!$this->isFileWritable($path)) {
                    if (!$this->setReadWritePermissions($path)) {
                        throw new \Exception("Write permissions can't be changed
for requested file $path");
                    }
                }
                $result = file_put_contents($path, $data, $flag);
            } else {
                throw new \Exception("Requested file $path doesn't exist");
            }
            return $result;
        }
        /**
         * @param $dirName
         * @return bool
         */
        public function isDirectoryExist($dirName)
        {
            return is_dir($dirName);
        }
        /**
         * @param $dirName
         * @return array
         */
        public function readFilesInDirectory($dirName)
        {
            $files = [];
            if ($handle = opendir($dirName)) {
                while (false != ($file = readdir($handle))) {
                    if ($file != '.' && $file != '..') {
                        $files[] = $file;
                    }
                }
            }
            return $files;
        }
        /**
         * @param string $path
         * @return bool
         */
        public function isFileExist(string $path): bool
        {
            return file_exists($path);
        }
        /**
         * @param string $path
```

```
         * @return bool
         */
        public function isFileReadable(string $path): bool
        {
            return is_readable($path);
        }
        /**
         * @param string $path
         * @return bool
         */
        public function isFileWritable(string $path): bool
        {
            return is_writable($path);
        }
        public function setReadWritePermissions(string $path): bool
        {
            return chmod($path, 0600);
        }
    }
```

Клас Logger.php:

```
    <?php
    declare(strict_types=1);
    namespace Core;
    use DateTime;
    class Logger
    {
        /**
         * Path to log file
         */
        public const LOG_FILE_PATH = 'log/exception.log';
        /**
         * @var FileManager
         */
        private $fileManager;
        public function __construct()
        {
            $this->fileManager = new FileManager();
        }
        /**
         * Adding records to the log file
         *
         * @param string $message
         * @param array $errorTrace
         */
        public function log(string $message, array $errorTrace = [])
        {
            $dateTime = new DateTime("now");
            $dateTime->setTimestamp(time());
            $dateTime = $dateTime->format('d.m.Y, H:i:s');
            $encodedErrorTrace = json_encode($errorTrace);
            try {
                $this->fileManager->write(self::LOG_FILE_PATH, "[$dateTime]
$message: $encodedErrorTrace" . PHP_EOL, FILE_APPEND);
            } catch (\Exception $error) {
                http_response_code(500);
                exit(json_encode(['success' => 0, 'message' => $error-
>getMessage()]));
```

```
            }
        }
    }
```

Інтерфейс ModelInterface.php:

```
    <?php
    declare(strict_types=1);
    namespace Core;
    use DateTime;
    class Logger
    {
        /**
         * Path to log file
         */
        public const LOG_FILE_PATH = 'log/exception.log';
        /**
         * @var FileManager
         */
        private $fileManager;
        public function __construct()
        {
            $this->fileManager = new FileManager();
        }
        /**
         * Adding records to the log file
         *
         * @param string $message
         * @param array $errorTrace
         */
        public function log(string $message, array $errorTrace = [])
        {
            $dateTime = new DateTime("now");
            $dateTime->setTimestamp(time());
            $dateTime = $dateTime->format('d.m.Y, H:i:s');
            $encodedErrorTrace = json_encode($errorTrace);
            try {
                $this->fileManager->write(self::LOG_FILE_PATH, "[$dateTime]
$message: $encodedErrorTrace" . PHP_EOL, FILE_APPEND);
            } catch (\Exception $error) {
                http_response_code(500);
                exit(json_encode(['success' => 0, 'message' => $error-
>getMessage()]));
            }
        }
    }
```

Клас City.php:

```
    <?php
    namespace Model;
    use Core\AbstractModel;
    class City extends AbstractModel
    {
        /**
         * Constructor for Country model
```

```
     */
    public function __construct()
    {
        $this->_init('city', 'entity_id');
        parent::__construct();
    }
}
```

Клас CityRepository.php:

```
<?php
namespace Model;
class CityRepository
{
    private $cityModel;
    public function __construct()
    {
        $this->cityModel = new City();
    }
    /**
     * @return bool|array
     */
    public function getList(): bool|array
    {
        return $this->cityModel->load();
    }
}
```

Клас Country.php:

```
<?php
namespace Model;
use Core\AbstractModel;
class Country extends AbstractModel
{
    /**
     * Constructor for Country model
     */
    public function __construct()
    {
        $this->_init('country', 'entity_id');
        parent::__construct();
    }
}
```

Клас CountryRepository.php:

```
<?php
namespace Model;
class CountryRepository
{
    private $countryModel;
```

```
        public function __construct()
        {
            $this->countryModel = new Country();
        }
        /**
         * @return bool|array
         */
        public function getList(): bool|array
        {
            return $this->countryModel->load();
        }
    }
```

Клас Place.php:

```
    <?php
    namespace Model;
    use Core\AbstractModel;
    class Place extends AbstractModel
    {
        /**
         * Constructor for Place model
         */
        public function __construct()
        {
            $this->_init('place', 'entity_id');
            parent::__construct();
        }
        /**
         * @param int $cityId
         * @return array|bool
         */
        public function getHotelsByCityId(int $cityId)
        {
            $placeTypeId = $this->getPlaceTypeIdByName('hotel');
            return $this->load(0, "SELECT * FROM $this->tableName WHERE city_id
= $cityId and place_type_id = $placeTypeId");
        }
        /**
         * @param int $cityId
         * @return array|bool
         */
        public function getRestaurantByCityId(int $cityId)
        {
            $placeTypeId = $this->getPlaceTypeIdByName('restaurant');
            return $this->load(0, "SELECT * FROM $this->tableName WHERE city_id
= $cityId and place_type_id = $placeTypeId");
        }
        /**
         * @param string $placeTypeName
         * @return int
         */
        private function getPlaceTypeIdByName(string $placeTypeName)
        {
            $placeType = new PlaceType();
            $placeTypes = $placeType->getList();
            if (!empty($placeTypes) && is_array($placeTypes)) {
                foreach ($placeTypes as $placeType) {
```

```
                    if (!empty($placeType['name']) && $placeType['name'] ===
$placeTypeName) {
                        return  $placeType['entity_id'] ?? 0;
                    }
                }
            }
            return 0;
        }
    }
```

Клас PlaceRepository.php:

```
<?php
namespace Model;
use Core\FileManager;
class PlaceRepository
{
    private $placeModel;
    public function __construct()
    {
        $this->placeModel = new Place();
    }
    /**
     * @param int $cityId
     * @return bool|array
     */
    public function getHotelByCityId(int $cityId): bool|array
    {
        return $this->placeModel->getHotelsByCityId($cityId);
    }
    /**
     * @param int $cityId
     * @return bool|array
     */
    public function getRestaurantByCityId(int $cityId): bool|array
    {
        return $this->placeModel->getRestaurantByCityId($cityId);
    }
}
```

Клас PlaceType.php:

```
<?php
namespace Model;
use Core\AbstractModel;
class PlaceType extends AbstractModel
{
    /**
     * Constructor for Place model
     */
    public function __construct()
    {
        $this->_init('place_type', 'entity_id');
        parent::__construct();
    }
    /**
```

```
         * @return array|bool
         */
        public function getList()
        {
            return $this->load();
        }
    }
```

Клас PlaceTypeRepository.php:

```
    <?php
    namespace Model;
    class PlaceTypeRepository
    {
        private $placeTypeModel;
        public function __construct()
        {
            $this->placeTypeModel = new PlaceType();
        }
        /**
         * @return array|bool
         */
        public function getList()
        {
            return $this->placeTypeModel->getList();
        }
    }
```

Клас User.php:

```
    <?php
    namespace Model;
    use Core\AbstractModel;
    class User extends AbstractModel
    {
        /**
         * Constructor for User model
         */
        public function __construct()
        {
            $this->_init('user', 'entity_id');
            parent::__construct();
        }
        /**
         * @return $this
         */
        protected function beforeSave(): static
        {
            parent::beforeSave();
            if(!($email = $this->getEmail())) {
                http_response_code(400);
                exit(json_encode(['success' => 0, 'message' => "You should enter
the email"]));
            }
            $connection = $this->getConnection();
```

```
            $bind = ['email' => $email];
            $sql = "SELECT * FROM $this->tableName WHERE  email = :email";
            $statement = $connection->prepare($sql);
            $statement->execute($bind);
            $result = $statement->fetchColumn();
            if ($result) {
                http_response_code(400);
                exit(json_encode(['success' => 0, 'message' => 'Customer with
the same email address already exists']));
            }
            return $this;
        }
        /**
         * @param $email
         * @return mixed
         */
        public function getByEmail($email)
        {
            $connection = $this->getConnection();

            $bind = ['email' => $email];

            $sql = "SELECT * FROM $this->tableName WHERE  email = :email";

            $connection->beginTransaction();

            $statement = $connection->prepare($sql);

            $statement->execute($bind);

            $result = $statement->fetch();

            $connection->commit();

            if (empty($result)) {
                http_response_code(400);
                exit(json_encode(['success' => 0, 'message' => "Customer with
such email doesn't exist"]));
            }
            return $result;
        }
        /**
         * @return $this|User|void
         */
        protected function beforeUpdate()
        {
            parent::beforeUpdate();
            if(!$this->getEmail()) {
                http_response_code(400);
                exit(json_encode(['success' => 0, 'message' => "You should enter
the email"]));
            }
            $connection = $this->getConnection();
            $bind = ['email' => $this->getEmail()];
            $bind['entity_id'] = $this->getData($this->idFieldName);
            $sql = "SELECT * FROM $this->tableName WHERE  email = :email AND
$this->idFieldName != :entity_id";
            $statement = $connection->prepare($sql);
            $statement->execute($bind);
            $result = $statement->fetchColumn();
            if ($result) {
                http_response_code(400);
```

```
                exit(json_encode(['success' => 0, 'message' => 'Customer with
the same email address already exists']));
            }
            return $this;
        }
    }
```

Клас UserRepository.php:

```
<?php
declare(strict_types=1);
namespace Model;
class UserRepository
{
    /**
     * @var User
     */
    private $userModel;
    public function __construct()
    {
        $this->userModel = new User();
    }
    /**
     * @param int $id
     * @return array|bool
     */
    public function getById(int $id): bool|array
    {
        return $this->userModel->load($id);
    }
    /**
     * @param array $data
     * @return array
     */
    public function save(array $data)
    {
        $this->userModel->setData($data);
        return $this->userModel->save();
    }
    /**
     * @param array $data
     * @param $id
     * @return array
     */
    public function update(array $data, $id)
    {
        $this->userModel->setData($data);
        return $this->userModel->update($id);
    }
    /**
     * @param int $id
     * @return bool
     */
    public function delete(int $id)
    {
        return $this->userModel->delete($id);
    }
    /**
     * @return bool|array
```

```
         */
        public function getList(): bool|array
        {
            return $this->userModel->load();
        }
        public function getByEmail($email)
        {
            return $this->userModel->getByEmail($email);
        }
    }
```

Б.2 Програний код клієнтської частини додатку

Інтерфейс RestApi.java:

```
    package com.university.easytrip.api;
    import com.university.easytrip.model.BaseResponse;
    import com.university.easytrip.model.City;
    import com.university.easytrip.model.Country;
    import com.university.easytrip.model.Place;
    import com.university.easytrip.model.PlaceType;
    import com.university.easytrip.model.User;
    import java.util.ArrayList;
    import java.util.List;
    import io.reactivex.rxjava3.core.Observable;
    import retrofit2.http.Body;
    import retrofit2.http.DELETE;
    import retrofit2.http.GET;
    import retrofit2.http.POST;
    import retrofit2.http.PUT;
    import retrofit2.http.Path;
    import retrofit2.http.Query;
    public interface RestApi {
        // User's endpoints
        @GET("user/{id}")
        Observable<BaseResponse<User>> getUser(@Path("id") int userId);
        @GET("user/by-email")
        Observable<BaseResponse<User>> getUserByEmail(@Query("value") String
email);
        @POST("user")
        Observable<BaseResponse<User>> saveUser(@Body User user);
        @PUT("user/{id}")
        Observable<BaseResponse<User>> updateUser(@Path("id") int userId, @Body
User user);
        @DELETE("user/{id}")
        Observable<BaseResponse<String>> deleteUser(@Path("id") int userId);
        // Country's endpoints
        @GET("country")
        Observable<BaseResponse<ArrayList<Country>>> getCountryList();
        // City's endpoints
        @GET("city")
        Observable<BaseResponse<ArrayList<City>>> getCityList();
        // PlaceType's endpoints
        @GET("placetype")
        Observable<BaseResponse<PlaceType>> getPlaceTypeList();
        // Place's endpoints
        @GET("place/hotel/{id}")
```

```
        Observable<BaseResponse<List<Place>>> getHotelsByCityId(@Path("id") int
cityId);
        @GET("place/restaurant/{id}")
        Observable<BaseResponse<List<Place>>> getRestaurantsByCityId(@Path("id")
int cityId);
    }
```

Клас RestClient.java:

```
    package com.university.easytrip.api;
    import androidx.annotation.NonNull;
    import java.io.IOException;
    import okhttp3.Credentials;
    import okhttp3.HttpUrl;
    import okhttp3.Interceptor;
    import okhttp3.OkHttpClient;
    import okhttp3.Request;
    import okhttp3.Response;
    import okhttp3.logging.HttpLoggingInterceptor;
    import retrofit2.Retrofit;
    import retrofit2.adapter.rxjava3.RxJava3CallAdapterFactory;
    import retrofit2.converter.gson.GsonConverterFactory;
    public class RestClient {
        private static Retrofit retrofitClient;
        public static String credentials = Credentials.basic("test_api", "123");
        public static final String baseUrl = "http://10.0.2.2/";
        private RestClient() { }
        public static Retrofit createRetrofitClient() {
            if (retrofitClient == null) {
                retrofitClient = new Retrofit.Builder()
                        .baseUrl(baseUrl)
                        .addConverterFactory(GsonConverterFactory.create())

.addCallAdapterFactory(RxJava3CallAdapterFactory.create())
                        .client(RestClient.createOkHttpClient())
                        .build();
            }
            return retrofitClient;
        }

        private static OkHttpClient createOkHttpClient() {
            OkHttpClient.Builder httpClient = new OkHttpClient.Builder();
            httpClient.addInterceptor(new Interceptor() {
                @NonNull
                @Override
                public Response intercept(@NonNull Chain chain) throws
IOException {
                    Request request = chain.request().newBuilder()
                            .addHeader("Authorization", credentials)
                            .build();
                    return chain.proceed(request);
                }
            });
            HttpLoggingInterceptor logging = new HttpLoggingInterceptor();
            logging.level(HttpLoggingInterceptor.Level.BODY);
            httpClient.addInterceptor(logging);
            return httpClient.build();
        }
    }
```

Клас ProgressBar.java:

```
package com.university.easytrip.helper;
import android.app.Activity;
import android.view.LayoutInflater;
import androidx.appcompat.app.AlertDialog;
import com.university.easytrip.R;
public class ProgressBar {
    private Activity activity;
    private AlertDialog alertDialog;
    public ProgressBar(Activity activity) {
        this.activity = activity;
    }
    public void startProgressBar() {
        AlertDialog.Builder builder = new
AlertDialog.Builder(this.activity);
        LayoutInflater layoutInflater = this.activity.getLayoutInflater();
        builder.setView(layoutInflater.inflate(R.layout.progress_bar,
null));
        builder.setCancelable(false);
        alertDialog = builder.create();
        alertDialog.show();
    }
    public void stopProgressBar() {
        alertDialog.dismiss();
    }
}
```

Клас UserLocalData.java:

```
package com.university.easytrip.helper;
import android.content.Context;
import android.content.SharedPreferences;
import com.university.easytrip.model.User;
public class UserLocalData {
    public static SharedPreferences dataStore;
    public static final String USER_DATA_SOURCE = "user_data";
    public static final String USER_ID = "id";
    public static final String USER_FIRST_NAME = "firstName";
    public static final String USER_LAST_NAME = "lastName";
    public static final String USER_EMAIL = "email";
    public static final String USER_PASSWORD = "password";
    public static final String USER_LOGGED_IN = "loggedIn";
    private UserLocalData() {}
    public static void setUserDataContext(Context context) {
        if (dataStore == null) {
            dataStore = context.getSharedPreferences(USER_DATA_SOURCE, 0);
        }
    }
    public static void saveUserData(User user) {
        SharedPreferences.Editor editor = dataStore.edit();
        editor.putInt(USER_ID, user.getEntityId());
        editor.putString(USER_FIRST_NAME, user.getFirstName());
        editor.putString(USER_LAST_NAME, user.getLastName());
        editor.putString(USER_EMAIL, user.getEmail());
        editor.putString(USER_PASSWORD, user.getPassword());
        editor.putBoolean(USER_LOGGED_IN, false);
```

```
        editor.apply();
    }
    public static User getLoggedUserData() {
        int id = dataStore.getInt(USER_ID, 0);
        String firstName = dataStore.getString(USER_FIRST_NAME, "");
        String lastName = dataStore.getString(USER_LAST_NAME, "");
        String email = dataStore.getString(USER_EMAIL, "");
        String password = dataStore.getString(USER_PASSWORD, "");
        return new User(id, firstName, lastName, email, password);
    }
    public static void setUserLoggedIn(boolean loggedIn) {
        SharedPreferences.Editor editor = dataStore.edit();
        editor.putBoolean(USER_LOGGED_IN, loggedIn);
        editor.apply();
    }
    public static boolean isUserLoggedIn() {
        return dataStore.getBoolean(USER_LOGGED_IN, false);
    }
    public static void resetUserData() {
        SharedPreferences.Editor editor = dataStore.edit();
        editor.clear();
        editor.apply();
    }
}
```

Клас BaseResponse.java:

```
package com.university.easytrip.model;
import com.google.gson.annotations.SerializedName;
public class BaseResponse <T> {
    @SerializedName("success")
    private int success;
    @SerializedName("response")
    private T response;
    @SerializedName("message")
    private String message;
    public int getSuccess() {
        return success;
    }
    public String getMessage() {
        return message;
    }
    public T getResponse() {
        return response;
    }
}
```

Клас City.java:

```
package com.university.easytrip.model;
import androidx.annotation.NonNull;
import com.google.gson.annotations.SerializedName;
import java.util.Comparator;
public class City{
    @SerializedName("name")
    private String name;
```

```
    @SerializedName("entity_id")
    private int cityId;
    @SerializedName("country_id")
    private int countryId;
    public City(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
    public int getCityId() {
        return this.cityId;
    }
    public int getCountryId() {
        return this.countryId;
    }
    @NonNull
    @Override
    public String toString() {
        return this.name;
    }
}
```

Клас Country.java:

```
package com.university.easytrip.model;
import androidx.annotation.NonNull;
import com.google.gson.annotations.SerializedName;
public class Country {
    @SerializedName("name")
    private String name;
    @SerializedName("entity_id")
    private int countryId;
    public Country(int countryId, String name) {
        this.countryId = countryId;
        this.name = name;
    }
    @NonNull
    @Override
    public String toString() {
        return this.name;
    }
    public int getCountryId() {
        return this.countryId;
    }
    public String getName() {
        return name;
    }
}
```

Клас Place.java:

```
package com.university.easytrip.model;
import androidx.annotation.NonNull;
import com.google.gson.annotations.SerializedName;
public class Country {
```

```
    @SerializedName("name")
    private String name;
    @SerializedName("entity_id")
    private int countryId;
    public Country(int countryId, String name) {
        this.countryId = countryId;
        this.name = name;
    }
    @NonNull
    @Override
    public String toString() {
        return this.name;
    }
    public int getCountryId() {
        return this.countryId;
    }
    public String getName() {
        return name;
    }
}
```

Клас PlaceType.java:

```
package com.university.easytrip.model;
import com.google.gson.annotations.SerializedName;
public class PlaceType {
    @SerializedName("name")
    private String name;
    @SerializedName("entity_id")
    private int placeTypeId;
    public PlaceType(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
    public int getPlaceTypeId() {
        return placeTypeId;
    }
}
```

Клас User.java:

```
package com.university.easytrip.model;
import com.google.gson.annotations.SerializedName;
public class User extends BaseResponse {
    @SerializedName("entity_id")
    private int id;
    @SerializedName("first_name")
    private String firstName;
    @SerializedName("last_name")
    private String lastName;
    @SerializedName("email")
    private String email;
    @SerializedName("password")
    private String password;
```

```
        public User(int id, String firstName, String lastName, String email,
String password) {
            this.id = id;
            this.firstName = firstName;
            this.lastName = lastName;
            this.email = email;
            this.password = password;
        }
        public int getEntityId() {
            return id;
        }
        public void setEntityId(int id) {
            this.id = id;
        }
        public String getFirstName() {
            return firstName;
        }
        public void setFirstName(String firstName) {
            this.firstName = firstName;
        }
        public String getLastName() {
            return lastName;
        }
        public void setLastName(String lastName) {
            this.lastName = lastName;
        }
        public String getEmail() {
            return email;
        }
        public void setEmail(String email) {
            this.email = email;
        }
        public String getPassword() { return password; }
        public void setPassword(String password) { this.password = password; }
    }
```

Клас MainActivity.java:

```
package com.university.easytrip.activity;
import androidx.appcompat.app.AppCompatActivity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

import com.university.easytrip.R;
import com.university.easytrip.databinding.ActivityMainBinding;
public class MainActivity extends NavigationDrawerActivity {
    private ActivityMainBinding binding;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        binding = ActivityMainBinding.inflate(getLayoutInflater());
        setContentView(binding.getRoot());
    }
}
```

Клас LoginActivity.java:

```
package com.university.easytrip.activity;
import android.content.Intent;
import android.os.Bundle;
import android.text.Editable;
import android.text.TextWatcher;
import android.util.Log;
import android.view.View;
import android.widget.Toast;
import com.university.easytrip.databinding.ActivityLoginBinding;
import com.google.gson.Gson;
import com.google.gson.TypeAdapter;
import com.university.easytrip.api.RestApi;
import com.university.easytrip.api.RestClient;
import com.university.easytrip.helper.ProgressBar;
import com.university.easytrip.helper.UserLocalData;
import com.university.easytrip.model.BaseResponse;
import com.university.easytrip.model.User;
import java.io.IOException;
import io.reactivex.rxjava3.android.schedulers.AndroidSchedulers;
import io.reactivex.rxjava3.annotations.NonNull;
import io.reactivex.rxjava3.core.Observer;
import io.reactivex.rxjava3.disposables.CompositeDisposable;
import io.reactivex.rxjava3.disposables.Disposable;
import io.reactivex.rxjava3.schedulers.Schedulers;
import okhttp3.ResponseBody;
import retrofit2.HttpException;
public class LoginActivity extends NavigationDrawerActivity {
    private RestApi restApi;
    private ProgressBar progressBar;
    private Intent redirectToAccount, redirectToRegistration;
    private CompositeDisposable disposable;
    private ActivityLoginBinding binding;
    private String currentEmail, currentPassword;
    private boolean isValidPasswd, isValidEmail;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        binding = ActivityLoginBinding.inflate(getLayoutInflater());
        setCurrentActivityTitle("Login");
        setContentView(binding.getRoot());
        disposable = new CompositeDisposable();
        // Initialize progress bar
        progressBar = new ProgressBar(this);
        UserLocalData.setUserDataContext(this);
        // Creating rest api client
        restApi = RestClient.createRetrofitClient().create(RestApi.class);
        redirectToAccount = new Intent(this, ProfileActivity.class);
        redirectToRegistration= new Intent(this, RegisterActivity.class);
        // On register click event
        binding.registerNowBtn.setOnClickListener(new View.OnClickListener()
{
            @Override
            public void onClick(View view) {
                startActivity(redirectToRegistration);
            }
        });
        // On login click event
        binding.loginBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                progressBar.startProgressBar();
                if (!isValidEmail || !isValidPasswd ||
currentEmail.isEmpty() || currentPassword.isEmpty()) {
```

```
                        progressBar.stopProgressBar();
                        return;
                    }
                    restApi.getUserByEmail(currentEmail)
                            .subscribeOn(Schedulers.io())
                            .observeOn(AndroidSchedulers.mainThread())
                            .subscribe(new  Observer<BaseResponse<User>>() {
                                @Override
                                public void onSubscribe(@NonNull Disposable d) {
                                    disposable.add(d);
                                }
                                @Override
                                public void onNext(@NonNull BaseResponse<User>
user) {
                                    if (user.getSuccess() == 0) {
                                        String errorMessage = user.getMessage()
!= null ? user.getMessage() : "";
                                        Toast.makeText(LoginActivity.this,
"Login to account has failed. " + errorMessage, Toast.LENGTH_SHORT).show();
                                        Log.d("Error", "Error on login to an
account: " + errorMessage);
                                        return;
                                    } else if(user.getSuccess() == 1) {
                                        if (user.getResponse() != null) {
                                            User savedUser = user.getResponse();
                                            if
(savedUser.getPassword().equals(currentPassword)) {

UserLocalData.saveUserData(savedUser);

UserLocalData.setUserLoggedIn(true);

Toast.makeText(LoginActivity.this, "Login has been completed!",
Toast.LENGTH_SHORT).show();
                                                Log.d("Success", "Login has been
completed");

startActivity(redirectToAccount);
                                                return;
                                            } else {

Toast.makeText(LoginActivity.this, "You have entered incorrect password!",
Toast.LENGTH_SHORT).show();
                                            }
                                        }
                                    }
                                    Toast.makeText(LoginActivity.this, "The
internal error has happened. Try again later", Toast.LENGTH_SHORT).show();
                                    Log.d("Error", "Internal server error on
login to account");
                                }
                                @Override
                                public void onError(@NonNull Throwable e) {
                                    try {
                                        HttpException exception =
((HttpException) e);
                                        if (exception.code() == 400) {
                                            ResponseBody responseBody =
exception.response().errorBody();
                                            Gson gson = new Gson();
                                            TypeAdapter<BaseResponse> adapter =
gson.getAdapter(BaseResponse.class);
                                            try {
```

```
                                                    BaseResponse errorParser =
adapter.fromJson(responseBody.string());
                                                    if (errorParser.getMessage() !=
null && !errorParser.getMessage().isEmpty()) {

Toast.makeText(LoginActivity.this, errorParser.getMessage(),
Toast.LENGTH_SHORT).show();

progressBar.stopProgressBar();
                                                        return;
                                                    } else {
                                                        throw new Exception("The
internal error has happened. Try again later");
                                                    }
                                                } catch (IOException error) {

Toast.makeText(LoginActivity.this, "The internal error has happened. Try again
later", Toast.LENGTH_SHORT).show();
                                                    Log.d("Error", "Critical
internal error: " + error.getMessage());
                                                    progressBar.stopProgressBar();
                                                    return;
                                                }
                                            }
                                        } catch (Exception error) {
                                            progressBar.stopProgressBar();
                                            Toast.makeText(LoginActivity.this, "The
internal error has happened. Try again later", Toast.LENGTH_SHORT).show();
                                            Log.d("Error", "Internal server error on
login to account");
                                        }
                                    }
                                    @Override
                                    public void onComplete() {
                                        progressBar.stopProgressBar();
                                    }
                                });
                    }
                });
                binding.emailInput.addTextChangedListener(new TextWatcher() {
                    @Override
                    public void beforeTextChanged(CharSequence charSequence, int i,
int i1, int i2) { }
                    @Override
                    public void onTextChanged(CharSequence charSequence, int i, int
i1, int i2) {
                        isValidEmail = true;
                        String inputValue = charSequence.toString();
                        if (inputValue.isEmpty()) {
                            binding.emailInputLayout.setError("Field can't be
empty");
                            isValidEmail = false;
                        } else if (inputValue.contains(" ")) {
                            binding.emailInputLayout.setError("Field can't contain
whitespaces");
                            isValidEmail = false;
                        } else if (!inputValue.matches("[a-zA-Z0-9._-]+@[a-
z]+\\.+[a-z]+")) {
                            binding.emailInputLayout.setError("Invalid format");
                            isValidEmail = false;
                        }
                        if (isValidEmail) {
                            currentEmail = inputValue;
```

```
                        binding.emailInputLayout.setHelperText(null);
                        binding.emailInputLayout.setError(null);
                    }
                }
                @Override
                public void afterTextChanged(Editable editable) { }
            });
            binding.passwordInput.addTextChangedListener(new TextWatcher() {
                @Override
                public void beforeTextChanged(CharSequence charSequence, int i,
int i1, int i2) { }
                @Override
                public void onTextChanged(CharSequence charSequence, int i, int
i1, int i2) {
                    isValidPasswd = true;
                    String inputValue = charSequence.toString();
                    if (inputValue.isEmpty()) {
                        binding.passwordInputLayout.setError("Field can't be
empty");
                        isValidPasswd = false;
                    } else if (inputValue.contains(" ")) {
                        binding.passwordInputLayout.setError("Field can't
contain whitespaces");
                        isValidPasswd = false;
                    } else if (!inputValue.matches("^(?=.*\\d).+$")) {
                        binding.passwordInputLayout.setError("Password should
contain at least one number");
                        isValidPasswd = false;
                    } else if (!inputValue.matches("^(?=.*[A-Z]).+$")) {
                        binding.passwordInputLayout.setError("Password should
contain at least one upper case character");
                        isValidPasswd = false;
                    } else if (!inputValue.matches("^(?=.*[a-z]).+$")) {
                        binding.passwordInputLayout.setError("Password should
contain at least one lower case character");
                        isValidPasswd = false;
                    }
                    if (isValidPasswd) {
                        currentPassword = inputValue;
                        binding.passwordInputLayout.setHelperText(null);
                        binding.passwordInputLayout.setError(null);
                    }
                }
                @Override
                public void afterTextChanged(Editable editable) {
                }
            });
        }

        @Override
        protected void onDestroy() {
            disposable.dispose();
            super.onDestroy();
        }
    }
```

Клас ChangePasswordActivity.java:

```
package com.university.easytrip.activity;
```

```
import androidx.appcompat.app.AppCompatActivity;
import android.content.Intent;
import android.os.Bundle;
import android.text.Editable;
import android.text.TextWatcher;
import android.util.Log;
import android.view.View;
import android.widget.Toast;
import android.widget.Toolbar;
import com.google.android.material.appbar.MaterialToolbar;
import com.google.gson.Gson;
import com.google.gson.TypeAdapter;
import com.university.easytrip.api.RestApi;
import com.university.easytrip.api.RestClient;
import com.university.easytrip.databinding.ActivityChangePasswordBinding;
import com.university.easytrip.helper.ProgressBar;
import com.university.easytrip.helper.UserLocalData;
import com.university.easytrip.model.BaseResponse;
import com.university.easytrip.model.User;
import java.io.IOException;
import io.reactivex.rxjava3.android.schedulers.AndroidSchedulers;
import io.reactivex.rxjava3.annotations.NonNull;
import io.reactivex.rxjava3.core.Observer;
import io.reactivex.rxjava3.disposables.CompositeDisposable;
import io.reactivex.rxjava3.disposables.Disposable;
import io.reactivex.rxjava3.schedulers.Schedulers;
import okhttp3.ResponseBody;
import retrofit2.HttpException;
public class ChangePasswordActivity extends AppCompatActivity {
    private ActivityChangePasswordBinding binding;
    private ProgressBar progressBar;
    private RestApi restApi;
    private String oldPassword;
    private CompositeDisposable disposable = new CompositeDisposable();
    private boolean isValidPasswordOld, isValidPasswordNew,
isValidPasswdRepeat;
    private Intent redirectToAccount, redirectToLogin;
    private User user;
    private UserLocalData userLocalData;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        binding =
ActivityChangePasswordBinding.inflate(getLayoutInflater());
        setContentView(binding.getRoot());
        // Add back button to the up bar
        MaterialToolbar toolbar = (MaterialToolbar) binding.topAppBar;
        setSupportActionBar(toolbar);
        getSupportActionBar().setDisplayHomeAsUpEnabled(true);
        // Initialize progress bar
        progressBar = new ProgressBar(this);
        // Creating rest api client
        restApi = RestClient.createRetrofitClient().create(RestApi.class);
        redirectToAccount = new Intent(this, ProfileActivity.class);
        redirectToLogin = new Intent(this, LoginActivity.class);
        UserLocalData.setUserDataContext(this);
        user = UserLocalData.getLoggedUserData();
        oldPassword = user.getPassword();
        if (oldPassword.isEmpty()) {
            UserLocalData.setUserLoggedIn(false);
            UserLocalData.resetUserData();
            Toast.makeText(ChangePasswordActivity.this, "Please, login
again", Toast.LENGTH_SHORT).show();
```

```
                startActivity(redirectToLogin);
            }
            binding.changePasswordBtn.setOnClickListener(new
View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    progressBar.startProgressBar();
                    if (!isValidPasswordOld || !isValidPasswordNew ||
!isValidPasswdRepeat) {
                        progressBar.stopProgressBar();
                        return;
                    }
                    int userId = 0;
                    if (user.getEntityId() == 0) {
                        UserLocalData.setUserLoggedIn(false);
                        UserLocalData.resetUserData();
                        Toast.makeText(ChangePasswordActivity.this, "Please,
login again", Toast.LENGTH_SHORT).show();
                        startActivity(redirectToLogin);
                    }
                    progressBar.startProgressBar();

user.setPassword(binding.newPasswordInput.getText().toString());
                    UserLocalData.saveUserData(user);
                    userId = user.getEntityId();
                    restApi.updateUser(userId, user)
                            .subscribeOn(Schedulers.io())
                            .observeOn(AndroidSchedulers.mainThread())
                            .subscribe(new  Observer<BaseResponse<User>>() {
                                @Override
                                public void onSubscribe(@NonNull Disposable d) {
                                    disposable.add(d);
                                }
                                @Override
                                public void onNext(@NonNull BaseResponse<User>
user) {
                                    if (user.getSuccess() == 0) {
                                        String errorMessage = user.getMessage()
!= null ? user.getMessage() : "";

Toast.makeText(ChangePasswordActivity.this, "Changing password operation has
failed. Try again later", Toast.LENGTH_SHORT).show();
                                        Log.d("Error", "Error on deleting
account: " + errorMessage);
                                        return;
                                    } else if(user.getSuccess() == 1) {
                                        if (user.getResponse() != null) {

UserLocalData.setUserLoggedIn(false);
                                            UserLocalData.resetUserData();

Toast.makeText(ChangePasswordActivity.this, "Password has been changed",
Toast.LENGTH_SHORT).show();
                                            Log.d("Success", "Password has been
changed");
                                            startActivity(redirectToAccount);
                                            return;
                                        }
                                    }
                                    Toast.makeText(ChangePasswordActivity.this,
"The internal error has happened. Try again later", Toast.LENGTH_SHORT).show();
                                    Log.d("Error", "Internal server error on
changing password");
```

```
                                    }
                                    @Override
                                    public void onError(@NonNull Throwable e) {
                                        try {
                                            HttpException exception =
((HttpException) e);
                                            if (exception.code() == 400) {
                                                ResponseBody responseBody =
exception.response().errorBody();
                                                Gson gson = new Gson();
                                                TypeAdapter<BaseResponse> adapter =
gson.getAdapter(BaseResponse.class);
                                                try {
                                                    BaseResponse errorParser =
adapter.fromJson(responseBody.string());
                                                    if (errorParser.getMessage() !=
null && !errorParser.getMessage().isEmpty()) {

Toast.makeText(ChangePasswordActivity.this, "Changing password operation has
failed. Try again later", Toast.LENGTH_SHORT).show();
                                                        Log.d("Error", "Internal
server error on changing password " + errorParser.getMessage());

progressBar.stopProgressBar();
                                                        return;
                                                    } else {
                                                        throw new Exception("The
internal error has happened. Try again later");
                                                    }
                                                } catch (IOException error) {

Toast.makeText(ChangePasswordActivity.this, "The internal error has happened. Try
again later", Toast.LENGTH_SHORT).show();
                                                    Log.d("Error", "Critical
internal error: " + error.getMessage());
                                                    progressBar.stopProgressBar();
                                                    return;
                                                }
                                            }
                                        } catch (Exception error) {
                                            progressBar.stopProgressBar();

Toast.makeText(ChangePasswordActivity.this, "The internal error has happened. Try
again later", Toast.LENGTH_SHORT).show();
                                            Log.d("Error", "Internal server error on
changing password");
                                        }
                                    }
                                    @Override
                                    public void onComplete() {
                                        progressBar.stopProgressBar();
                                    }
                                });
                    }
                });
        binding.oldPasswordInput.addTextChangedListener(new TextWatcher() {
                    @Override
                    public void beforeTextChanged(CharSequence charSequence, int i,
int i1, int i2) { }

                    @Override
                    public void onTextChanged(CharSequence charSequence, int i, int
i1, int i2) {
```

```
                    isValidPasswordOld = true;
                    String passwd = charSequence.toString();
                    if (passwd.isEmpty()) {
                        binding.oldPasswordInputLayout.setError("Field can't be
empty");
                        isValidPasswordOld = false;
                    } else if (passwd.contains(" ")) {
                        binding.oldPasswordInputLayout.setError("Field can't
contain whitespaces");
                        isValidPasswordOld = false;
                    } else if (!passwd.matches("^(?=.*\\d).+$")) {
                        binding.oldPasswordInputLayout.setError("Password should
contain at least one number");
                        isValidPasswordOld = false;
                    } else if (!passwd.matches("^(?=.*[A-Z]).+$")) {
                        binding.oldPasswordInputLayout.setError("Password should
contain at least one upper case character");
                        isValidPasswordOld = false;
                    } else if (!passwd.matches("^(?=.*[a-z]).+$")) {
                        binding.oldPasswordInputLayout.setError("Password should
contain at least one lower case character");
                        isValidPasswordOld = false;
                    } else if (!passwd.equals(oldPassword)) {
                        binding.oldPasswordInputLayout.setError("Your current
password doesn't match. Try to memorize it");
                        isValidPasswordOld = false;
                    }
                    if (isValidPasswordOld) {
                        binding.oldPasswordInputLayout.setHelperText(null);
                        binding.oldPasswordInputLayout.setError(null);
                    }
                }
                @Override
                public void afterTextChanged(Editable editable) { }
            });
            binding.newPasswordInput.addTextChangedListener(new TextWatcher() {
                @Override
                public void beforeTextChanged(CharSequence charSequence, int i,
int i1, int i2) { }
                @Override
                public void onTextChanged(CharSequence charSequence, int i, int
i1, int i2) {
                    isValidPasswordNew = true;
                    String passwd = charSequence.toString();
                    if (passwd.isEmpty()) {
                        binding.newPasswordInputLayout.setError("Field can't be
empty");
                        isValidPasswordNew = false;
                    } else if (passwd.contains(" ")) {
                        binding.newPasswordInputLayout.setError("Field can't
contain whitespaces");
                        isValidPasswordNew = false;
                    } else if (!passwd.matches("^(?=.*\\d).+$")) {
                        binding.newPasswordInputLayout.setError("Password should
contain at least one number");
                        isValidPasswordNew = false;
                    } else if (!passwd.matches("^(?=.*[A-Z]).+$")) {
                        binding.newPasswordInputLayout.setError("Password should
contain at least one upper case character");
                        isValidPasswordNew = false;
                    } else if (!passwd.matches("^(?=.*[a-z]).+$")) {
                        binding.newPasswordInputLayout.setError("Password should
contain at least one lower case character");
```

```
                        isValidPasswordNew = false;
                    }
                    if (isValidPasswordNew) {
                        binding.newPasswordInputLayout.setHelperText(null);
                        binding.newPasswordInputLayout.setError(null);
                    }
                }
                @Override
                public void afterTextChanged(Editable editable) { }
            });
            binding.repeatNewPasswordInput.addTextChangedListener(new
TextWatcher() {
                @Override
                public void beforeTextChanged(CharSequence charSequence, int i,
int i1, int i2) { }
                @Override
                public void onTextChanged(CharSequence charSequence, int i, int
i1, int i2) {
                    isValidPasswdRepeat = true;
                    String passwd = charSequence.toString();
                    if (charSequence.toString().isEmpty()) {
                        binding.repeatNewPasswordInputLayout.setError("Field
can't be empty");
                        isValidPasswdRepeat = false;
                    } else if (charSequence.toString().contains(" ")) {
                        binding.repeatNewPasswordInputLayout.setError("Field
can't contain whitespaces");
                        isValidPasswdRepeat = false;
                    } else if (!passwd.matches("^(?=.*\\d).+$")) {
                        binding.repeatNewPasswordInputLayout.setError("Password
should contain at least one number");
                        isValidPasswdRepeat = false;
                    } else if (!passwd.matches("^(?=.*[A-Z]).+$")) {
                        binding.repeatNewPasswordInputLayout.setError("Password
should contain at least one upper case character");
                        isValidPasswdRepeat = false;
                    } else if (!passwd.matches("^(?=.*[a-z]).+$")) {
                        binding.repeatNewPasswordInputLayout.setError("Password
should contain at least one lower case character");
                        isValidPasswdRepeat = false;
                    } else if
(!passwd.equals(binding.newPasswordInput.getText().toString())) {
                        binding.repeatNewPasswordInputLayout.setError("Passwords
should match");
                        isValidPasswdRepeat = false;
                    }
                    if (isValidPasswdRepeat) {

binding.repeatNewPasswordInputLayout.setHelperText(null);
                        binding.repeatNewPasswordInputLayout.setError(null);
                    }
                }
                @Override
                public void afterTextChanged(Editable editable) { }
            });
        }
        @Override
        protected void onDestroy() {
            disposable.dispose();
            super.onDestroy();
        }
    }
```

Клас NavigationDrawerActivity.java:

```
package com.university.easytrip.activity;
import androidx.annotation.NonNull;
import androidx.appcompat.app.ActionBarDrawerToggle;
import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.widget.Toolbar;
import androidx.core.view.GravityCompat;
import androidx.drawerlayout.widget.DrawerLayout;
import android.content.Context;
import android.content.Intent;
import android.net.ConnectivityManager;
import android.net.NetworkInfo;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.FrameLayout;
import android.widget.Toast;
import com.university.easytrip.R;
import com.google.android.material.appbar.MaterialToolbar;
import com.google.android.material.navigation.NavigationView;
import com.university.easytrip.helper.ProgressBar;
import com.university.easytrip.helper.UserLocalData;
public class NavigationDrawerActivity extends AppCompatActivity implements
NavigationView.OnNavigationItemSelectedListener {
          DrawerLayout drawerLayout;
          private NavigationView navigationView;
          private ProgressBar progressBar;
          MaterialToolbar toolbar;
          @Override
          public void setContentView(View view) {
              drawerLayout = (DrawerLayout)
getLayoutInflater().inflate(R.layout.activity_navigation_drawer, null);
              FrameLayout frameLayout =
drawerLayout.findViewById(R.id.activityContainer);
              frameLayout.addView(view);
              super.setContentView(drawerLayout);
              UserLocalData.setUserDataContext(this);
              Toolbar toolbar = drawerLayout.findViewById(R.id.toolBar);
              setSupportActionBar(toolbar);
              // Initialize progress bar
              progressBar = new ProgressBar(this);
              navigationView = drawerLayout.findViewById(R.id.nav_view);
              navigationView.setNavigationItemSelectedListener(this);
              navigationView.bringToFront();
              ActionBarDrawerToggle actionBarDrawerToggle = new
ActionBarDrawerToggle(
                      this,
                      drawerLayout,
                      toolbar,
                      R.string.drawer_opened_label,
                      R.string.drawer_closed_label
              );
              drawerLayout.addDrawerListener(actionBarDrawerToggle);
              actionBarDrawerToggle.syncState();
              boolean isUserLoggedIn = UserLocalData.isUserLoggedIn();
              // Hide/show navigation drawer menu items for users/guest
              Menu menu = navigationView.getMenu();
              MenuItem customerMenuItem =
menu.findItem(R.id.profile_user_group_title);
```

```
            MenuItem guestMenuItem =
menu.findItem(R.id.profile_guest_group_title);
            if (isUserLoggedIn) {
                customerMenuItem.setVisible(true);
                guestMenuItem.setVisible(false);
            } else {
                customerMenuItem.setVisible(false);
                guestMenuItem.setVisible(true);
            }
        }
        @Override
        public boolean onNavigationItemSelected(@NonNull MenuItem item) {
            progressBar.startProgressBar();
            drawerLayout.closeDrawer(GravityCompat.START);
            switch (item.getItemId()) {
                case R.id.home_menu:
                    startActivity(new Intent(this, MainActivity.class));
                    break;
                case R.id.restaurant_menu:
                    if (UserLocalData.isUserLoggedIn()) {
                        startActivity(new Intent(this,
PlaceListRestaurantActivity.class));
                        Toast.makeText(NavigationDrawerActivity.this, "Search
for restaurants", Toast.LENGTH_SHORT).show();
                    } else {
                        startActivity(new Intent(this, LoginActivity.class));
                        Toast.makeText(NavigationDrawerActivity.this, "You
should login first", Toast.LENGTH_SHORT).show();
                    }
                    break;
                case R.id.hotel_menu:
                    if (UserLocalData.isUserLoggedIn()) {
                        startActivity(new Intent(this,
PlaceListHotelActivity.class));
                        Toast.makeText(NavigationDrawerActivity.this, "Search
for hotels", Toast.LENGTH_SHORT).show();
                    } else {
                        startActivity(new Intent(this, LoginActivity.class));
                        Toast.makeText(NavigationDrawerActivity.this, "You
should login first", Toast.LENGTH_SHORT).show();
                    }
                    break;
                case R.id.profile_menu:
                    startActivity(new Intent(this, ProfileActivity.class));
                    Toast.makeText(NavigationDrawerActivity.this, "Profile
page", Toast.LENGTH_SHORT).show();
                    break;
                case R.id.logout_menu:
                    UserLocalData.setUserLoggedIn(false);
                    UserLocalData.resetUserData();
                    startActivity(new Intent(this, LoginActivity.class));
                    Toast.makeText(NavigationDrawerActivity.this, "You have
logged out", Toast.LENGTH_SHORT).show();
                    break;
                case R.id.login_menu:
                    startActivity(new Intent(this, LoginActivity.class));
                    break;
                case R.id.register_menu:
                    startActivity(new Intent(this, RegisterActivity.class));
                    Toast.makeText(NavigationDrawerActivity.this, "Register
page", Toast.LENGTH_SHORT).show();
                    break;
            }
```

```
            progressBar.stopProgressBar();
            return true;
        }
        protected void setCurrentActivityTitle(String title) {
            if (getSupportActionBar() != null) {
                getSupportActionBar().setTitle(title);
            }
        }
        public boolean isNetworkConnected() {
            ConnectivityManager connectivityManager = (ConnectivityManager)
getSystemService(Context.CONNECTIVITY_SERVICE);
            NetworkInfo networkInfo =
connectivityManager.getActiveNetworkInfo();

            return networkInfo != null && networkInfo.isConnectedOrConnecting();
        }
    }
```

Клас PlaceHotelActivity.java:

```
    package com.university.easytrip.activity;
    import androidx.appcompat.app.AppCompatActivity;
    import android.os.Bundle;
    import android.util.DisplayMetrics;
    import android.view.View;
    import android.widget.ImageView;
    import android.widget.TextView;
    import com.bumptech.glide.Glide;
    import com.bumptech.glide.request.RequestOptions;
    import com.google.android.material.appbar.MaterialToolbar;
    import com.university.easytrip.R;
    import com.university.easytrip.api.RestClient;
    import com.university.easytrip.databinding.ActivityLoginBinding;
    import com.university.easytrip.databinding.ActivityPlaceHotelBinding;
    public class PlaceHotelActivity extends AppCompatActivity {
        ActivityPlaceHotelBinding binding;
        @Override
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            binding = ActivityPlaceHotelBinding.inflate(getLayoutInflater());;
            setContentView(binding.getRoot());
            Bundle bundle = getIntent().getExtras();
            ImageView imageView = binding.placeImage;
            TextView titleTextView = binding.titleText;
            TextView addressTextView = binding.addressText;
            TextView workingHoursTextView = binding.workingHoursText;
            TextView phoneTextView = binding.phoneText;
            TextView descriptionTextView = binding.descriptionText;
            String title = bundle.getString("title");
            String description = bundle.getString("description");
            String phone = bundle.getString("phone");
            String address = bundle.getString("address");
            String workingHours = bundle.getString("working_hours");
            String logo = bundle.getString("logo");
            DisplayMetrics displayMetrics = new DisplayMetrics();
            getWindowManager().getDefaultDisplay().getMetrics(displayMetrics);
            int screenHeight = displayMetrics.heightPixels;
            int screenWeight = displayMetrics.widthPixels;
            Glide.with(this)
```

```
                    .load(RestClient.baseUrl + logo)
                    .placeholder(R.drawable.loading)
                    .error(R.drawable.no_image)
                    .apply(new RequestOptions().override(screenHeight,
screenWeight))
                    .into(imageView);
            addressTextView.setText(address);
            phoneTextView.setText(phone);
            workingHoursTextView.setText(workingHours);
            titleTextView.setText(title);
            descriptionTextView.setText(description);
            // Add back button to the up bar
            MaterialToolbar toolbar = binding.topAppBar;
            setSupportActionBar(toolbar);
            getSupportActionBar().setDisplayHomeAsUpEnabled(true);
            getSupportActionBar().setDisplayShowHomeEnabled(true);
            toolbar.setNavigationOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    finish();
                }
            });
        }
    }
```

Клас PlaceListHotelActivity.java:

```
package com.university.easytrip.activity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.AutoCompleteTextView;
import android.widget.Spinner;
import android.widget.Toast;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;
import com.google.gson.Gson;
import com.google.gson.TypeAdapter;
import com.university.easytrip.R;
import com.university.easytrip.adapter.HotelAdapter;
import com.university.easytrip.api.RestApi;
import com.university.easytrip.api.RestClient;
import com.university.easytrip.databinding.ActivityPlaceListHotelBinding;
import com.university.easytrip.helper.ProgressBar;
import com.university.easytrip.model.BaseResponse;
import com.university.easytrip.model.City;
import com.university.easytrip.model.Country;
import com.university.easytrip.model.Place;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import io.reactivex.rxjava3.android.schedulers.AndroidSchedulers;
import io.reactivex.rxjava3.annotations.NonNull;
import io.reactivex.rxjava3.core.Observer;
import io.reactivex.rxjava3.disposables.CompositeDisposable;
import io.reactivex.rxjava3.disposables.Disposable;
import io.reactivex.rxjava3.schedulers.Schedulers;
```

```
      import okhttp3.ResponseBody;
      import retrofit2.HttpException;
      public class PlaceListHotelActivity extends NavigationDrawerActivity {
          private ActivityPlaceListHotelBinding binding;
          private ArrayList<Country> countries;
          private ArrayList<City> cities;
          private ArrayList<Place> hotelList;
          ArrayAdapter<Country> countryAdapter;
          ArrayAdapter<City> cityAdapter;
          private ProgressBar progressBarCountry, progressBarCity,
progressBarHotels;
          private CompositeDisposable disposable;
          private RecyclerView recyclerView;
          private RestApi restApi;
          private Integer selectedCityId = 0;
          private Integer selectedCountryId = 0;
          @Override
          protected void onCreate(Bundle savedInstanceState) {
              super.onCreate(savedInstanceState);
              binding =
ActivityPlaceListHotelBinding.inflate(getLayoutInflater());
              setContentView(binding.getRoot());
              //Recycle view init
              recyclerView = binding.placeList;
              recyclerView.setHasFixedSize(true);
              recyclerView.setLayoutManager(new LinearLayoutManager(this));
              // Initialize progress bar
              progressBarCountry = new ProgressBar(this);
              progressBarCity = new ProgressBar(this);
              progressBarHotels = new ProgressBar(this);
              disposable = new CompositeDisposable();
              countries = new ArrayList<>();
              cities = new ArrayList<>();
              hotelList = new ArrayList<>();
              restApi = RestClient.createRetrofitClient().create(RestApi.class);
              binding.dropdownCountryView.setOnItemClickListener(new
AdapterView.OnItemClickListener() {
                  @Override
                  public void onItemClick(AdapterView<?> adapterView, View view,
int position, long l) {
                      selectedCountryId =
((Country)adapterView.getItemAtPosition(position)).getCountryId();
                      ArrayList<City> citiesForSelectedCountry = new
ArrayList<>();
                      if (cities != null && !cities.isEmpty()) {
                          for (City city : cities) {
                              if (city.getCountryId() == selectedCountryId) {
                                  citiesForSelectedCountry.add(city);
                              }
                          }
                          if (citiesForSelectedCountry.isEmpty()) {
                              citiesForSelectedCountry.add(new City("No
selection..."));
                          }
                          cityAdapter = new
ArrayAdapter<>(getApplicationContext(), R.layout.dropdown_item,
citiesForSelectedCountry);
                          binding.dropdownCityView.setAdapter(cityAdapter);
                          binding.dropdownCityView.setText("", false);
                      }
                  }
              });
```

```
            binding.dropdownCityView.setOnItemClickListener(new
AdapterView.OnItemClickListener() {
                @Override
                public void onItemClick(AdapterView<?> adapterView, View view,
int position, long l) {
                    selectedCityId =
((City)adapterView.getItemAtPosition(position)).getCityId();
                }
            });
        restApi.getCountryList()
                    .subscribeOn(Schedulers.io())
                    .observeOn(AndroidSchedulers.mainThread())
                    .subscribe(new  Observer<BaseResponse<ArrayList<Country>>>()
{
                        @Override
                        public void onSubscribe(@NonNull Disposable d) {
                            progressBarCountry.startProgressBar();
                            disposable.add(d);
                        }
                        @Override
                        public void onNext(@NonNull
BaseResponse<ArrayList<Country>> country) {
                            if (country.getSuccess() == 0) {
                                String errorMessage = country.getMessage() !=
null ? country.getMessage() : "";
                                Toast.makeText(PlaceListHotelActivity.this,
"Something went wrong. Try later", Toast.LENGTH_SHORT).show();
                                Log.d("Error", "Error on getting country list: "
+ errorMessage);
                                return;
                            } else if(country.getSuccess() == 1) {
                                if (country.getResponse() != null) {
                                    countries = country.getResponse();
                                    countryAdapter = new
ArrayAdapter<>(getApplicationContext(), R.layout.dropdown_item,
country.getResponse());

binding.dropdownCountryView.setAdapter(countryAdapter);
                                    countryAdapter.notifyDataSetChanged();
                                    binding.dropdownCountryView.setThreshold(1);
                                    Log.d("Success", "Country list has been
caught");
                                    return;
                                }
                            }
                            Toast.makeText(PlaceListHotelActivity.this, "The
internal error has happened. Try again later", Toast.LENGTH_SHORT).show();
                            Log.d("Error", "Internal server error on getting
country list");
                        }
                        @Override
                        public void onError(@NonNull Throwable e) {
                            try {
                                HttpException exception = ((HttpException) e);
                                if (exception.code() == 400) {
                                    ResponseBody responseBody =
exception.response().errorBody();
                                    Gson gson = new Gson();
                                    TypeAdapter<BaseResponse> adapter =
gson.getAdapter(BaseResponse.class);
                                    try {
                                        BaseResponse errorParser =
adapter.fromJson(responseBody.string());
```

```
                                        if (errorParser.getMessage() != null &&
!errorParser.getMessage().isEmpty()) {

Toast.makeText(PlaceListHotelActivity.this, errorParser.getMessage(),
Toast.LENGTH_SHORT).show();

progressBarCountry.stopProgressBar();
                                            return;
                                        } else {
                                            throw new Exception("The internal
error has happened. Try again later");
                                        }
                                    } catch (IOException error) {
                                        Log.d("Error", "Critical internal error:
" + error.getMessage());

Toast.makeText(PlaceListHotelActivity.this, "The internal error has happened. Try
again later", Toast.LENGTH_SHORT).show();
                                        progressBarCountry.stopProgressBar();
                                        return;
                                    }
                                }
                            } catch (Exception error) {
                                progressBarCountry.stopProgressBar();
                                Toast.makeText(PlaceListHotelActivity.this, "The
internal error has happened. Try again later", Toast.LENGTH_SHORT).show();
                                Log.d("Error", "Internal server error on getting
country list");
                            }
                        }
                        @Override
                        public void onComplete() {
                            progressBarCountry.stopProgressBar();
                        }
                    });
            restApi.getCityList()
                    .subscribeOn(Schedulers.io())
                    .observeOn(AndroidSchedulers.mainThread())
                    .subscribe(new  Observer<BaseResponse<ArrayList<City>>>() {
                        @Override
                        public void onSubscribe(@NonNull Disposable d) {
                            progressBarCity.startProgressBar();
                            disposable.add(d);
                        }
                        @Override
                        public void onNext(@NonNull
BaseResponse<ArrayList<City>> city) {
                            if (city.getSuccess() == 0) {
                                String errorMessage = city.getMessage() != null
? city.getMessage() : "";
                                Toast.makeText(PlaceListHotelActivity.this,
"Something went wrong. Try later", Toast.LENGTH_SHORT).show();
                                Log.d("Error", "Error on getting city list: " +
errorMessage);
                                return;
                            } else if(city.getSuccess() == 1) {
                                if (city.getResponse() != null) {
                                    cities = city.getResponse();
                                    cityAdapter = new
ArrayAdapter<>(getApplicationContext(), R.layout.dropdown_item,
city.getResponse());

binding.dropdownCityView.setAdapter(cityAdapter);
```

```
                                    binding.dropdownCityView.setThreshold(1);
                                    Log.d("Success", "Country list has been
caught");
                                    return;
                                }
                            }
                            Toast.makeText(PlaceListHotelActivity.this, "The
internal error has happened. Try again later", Toast.LENGTH_SHORT).show();
                            Log.d("Error", "Internal server error on getting
country list");
                        }
                        @Override
                        public void onError(@NonNull Throwable e) {
                            try {
                                HttpException exception = ((HttpException) e);
                                if (exception.code() == 400) {
                                    ResponseBody responseBody =
exception.response().errorBody();
                                    Gson gson = new Gson();
                                    TypeAdapter<BaseResponse> adapter =
gson.getAdapter(BaseResponse.class);
                                    try {
                                        BaseResponse errorParser =
adapter.fromJson(responseBody.string());
                                        if (errorParser.getMessage() != null) {

Toast.makeText(PlaceListHotelActivity.this, errorParser.getMessage(),
Toast.LENGTH_SHORT).show();
                                            progressBarCity.stopProgressBar();
                                            return;
                                        } else {
                                            throw new Exception("The internal
error has happened. Try again later");
                                        }
                                    } catch (IOException error) {
                                        Log.d("Error", "Critical internal error:
" + error.getMessage());

Toast.makeText(PlaceListHotelActivity.this, "The internal error has happened. Try
again later", Toast.LENGTH_SHORT).show();
                                        progressBarCity.stopProgressBar();
                                        return;
                                    }
                                }
                            } catch (Exception error) {
                                progressBarCity.stopProgressBar();
                                Toast.makeText(PlaceListHotelActivity.this, "The
internal error has happened. Try again later", Toast.LENGTH_SHORT).show();
                                Log.d("Error", "Internal server error on getting
city list");
                            }
                        }
                        @Override
                        public void onComplete() {
                            progressBarCity.stopProgressBar();
                        }
                    });
            binding.searchPlacesBtn.setOnClickListener(new
View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    if (selectedCityId == 0) {
```

```
                        Toast.makeText(PlaceListHotelActivity.this, "Please,
select city before searching", Toast.LENGTH_SHORT).show();
                        return;
                    }
                    restApi.getHotelsByCityId(selectedCityId)
                            .subscribeOn(Schedulers.io())
                            .observeOn(AndroidSchedulers.mainThread())
                            .subscribe(new
Observer<BaseResponse<List<Place>>>() {
                                @Override
                                public void onSubscribe(@NonNull Disposable d) {
                                    progressBarHotels.startProgressBar();
                                    disposable.add(d);
                                }
                                @Override
                                public void onNext(@NonNull
BaseResponse<List<Place>> hotel) {
                                    if (hotel.getSuccess() == 0) {
                                        String errorMessage = hotel.getMessage()
!= null ? hotel.getMessage() : "";

Toast.makeText(PlaceListHotelActivity.this, "Something went wrong. Try later",
Toast.LENGTH_SHORT).show();
                                        Log.d("Error", "Error on getting city
list: " + errorMessage);
                                        return;
                                    } else if(hotel.getSuccess() == 1) {
                                        if (hotel.getResponse() != null) {
                                            HotelAdapter hotelAdapter = new
HotelAdapter(PlaceListHotelActivity.this, new ArrayList(hotel.getResponse()));

recyclerView.setAdapter(hotelAdapter);
                                            Log.d("Success", "Country list has
been caught");
                                            return;
                                        }
                                    }
                                    Toast.makeText(PlaceListHotelActivity.this,
"The internal error has happened. Try again later", Toast.LENGTH_SHORT).show();
                                    Log.d("Error", "Internal server error on
getting country list");
                                }
                                @Override
                                public void onError(@NonNull Throwable e) {
                                    try {
                                        HttpException exception =
((HttpException) e);
                                        if (exception.code() == 400) {
                                            ResponseBody responseBody =
exception.response().errorBody();
                                            Gson gson = new Gson();
                                            TypeAdapter<BaseResponse> adapter =
gson.getAdapter(BaseResponse.class);
                                            try {
                                                BaseResponse errorParser =
adapter.fromJson(responseBody.string());
                                                if (errorParser.getMessage() !=
null && !errorParser.getMessage().isEmpty()) {

Toast.makeText(PlaceListHotelActivity.this, errorParser.getMessage(),
Toast.LENGTH_SHORT).show();

progressBarHotels.stopProgressBar();
```

```
                                                    return;
                                                } else {
                                                    throw new Exception("The
internal error has happened. Try again later");
                                                }
                                            } catch (IOException error) {
                                                Log.d("Error", "Critical
internal error: " + error.getMessage());

Toast.makeText(PlaceListHotelActivity.this, "The internal error has happened. Try
again later", Toast.LENGTH_SHORT).show();

progressBarHotels.stopProgressBar();
                                                return;
                                            }
                                        }
                                    } catch (Exception error) {
                                        progressBarHotels.stopProgressBar();

Toast.makeText(PlaceListHotelActivity.this, "The internal error has happened. Try
again later", Toast.LENGTH_SHORT).show();
                                        Log.d("Error", "Internal server error on
getting hotels list");
                                    }
                                }
                                @Override
                                public void onComplete() {
                                    progressBarHotels.stopProgressBar();
                                }
                            });
                }
            });
        }
        @Override
        protected void onDestroy() {
            disposable.dispose();
            super.onDestroy();
        }
    }
```

Клас HotelAdapter.java:

```
package com.university.easytrip.adapter;
import android.content.Context;
import android.content.Intent;
import android.graphics.text.LineBreaker;
import android.os.Build;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.TextView;
import androidx.annotation.NonNull;
import androidx.constraintlayout.widget.ConstraintLayout;
import androidx.recyclerview.widget.RecyclerView;
import com.bumptech.glide.Glide;
import com.bumptech.glide.request.RequestOptions;
import com.university.easytrip.R;
```

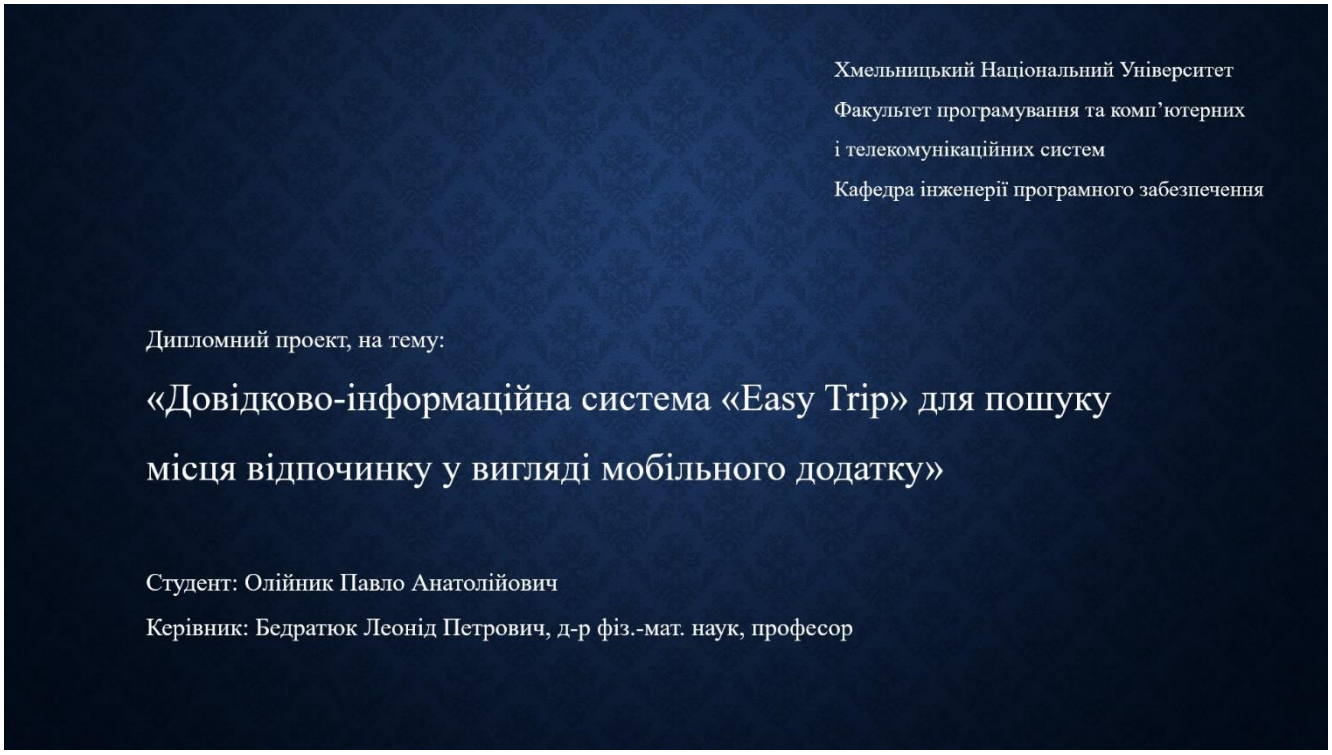
```
      import com.university.easytrip.activity.PlaceHotelActivity;
      import com.university.easytrip.api.RestClient;
      import com.university.easytrip.model.Place;
      import java.util.ArrayList;
      import java.util.List;
      public class HotelAdapter extends
RecyclerView.Adapter<HotelAdapter.HotelHolder> {
          private Context context;
          private List<Place> hotelList;
          public HotelAdapter(Context context, ArrayList<Place> hotels) {
              this.context = context;
              this.hotelList = hotels;
          }
          @NonNull
          @Override
          public HotelHolder onCreateViewHolder(@NonNull ViewGroup parent, int
viewType) {
              View view =
LayoutInflater.from(this.context).inflate(R.layout.activity_place_list_hotel_item,
parent, false);
              return new HotelHolder(view);
          }
          @Override
          public void onBindViewHolder(@NonNull HotelHolder holder, int position)
{
              Place hotel = hotelList.get(position);
              holder.hotelDescription.setText(hotel.getDescription());
              holder.hotelTitle.setText(hotel.getName());
              Glide.with(context)
                      .load(RestClient.baseUrl + hotel.getLogo())
                      .placeholder(R.drawable.loading)
                      .error(R.drawable.no_image)
                      .apply(new RequestOptions().override(165, 145))
                     .into(holder.hotelImage);
              holder.constraintLayout.setOnClickListener(new
View.OnClickListener() {
                  @Override
                  public void onClick(View v) {
                      Intent intent = new Intent(context ,
PlaceHotelActivity.class);
                      Bundle bundle = new Bundle();
                      bundle.putString("title" , hotel.getName());
                      bundle.putString("description" ,hotel.getDescription());
                      bundle.putString("phone" ,hotel.getPhone());
                      bundle.putString("working_hours" ,hotel.getWorkingHours());
                      bundle.putString("address" , hotel.getAddress());
                      bundle.putString("logo" , hotel.getLogo());
                      intent.putExtras(bundle);
                      context.startActivity(intent);
                  }
              });
          }
          @Override
          public int getItemCount() {
              return hotelList.size();
          }
          public class HotelHolder extends RecyclerView.ViewHolder {
              ImageView hotelImage;
              TextView hotelTitle, hotelDescription;
              ConstraintLayout constraintLayout;
              public HotelHolder(@NonNull View itemView) {
                  super(itemView);
```

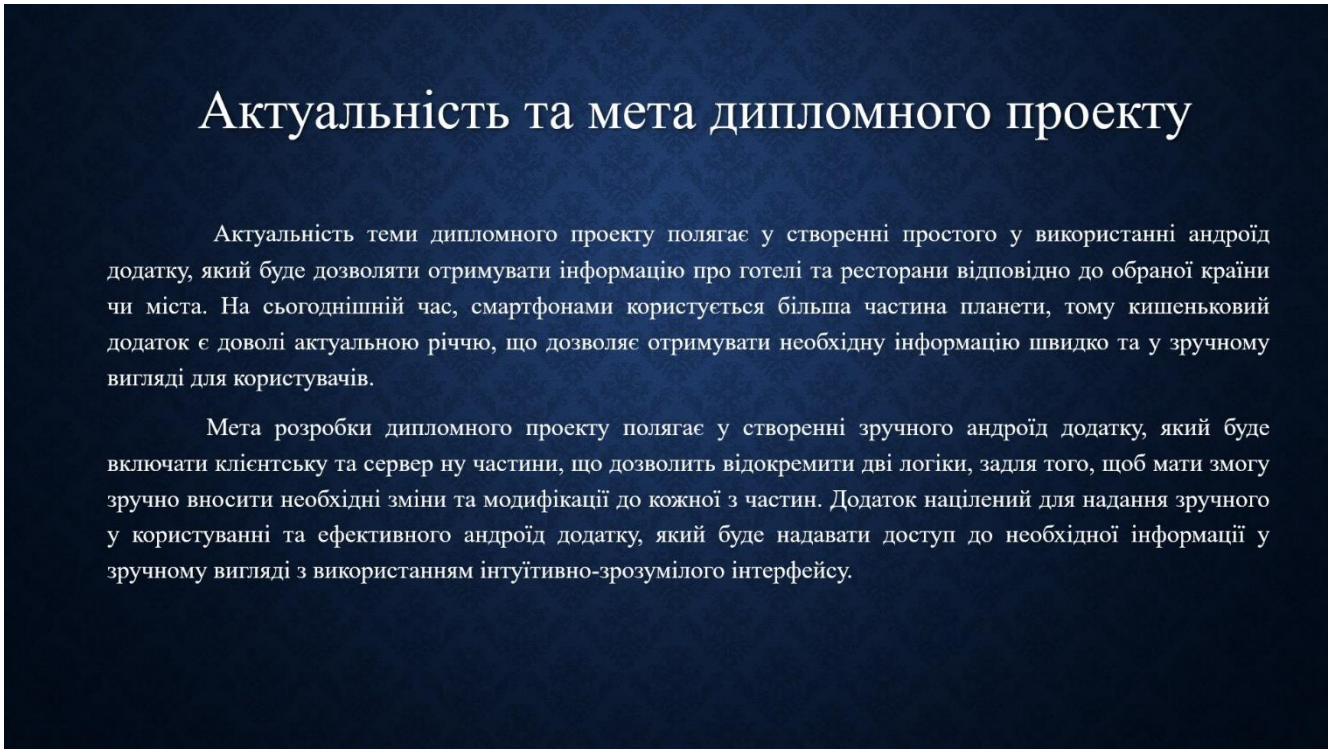
```
                    this.hotelImage =
itemView.findViewById(R.id.place_list_hotel_item_image);
                    this.hotelTitle =
itemView.findViewById(R.id.place_list_hotel_item_title_label);
                    this.hotelDescription =
itemView.findViewById(R.id.place_list_hotel_item_description_label);
                    this.constraintLayout =
itemView.findViewById(R.id.place_list_hotel_item_constraint_main);
                }
            }
        }
```

ДОДАТОК В

(Обов’язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ



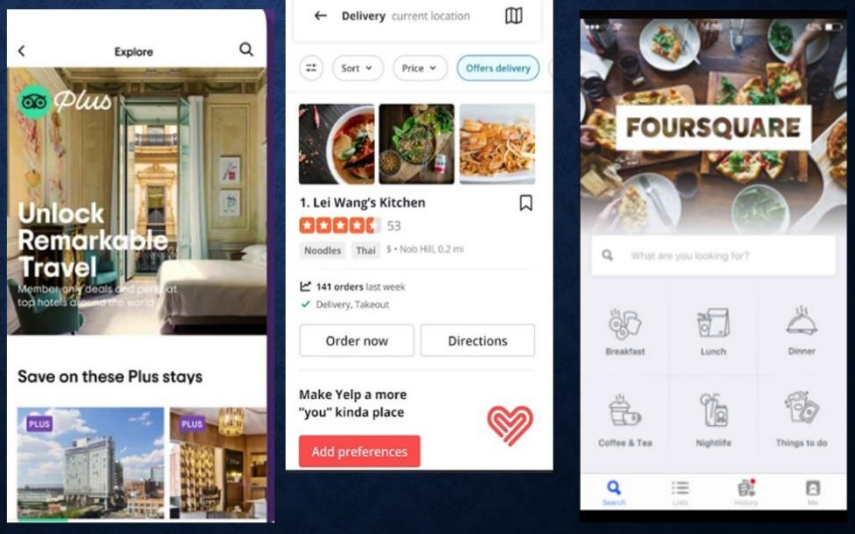


Завдання дипломного проекту

- провести аналіз предметної області, виділити базові поняття;
- проаналізувати існуючі програмні рішення, що пропонують схожий функціонал у порівнянні із розроблюваним мобільним застосунком;
- зробити проектування, яке буде наглядно представляти структуру додатку та його окремих частин;
- реалізувати клієнтську частину з інтерфейсом користувача;
- реалізувати серверну частину, яка буде отримувати вхідні запити та видавати відповідні відповіді;
- провести тестування додатку, щоб виявити несправності у роботі та вирішити їх.

Наявне програмне забезпечення

- TripAdvisor;
- Yelp;
- Foursquare;



Проектування мобільного додатку

Мобільний додаток розробляється із використанням двох незалежних частин, якими є клієнт, що допомагає відображати візуальний інтерфейс користувача, а також сервер, що призначений для надання інтерфейсу клієнту через який проводиться спілкування між ними. Дані між клієнтом та сервером будуть передаватись через веб-мережу у форматі JSON. Також, на стороні серверу міститься база даних, яка виступає сховищем даних, дані якої повертаються клієнтам або модифікуються відповідно до певних HTTP запитів. Задля реалізації даної поведінки, буде використана саме клієнт-серверна архітектура.

Клієнт-серверна архітектура – це один із видів архітектурних підходів, що використовується при проектуванні програмного забезпечення. В даній архітектурі виділяють два основних поняття якими є клієнт та сервер. Спілкування базується на використанні мережевого протоколу HTTP.

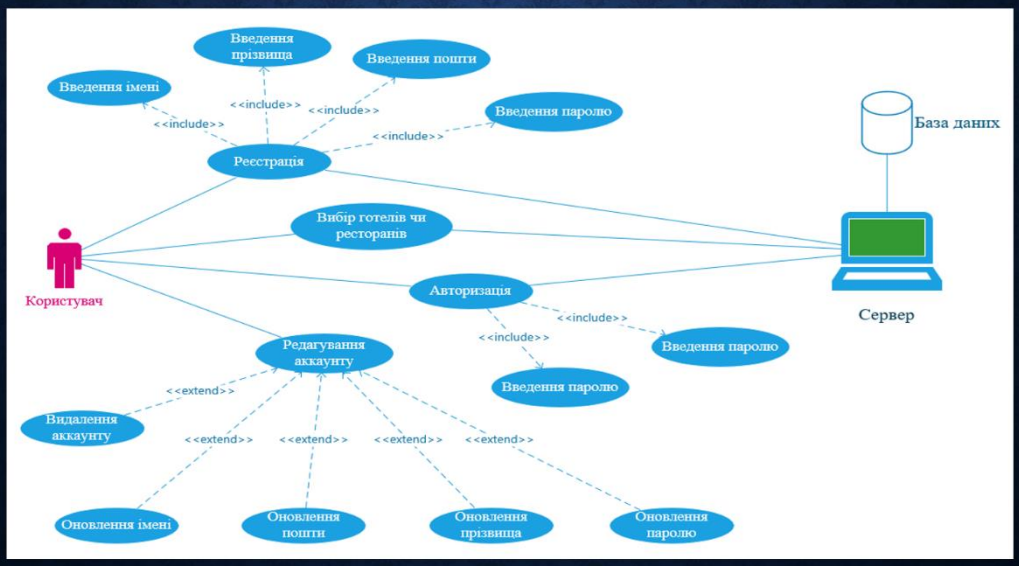
Проектування мобільного додатку

Необхідно визначити спосіб взаємодії клієнтської та серверної частин. Це можна зробити з використанням відомого архітектурного підходу, який саме призначений для стандартизації, або простіше кажучи для опису взаємодії клієнта із сервером, що має назву REST API.

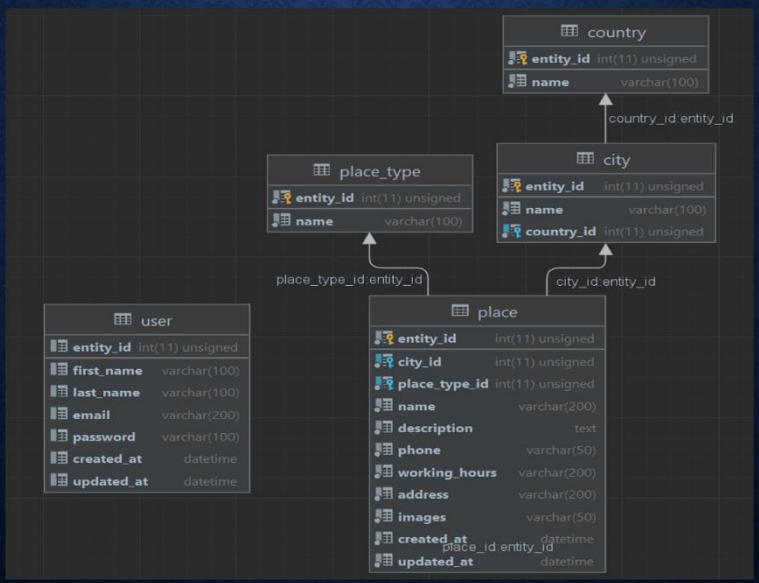
REST API – це архітектурний підхід, що описує правила взаємодії між клієнтом та сервером. REST використовує протокол HTTP для передачі даних через мережу. Це дозволяє використовувати такі найбільш поширені HTTP методи як POST, PUT, DELETE та GET. Дані методи відповідають базовим операціями по створенню, оновленню, видалення та доступу до даних у базі даних. У REST архітектурі використовуються такі формати для передачі даних як XML та JSON.

Для взаємодії надаються інтерфейси, а саме URI адреси, через які відбувається доступ клієнта до даних на віддаленому сервері. Це дозволяє надзвичайно спростити налагодження комунікації, оскільки потрібно лише знати шлях, тобто адресу куди потрібно відсилати дані, а також формат даних, що відсилаються.

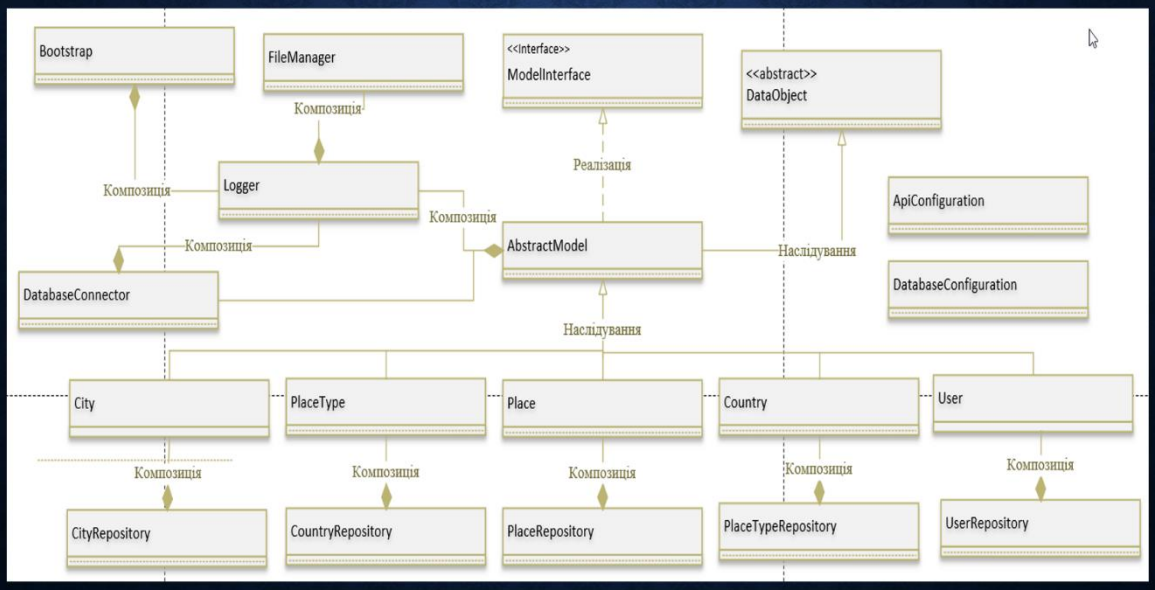
Діаграма варіантів використання



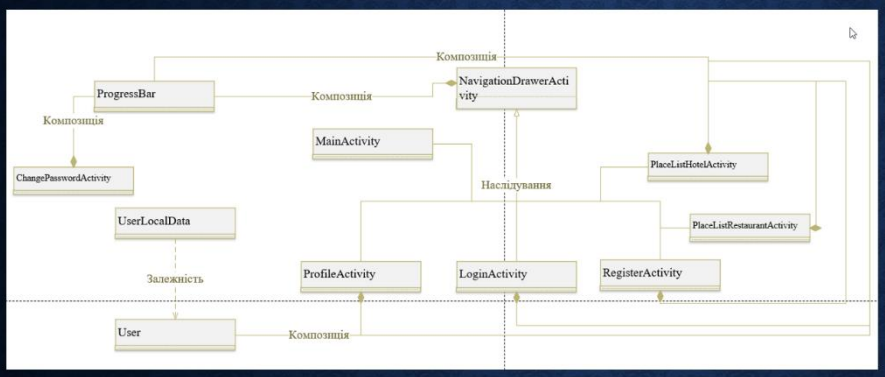
Проектування структури бази даних



Діаграма класів серверної частини



Діаграма класів клієнтської частини



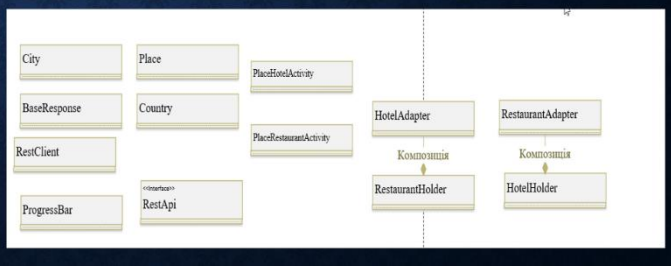
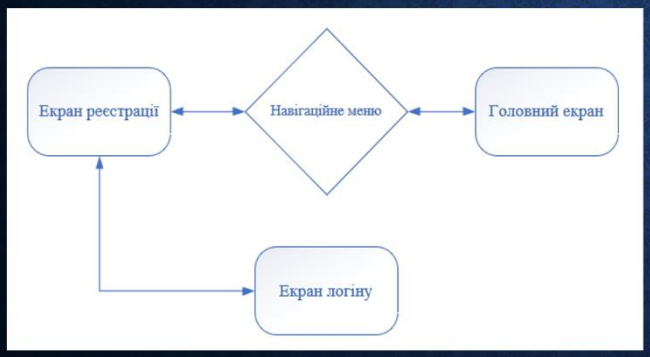
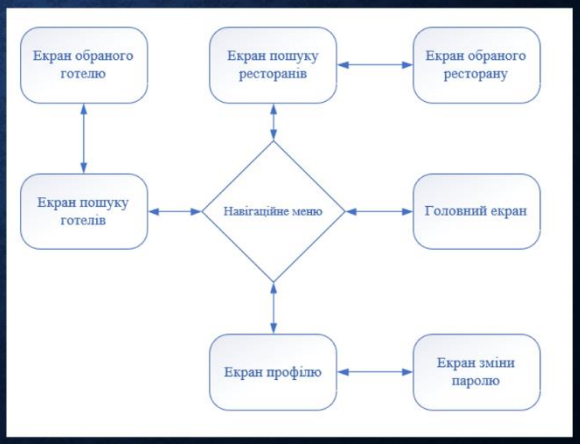
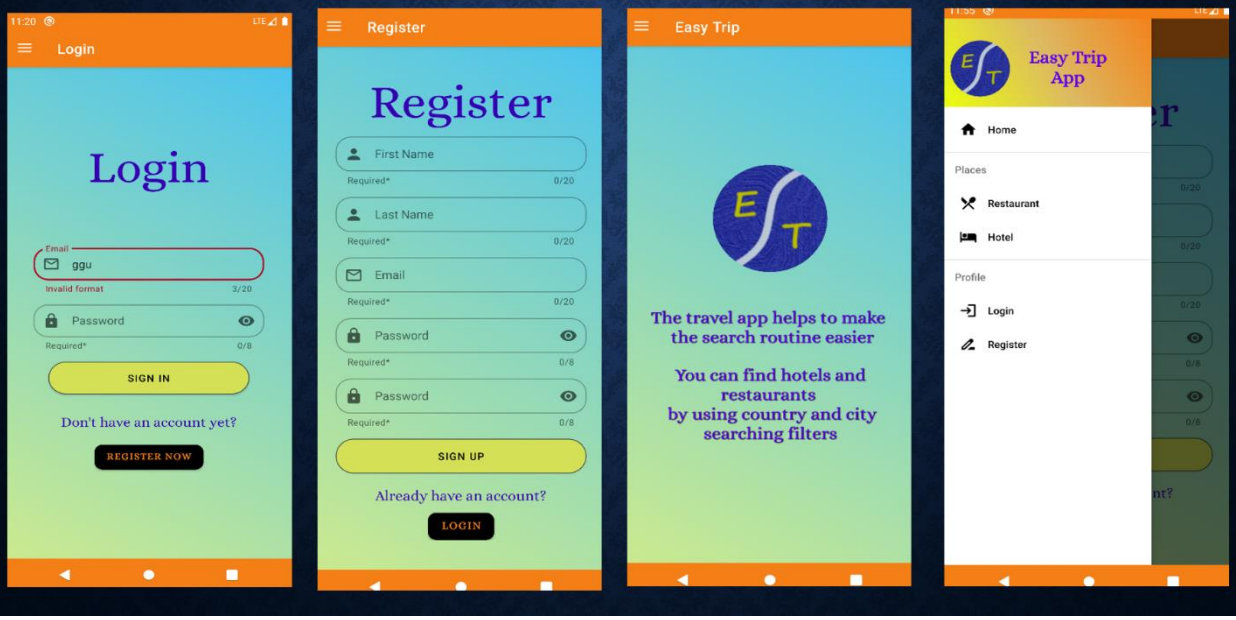


Схема переходів між екранами для не зареєстрованих(ліворуч) та зареєстрованих(праворуч) користувачів

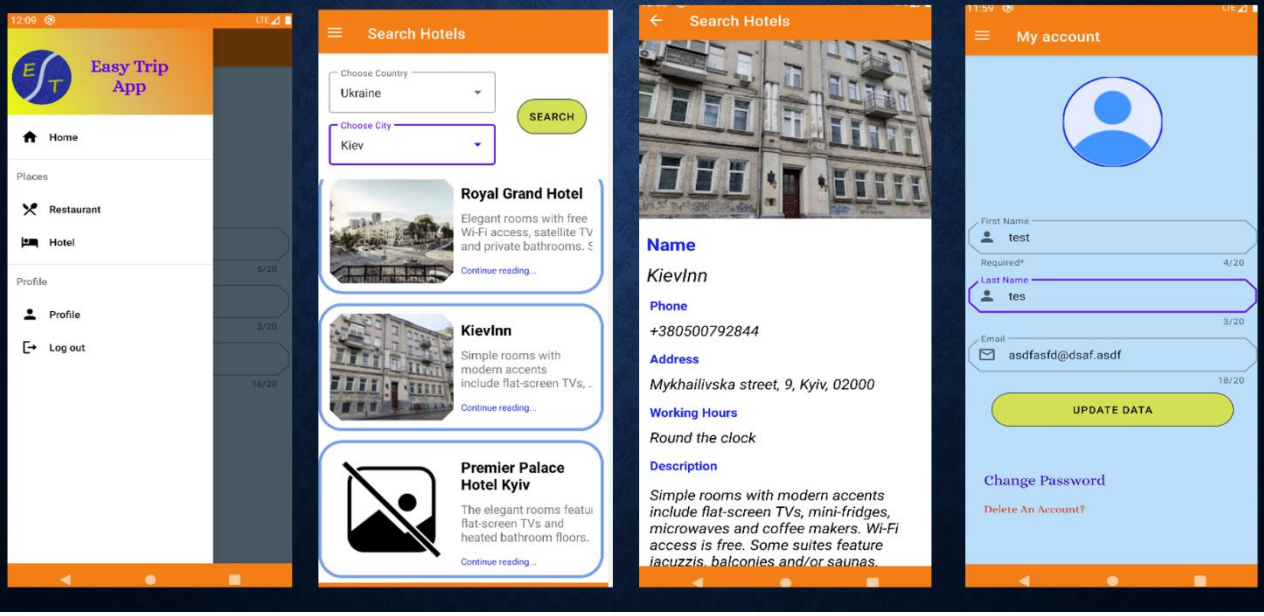




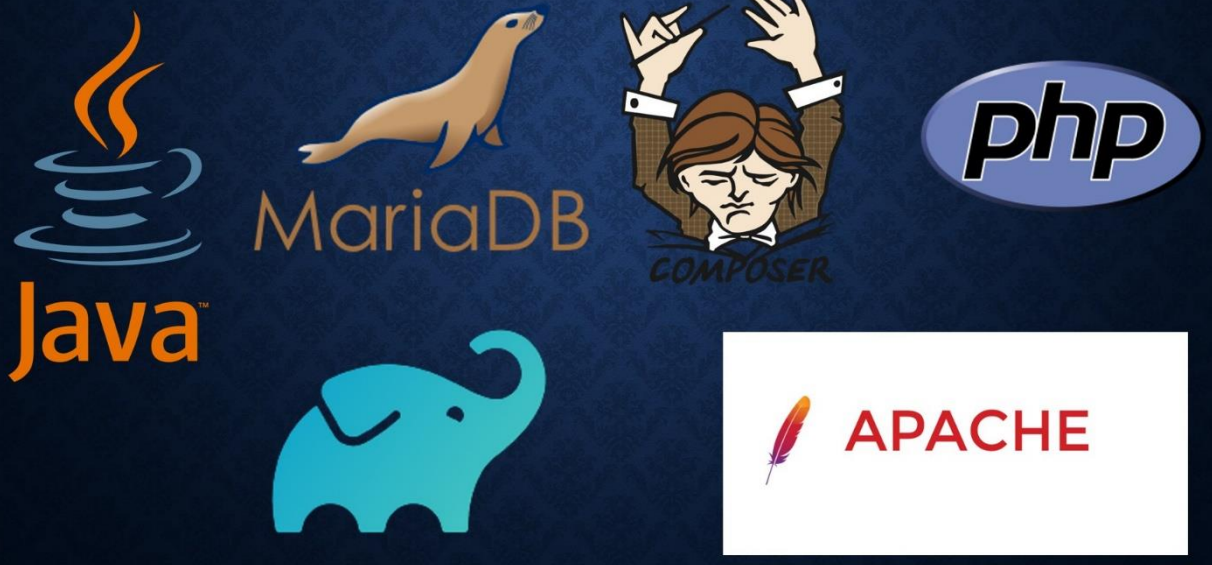
Зовнішній вигляд інтерфейсу додатку



Зовнішній вигляд інтерфейсу додатку



Інструменти та технології



Висновки

Під час виконання дипломної роботи було проведено детальний аналіз предметної області, а також огляд існуючих програмних засобів. Виділено основні вимоги, яким повинен відповідати мобільний застосунок, після чого було розроблене технічне завдання. Здійснено проектування додатку, завдяки якому була підібрана клієнт-серверна архітектура, а також проведена декомпозиція на дві незалежні частини додатку, а саме клієнт та сервер. Створена архітектура бази даних, яка подана у вигляді фізичної моделі. Також були створені діаграми класів для серверної та клієнтської частин, що дозволило зрозуміти способи їх подальшої реалізації.

На основі зробленого проектування здійснено реалізацію серверної та клієнтської частин, а також структури бази даних. Було розроблене керівництво користувача, а також неведені технічні характеристики додатку. Для розробленого мобільного застосунку проведене тестування з використанням методики «чорний ящик», функціонального та конфігураційного тестування.

В результаті роботи над дипломним проектом був розроблений мобільний додаток, що працює на операційній системі Android, який допомагає шукати інформацію про готелі та ресторани та виводити її у зручному для користувача вигляді. Додаток оснащений доволі простим, але привабливим дизайном. Завдяки дипломному проекту, вдалось покращити та набути нові навички при розробці програмного забезпечення, що допоможе мені у майбутніх проектах, а також у подальшій роботі по спеціальності. Після захисту дипломного проекту, планується продовження роботи над програмним продуктом задля покращення продуктивності його роботи, а також для додавання нового функціоналу, що дозволить привернути увагу користувачів.

Завідувачу кафедри інженерії програмного забезпечення проф. Бедратюку Л. П.

здобувача вищої освіти
Олійника П. А.
Прізвище, ініціали

факультет ІТ, 4 курс, група ІПЗ-18-1

ЗАЯВА

З правилами чинного Положення «Про дотримання академічної доброчесності в Хмельницькому національному університеті» від 26.09.2020 (зі змінами від 26.11.2020), згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

30.05.2022
дата


підпис

Fri May 27 10:05:39 EEST 2022, Хіврич Володимир Русланович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 20.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 10%**

ID: 104077 Назва: Довідково-інформаційна система «Easy Trip» для пошуку місця відпочинку у вигляді мобільного додатку Додано в БД: 2022-05-27 Автора: П.А.Олійник Керівники: Л. П. Бедратюк Консультанти: Опоненти:	Документ		Сумарний збіг по Базі Даних	
	Символи	Лексеми	Символи	Лексеми
	128194	1180	31193 (24%)	288 (24%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми
103252	Назва: Зіт з переддипломної практики Додано в БД: 2022-05-04 Автора: Олійник П.А. Керівники: Бедратюк Л. П. Консультанти: Опоненти:	25798 (20.0%)	222 (19.0%)

UNICHECK

Ім'я користувача:
Кафедра ІПЗ

ID перевірки:
1011347849

Дата перевірки:
27.05.2022 11:10:34 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
27.05.2022 11:14:09 EEST

ID користувача:
100005589

Назва документа: **ДПІПЗ.Олійник.П.А**

Кількість сторінок: **121** Кількість слів: **20512** Кількість символів: **160638** Розмір файлу: **2.06 MB** ID файлу: **1011233710**

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

8.98%
Схожість

Найбільша схожість: **4.64%** з джерелом з Бібліотеки (ID файлу: **1011233715**)

4.73% Джерела з Інтернету	340	Сторінка 123
6.11% Джерела з Бібліотеки	136	Сторінка 127

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	3
Підозріле форматування	27 сторінок

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА ДИПЛОМНИЙ ПРОЕКТ
освітнього ступеня «Бакалавр»

Дипломник Олійник Павло Анатолійович

Тема Довідково-інформаційна система «Easy Trip» для пошуку місця відпочинку у вигляді мобільного додатку

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг дипломного проекту:

Кількість листів креслень _____; кількість сторінок записки 180

1.Короткий зміст пояснювальної записки та прийнятих рішень У дипломній роботі було проаналізовано предметну область, а також наведено актуальність розробки мобільного додатку. Подано діаграми, що наглядно представляють архітектуру додатку, а також виділено основні архітекурні підходи. На основі обраної архітектури відбулась реалізація програмного продукту. Проведено ряд тестувань, що виконувались задля виявлення помилок у роботі додатку.

2. Висновок про відповідність проекту поставленому завданню Дипломна робота була виконана з дотриманням усіх вимог, а також стандартів стосовно оформлення.

3. Характеристика виконання кожного розділу проекту, ступінь використання останніх досягнень науки і техніки та передових методів роботи Вступ містить пояснювальну частину, що наводить актуальність розробки програмного продукту, а також мету, що описує виконання послідовних кроків на основі яких відбувається написання роботи. Перший розділ містить пояснення щодо опису та аналізу прдеметної області у якій ведеться розробка. Також, у першому розділі провоодиться аналіз існуючого програмного забезпечення, що пропонує схожий функціонал у порівнянні із додатком над яким ведеться розробка. Було виділено ряд вимог, яким повинен відповідати мобільний додаток, а також розроблено технічне завдання. У другому розділі було проведено аналіз та проектування архітектури додатку, були виділені основні типи архітектур, що використовуються у процесі роботи над програмним забезпеченням, а також подано ряд діаграм, що дозволяють якісно та в повній мірі відобразити суть основних модулів системи. Розроблено інтерфейс користувача, у якому показаний зовнішній вигляд майбутнього додатку, а також схема переходів між екранами. У третьому розділі було проведено безпосередню реалізацію мобільного додатку, а також наведено пояснення основних моментів роботи програмного коду і подано його окремі частини. У четвертому розділі було проведено ряд тестувань, що дозволили підтвердити працездатність мобільного додатку.

4. Позитивні сторони проекту Розробка даного мобільного додатку є доволі актуальною, оскільки подорожі є широкорозповсюдженим явищем сьогодення і тому важливо отримувати інформацію швидко та у стислому вигляді, що досягається за допомогою використання смартфону, який завжди є під рукою.

5. Негативні сторони проекту Було б добре реалізувати додаткові функції у додатку, такі як наприклад можливість написання відгуків для окремого готелю чи ресторану, що дозволить краще орієнтуватись користувачам при виборі того чи іншого місця.

6. Оцінка графічного оформлення та пояснювальної записки проекту Графічне оформлення було виконане згідно тематики дипломного проекту, а також складається із рисунків та різноманітних діаграм. Пояснювальна записка виконана із дотриманням вимог чинних стандартів.

7. Відгук про дипломний проект в цілому Зміст дипломної роботи поданий доволі чітко та структуровано. Пояснення усієї роботи є досить детальним та зрозумілим, що дозволяє надзичайно легко орієнтуватись у наведених поняттях, а також у роботі вцілому. Графічна частина якісно відображає елементи, які пояснюються у тексті.

8. Інші зауваження

9. Оцінка дипломного проекту Дипломний проект виконано з дотриманням усіх вимог, який у повній мірі відповідає тематиці роботи та заслуговує на оцінку «відмінно».

РЕЦЕНЗЕНТ (прізвище, ім'я, по-батькові,посада, місце роботи)

Муляр І В, к. т. н., доцент кафедри кібербезпеки

«25» травня 2022 р.

(підпис)