Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: **Розробка мобільного застосунку для управління організацією велопробігів**

Виконала: студентка IV курсу, групи СТ-41
спеціальності 126 Інформаційні системи та технології
(шифр і назва спеціальності)

(підпис) Генсьора І.К. (прізвище та ініціали)

Керівник (підпис) Кравчук Г.Б. (прізвище та ініціали)
Нормоконтроль (підпис) Шимчук Г.В. (прізвище та ініціали)
Завідувач кафедри (підпис) Боднарчук І.О. (прізвище та ініціали)
Рецензент (підпис) Жаровський Р.О. (прізвище та ініціали)

Тернопіль
2024

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«____» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 126 Інформаційні системи та технології
(шифр і назва спеціальності)

Студенту Генсьора Ірина Костянтинівна
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка мобільного застосунку для управління організацією велопробігів

Керівник роботи Кравчук Галина Богданівна асистент кафедри КН
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «*29*» *квітня* 2024 року № *4/7-471*

2. Термін подання студентом завершеної роботи 24 червня 2024р.

3. Вихідні дані до роботи Наукові публікації, електронні ресурси, підручники , посібники

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. Розділ 1. Аналіз предметної області. 1.1. Обгрунтування розробки мобільного застосунку. 1.2. Огляд і порівняльний аналіз аналогів. 1.3. Специфікація вимог та вибір засобів та методу розробки мобільного застосунку. Розділ 2. Проектування мобільного застосунку. 2.1. Архітектура мобільного застосунку для управління організацією велопробігів. 2.2. Проектування бази даних.242.3. Розробка API. Розділ 3. Програмна реалізація мобільного за стосунку. 3.1. Аналіз та вибір засобів реалізвції мобільного застосунку. 3.2. Програмна реалізація інтерфейсної частини мобільного застосунку. 3.3. Структура проекту. 3.3. Опис проекту API. 3.4. Зберігання вихідних даних. 3.5. Тестування додатку. 4. Безпека життєдіяльності, основи охорони праці. Висновки. Перелік джерел. Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Тема. 2. Огляд аналогів. 3. Мікросервісна архітектура .4. Діаграма варіантів використання. 5. Діаграми послідовності. 6. Проект бази даних. 7. Структура проекту інтерфейсу мобільного застосунку. 8. Тестування. 9-11. Інструкція користувача мобільного застосунку для управління організацією велопробігів. 12.Висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Сенчишин В. С. кандидат технічних наук, доцент кафедри МТ	12.06.2024	15.06.2024

7. Дата видачі завдання 29 січня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	30.01.2024	*Виконано*
2.	Підбір джерел щодо розробки мобільного застосунку для управління організацією велопробігів.	31.01.2024-03.02.2024	*Виконано*
3.	Опрацювання джерел по темі кваліфікаційної роботи	04.02.2024-06.02.2024	*Виконано*
4.	Виконання дослідження щодо розробки мобільного застосунку для управління організацією велопробігів**.**	07.02.2024-11.02.2024	*Виконано*
5.	Оформлення розділу «Аналіз предметної області»	03.06.2024-05.06.2024	*Виконано*
6.	Оформлення розділу «Проектування мобільного	06.06.2024-08.06.2024	*Виконано*
	застосунку»		
6.	Оформлення розділу «Програмна реалізація	09.06.2024-11.06.2024	*Виконано*
	мобільного застосунку»		
7.	Виконання завдання до підрозділу «Безпека життєдіяльності»	12.06.2024-13.06.2024	*Виконано*
8.	Виконання завдання до підрозділу «Основи охорони	14.06.2024-15.06.2024	*Виконано*
	праці»		
9.	Оформлення кваліфікаційної роботи	16.06.2024-17.06.2024	*Виконано*
10.	Нормоконтроль	18.06.2024-19.06.2024	*Виконано*
11.	Перевірка на плагіат	20.06.2024	*Виконано*
12.	Попередній захист кваліфікаційної роботи	21.06.2024	*Виконано*
13.	Захист кваліфікаційної роботи	24.06.2024	

Студент _____ Генсьора І.К.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Кравчук Г.Б.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка мобільного застосунку для управління організацією велопробігів // Кваліфікаційна робота освітнього рівня «Бакалавр» // Генсьора Ірина Костянтинівна // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СТ-41// Тернопіль, 2024 // С.- 66 , рис. – 44, табл. – 1, додат. – 1, бібліогр. – 36.

Ключові слова: мобільний застосунок, веб-застосунок, організація велопробігів, проектування мобільного за стосунку

Кваліфікаційна робота присвячена розробці мобільного застосунку для організації спільних велотурів. В першому розділі кваліфікаційної роботи здійснено опис предметної області,здійснено порівняльний аналіз відомих застосунків, виділено основні засоби для реалізації мобільного застосунку для управління організацією велопробігів.

В другому розділі кваліфікаційної роботи описано архітектуру розробленого мобільного застосунку, структуру бази даних та особливості реалізації API для взаємодії з базою даних.

В третьому розділі кваліфікаційної роботи описано особливості програмної реалізації мобільного застосунку для управління організацією велопробігів, а також подано структуру проекту та структуру API методів. Проведено тестування мобільного застосунку та охарактеризовано основні особливості роботи із Android додатком для проектування та генерації SQL коду.

ANNOTATION

Development of a mobile application for managing the organization of bicycle races // Qualification work of the educational level "Bachelor" // Gensiora Iryna Kostyantynivna // Ternopil Ivan Pulyu National Technical University, Computer and Information Systems and Software Engineering Faculty, Computer Sciences Department, ST Group -41// Ternopil, 2024 // P. 66, fig. - 44, tab. - 1, add. – 1, bibliography - 36.

Keywords: mobile application, web application, organization of bicycle races, mobile design for the relationship

The qualification work is devoted to the development of a mobile application for the organization of joint bicycle tours. In the first section of the qualification work, a description of the subject area was carried out, a comparative analysis of known applications was carried out, the main tools for the implementation of a mobile application for managing the organization of bicycle races were selected.

The second section of the qualification work describes the architecture of the developed mobile application, the structure of the database and the features of the API implementation for interaction with the database.

The third section of the qualification work describes the features of the software implementation of the mobile application for managing the organization of bicycle races, as well as the structure of the project and the structure of API methods. The mobile application was tested and the main features of working with the Android application for designing and generating SQL code were characterized.

ЗМІСТ

ВСТУП

Актуальність теми. Мобільні пристрої, невід'ємна частина життя сучасної людини. Мобільні пристрої широко впроваджуються у сферу сервісу, у зв'язку з чим з'являється потреба у розробці мобільних застосунків, що оптимізують діяльність людини.

На даний момент при плануванні спільних велопробігів виникають проблеми, щодо організації поїздок дану проблему частково вирішують групи у соціальних мережах, але їх велике кількість, в результаті чого:

1. Важко розміщувати інформацію, через множину груп.
2. Відсутня прозорість заходів, оскільки учасники набираються з різних груп.
3. Ймовірність відсутності актуальності заходу.
4. Довготривале повідомлення інформації до об'єкту.
5. Складна організованість заходів.

Мета та завдання дослідження. Мета роботи – розробити мобільний застосунок для організації спільних велотурів.

Завдання:

1. Виконати аналіз стану проблеми та підходів до її вирішення.
2. Обґрунтувати вибір технологій реалізації та необхідних програмних платформ.
3. Відповідно до сформованих вимог провести розробку мобільного додатку.

Практичне значення одержаних результатів. Актуальність даного мобільного застосунку підтверджується рішенням описаних проблем. Мобільний застосунок буде виступати єдиною точкою входу та виходу інформації. Буде зручніше організовувати процес, зручніше вносити коригування, своєю чергою учасникам буде легше знаходити необхідну інформацію і також відстежувати внесені зміни до заходу.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Обґрунтування розробки мобільного застосунку

Мобільний застосунку - це програмне забезпечення, спеціально розроблене під конкретну мобільну платформу (iOS, Android, Windows Phone і т.д.). Призначено для використання на смартфонах, планшетах, розумному годиннику та інших мобільних пристроях [1, 2].

Програмне забезпечення (далі «ПЗ») - це взаємодія кожною з частин системи логічних ланцюжків нулів і одиниць, що працюють за певним алгоритмом обробки та роботи з інформацією, які так само можуть бути програмами. Тобто якщо говорити простими словами, то це програма, розроблена якоюсь мовою програмування, в якій зашитий алгоритм її роботи.

Мобільна платформа – це операційна система для телефонів за аналогією до комп'ютерів [3].

На поточний день, активну популярність набирає активний спорт катання на велосипедах. Для організації спільного катання, а також участі у такому заході, використовується оголошення в групах соціальних мережах. Але таких груп просто величезна кількість, доводиться дивитися всі групи, всі пропозиції і на це витрачаються колосальні зусилля, що навіть бажання покататися просто пропадає.

Таким чином, виникають проблеми:

Достовірність інформації – не актуалізується інформація, після внесення коригувань.

Доступність інформації – відсутність соціальних мереж.

Прозорість інформації – немає чіткого розуміння, скільки і хто саме є учасником або організатором.

Відсутність онлайн спілкування за допомогою телефону з організатором заходу.

Для організації заходу необхідно витратити чимало зусиль.

Після виявлення проблемних місць, з'являється два варіанти рішення – це створення сайту або створення мобільного застосунку.

Розглянемо переваги мобільного застосунку по відношенню до сайту:

Продуктивність - додаток достатньо один раз встановити телефон для роботи з ним. Після цього робота з додатком буде в дві дії: відкрити телефон, відкрити додаток. Для використання сайту необхідно виконувати три дії: відкрити телефон, відкрити браузер, відкрити сайт. Додатковий фактор вибору мобільної програми, браузери на сьогоднішній день дуже ресурсомісткі.

Доступність повідомлення інформації – за допомогою мобільного застосунку досить легко розміщувати та читати інформацію. Додатково можливо реалізувати push - сповіщення скоротять час повідомлення інформації до об'єкту.

Виходячи з вище переліченого, вибір був зроблений на користь мобільного застосунку. Після реалізації мобільного застосунку з'явиться можливість в одному місці знаходити відповідні маршрути, поїздки, можна листуватися з організатором, що полегшить організованість та узгодженість маршруту. Дзвонити, побачити всіх учасників, побачити фото учасників, стати самому організатором поїздки та саме головне, це завжди під рукою в телефоні.

1.2 Огляд і порівняльний аналіз аналогів

Велосипедні програми для планування маршруту та навігації допоможуть вам знайти більше доріг і місць для їзди.

Найкращі велопрограми, призначені для планування маршрутів, дозволять вам планувати власні поїздки, а також відкривати маршрути від інших гонщиків.

Деякі додатки виконають важку роботу та спланують для вас маршрут, якщо ви введете пункт призначення, що ідеально підходить для пригод у дорозі чи велосипедних прогулянок містом.

Bikemap (рисунок 1.1.). Bikemap – це програма для планування маршруту та навігації.

Рисунок 1.1. – Мобільний додаток Bikemap

Bikemap – це програма для iPhone та Android, яка пропонує планування маршруту, навігацію, оновлення в реальному часі та багато іншого.

Оновлення програми в режимі реального часу дозволяють попереджати інших користувачів Bikemap про проблеми, які виникають під час поїздки. Це не те, що ми знайшли особливого застосування, але може бути більш привабливим для велосипедистів, які регулярно їздять у міському середовищі.

Інші функції включають архів із понад семи мільйонів створених користувачами маршрутів, колекцій маршрутів і статистики поїздок у програмі.

Більшість функцій Bikemap можна використовувати безкоштовно, але є також послуга Bikemap Premium, яка відкриває додаткові можливості

картографування, включаючи зручні для велосипедистів шари карти та 3D-види запланованих маршрутів, а також офлайн-навігацію.

Komoot (рисунок 1.2). Додаток Komoot пропонує багато інформації про ваш маршрут.

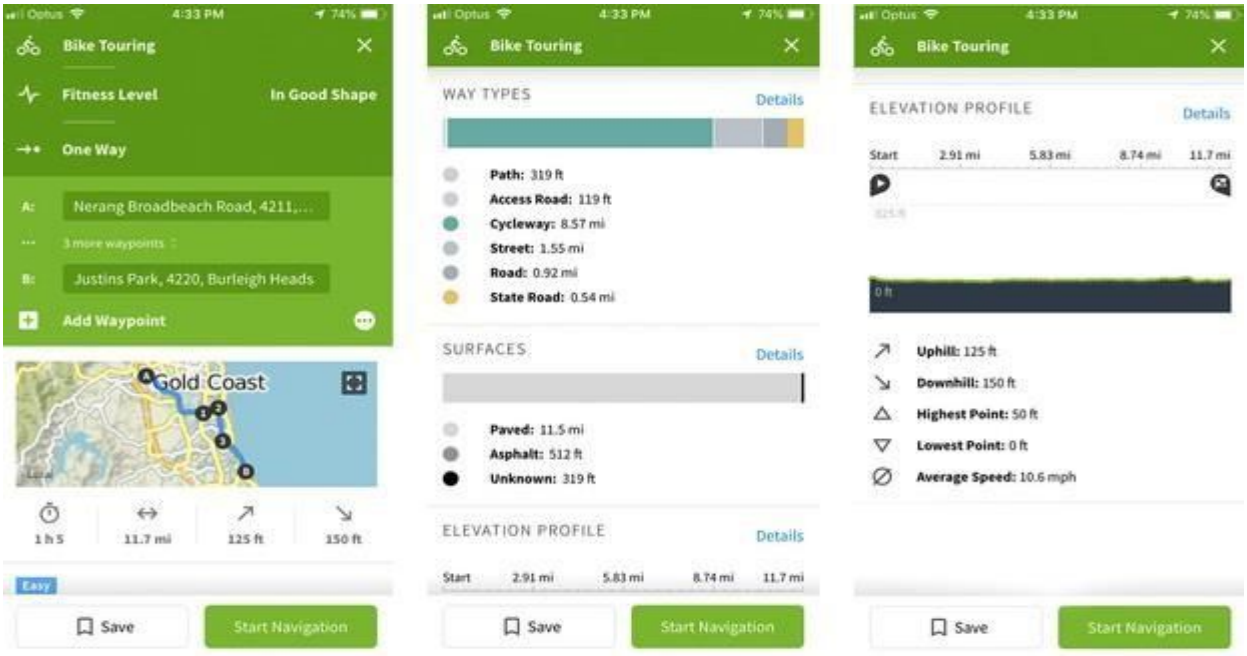

Рисунок 1.2. – Мобільний додаток Komoot

Хоча Карти Google є, мабуть, золотим стандартом для роботи з навігацією в цілому, іноді вони можуть бути дещо короткими для велосипедних маршрутів.

Komoot використовує базу даних OpenStreetMap із відкритим вихідним кодом і дозволяє планувати поїздки на шосе , гірському велосипеді, а також поїздки на роботу. Велика різниця з Google Maps полягає в маршруті, де Komoot намагається вибрати найефективніший маршрут, беручи до уваги те, наскільки дорога чи стежка зручні для велосипедистів, а також вашу фізичну форму.

Використовуючи початкову та кінцеву точки, Komoot повідомить вам складність, необхідну фізичну підготовку, дорожнє покриття, яке ви зустрінете, і профіль висоти для завантаження.

Коли ви почнете свій маршрут, він покаже вам швидкість, пройдену відстань, відстань, що залишилася, і дозволить легко змінювати маршрут на ходу. Ви також можете переглянути інші рекомендації маршрутів у вашому регіоні.

У Komoot також є підібрані основні моменти, запропоновані місцевими гонщиками. Це може бути чудовим способом виявити невідомі дорогоцінні камені у вашій місцевості.

Преміум-підписка відкриває додаткові функції, такі як багатоденний планувальник і відстеження в реальному часі.

Найкращі велопрограми для відстеження записуватимуть вашу поїздку та покажуть вашу швидкість, відстань, маршрут та інші показники, наприклад висоту.

Це, мабуть, найпопулярніші додатки для велотуризму, серед яких Strava лідирує (і пропонує багато інших функцій, включаючи сегменти, таблиці лідерів і планування маршруту).

Strava (рисунок 1.3.) Перевагою Strava є його соціальна складова. Багато райдерів використовують комп’ютер GPS для запису та завантаження поїздок у Strava, а потім за допомогою програми перевіряють, що роблять їхні друзі.

Рисунок 1.3. – Мобільний застосунок Strava

Усі поїздки, завантажені в Strava, забезпечують автоматичний рейтинг вашого часу на популярних ділянках доріг і стежок, відомих як «сегменти» мовою Strava, а також GPS-карту місця, де ви їхали.

Функція в режимі реального часу, яка повідомляє вам, наскільки швидко ви відстежуєте вибраний сегмент, наприклад місцевий складний підйом.

1.3 Специфікація вимог та вибір засобів та методу розробки мобільного застосунку

Після прийняття остаточного рішення про початок розробки мобільного застосунку з'являється питання щодо його архітектури, яку саме архітектуру реалізації програмного забезпечення вибрати, а також вибору мов програмування, необхідних фреймворків, програмних забезпечень і т.д.

Архітектура – це сукупність роботи компонентів у програмному забезпеченні в єдиному представлені. У розробці мобільного застосунку була обрано мікросервісна архітектура.

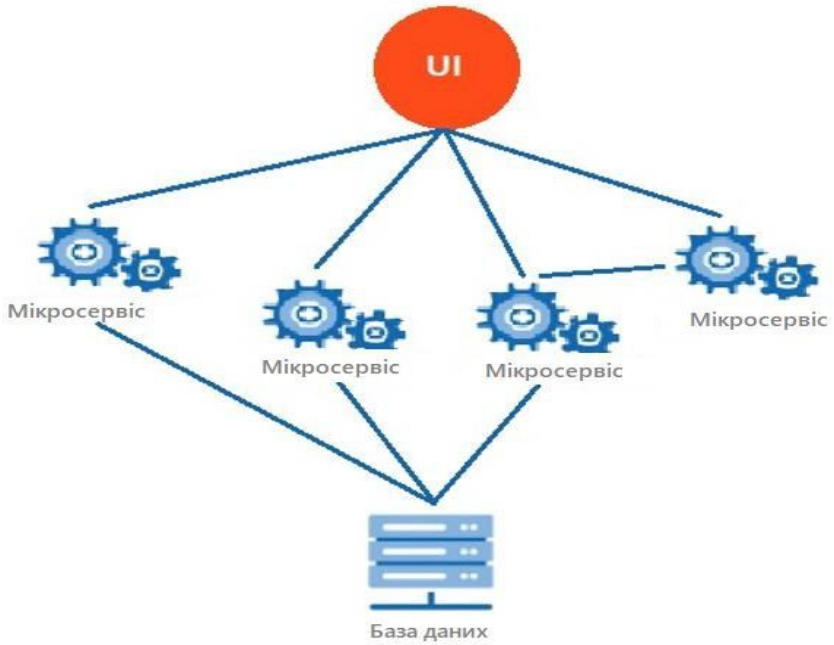

Рисунок 1.4 – Мікросервісна архітектура

Мікросервісна архітектура реалізується на основі невеликих, модульних сервісів, кожен з яких виконує своє унікальне завдання.

Дані модулі взаємодіють (спілкуються) за допомогою RESTAPI, з використанням формату JSON [6].

У свою чергу REST API (Representational State Transfer) - це загальні принципи взаємодії програми з сервером [7,9]. JSON – це формат для зберігання та обміну інформацією, в основному використовується для передачі даних між сервером та користувачем [8].

Основні плюси вибору мікросервісної архітектури:

Спрощує внесення доопрацювань/виправлень, тобто легше масштабувати додаток, оскільки сервіси не залежні.

Зручний при розробці декількома розробниками, знову ж таки незалежності сервісів.

При необхідності можливо однією дією вимкнути сервіс, що дозволяє знизити навантаження на сервер.

Безпека. Можна, можливо сміливо гарантувати, що передаватися буде тільки та інформація, яка запитується, оскільки спілкування відбувається за API.

Після того, як була обрано архітектура ПЗ, вже зрозуміло, що буде наше мобільний додаток ділитися на 3 частини: інтерфейсна частина (front-end), API (back-end), база даних.

В першу чергу, було прийнято рішення про вибір СУБД для подальшої роботи з базою даних.

На даний момент популярними СУБД є:

- Oracle.0
- MySQL.
- Microsoft SQL Server.
- Mongo DB.

Для того щоб вибрати конкретну СУБД нам необхідні критерії, на підставі чого і буде зроблено відповідний вибір:

База даних повинна бути реляційною, де дані організовані у вигляді різних таблиць, а таблиця складається з стовпців (полів) та рядків (записів).

СУБД надається на безкоштовній основі.

Від компанії, що супроводжують дані СУБД, повинна бути доступна коректна та актуальна документації.

Для зручності та наочності підходящих варіантів, виходячи з вищеописаних критеріїв, подамо відомості про СУБД у вигляді таблиці.

Таблиця 1.1. – Порівняння СУБД

Найменування СУБД	**1 критерій**	**2 критерій**	**3 критерій**
Oracle	+	-	+
MySQL	+	+	+
Microsoft SQL Server	+	-	+
Mongo DB	-	+	+

Виходячи з наведеною таблиці, робимо вибір на користь MySQL, так як ця СУБД відповідає всім заданим критеріям. MySQL - це система управління базами даних, що розповсюджується як вільне програмне забезпечення [9, 10, 13]. Додатково можна, можливо вказати, що дана СУБД показала надійність у роботі, можливість забезпечення безпеки даних, а також надає зручний інтерфейс для користувача.

Після вибору СУБД наступним кроком є реалізація інтерфейсу програмного забезпечення, а саме необхідно визначиться з вибором мови програмування або фреймворка та середовища програмування інтерфейсу. Для

того, щоб не писати інтерфейс двічі під різні платформи мобільних пристроїв, будемо розглядати виключно кросплатформні інструменти програмування.

Кросплатформність – це властивість сторінки, що дозволяє працювати відразу на кількох видах операційних систем чи платформ [10].

Популярні інструменти:

Apache Cordova – мобільне середовище розробки додатків. Дозволяє програмістам створювати програми для мобільних пристроїв за допомогою CSS3, HTML5 та JavaScript.

Це забезпечується за рахунок перетворення з CSS, HTML та JavaScript в код, який будь-яка платформа сприймає як елемент web. Випущений у 2012 році. Основним мінусом є ресурсомісткість, внаслідок чого додаток підгальмовує на слабких мобільних пристроях [11].

Flutter – середовище розробки з відкритим вихідним кодом для розробки мобільних додатків, випущений від компанії Google. Дане середовище було випущене відносно нещодавно у 2018, раніше мала назва Sky, та спеціалізована тільки під Android. На жаль, багато мінуси, проблеми були успадковані від попередніх версій [12]. На офіційному сайті Google, для цього фреймворка вказано три основні плюси:

Швидка розробка (властиво тільки для досвідченого розробника);

Виразний та гнучкий інтерфейс (фреймворк сам уміє налаштовувати шрифти, розташування та фізику скролів і т.д.);

Продуктивність (власний графічний двигун).

React Native – фреймворк для розробки мобільних додатків, реалізований на основі React (це інструмент для створення користувацьких веб-сторінок). Випущений був у 2015 році. React Native популярний так як компактний і відрізняється високою продуктивністю. Для написання коду використовується JavaScript. Особливу популярність додало використання великими компаніями, такими як: Facebook, Instagram, Skype, Tesla і т.д., що підтверджує його перспективу на майбутні роки.

Аналізуючи вищенаписане, робимо висновок, що Flutter не підходить, оскільки ще знаходиться на стадії розробки та вдосконалення. Також можна, можливо виключити Apache Cordova у зв'язку з низькою продуктивністю, відповідно вибір падає на React Native. На додаток до React Native можна, можливо сказати, що в нього відсутні звична html розмітка, присутні свої, вбудовані компоненти. Також присутній всім звична формальна мова опису зовнішнього виду компонентів (CSS). У свою чергу CSS дасть нам можливість налаштовувати компоненти повністю на наш бажанню.

Наступним і також важливим кроком є вибір мови програмування для back-end. Якщо у пошуковій системі знайти «рейтинг мов програмування backend», лідером буде PHP і на другому місці .Net, причому відрив буде колосальним, тому що на PHP реалізовано понад 79% відсотків всіх веб-сайтів [14]. Тому вибір очевидний у бік PHP.

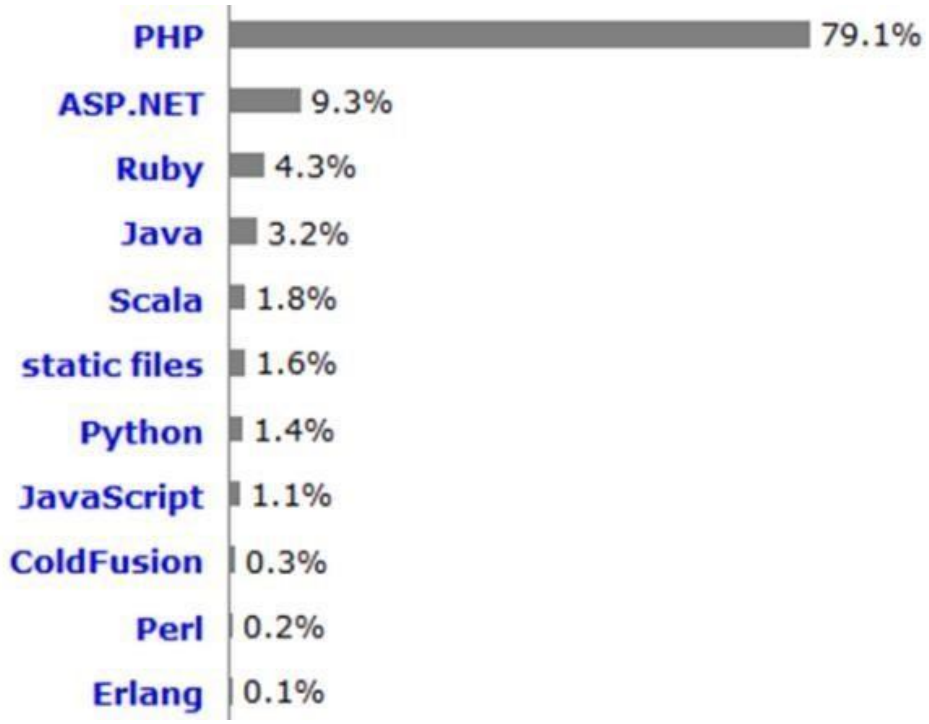

Рисунок 1.5. – Рейтинг мов програмування для back-end

PHP - це поширена мова програмування спільного призначення з відкритим вихідним кодом. PHP спеціально сконструйований для веб-розробок і його код може впроваджуватися безпосередньо у HTML.

Після вибору мов програмування необхідно визначиться зі середовищем розробки. На даний момент більше 50 програм для написання коду, причому частина їх спеціалізоване, тобто призначено для певної мови, а частина є універсальними. Розглянемо найбільш придатні для React Native та PHP.

Популярні редактори для React-Native:

Nuclide – спеціалізоване середовище розробки з відкритим кодом, представлене від компанії Facebook. Досить простий та зручний інтерфейс, представляє першокласне середовище розробки для проекту React Native, має вбудовану підтримку даного фреймворка. Більше не супроводжується, перебуває у фінальній версії. Відсутнє повноцінна підтримка Windows [16].

Visual Studio Code – універсальний редактор вихідного коду, представлений від компанії Microsoft. На поточний день супроводжується. У редакторі є функціонал із встановлення компонентів, який дозволяють розробляти на будь-якій мові програмування. Легкий і дуже приємний, а також інтуїтивний інтерфейс [8].

Очевидно, був обраний VS Code, який після встановлення додаткових компонентів для фреймворку React Native для оформлення синтаксиста коду дозволяє легко працювати в даному програмному забезпеченні.

Популярні редактори для мови програмування PHP:

NetBeans – вільна інтегроване середовище розробки програм. Підтримує багато мов програмування та велику кількість фреймворків. Перший випуск 1997 рік, на даний момент супровід ведеться розробниками на добровільній основі, але спонсорування йде від компанії Oracle [9].

PHPStorm – кросплатформне середовище розробки. Програмне забезпечення є комерційне, але для викладачів та студентів ВНЗ надається безкоштовно, також є 30-денний безкоштовний період експлуатації, випущено у

2009 році. Це програмне забезпечення реалізовано на основі іншого середовища розробки IntelliJ IDEA та спеціалізоване, виходячи з назви за мовою програмування PHP [10].

Додатковим та головним плюсом є Xdebug, що представляє собою розширення для PHP, призначений для налагодження коду.

Дані програмні забезпечення є рівносильними та рівноцінними, якихось жорстких критеріїв переваги на якийсь один додаток немає, можна сказати, кожен вибирає те, що йому більше подобається. Мій вибір був зроблений у бік середовища розробки PHPStorm, з додатковим розширенням Xdebug, яке скорочувало час на пошук помилок у рази, дозволяючи відловлювати помилки, через точку зупинки проходячи дебагом.

Крім того, щоб зрештою все запрацювало на локальній машині, необхідний також локальний сервер. Для Фреймворку React Native можливо додатково встановити компонент Expo.

Expo – це набір інструментів, які виконують складання програми на локальному порту. Далі за допомогою спеціального ПЗ для мобільного пристрою, є можливість в онлайн режимі працювати з кодом та одночасно бачити внесені зміни в нашому мобільному додатку.

Для розгортання на back-endi бази даних необхідний повноцінний сервер, можна розглянути найбільш популярні готові сервери Denwer та OpenServer, а також XAMPP.

XAMPP – майже голий локальний сервер, що містить Apache. Має величезнеу кількість додаткових розширень/бібліотек, що дає можливість запускати повноцінний сервер. Через його простоту, продуктивність даного сервера переважає над іншими, але новачкові підняти даний сервер без прочитання документації неможливо, але документація актуальна та докладна.

OpenServer – порівняно з іншими локальними серверами, цей сервер більший автоматизований, причому весь програмний комплекс має в рази більше

компонентів з його налаштування, а також має багатий набір допоміжного програмного забезпечення, які не вимагають інсталяцію.

Супровід відбувається до сьогодні. Підтримує актуальні операційні системи, актуальні версії мов програмування неймовірно простий та доброзичливий інтерфейс. Вміє автоматично налаштовувати системні налаштування комп'ютера під свої настроювальні дані. Основним, але не критичним мінусом є його тривалість запуску, а також рестарту при зміні конфігураційних файлів [15].

Збірка OpenServer включає в себе:

HTTP модулі: Apache.

Nginx.

СУБД модулі: MySQL.

PostgreSQL.

PHP модулі:

PHP версії 5.x.

PHP версії 7.x.

Додатковий набір інструментів:

HeidiSQL

Adminer

PHPMyAdmin

PHPPgAdmin

PgAdmin

Perl

FTP сервер

Sendmail

Memcachedсервер.

Звичайно ж, вибір очевидний і це буде OpenServer, оскільки він дозволяє більше гнучко налаштовувати сервер під себе, під свої потреби. На відміну від Denwer підтримує свіжі версії PHP, а на відміну від XAMPP підтримує актуальні

операційні системи. Додатково OpenServer легше розгорнути та підняти, а також має великий вибір що підключаються компонентів.

В результаті було прийнято рішення використовувати:

Для реалізації інтерфейсу (front-end):

- фреймворк - React Native;
- середовище розробки - Visual Studio Code; Для реалізації API (back-end):
- мова програмування PHP, Фреймворк Yii; - середовище розробки - PhpStorm.

Для реалізації бази даних:

СУБД - OpenServer.

1.4 Висновки до першого розділу

Здійснено опис предметної області, порівняльний аналіз відомих застосунків, виділено основні засоби для реалізації мобільного застосунку для управління організацією велопробігів.

РОЗДІЛ 2. ПРОЕКТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ

2.1 Архітектура мобільного застосунку для управління організацією велопробігів.

На рисунку 2.1 представлено діаграма варіантів використання для мобільного додатку для управління організацією велопробігів.

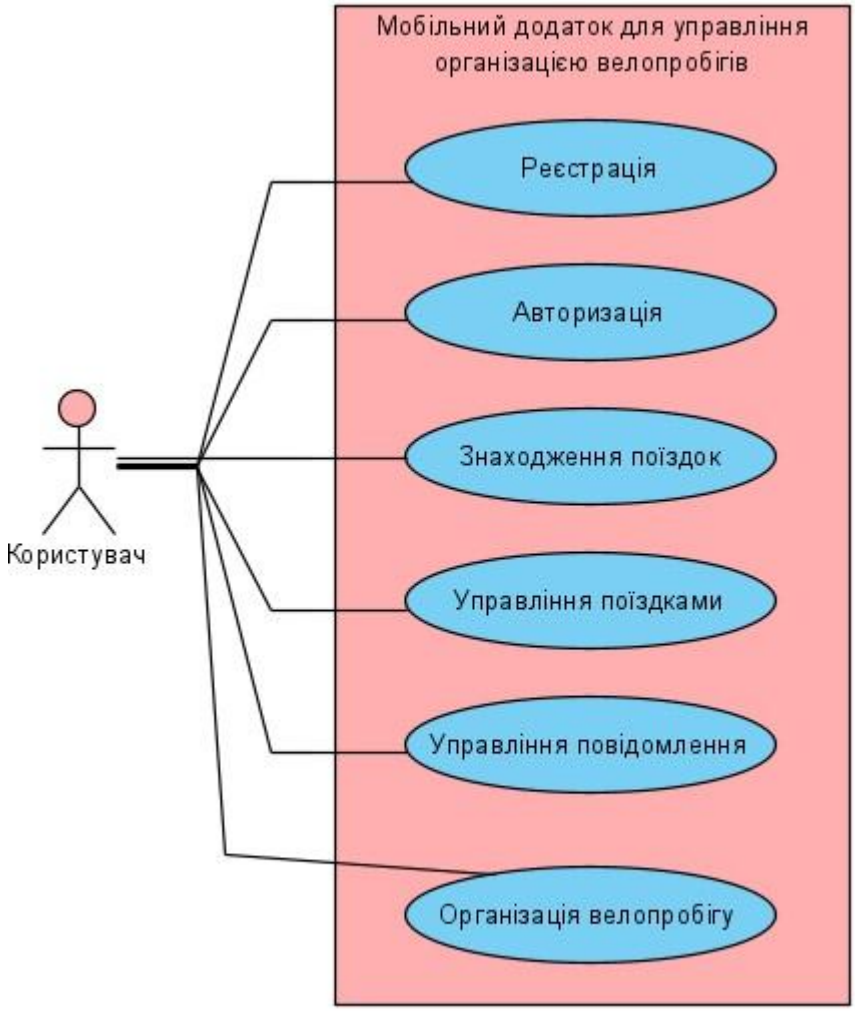

Рисунок2.1. – Діаграма варіантів використання

Перед початком розробки системи та для спрощення подальшої роботи складемо та розглянемо деякі UML діаграми з розробки та роботи майбутнього додатку.

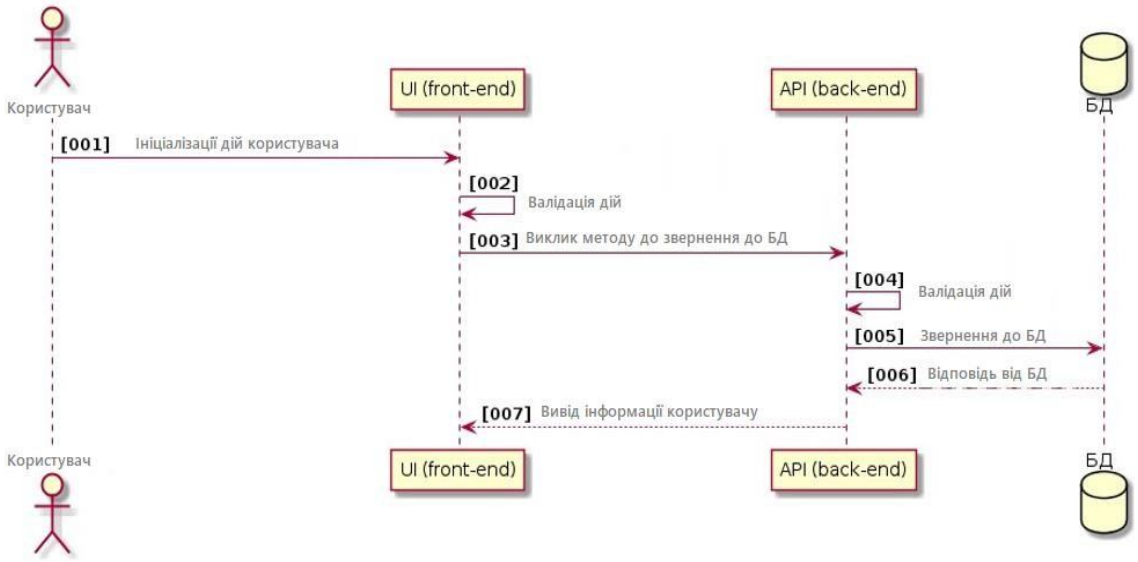

Рисунок 2.2. – Діаграма послідовності роботи програми

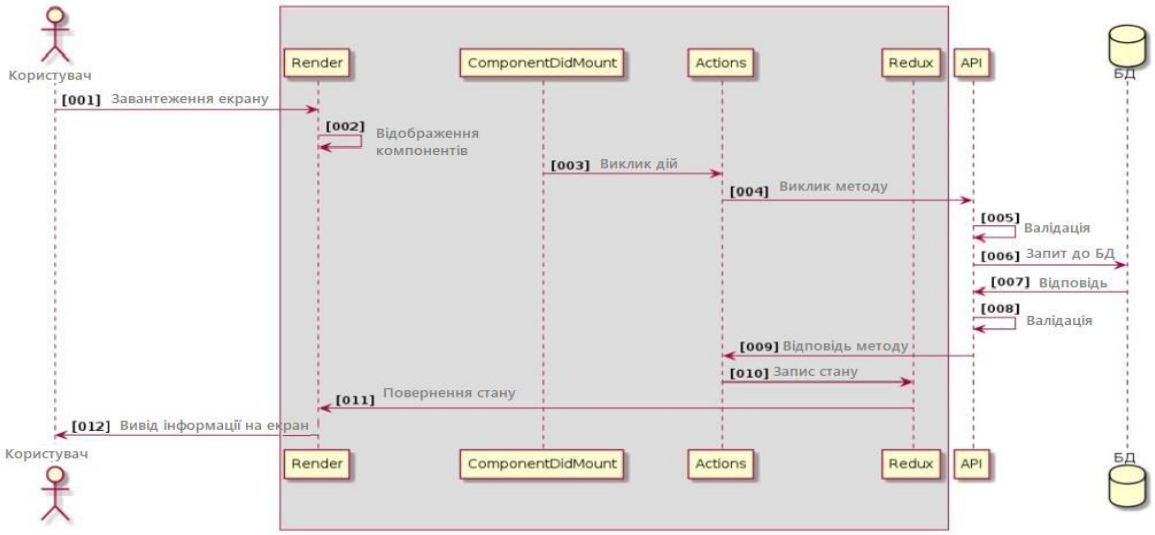

Рисунок 2.3. – Діаграма послідовності виводу інформації на екран програми

Як можна постерігати ініціатором всіх дій буде користувач, він працює з інтерфейсною частиною програми, далі інша частина програми викликає необхідну функціональність back-end, також за необхідності передаючи якісь параметри, а back-end вже проводить валідацію, якщо це потрібно та спілкується з базою даної. Аналогічно відбувається відповідь від усіх методів.

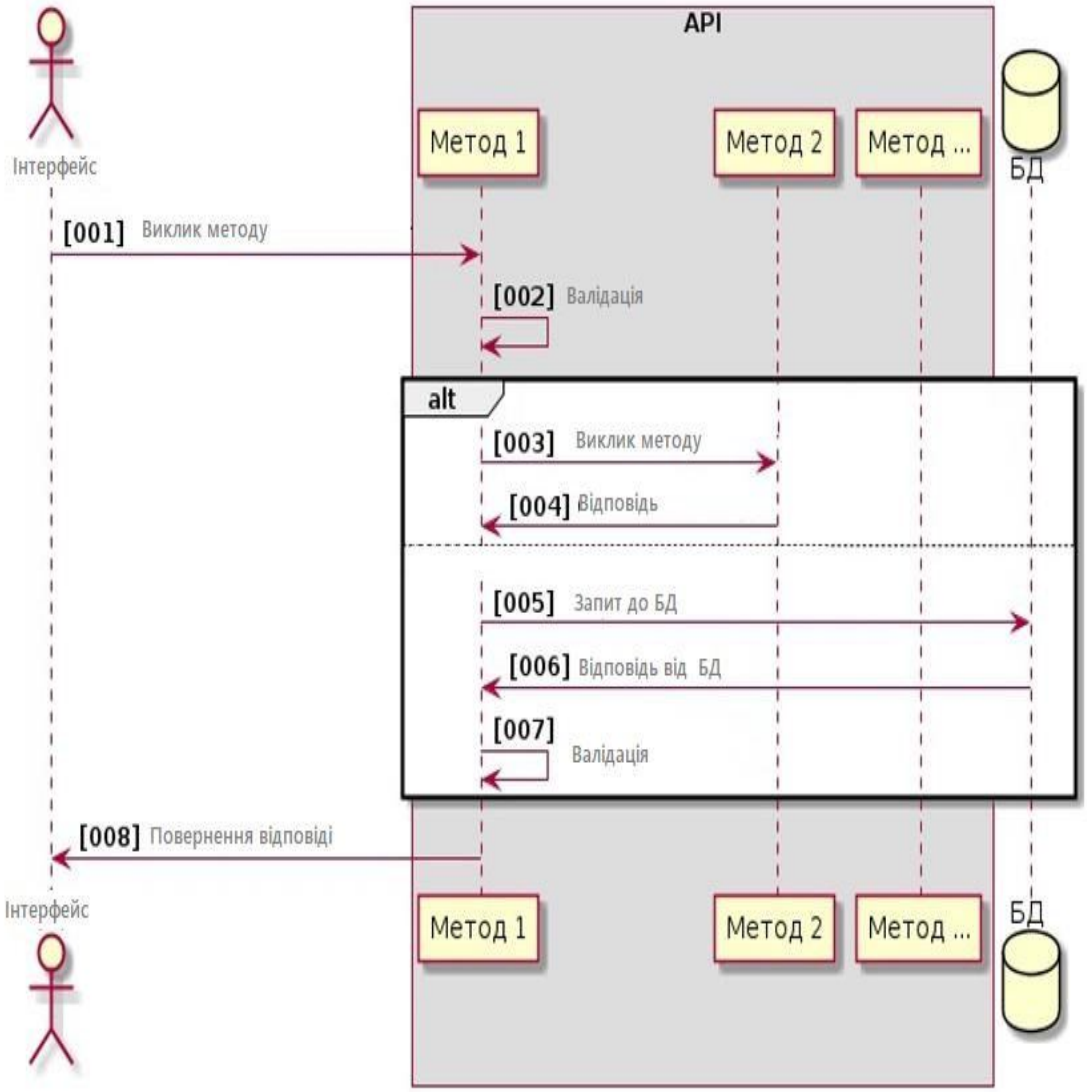

Рисунок 2.4. – Діаграма послідовності роботи методів

Завдяки даному підходу, ми визначилися з основними екранами ПЗ та з основним функціоналом, звичайно, дана схема не є кінцевою і буде розвиватися, декомпозуватися на більш дрібні.

2.2 Проектування бази даних.

Після розробки дизайну, ми вже розуміємо, яка інформація у нас буде відображатись у додатку, внаслідок чого приступаємо до розробки бази даних.

Проектування бази даних здійснювалося за допомогою онлайн- сервісу DBDesigner (адреса - https://app.dbdesigner.net/designer/schema/391000).

Даний сервіс безкоштовний та самим головним плюсом є те, що після закінчення можна, можливо цей проект експортувати у вигляді готового SQL запиту для створення бази даних, що дозволяє нам заощадити час.

Результат проектування бази даних представлено на рисунку 2.5, яка складається із шести таблиць:

User – інформація про користувачів.

Поля (тип даних):

Id (integer) - унікальне поле таблиці.

Username (varchar (100)) – найменування користувача.

Email (varchar (100)) – електронна пошта.

Phone (varchar (11)) – номер телефону.

Comment (varchar (250)) – додаткова інформація про себе.

Image (binary) - картинка-аватар.

BirthDate (date) - дата народження.

PasswordHash (varchar (max)) - хеш пароля.

CreatedAd (datetime) – дата реєстрації користувача.

CityId (integer) – населений пункт.

Cities – довідник населених пунктів.

Поля (тип даних):

Id (integer) - унікальне поле таблиці.

Name (varchar (255)) – найменування населеного пункту.

Trips – інформація про маршрути.

Id (integer) - унікальне поле таблиці.

UserId (integer) - id користувача, що є організатором.

FromCityId (integer) – населений пункт відправлення.

ToCityId (integer) – населений пункт прибуття.

Comment (varchar (max)) – додаткова інформація.

PlaceMeet (varchar (255)) – інформація про місцезнаходження зустрічі.

StartDateTime (datetime) – дата та час початку.

Deleted (boolean) - ознака актуальності.

Bookings – інформація про учасників.

Поля (тип даних):

Id (integer) - унікальне поле таблиці.

UserID (integer) - id користувача.

TripId (integer) - id маршруту.

Dialogues – діалоги користувачів.

Поля (тип даних):

Id (integer) - унікальне поле таблиці.

Code (varchar (max)) – унікальний код діалогу.

SenderId (integer) - id користувача відправника.

ReceiverId (integer) - id користувача одержувача.

LastMessage (varchar (max)) - останнє відправлене повідомлення.

LastMessageUserId (integer) – id користувача, що відправив останнє повідомлення.

Created (datetime) – дата створення діалогу.

Updated (datetime) – дата останнього відправленого повідомлення.

Messages – повідомлення діалогів.

Поля (тип даних):

Id (integer) - унікальне поле таблиці.

UserId (integer) - id користувача.

DialogueId (integer) - id діалогу.

Text (varchar (max)) - текст повідомлення.

Created (datetime) – час відправки повідомлення.

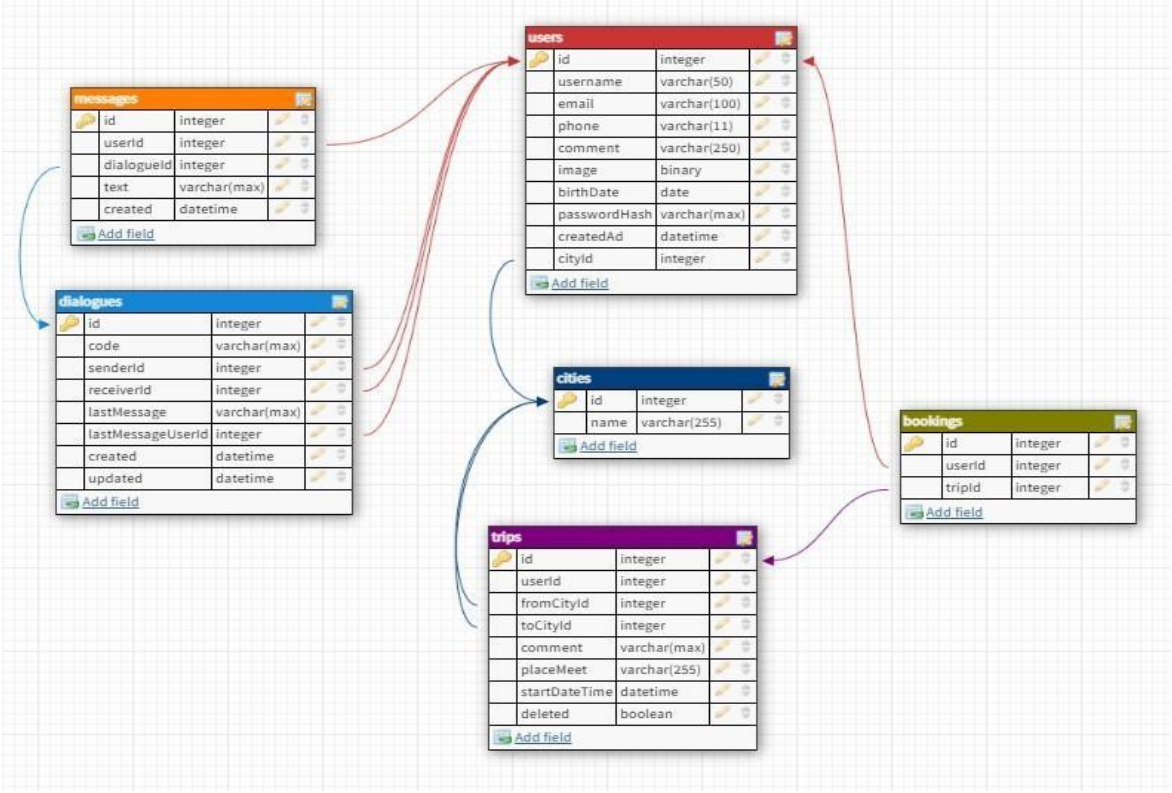

Рисунок 2.5. – Проект бази даних

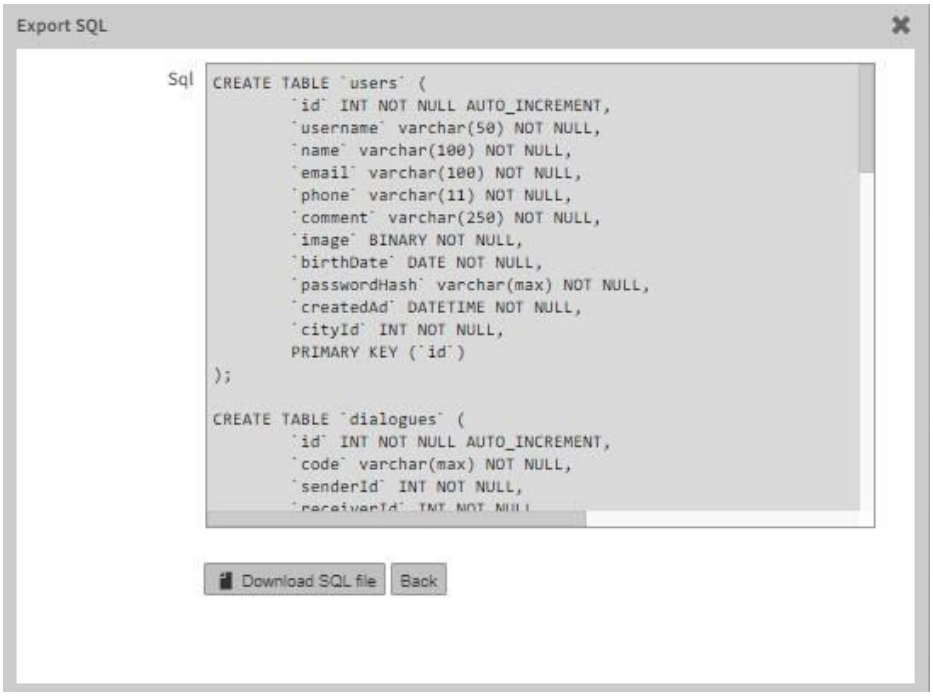

Рисунок 2.6. – Згенерований SQLзапит зі створення таблиць у БД.

Після генерації SQL запиту, його необхідно доопрацювати, додавши запит на створення бази даних. Відповідно залишається тільки виконати даний запит на нашому локальному сервері та з'явиться фізична готова база даних, яку залишається тільки наповнювати даними.

2.3 Розробка API.

Відповідно до прийнятого рішення, взаємодію з базою даних, реалізовуватимемо за допомогою API, кожен метод якого буде мати своє унікальне завдання, а звернення до них буде шляхом виклику з інтерфейсної частини програми або з іншого способу.

Для розробки API програми використовуємо Фреймворк «Yii» (Easy efficient extensible) призначений для швидкої розробка веб-додатків. Як середовище розробки буде PHPStrom. Дистрибутив програмного забезпечення

PHPStrom, можна взяти з офіційного сайту https://www.jetbrains.com/phpstorm.

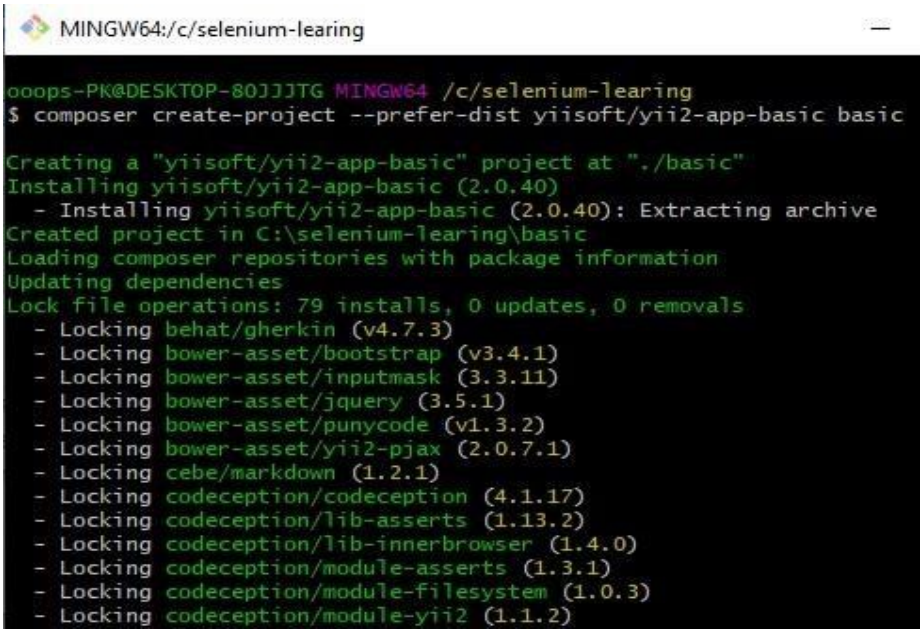

Рисунок 2.7. – Створення проекту на основі Фреймворка Yii

Для встановлення Фреймворку Yii необхідно встановити Composer.

Завантажити його можна з офіційного сайту https://getcomposer.org/download.

Далі згідно офіційною документації Фреймворка Yii рекомендується встановити шаблон, що має, замість створення проекту з нуля, виконаємо команду composer create-project -- prefer-dist yiisoft /yii2-app-basic basic».

Оскільки це шаблон проекту веб-сайту, в ньому присутня величезна кількість файлів, не потрібних нам. Видаляємо всі файли, залишаючи тільки папки: vendor, web, config та файли composer.json, composer.lock. Тепер ми маємо голий проект.

Додатково для коректної роботи сервера (згідно офіційною документації https://www.yiiframework.com/doc/guide/2.0/ua/start-installation), необхідно налаштувати конфігурацію Nginx (http-сервер, що приймає запити від користувача), який, як писалося раніше, вбудований в «openServer».

Після здійснення всіх налаштувань, створюємо підключення до бази даних. Для зручності параметри підключення до бази даних винесемо у файл env, який створимо в корені проекту, а далі викликатимемо через функцію «getenv ('найменування параметра ')».

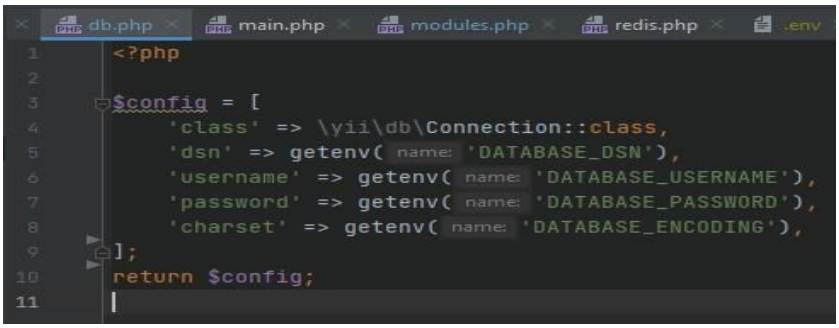

Рисунок 2.8. – Підключення до бази даних

Потім додатково створимо два файли конфігурації «request.php» та «response.php». У даних файлах необхідно налаштувати обробку форми запиту та відповіді. Для цього в "request.php" включимо синтаксичний аналіз вхідного JSON (потрібно під час виклику методу POST), а в файлі «response.php» реалізуємо формат повернення даних також у форматі JSON.

Рисунок 2.9. – Код файлу « request.php »

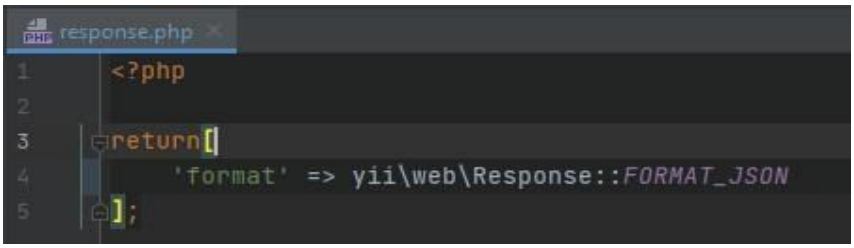

Рисунок 2.10. – Код файлу « response.php »

Як можна спостерігати на рисунку 2.10, додатково був додано параметр «baseUrl», який додасться на адресу методів (параметр не обов'язковий, робиться для наочності адреси API).

На наступному кроці перейдемо до створення методу отримання всіх населених пунктів з таблиці «cities» раніше створеної бази даних. Для цього в папці "app" створимо папку "modules". У папці modules створимо папку v1, що означає першу версію методів. Використання версій методів є рекомендованим, тому що в майбутньому при доопрацюванні методів замість зміни першої версії, краще додати другу версію, і тоді з'являється впевненість, що перша версія буде продовжувати працювати коректно.

У папці «v1» створимо ще підпапки "controllers", "repositories", "useCase". Також тут створимо файл із порожнім класом «module.php», необхідний для успадкування типу вбудованого функціонального класу.

```php
<?php

namespace app\modules\v1;
/**
 * Class Module
 * @package app\modules\v1
 */
class Module extends \yii\base\Module
{

}
```

Рисунок 2.11. – Код файлу « module.php »

Переходимо до реалізації методу отримання даних з бази даних. Починаємо із створення SQL запиту з пошуку інформації про населені пункти в базі даних. Для цього в папці "repositories" створимо файл

"CitiesRepository.php". Після оголосимо в ньому клас та за допомогою dbQuery перетворюємо стандартний sql запит на вибірку всіх даних "SELECT id, name FROM cities".

```php
<?php

namespace app\modules\v1\repositories;

use yii\db\Query as DbQuery;

/**
 * Class CitiesRepository
 * @package app\modules\current\repositories
 */
class CitiesRepository
{
    /**
     * @return array
     */
    public function getAll()
    {
        return (new DbQuery())
            ->from( tables: '{{%cities}}')
            ->select([
                'cities.id',
                'cities.name',
            ])
            ->all();
    }

}
```

Рисунок 2.12. – Пошук населених пунктів з бази даних

Після того, як був написаний метод отримання даних, переходимо до написання логіки запиту та обробки відповіді. Додамо файли в папці "useCase": "ListCitiesResponse.php " (в даному файлі просто вказана функція конструктора, що повертається) результату і «ListCitiesUseCase.php», в якому буде також вказаний конструктор та виклик раніше написаного методу отримання інформації щодо населених пунктів.

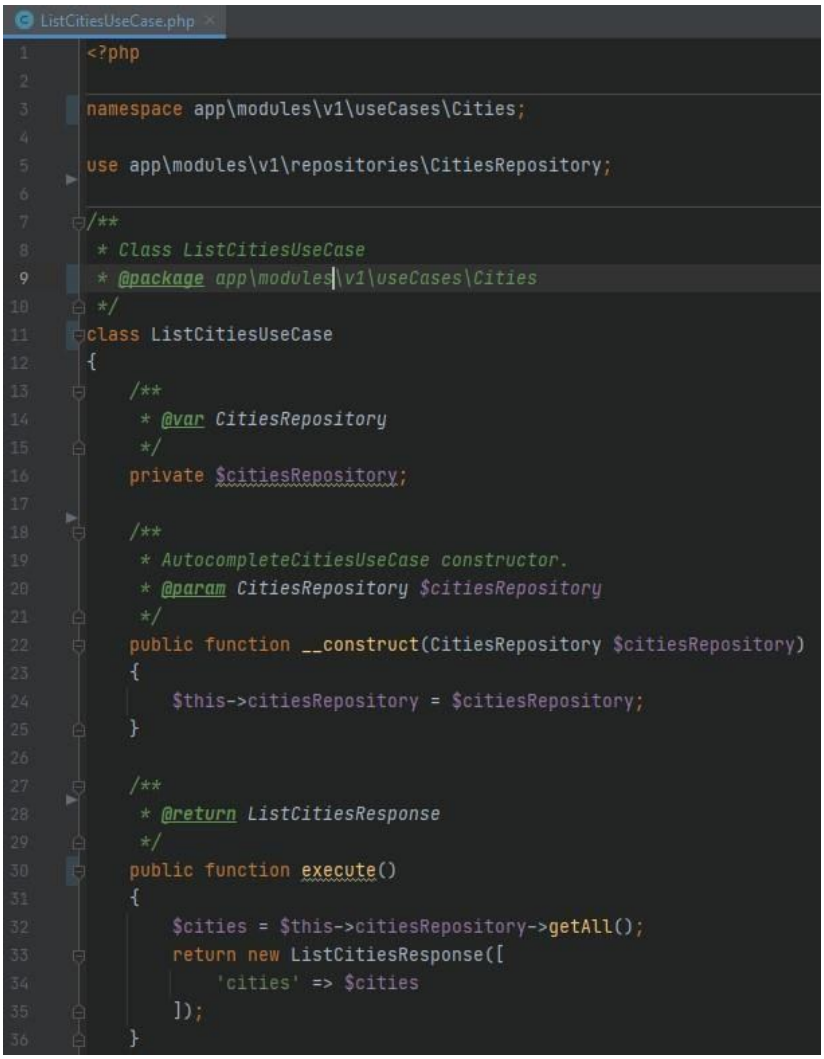

Рисунок 2.13. – Код файлу «ListCitiesUseCase.php»

Потім у папці "controllers" додамо файл "CitiesController.php". У цьому файлі зашита загальна логіка щодо прийняття запиту, його обробка та

поверненню результату. Оскільки вхідні параметри в даному методі відсутні буде лише результат.

Як можна, можливо побачити на рисунку 2.14 у нас спочатку викликається ініціалізація властивостей для створюваного об'єкта $ useCase.

Після викликаємо метод execute, раніше описаному файлі «List Cities Use Case. php », який у свою чергу викликає метод з репозиторія з отримання інформації безпосередньо з бази даних. Дана хитра логіка потрібна, для коректного перетворення повертається відповіді та легкості подальшого супроводу, таким чином, це дозволить уникнути дублювання коду.

Рисунок 2.14.– Основна логіка обробки методу отримання населених пунктів

Після написання методу, залишається тільки привласнити йому адресу.

Для цього повернемося в папку "config" і додамо файл "url-manager.php".

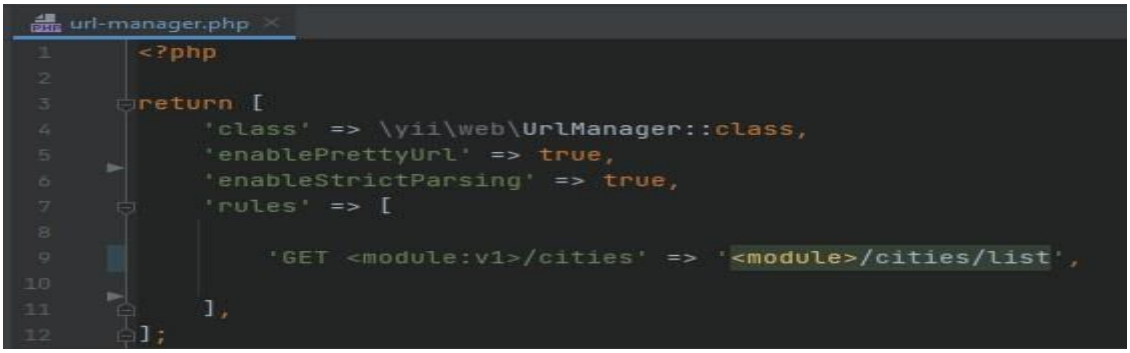

Рисунок 2.15. – url адреси методів

Обов'язково вказуємо додатково два параметри: «enablePrettyUrl» (включає використання гарних url адрес) і «enableStrictParsing» (включає суворий синтаксичний аналіз перевірки url адрес).

Рисунок 2.16. – Налаштування адреси версійності методів

У зв'язку з тим, що ми використовуємо підтримку версійності методів, в url вказуємо тег module та версію методу, а також додаємо файл «modules.php», в якому вказуємо повну адреса відповідності версій та методів.

2.4 Висновки до другого розділу

У даному розділі описано архітектуру розробленого мобільного застосунку, структуру бази даних та описано особливості реалізації API для взаємодії з базою даних.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ МОБІЛЬНОГО ЗАСТОСУНКУ

3.1 Програмна реалізація інтерфейсної частини мобільного застосунку

На цьому етапі приступаємо до програмної реалізації інтерфейсної частини мобільного застосунку. Першим кроком необхідно встановити програмне забезпечення, яке необхідні для програмування, тобто встановимо Visual Studio Code. Дистрибутив програмного забезпечення Visual Studio Code, можна взяти з офіційного сайту https://code.visualstudio.com.

Після встановлення Visual Studio Code, приступаємо до створення проекту. Для цього необхідно встановити набір інструментів Expo, виконавши через командний рядок наступне *npm install - global expo -cli*.

Далі, продовжуючи роботу в командній стрічці, виконаємо команду зі створення порожнього проекту (рисунок 3.1) *init FrontEnd - template = blank*.

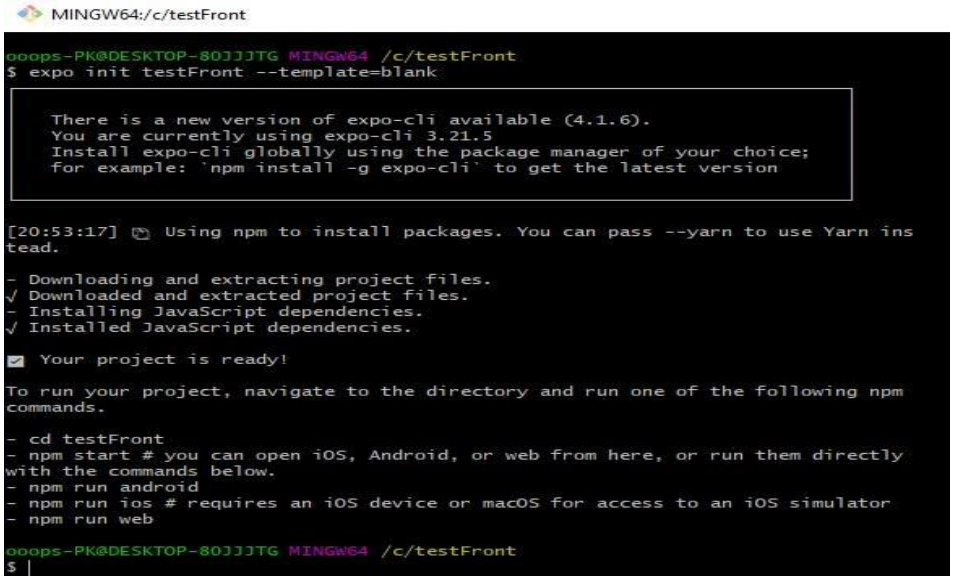

Рисунок 3.1. – Створення порожнього шаблону проекту

Відповідно після виконання даної команди, у нас створиться проект мобільного застосунку з порожнім шаблоном файлів, який вже зараз можна запустити та протестувати на телефоні.

Наступним кроком необхідно створенийц проект відкрити за допомогою Visual Studio Code, натисканням поєднання клавіш « Ctrrl +O», після обравши папку, раніше створеного проекту.

Далі створимо папку «screens» у нашому проекті, яка буде зберігати у собі всі екрани додатку. У ній створимо файл MyProfile.js, призначений для відображення інформації профілю користувача.

Важливо пам'ятати, що розробляти екрани програми будемо через класи, а не через стрілочні функції, так як при використанні асинхронних запитів до api, продуктивність буде краще. Тому створюємо шаблон екрану згідно рисунку

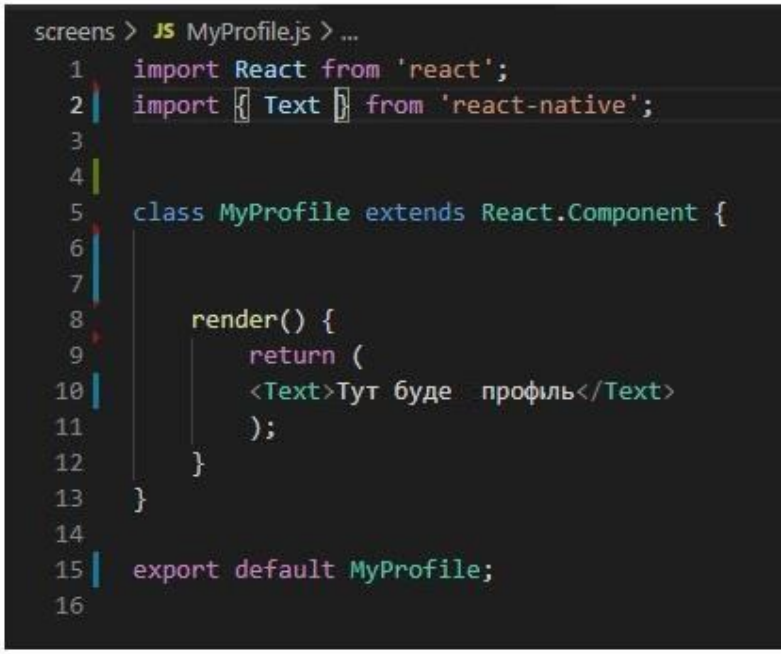

Рисунок 3.2. – Шаблон екрану профілю користувача

Подібним чином створимо ще три основні екрани («Мої маршрути», «Знайти маршрут», «Діалоги») користувача, змінюючи тільки найменування та виведений текст, для візуальної відмінності.

В нашому додатку буде не один екран, а кілька, для зручності налаштування навігаторів програми винесемо в окрему папку проекту "Navigators" (рисунок 3.3).

Рисунок 3.3. – Код навігаційної панелі

Після того, як будуть створені основні екрани програми, переходимо до створення навігації в мобільному додатку. Використовуватимемо офіційну бібліотеку React Navigation (документація - https://reactnavigation.org/docs). Для встановлення нижньої навігаційної панелі необхідно виконати всередині нашого проекту команду *«npm install @reactnavigation/bottom-tabs»*, в командній стрічці. Також встановимо «*npm install @react-navigation/stack*», призначений для компонування кількох екранів в один шлях.

Далі, додаємо файл "appNavigator.js" до папки "Navigators". У цьому файлі буде лежати налаштування по роботі та стилістиці нижної навігаційної панелі додатку. Згідно з рисунком 3.4, в основній частині програми у нас буде чотири екрана, для яких ми раніше створили порожні шаблони, тому в навігаційній панелі у нас також буде чотири кнопки. Тут же в налаштуваннях меню, робимо умову, що якщо екран обраний, то кнопку оформляємо в одному стилі, якщо не обраний, то в іншому.

Після, відразу зробимо шаблон стек-навігатора для створених екранів, тобто в цій же папці додамо файли: myTripNavigator.js (включає екран моїх маршрутів і мною вибраних), «tripNavigator.js» (включає екран всіх маршрутів), «dialoguesNavigator.js» (включає екран діалогів), «profileNavigator.js» (включає екран профілю користувача). А після ці стекнавігатори включимо у файл "appNavigator.js" (рисунок 3.4).

navigators > JS profileNavigator.js > ...

```
1   import React from 'react';
2   import {createStackNavigator} from 'react-navigation-stack';
3   import MyProfile from '../screens/MyProfile';
4
5   export const profileNavigator = createStackNavigator(
6       {
7           Profile: MyProfile,
8       },
9       {
10          initialRouteName: 'Profile',
11          defaultNavigationOptions: {
12              headerShown: false,
13          }
14      },
15  );
```

Рисунок 3.4. – Код «profileNavigator.js»

Тобто надалі, при створенні нових екранів, вони будуть додавати у відповідні стек-навігатори, наприклад, у стек-навігатори «profileNavigator», додасться екран «Редагування профілю». І при натисканні на кнопку назад,

перебуваючи на екрані «Редагування профілю», відкриватися завжди буде екран «Профіль».

У стек-навігаторі «dialoguesNavigator», додасться екран «Діалог з користувачем», при спробі повернутися назад, завжди відкриватиметься екран «Діалоги». За аналогією працюють інші стек-навігатори.

Далі потрібно змінити файл "app.js". Цей файл ініціює програму та виводить компоненти в заданій послідовності. Додатково пам'ятаємо, що в даному файлі не повинно бути логіки. Тобто в даному файлі необхідно зробити, щоб він виводив користувачеві раніше доданий нами файл "appNavigator.js", який викличе інший стек-навігатора, в якому буде вказано конкретний екран програми.

App.js > default

```
1   import React from 'react';
2   import {createAppContainer} from 'react-navigation';
3   import {rootNavigator} from './navigators/rootNavigators';
4
5   const AppContainer = createAppContainer(rootNavigator);
6
7   class App extends React.Component {
8       render() {
9           return (
10            <AppContainer/>
11          );
12      }
13  }
14
15  export default App;
```

Рисунок 3.5. – Код файлу «app.js»

Після підготовки основної програми, нам необхідно реалізувати авторизацію користувача та відповідно його реєстрацію. Тому за аналогією з минулим досвідом, додаємо два нових екрани.

На екрані «Авторизація» додаємо два поля для введення номера телефону та пароля, а також дві кнопки «Увійти» та «Реєстрація». Відповідно, при натисканні на кнопку «Реєстрація» повинен відкриватися однойменний екран, а при натисканні на кнопку «Увійти» повинен викликатися метод роботи з базою даних «auth/login». У разі успішної авторизації відкривається основна частина додатку, інакше виводяться помилки.

Перехід між екранами виконуємо за допомогою вбудованої функції onPress, вказуючи в ній функцію навігатора *"this.props.navigation. navigate("AuthRegister")"* (у дужках вказується найменування екрана).

Після реалізації авторизації та реєстрації, тепер при відкритті мобільного застосунку, нам потрібне чітке розуміння, чи авторизований користувач, для розуміння який екран виводити користувачеві. Якщо користувач не авторизований, виводимо екран «Авторизації», інакше відкриваємо основну частину програми.

Тому додамо ще один екран з найменуванням «InitialScreen.js», в якому перевірятимемо, чи є у користувача токен (який зберігається в пам'яті телефону і присвоюється йому в момент авторизації) чи ні. Якщо токен відсутній або некоректний, то відкриває стек-навігатор авторизації, інакше відкриваємо стекнавігатор основної частини програми (рисунок 3.6).

При ініціалізації мобільного застосунку, також додатково в асинхронному режимі необхідно запитувати довідник міст, при допомозі виклику методу "/cities". Для отримання даних будемо використовувати вбудовану функцію в React «fetch», яка може відправляти мережеві запити на сервер та підвантажувати необхідну нову інформацію в міру необхідності. Також для вилучення даних використовуємо функцію Redux, яка дозволяє керувати станом додатку.

```
async initialize() {

    try {
        let authToken = !!this.props.authToken;
        let currentUser = {};

        if (authToken) {
            global.defaultFetchHeaders['Authorization'] = 'Bearer ' + this.props.authToken;
            currentUser = this.props.authToken ? await this.getAccountCredentials() : {};
        }
        const screen = currentUser.id ? 'App' : 'Auth';
        setTimeout(() => this.props.navigation.replace(screen), 500);
        await this.props.citiesActions.getCities();
    } catch(error) {
        this.props.navigation.replace('Auth');
    }
}
```

Рисунок 3.6. – Перевірка наявності токена користувача

Для цього створимо нову папку Actions і в ній створимо файл cities.js. Далі створимо папку "Config", створимо два файли: "api.js" та "apiUrls.js". В api.js вкажемо дві константи, перша – версія api, друга - адреса звернення до api. У файлі apiUrls.js будемо складувати адреси методів по роботі з базою даних, в даному випадку додамо адресу методу для отримання довідників населених пунктів (рисунок 3.7).

```
config > JS apiUrls.js > ...

1  // Cities
2  export const LIST_CITIES_URL = '/cities';
3
```

Рисунок 3.7. – Метод видачі довідника населених пунктів

Після того, як були вказані змінні за зверненням до необхідного методу, повертаємо у файл «cities.js» та розробляємо виклик методу за допомогою вбудованої функції «fetch», яка може відправляти мережеві запити на сервер.

Далі, щоб працювати з отриманою інформацією в будь-якому місці скористаємося раніше викликаним методом Redux. Тому додамо папку

"reducer", а з нею файл "citiesReducer.js" (рисунок 3.8).

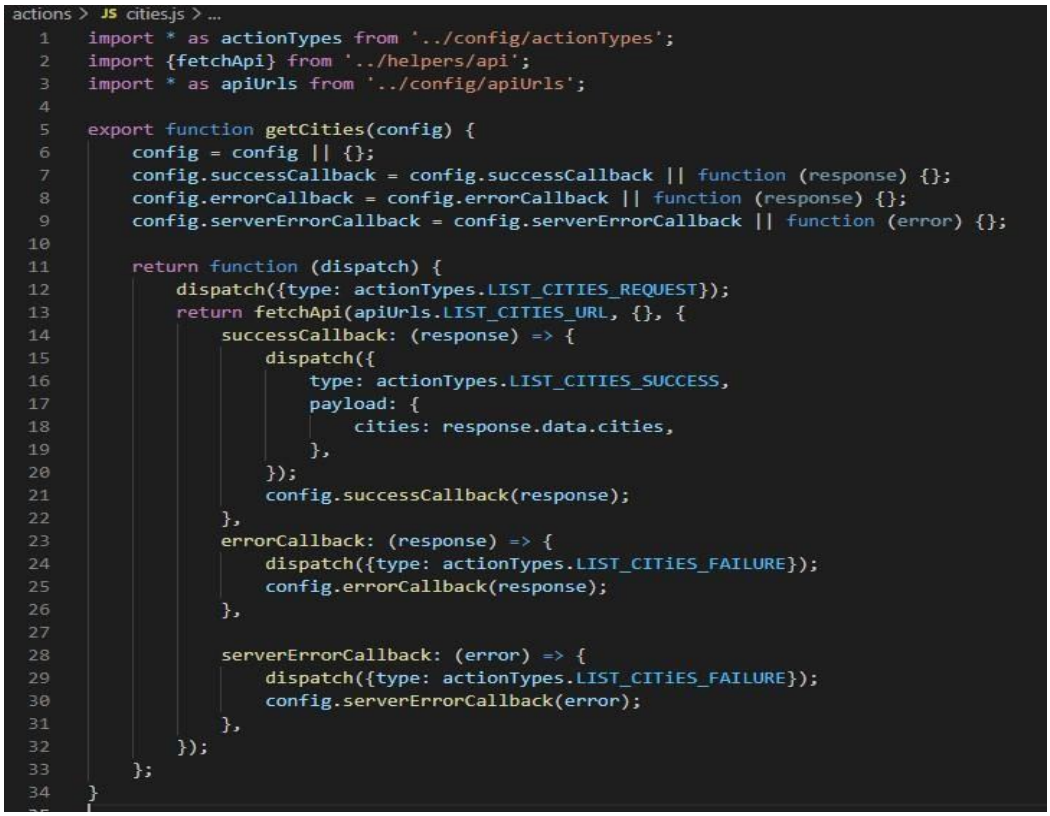

Рисунок 3.8. – Код дії отримання інформації з бази даних

У даному файлі будемо отримувати інформацію та записувати в стан отриману інформацію з якоїсь дії, в нашому прикладі з дії getCities (файл "cities.js"). У подальшому, якщо ця інформація потрібно, ми на будь-якому екрані можна легко її отримати, не викликаючи повторно метод роботи з базою даних.

Рисунок 3.9. – Отримання стану найменування населених пунктів.

Тобто алгоритм роботи з базою даних наступний:

1) Завантажуємо екран.

2) Викликаємо дію з папки «Actions», зав'язану на певний метод спілкування з базою даних.

3) Передаємо отримані дані «Redux » - записуємо у певний стан.

4) При необхідності виводимо на екран необхідні дані.

Повернемося на екран профілю користувача. Замість раніше створеного шаблону, додамо поля з виводу інформації з бази даних, таких як найменування користувача, дату народження, електронну пошту, номер телефону, а також отримання зображення користувача. Далі для заповнення полів профілю користувача, необхідно звернутися до методу «user/:id». Для цього за аналогією з дією getCities додаємо дію getUser, а також новий файл Redux.

Після у файлі екрана профілю користувача (рисунок 3.10) необхідно додати компонент «componentDidMount», який викликається відразу після рендеру екрану.

У цьому компоненті вказати виклик раніше доданої дії getUser для отримання даних про користувача, а також додати виклик стану, куди дія getUser запише отримані дані. І отже для виведення інформації на екран, достатньо її отримати з стану командою “this.props.' найменування стану '.' найменування поля».

```
class MyProfile extends React.Component {

    componentDidMount() {
        await this.props.accountActions.getUser({ id: this.props.currentUser.id });
    }

    render() {
        return (
            <View style={stylesCommon.safeContainer} forceInset={{ bottom: 'never' }}>
                <ScrollView showsVerticalScrollIndicator={false} scrollIndicatorInsets={{ right: 1 }} re
                        <View>
                            <View style={styles.container}>
                                <GroupItem>
                                    <FiledFullName>{this.props.user.user.name}</FiledFullName>
                                </GroupItem>
                            </View>
                            <View style={{ flex: 1 }}>
                                <GroupTextView>
                                        <GroupText>
                                            <GrayText>Дата</GrayText>
                                            <Text style={{ fontSize: 18 }}>{getGenitiveDate(this.props.u
                                        </GroupText>
                                        <GroupText>
                                            <GrayText></GrayText>
                                            <Text style={{ fontSize: 18 }}>{this.props.user.user.email}<
                                        </GroupText>
                                </GroupTextView>
                            </View>
                        </View>
                </ScrollView>
            </View>
        );
    }
}

const mapStateToProps = (state) => ({
    user: state.user.view.user
});
const mapDispatchToProps = (dispatch) => ({
    accountActions: bindActionCreators(accountActions, dispatch),
});
export default connect(mapStateToProps, mapDispatchToProps)(MyProfile);
```

Рисунок 3.10. – Код сторінки профілю користувача

За таким же принципом реалізуються інші екрани додатку.

3.2 Структура проекту

Структура проекту мобільного застосунку представлена на рисунку 3.11.

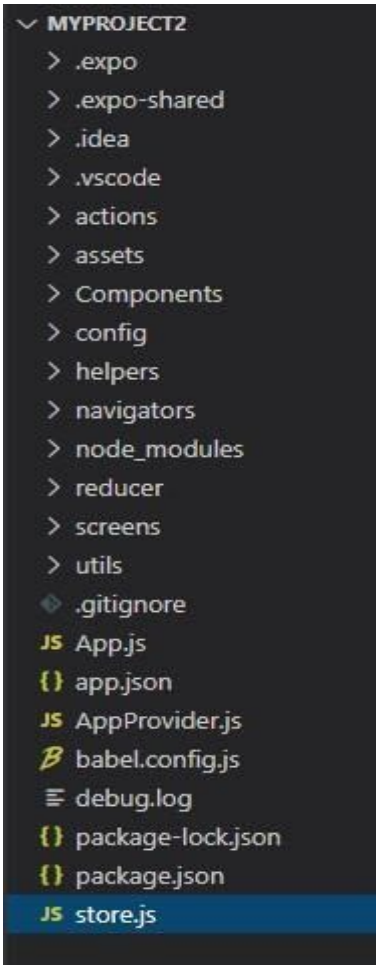

Рисунок 3.11. – Структура проекту інтерфейсу мобільного застосунку

Опис призначення основних папок проекту:

Actions – дії для запиту даних з бази даних.

Assets – зберігання стилів компонентів.

Components – опис компонентів, що використовуються на екранах.

Config - настроювальні файли, конфігурація проекту.

Helpers – додаткові функції для проекту.

Navigators – зберігання налаштувань навігації екранів додатку.

Reducer – функції зберігання інформації у стані.

Screens – екрани додатку.

Utils – одноразові функції, винесені з папки Helpers.

Опис екранів мобільного додатку:

AddTripScreen.js – екран із додаванням поїздки.

AuthLoginScreen.js – екран авторизації.

AuthRegisterScreen.js – екран реєстрації.

DialoguesScreen.js – екран діалогів.

EditMyProfileScreen.js – екран редагування профілю.

EditTripScreen.js – екран редагування поїздки.

HomeScreen.js – екран зі всім списком поїздок.

InitialScreen.js – завантажувальний екран при запуску програми визначає потрібно чи користувачеві авторизація.

MessagesScreen.js – екран діалогу з конкретним користувачем.

MyHomeScreen.js – екран з моїми поїздками.

MyProfileScreen.js – екран профілю поточного користувача.

TripScreeen.js – екран інформації про поїздку.

3.3 Опис проекту API

Структура проекту API мобільного застосунку представлена на рисунку 3.13. Опис призначення основних папок:

App: Components – містить настроювальні дані, наприклад інформацію статусу відповіді на запити, клас обертання у транзакцію запиту до бази даних за потреби і т.д.

Entities – містить файли з сутностями.

Exceptions – файли із класами обробки винятків методів, у разі непередбачуваної ситуації для виводу зрозумілої інформації на екран.

Filters - зберігати в собі єдиний файл з класом по роботі з токеном користувача, його перевірку, обробку і т.д.

Helpers – містить допоміжні класи, оскільки формалізація номера телефону, генерація унікального діалогу з користувачами.

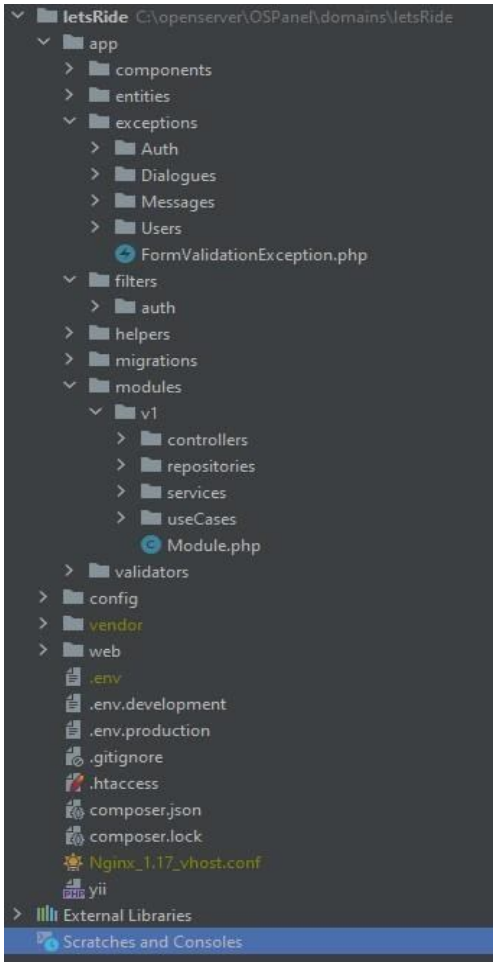

Рисунок 3.13. – Структура проекту API

Modules – містить множину файлів, пов'язаних з основною роботи методів та базою даної. Зберігати в собі перетворені на PHP SQL запити до бази даних для вибірки даних. А також додаткову перевірку, з більш глибокою складністю обробки даних, отриманих з бази даних.

Validators – клас для додаткової валідації номеру телефону.

Migrations – механізм відстеження змін у структурі бази даних.

Config - зберігання конфігураційних файлів проекту.

Vendor – зберігання залежностей Фреймворка Yii Web – зберігання кореневого файлу " index.php " проекту.

Реалізовано наступні методи:

/ auth / login (GET) - авторизація.

/ auth / logout (POST) – вихід з облікового запису.

/ auth / register (PSOT) – реєстрація, додавання користувача.

/ account (GET) – отримання інформації про користувача.

/ account (POST) – зміни даних профілю.

/ cities (GET) – отримання списку населених пунктів.

/ chat / dialogues (GET) – отримання діалогів.

/ chat / users / id / messages (GET) - отримання списку повідомлень діалогу.

/ chat / users / id / messages (POST) – створення повідомлення.

/ users / id (GET) – отримання інформації про користувача.

/ trips (GET) – отримання списку маршрутів.

/ trips / id (GET) – отримання інформації щодо певного маршруту.

/ trips / id / remove (POST) – видалення маршруту.

/ trips (POST) - створення маршруту.

/ trips / id (POST) – зміна маршруту.

/ account / bookings / create – приєднання до маршруту.

/ account / bookings / remove – скасування участі.

/ account / bookings / reject – відхилення учасника організатором.

/ account / routes-bookings (Get) – отримання учасників.

3.4 Зберігання вихідних даних та тестування застосунку

Для зберігання та контролю версійності проекту, використовуємо Git. Також за допомогою контролю версій у критичні моменти, можливо, повністю відкотити проект до попереднього стану в одну команду.

Кожен день вечора, в яких здійснювалася розробка, фіксуємо зміни за допомогою команди git commit, а потім командою git push виробляємо закачування внесених змін проекту на віддалений репозиторій (рисунок 3.15).

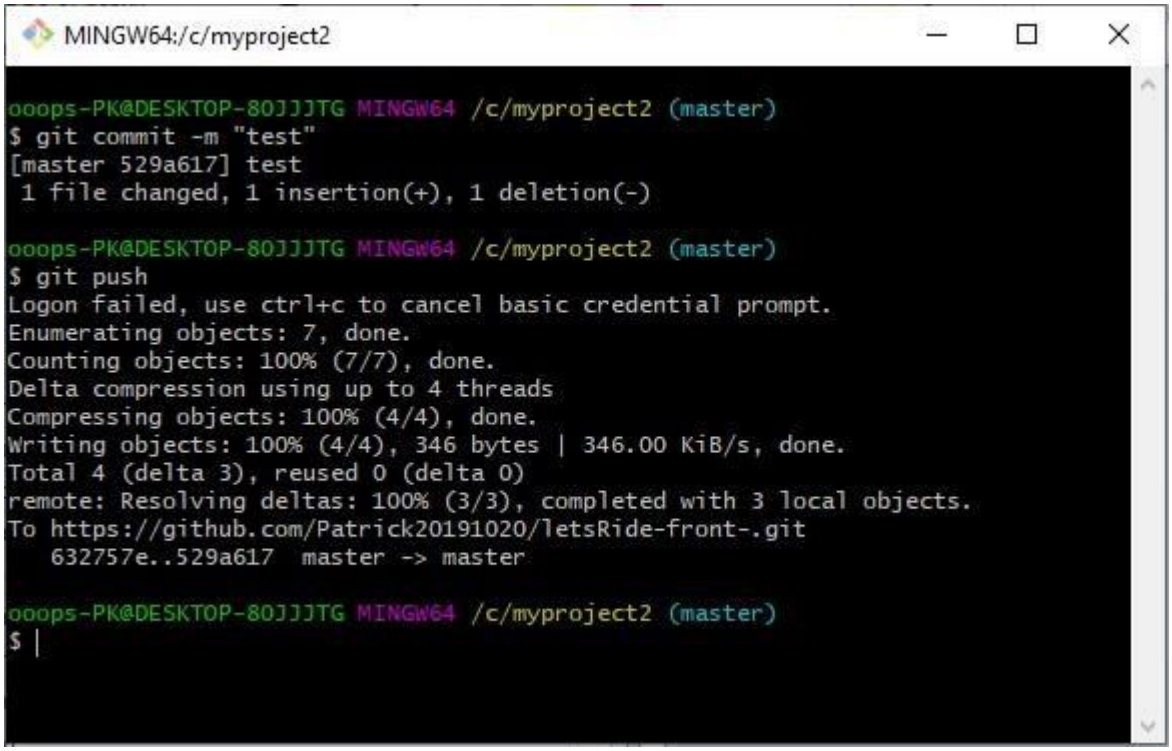

Рисунок 3.15. – Фіксація внесених змін до проекту на віддалений репозиторій.

Тестування методів звернення до бази даних здійснювалося за допомогою програмного забезпечення Postman. Окремо був протестований виклик кожного запиту, перевірена на коректність структури запиту та відповіді. Додатково в цей момент перевірялася навантаження на базу даних та наявність блокуючих запитів.

Результати:

1. Тестування завершено успішно, оброблено успішно 100% запитів.
2. Блокування на базі даних під час тесту не зафіксовано.
3. Утилізація апаратних ресурсів у межах норми.

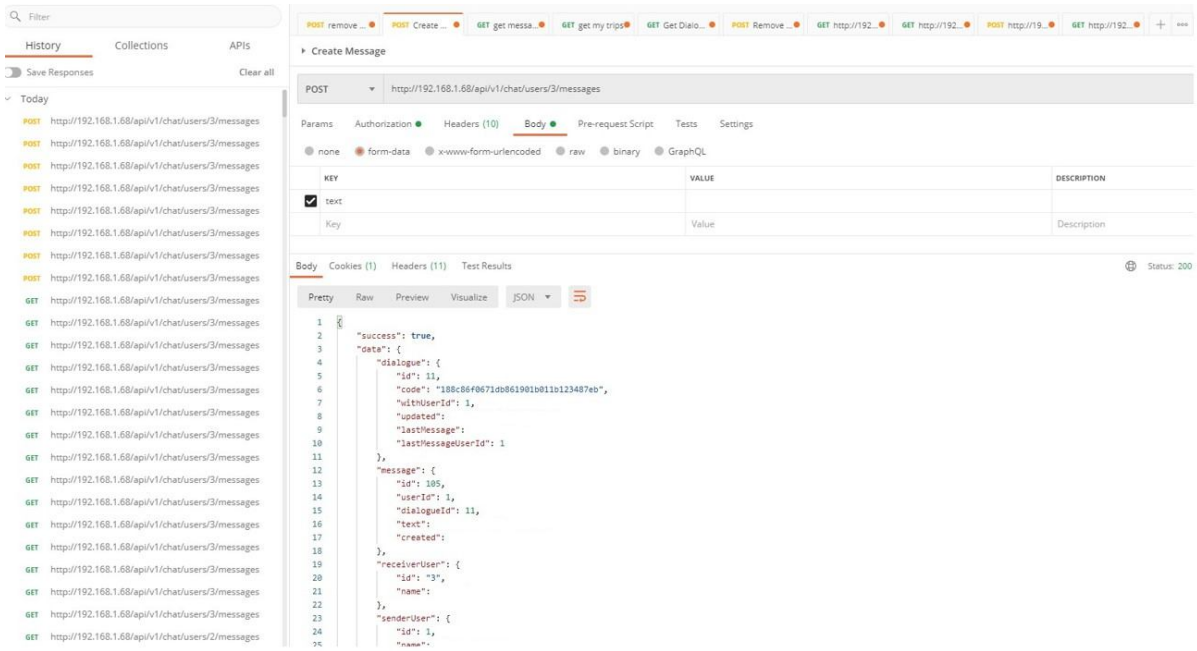

Рисунок 3.16. – Приклад запиту та відповіді у ПЗ Postman

Також було проведено навантажувальне тестування, відповідні результати представлено на рисунку 3.17.

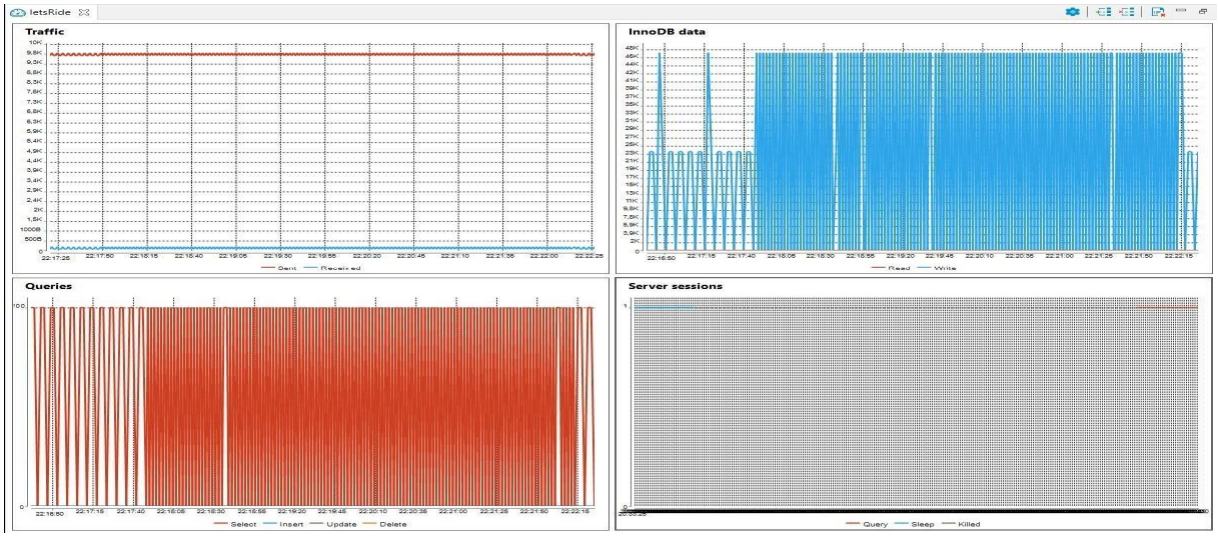

Рисунок 3.17. – Статистика навантаження на базу даних

Для забезпечення швидкодії роботи мобільного додатку необхідно було протестувати час та переіодичність блокування бази даних. На рисунку 3.18. представлено відповідні результати.

	Id	User	Host	db	Command	Time	State	Info
1	62	root	localhost:57169	[NULL]	Sleep	1		[NULL]
2	280	root	localhost:59765	letsride	Query	0	checking query cache for query	/* ApplicationName=DBeaver 7.3.2 - SQLEditor <Script-10.sql> */ show processlist

Рисунок 3.18. – Результат тестування блокування бази даних

Необхідно відзначити, що для усівх тестових випадків тестування було успішним.

3.5 Інструкція користувача мобільного додатку для управління організацією велопробігів

Перший екран, що зустрічає нового користувача, це екран авторизації (рисунок 3.19.).

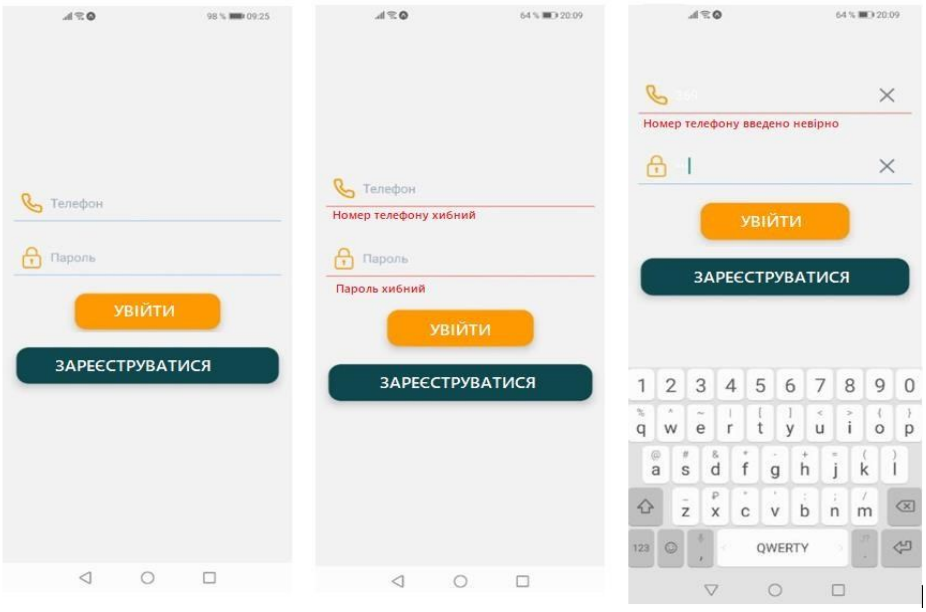

Рисунок 3.19. – Екран авторизації

Додатково налаштована валідація вхідних даних, при натисканні кнопки «Увійти». Кожне поле, що вводиться свій тип клавіатури, виходячи з необхідної

інформацією, наприклад для поля «Телефон» відкривається клавіатура виключно з цифрами "phone-pad", для введення пароля стандартна клавіатура.

Це правило передбачено для всіх подібних полів. Оскільки користувач новий, то необхідно натиснути кнопку «Зареєструватися» та перейти на екран реєстрації користувача.

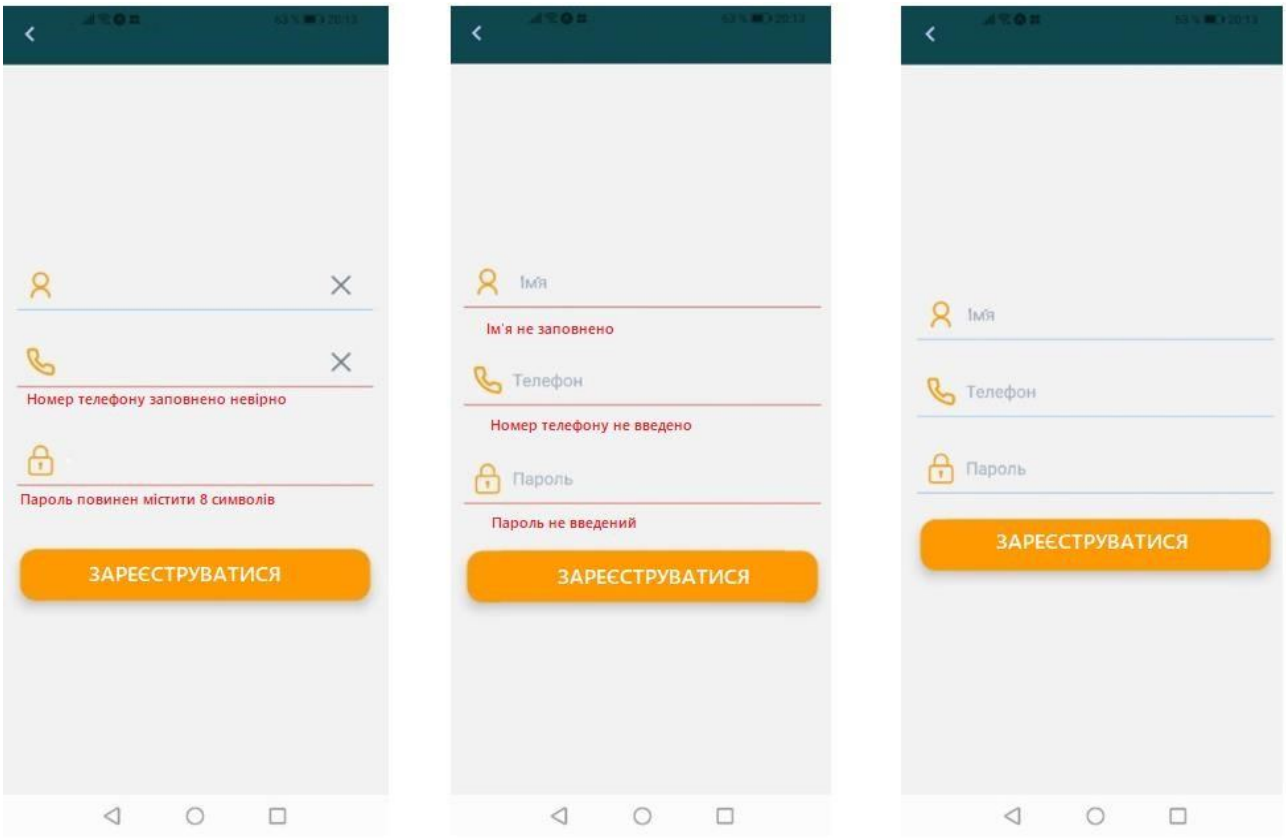

Рисунок 3.20. – Екран реєстрації користувачів.

Відмінною особливість є те, що з'явився заголовок та кнопка повернення на попередній екран. Після натискання на кнопку «Зареєструватися» та успішного виконання реєстрації, автоматично перекине на екран авторизації для входу.

Після проходження успішної авторизації, буде за умовчанням екран профілю поточного користувача.

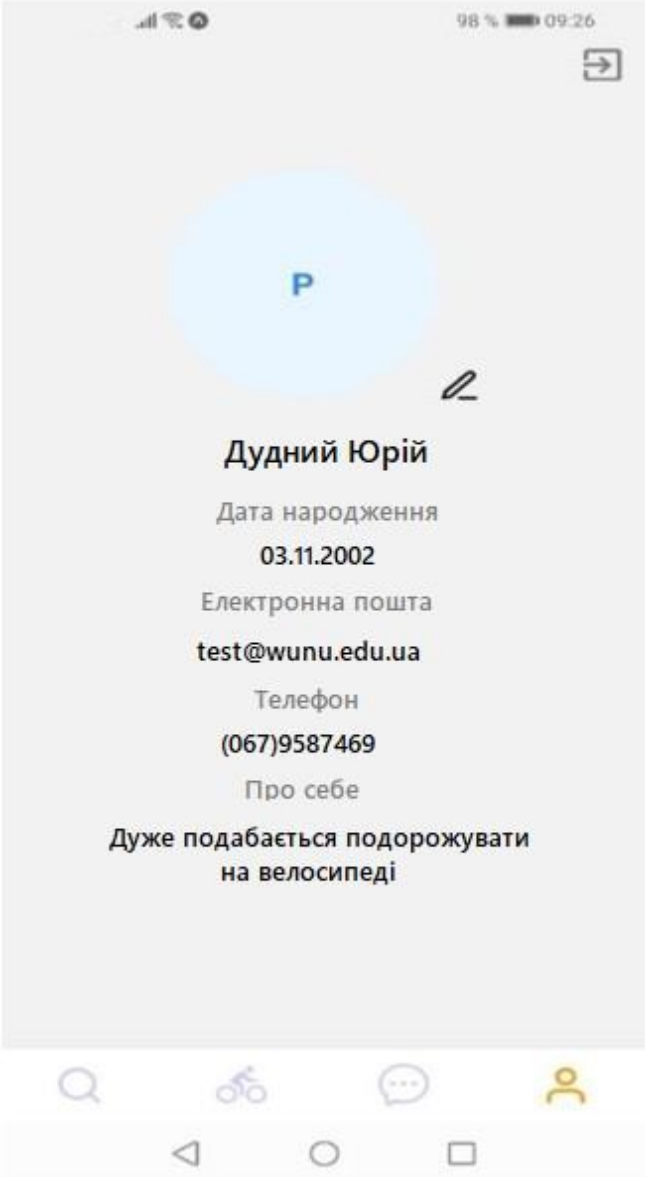

Рисунок 3.21. – Екран профілю користувача

На цьому екрані буде вказано основну інформація про користувача, яку він вніс під час реєстрації. Також тут є кнопка виходу з облікового запису та кнопка редагування профілю. При виході перекине на екран авторизації, а при натисканні на кнопку редагування, на екран редагування користувача.

На екрані можливо заповнити або змінити поточну інформацію про користувача, а також додати картинку, зробити це можливо двома шляхами, або відразу відкрити камеру і сфотографуватися або відкрити каталог фотографій

телефону і завантажити фото. Вся внесена інформація зберігається при натисканні на кнопку «Зберегти».

Не завжди є можливість зателефонувати людині, у додатку реалізований чат у реальному часу.

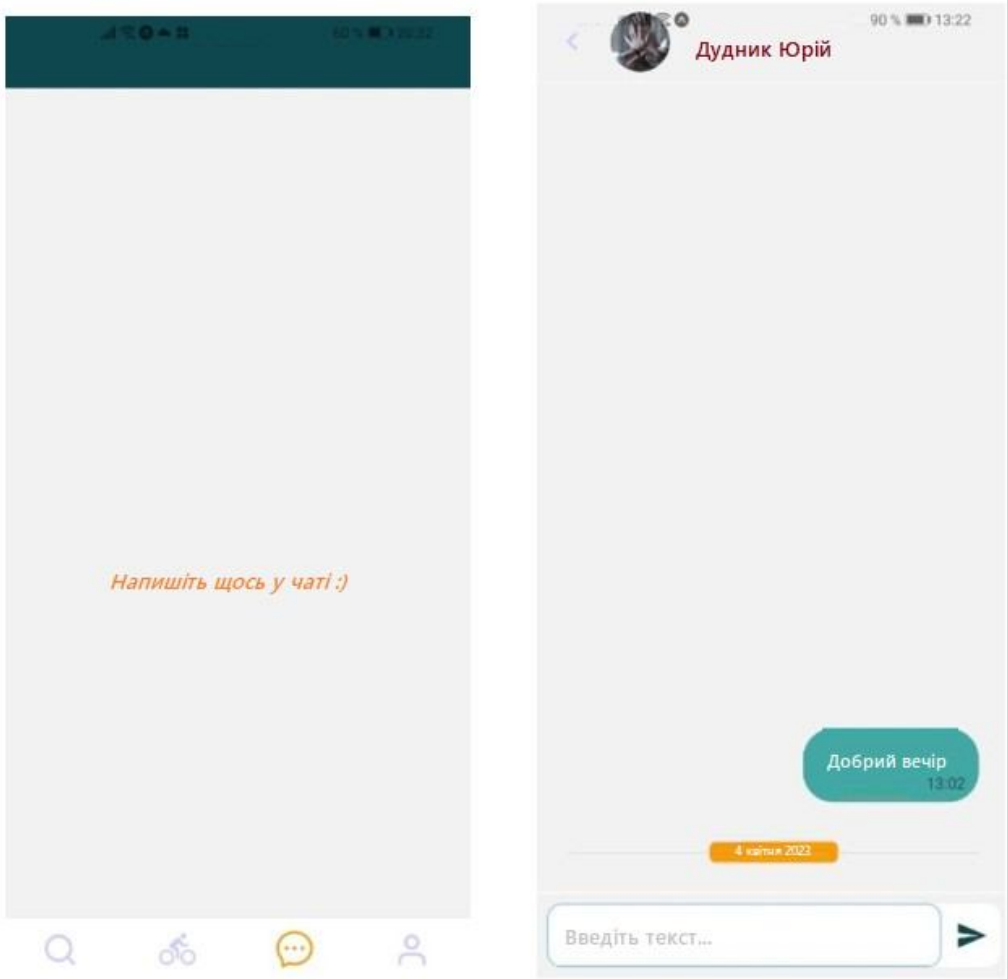

Рисунок 3.22. – Діалоги та чат з користувачем

На першому зображенні можна, можливо спостерігати вигляд екрана, за відсутності діалогів. Для відкриття чату, який зображено на другому зображенні, необхідно натиснути в будь-якому місці конкретного діалогу. На екрані чату з користувачем, для зручності було заховано навігаційне меню, а для повернення на попередній екран, необхідно скористатися кнопкою в заголовку.

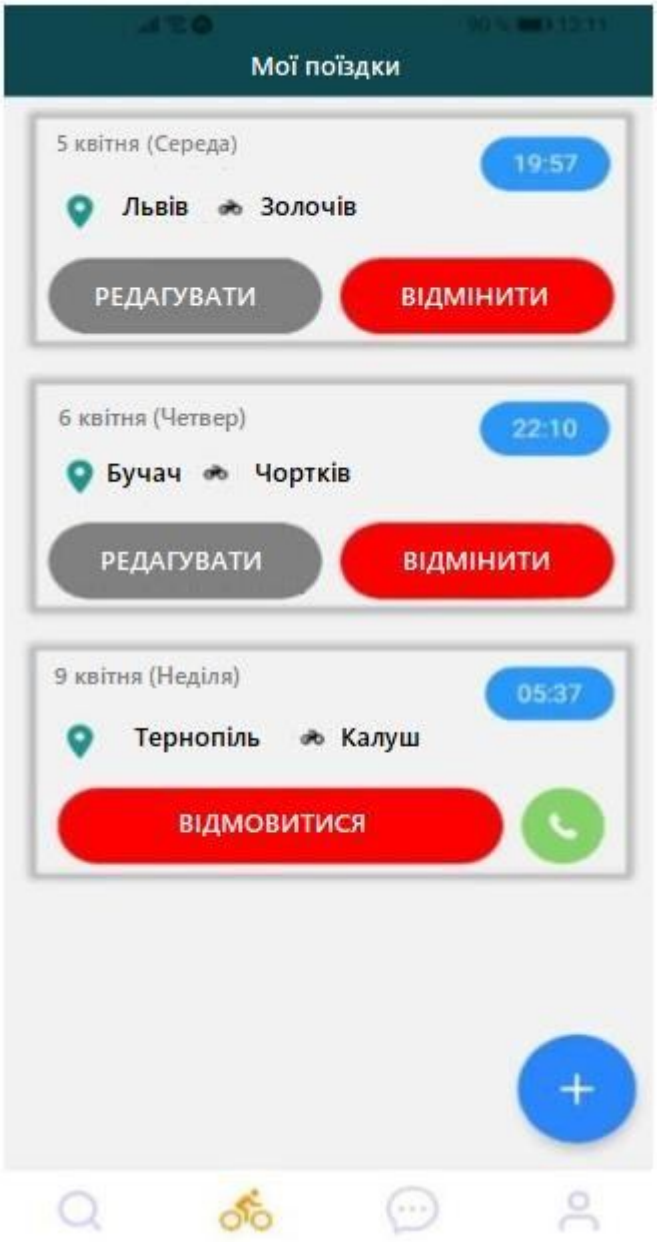

Рисунок 3.23. – Екран мої поїздки

На цьому екрані виводиться в блоковому вигляді інформація про поїздки, де поточний користувач організатор або учасник. І додаткова кнопка з додавання новою поїздки. Звідси з'являється можливість скасувати або відредагувати свої поїздки або відмовитися або зателефонувати організатору, де користувач виступає у ролі учасника.

Під час перегляду детальної інформації, різниця полягає не тільки у відображенні інформації, але у можливих виконуваних діях, можливо приєднатися, написати організатору, редагувати або скасувати, якщо поточний користувач організатор, або відмовитися, якщо учасник.

У режимі редагування, можливо, не тільки змінити інформацію, але також видалити тих, хто вже приєднався користувачів, далі для таких користувачів вона буде не видно. Збереження інформації викликається натисканням відповідної кнопки "Зберегти маршрут". Скасувати редагування можливо поверненням на попередній екран при натисканні кнопки в заголовку або при використанні навігаційної панелі.

3.6 Висновки до третього розділу

У даному розділі описано особливості програмної реалізації мобільного застосунку для управління організацією велопробігів, описано структуру проекту та структуру API методів.

Описано процес тестування мобільного застосунку та описано основні особливості роботи із Android додатком для проектування та генерації SQL коду.

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Інженерний захист персоналу об’єкту та населення. Правила застосування.

Функціонування на території нашої країни численних об’єктів підвищеної небезпеки, переважно в зонах з підвищеною концентрацією населення, різко посилює небезпеку великих техногенних катастроф, провокує та збільшує негативну дію особливо небезпечних стихійних явищ. Щороку втрати від таких надзвичайних ситуацій вимірюються тисячами людських життів, мільярдними збитками та невиправною шкодою для природного середовища.

Масштабність і багатогранність завдань щодо протидії сучасним природним і техногенним загрозам вимагають висококваліфікованої, технічно оснащеної, мобільної державної системи цивільного захисту.

Така система визнана складовою національної безпеки, а виконання її завдань - важливим обов’язком органів виконавчої влади всіх рівнів. Відповідно, з метою наближення до світових стандартів, від назви основного інструмента державної політики у сфері протидії наслідкам катастроф - цивільна оборона ми переходимо до назви - цивільний захист. І це не випадково. Сукупність завдань, що стоять перед службами цивільної оборони багатьох країн, більше пов’язані сьогодні з проблемами мирного часу, що дозволяє говорити про цивільний захист населення і територій, а не про цивільну оборону у воєнний час.

До захисних споруд цивільного захисту належать:

1) сховище – герметична споруда для захисту людей, в якій протягом певного часу створюються умови, що виключають вплив на них небезпечних факторів, які виникають внаслідок надзвичайної ситуації, воєнних (бойових) дій та терористичних актів;

2) протирадіаційне укриття – негерметична споруда для захисту людей, в якій створюються умови, що виключають вплив на них іонізуючого опромінення у разі радіоактивного забруднення місцевості;

3) швидкоспоруджувана захисна споруда цивільного захисту – захисна споруда, що зводиться із спеціальних конструкцій за короткий час для захисту людей від дії засобів ураження в особливий період.

Для захисту людей від деяких факторів небезпеки, що виникають внаслідок надзвичайних ситуацій у мирний час, та дії засобів ураження в особливий період також використовуються споруди подвійного призначення та найпростіші укриття.

Споруда подвійного призначення – це наземна або підземна споруда, що може бути використана за основним функціональним призначенням і для захисту населення.

Найпростіше укриття – це фортифікаційна споруда, цокольне або підвальне приміщення, що знижує комбіноване ураження людей від небезпечних наслідків надзвичайних ситуацій, а також від дії засобів ураження в особливий період.

За місцем розташування сховища можуть бути вбудовані і окремо розташовані. До вбудованих відносяться сховища, які розташовані в підвальних приміщеннях будинків, а до окремо розташованих – сховища, які розташовані за межами будинків і споруд.

Для вирішення питань щодо укриття населення в захисних спорудах цивільного захисту центральні органи виконавчої влади, місцеві державні адміністрації, органи місцевого самоврядування та суб’єкти господарювання завчасно створюють фонд таких споруд.

Порядок створення, утримання фонду захисних споруд цивільного захисту та ведення його обліку визначається Кабінетом Міністрів України. Проектування, будівництво, пристосування і розміщення захисних споруд та

об’єктів подвійного призначення здійснюються згідно з нормами, які розробляються відповідно до Закону України "Про будівельні норми".

Вимоги щодо утримання та експлуатації захисних споруд визначаються центральним органом виконавчої влади, який забезпечує формування та реалізує державну політику у сфері цивільного захисту.

Утримання захисних споруд цивільного захисту у готовності до використання за призначенням здійснюється суб’єктами господарювання, на балансі яких вони перебувають (у тому числі споруд, що не увійшли до їх статутних капіталів у процесі приватизації (корпоратизації), за рахунок власних коштів.

У разі використання однієї захисної споруди кількома суб’єктами господарювання вони беруть участь в утриманні споруди відповідно до укладених між ними договорів.

4.2 Запобігання наслідкам аварії на виробництвах із застосуванням аміаку.

Для отримання низьких температур технологічними схемами компресорного цеху багатьох промислових підприємств харчової та переробної промисловості передбачено застосування токсичної речовини – аміаку.

Потенційна небезпека таких технологічних схем полягає у порушенні герметичності обладнання і трубопроводів, що містять аміак. Найбільшу небезпеку з цієї точки зору являють собою руйнування автоцистерн з рідким аміаком; руйнування напірних трубопроводів компресорів; порушення герметичності відокремлювачів рідини, лінійних та циркуляційних ресиверів, запірної арматури, батарей холодильних камер.

Наслідком таких аварій є виникнення загазованості виробничого приміщення, відкритого майданчика цеху і підприємства в цілому, а також

прилеглих житлових районів; утворення вибухонебезпечної суміші аміаку з повітрям в приміщеннях, внаслідок чого можливі вибухи і пожежі.

Джерелами локальних викидів аміаку можуть служити процеси стиснення газоподібного і нагнітання рідкого аміаку, а також зливно-наливні операції.

Аварії (катастрофи) на підприємствах, транспорті та продуктопроводах можуть супроводжуватися викидом (виливом) в атмосферу і на прилеглу територію небезпечних хімічних речовин (НХР), таких як хлор, аміак, синильна кислота, фосген, сірчаний ангідрид та інші. Це являє серйозну небезпеку для населення, заражене повітря уражає органи дихання, а також очі, шкіру та інші органи.

Фактори небезпеки викиду (розливу) хімічно небезпечних речовин: забруднення навколишнього середовища, небезпека для всього живого, що опинилося на забрудненій місцевості (загибель людей, тварин, знищення посівів та ін.), крім того, внаслідок можливого хімічного вибуху виникнення сильних руйнувань на значній території.

Аміак – безбарвний газ з характерним різким запахом і їдким смаком. Він майже у два рази легший від повітря.

За звичайних умов аміак легко зріджується під тиском, а при випаровуванні поглинає тепло – сильно охолоджується. Ця властивість використовується у промислових та побутових холодильниках на м'ясокомбінатах, молокозаводах, овочевих базах, тобто там, де є необхідність в охолодженій продукції. Крім того, він є сировиною багатьох хімічних виробництв. Аміак зберігається і транспортується у зрідженому стані.

Він один з найважливіших продуктів сучасної хімічної промисловості. Головною галуззю його застосування є виробництво нітратної кислоти і азотних добрив. Крім того, аміак використовують для виробництва багатьох інших хімічних продуктів. Останнім часом зріджений аміак і водний розчин аміаку стали широко застосовувати безпосередньо як азотне добриво.

Як рідина, аміак легший за воду, має меншу густину і при виході на повітря утворює слабкий дим. Вогненебезпечний, створює вибухові суміші з повітрям, отруйний. Особливо небезпечний для очей.

У випадку розливу рідкого аміаку і його концентрованих розчинів не можна доторкатися до розлитої рідини.

Ознаки отруєння аміаком:

- нежить, кашель, важке дихання, задуха;

- підвищене серцебиття, порушена частота пульсу;

- при контакті з рідким аміаком виникає обмороження, можливий опік з пухирями, виразки.

Перша допомога при отруєнні аміаком:

- одягніть протигаз і виведіть ураженого на свіже повітря;

- дайте подихати зволоженим повітрям (теплими водяними парами 10%-ного розчину ментолу в хлороформі);

- дайте йому теплого молока з «Боржомі» або харчовою содою;

- при задусі необхідний кисень;

- при спазмі голосових щілин забезпечте тепло на ділянку шиї, теплі ванночки, інгаляцію;

- при зупинці дихання проведіть серцево-легеневу реанімацію;

- при потраплянні в очі − промийте водою або 0,5-1%-ним розчином квасців, вазеліновою або оливковою олією;

- при уражені шкіри − обмийте чистою водою, зробіть примочки з 5%-ного розчину оцтової, лимонної або соляної кислоти.

4.3 Висновок до четвертого розділу

Захисні споруди цивільного захисту можуть використовуватися у мирний час для господарських, культурних і побутових потреб у порядку, що визначається Кабінетом Міністрів України. З моменту виключення захисної

споруди із фонду споруд цивільного захисту вона втрачає статус захисної споруди цивільного захисту.

Володіння, користування та розпорядження спорудами, які втратили статус захисних споруд цивільного захисту, здійснюється відповідно до закону. Захисні споруди цивільного захисту державної та комунальної власності не підлягають приватизації (відчуженню).

При отруєнні аміаком винести потерпілого із зони зараження, шкіру, рот, ніс промити водою. В очі закапати по дві-три краплі 30% альбуциду, в ніс - оливкове масло. При необхідності відправити потерпілого до медичного закладу.

ВИСНОВКИ

У процесі розробки мобільного застосунку та виконання даної кваліфікаційної роботи реалізовано:

1. Проведено аналіз стану проблеми та виявлено підходи її рішення. Була підтверджено актуальність розроблюваного мобільного застосунку.

2. Здійснено аналіз та обґрунтований вибір технологій реалізації та програмних платформ. Нами вибрано: мову програмування PHP, Фреймворк React Native, Фреймворк Yii, mySql, openServer, Ide Visual Studio Code, IdePhpStorm, сховище вихідних кодів проекту та контроль версій Git, операційна система Windows 10, Postman.

3. Розроблений мобільний застосунок на підставі проведеного аналізу вимог.

Цей застосунок дозволяє виступати у ролі організатора спільних велопробігів або бути учасником, вже створеного. Також присутня можливість оперативної комунікації з організатором маршруту через онлайн чат. Може бути використано будь-якою людиною, за наявності смартфона та встановленої на ньому, інтерфейсної частини мобільного застосунку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Android (operating system) [Електронний ресурс] // Wikipedia – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Android_(operating_system).

2. Створення графічного інтерфейсу: [Електронний ресурс] – Режим доступу до ресурсу: https://metanit.com/java/android/1.3.php

3. Створення Firebase: [Електронний ресурс] – Режим доступу до ресурсу: https://firebase.google.com/

4. Діаграма варіантів використання : [Електронний ресурс] – Режим доступу до ресурсу: http://khpiiip.mipk.kharkiv.edu/library/case/leon/gl4/gl4.html

5. Основи UML: [Електронний ресурс] – Режим доступу до ресурсу: https://pro-prof.com/archives/2594

6. Application Sandbox [Електронний ресурс] // Source Android – Режим доступу до ресурсу: https://source.android.com/security/app-sandbox.

7. Data and file storage overview [Електронний ресурс] // Android Developer – Режим доступу до ресурсу: https://developer.android.com/guide/topics/data/data-storage#filesInternal.

8. Man-in-the-Disk: Android Apps Exposed via External Storage [Електронний ресурс] // CHECK POINT RESEARCH. . – Режим доступу до ресурсу: https://research.checkpoint.com/androids-man-in-the-disk/.

9. Man-in-the-Disk: A New Attack Surface for Android Apps [Електронний ресурс] // Check Point Blog. – Режим доступу до ресурсу: 71 https://blog.checkpoint.com/2018/08/12/man-in-the-disk-a-new-attacksurface-for-android-apps/.

10. DEF CON 26 - Slava Makkaveev - Man In The Disk [Електронний ресурс] // DEFCONConference YouTube channel. – Режим доступу до ресурсу: https://www.youtube.com/watch?v=vvfs0u1or3M.

11. Конспект лекцій з дисципліни «Програмування для мобільних пристроїв» для студентів денної форми навчання спеціальності 126

«Інформаційні системи та технології» / Укладачі: Готович В.А., Михайлович Т.В. – Тернопіль : Тернопільський національний технічний університет імені Івана Пулюя, 2020. – 216 с.

12. Fortnite Installer downloads are vulnerable to hijacking [Електронний ресурс] // Google Issue Tracker. – 2018. – Режим доступу до ресурсу: https://issuetracker.google.com/issues/112630336

13. MacOS X GateKeeper Bypass [Електронний ресурс] // Filippo Cavallarin. – 2019. – Режим доступу до ресурсу: https://www.fcvl.net/vulnerabilities/macosx-gatekeeper-bypass.

14. Android Q privacy change: Scoped storage [Електронний ресурс] // Developers Android – Режим доступу до ресурсу: https://developer.android.com/preview/privacy/scoped-storage.

15. Створення месенджерів для навчального закладу: [Електронний ресурс] – Режим доступу до ресурсу: https://vrc.rv.ua/case_study/vetmarketing/

16. 1Пошук цільової аудиторії: [Електронний ресурс] – Режим доступу до ресурсу: https://creativesmm.com.ua/jak-znajjtu-svoju-cilovuauditoriju/

17. Pro Git 2nd Edition [Електронний ресурс] / Scott Chacon, Ben Straub — 2014 — Режим доступу: https://git-scm.com/book/en/v2 .

18. Документація Docker Hub [Електронний ресурс] — Режим доступу: https://docs.docker.com/docker-hub/ .

19. PCMag [Електронний ресурс] режим доступу: https://www.pcmag.com

20. Mixpanel. [Електронний ресурс] режим доступу: https://mixpanel.com Indeed. [Електронний ресурс] режим доступу: https://www.indeed.com

21. Toptal. [Електронний ресурс] режим доступу: https://www.toptal.com

22. Course Report. [Електронний ресурс] режим доступу: https://www.coursereport.com

23. Perfecto. [Електронний ресурс] режим доступу: https://www.perfecto.io

24. Ray Wenderlich. [Електронний ресурс] режим доступу:https://www.raywenderlich.com

25. Web Design. [Електронний ресурс] режим доступу:https://webdesign.tutsplus.com

26. Коноваленко І.В., Федорів П.С. Системне програмування у Windows з прикладами на Delphi. Навч. посіб. Для тех. спец. Вищих навчальних закладів. Тернопіль: ТНТУ ім. І. Пулюя, 2012. 320 с.

27. Коноваленко І. В. Платформа .NET та мова програмування C# 8.0 : навчальний посібник / І. В. Коноваленко, П. О. Марущак. – Тернопіль : ФОП Паляниця В. А., 2020. – 320 с.

28. Комп'ютерні мережі: [навчальний посібник] / А. Г. Микитишин, М. М. Митник, П. Д. Стухляк, В. В. Пасічник. — Львів: «Магнолія 2006», 2013. — 256 с. ISBN 978-617-574-087-3

29. Петрик М.Р. Моделювання програмного забезпечення : науково методичний посібник / М.Р. Петрик, О.Ю. Петрик – Тернопіль : Вид-во ТНТУ імені Івана Пулюя, 2015. – 200 с.

30. Software Testing Help. [Електронний ресурс] режим доступу: https://www.softwaretestinghelp.com

31. Coder Lessons. [Електронний ресурс] режим доступу: https://coderlessons.com

32. Data Science. [Електронний ресурс] режим доступу: https://datascience.eu/ua

33. Алексейчук І.С. Про технологію створення системи тестування / І.С. Алексейчук // Нові технології навчання: Науково-методичний збірник. — К.: НМЦВД, 2022. — С.43-92.

34. Харченко В.П., Знаковська Є.А., Бородін В.А. Операційні системи та системи програмування: навч. посіб. Київ: Вид-во Нац. авіац. ун-ту «НАУдрук», 2012. 360с.

35. Осадчий В.В. Основи розробки веб-додатків. Навчальний посібник / В.В. Осадчий, В.С. Круглик – Мелітополь: ТОВ «Видавничий будинок ММД», 2012. – 540 с.

36. Пасічник О.Г., Пасічник О.В., Стеценко І.В. Основи веб-дизайну. [Навч. посіб.]. К.: Вид. група BHV. 2009. 336 с.

ДОДАТКИ

Додаток А

ЛІСТИНГ ПРОГРАМНОГО КОДУ МОБІЛЬНОГО ЗАСТОСУНКУ

```
import
Minipass
from
'minipass' ;
        import * as path from 'path';
        import { ReadEntry } from 'tar';
                                                                export function sanitizedName(name: string) {
            return name
              .replace(/[\W_]+/g, '')
              .normalize('NFD')
              .replace(/[\u0300-\u036f]/g, '');
            }
                        class Transformer extends Minipass {    data: string;
                                                                    constructor(private name: string) {
              super();
              this.data = '';
            }
          write(data: string) {
this.data += data;      return true;
            }
            end() {
              let replaced = this.data
                .replace(/Hello App Display Name/g, this.name)
                .replace(/HelloWorld/g, sanitizedName(this.name))
           .replace(/helloworld/g, sanitizedName(this.name.toLowerCase()));
         super.write(replaced);          return super.end();
            }
          }


        // Files and directories that have `_` as their first character in React Native CLI templates
        // These should be transformed to have `.` as the first character
        const UNDERSCORED_DOTFILES = [
            'buckconfig',
            'eslintrc.js',
            'flowconfig',
            'gitattributes',
            'gitignore',


            'prettierrc.js',
            'watchmanconfig',
            'editorconfig',
            'bundle',
            'ruby-version',
            'node-version',
            'xcode.env',
          ];
                        export function createEntryResolver(name: string) {          return (entry:
ReadEntry) => {       if (name) {
```

```
                // Rewrite paths for bare workflow
                entry.path = entry.path
.replace(
                    /HelloWorld/g,
                    entry.path.includes('android') ? sanitizedName(name.toLowerCase()) : sanitizedName(name)
                  )
                  .replace(/helloworld/g, sanitizedName(name).toLowerCase());
              }
            if (entry.type && /^file$/i.test(entry.type) && path.basename(entry.path) === 'gitignore') {
// Rename `gitignore` because npm ignores files named `.gitignore` when publishing.
              // See: https://github.com/npm/npm/issues/1862
entry.path = entry.path.replace(/gitignore$/, '.gitignore');
              }
            for (const fileName of UNDERSCORED_DOTFILES) {
entry.path = entry.path.replace(`_${fileName}`, `.${fileName}`);
              }
            };
          }
                                    export function createFileTransform(name: string) {      return (entry:
                                                                                              ReadEntry) => {
            // Binary files, don't process these (avoid decoding as utf8)
            if (
              ![
                '.png',
                '.jpg',
                '.jpeg',
                '.gif',
                '.webp',
                '.psd',
                '.tiff',
                '.svg',
                '.jar',
                '.keystore',


                // Font files
                '.otf',
                '.ttf',
              ].includes(path.extname(entry.path)) &&
              name
            ) {
              return new Transformer(name);
            }
            return undefined;
          };
        }
import JsonFile from '@expo/json-file'; import * as
PackageManager from '@expo/package-manager'; import
spawnAsync from '@expo/spawn-async'; import chalk from
'chalk'; import { readdirSync } from 'fs-extra'; import getenv
from 'getenv'; import got from 'got'; // @ts-ignore import
merge from 'lodash/merge'; import ora from 'ora'; import *
```

```
as path from 'path'; import { Stream } from 'stream'; import
tar from 'tar'; import { promisify } from 'util';

import Logger from './Logger'; import { createFileTransform,
createEntryResolver } from './createFileTransform';

// @ts-ignore const pipeline =
promisify(Stream.pipeline);

const isMacOS = process.platform === 'darwin';

const packageJSON = require('../package.json');

export type PackageManagerName = 'yarn' | 'npm';

/**
 * Extract a template app to a given file path and clean up any properties left over from npm to
* prepare it for usage.
 */
export async function extractAndPrepareTemplateAppAsync(projectRoot: string) {
const projectName = path.basename(projectRoot);

  await downloadAndExtractNpmModule(projectRoot, 'expo-template-bare-minimum', projectName);

  const config: Record<string, any> = {
name: projectName,
   expo: {      name:
projectName,      slug:
projectName,
   },
  };

  let appFile = new JsonFile(path.join(projectRoot,
'app.json'));   let appJson = merge(await appFile.readAsync(),
config);   await appFile.writeAsync(appJson);

  let packageFile = new JsonFile(path.join(projectRoot, 'package.json'));
let packageJson = await packageFile.readAsync();
  // name and version are required for yarn workspaces (monorepos)
const inputName = 'name' in config ? config.name : config.expo.name;
packageJson.name = sanitizeNpmPackageName(inputName);
  // These are metadata fields related to the template package, let's remove them from the package.json.
  // A good place to start
packageJson.version = '1.0.0';
```

```
packageJson.private = true;
delete
packageJson.description;
delete packageJson.tags;
delete packageJson.repository;

  await packageFile.writeAsync(packageJson);

  return projectRoot;
}

export function sanitizeNpmPackageName(name: string): string {   //
https://github.com/npm/validate-npm-package-name/#naming-rules
return (
    applyKnownNpmPackageNameRules(name) ||
    // If nothing is left use 'app' like we do in Xcode projects.
    'app'
  );
}

function applyKnownNpmPackageNameRules(name: string): string | null {
  // https://github.com/npm/validate-npm-package-name/#naming-rules

  // package name cannot start with '.' or '_'.
  while (/^(\.|_)/.test(name)) {
    name = name.substring(1);
  }

  name = name.toLowerCase().replace(/[^a-zA-Z._\-/@]/g, '');

  return (
name
      // .replace(/![a-z0-9-._~]+/g, '')
      // Remove special characters
      .normalize('NFD')
      .replace(/[\u0300-\u036f]/g, '') || null
  );
}

export function validateName(name?: string): string |
true {   if (typeof name !== 'string' || name === '') {
return 'The project name can not be empty.';
  }
  if (!/^[a-z0-9@.\-_]+$/i.test(name)) {     return 'The project name
can only contain URL-friendly characters.';
```

```
  }
  return true;
}

export async function
initGitRepoAsync(   root: string,   flags:
{ silent: boolean } = { silent: false }
) {
  // let's see if we're in a git tree   try {     await spawnAsync('git', ['rev-parse', '--is-inside-
work-tree'], { stdio: 'ignore', cwd: root });     !flags.silent && Logger.gray('New project is
already inside of a Git repo, skipping git init.');     return false;   } catch (e) {     if (e.errno
=== 'ENOENT') {
     !flags.silent && Logger.gray('Unable to initialize Git repo. `git` not in PATH.');
return false;
   }
  }

  // not in git tree, so let's init   try {     await spawnAsync('git', ['init'], { stdio: 'ignore', cwd: root });
await spawnAsync('git', ['add', '-A'], { stdio: 'ignore', cwd: root });     const commitMsg = `Initial
commit\n\nGenerated by ${packageJSON.name} ${packageJSON.version}.`;     await spawnAsync('git',
['commit', '-m', commitMsg], {
     stdio: 'ignore',
cwd: root,
   });

   !flags.silent && Logger.gray('Initialized a Git repository.');
return true;
  } catch (e) {
   // no-op -- this is just a convenience and we don't care if it fails
return false;
  }
}

export async function
installDependenciesAsync(   projectRoot:
string,   packageManager:
PackageManagerName,
  flags: { silent: boolean } = { silent: false }
) {
  const options = { cwd: projectRoot, silent: flags.silent };   if
(packageManager === 'yarn') {     const yarn = new
PackageManager.YarnPackageManager(options);     await
yarn.installAsync();
  } else {
   await new PackageManager.NpmPackageManager(options).installAsync();
```

```
  }
}

// Any of these files are allowed to exist in the projectRoot
const tolerableFiles = [
  // System
  '.DS_Store',
  'Thumbs.db',
  // Git
  '.git',
  '.gitattributes',
  '.gitignore',
  // Project
  '.npmignore',
  '.travis.yml',
  'LICENSE',
  'docs',
  '.idea',
  // Package manager
  'npm-debug.log',   'yarn-
debug.log',
  'yarn-error.log',
];

export function getConflictsForDirectory(projectRoot: string): string[] {
return readdirSync(projectRoot).filter(
    (file: string) => !(/\.iml$/.test(file) || tolerableFiles.includes(file))
  );
}

export function
logProjectReady({   cdPath,
packageManager,
}: {
  cdPath: string;
packageManager: string;
}) {
  Logger.nested(chalk.bold(`✓ Your project is ready!`));
  Logger.newLine();

  // empty string if project was created in current
directory   if (cdPath) {     Logger.nested(
      `To run your project, navigate to the directory and run one of the following ${packageManager} commands.`
    );
    Logger.newLine();
```

```
    Logger.nested(`- ${chalk.bold('cd ' + cdPath)}`);
  } else {
    Logger.nested(`To run your project, run one of the following ${packageManager} commands.`);
Logger.newLine();
  }

  Logger.nested(`- ${chalk.bold(packageManager === 'npm' ? 'npm run android' : 'yarn android')}`);

  let macOSComment = '';
if (!isMacOS) {
    macOSComment =
      ' # you need to use macOS to build the iOS project - use the Expo app if you need to do iOS development without a Mac';
  }
  Logger.nested(
    `- ${chalk.bold(packageManager === 'npm' ? 'npm run ios' : 'yarn ios')}${macOSComment}`
  );

  Logger.nested(`- ${chalk.bold(packageManager === 'npm' ? 'npm run web' : 'yarn web')}`);
}

export async function installPodsAsync(projectRoot:
string) {   let step = logNewSection('Installing CocoaPods.');
if (process.platform !== 'darwin') {
    step.succeed('Skipped installing CocoaPods because operating system is not on macOS.');
return false;
  }
  const packageManager = new
PackageManager.CocoaPodsPackageManager({     cwd:
path.join(projectRoot, 'ios'),     log: Logger.nested,     silent:
!getenv.boolish('EXPO_DEBUG', false),
  });

  if (!(await packageManager.isCLIInstalledAsync())) {
try {
     step.text = 'CocoaPods CLI not found in your PATH, installing it
now.';      step.render();      await packageManager.installCLIAsync();
step.succeed('Installed CocoaPods CLI');      step =
logNewSection('Running `pod install` in the `ios` directory.');
    } catch (e) {
step.stopAndPersist({
symbol: '⚠ ',
text: chalk.red(
        'Unable to install the CocoaPods CLI. Continuing with initializing the project, you can install CocoaPods afterwards.'
      ),
    });
```

```
    if (e.message) {
      Logger.nested(`- ${e.message}`);
    }
    return false;
  }
 }

 try {
  await packageManager.installAsync();
step.succeed('Installed pods and initialized Xcode workspace.');
return true;  } catch (e) {    step.stopAndPersist({      symbol:
'⚠ ',      text: chalk.red(

      'Something went wrong running `pod install` in the `ios` directory. Continuing with initializing the project, you can debug this
afterwards.'
    ),
  });
  if (e.message) {
    Logger.nested(`- ${e.message}`);
  }
  return false;
 }
}

export function logNewSection(title:
string) {  let spinner =
ora(chalk.bold(title));  spinner.start();
return spinner;
}

async function getNpmUrlAsync(packageName: string): Promise<string> {
const url = (await spawnAsync('npm', ['v', packageName,
'dist.tarball'])).stdout;

 if (!url) {
  throw new Error(`Could not get npm url for package "${packageName}"`);
 }

 return url;
}

async function downloadAndExtractNpmModule(
 root: string,  npmName: string,
projectName: string ): Promise<void> {  const
url = await getNpmUrlAsync(npmName);
```

```
  return pipeline(
got.stream(url),
tar.extract(
    {
      cwd: root,       transform:
createFileTransform(projectName),       onentry:
createEntryResolver(projectName),       strip: 1,
    },
    []
   )
 );
}
```