Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут деревообробних та
комп'ютерних технологій і дизайну
(повне найменування інституту, назва факультету (відділення))

Кафедра інформаційних технологій
(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка

до дипломної роботи

другий (магістерський)
(рівень вищої освіти)

на тему: «Розроблення сервісу прокату мобільного електротранспорту»

Виконав: студент 6 курсу групи КН-61м
спеціальності
122 "Комп'ютерні науки"
(шифр і назва напряму підготовки, спеціальності)

Пісецький Т. Я.
(прізвище та ініціали)

Керівник Процик Ю. С.
(прізвище та ініціали)

Рецензент
(прізвище та ініціали)

Львів – 2022

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ деревообробних та комп'ютерних технологій і дизайну
Кафедра інформаційних технологій
Рівень вищої освіти другий (магістерський)
Спеціальність 122 "Комп'ютерні науки"
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ Крошний І. М.
"____" _____20__ року

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Пісецькому Тарасу Ярославовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Розроблення сервісу прокату мобільного електротранспорту*

керівник роботи *Процик Юрій Степанович, к.ф.-м.н.*,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від "*13*" *грудня 2021 року* № *С-617*

2. Термін подання студентом роботи *12 грудня 2022 року*

3. Вихідні дані до роботи *Аналіз шляхів вирішення задачі, програмна реалізація додатку*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) *Вступ. Розділ 1. Стан проблемної області. Розділ 2. Інформаційне забезпечення. Розділ 3. Математичне забезпечення. Розділ 4. Програмне забезпечення. Розділ 5. Розроблення стартап-проекту. Висновки. Список використаної літератури. Додатки*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
слайди для доповіді (підготовка матеріалу для доповіді загальним обсягом 10-12 слайдів)

6. Дата видачі завдання *20 грудня 2021 року*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1.	Огляд літератури згідно досліджуваної теми. Збір необхідних матеріалів	*21.12.2021 р.* *21.01.2022 р.*	
2.	Постановка задачі і її формалізація	*22.01.2022 р.* *28.02.2022 р.*	
3.	Розроблення бекенд частини додатку	*01.03.2022 р.* *11.06.2022 р.*	
4.	Проектування візуальної UI/UX частини	*12.06.2022 р.* *01.07.2022 р.*	
5.	Розробка фронтенд частини додатку	*02.07.2022 р.* *09.10.2022 р.*	
6.	Об’єднання обох частин в єдине ціле	*10.10.2022 р.* *24.10.2022 р.*	
7.	Тестування роботи додатку	*25.10.2022 р.* *19.11.2022 р.*	
8.	Аналіз дипломної роботи як стартап-проекту	*20.11.2022 р.* *29.11.2022 р.*	
9.	Оформлення пояснювальної записки та здача на рецензування	*30.11.2022 р.* *12.12.2022 р.*	

Студент _____ Пісецький Т. Я.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Процик Ю. С.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Дипломна робота містить 83 сторінок пояснювальної записки, 50 рисунків, 2 додатки, 18 джерел.

У роботі проаналізовано доступність та зручність оренди електричних пристроїв, таких як електровелосипеди, електросамокати, гіроскутери тощо. Визначена ціль та рівень потреби у зручній системі для полегшення пересування містом. Створено програмне забезпечення та користувацький інтерфейс для використання сервісу прокату без посередників. Розроблений інтерфейс забезпечує зручну взаємодію між користувачем та сервісом. Реалізований функціонал для ефективного та безпечного використання веб-застосунку.

Ключові слова:

Python, Flask, REST, API, JavaScirpt, React, веб-додаток, backend, frontend, оренда, електротранспорт.

ABSTRACT

The thesis contains 83 pages of explanatory note, 50 figures, 2 appendices, 18 used literary sources.

In current work the availability and convenience of renting electrical devices, such as electric bicycles, electric scooters, gyroscooters, etc were analyzed. The purpose and level of need of a convenient system to facilitate movement through the city is defined. Software has been created for easy use of the rental service without intermediaries. The developed interface provides convenient interaction between the user and the service. Implemented features for easy and safe use of web applications.

Keywords:

Python, Flask, REST, API, JavaScirpt, React, web-application, backend, frontend, rental, electric transport.

ТЕХНІЧНЕ ЗАВДАННЯ

Необхідно розробити додаток для сервісу прокату мобільного електротранспорту з використанням клієнт-серверної архітектури.

Для повноцінного функціонування додатку потрібно створити backend частину на мові програмування Python з використанням фреймворку для створення веб-додатків Flask, розробити API з використанням архітектури REST для зв’язку клієнта та сервера, запустити графову базу даних на основі Neo4j. Frontend частину додатку реалізувати мовою програмування JavaScript з використанням графічного фреймворку React. Протестувати працездатність REST API за допомогою Postman, інструмента для тестування API, а також роботу додатку у різних браузерах (Google Chrome, Firefox, Safari, Edge) та на різних платформах (Windows, Android, iOS).

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

API - Application Programming Interface

REST - Representational State Transfer

SSL - Secure Sockets Layer

IP - Internet Protocol

JWT - JSON Web Token

GIL - Global Interpreter Lock

HTTP - HyperText Transfer Protocol

MVC - Model-view-controller

ASGI - Asynchronous Server Gateway Interface

WSGI - Web Server Gateway Interface

URI - Uniform Resource Identifier

HTML - HyperText Markup Language

SOAP - Simple Object Access Protocol

TLS - Transport Layer Security

HTTPS - HyperText Transfer Protocol Secure

URL - Uniform Resource Locator

DES - Data Encryption Standard

DdoS - Denial-of-service attack

SQL - Structured Query Language

VPN - Virtual Private Network

DNS - Domain Name System

IDE - Integrated Drive Electronics

CSS - Cascading Style Sheets

SSH - Secure SHell

FTP - File Transfer Protocol

ПК - Персональний комп’ютер

UI - User Interface

NAT - Network Address Translation

БД - база даних

URL - Uniform Resource Locator

JSON - JavaScript Object Notation

DOM - Document Object Model

URL - Uniform Resource Locator

HTML - Hyper Text Markup Language

AJAX - Asynchronous JavaScript and XML

SPA - Single Page Application

UI - User Interface

UX - User Experience

NPM - node package manager

NVM - node version manager

API - Application Programming Interface

CSS - Cascading Style Sheets

JS - JavaScript

HTTP - Hyper Text Transfer Protocol

ВСТУП

З кожним днем мобільні види електротранспорту стають все більш і більш популярними. Не зважаючи на це, не кожен пересічний громадянин нашої країни в змозі дозволити собі один з таких, дивлячись з фінансової точки зору. Деякі люди просто вважають недоцільним, витрачати кошти на такі речі. В таких випадках на допомогу приходить сервіс з прокату компактного електротранспорту.

Якщо ви в пошуку одного з таких сервісів, для комфортної оренди не лише самокатів, а й електровелосипедів та ховербордів, то наш сервіс до ваших послуг. Для зручної оренди такого роду електротранспорту пропонуємо вам застосунок BitRoller. Цей застосунок відрізняється від аналогів своєю інтуїтивністю, лаконічністю інтерфейсу та звичайно ж розширеним функціоналом. Зокрема, до так званих “фішок” цього сервісу можна віднести функцію прокладання маршрутів від точки місцезнаходження, до бажаної кінцевої точки. Подекуди так бракує цього функціоналу в сервісах-аналогах і користувачі часто стикаються з неприємною ситуацією, коли для оренди пристрою вони використовують додаток, запропонований сервісом, а після його розблокування змушені відкривати, наприклад, google maps, для того, щоб прокласти зручний маршрут. Також в стадії розробки перебуває функція “Режим туриста”, яка передбачає в собі прокладання туристичних маршрутів через відомі історичні та пам’ятні місця, при цьому, після досягненні однієї з таких цілей, буде запропоновано прослухати коротку інформацію, про цю локацію, включаючи різноманітні цікаві факти не лише з минулого, а й з теперішнього. Ще одним з нововведень є прокладання маршруту до бажаного електричного пристрою, за моїми спостереженнями, цей функціонал також відсутній в додатках-аналогах, а це значно спрощує пошук одного з таких пристроїв. Додаток був створений з використанням різноманітних сучасних методик написання коду та сучасного програмного забезпечення.

Основною причиною вибору такої теми дипломної роботи слугувало те, що на сьогоднішній день актуальним залишається вирішення двох глобальних

проблем. таких як погіршення екологічної ситуації в мегаполісах та швидке збільшення трафіку на дорогах великих міст. Проблема екології полягає в різкому збільшенні приватного та громадського транспорту з двигунами внутрішнього згоряння, що суттєво збільшує викиди вихлопних газів в атмосферу, спричиняючи забруднення повітря. Інша проблема полягає в тому, що з кожним днем жителям міста стає все складніше добратися до місця роботи, чи навчання через велику кількість заторів на дорогах, відповідно туристам також стає складніше подорожувати містом, з кожним роком, час, необхідний для того, щоб дістатися в мегаполісі з одної і тої ж точки “А” в точку “Б” зростає. Це призводить до погіршення комфорту життя населення, а також, не варто забувати про те, що час - це гроші, тому відповідно це також має вплив й на економіку та логістику міста. Сервіс для оренди мобільного електротранспорту BitRoller, якраз слугує для часткового вирішення цих проблем. Завдяки, можливостям, які надає цей сервіс, люди отримують змогу за доступну ціну використовувати екологічно чистий вид транспорту, та за рахунок своєї портативності, використання таких пристроїв також зменшує і трафік на дорогах міста.

Актуальність цієї теми зумовлена тим, що популярні зараз сервіси-аналоги для оренди портативних електричних транспортних засобів зосереджені в більшості випадків лише на самокатах. Наступним аспектом є обмежений функціонал додатків, які використовують аналогічні сервіси, що може не відповідати вимогам всіх користувачів, та значно знижувати зручність користування. Також аналоги не дають можливість використовувати web-інтерфейс, що в свою чергу призводить до потреби скачування додатку, але не завжди є можливість та швидке інтернет підключення для цього. Враховуючи всі вище перераховані аспекти мій вибір впав саме на вирішення цих проблем, відповідно на створення додатку для нового сервісу прокату компактних електричних транспортних засобів під назвою BitRoller.

Об’єкт дослідження – сервіс прокату мобільного електротранспорту.

Предмет дослідження – технології, підходи реалізації та програмні рішення для створення сервісу прокату мобільного електротранспорту.

Метою роботи є створення сервісу для прокату мобільного електротранспорту з функціоналом, який буде відповідати всім забаганкам користувачів й мати переваги над типовими рішеннями конкурентів.

Завдання, які потрібно вирішити:

- реалізувати backend сервісу для оренди електротранспорту;
- створити API для комунікації клієнта з сервером;
- реалізувати лаконічний та інтуїтивний frontend;
- протестувати роботу створеного сервісу;
- спрогнозувати актуальність даного сервісу та створити бізнес-план.

Практичне значення полягає у використанні даного сервісу жителями міста та туристами для оренди мобільного електротранспорту з метою швидкого пересування з точки А в точку Б в умовах великого міста та для різноманітних туристичних поїздок містом.

Наукова новизна. В процесі розроблення сервісу для прокату мобільного електротранспорту використано сучасні підходи у створенні веб-додатків з клієнт-серверною архітектурою. В результаті розробки створено функціонал, який не має аналогів, що має позитивно позначитися на досвіді користування таким сервісом.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1 Важливість використання електротранспорту

Останніми роками люди та уряди в усьому світі усвідомили екологічний хаос, який у всьому світі спричинило надмірне використання викопного палива та нафтопродуктів. У результаті ми починаємо рухатися до глибокої енергетичної зміни.

Зіткнувшись з цією реальністю, уряди підписали міжнародні угоди, зобов'язавшись розробити державну політику, яка сприятиме зменшенню забруднюючих речовин і викидів вуглецю.

Перехід на мобільний електричний транспорт – це один зі шляхів, який сприяє зміцненню екологічної ситуації та вирішенню екологічної кризи. Однак існують фактори, які все ще не дають можливості для масового впровадження мобільного електротранспорту принаймні у всіх великих містах.

1.2 Переваги та недоліки електротранспорту

Переваги мобільного електротранспорту:

- *Універсальність.* Мобільний електротранспорт не обмежується лише електросамокатами; також були розроблені електричні варіанти різноманітного звичного для нас мобільного транспорту, такі як електричні мотоцикли, велосипеди, скутери. Не варто забувати, що користується популярністю також і електричний громадський транспорт великої місткості.
- *Нульове забруднення.* Електротранспорт не викидає парникових газів, тому немає забруднення атмосфери. Крім того, він практично не виробляє шуму, а також усуває шумове забруднення.
- *Електроенергія дешевша за паливо.* Зарядка будь-якого типу електротранспорту дешевша, ніж використання палива під час поїздки

автомобілем з двигуном внутрішнього згоряння. У Європі підзарядка, еквівалентна 100 км автономії, коштує близько 1 євро.

- *Менше механічних поломок.* Припинення використання систем внутрішнього згоряння зменшує кількість деталей у збірці цих електричних транспортних засобів, таким чином також зменшуючи ймовірність поломок у їхніх механізмах і, відповідно, також зменшуючи витрати на технічне обслуговування.

Недоліки електротранспорту:

- *Високі ціни.* На жаль, в наш час все ще важко дозволити собі від електричного самоката до автомобіля, ціни на ці технології ще не настільки доступні для середньої кишені.
- *Відсутність інфраструктури.* Правда полягає в тому, що, все ще недостатньо інфраструктури, щоб гарантувати, що потреби в енергозабезпеченні цих транспортних засобів будуть задоволені. В той час, коли автозаправні станції розташовані кожні декілька кілометрів вздовж автомагістралей, та по декілька штук в місті, то кількість зарядних станцій для електротранспорту, можна «порахувати на пальцях», принаймні це стосується нашої країни як мінімум.
- *Автономність.* Хоча вже є кілька моделей із запасом ходу трохи більше 500 км, більшість із цих пристроїв не підходять для дуже тривалих подорожей, а у випадку з автомобілями вони поки що не рекомендовані для водіння за межами міста.
- *Час зарядки.* Однією з основних перешкод для користувачів, які переходять на електромобіль, є час зарядки, який може коливатися від півгодини до 12 годин залежно від типу розетки. З іншого боку, наповнення бака займає 5 хвилин.
- *Деградація батареї.* Постійне заряджання, особливо при використанні розеток з різною потужністю, спричиняє погіршення роботи акумуляторів цього електротранспорту. Окрім необхідності їх заміни, утилізація цих батарей може ускладнитися та забруднювати навколишнє середовище.

Зміни неминучі, і на краще! Альтернативна вартість — це добробут людей і стійкість планети. Зміна способу нашого пересування стане маленькою піщинкою для загального блага.

1.3 Актуальність прокату мобільного електротранспорту

Мобільний електротранспорт – це компактний транспортний засіб з електроприводом. Цей механізм частково або повністю забезпечує його рух. Від звичайного його відрізняють 3 ознаки: електродвигун, акумулятор і контролер. Щоб повністю зарядити, наприклад, середньостатистичний електровелосипед знадобиться 6 годин. Для зарядки акумулятора до 75% потрібно до 4 годин.

Мобільний електротранспорт поєднує в собі як різноманітні види вже звичного для нас компактного транспорту (велосипеди, скутери, самокати), обладнаного електродвигуном, так і сучасні види мобільного транспорту, які існують лише в сукупності з електродвигуном (сігвеї, ховерборди).

Прокат цього виду транспорту – актуальна і популярна послуга, яка дозволяє людям не витрачати гроші на покупку одного з таких. Мобільний електротранспорт – це можливість вдихнути свіжість і динамічність під час пересування по місту. Гнучкий, екологічно чистий і екологічно чистий транспорт не тільки доставить вас вчасно, але й зекономить вам копійчину.

Оренда такого типу транспорту, це хороше рішення, щоб уникнути затор при використанні громадського транспорту чи приватного автомобіля, уникнути натовпу в громадському транспорті, а також це хороше рішення, щоб зробити свій внесок в екологію навколишнього середовища. Хочу також додати, що сучасні рішення оренди таких пристроїв дозволяють знайти один з них в багатьох точках міста, орендувати, здійснити поїздку, оплатити і залишити транспорт у зручному для вас місці. Це значно збільшує гнучкість вашої подорожі, так як ви в будь-який момент

можете здати орендований транспорт і змінити спосіб пересування, або ж залишити його біля входу в потрібну вам будівлю. Це значно скорочує час добирання з точки «А» в точку «Б», навряд чи при використанні особистого автомобіля, ви б знайшли місце для паркування поблизу місця призначення, або навряд чи потрібна зупинка громадського транспорту буде знаходитись в районі 10 м. від кінцевої точки подорожі.

Висновки до розділу

На сьогоднішній день, електротранспорт є не лише вирішенням багатьох екологічних проблем, а й урбаністичних. Зрозуміло, що як будь-яка річ, він має як переваги, так і недоліки, але з кожним роком недоліків стає все менше, а переваги давно вже перекрили усі мінуси цього виду транспорту.

Зараз особливо набирає популярності новий тренд на різні види компактного електротранспорту. У великих містах є досить поширена проблема з трафіком на дорогах, тому іноді людям важко дістатись до своїх місць призначень, таких як місце роботи, заклад навчання та інше, тому жителі великих міст все частіше використовують для пересування мобільний електротранспорт. Але не кожен може придбати собі один, або рахує, що це зайві витрати. Якщо покупка одного з них – це не вихід, то є можливість орендувати його. В теорії, ви можете в будь-якій точці міста обрати один з таких пристроїв і вчасно та зручно дістатись до місця призначення. Звичайно, ви можете використовувати ці пристрої для туристичних маршрутів також.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Асинхронне API

Для web-програм є надзвичайно важливою змога обробки великої кількості запитів в один час. Це сприяє зменшенню часу, необхідного для обробки запитів. Такий підхід називається паралельністю і він просто необхідний у випадках, коли від декількох не пов’язаних між собою клієнтів, за короткий проміжок часу, надходить значна кількість запитів.

Якщо ви здебільшого працювали з REST API, можливо, ви не настільки знайомі з асинхронними протоколами API, такими як WebSocket і gRPC. Асинхронні API дозволяють передавати дані, надсилати кілька запитів одночасно та інтелектуально керувати зв’язком між службами, тоді як синхронні API вимагають від вас робити новий запит кожного разу, коли вам потрібні дані. Ви можете робити багато однакових речей з обома типами API, але ви побачите, що деякі варіанти використання краще підходять для одного чи іншого. Давайте детальніше розглянемо, чим асинхронні API відрізняються від синхронних API.

Синхронні API часто використовують HTTP або HTTPS для транспортування, а HTTP є односпрямованим протоколом. Клієнт надсилає запит на сервер, а потім сервер надсилає відповідь HTTP або HTTPS. Асинхронні API, як правило, використовують двонаправлені протоколи, такі як HTTP/2. Коли ви використовуєте двонаправлений протокол, клієнт і сервер можуть підтримувати з’єднання, надсилаючи й отримуючи дані стільки, скільки їм потрібно. Ви можете подумати про це так: коли ви робите синхронний запит, ви отримуєте дані з сервера, але ви можете використовувати асинхронний запит, щоб попросити сервер надіслати вам останні дані.

Коли ви надсилаєте запит до асинхронного API, ваша програма може продовжувати обробляти інші інструкції для вашого користувача, поки вона очікує

відповіді від сервера API. Це відрізняється від синхронного API, де програма не продовжуватиме роботу, доки не отримає відповідь. Якщо ресурс, служба або сховище даних недоступні відразу після того, як ви робите запит, асинхронний API може використовувати зворотний виклик, наприклад веб-хук, щоб повідомити вашу програму, коли ресурс буде готовий.

Не всі архітектури API точно вписуються в синхронні або асинхронні мітки. Наприклад, GraphQL можна вважати синхронним, оскільки ви надсилаєте запити через HTTP, але він також підтримує асинхронний обмін повідомленнями за допомогою WebSockets за допомогою свого сервера підписки. Ви можете зареєструватися на сервері підписки GraphQL, щоб отримувати асинхронні оновлення, з додатковою перевагою вибору саме полів, які ви хочете включити у відповідь.

Мікросервіси є чудовим варіантом використання асинхронних запитів API. Коли ви налаштовуєте внутрішні та зовнішні служби програми на асинхронний зв’язок, служби можуть надсилати одна одній повідомлення, не обов’язково будучи доступними одночасно, часто за допомогою брокера повідомлень для обробки запитів. Замість того, щоб запитувати нову інформацію, служби можуть підписуватися на важливі для них події та отримувати push-оновлення. Архітектура, керована подіями, спирається на цей шаблон зв’язку між службами публікація-підписка. На цій діаграмі показано приклад двох виробників подій, які діляться подіями з трьома службами за допомогою шаблону публікації-підписки:

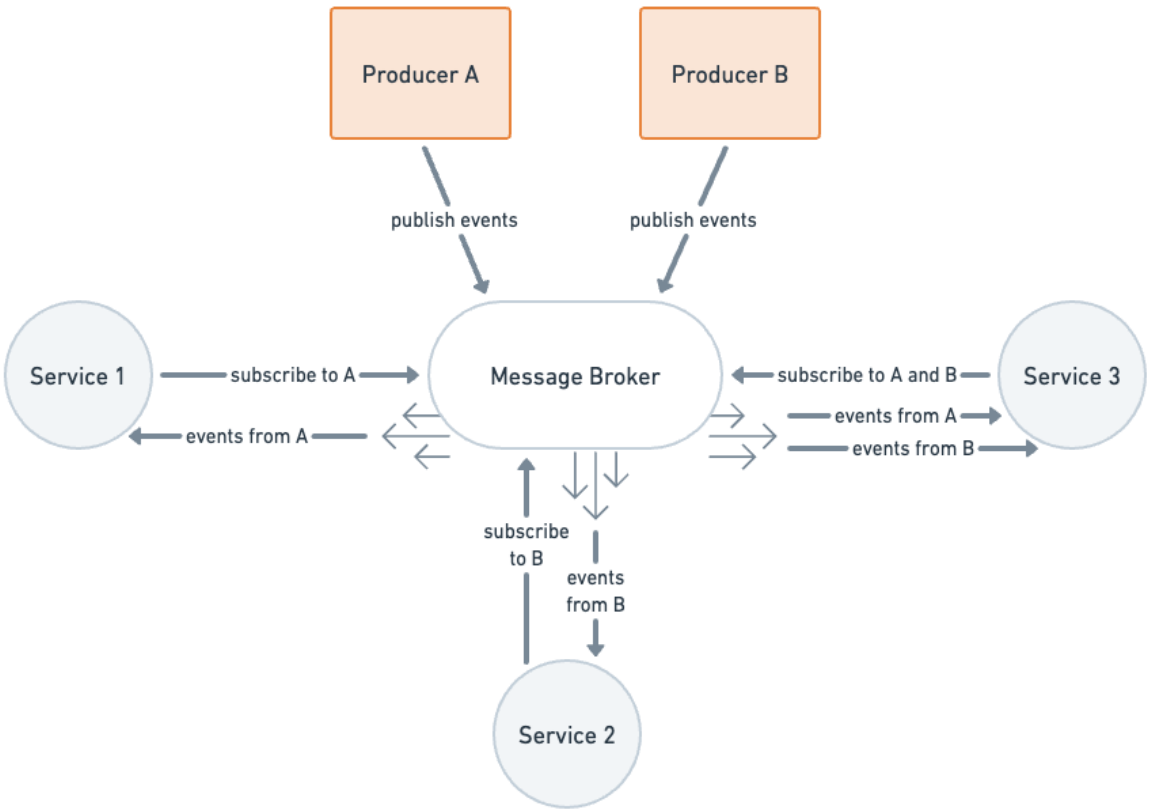


Рисунок 2.1 – Приклад шаблону публікації-підписки

HTTP/2 також підтримує необмежену кількість повідомлень через одне з'єднання, тому служби можуть обмінюватися даними без необхідності ініціювати нове з'єднання кожного разу, коли вони роблять запит, як це відбувається з HTTP. Це зменшує ймовірність зіткнення з обмеженнями TCP-з'єднання браузера. Це також дозволяє програмам спілкуватися з API у фоновому режимі під час виконання інших завдань, що призводить до кращої продуктивності та масштабування для програми з великою кількістю активності користувачів. Інфраструктура gRPC є популярним вибором для внутрішніх служб, оскільки вона дозволяє їм ефективно передавати дані один одному за допомогою структурованих корисних даних.

Асинхронні API необхідні для двонаправленої потокової передачі. Якщо у вас є двонаправлене з'єднання, клієнт і сервер можуть постійно надсилати повідомлення один одному. HTTP/2 підтримує цей тип підключення за замовчуванням. Хоча технічно можливо передавати дані через HTTP (зазвичай відоме

як фрагментація), це додає складності на стороні клієнта, і ви можете сповільнюватися через обмеження буфера та інші проблеми. Ви матимете набагато кращий досвід, якщо використовуватимете HTTP/2 для потокової передачі даних.

Приклади використання асинхронного API

Ось кілька прикладів програм, які можуть використовувати асинхронний API:

- Обмін повідомленнями
- Соціальні мережі
- Мобільні ігри
- Банківська справа

У багатьох випадках програми виграють від використання комбінації синхронних і асинхронних API. Хоча асинхронні API можуть запропонувати такі переваги, як швидший зв’язок, швидший час відповіді та надійне масштабування, у синхронних API є переваги. Наприклад, якщо вам потрібно переконатися, що запити обробляються в певному порядку, синхронні виклики API підійдуть краще. Синхронні API також менш складні в налаштуванні, тому вони все ще ідеальні для прямого шаблону запит-відповідь. Якщо ви ще не використовуєте асинхронні API, ви можете почати їх інтегрувати, зберігаючи функціональні можливості наявних REST API.

2.2 Безпека даних

2.2.1 Найкращі практики безпеки API

API часто самостійно документують інформацію, таку як їх реалізація та внутрішня структура, яка може бути використана в якості розвідувальних даних для кібератаки. Додаткові вразливості, такі як слабка автентифікація, відсутність шифрування, недоліки бізнес-логіки та незахищені кінцеві точки, роблять API вразливими до атак

Захист вашого API від атак, описаних вище, повинен базуватися на наступному:

Аутентифікація - визначення особи кінцевого користувача. У REST API базова аутентифікація може бути реалізована за допомогою протоколу TLS, але OAuth 2 і OpenID Connect є більш безпечними альтернативами.
Авторизація - визначення ресурсів, до яких може отримати доступ ідентифікований користувач. API повинен бути побудований і протестований таким чином, щоб запобігти доступу користувачів до функцій або операцій API за межами їх попередньо визначеної ролі. Наприклад, клієнту API, який має доступ лише для читання, не слід дозволяти доступ до кінцевої точки, що надає адміністративні функції.

Додаткові кращі практики включають перевірку викликів API на відповідність схемам API, які чітко описують очікувані структури. Сканування корисного навантаження та перевірка схем може запобігти ін'єкціям коду, деклараціям зловмисних об'єктів та атакам синтаксичного аналізатора. Присвоєння токена API для кожного виклику API перевіряє вхідні запити і запобігає атакам на кінцеві точки.

Нарешті, важливо захистити всі ваші веб-сторінки за допомогою TLS/SSL, які шифрують і аутентифікують дані, що передаються, включаючи ті, що відправляються через веб-API. Це допомагає зменшити загрозу MITM-атак, запобігаючи перехопленню трафіку сайту.

2.2.2 Шифрування даних

Шифрування в кібербезпеці — це перетворення даних із читабельного формату в закодований формат. Зашифровані дані можна прочитати або обробити лише після того, як їх було розшифровано.

Шифрування є основним будівельним блоком безпеки даних. Це найпростіший і найважливіший спосіб гарантувати, що інформація комп’ютерної системи не може бути вкрадена та прочитана кимось, хто хоче використати її для зловмисних цілей.

Шифрування безпеки даних широко використовується окремими користувачами та великими корпораціями для захисту інформації користувачів, яка передається між браузером і сервером. Ця інформація може включати все: від платіжних даних до особистої інформації. Програмне забезпечення для шифрування даних, також відоме як алгоритм шифрування або шифр, використовується для розробки схеми шифрування, яку теоретично можна зламати лише за допомогою великої обчислювальної потужності.

Коли інформація чи дані передаються через Інтернет, вони проходять через ряд мережевих пристроїв по всьому світу, які є частиною загальнодоступного Інтернету. Оскільки дані переміщуються через загальнодоступний Інтернет, існує ймовірність, що вони можуть бути скомпрометовані або викрадені хакерами. Щоб запобігти цьому, користувачі можуть інсталювати спеціальне програмне або апаратне забезпечення для забезпечення безпечної передачі даних або інформації. Ці процеси відомі як шифрування в безпеці мережі.

Шифрування передбачає перетворення зрозумілого людині відкритого тексту в незрозумілий текст, який відомий як зашифрований текст. По суті, це означає взяти читабельні дані та змінити їх так, щоб вони виглядали випадковими. Шифрування передбачає використання криптографічного ключа, набору математичних значень, узгоджених як відправником, так і одержувачем. Одержувач використовує ключ для розшифровки даних, перетворюючи їх назад у читабельний відкритий текст.

Чим складніший криптографічний ключ, тим безпечніше шифрування, оскільки треті сторони менш імовірно розшифрують його за допомогою атак грубої сили (тобто спроби випадкових чисел, доки не буде вгадана правильна комбінація).

Шифрування також використовується для захисту паролів. Методи шифрування паролів кодують ваш пароль, тому хакери його не можуть прочитати.

Найпоширеніші методи шифрування.

Два найпоширеніші методи шифрування — це симетричне та асиметричне шифрування. Назви вказують на те, чи використовується той самий ключ для шифрування та дешифрування:

- Симетричні ключі шифрування: це також відоме як шифрування закритим ключем. Ключ, який використовується для кодування, такий самий, як і для декодування, що робить його найкращим для окремих користувачів і закритих систем. В іншому випадку ключ необхідно надіслати одержувачу. Це підвищує ризик зламу, якщо його перехопить третя сторона, наприклад хакер. Цей спосіб швидше, ніж асиметричний.

- Асиметричні ключі шифрування: у цьому типі використовуються два різні ключі — відкритий і приватний — які пов'язані між собою математично. Ключі — це, по суті, великі числа, які поєднані один з одним, але не є ідентичними, звідси термін асиметричний. Приватний ключ зберігається в таємниці власником, а відкритий ключ або передається авторизованим одержувачам, або надається широкому загалу.

Дані, зашифровані за допомогою відкритого ключа одержувача, можна розшифрувати лише за допомогою відповідного закритого ключа.

Алгоритми шифрування використовуються для перетворення даних у зашифрований текст. Алгоритм використовує ключ шифрування, щоб передбачувано змінювати дані, щоб, навіть якщо зашифровані дані виглядатимуть випадковими, їх можна було перетворити назад у відкритий текст за допомогою ключа дешифрування. Існує кілька різних типів алгоритмів шифрування, призначених для різних цілей. Нові алгоритми розробляються, коли старі стають небезпечними.

2.3 Поняття фронтенду та бекенду

Фронтенд (frontend) та бекенд (backend) - два найпопулярніші терміни, що використовуються у веб-розробці. Фронтенд - це те, що бачать і з чим взаємодіють користувачі, а бекенд - це те, як все працює. Кожна сторона повинна ефективно взаємодіяти та працювати з іншою як єдине ціле для покращення функціональності веб-сайту.

Фронтенд - це частина веб-сайту, яку користувачі можуть бачити і з якою вони можуть взаємодіяти, наприклад, графічний інтерфейс користувача (GUI) і командний рядок, включаючи дизайн, навігаційні меню, тексти, зображення, відео і т.д. Внутрішня частина веб-сайту, навпаки, є тією частиною веб-сайту, яку користувачі не можуть бачити та взаємодіяти з нею. Візуальні аспекти веб-сайту, які користувачі можуть бачити і відчувати, - це фронтенд. З іншого боку, все, що відбувається у фоновому режимі, можна віднести до бекенду.

Мови, що використовуються для фронтенду - HTML, CSS та JavaScript, а для бекенду - Java, Ruby, Python та .Net. Давайте зануримося в ці терміни глибше, щоб краще зрозуміти їх та дізнатися про деякі популярні технології фронтенду та бекенду, які використовуються в наш час.

Фронтенд розробка.

Частина веб-сайту, з якою користувач взаємодіє безпосередньо, називається інтерфейсом. Її також називають "клієнтською частиною програми". Вона включає в себе все, що користувачі відчувають безпосередньо: кольори і стилі тексту, зображення, графіки і таблиці, кнопки, кольори і навігаційне меню. HTML, CSS та JavaScript - це мови, що використовуються для розробки інтерфейсу. Адаптивність і продуктивність - дві основні цілі Front End. Розробник повинен забезпечити, щоб сайт

був адаптивним, тобто коректно відображався на пристроях усіх розмірів, жодна частина сайту не повинна поводитися ненормально незалежно від розміру екрану.

Розробка бекенду.

Бекенд - це серверна частина веб-сайту. Він зберігає і впорядковує дані, а також забезпечує нормальну роботу клієнтської частини веб-сайту. Це частина веб-сайту, яку ви не можете бачити та взаємодіяти з нею. Це частина програмного забезпечення, яка не вступає в прямий контакт з користувачами. Частини і характеристики, розроблені бекенд-дизайнерами, опосередковано доступні користувачам через фронтенд-додаток. Такі види діяльності, як написання API, створення бібліотек і робота з системними компонентами без користувацьких інтерфейсів або навіть систем наукового програмування, також включені в бекенд.

2.4. Поняття веб-додатку

Веб-додаток (веб-додаток) - це прикладна програма, яка зберігається на віддаленому сервері і доставляється через Інтернет через інтерфейс браузера. Веб-сервіси є веб-додатками за визначенням, і багато, хоча і не всі, веб-сайти містять веб-додатки. Будь-який компонент веб-сайту, який виконує певну функцію для користувача, кваліфікується як веб-додаток. [1]

Веб-додатки можуть бути розроблені для найрізноманітніших цілей і можуть використовуватися будь-ким: від організації до приватної особи з багатьох причин. До загальновживаних веб-додатків можна віднести веб-пошту, онлайн-калькулятори або магазини електронної комерції. Деякі веб-додатки можуть бути доступні лише за допомогою певного браузера; однак більшість з них доступні незалежно від браузера.

Веб-додатки не потрібно завантажувати, оскільки доступ до них здійснюється через мережу. Користувачі можуть отримати доступ до веб-додатків через веб-браузер, наприклад, Google Chrome, Mozilla Firefox або Safari.

Для роботи веб-додатку потрібен веб-сервер, сервер додатків і база даних. Веб-сервери керують запитами, які надходять від клієнта, в той час як сервер додатків виконує запитуване завдання. База даних може використовуватися для зберігання будь-якої необхідної інформації.

Веб-додатки, як правило, мають короткий цикл розробки і можуть створюватися невеликими командами розробників. Більшість веб-додатків написані на мовах JavaScript, HTML5 або каскадних таблиць стилів (CSS). Програмування на стороні клієнта зазвичай використовує ці мови, які допомагають створити інтерфейс програми. Програмування на стороні сервера виконується для створення сценаріїв, які буде використовувати веб-додаток. Такі мови, як Python, Java і Ruby, зазвичай використовуються в серверному програмуванні. [4]

Веб-додатки мають багато різних застосувань, а разом з ними і багато потенційних переваг. Деякі загальні переваги веб-додатків включають в себе наступні:

- Надання доступу декільком користувачам до однієї і тієї ж версії програми.
- Веб-програми не потрібно встановлювати.
- Доступ до веб-додатків можна отримати через різні платформи, такі як настільний комп'ютер, ноутбук або мобільний телефон.
- Доступ до них можна отримати за допомогою декількох браузерів.

У секторі мобільних обчислень веб-додатки іноді протиставляють нативним додаткам, тобто додаткам, розробленим спеціально для певної платформи або пристрою і встановленим на цьому пристрої. Однак ці два поняття не є взаємовиключними. Нативні програми (native applications) - це програми, які зазвичай завантажуються і створюються спеціально для певного типу пристрою, на який вони завантажуються. Нативні додатки зазвичай можуть використовувати апаратне забезпечення конкретного пристрою, наприклад, GPS або камеру в мобільному нативному додатку.

Програми, які поєднують ці два підходи, іноді називають гібридними

додатками. Гібридні додатки працюють подібно до веб-додатків, але встановлюються на пристрій як звичайні додатки. Гібридні програми також можуть використовувати ресурси конкретного пристрою, використовуючи внутрішні API. Завантажені нативні додатки іноді можуть працювати в автономному режимі; однак гібридні додатки не мають такої функціональності. Гібридні додатки, як правило, мають схожі елементи навігації з веб-додатками, оскільки вони засновані на веб-додатках.

Висновки до розділу

В процесі роботи було досліджено та проаналізовано методи для збирання власного серверу та способи розгортання на ньому веб-серверу та API. Також поглиблено досліджено використання REST API, та мікрофреймворків для його створення, різноманітних видів шифрування та захисту даних. Для розгортання Web частини додатку було поглиблено знання в мові програмування JavaScirpt з використанням графічної фреймворку React та середовища розробки Visual Studio Code. Також було проведене дослідження щодо пошуків найправильніших шляхів побудови архітектури проекту та написання якісного коду. Загалом сервіс з прокату мобільного електротранспорту використовує найновітніші технології, згадані в даному розділі.

РОЗДІЛ 3. МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Алгоритм кластеризації K-Means

Зі зростанням використання Інтернету в сучасному суспільстві кількість створених даних є незбагненно величезною. Незважаючи на те, що природа окремих даних є простою, величезна кількість даних, що підлягають аналізу, ускладнює їх обробку навіть для комп'ютерів.

Для управління такими процедурами потрібні інструменти аналізу великих обсягів даних. Методи і техніки інтелектуального аналізу даних у поєднанні з машинним навчанням дозволяють нам аналізувати великі обсяги даних у зрозумілий спосіб. k-середні - це метод кластеризації даних, який може бути використаний для неконтрольованого машинного навчання. Вона здатна класифікувати немарковані дані на заздалегідь визначену кількість кластерів на основі схожості (k). Принцип роботи даного алгоритму можна спостерігати на наведеному нижче рисунку (рис. 3.1):

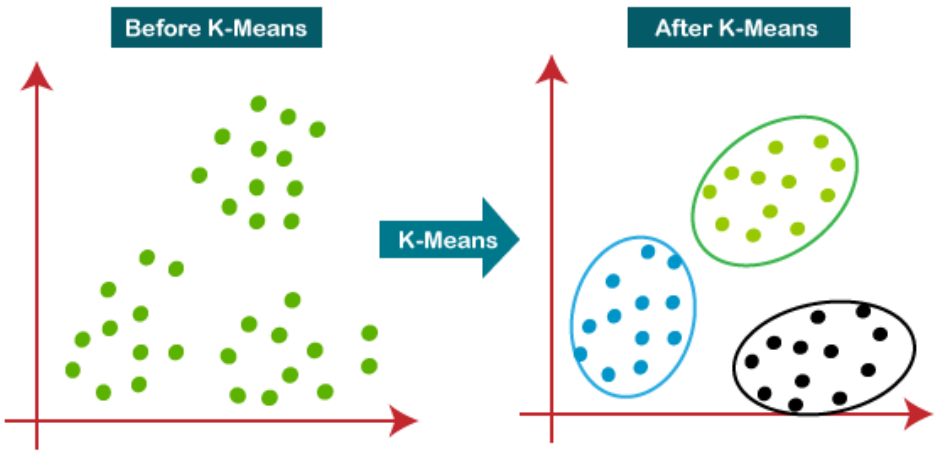


Рисунок 3.1 – Демонстрація роботи алгоритму кластеризації K-Means

Алгоритм кластеризації K-середніх обчислює центроїди і повторюється до тих пір, поки не буде знайдено оптимальний центроїд. Кількість кластерів передбачувано

відома. Він також відомий як алгоритм плоскої кластеризації. Кількість кластерів, знайдених за даними методом, позначається літерою "K" в K-середніх.

У цьому методі точки даних відносяться до кластерів таким чином, щоб сума квадратів відстаней між точками даних і центроїдом була якомога меншою. Важливо зазначити, що зменшення різноманітності всередині кластерів призводить до більшої кількості ідентичних точок даних в межах одного кластера.

Наступні етапи допоможуть нам зрозуміти, як працює метод кластеризації K-Means:

Крок 1: По-перше, нам потрібно вказати кількість кластерів, K, які потрібно згенерувати за допомогою цього алгоритму.

Крок 2: Далі вибираємо випадковим чином K точок даних і відносимо кожну з них до кластеру. Коротко кажучи, класифікуйте дані на основі кількості точок даних.

Крок 3: Тепер будуть обчислені центроїди кластерів.

Крок 4: Повторюйте наведені нижче кроки, поки не знайдете ідеальний центроїд, тобто розподіл точок даних по кластерах, які не відрізняються один від одного.

4.1 Спочатку буде розраховано суму квадратів відстаней між точками даних та центроїдами.

4.2 На цьому етапі необхідно віднести кожну точку даних до кластеру, який є найближчим до інших (центроїд).

4.3 Нарешті, обчислити центроїди для кластерів шляхом усереднення всіх точок даних кластера.

K-середні реалізують стратегію очікування-максимізації для вирішення проблеми. Крок очікування використовується для віднесення точок даних до найближчого кластера, а крок максимізації використовується для обчислення центроїда кожного кластера.

При використанні алгоритму K-середніх необхідно враховувати наступні моменти:

- При роботі з алгоритмами кластеризації, такими як K-середні, рекомендується нормалізувати дані, оскільки такі алгоритми використовують вимірювання на основі відстані для виявлення схожості між точками даних.
- Через ітеративну природу K-середніх та випадкову ініціалізацію центроїдів, K-середні можуть застрягти в локальному оптимумі і не збігатися до глобального оптимуму. Як наслідок, рекомендується використовувати різні ініціалізації центроїдів.

Застосування алгоритму кластеризації K-середніх. Продуктивність кластеризації K-середніх є достатньою для досягнення поставлених цілей. Коли мова йде про наступні сценарії, вона є корисною:

- Для отримання релевантної інформації з даних, з якими ми маємо справу.
- Для різних підгруп будуть створені окремі моделі в рамках підходу "кластеризувати, а потім прогнозувати".
- Сегментація ринку
- Кластеризація документів
- Сегментація зображень
- Стиснення зображень
- Сегментація клієнтів
- Аналіз тренду на динамічних даних

Нижче наведені деякі з переваг алгоритмів кластеризації K-середніх:

- Він простий у розумінні та застосуванні на практиці.
- K-середні будуть швидшими, ніж ієрархічна кластеризація, якщо ми маємо велику кількість змінних.
- Кластер екземпляра може бути змінений при повторному обчисленні центроїдів.

- У порівнянні з ієрархічною кластеризацією, K-середні створюють більш щільні кластери.

Деякі з недоліків методу кластеризації K-середніх є наступними:

- Кількість кластерів, тобто значення k, важко оцінити.
- Значний вплив на результат мають початкові вхідні дані, такі як кількість кластерів у мережі (значення k).
- Послідовність, в якій вводяться дані, має значний вплив на кінцевий результат.
- Він досить чутливий до зміни масштабу. Якщо ми перемасштабуємо наші дані, використовуючи нормалізацію або стандарти, кінцевий результат буде кардинально відрізнятися.
- Не рекомендується виконувати завдання кластеризації, якщо кластери мають складну геометричну форму.

Кожен інженер з машинного навчання хоче, щоб його алгоритми робили точні прогнози. Такі алгоритми навчання часто класифікуються як контрольовані або неконтрольовані. Кластеризація K-середніх є неконтрольованою технікою, яка не вимагає маркованої відповіді на задані вхідні дані.

Кластеризація за методом K-середніх є широко використовуваним підходом для кластеризації. Як правило, фахівці починають з вивчення архітектури набору даних. K-середні кластеризують точки даних в унікальні групи, що не перетинаються. Він працює дуже добре, коли кластери мають сферичну форму. Однак він страждає від того, що геометричні форми кластерів відхиляються від сферичної форми.

Крім того, він не дізнається кількість кластерів з даних і потребує, щоб вона була вказана заздалегідь. Завжди корисно розуміти припущення, що лежать в основі алгоритмів/методів, щоб краще зрозуміти сильні та слабкі сторони кожного методу. Це допоможе вам визначити, коли і за яких умов використовувати кожну форму.

Висновки до розділу

Алгоритм кластеризації K-Means є безсумнівно потужним інструментом для аналізу великих масивів даних. Цей метод широко використовується в багатьох сферах діяльності для кластеризації даних, але як і будь-який алгоритм, він не досконалий і відповідно не позбавлений мінусів, тому перед його використанням потрібно детально проаналізувати інформацію про нього. Розставивши всі «за» і «проти», я вирішив використати даний алгоритм в своїй дипломній роботі для знаходження ділянок з найактивнішим використанням орендованого транспорту. Таким чином, буде легко визначити ділянки з найбільшим попитом на послуги нашого сервісу. Відповідно, відштовхуючись від цих даних, розставляти самокати в потрібних місцях, що в свою чергу значно збільшить прибуток, бо одне з правил якісного ведення бізнесу, це породжувати пропозицію, там де є попит

РОЗДІЛ 4. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

4.1 Мікрофреймворк для веб розробки Flask

Flask - це мікрофреймворк для веб-розробки на мові Python. Мікрофреймворк означає, що він поставляється з дуже невеликим набором доступних функцій і шаблонного коду на початку розробки проекту. Це не означає, що для фреймворку Flask не існує бібліотек функцій і плагінів, але за замовчуванням Flask, як правило, не включає функції, якщо ви, як розробник, не вказали інше.

Це суттєво відрізняється від таких фреймворків, як Rails або Django (останній є ще одним дуже популярним веб-фреймворком Python), де велика кількість функцій і зручностей генерується для вас при створенні проекту. Наприклад, обидва ці фреймворки надають рівень абстракції бази даних, який дозволяє розробникам легко читати і записувати в бази даних через об'єктні моделі (наприклад, Rails через Active Model, Django через їх об'єктно-реляційний маппер).

Недоліком мікрофреймворку, очевидно, є відсутність цих функцій на початковому етапі. Багато, якщо не всі, функції більш широких фреймворків існують у бібліотеці Flask, але їх потрібно включати явно. Однак головна перевага полягає в тому, що якщо ви не збираєтеся використовувати ці функції, набагато простіше просто створити додаток на Flask і уникнути необхідності видаляти багато непотрібних файлів і залежностей. Додатки на мікрофреймворку мають набагато менший об'єм коду і завжди можуть бути масштабовані за допомогою додаткових інструментів за потреби.

У моєму випадку мікрофреймворк Flask виявився набагато простішим в реалізації, і надав мені гнучкість для включення додаткових модулів Python за необхідності.

Налаштування flask-сервера виявилося напрочуд простим. Перш за все, ви встановлюєте модуль flask в обраний вами менеджер середовища python. В

активованому середовищі створюєте новий файл (за замовчуванням Flask очікує, що він буде називатися app.py) і в ньому вводите наступне:

```
1  from flask import Flask
2  app = Flask(__name__)
3
4  @app.route('/')
5  def hello_world():
6      return 'Hello, World!'
```

Рисунок 4.1 – Налаштування flask-сервера

І це все! У шести рядках ми написали достатньо коду для працюючого Flask-сервера. Ось так просто. Набравши в командному рядку "flask run" в цей момент, ви завантажите сервер з єдиним маршрутом в кореневому домені, який повертає рядок "Hello, World!". Flask інтерпретує запит маршруту і запускає функцію, написану для цього маршруту. Значення, що повертається функцією, є тим, що повертається у відповідь на URL-запит, але ми розглянемо відповіді на запити набагато пізніше.

Шестирядковий сервер в першу чергу говорить про потужність Flask як мікрофреймворку; ми можемо створити працюючий сервер за допомогою надзвичайно мінімальної кількості коду і доповнювати його по мірі необхідності.

4.2 JSON Web Tokens

JWT або JSON Web Tokens найчастіше використовуються для ідентифікації автентифікованого користувача. Вони видаються сервером автентифікації і використовуються клієнт-сервером (для захисту його API). JSON Web Token - це відкритий галузевий стандарт, який використовується для обміну інформацією між двома об'єктами, як правило, клієнтом (наприклад, інтерфейсом вашого додатку) і сервером (бекендом вашого додатку).

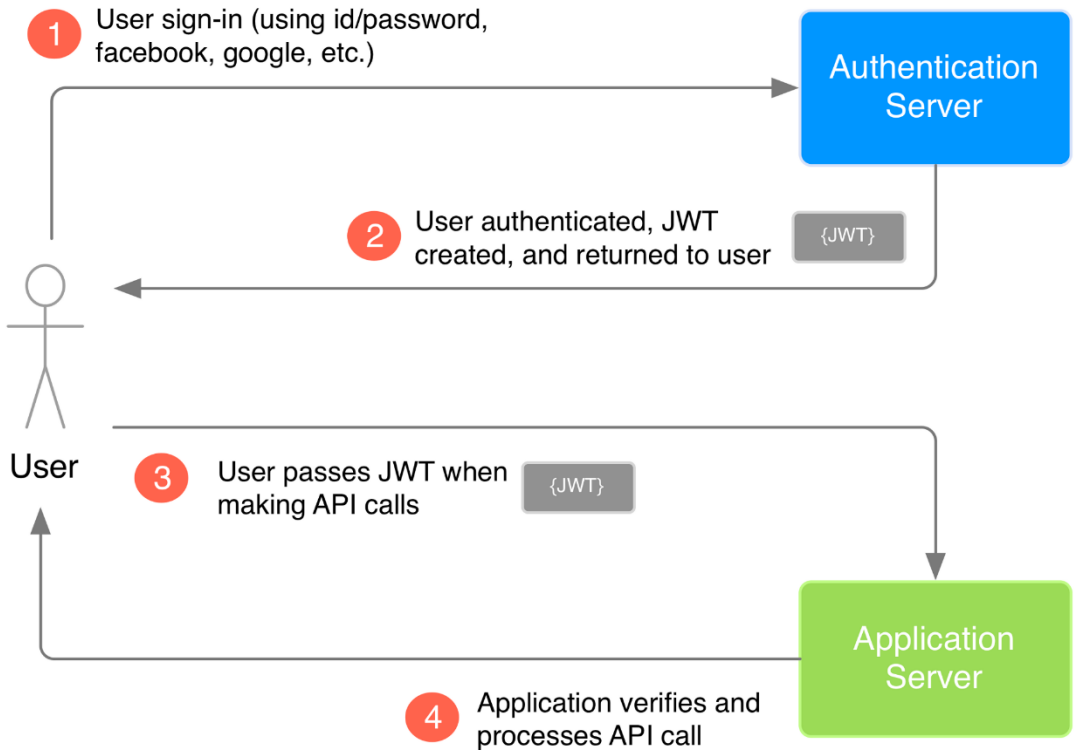


Рисунок 4.2 – Схема використання JSON Web Token (JWT)

Вони містять об'єкти JSON, які в свою чергу містять інформацію, якою потрібно обмінюватися. Кожен JWT також підписується за допомогою криптографії (хешування), щоб гарантувати, що вміст JSON (також відомий як твердження JWT) не може бути змінений клієнтом або зловмисником.

Вам може бути цікаво, чому сервер авторизації не може просто відправити інформацію у вигляді простого об'єкта JSON і чому він повинен перетворити її в "токен". Якщо сервер аутентифікації відправляє інформацію у вигляді простого JSON, API клієнтських додатків не матимуть можливості перевірити правильність вмісту, який вони отримують. Зловмисник може, наприклад, змінити ідентифікатор користувача, і API програми не матиме можливості дізнатися, що це сталося.

Через цю проблему безпеки сервер авторизації повинен передавати цю інформацію таким чином, щоб її можна було перевірити клієнтською програмою, і саме тут з'являється концепція "токена".

Простіше кажучи, токен - це рядок, що містить деяку інформацію, яку можна безпечно перевірити. Це може бути випадковий набір алфавітно-цифрових символів,

які вказують на ідентифікатор в базі даних, або це може бути закодований JSON, який може бути самостійно перевірений клієнтом (відомий як JWT).

Структура JWT.

JWT складається з трьох частин:

- ***Заголовок***: Складається з двох частин:
  - Алгоритм підпису, який використовується.
  - Тип токена, який, в даному випадку, здебільшого є "JWT".
- ***Корисне навантаження:*** Корисне навантаження містить вимоги або JSON-об'єкт.
- ***Підпис:*** Рядок, який генерується за допомогою криптографічного алгоритму, який може бути використаний для перевірки цілісності корисного навантаження JSON.

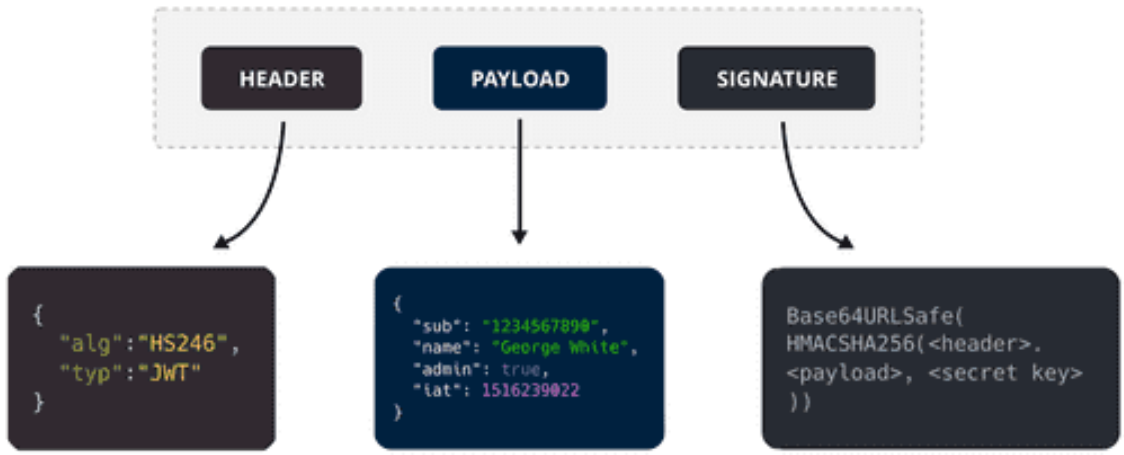


Рисунок 4.3 – Структура JSON Web Token (JWT)

Використання JWT має чимало переваг:

- Безпека: JWT підписуються цифровим підписом з використанням або секретного (HMAC), або відкритого/закритого ключа (RSA або ECDSA), що захищає їх від модифікації клієнтом або зловмисником.

- Зберігаються тільки на клієнті: Ви генеруєте JWT на сервері та надсилаєте їх клієнту. Потім клієнт надсилає JWT з кожним запитом. Це економить місце в базі даних.

- Ефективно: Перевірка JWT відбувається швидко, оскільки вона не вимагає пошуку в базі даних. Це особливо корисно у великих розподілених системах.

Однак є деякі недоліки:

- Невідкличність: Через їхню автономну природу та процес перевірки без громадянства може бути важко відкликати JWT до закінчення терміну його дії природним чином. Тому такі дії, як негайна заборона користувача, не можуть бути легко реалізовані. З огляду на це, існує спосіб підтримувати заборону/чорний список JWT, і завдяки цьому ми можемо негайно відкликати їх.

- Залежить від одного секретного ключа: Створення JWT залежить від одного секретного ключа. Якщо цей ключ скомпрометований, зловмисник може сфабрикувати власний JWT, який буде прийнятий рівнем API. Це, в свою чергу, означає, що якщо секретний ключ скомпрометований, зловмисник може підробити ідентифікаційні дані будь-якого користувача. Ми можемо зменшити цей ризик, час від часу змінюючи секретний ключ.

Підводячи підсумок, JWT є найбільш корисним для великомасштабних додатків, які не вимагають таких дій, як негайне блокування користувача.

4.3 Графова база даних DataBase Neo4j

Neo4j є провідною графовою базою даних. Вона пропонується на комерційній основі, повністю підтримується і має відкритий вихідний код. Створена в 2007 році, Neo4j - це база даних No-SQL, яка базується на Java, не обов’язково потребує схеми і

широко масштабована. Так що ж таке Neo4j? Вона зазвичай розглядається як найкраща графова база даних, доступна на сьогоднішній день.

По суті, графові бази даних зберігають дані у вигляді графів. Граф - це математична концепція, яка класифікує елементи з точки зору вершин (вузлів) і ребер (зв'язків) для розуміння зв'язків і закономірностей у досліджуваній інформації. При використанні графових баз даних, таких як Neo4j, ці графи часто представляються візуально.

Графові бази даних є відносно новим класом баз даних, які використовуються для випадків використання, які особливо зосереджені на зв'язках всередині даних. Іншими словами, хоча бази даних графів зберігають дані, такі як вузли та ребра, вони більше зосереджуються на взаємозв'язках, які часто приховані серед багатьох елементів у масивах даних. У графовій базі даних зв'язки є першокласними елементами, поряд з об'єктами даних.

Хоча Neo4j часто називають безсхематичною, кращим терміном є "необов'язкова схема", оскільки схема може бути використана (хоча і не є обов'язковою).

Ми живемо у світі, який стає дедалі більш взаємопов'язаним. Як наслідок, наші дані також стають все більш пов'язаними. Враховуючи обсяг даних, які виробляються в усьому світі щодня, цінність взаємозв'язків всередині даних швидко стає більш цінною, ніж самі дані. Унікальна цінність графових баз даних полягає в їх здатності виявляти нові взаємопов'язані знання, як в природному вигляді, так і в масштабі, у вигляді аналітичних висновків, які мають суттєвий вплив на бізнес та інші організації.

Основні переваги Neo4j (графової бази даних).

- ***Першопроходець.*** Neo4j більш широко прийнята на ринку, ніж будь-яке інше рішення. Засновник Neo4j Еміль Ефрейм (Emil Efreim) фактично ввів термін "база даних графів".

- ***Спільнота.*** Neo4j має процвітаючу спільноту користувачів, активні форуми, глибоку документацію та ресурси для будь-яких питань, пов'язаних з базами даних графів.

- ***Продуктивність.*** Neo4j є однією з небагатьох справжніх нативних баз даних графів, яка дозволяє безіндексну суміжність для значного підвищення продуктивності.

- ***Доступність.*** Від масивних додатків в режимі реального часу до аналітично орієнтованих додатків для машинного навчання на основі графів і графічних даних, Neo4j встановлює планку для задоволення вимог HA.

- ***Сумісність з ACID***. Продуктивність Neo4j як при читанні, так і при записі була наочно продемонстрована в масштабах підприємства. Він також підтримує цілісність завдяки справжній відповідності стандарту ACID, якого все ще не вистачає більшості інших пропозицій.

- ***Легкий доступ.*** Взаємодіяти з Neo4j легко, будь то через інтерфейс Neo4j Browser UI з мовою запитів Cypher (що набагато простіше, ніж альтернативи, такі як Gremlin) або через Java API.

- ***Неструктуровані / напівструктуровані / текстові дані.*** Отримання цінності з абсолютно величезного і зростаючого обсягу неструктурованих текстових даних, доступних сьогодні, ніколи не було простим завданням. Враховуючи унікальну здатність графіків вбудовувати значення і пов'язувати концепції, Neo4j є ідеальним рішенням для досягнення інсайту. Крім того, Neo4j є лідером в обробці природної мови (NLP) за допомогою графів.

- ***Наука про графічні дані.*** Neo4j є комерційним лідером у галузі науки про дані за допомогою графів, включаючи випадки використання NLP. Враховуючи, що Google стверджує, що майбутнє науки про дані будується навколо теорії мереж / баз даних графів, це є значною перевагою для Neo4j.

4.4 Cloudflare

Cloudflare - американська компанія, яка надає такі послуги, як DNS, мережа доставки контенту (CDN) та багато інших додаткових послуг, щоб зробити веб-сайти швидшими та безпечнішими. Послугами Cloudflare користуються понад 26 мільйонів сайтів, в результаті чого обробляється понад 1 мільярд IP-адрес щодня. Звучить дуже багато, і це дійсно так. Але чому Cloudflare настільки популярний? Чому ми, а також багато інших компаній, використовуємо Cloudflare?

Існує багато сервісів, які пропонує Cloudflare, в тому числі наступні:

- Мережа доставки контенту (CDN)
- Система доменних імен (DNS)
- Балансування навантаження
- Прискорені мобільні сторінки (AMP)
- Можливості кешування
- Потокове відео
- DDoS-захист
- Брандмауер веб-додатків (WAF)
- SSL/TLS-підтримка
- DNSSEC
- Аналітика
- Реєстратор доменів
- Робочі (для розробників)

Окрім своїх сервісів Cloudflare має стандартну інтеграцію з такими відомими платформами як IBM Cloud, WordPress, Google Cloud, Magento та Kubernates.

Для того, щоб зрозуміти, як працює Cloudflare, необхідно розглянути найпоширеніші помилки, які допускали сайти в минулому. При відвідуванні сайту

без Cloudflare відвідувач сайту запитує контент з сервера. Однак, коли на сервері було занадто багато відвідувачів одночасно, сервер перевантажувався з наслідками повільного або неробочого веб-сайту. Як власник веб-сайту, це не те, що ви хотіли б бачити. Саме тому Cloudflare винайшов рішення для цього.

Cloudflare розмістила власний сервер у вигляді величезної всесвітньої мережі між веб-сайтом та веб-сервером. Відвідувачі веб-сайту більше не спілкуються безпосередньо з сервером, а з мережею Cloudflare, яка вже зберігає вміст веб-сайту і завантажує його через сервер в залежності від місцезнаходження відвідувача. Наприклад: У вас є веб-сайт, який розміщений в Німеччині, і хтось із США намагається підключитися до вашого веб-сайту. Запит повинен подолати велику відстань (між веб-сайтом і веб-сервером). Cloudflare вирішує цю проблему, пропонуючи величезну мережу серверів по всьому світу. Американцеві більше не потрібно встановлювати з'єднання з німецьким сервером, він з'єднається з найближчим сервером Cloudflare десь у Сполучених Штатах.

Рисунок 4.4 – Місцерозташування серверів Cloudflare по всьому світу

Таким чином, Cloudflare розвантажує сервери і гарантує, що відвідувачі зможуть швидше побачити запитувану ними веб-сторінку.

Як вже згадувалося раніше, Cloudflare пропонує великий спектр послуг. Всі ці послуги сприяють виконанню місії компанії: допомагати будувати кращий Інтернет.

Cloudflare DNS

Cloudflare пропонує свій DNS-сервіс, який забезпечує, за її власними словами, найшвидший час відгуку (reaction time), неперевершену надмірність і передові заходи безпеки, такі як інтегрований DDoS-блокатор і DNSSEC. Завдяки всесвітньо розподіленій мережі Cloudflare, яка налічує понад 200 серверів, забезпечується безперебійність роботи на рівні 100%. Перенісши свої DNS до Cloudflare та використовуючи його SSL-сертифікати, ви автоматично отримуєте їх WAF (Web Application Firewall).

Оскільки Cloudflare є найбільшою компанією, вона сканує найбільшу кількість IP-адрес у всьому світі. Тому вони найшвидше виявляють всі IP-адреси з поганими намірами, які можуть бути негайно заблоковані.

DDoS-атак неможливо уникнути, проте їх можна зробити нешкідливими, блокуючи IP-адреси/діапазони або навіть цілі регіони.

Брандмауер веб-додатків (WAF)

WAF - це рішення для кожної організації, яка хоче захистити свій веб-сайт або інший додаток від кібератак без внесення змін в існуючу інфраструктуру. Cloudflare дозволяє додавати правила через WAF-панель, щоб бути захищеними ще краще.

Кожен запит на вхід в WAF буде перевірятися за правилами, які ви встановите в механізмі правил. Підозрілі запити будуть вчасно заблоковані та зафіксовані за потребами користувача, в той час як "хороші" запити будуть направлені за призначенням. Простим налаштуванням, яке можна зробити в WAF, є включення базового набору правил OWASP. Це дає вам стандартний захист від атак, які мають відношення, наприклад, до SQLi і XSS.

Cloudflare CDN

Раніше вже було згадано про функцію CDN. Мережа доставки контенту Cloudflare - це найбільша послуга, яку вони пропонують. Перевага цієї функції CDN полягає в тому, що ця функція дозволяє відвідувачеві спілкуватися з найближчим сервером Cloudflare. Це забезпечує більш швидку роботу веб-сайтів. Абсолютний must-have для сайтів, орієнтованих на міжнародний ринок.

CDN від Cloudflare пропонує ще більше переваг. Є вбудований модуль кешування. Цей модуль кешування гарантує, що не кожна відвідана сторінка повинна бути завантажена сервером знову.

Шифрування SSL/TLS

Так, коли ви використовуєте Cloudflare, ви також отримуєте це. Рівень захищених сокетів (SSL) - це стандартний протокол, який використовується для встановлення захищеного з'єднання.

Сьогодні кожен веб-сайт повинен мати SSL-сертифікат. Компанія Google зазначила, що сайти, які не мають дійсного SSL-сертифікату, страждають від цього в пошуковій видачі. Крім того, відвідувачі такого сайту матимуть згадку про те, що сайт не є захищеним. Ви, як власник сайту, можете отримати SSL-сертифікат через сервіс Cloudflare. Безумовно, це робить безпосередній внесок у виконання місії: побудувати кращий інтернет.

Як висновок, можна підсумувати, що Cloudflare є безпечним, надзвичайно стабільним, забезпечує кращу продуктивність вашого веб-додатку і не є дорогим або навіть безкоштовним. Cloudflare захищає вас від DDoS-атак, має надзвичайно швидкий DNS-сервіс та широкі можливості кешування.

4.5 Модуль AsyncIO

AsyncIO дозволяє використовувати асинхронне програмування з паралелізмом на основі процедур на мові Python. Хоча асинхронність доступна в Python вже багато

років, вона залишається однією з найцікавіших і в той же час однією з найнеприємніших областей Python. Новим розробникам просто важко почати працювати з асинхронністю.

В широкому розумінні під асинхронністю розуміється можливість реалізації асинхронного програмування на мові Python за допомогою підпрограм (coroutines). Конкретно, йдеться про два елементи:

- Додавання модуля "asyncio" до стандартної бібліотеки Python у версії Python 3.4.
- Додавання виразів async/await до мови Python у версії Python 3.5.

Разом модуль та зміни в мові полегшують розробку програм на Python, які підтримують паралелізм на основі підпрограм, неблокуючий ввід/вивід та асинхронне програмування.

Модуль "asyncio" надає функції та об'єкти для розробки програм на основі підпрограм з використанням парадигми асинхронного програмування. Зокрема, підтримується неблокуючий ввід/вивід з підпроцесами (для виконання команд) та з потоками (для програмування TCP-сокетів).

Центральним елементом модуля асинхронізації є цикл обробки подій. Це механізм, який запускає програму на основі підпрограм і реалізує кооперативну багатозадачність між підпрограмами.

Модуль надає як високорівневий, так і низькорівневий API. Високорівневий API призначений для нас, розробників додатків на Python. Низькорівневий API в більшості випадків призначений для розробників фреймворків, а не для нас. Більшість варіантів використання задовольняється за допомогою високорівневого API, який надає утиліти для роботи з підпрограмами, потоками, примітивами синхронізації, підпроцесами та чергами для обміну даними між підпрограмами.

Низькорівневий API є основою для високорівневого API і включає в себе внутрішню частину циклу обробки подій, транспортні протоколи, політики та інше.

Існує, мабуть, 3 причини найвищого рівня для використання асинхронізації в Python-проекті. Ось вони:

- Використання асинхронізації для впровадження підпрограм у вашій програмі.
- Використання асинхронізації для використання парадигми асинхронного програмування.
- Використання асинхронізації для використання неблокуючого вводу/виводу.

Іноді ми маємо контроль над функціональними та нефункціональними вимогами, а іноді ні. У випадках, коли ми маємо контроль, ми можемо вибрати використання асинхронного виконання по одній з причин, перерахованих вище. У випадках, коли ми не маємо контролю, ми можемо бути змушені обрати асинхронну роботу для того, щоб створити програму, яка вирішує конкретну проблему.

4.6 Бібліотека React

React.js був випущений інженером-програмістом, що працює в компанії Facebook - Джорданом Уоке в 2011 році. React - це бібліотека JavaScript, орієнтована на створення декларативних користувацьких інтерфейсів (UI) з використанням компонентної концепції. Вона використовується для роботи з шаром представлення і може застосовуватися для веб- та мобільних додатків. Основна мета React - бути широким, швидким, декларативним, гнучким і простим.

React - це не фреймворк, а саме бібліотека. Пояснюється це тим, що React займається тільки рендерингом інтерфейсів і залишає багато речей на розсуд окремих проектів. Стандартний набір інструментів для створення додатку з використанням ReactJS часто називають стеком. [5]

Розглянемо більш детально, що ж виділяє бібліотеку React на фоні інших фреймворків та бібліотек і робить її такою потужною та популярною для розробки

додатків.

Віртуальна модель об'єктів документа (VDOM).

Document Object Model (DOM) - це API для коректних HTML та добре сформованих XML документів. Віртуальний DOM - це представлення реального DOM, яке створюється/маніпулюється браузерами. Просунуті бібліотеки, такі як React, генерують дерево елементів в пам'яті, еквівалентне реальному DOM, яке формує віртуальний DOM декларативним способом. Віртуальний DOM є однією з особливостей, які роблять фреймворк таким швидким і надійним.

JSX.

React використовує синтаксичне розширення JavaScript, яке називається JSX. Ми використовуємо його для створення "елементів". JSX використовує препроцесори Babel для перетворення HTML-подібного тексту в JavaScript-файлах в JavaScript-об'єкти для аналізу. React не вимагає використання JSX, але більшість розробників вважають, що він робить код JavaScript більш зручним для користувача. Ми використовуємо JSX для створення React-компонентів, тому він є важливою частиною ReactJS.

React Native.

React Native - це JavaScript-фреймворк з відкритим вихідним кодом для створення додатків на різних платформах, таких як iOS, Android та UPD. Він базується на React і віддає всю свою велич розробці мобільних додатків. React Native використовує JavaScript для побудови інтерфейсу користувача програми, але також використовує власні представлення ОС. Це дозволяє реалізовувати код на мовах, нативних для ОС (Swift та Objective-C для iOS та Java і Kotlin для Android) для більш складних функцій.

Основні компоненти.

ReactJS - це бібліотека на основі компонентів, де компоненти роблять наш код

багаторазовим і розбивають наш інтерфейс на різні частини. Компоненти поділяються на два типи: компоненти класів та функціональні компоненти. Всі React-компоненти слідують принципу розділення проблем, що означає, що ми повинні розділити наш додаток на різні секції для вирішення окремих проблем.

Функціональні компоненти.

React-компоненти працюють подібно до функцій JavaScript. Компонент отримує випадкові вхідні дані, які ми називаємо пропсами, і завжди повинен повертати React-елемент, який визначає те, що має бути відображено користувачеві.

Компоненти класів

Компонент Class повинен мати інструкцію extends `React.Component`. Цей оператор створює підклас `React.Component`, який дозволяє вашому компоненту отримати доступ до функцій `React.Component`.

Отже, головне питання - чому варто обрати ReactJS як стек для фронтенд-розробки, коли існує безліч інших. Ось кілька причин:

- ***Швидкісний***. React дозволяє розробникам використовувати окремі частини свого додатку як на стороні клієнта, так і на стороні сервера, і будь-які внесені ними зміни не вплинуть на логіку роботи програми. Це робить процес розробки надзвичайно швидким.

- ***Підтримка компонентів.*** Використання HTML-тегів і JS-кодів дозволяє легко працювати з величезним набором даних, що містить DOM. React виступає в ролі посередника, який представляє DOM і допомагає вирішити, який компонент вимагає змін для отримання точних результатів.

- ***Простий у використанні та вивченні.*** ReactJS неймовірно зручний для користувача і робить будь-який UI інтерактивним. Він також дозволяє швидко і

ефективно створювати додатки, що економить час як для клієнтів, так і для розробників.

- ***Дружній до SEO.*** Поширеною проблемою, на яку скаржаться більшість веб-розробників, є те, що традиційні JavaScript-фреймворки часто мають проблеми з SEO. ReactJS вирішує цю проблему, допомагаючи розробникам легко орієнтуватися в різних пошукових системах завдяки тому, що ReactJS-додаток може працювати на сервері, а віртуальний DOM рендерить і повертає його браузеру у вигляді веб-сторінки.

- ***Одностороннє зв'язування даних.*** Одностороння прив'язка даних має на увазі, що абсолютно будь-яка людина може відстежити всі зміни, які були внесені в сегмент даних. Це також одна з причин, яка робить React таким простим.

4.7 Google maps API

Карти Google - це провідна платформа для візуалізації даних про місцезнаходження в Інтернеті. Ви можете використовувати її для того, щоб знайти найшвидший шлях до будинку вашого друга, яким автобусом доїхати до роботи або де взяти їжу о 2 годині ночі.

Однак функціональність Карт Google виходить далеко за рамки мобільного додатку - якщо ви маєте бізнес-сайт або додаток і хочете будь-яким чином включити відображення місцезнаходження, Карти Google пропонують API для розробників, які надають доступ до безлічі географічних даних Google.

За допомогою Google Maps API ви можете відображати інтерактивні карти та налаштовувати їх на своєму веб-сайті як завгодно. Якщо у вас туристичний сайт, ви можете використовувати карти, щоб допомогти користувачам створювати маршрути. Служба доставки або спільного використання автомобілів може використовувати Карти Google для показу маршрутів водіїв. Дані Карт Google також оновлюються в

режимі реального часу, а це означає, що карти, які ви створюєте за допомогою Google API, завжди будуть актуальними для відвідувачів.

Наразі платформа карт Google пропонує декілька API для різних аспектів свого сервісу. Існує Maps Static API для простих вбудовувань Карт Google, Maps JavaScript API для інтерактивних і настроюваних карт, Places API для доступу до даних про об'єкти інтересу, а також Directions API для прокладання маршрутів до місця розташування, і це лише деякі з них.

Ви, ймовірно, не будете використовувати всі API Карт Google на своєму веб-сайті, але вам може знадобитися кілька інтеграцій API залежно від того, які функції ви шукаєте. Зареєструвавшись для використання API, ви отримаєте доступ до всіх цих API, а також до інших інструментів для розробників.

4.8 Розробка backend частини

4.8.1 Встановлення та налаштування графової бази даних NEO4J

Проаналізувавши різні варіанти програм, бібліотек та фреймворків, було визначено основний набір програмного забезпечення, необхідного для роботи над дипломним проектом. Першим кроком після цього, постало питання запуску та настройки бази даних. В якості технічного забезпечення, під час роботи використовувався комп’ютер з процесором Intel core i7-1165g7 з вісьмома потоками, та 16 Гб оперативної пам’яті. В якості операційної системи, було обрано Linux, в якості дистрибутиву, слугував Ubuntu. Саме на цьому апаратному забезпеченні було запущено сервер. В якості мережевого протоколу для управління сервером, було обрано SSH.

На рисунку 4.5 можна побачити процес підключення до сервера та авторизації за допомогою ssh.

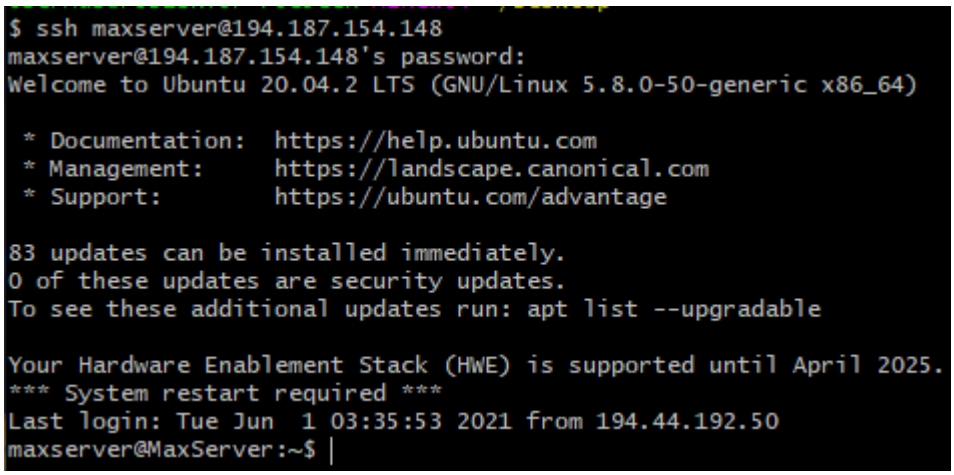


Рисунок 4.5 – Процес підключення до сервера за допомогою ssh

Наступним кроком було отримано права суперкористувача, використовуючи команду ***sudo -i***.

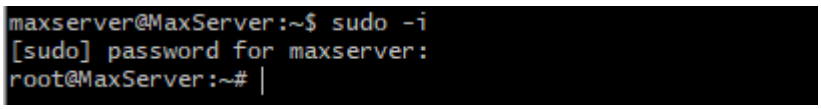


Рисунок 4.6 – Процес отримання прав суперкористувача

Далі за допомогою менеджера пакунків (Advanced Packaging Tool) операційних систем Debian, було оновлено базу даних з доступними для становлення пакетами. **apt update**

Та встановлено базу даних NEO4J за допомогою установки відповідного пакету **apt install neo4j**.

Встановивши необхідний пакет, наступним кроком був процес налаштування бази даних. Усі маніпуляції з налаштуванням бази даних проводилися в файлі **neo4j.conf**.

В якості засобу для редагування текстових файлів, було використано консольний текстовий редактор для UNIX-подібних операційних систем – **nano**. Команда для редагування виглядає наступним чином: **nano /etc/neo4j/neo4j.conf**. В

якості внесених змін, змінній **dbms.default_listen_address** було надано значення **0.0.0.0**, що було необхідним для отримання можливості керування базою даних.

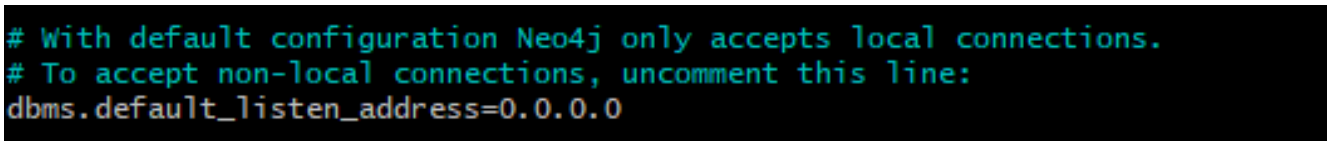


Рисунок 4.7 – Редагування IP адреси для графової бази даних NEO4J

Було створено базу даних та юзера neo4j, щоб отримати доступ до сховища інформації через протокол BOLT, та також за допомогою браузера. Використано команду **cypher-shell** та надано користувачу пароль. Перезапуск бази даних відбувається внаслідок виконання команди **sudo service neo4j restart**, перезапуск необхідний для того, щоб налаштування вступили в силу. Перевірку статусу було виконано командою **service neo4j status.**

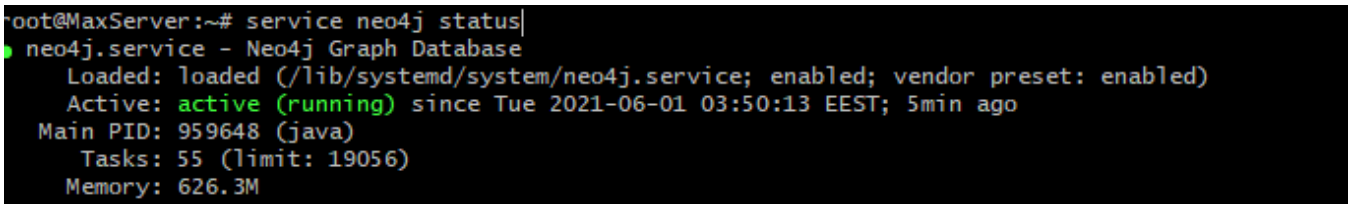


Рисунок 4.8 – Вивід команди «service neo4j status», який включає в собі статус бази даних

Після даних операцій, стало можливо пройти аутентифікацію за посиланням http://194.187.154.148:7474/browser/, та підключення до бази через протокол BOLT та NEO4J.

Завдяки проведеним вище маніпуляціям, було отримано змогу залогінитися за адресом http://194.187.154.148:7474/browser/. В свою чергу, після авторизації, появляється можливість приєднання до бази даних, використовуючи вже вищезгаданий протокол (BOLT) та NEO4J.

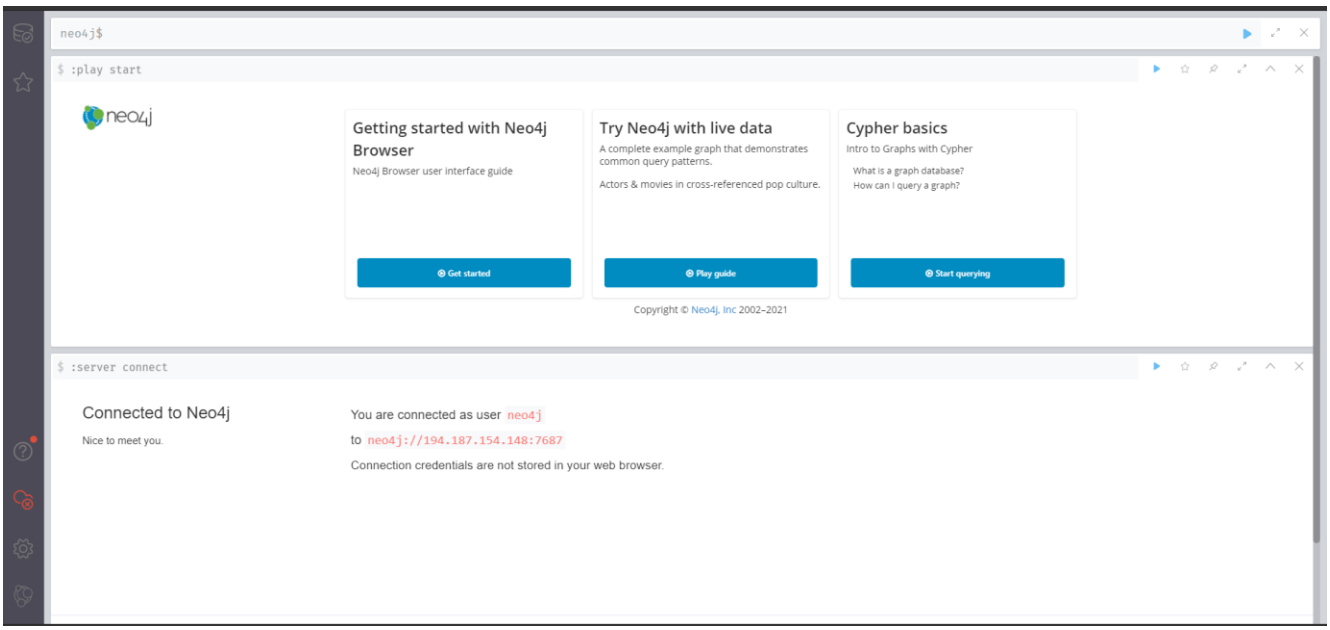


Рисунок 4.9 – Графічний інтерфейс бази даних

4.8.2 Встановлення бібліотек

Після проведення планування роботи та аналізу, було складено список необхідних бібліотек для роботи над проектом. Звичайно, список поповнювався в ході роботи, адже все наперед продумати неможливо. Остаточний набір необхідних python-бібліотек виглядає наступним чином:

- aioflask
- flask_restful
- flask_jwt_extended
- flask_bcrypt
- flask_bootstrap
- flask_cors
- uuid
- py2neo
- qrcode
- googlemaps
- asyncio
- yaml
- json
- typing
- datatime

Для встановлення бібліотек у python, використовується команда **pip install «library-name»**. Помимо сторонніх бібліотек, під час написання проекту, також було використано нативні пайтонівські бібліотеки.

4.8.3 Створення базової структури проекту

Особливістю мікрофреймворка Flask є те, що він не ставить вимоги перед якоюсь певною структурою проекту, тому це робить розробку більш гнучкою. Отже завдяки цьому структура проекту організована відповідно до моїх вподобань, з метою збільшення комфорту роботи над ним, та в разі потреби, внесення корективів в майбутньому та доопрацювання.

В якості основної папки проекту було створено нову папку та присвоєно ім’я **bitroller.** Саме тут будуть розташовуватися всі проектні файли та папки. В основній папці, зразу можна побачити два файли: **config.yml** та **init.py. config.yml** являє собою файл конфігурації проекту, тим часом **init.py** відповідає за запуск проекту. Також бачимо каталог **app**, перейдемо в нього.

Знаходячись вже в папці **app**, можемо побачити ще різноманітні файли та папки. Одним з таких каталогів є **admin_api**, в якому знаходяться файли, які відповідають за панель адміністрування API. Реалізація даної панелі адміністратора відбувалася з використанням jinja2. Зокрема у каталозі **static** ми також можемо побачити файли користувачів та файли з даними про транспорт, зокрема створені QR-коди, які відповідають кожній одиниці транспорту. У папці **templates** знаходиться html-файл зі сторінкою панелі адміністрування.

Перейшовши в каталог **database**, можна зустріти файли, які відповідають за роботу з базою даних, в моєму випадку з графовою базою даних NEO4J. Для приєднання до бази даних використовується файл **connection.py,** тим часом інший файл **database.py** слугує для коректної взаємодії з нею.

Відповідно до назви каталогу, в **models** знаходяться файли з налаштуванням структури даних, для збереження в NEO4J.

Також в каталозі а**pp** можемо зустріти файл **config.py**, який має в собі функції, необхідні для обробки певних настройок.

Файл **config.py** відповідає за ініціалізацію конфігураційного файлу, та містить у собі функції обробки деяких налаштувань.

Основним файлом, на мою думку, є файл з маршрутами (routes) API, цей файл також знаходиться в даному каталозі і має назву **webports.py**.

Звернімо увагу також і на додаткові файли проекту. До них я відношу **saferequest.py** та **app_func.py**. Додатковими я їх вважаю, тому що вони відповідають за додатковий функціонал, зокрема **app_func.py** за розрахунок вартості поїздки та очищення даних, тим часом **saferequest.py** призначений для обробляння запитів.

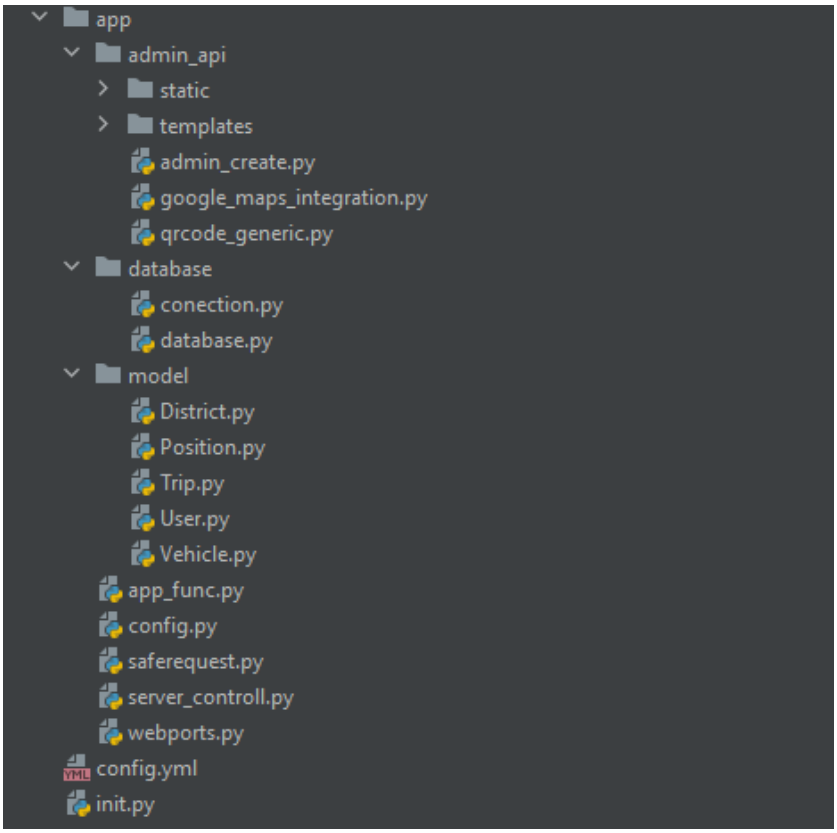


Рисунок 4.10 – Будова проекту

4.8.4 Реалізація функцій для запитів та обробки

Глянувши на рис. 4.11 можна побачити код створення так званих маршрутів (routes), які відповідають за введення та виведення даних.

```
1   from flask import Flask, request
2
3   app = Flask(__name__)
4   @app.route('/', methods=['GET', 'POST'])
5   def receive_message():
6       return "Hello World!"
7
8
9   if __name__ == '__main__':
10      app.run()
```

Рисунок 4.11 – Демонстрація реалізації маршруту у мікрофреймворку flask

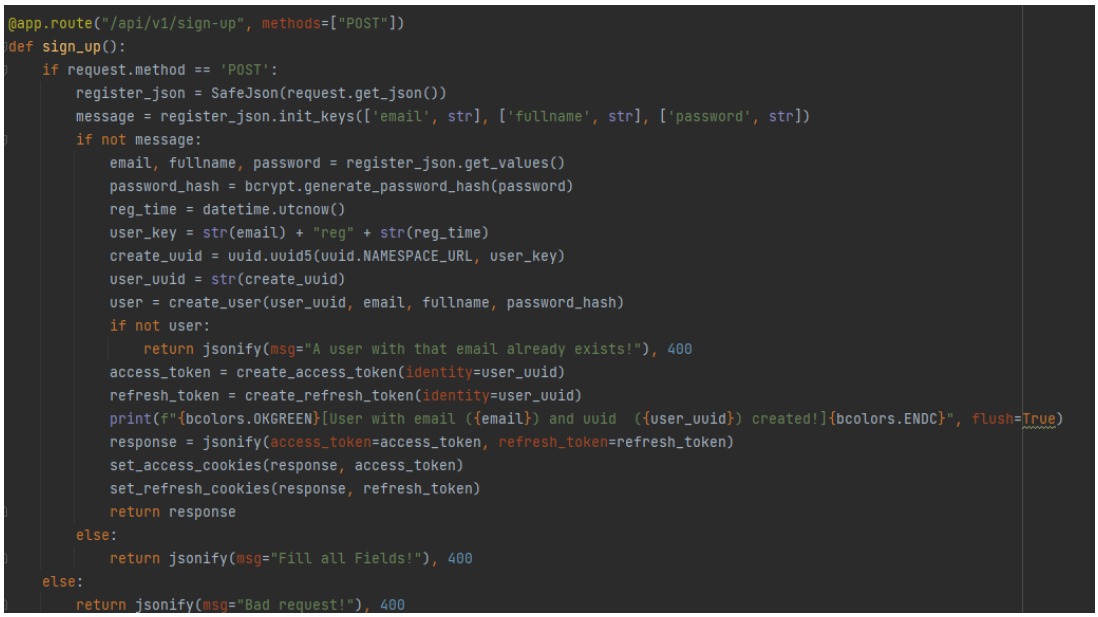
Реалізація маршрутів починається з написання app.route та вказування силки, тобто так званого шляху. Наступним кроком необхідно прописати функцію маршруту. Дивлячись знову ж таки на рис. 4.11, ми бачимо, що в результаті виконання цього коду ми отримаємо вивід «Hello World!», перейшовши по силці localhost:3000.

Детальніше можна роздивитися створення маршруту на прикладі реєстрації, яка знаходиться в API проекту.

Спершу на даний маршрут відсилається інформація у форматі JSON. Далі вона розкодовується та виводяться значення email, fullname, password. Далі перевіряється чи зайнята електронна пошта, якщо так, то повертається помилка, що пошта зайнята, якщо ні, то генерується хеш паролю, унікальний номер користувача. Усі параметри передаються у функцію створення користувача. Далі на основі номеру користувача генеруються токени доступу та оновлення та відправляються клієнту.

Починається все з того, що на вказаний маршрут відправляються дані JSON формату. Потім отримана інформація декодується та відбувається вивід таких значень, як password, email, fullname. Наступним кроком відбувається перевірка

електронної пошти. А саме, йде перевірка того чи вона не є зайнятою. Якщо вона виявляється вже зареєстрованою за іншим користувачем, то, очевидно, повертається відповідна помилка, яка сповіщає користувача про це. Якщо ж ні за яким користувачем даний e-mail не записаний, то відбувається створення хеш-паролю, що являє собою персональний і унікальний номер, за яким в подальшому буде ідентифікуватися користувач. Функція, яка відповідає за створення нових користувачів отримує даний ідентифікаційний номер користувача та всі вказані параметри при реєстрації. І потім вже базуючись на отриманому унікальному номері, відбувається так звана генерація токенів, необхідних для доступу, після чого вони надсилаються юзеру.



```python
@app.route("/api/v1/sign-up", methods=["POST"])
def sign_up():
    if request.method == 'POST':
        register_json = SafeJson(request.get_json())
        message = register_json.init_keys(['email', str], ['fullname', str], ['password', str])
        if not message:
            email, fullname, password = register_json.get_values()
            password_hash = bcrypt.generate_password_hash(password)
            reg_time = datetime.utcnow()
            user_key = str(email) + "reg" + str(reg_time)
            create_uuid = uuid.uuid5(uuid.NAMESPACE_URL, user_key)
            user_uuid = str(create_uuid)
            user = create_user(user_uuid, email, fullname, password_hash)
            if not user:
                return jsonify(msg="A user with that email already exists!"), 400
            access_token = create_access_token(identity=user_uuid)
            refresh_token = create_refresh_token(identity=user_uuid)
            print(f"{bcolors.OKGREEN}[User with email ({email}) and uuid  ({user_uuid}) created!]{bcolors.ENDC}", flush=True)
            response = jsonify(access_token=access_token, refresh_token=refresh_token)
            set_access_cookies(response, access_token)
            set_refresh_cookies(response, refresh_token)
            return response
        else:
            return jsonify(msg="Fill all Fields!"), 400
    else:
        return jsonify(msg="Bad request!"), 400
```

Рисунок 4.12 – Програмна реалізація реєстрації нового юзера

```python
def create_user(user_uuid: str, email: str, fullname: str,  password_hash: str) -> Optional[User]:
    if find_email(email):
        return None
    user = User()
    user.node_type = 'USER'
    user.uuid = user_uuid
    user.email = email
    user.fullname = fullname
    user.profileImage = "https://img.icons8.com/bubbles/2x/user-male.png"
    user.password_hash = password_hash
    user.blocked = False
    user.current_Vehicle = None
    user.current_trip = None
    neo4j.create(user)
    print(f"{bcolors.OKGREEN}[User successfully added!]{bcolors.ENDC}", flush=True)
    return user
```

Рисунок 4.13 – Програмна реалізація запису інформації про юзера в базу даних

4.8.5 Налаштування сервера

В якості операційної системи було вибрано ОС Linux. Цей вибір був зроблений не дарма, адже більшість серверів працює саме на цій операційній системі. У порівнянні з класикою, з Windows, вибір був очевидний, тому, що ця ОС гірше справляється з такими завданнями, як використання декількома юзерами, а також розподілом доступу до ресурсів системи. Ці речі їй вдаються гірше, адже, якщо пригадати, то ідея цієї операційної системи полягала в тому, що вона буде заточена під одного користувача.

Тим часом, як операційна система Linux, з самого початку заточувалася під багатокористувацьке використання і відповідно їй доступний і багатокористувацький функціонал. Такий підхід надав можливість застосункам обмежувати доступ до деяких файлів. Ну і до переваг даної ОС для цього проекту і саме для запуску сервера є звичайно хороший захист, перевірений роками та можливості контролю над технічним пристроєм, які надає Linux. Зокрема, до таких можна віднести керування різноманітними складовими пристрою, чи то камерою, чи то мікрофоном, чи то навіть процесором.

Після встановлення ОС було налаштовано модулі такі як FTP та SSH для віддаленої роботи з сервером, та добавлено нові правила фаєрволу, а саме nftables, для коректної роботи серверу та його захисту.

Операційну систему змінювати не прийшлось, так як на одній з моїх машин вже стояв Linux з дистрибутивом Ubuntu. Залишалося лише налаштувати модулі для віддаленого керування сервером, для нормальної роботи серверу, ще також було необхідно прописати правила для фаєрволу, так звані nftables. В якості модулів для віддаленого управління сервером було вибрано FTP та SSH.

FTP означає File Transfer Protocol, протокол, створений для передачі файлів із віддаленого місця на локальний комп’ютер або навпаки.

З іншого боку, Secure Shell, або SSH, є мережевим протоколом, який полегшує зв'язок між двома комп'ютерами, незалежно від відстані, якщо є електричний шлях для проходження зв'язку.

Як протокол передачі файлів, FTP здатний лише передавати файли з однієї точки в іншу, а також деякі основні операції з файлами, такі як копіювання, переміщення або видалення файлів і каталогів. SSH виходить далеко за рамки цього, оскільки дозволяє користувачеві видавати команди, які можуть бути інтерпретовані і виконані на віддаленому комп'ютері сервером, що слухає. Він також може використовуватися для тунелювання, моніторингу певних служб і додатків, що працюють, і навіть для передачі файлів.

4.8.6 Налаштування та запуск API

Для початку, було встановлено python на сервер та усі використані бібліотеки, створено директорію серверу та завантажено його файл і файли.
Відредаговано файл config.yaml наступним чином:

Першим кроком необхідно було інсталювати python на сервер та підтягнути всі необхідні бібліотеки, які були згадані раніше. Після чого було необхідно створити серверну папку та розмістити туди всі необхідні файли. Також для налаштування потрібно було відредагувати вже згаданий також раніше файл **config.yaml**, відповідно до того, як це продемонстровано на рисунку 4.14.

Рисунок 4.14 – Лістинг файлу config.yaml, який відповідає за конфігурацію проекту

Відповідно до рисунку 4.14, бачимо, що ми вимкнули debug та reloader та cookie, присвоївши відповідним змінним значення false. Як можна замітити також було поміняно порт застосунку. Ну і в якості адреси для db_host, було прописано localhost, так як відповідно до нашої ситуації, коли API та база даних розташовані на однаковому сервері це є коректним рішенням.

Для додавання створеного бекенду в набір сервісів дистрибутива Ubuntu було використано команду **sudo nano /etc/systemd/system/bitroller_server.service**

В результаті було створено сервісний ініціалізатор з наведеним нижче кодом.

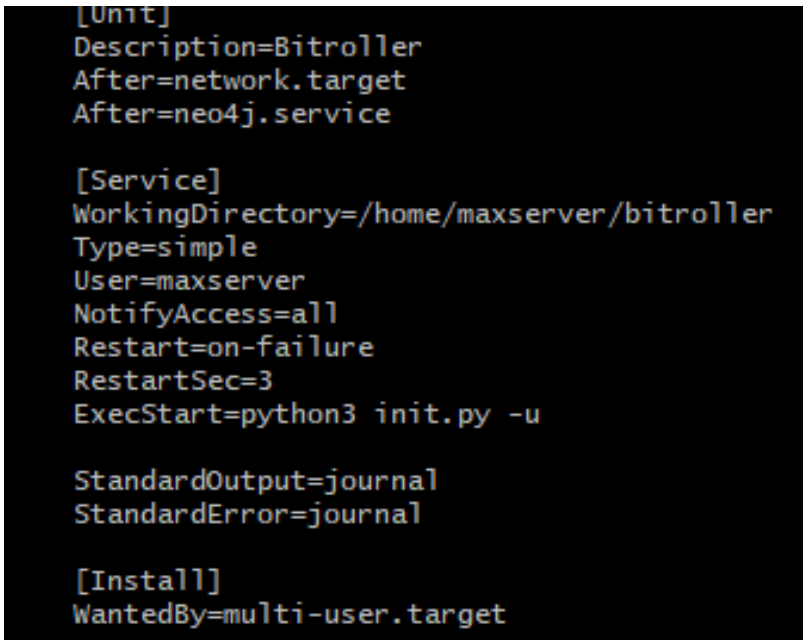


Рисунок 4.15 – Налаштування сервісного файлу в дистрибутиві Ubuntu

Було встановлено, щоб сервіс вступав в дію, тобто запускався після ініціалізації бази даних NEO4J та network. Також було виставлено ще деякі параметри, такі як шлях до каталогу, стартовий режим, та команда, яка буде відповідати за запуск. Також сервіс був доданий до списку автозапуску.

4.8.7 Встановлення та налаштування Apache

В якості веб-серверу було використано apache. Першим кроком, я прописав команду **sudo apt-get install apache2** для встановлення пакету з сервером. Для сайту був необхідний окремий каталог, відведений під нього, тому було використано команду **sudo mkdir /var/www/bitroller**, відповідно було також додано права для доступу до даної директорії.

Далі, був створений конфігураційний host файл з наступним змістом (рис. 4.16.). В даному файлі маємо змогу для створення віртуальних хостів.

```
<VirtualHost *:80>
    ServerAdmin maxnomad123@gmail.com
    ServerName bitroller.club
    ServerAlias www.bitroller.club
    #Redirect permanent / bitroller.club/

    DocumentRoot /var/www/bitroller
    <Directory "/var/www/bitroller/">

Options FollowSymLinks
    AllowOverride All
    Require all granted
</Directory>
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Рисунок 4.16 – Створення віртуальних хостів

Як ми бачимо, в даному файлі також був вказаний шлях до каталогу сайту та надано йому доменне ім’я.

При запиті спрацьовує перенаправлення на папку з сайтом. Активація даної конфігурації відбувається командою sudo a2ensite bitroller.conf

В результаті запиту нас буде перенаправлено на вказаний каталог зі сайтом. Щоб активувати вищезгадану конфігурацію, потрібно запустити команду **sudo a2ensite bitroller.conf.**

І вкінці потрібно було лише закинути вже готовий сайт в каталог, який ми спеціально створювали для цього і вказували шлях до нього у файлі конфігурації та після цього можна спокійно перезавантажувати apache.

/var/www/bitroller/

Имя	Размер	Изменено	Права	Владел...
..				
static		24.04.2021 21:41	rwxrwxrwx	1000
robots.txt	1 KB	30.05.2021 22:18	rwxrwxrwx	1000
manifest.json	1 KB	30.05.2021 22:18	rwxrwxrwx	1000
logo512.png	10 KB	30.05.2021 22:18	rwxrwxrwx	1000
logo192.png	6 KB	30.05.2021 22:18	rwxrwxrwx	1000
index.html	3 KB	30.05.2021 22:18	rwxrwxrwx	1000
bitroller.svg	1 KB	30.05.2021 22:18	rwxrwxrwx	1000
asset-manifest.json	3 KB	30.05.2021 22:18	rwxrwxrwx	1000

Рисунок 4.17 – Вивантаження необхідних файлів на сервер, використовуючи FTP клієнт

4.9 Розробка frontend частини

4.9.1 Створення архітектури

Після вибору списку бібліотек, я взявся за створення структури проекту, де була визначена загальна архітектура. А також, взаємозв'язки між модулями, зв'язки між компонентами та їхніми стилями тощо (рис. 4.18). На рисунку можна замітити чіткий поділ між графічною складовою проекту, які розміщені в каталогах «components» і «modules» і між BLL, розшифрувавши абревіатуру це означає BusinessLogicLayer. Файли BLL розташовані в каталозі «redux».Решту всіляких допоміжних речей розкидані по різним каталогам, наприклад, зображення знаходяться в директорії assets, тим часом взаємні для всього проекту функції в директорії shared тощо.

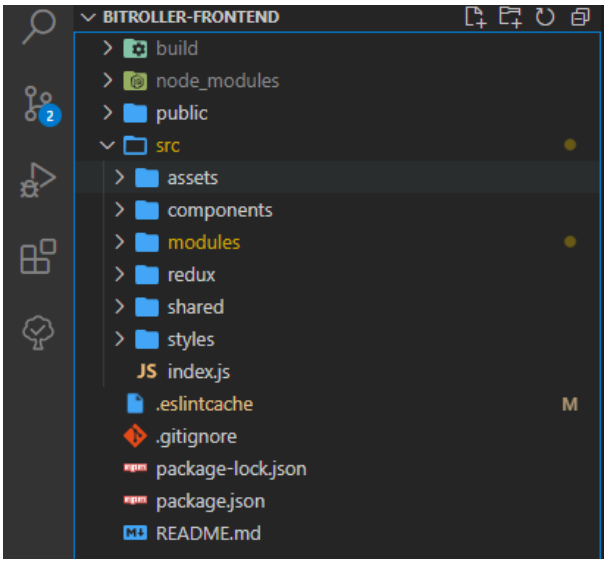


Рисунок 4.18 – Загальна структура проекту

4.9.2 Створення чистих функцій-компонент

Чиста функція - це функція (блок коду), яка завжди повертає один і той же результат при передачі одних і тих же аргументів. Вона не залежить від зміни стану або даних під час виконання програми. Скоріше, вона залежить тільки від вхідних аргументів. Крім того, чиста функція не спричиняє жодних видимих побічних ефектів, таких як мережеві запити, мутація даних тощо.

Метод створення чистих функцій-компонент, який використаний при написанні додатку, полягає в огортанні в memo при експорті компоненти (рис 4.19).

```
    </div>
  )
}

export default memo(HistoryPage);
```

Рисунок 4.19 – Експорт компоненти огорнутої в React.memo

Якщо детальніше розібратися в будові самої компоненти, то можна замітити, що дві частини складають тіло компоненти. В першій частині описуються функції, оголошуються змінні, виконуються певні обчислення, тому ця частина вважається логічною (рис. 4.20). Це місце де проходять всі перетворення та маніпуляції над змінними.

```
const HistoryPage = () => {

  const lottie_container = useRef(null);

  useEffect(() => {
    lottie.loadAnimation({
      container: lottie_container.current,
      renderer: 'svg',
      loop: true,
      autoplay: true,
      animationData: tripsImage.default
    })
  }, [])

  const dispatch = useDispatch();

  const trips = useSelector((state) => state.history.trips);
  const totalPrice = useSelector((state) => state.history.totalPrice);
  const totalDistance = useSelector((state) => state.history.totalDistance);
  const vehiclePecentage = useSelector((state) => state.history.vehiclePecentage);
  const favouriteDistrict = useSelector((state) => state.history.favouriteDistrict);

  /**
   * @desc [Hook effect] Receive history on mount.
   */
  useEffect(() => {
    dispatch(
      receiveHistory(),
    );
  }, []);
```

Рисунок 4.20 – Логічне тіло компоненти

Другою по порядку, але не за значенням є return (рис. 4.20). Return компоненти реакту не має ніякої істотної різниці від привичного return-у звичайної функції. (це зумовлено тим що компонента це теж свого роду функція, яка повертає .jsx). Для того, щоб створити компоненту в return нам треба написати. Простіше говорячи, під компонентою, можемо вважати собі блок html-розмітки, загорнутий в JavaScript. Це чимось нагадує сладний механізм годинника, тільки маючи всі деталі (компоненти), можна приступити до збирання суцільного робочого механізму, в моєму випадку це додатку.

```
return (
  <div className='history-page'>
    <Header />
    { trips.length !== 0
      ? <div className='history-page-content'>
        <HistoryPageTrips trips={trips} />
        <HistoryPageInfo
          totalPrice={totalPrice}
          totalDistance={totalDistance}
          vehiclePecentage={vehiclePecentage}
          favouriteDistrict={favouriteDistrict}
        />
      </div>
      : <div className='history-lottie'>
        <div className='lottie-trips' ref={lottie_container}></div>
        <p>You have no any trips yet. Try right now!</p>
      </div>
    }
    <Footer />
  </div>
)
```

Рисунок 4.21 – Return-компоненти з описаним html (кастомним)

4.9.3 Створення UI/UX

Перейдемо до графічної складової. Дизайн додатку був розроблений у сервісі Figma. За допомогою цього сервісу було прийнято певну концепцію того, як має виглядати додаток, та приблизна палітра кольорів, яка буде використовуватися. (рис 4.22).

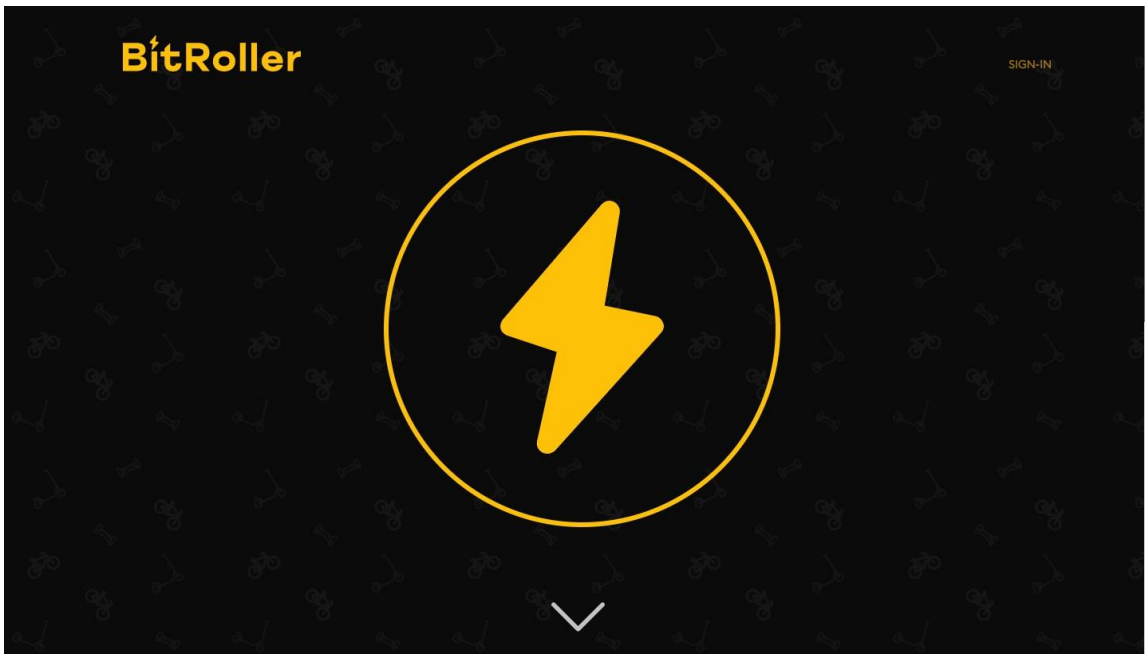


Рисунок 4.22 – Приклад графічного дизайну додатку

Стосовно основних кольорів, то мій вибір впав на двох: жовтому то чорному.

Як можна побачити з (рис 4.23) загальний дизайн сторінок виконаний в мінімалістичному та лаконічному стилі.

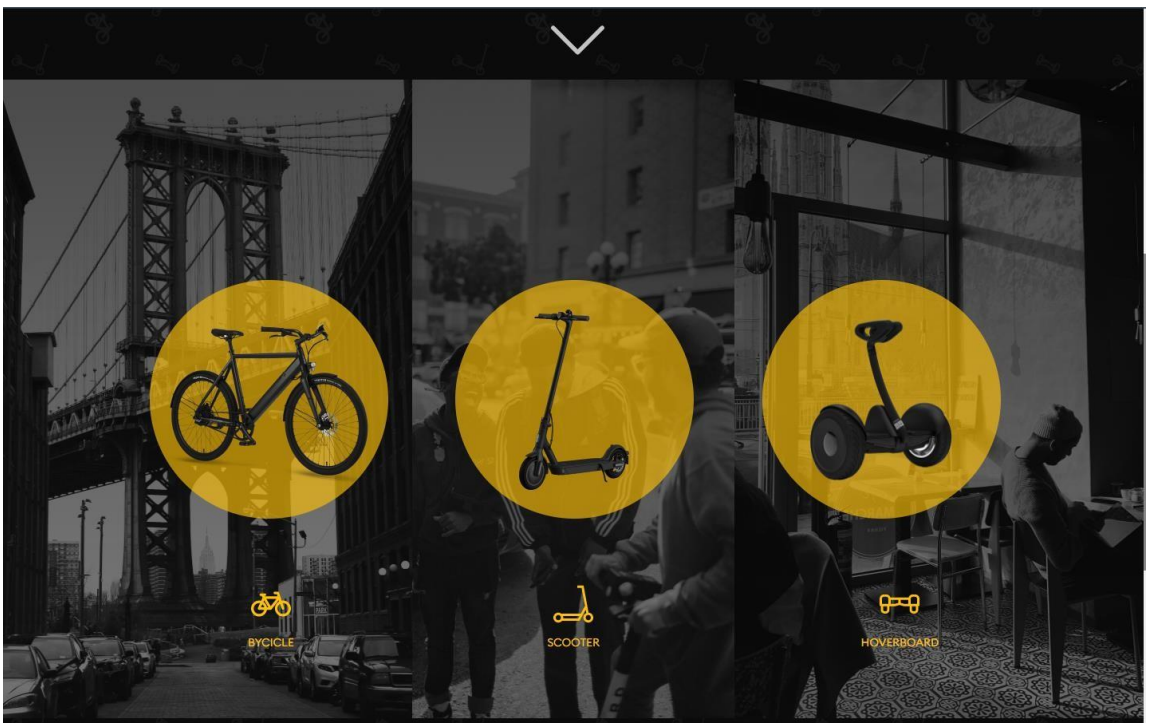


Рисунок 4.23 – Одна з частин головного екрану, де використано мінімалізм в поєднанні з модернізмом

Додаток був розроблений адаптивним, тому може відкриватися і адаптуватися до будь-якого розширення та формфактора екрану. (рис 4.24, 4.25). Нижче наведені приклади роботи додатку на різних пристроях, в першому випадку на ноутбуці, в другому на смартфоні.

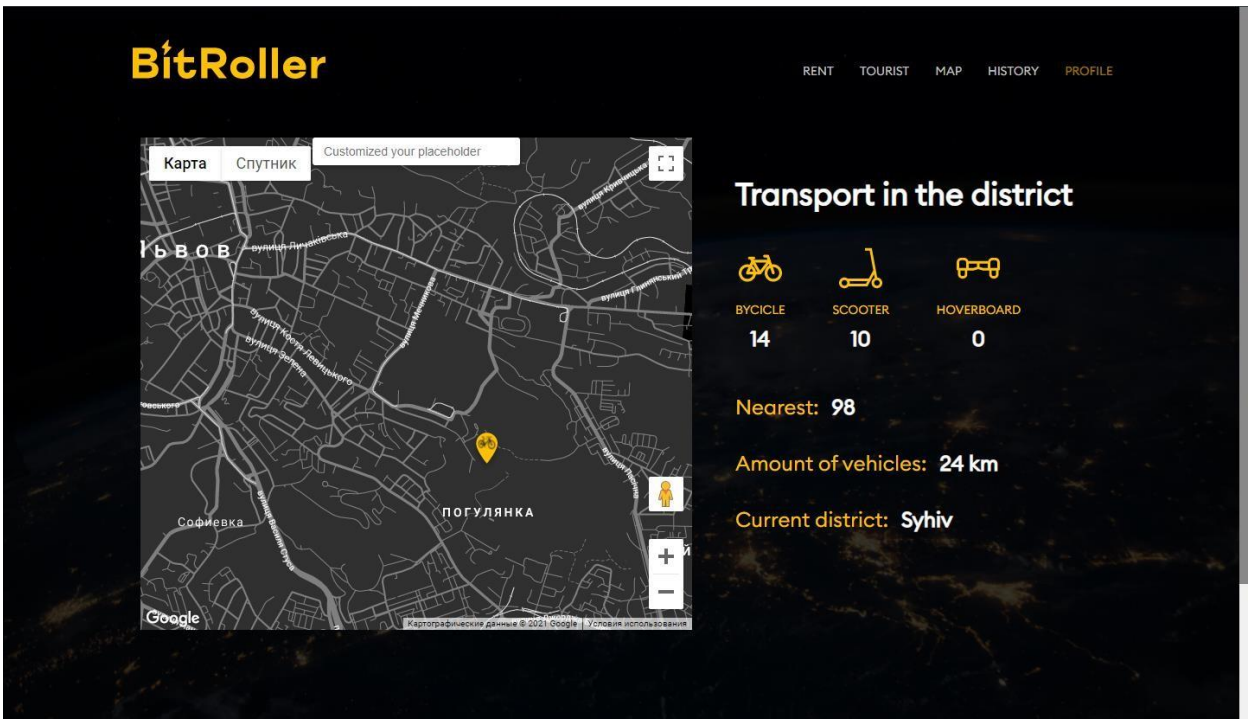


Рисунок 4.24 – Вигляд сторінки в режимі desktop

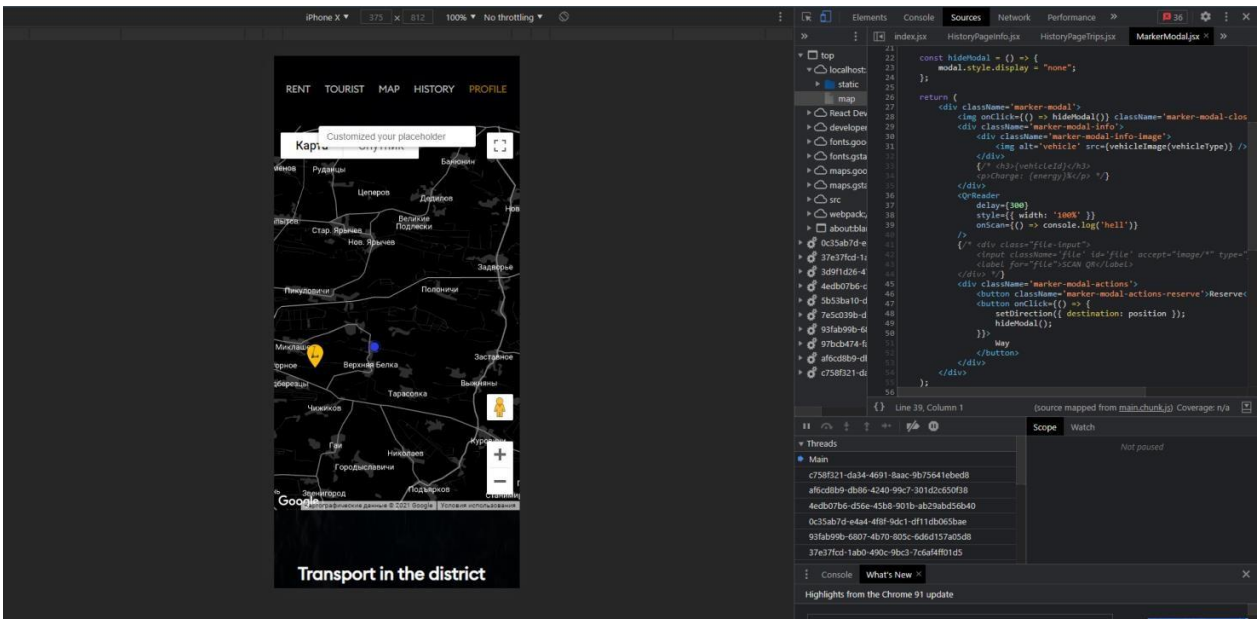


Рисунок 4.25 – Вигляд сторінки на мобільному пристрої

4.10 Тестування коректності роботи backend частини

Зробимо GET запити на силку api.bitroller.club, bitroller.club та neo4j.bitroller.club за захищеним протоколом https. В результаті отримано статус-код 200, що означає, що всі запити було виконано вдало.

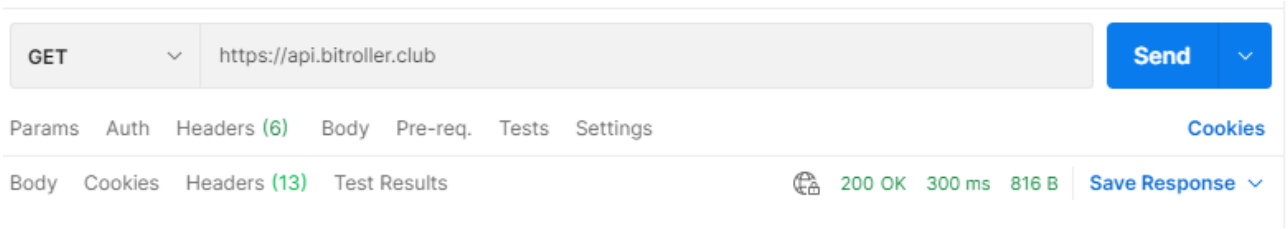


Рисунок 4.26 – Запит до api.bitroller.club

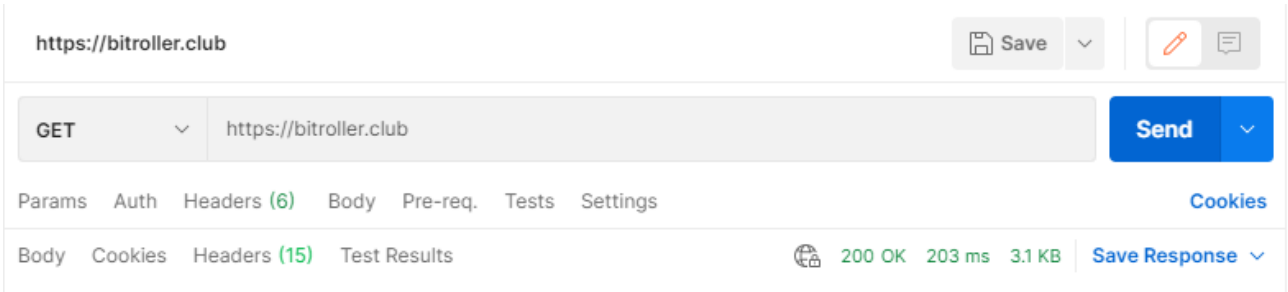


Рисунок 4.27 – Запит до bitroller.club

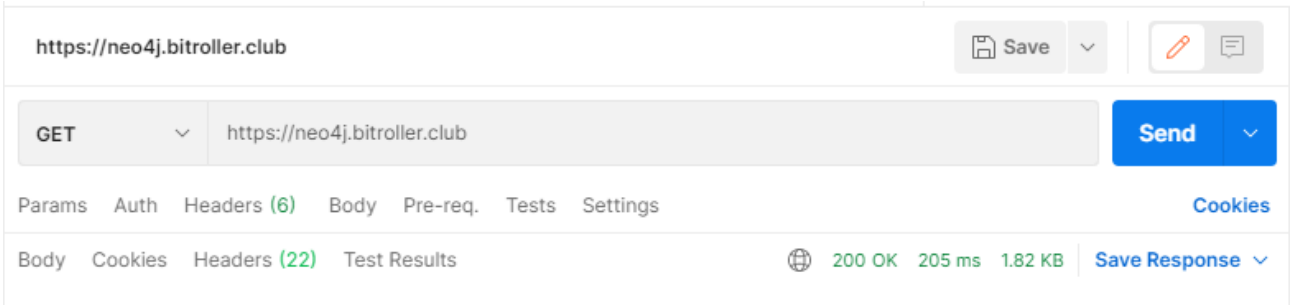


Рисунок 4.28 – Запит до neo4j.bitroller.club

Це може означати лише те, що всі засоби, такі як Cloudflare, проксі сервер, веб сервер, база даних та backend працюють коректно.

Робота з API

Тестування панелі для адміністрування. Перейти за посиланням https://api.bitroller.club/api/v1/admin. Для перевірки функціоналу розміщення нового транспорту на карті, можемо поставити мітку в вибраному місці, наступним кроком обираємо тип транспорту, який ми хочемо розмістити та натискаємо на кнопку «Створити».

Як результат, створено необхідний нам тип транспорту в обраному місці (рис. 4.26)

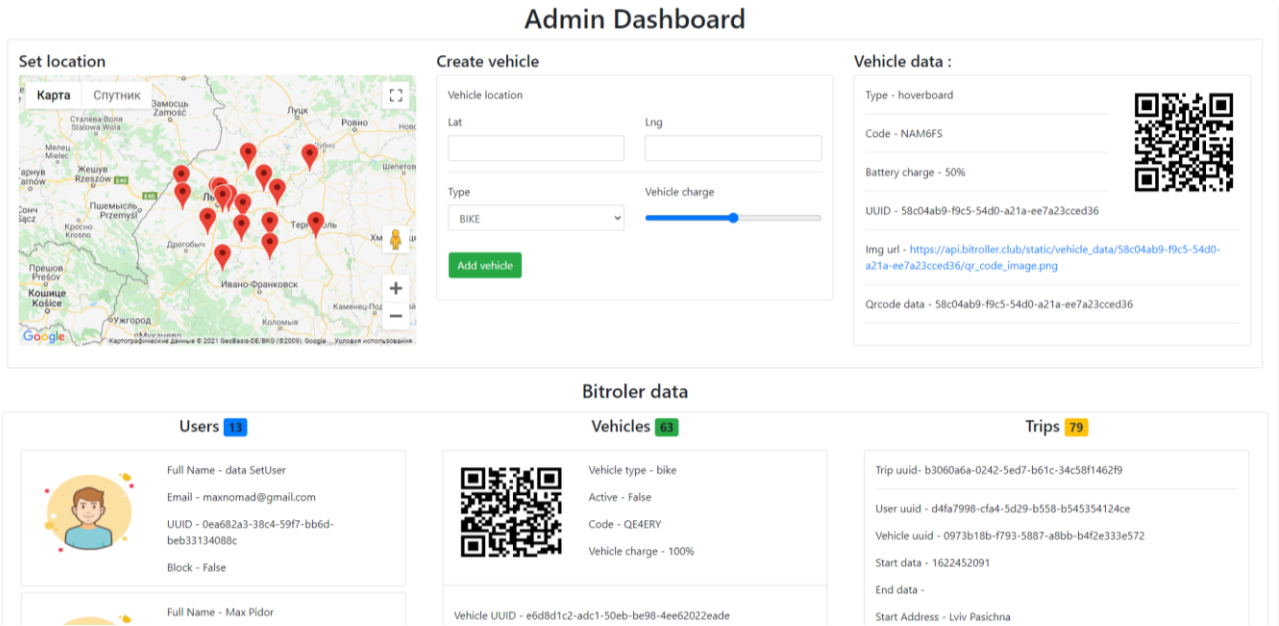


Рисунок 4.29 – Створення транспортного засобу

Для того, щоб створити юзера, виконаємо GET запит на маршрут api.bitroller.club/api/v1/sign-up з таким наповненням body:

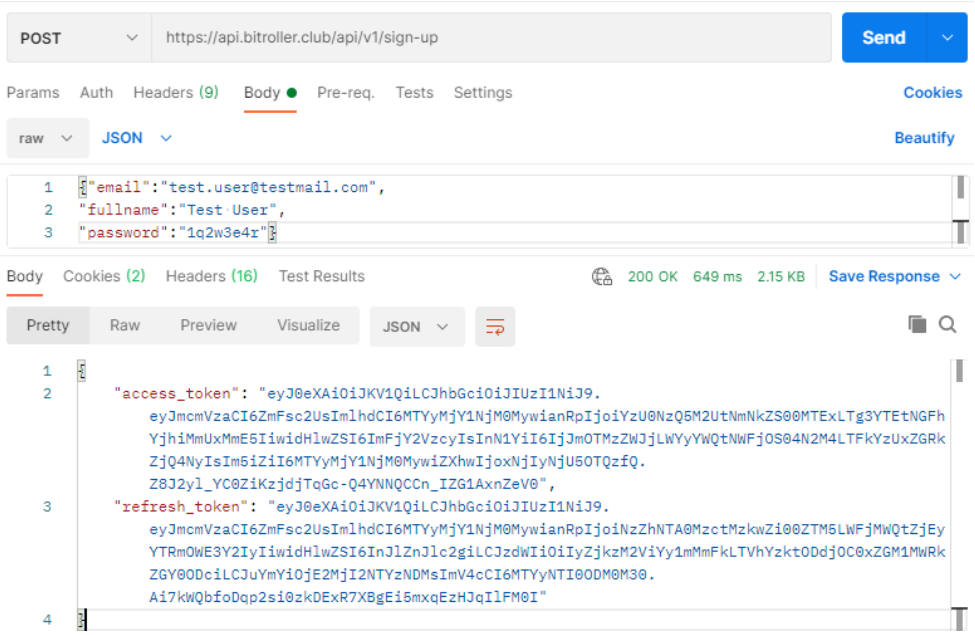


Рисунок 4.30 – Реєстрація користувача

Користувача було успішно створено, так як в результаті тестування отримано статус-код 200, а також повернуто access refresh токени.

Наступним кроком протестуємо коректність роботи авторизації юзера. Виконаємо GET запит на маршрут api.bitroller.club/api/v1/sign-in та слідуючим наповненням body:

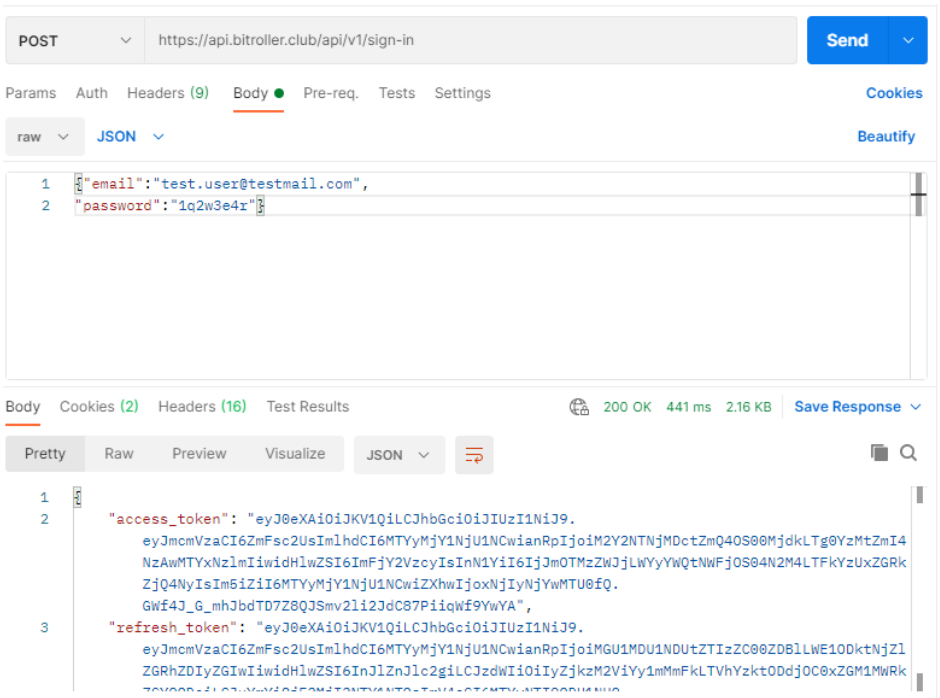


Рисунок 4.31 – Авторизація користувача

В ході процесу тестування, получено access refresh токени та статус-код 200, що свідчить про те, що користувач успішно авторизувався.

Перейшовши у профіль і виконавши запит на маршрут api.bitroller.club/api/v1/profile повернуться дані про користувача, який авторизувався (рис. 4.32).

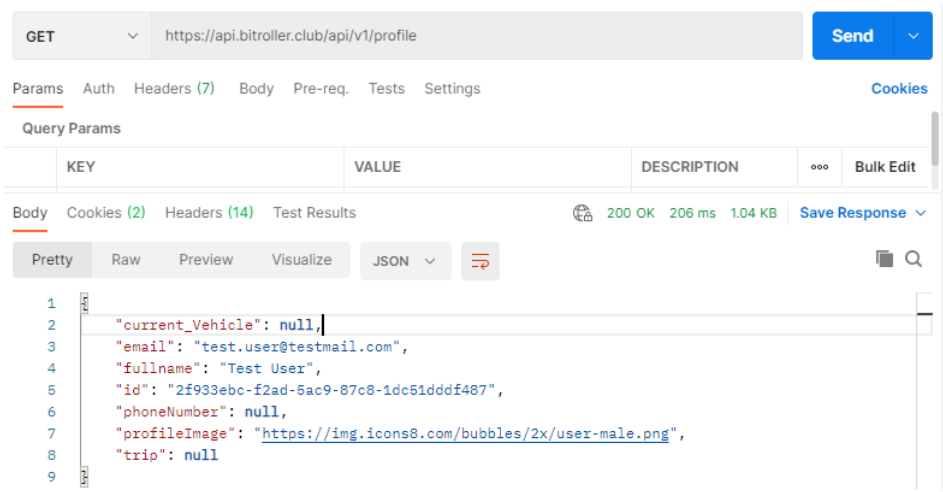


Рисунок 4.32 – Дані користувача

Для отримання всіх пристроїв на маршруті api.bitroller.club/api/v1/vehicles методом get.

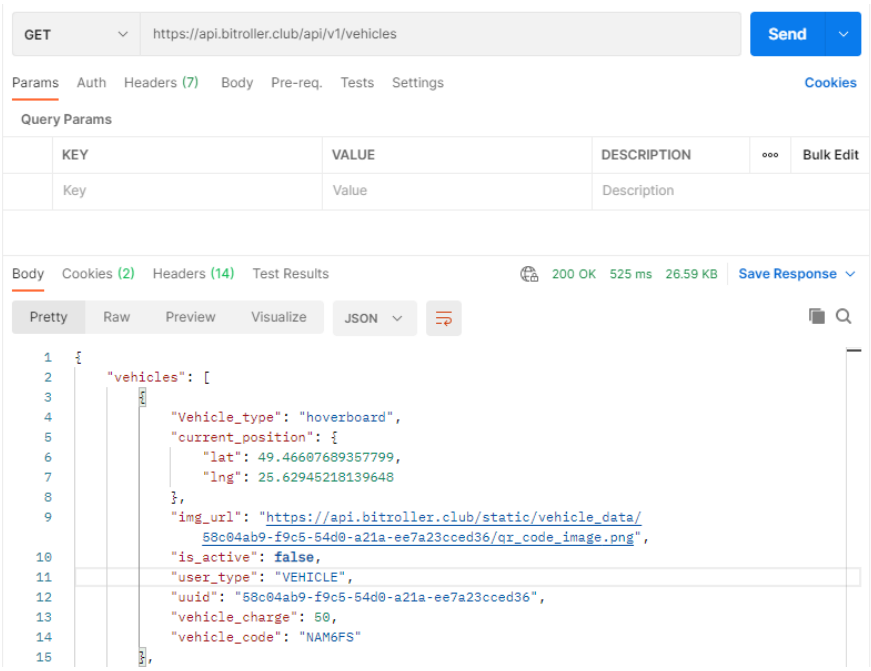


Рисунок 4.33 – Отримання списку транспорту

Повернено статус 200.

Проводимо перевірку статусу транспорту, is_active = false, це означає, що обраний пристрій вільний. Копіюємо vehicle_code – NAM6FS

Початок поїздки виконано на маршруті api.bitroller.club/api/v1/vehicle-connect, тобто цей процес також супроводжується прив'язуванням користувача до пристрою. Він в змозі набути двох параметрів: id – це унікальний номер пристрою, та code – його код.

Створюємо прив'язку пристрою за допомогою запиту: api.bitroller.club/api/v1/vehicle-connect?code=NAM6FS

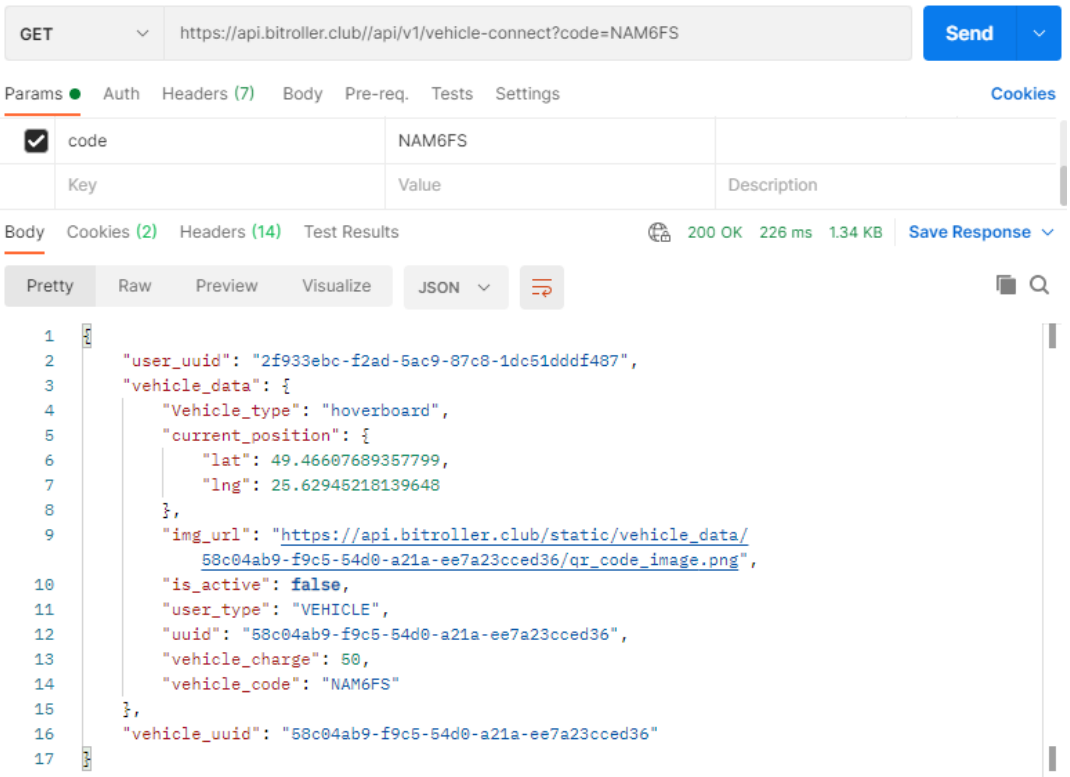


Рисунок 4.34 – Прив'язка користувача до транспорту та створення поїздки

На цьому етапі поїздка вже створено і включено розрахунок вартості поїздки.

Для того, щоб завершити поїздку створимо POST запит на маршрут **api.bitroller.club/api/v1/end-trip** та body у форматі json з локацією. Як результат, поїздка завершиться і місце розташування електричного транспортного засобу набуде нового значення в базі даних.

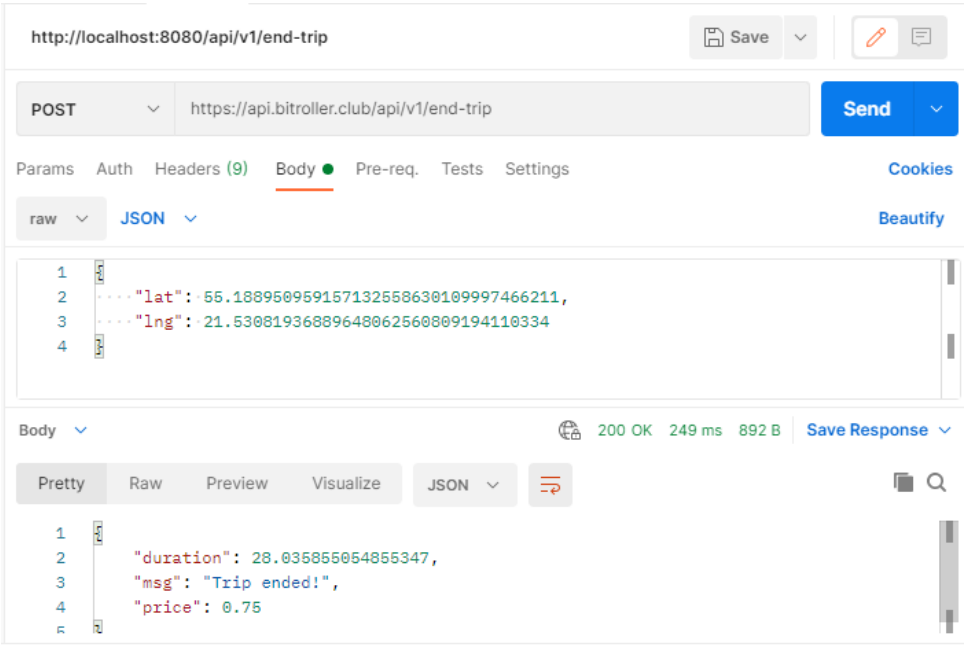


Рисунок 4.35 – Кінець поїздки

Для того, щоб переглянути всі завершені поїздки, створимо GET запит на маршрут **api.bitroller.club/api/v1/trips**, та в результаті дістанемо інформацію про всі поїздки, більш того, також отримаємо інформацію про транспорт, який використовувався.

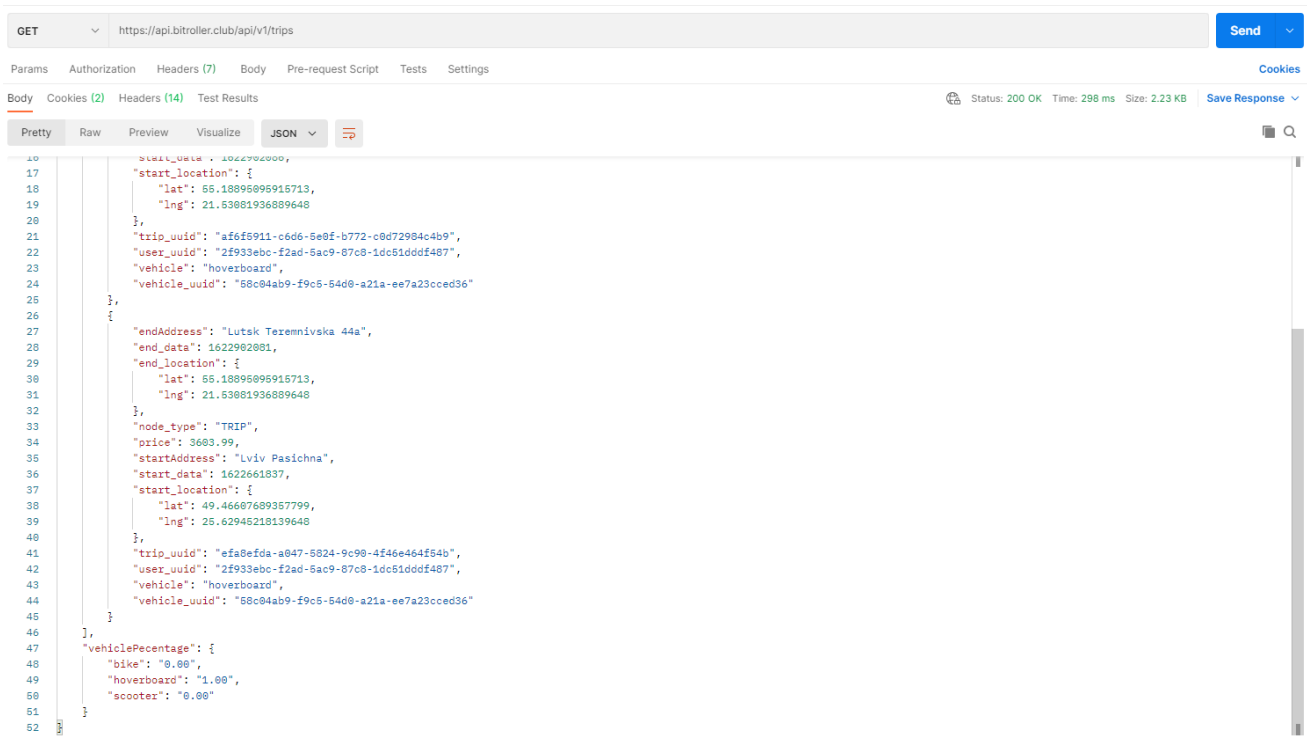


Рисунок 4.36 – Список поїздок

Знаходження пристроїв зробимо на маршруті api.bitroller.club/api/v1/vehicle та опцією code=NAM6FS. В результаті получимо дані про тип транспорту з шуканим індивідуальним номером.

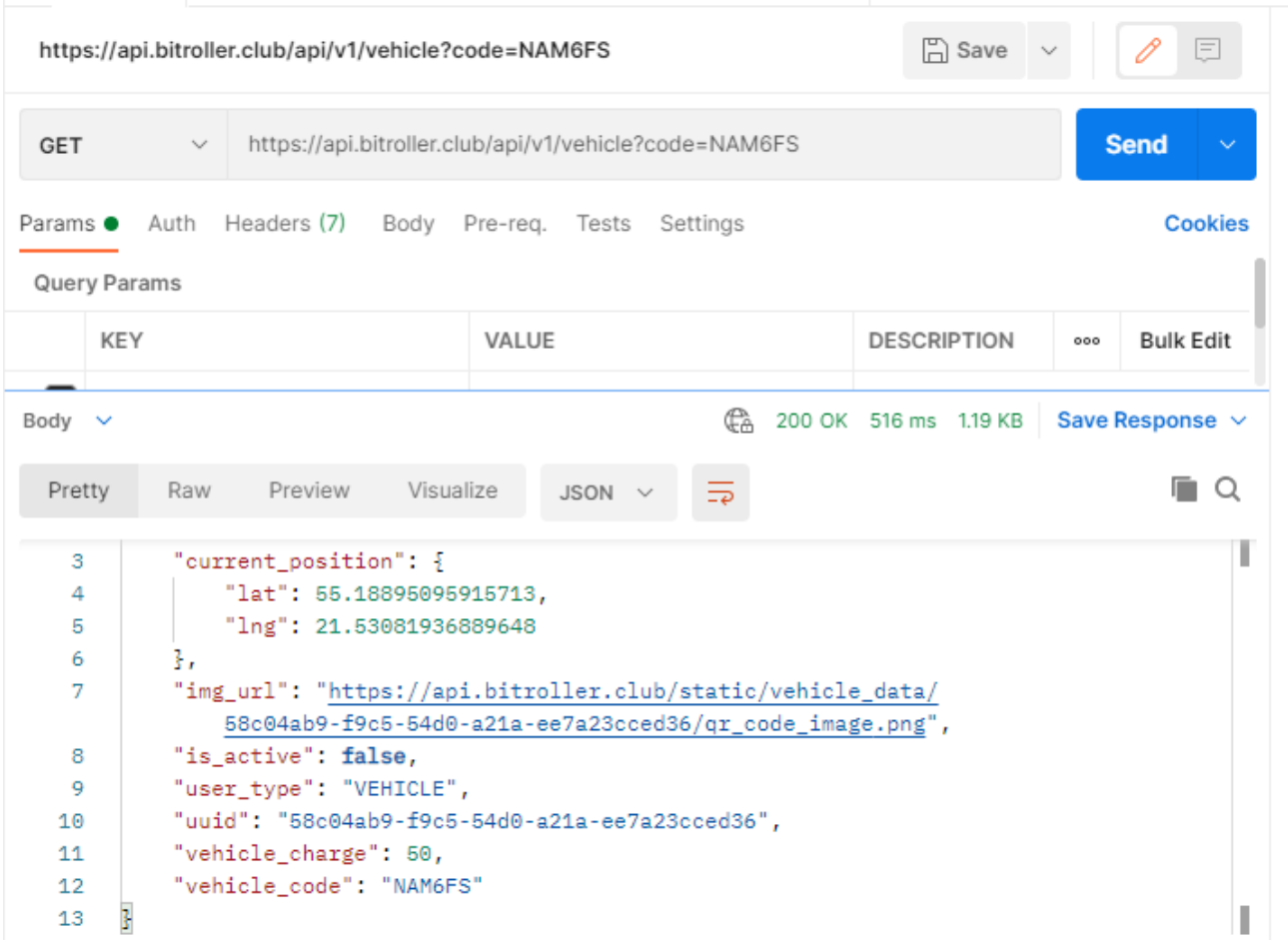


Рисунок 4.37 – Пошук транспорту

Використовуючи алгоритм K-means проаналізовано використання сервісу. Внаслідок аналізу було продемонстровано на карті зони з найбільшим попитом користувачів.

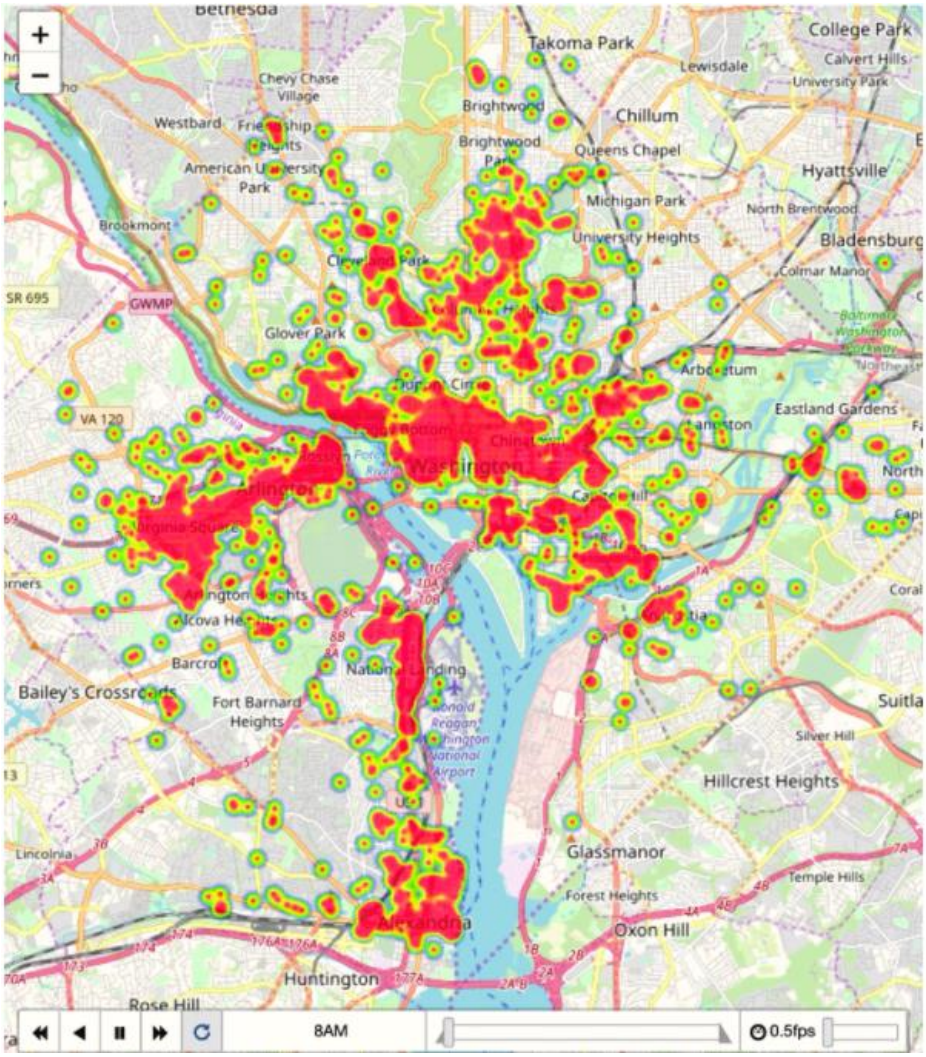


Рисунок 4.38 – Активні зони

Висновки до розділу

В цьому розділі досліджено та проаналізовано програмне забезпечення, необхідне для написання даної роботи. Загалом було зібрано власний сервер та розгорнуто на ньому веб-сервер та API. Для реалізації даного продукту, використано REST API, та мікрофреймворки для його створення, різноманітні види шифрування та захисту даних. Web частина додатку реалізована мовою програмування JavaScirpt з використанням графічної фреймворку React у середовищі Visual Studio Code.

РОЗДІЛ 5. РОЗРОБЛЕННЯ СТАРТАП-ПРОЕКТУ

5.1 Опис ідеї проекту

Проблеми та причина створення. Наразі в даний час достатньо актуальним є питання перенасиченості (перевантаження) на дорогах великих міст, здійснюється пошук рішень для виходу з даної ситуації. Без завантажених маршрутних таксі та інших видів громадського транспорту, без стабільних корків на дорогах, без втрат часу (який має вартісний еквівалент) на переміщення в просторі

Мій проект "BitRoller" дасть змогу втілити це у реальність, оскільки створений зручний, простий у використанні та сучасний сервіс з оренди самокатів, який буде кращим за конкурентів в цьому напрямі!

Рішення є простим – електротранспорт, такий як самокати, велосипеди, ховерборди є маневреними, швидкими і що найголовніше - ними можна пересуватись на спеціально відведених доріжках, які зазвичай вільні, тому ніхто не буде заважати.

Звісно подібні сервіси вже існують, проте вони не надають всіх тих функцій, що реалізовано у даному проекті. По-перше, у нас є вибір у пристроях. По-друге, вмієте користуватись ховербордом, беріть його, кому зручніше на велосипеді – теж є. Інші схожі рішення від інших компаній надають лише самокати, але не усім це підходить. Також іноді буває складно знайти, де саме припаркований пристрій, але не у BitRoller, тому що у нас є функція прокладання шляху до нього. Можна користуватись туристичним режимом, який буде складати маршрути саме для вас також в нас присутні і багато інших корисних функцій. Найголовнішим, я вважаю, є підтримка роботи в браузерів, такого функціоналу не надають більшість компаній.

5.2 Розроблення ринкової та маркетингової стратегії

Для кого це? Сервісом можуть користуватись особи, які старші 14 років. Ціни доступні, на рівні компаній конкурентів, хоча можливостей та функціоналу ми

надаємо більше. Кудись поспішаєш, а таксі дуже дорого, або великі затори? Рішення одне – BitRoller. Вам хочеться прогулятись по місту, і подивитись на визначні місця? Будь ласка, вихід є сервіс навіть запропонує вам допомогу в складанні оптимального маршруту.

Оплата проводиться за допомогою захищених сервісів електронного банкінгу. У нашій ситуації є кілька видів оплати: google-pay, apple-pay, тощо, або за допомогою прив'язування картки до профілю, що однією перевагою так як в інших додатках по оренді електротранспорту є тільки один варіант оплати, і це через apple-pay/google-pay, а наскільки мені відомо, не всі люди мають підключеними ці функції.

Щодо ризиків та доходів то хостинг серверу на велику кількість користувачів буде обходитись орієнтовно в 10 000 гривень за місяць. На вдосконалення платформи може йти від 30000 гривень за місяць. Витрати на маркетинг до 25 000 гривень за місяць, в це входить покупка реклами в різних блогерів, розміщення вуличної реклами та реклами в мережі. Для розміщення транспорту по місту потрібно декілька працівників, тож зарплата працівникам до 40 000 гривень за місяць (рис. 5.1).

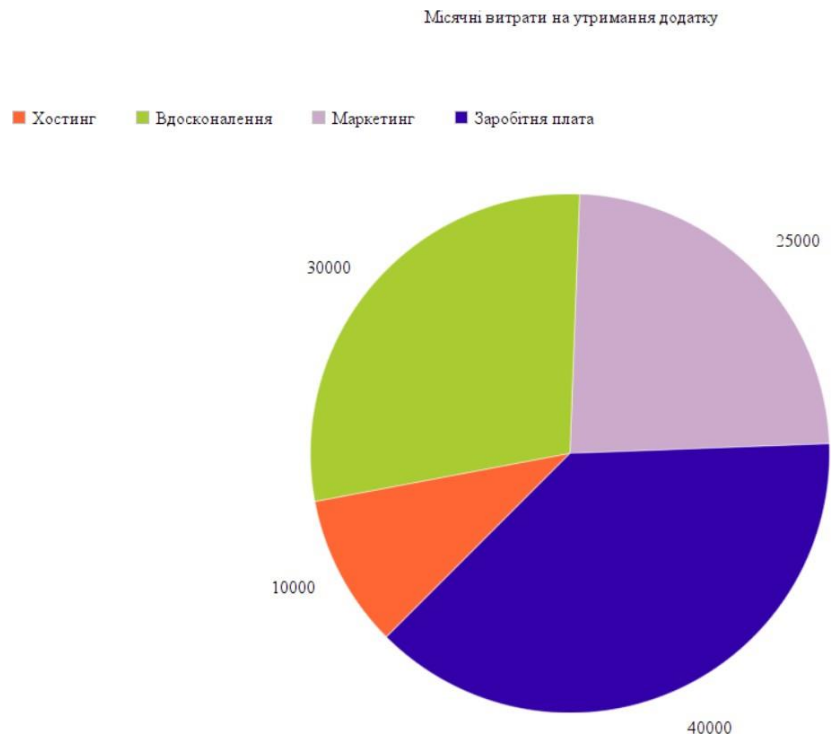


Рисунок 5.1 – Кругова діаграма місячних витрат

Дохід від утримання сервісу буде рости відповідно до кількості користувачів сервісом. Тому можна припустити, що дохід кожного місяця буде рости в арифметичній прогресії. Якщо припустити, що кожен користувач буде витрачати по 200 гривень на місяць (особисто я витрачаю набагато більше, оскільки активно використовую такий транспорт). То маючи десь 2 тисячі користувачів, прибуток буде складати умовних 400 тисяч гривень. В такому разі чистий дохід буде складати близько 295 тисяч гривень.

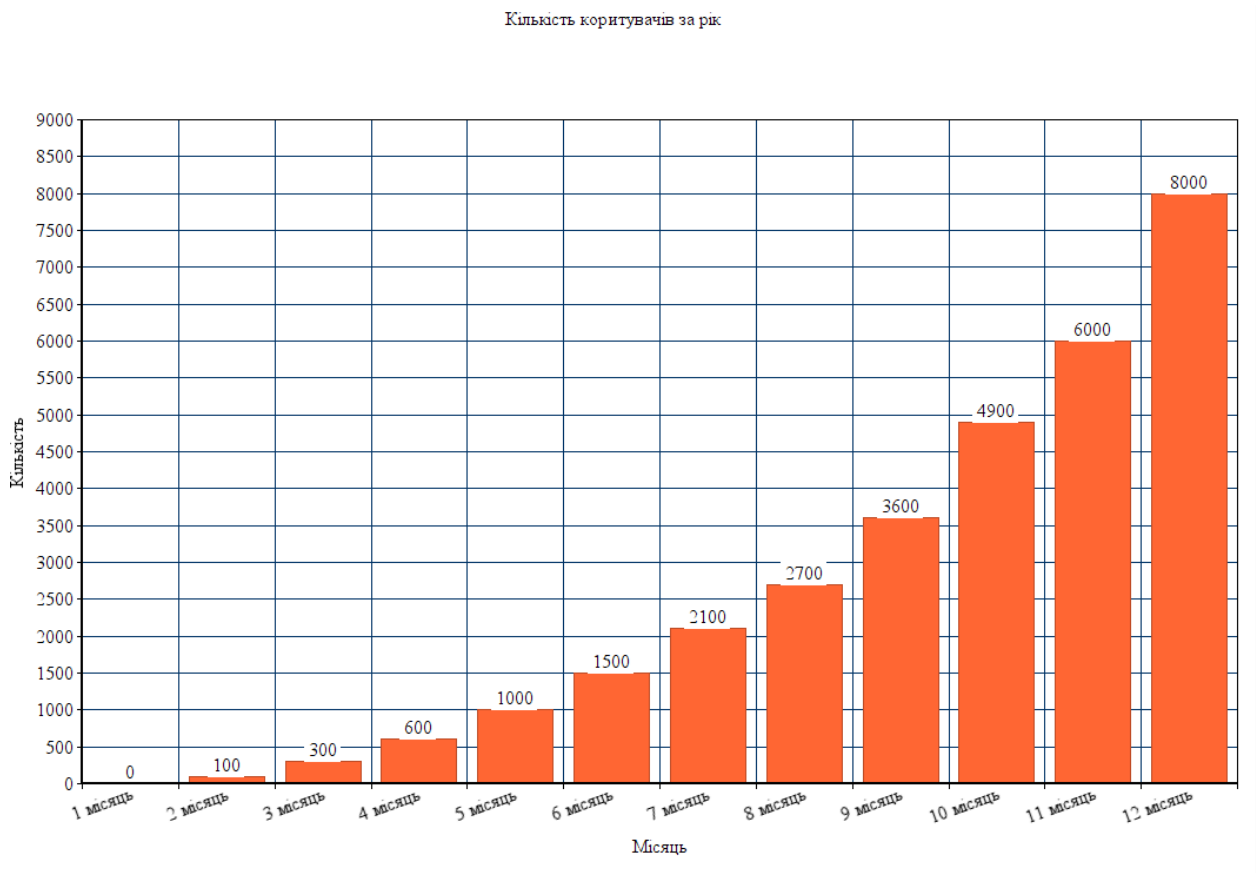


Рисунок 5.2 – Графік співвідношення місяця та кількості користувачів

Враховуючи, що буде задіяна реклама, та так зване «сарафанне радіо - ОБС», а також населення Львова, яке за офіційними даним перевищує 1 мільйон осіб, такий графік є досить реальним. А якщо слідувати такій гістограмі, то чистий дохід за умови, що кожен користувач буде витрачати 200 гривень за місяць, мав би становити 4.900.000 грн. І це тільки за перший рік.

5.3 Результати роботи сервісу для прокату мобільного електротранспорту

Вперше зайшовши на сайт можна зустріти форму реєстрації нового користувача, саме тут починається екскурс додатком, для нового користувача. (рис. 5.3).

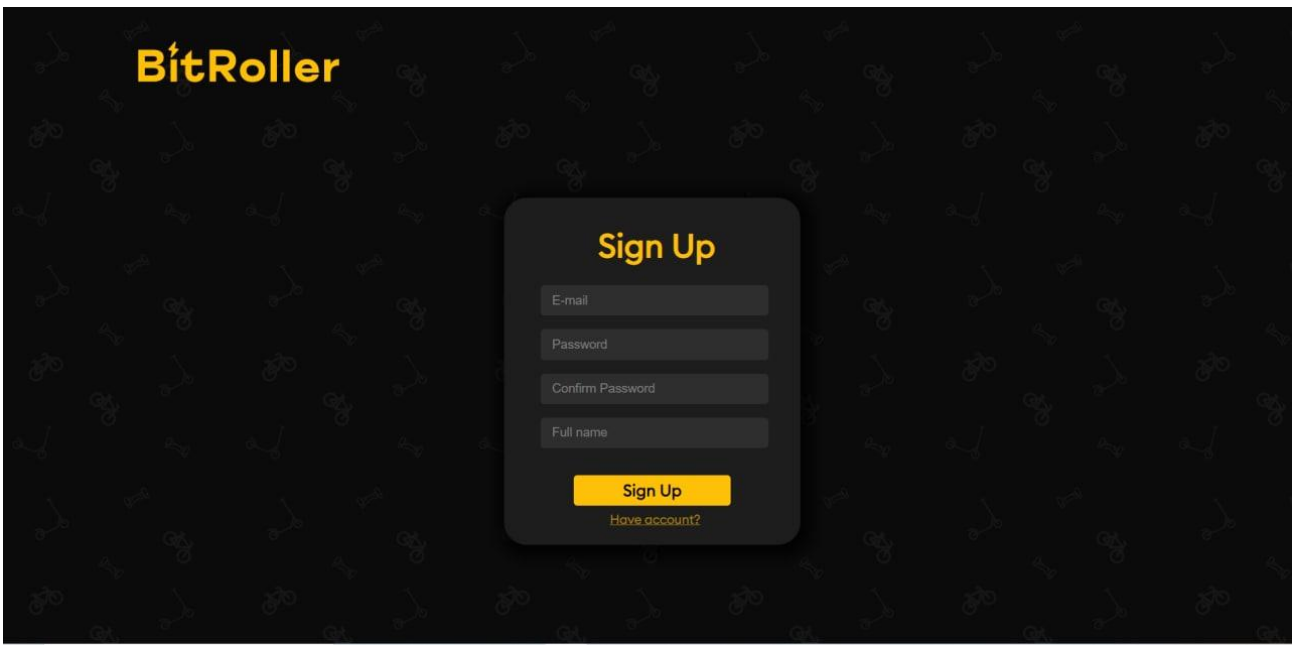


Рисунок 5.3 – Реєстрація нового користувача

Після успішної реєстрації, користувач потрапляє на сторінку налаштування профілю, тут він матиме змогу додати додаткові дані про себе, такі як номер телефону та фотокартку, прив’язати банківську картку до аккаунту, а також змінити всі важливі дані про себе в разі потреби. Наприклад часто трапляється таке, що людина змінила номер телефону, чи поштову скриньку, в даному випадку це не є проблемою. (рис. 5.4). Налаштувавши профіль користувача, можна впевнено рухатися далі і продовжувати тестувати функціонал додатку. Наступним кроком пропоную перейти до сторінки з картою. (рис. 5.5)

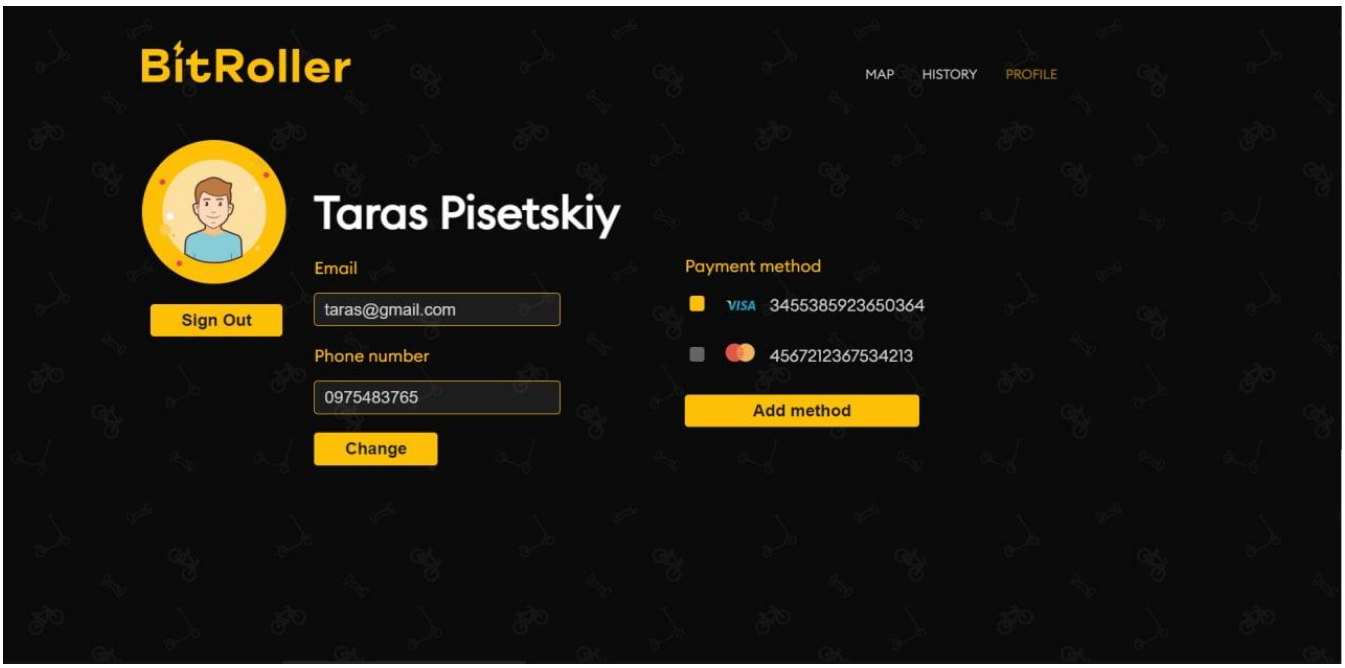


Рисунок 5.4 – Профіль користувача

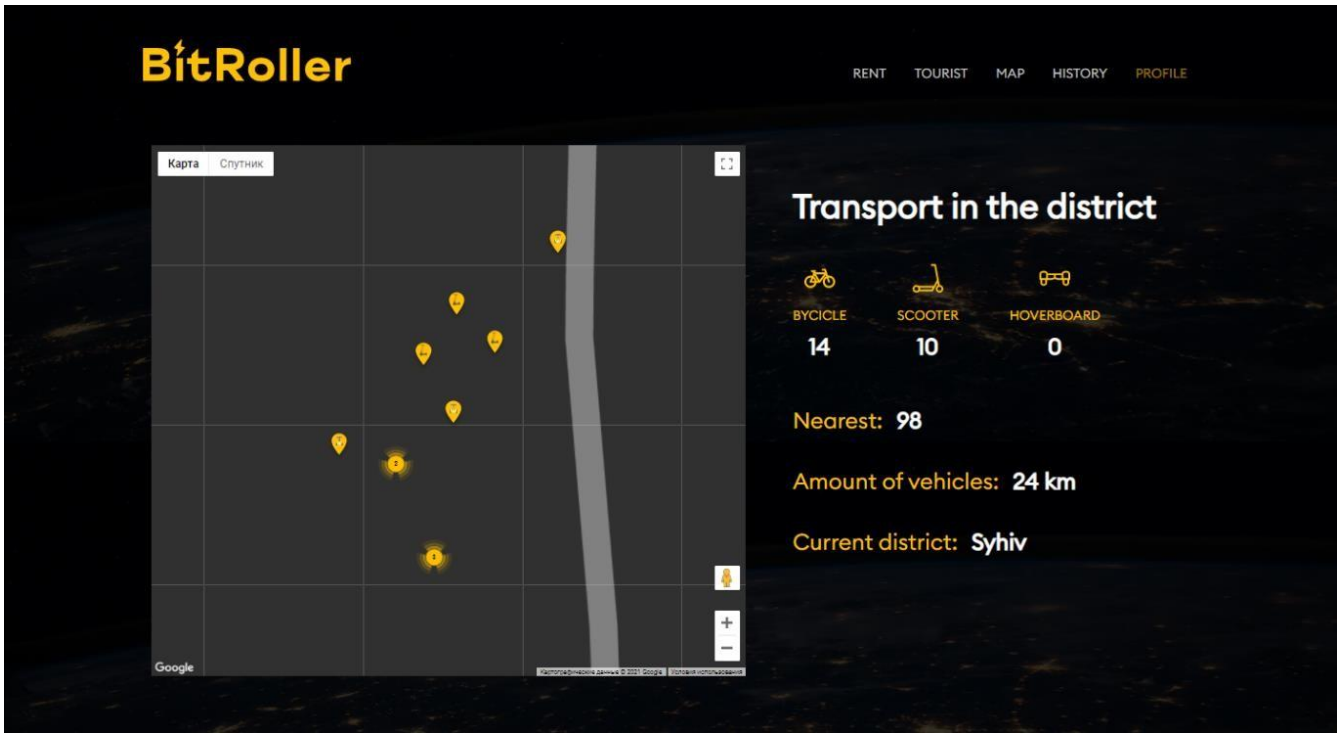


Рисунок 5.5 – Сторінка з картою

В лівій частині екрану, можемо побачити знайому всім нам google карту, але в модернових темних тонах. Але можемо побачити, що на відміну від класичного

google карти, тут в нас наявні всілякі маркери. Цими маркерами і позначається наш електротранспорт поблизу. Як можна замітити, маркери є різних типів, кожен тип маркера відповідає своєму типу компактного електричного транспорту. З правого ж боку знаходиться інформаційна панель, яка демонструє користувачеві інформацію, щодо доступного транспорту в його місці перебування. Як бачимо доступна інформація про кількість транспорту кожного типу, доступного в певному районі міста, район вибирається відповідно до місця розташування користувача. Також доступна інформація стосовно відстані до найближчого пристрою.

Якщо користувач таки вирішив скористуватися нашими послугами, йому потрібно підійти до будь-якого з доступних пристроїв та натиснути на маркер, який позначений на карті. В результаті цього, відкриється діалогове вікно з даними про цей транспорт та в виділеній області відкриється камера для сканування qr-коду. (рис. 5.6).

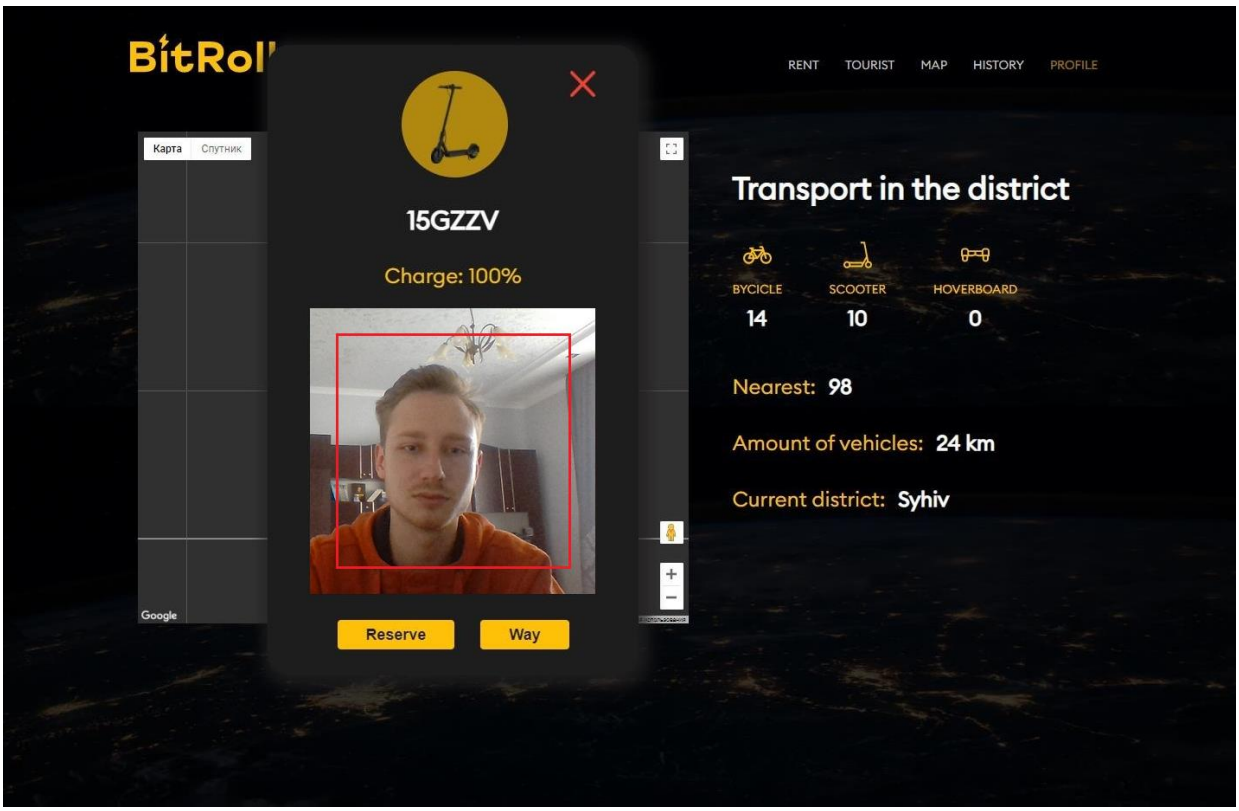


Рисунок 5.6 - Відкрите діалогове вікно

На цьому діалоговому вікні, як вже згадувалося раніше, нам доступна

інформація про обраний пристрій, то ж розглянемо детальніше, що з цього може пригодитися користувачеві. Перш за все, ми можемо побачити картинку того типу пристрою, який ми обрали. Наступним йде унікальний номер пристрою, він може пригодитися користувачеві для того, щоб сповістити в службу технічної підтримку в разі виникнення поломки, чи певних інших неприємних ситуацій, які відносяться саме до цього девайсу. Звичайно нам доступний сканер qr-коду, відсканувавши відповідний qr-код на пристрої, користувач отримає змогу почати поїздку. Хочу зауважити, що на рис. 5.6 відкрита фронтальна камера, це результат того, що для тестування я використовував ноутбук і це ще раз підтверджує, що тепер можна орендувати електротранспорт і з розрядженим смартфоном, але маючи при собі ноутбук з доступом до інтернету. На мобільних пристроях, по стандарту буде відкриватися основна камера. Наступним в очі кидається копка «Way», яка зумовлює собою функціонал прокладання маршруту до цього самокату (рис. 5.7), що буде дуже корисним, коли самокат знаходиться, наприклад, в якомусь провулку. Також можемо побачити і кнопку резервування, яка дозволяє зарезервувати пристрій на певний час, тим самим убезпечивши вас від прикрої ситуації, коли хтось забере пристрій у вас з-під носа.

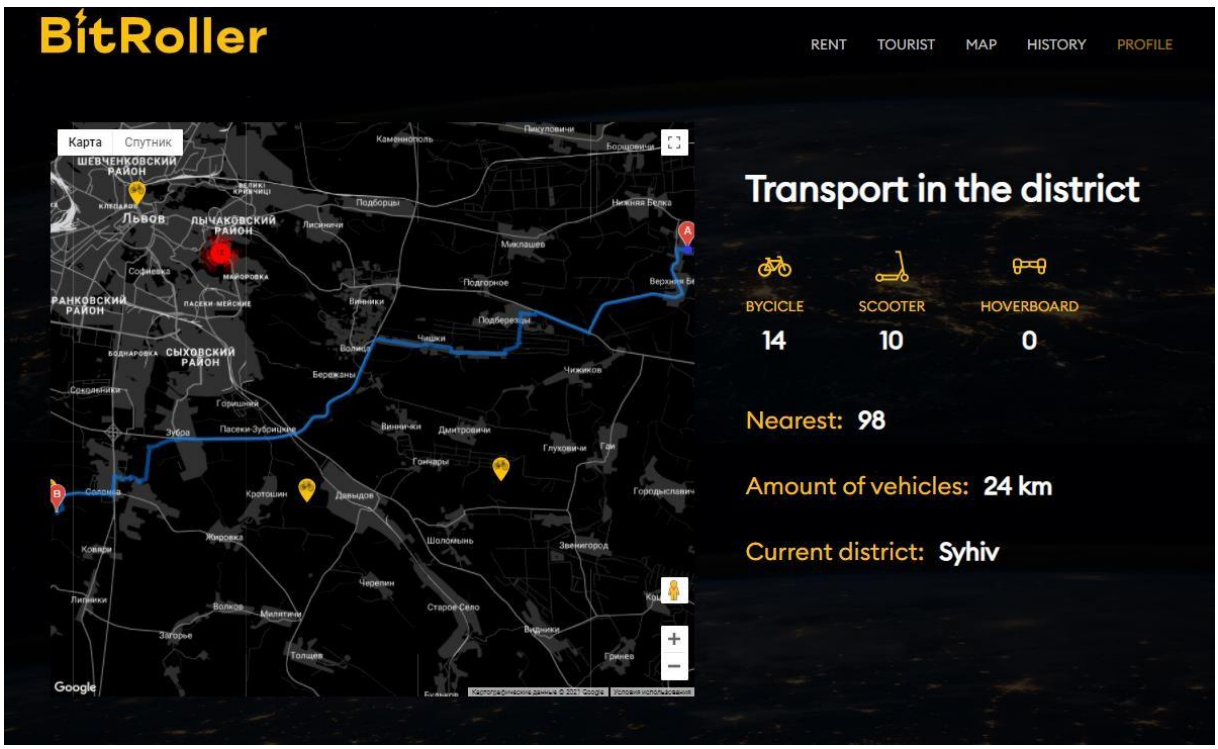


Рисунок 5.7 – Прокладання шляху до пристрою

Детальніше глянути на приклад сканування qr-коду можна на рисунку нижче.

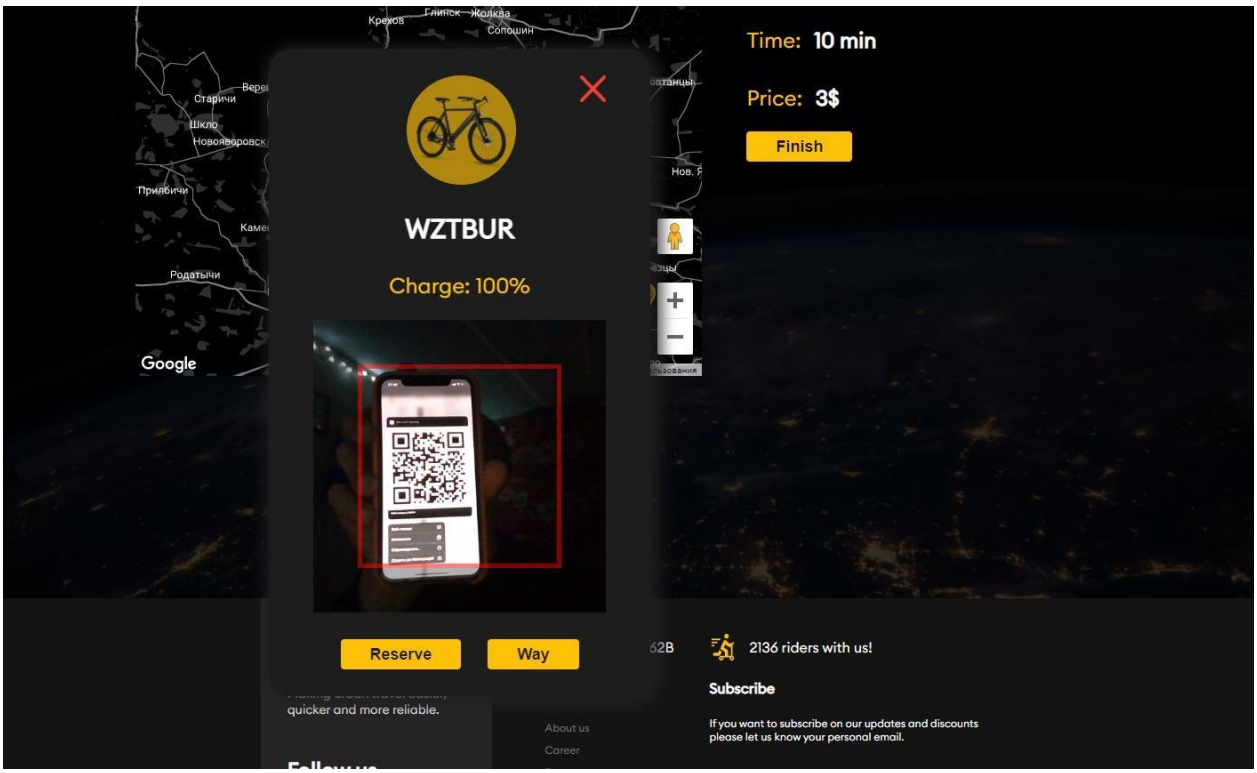


Рисунок 5.8 – Сканування qr-коду

Після сканування qr-коду відбувається запит на сервер, в наслідок чого розпочинається поїдка. Під «розпочинається поїздка», я маю на увазі запуск таких процесів як: початок відліку часу поїздки, розрахунок вартості поїздки, відповідно до встановленого тарифу, також не варто забувати, що на період поїздки, маркер електротранспорту зникає з карти, що в свою чергу робить його недоступним для інших користувачів. Після сканування коду та початку поїздки, користувач зустрічає вже інше вікно (рис. 5.9), на якому він може бачити вже інформацію не про пристрій, а про поїздку. Тут можна побачити обраний тип транспорту, час поїздки, та розрахована вартість поїздки. Також, що є плюсом, в лівій частині екрану ви досі можете користуватися картою. Для того, щоб завершити поїздку, вам потрібно зупинити електротранспорт і натиснути на кнопку «Finish». Після цього з вашої карти буде знято розраховану вартість поїздки, а маркер девайсу появиться в тому місці, де зупинився користувач.

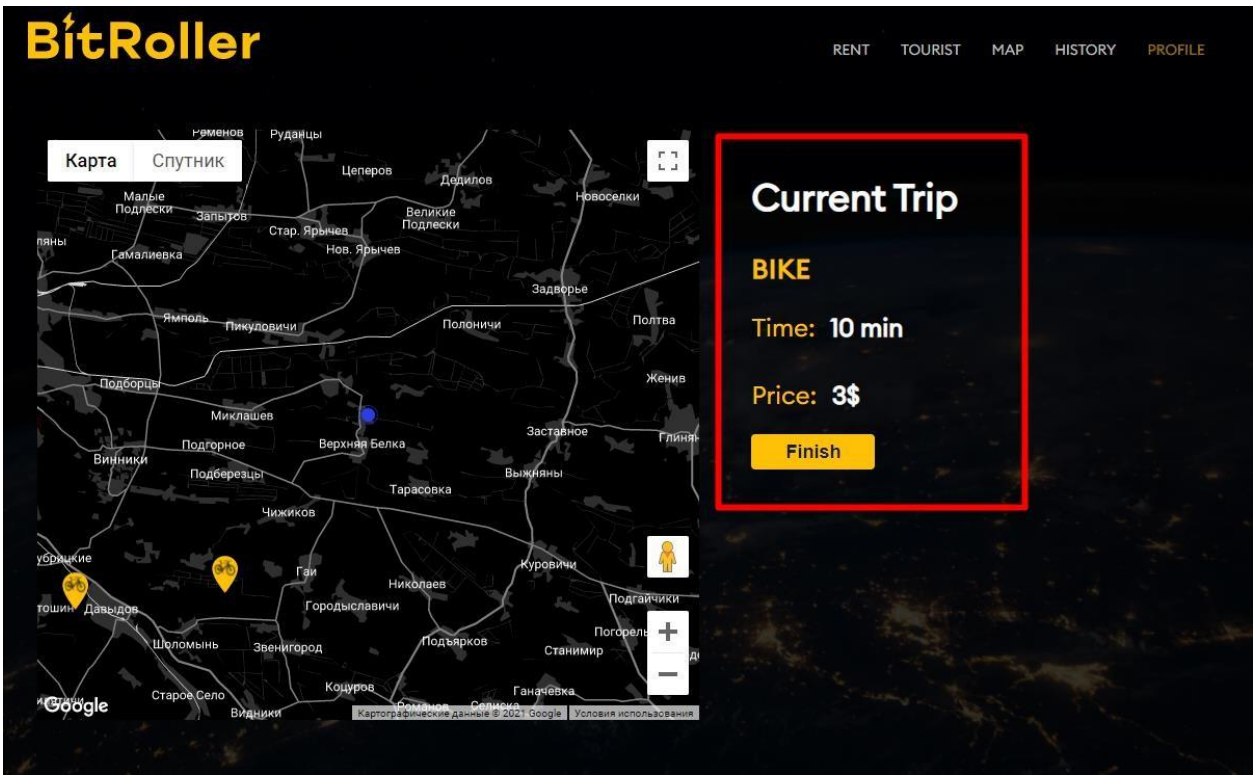


Рисунок 5.9 – Сторінка активної поїздки

Після завершення поїздки на сторінці history з’являється нова позиція (рис. 5.10).

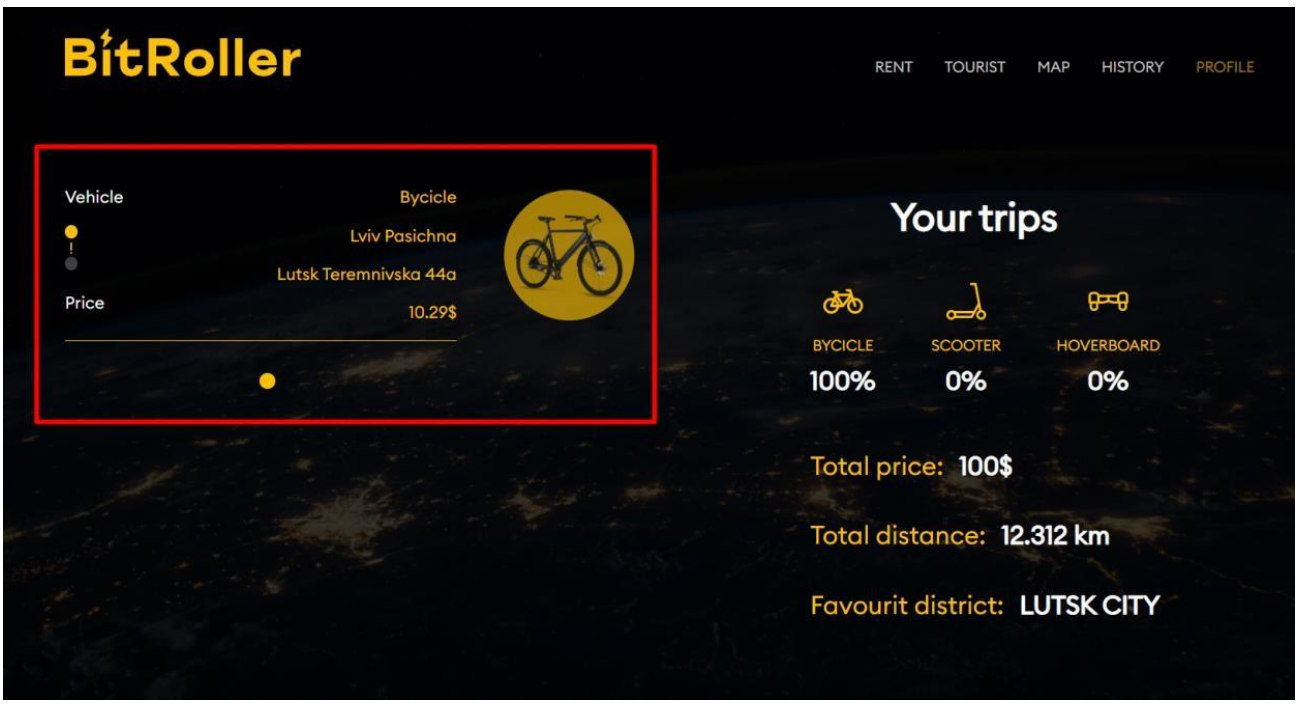


Рисунок 5.10 – Додана поїздка. На цій сторінці відображаються всі поїздки, які були в користувача

Висновки до розділу

В даному розділі було сформульовано ідею стартап-проекту та розроблено ринкову та маркетингову стратегію. Проаналізувавши наведену інформацію, можна зробити висновок, що при правильному управлінні стартапом, та за сприятливих зовнішніх чинників, проект має вийти прибутковим, обійшовши конкурентів завдяки ширшому функціоналу та відкритості до реалізації нових ідей.

Також було представлено роботу створеного сервісу для прокату мобільного електротранспорту. У висновку, хочу сказати, що інтерфейс додатку вийшов лаконічним, в хорошій колірній гамі, що буде мати суттєвий вплив на досвід користування додатком кінцевого користувача. Тяжко не підмітити, що розроблений додаток відмічається своєю адаптивністю, адже однаково зручний та продуктивний, що на телефоні, що на ноутбуці.

ВИСНОВКИ

Дипломний проект присвячений вивченню створення комерційних IT-проектів для мультиплатформ. Під час його написання, було розроблено REST API використовуючи мікрофреймворк flask та проведено тестування API через засіб тестування Postman. Розроблено та проведено конфігурацію власного серверу, піднятого на ОС Linux. Також для переадресації даних проведено конфігурацію веб-сервера та проксі. В якості бази даних було вибрано графову базу даних Neo4j. Проаналізовано головні способи захисту даних, в результаті дослідження та аналізу, було обрано метод JWT токенів та реалізовано його в програмному продукті. Реалізовано алгоритм кластеризації K-means, для визначення та демонстрації найбільш активних ділянок використання сервісу.

Також було розроблено frontend-частину для сервісу прокату мобільного електротранспорту з використанням мови програмування JavaScript, та використовуючи фреймворк React. Було досліджено різноманітні технології та методики по розробці веб-додатків, внаслідок глибокого аналізу, було відібране лише найкраще та найбільш підходяще і втілене в життя під час написання додатку. Приділено чимало уваги UI/UX дизайну, досліджено різноманітні технології та інструменти, для створення графічної частини проекту, всі результати дослідження та наглядні приклади використання описано в записці до дипломного проекту. В загальному, можна сказати, що мій проект полягав в розв’язанні актуальної проблеми на теперішній час, при цьому враховуючи всі аспекти комерційної розробки.

Розроблена ринкова та маркетингова стратегія, також внаслідок аналізу даної теми, було прораховано усі можливі ризики які можуть виникнути під час реалізацію цього стартап проекту. Досліджена потреба простих людей та в результаті впроваджено інноваційний функціонал в сервісі прокату мобільного електротранспорту, який відповідав би потребі людей і породжував пропозицію на їхній попит. Загалом було представлене вирішення глобальних проблем, пов’язаних з екологією та перенасиченістю транспорту на дорогах великих міст.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Оліщук Андрій Володимирович Розробка Web-додатків на PHP 5. Професійна робота. — М.: «Вільямс», 2006. — С. 352.
2. Засоби розробки веб-додатків: [Електронний ресурс] – Режим доступу: https://paperform.co/blog/web-development-tools
3. Моделі життєвого циклу: [Електронний ресурс] – Режим доступу: https://evergreens.com.ua/ua/articles/software-development-metodologies.html
4. UI/UXdesign: [Електронний ресурс] – Режим доступу: https://dan-it.com.ua/uk/vse-pro-profesiju-ui-ux-dizajnera/
5. Фреймворк React: [Офіційна документація] – Режим доступу: https://reactjs.org/
6. Adam Boduch. Flux Architecture. — Packt Publishing, 2016. — 352 с.
7. Тестування: Основні визначення, аксіоми та принципи. Дідковська М.В., Тимошенко Ю.О., - mmsa.kpi.ua, 2010 – 15 с.
8. Jest-бібліотека: [Офіційна документація] – Режим доступу: https://jestjs.io/uk/docs/tutorial-react
9. Git – книга GitMagic – [Електронний ресурс] – Режим доступу: https://jestjs.io/uk/docs/tutorial-react
10. Рішення по дизайну (Figma) – [Електронний ресурс] – Режим доступу: https://www.figma.com/
11. Карти Google – [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Карти_Google
12. Backend – [Електронний ресурс] – Режим доступу: https://techterms.com/definition/backend
13. AsyncIo – [Електронний ресурс] – Режим доступу: https://realpython.com/async-io-python/
14. Comparative study on Python web frameworks: Flask and Django – [Електронний ресурс] – Режим доступу:https://core.ac.uk/download/pdf/323461341.pdf

15. Python web frameworks – [Електронний ресурс] – Режим доступу: https://www.infoworld.com/article/3105502/review-13-python-web-frameworks-compared.html?page=2

16. What Is Apache Web Server? A Basic Look at What It Is and How It Works – [Електронний ресурс] – Режим доступу: https://kinsta.com/knowledgebase/what-is-apache/

17. Apache HTTP Server – [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Apache_HTTP_Server

18. Weighted K-Means Clustering of GPS Coordinates — Python. – [Електронний ресурс] – Режим доступу: https://medium.datadriveninvestor.com/weighted-k-means-clustering-of-gps-coordinates-python-7c6270846163

19. Security of JSON Web Tokens (JWT) – [Електронний ресурс] – Режим доступу: https://cyberpolygon.com/materials/security-of-json-web-tokens-jwt/

20. П’ять простих кроків для розуміння JSON Web Tokens (JWT) – [Електронний ресурс] – Режим доступу: https://habr.com/ru/post/340146/

21. Django vs. Flask in 2021: Which Framework to Choose - https://testdriven.io/blog/django-vs-flask/

ДОДАТКИ

ДОДАТОК А

Лістинг коду, використаного в backend частині додатку.

init.py

```
from app.config import cfg, bcolors
from aioflask import Flask
from flask_restful import Api
from flask_jwt_extended import JWTManager
from flask_bcrypt import Bcrypt
from flask_bootstrap import Bootstrap4
from app.admin_api.google_maps_integration import *


app = Flask(__name__, template_folder=(cfg['path']['template_folder']),
static_folder=(cfg['path']['static_folder']))

# Initialize the extension
api = Api(app)
jwt = JWTManager(app)
bcrypt = Bcrypt(app)
Bootstrap4(app)
bcolors()

from app.webports import *
from app.admin_api.admin_create import *

if __name__ == '__main__':
    app.run(host=(cfg['app']['host']),
                port=(cfg['app']['port']), debug=(cfg['app']['debug']),
use_reloader=(cfg['app']['use_reloader']))
```

config.yml

```
app:
   debug: True
   use_reloader: True
```

```
    host: '0.0.0.0'
    port: 25565
    SECRET_KEY: '5dca-91ab-d8b4bd727450'
    JWT_SECRET_KEY: 'ae7536b5-53a4-5dca'
    JWT_BLACKLIST_ENABLED: True
    JWT_COOKIE_SECURE: False
    JWT_COOKIE_CSRF_PROTECT: False
    jWT_COOKIE_CSRF_PROTECT: False
    JWT_ACCESS_TOKEN_EXPIRES: 3600
    enable_cookies: False
neo4j:
    db_user: 'neo4j'
    db_pass: '1q2w3e4r'
    db_host: bolt://localhost:7687
path:
    template_folder: './app/admin_api/templates'
    static_folder: './app/admin_api/static'
    vehicle_data: 'app/admin_api/static/vehicle_data/'
    qr_folder: 'http://localhost:25565/static/vehicle_data/'
google_api:
    key: "AIzaSyCOpbx_dezg9g6_LWAiLW0RmU5fDqHtlyw"
admin:
    post_limit: 100
management:
    price_cof: 1.5
```

admin_create.py

```
from aioflask import render_template, request
from datetime import datetime
from init import app, cfg
from app.database.database import *
from app.admin_api.qrcode_generic import create_qr_code
from app.app_func import *
import uuid
import os


google_api_key = (cfg['google_api']['key'])
```

```
@app.route("/api/v1/admin", methods=["GET", "POST"])
async def admin_data():
    users, users_num, vehicles, vehicles_num, trips, trips_num = await
get_bit_data_from_db()
    coordinates_list = await get_cords_list(vehicles)
    folder_path = (cfg['path']['vehicle_data'])
    if request.method == 'POST':
        login_result = request.form
        vehicle_type = login_result['Type']
        vehicle_lat = login_result['lat']
        vehicle_lng = login_result['lng']
        lat = float(vehicle_lat)
        lng = float(vehicle_lng)
        location = str({'lat': lat, 'lng': lng})
        vehicle_charge = login_result['charge']
        reg_time = datetime.utcnow()
        Vehicle_key = str(vehicle_type) + "reg" + str(reg_time)
        create_uuid = uuid.uuid5(uuid.NAMESPACE_URL, Vehicle_key)
        vehicle_uuid = str(create_uuid)
        if not os.path.exists(folder_path + vehicle_uuid):
            os.mkdir(folder_path + vehicle_uuid)
        qr_code, img_url, qr_data = await create_qr_code(vehicle_uuid)
        qr_code.save(folder_path + vehicle_uuid + '/qr_code_image.png')
        vehicle_code = await id_generator()
        await create_vehicle(vehicle_uuid, vehicle_type, img_url, vehicle_code,
int(vehicle_charge), location)
        print(f"{bcolors.OKGREEN}[Vehicle with uuid ({vehicle_uuid}) and type
({vehicle_type}) was created!]{bcolors.ENDC}", flush=True)
        users, users_num, vehicles, vehicles_num, trips, trips_num = await
get_bit_data_from_db()
        coordinates_list = await get_cords_list(vehicles)
        return await render_template("create.html", vehicle_uuid=vehicle_uuid,
img_url=img_url, qr_data=qr_data, VehicleType=vehicle_type,
VehicleCode=vehicle_code, vehicle_charge=vehicle_charge, users=users,
users_num=users_num, vehicles=vehicles, vehicles_num=vehicles_num, trips=trips,
trips_num=trips_num, coordinates_list=coordinates_list,
google_api_key=google_api_key)
    else:
        print(f"{bcolors.FAIL}[Can not create vehicle]{bcolors.ENDC}", flush=True)
    return await render_template("create.html", users=users, users_num=users_num,
vehicles=vehicles, vehicles_num=vehicles_num, trips=trips, trips_num=trips_num,
```

```
coordinates_list=coordinates_list, google_api_key=google_api_key)
```

conection.py

```
from init import cfg, bcolors
from py2neo import Graph


def db_auth():
    user = (cfg['neo4j']['db_user'])
    pwd = (cfg['neo4j']['db_pass'])
    url = (cfg['neo4j']['db_host'])
    try:
        graph = Graph(url, user=user, password=pwd)
        if graph is not None:
            print(f"{bcolors.HEADER}[Successfully connected to
database!]{bcolors.ENDC}", flush=True)
    except Exception as e:
        print(f"{bcolors.FAIL}Failed to create the driver:{bcolors.ENDC}", e,
flush=True)
    return graph
```

database.py

```
from app.database.conection import db_auth
from typing import Optional
from app.model.User import User
from app.model.Vehicle import Vehicle
from app.model.Trip import Trip
from app.model.Position import Position
from app.config import cfg, bcolors
neo4j = db_auth()

def find_uuid(email: str):
    user_uuid = neo4j.evaluate(f"match (a:User{{email:'{email}'}}) return a")
    if user_uuid is None:
        return None
    else:
        uuid = user_uuid['uuid']
    return uuid
def find_user_trips(uuid: str):
    user_trips = neo4j.run(f"match (a:Trip{{user_uuid:'{uuid}'}})  WHERE
```

```
a.end_location IS NOT NULL  return a ORDER BY a DESC").data()
    if user_trips is None:
        return None
    return [dict(temp["a"]) for temp in user_trips]

def find_user_current_trip(uuid: str):
    user_current_trip = neo4j.evaluate(f"MATCH (a:Trip) WHERE
a.trip_uuid='{uuid}'RETURN a")
    if user_current_trip is None:
        return None
    return user_current_trip

def find_user_trips_data(uuid: str):
    bike_data = neo4j.run(f"MATCH (a:Trip) WHERE a.user_uuid='{uuid}' and
a.vehicle='bike' RETURN a").data()
    hoverboard_data = neo4j.run(f"MATCH (c:Trip) WHERE c.user_uuid='{uuid}' and
c.vehicle='hoverboard' RETURN c").data()
    scooter_data = neo4j.run(f"MATCH (b:Trip) WHERE b.user_uuid='{uuid}' and
b.vehicle='scooter' RETURN b").data()
    len_bike_data = len(bike_data)
    len_scooter_data = len(scooter_data)
    len_hoverboard_data = len(hoverboard_data)
    all_tips = len_bike_data + len_hoverboard_data + len_scooter_data
    if all_tips != 0:
        pers_bike_data = len_bike_data/all_tips
        pers_bike = ("%.2f" % pers_bike_data)
        pers_hoverboard_data = len_hoverboard_data/all_tips
        hoverboard_data = ("%.2f" % pers_hoverboard_data)
        pers_scooter_data = len_scooter_data/all_tips
        scooter_data = ("%.2f" % pers_scooter_data)
        return pers_bike, hoverboard_data, scooter_data
    return None, None, None

def find_vehicle(uuid: str):
    vehicle_data = neo4j.evaluate(f"match (a:Vehicle{{uuid:'{uuid}'}}) return a")
    if vehicle_data is None:
        return None
    return vehicle_data

async def get_bit_data_from_db():
    limit = (cfg['admin']['post_limit'])
```

```
    users_data = neo4j.run(f"match (a:User) return a ORDER BY a DESC limit
{limit}").data()
    users_num = neo4j.evaluate(f"match (a:User) return COUNT(a)")

    vehicles_data = neo4j.run(f"match (b:Vehicle) return b ORDER BY b DESC limit
{limit}").data()
    vehicles_num = neo4j.evaluate(f"match (b:Vehicle) return COUNT(b)")

    trips_data = neo4j.run(f"match (c:Trip) return c ORDER BY c DESC limit
{limit}").data()
    trips_num = neo4j.evaluate(f"match (c:Trip) return COUNT(c)")

    users = [dict(temp["a"]) for temp in users_data]
    vehicles = [dict(temp["b"]) for temp in vehicles_data]
    trips = [dict(temp["c"]) for temp in trips_data]
    return users, users_num, vehicles, vehicles_num, trips, trips_num

def get_loc_data_from_db():
    trips_data = neo4j.run(f"match (c:Trip) WHERE c.end_location IS NOT NULL
return c ").data()
    trips = [dict(temp["c"]) for temp in trips_data]
    return trips

def find_vehicle_by_code(vehicle_code: str):
    vehicle_data = neo4j.evaluate(f"match
(a:Vehicle{{vehicle_code:'{vehicle_code}'}}) return a")
    if vehicle_data is None:
        return None
    return vehicle_data

def find_vehicle_by_args(vehicle_code: str, vehicle_uuid: str):
    if vehicle_code is not None:
        vehicle_data = neo4j.evaluate(f"match
(a:Vehicle{{vehicle_code:'{vehicle_code}'}}) return a")
        return vehicle_data
    if vehicle_uuid is not None:
        vehicle_data = neo4j.evaluate(f"match (a:Vehicle{{uuid:'{vehicle_uuid}'}})
return a")
        return vehicle_data
```

```
def activate_vehicle(vehicle_uuid: str, user_uuid: str):
    change_vehicle = neo4j.evaluate(f"MATCH(x: Vehicle) WHERE x.uuid =
'{vehicle_uuid}' SET x.is_active = True, x.current_User = '{user_uuid}' RETURN
x.is_active as is_active, x.current_User as current_User")
    change_vehicle_user = neo4j.evaluate(f"MATCH(x: User) WHERE x.uuid =
'{user_uuid}' SET x.current_Vehicle = '{vehicle_uuid}' RETURN x.current_Vehicle as
current_Vehicle")
    print(f"{bcolors.WARNING}[User ({user_uuid}) activate vehicle with uuid
({vehicle_uuid})!]{bcolors.ENDC}", flush=True)
    return change_vehicle, change_vehicle_user

def end_trip(vehicle_uuid: str, user_uuid: str, end_data: int, trip_uuid: str,
end_location: str, price: float):
    set_end_data = neo4j.evaluate(f"MATCH(x: Trip) WHERE x.trip_uuid =
'{trip_uuid}' SET x.end_data = {end_data}, x.end_location = '{end_location}',
x.price = {price} RETURN x.end_data as end_data, x.end_location as end_location,
x.price as price")
    clear_vehicle = neo4j.evaluate(f"MATCH(x: Vehicle) WHERE x.uuid =
'{vehicle_uuid}' SET x.is_active = False, x.current_User = null,
x.current_position = '{end_location}' RETURN x.is_active as is_active,
x.current_User as current_User, x.current_position as current_position")
    clear_vehicle_user = neo4j.evaluate(f"MATCH(x: User) WHERE x.uuid =
'{user_uuid}' SET x.current_Vehicle = null RETURN x.current_Vehicle as
current_Vehicle")
    null_current_trip = neo4j.evaluate(f"MATCH(x: User) WHERE x.current_trip =
'{trip_uuid}' SET x.current_trip = null RETURN x.current_trip as current_trip")
    print(f"{bcolors.WARNING}[User ({user_uuid}) end trip with vehicle
({vehicle_uuid})!]{bcolors.ENDC}")
    return clear_vehicle, clear_vehicle_user, set_end_data, null_current_trip

def if_block(email: str):
    user_uuid = neo4j.evaluate(f"match (a:User{{email:'{email}'}}) return a")
    if user_uuid is None:
        return False
    else:
        uuid = user_uuid['blocked']
    return uuid

def get_user(uuid: str):
    user_data = neo4j.evaluate(f"match (a:User{{uuid:'{uuid}'}}) return a")
    if user_data is None:
```

```
        return None
    return user_data

def find_email(email: str):
    user_data = neo4j.evaluate(f"match (a:User{{email:'{email}'}}) return a")
    if user_data is None:
        return None
    else:
        email = user_data['email']
    return email

def find_number(phoneNumber: str):
    user_data = neo4j.evaluate(f"match (a:User{{phoneNumber:'{phoneNumber}'}})
return a")
    if user_data is None:
        return None
    else:
        email = user_data['phoneNumber']
    return email

def get_password(email: str):
    user_data = neo4j.evaluate(f"match (a:User{{email:'{email}'}}) return a")
    if user_data is None:
        return None
    else:
        password_hash = user_data['password_hash']
    return password_hash

def create_user(user_uuid: str, email: str, fullname: str,  password_hash: str) ->
Optional[User]:
    if find_email(email):
        return None
    user = User()
    user.node_type = 'USER'
    user.uuid = user_uuid
    user.email = email
    user.fullname = fullname
    user.profileImage = "https://img.icons8.com/bubbles/2x/user-male.png"
    user.password_hash = password_hash
    user.blocked = False
    user.current_Vehicle = None
```

```
    user.current_trip = None
    neo4j.create(user)
    print(f"{bcolors.OKGREEN}[User successfully added!]{bcolors.ENDC}",
flush=True)
    return user

def create_trip(user_uuid: str, vehicle_uuid: str, trip_uuid: str,  vehicle: str,
start_data: int, start_location: str) -> Optional[User]:
    trip = Trip()
    trip.node_type = 'TRIP'
    trip.user_uuid = user_uuid
    trip.trip_uuid = trip_uuid
    trip.vehicle_uuid = vehicle_uuid
    trip.startAddress = "Lviv Pasichna"
    trip.endAddress = "Lutsk Teremnivska 44a"
    trip.start_location = start_location
    trip.end_location = None
    trip.price = 0
    trip.vehicle = vehicle
    trip.start_data = start_data
    trip.end_data = None
    neo4j.create(trip)
    create_rel = neo4j.evaluate(
        f" MATCH(a: User), (b : Trip) WHERE a.uuid = '{user_uuid}' and b.trip_uuid
= '{trip_uuid}' create(a) - [r: HAS]->(b) RETURN a, b")
    create_rel_vehicle = neo4j.evaluate(
        f" MATCH(a: Vehicle), (b : Trip) WHERE a.uuid = '{vehicle_uuid}' and
b.trip_uuid = '{trip_uuid}' create(b) - [r: USE]->(a) RETURN a, b")
    set_end_data = neo4j.evaluate(
        f"MATCH(x: User) WHERE x.uuid = '{user_uuid}' SET x.current_trip =
'{trip_uuid}' RETURN x.current_trip as current_trip")
    set_current_user = neo4j.evaluate(
        f"MATCH(x: Vehicle) WHERE x.uuid = '{vehicle_uuid}' SET x.current_User=
'{user_uuid}' RETURN x.current_User as current_User")
    print(f"{bcolors.OKGREEN}[Trip successfully created!]{bcolors.ENDC}",
flush=True)
    return trip, set_end_data, create_rel, create_rel_vehicle, set_current_user


def set_pos(lat: str, lon: str) -> Optional[User]:
    position = Position()
```

```
    position.lat = lat
    position.lon = lon
    neo4j.create(position)
    return position

async def create_vehicle(vehicle_uuid: str, VehicleType: str, img_url: str,
vehicle_code: str, vehicle_charge: int, location: str)-> Optional[User]:
    vehicle = Vehicle()

    vehicle.user_type = 'VEHICLE'
    vehicle.uuid = vehicle_uuid
    vehicle.Vehicle_type = VehicleType
    vehicle.img_url = img_url
    vehicle.is_active = False
    vehicle.current_User = None
    vehicle.vehicle_code = vehicle_code
    vehicle.vehicle_charge = vehicle_charge
    vehicle.current_position = location
    neo4j.create(vehicle)
    print(f"{bcolors.OKGREEN}[Vehicle successfully added!]{bcolors.ENDC}",
flush=True)
    return vehicle

def get_vehicles():
    get_all_vehicles = neo4j.run(f"MATCH (x:Vehicle) WHERE x.is_active = False
RETURN x").data()
    vehicles_list = [dict(temp["x"]) for temp in get_all_vehicles]
    return vehicles_list

def edit_user_email(uuid: str, email: str) -> Optional[User]:
    new_user_email = neo4j.run(f"MATCH (x:User) WHERE x.uuid='{uuid}' SET x.email
= '{email}' RETURN x.email as email")
    return new_user_email

def edit_user_number(uuid: str, phoneNumber: str) -> Optional[User]:
    new_user_email = neo4j.run(f"MATCH (x:User) WHERE x.uuid='{uuid}' SET
x.phoneNumber = '{phoneNumber}' RETURN x.phoneNumber as phoneNumber")
    return new_user_email

def get_profile(uuid: str) -> Optional[User]:
    user_profile = neo4j.run(
```

```
            f"MATCH (x:User) WHERE x.uuid='{uuid}' RETURN x.email as email, x.fullname
        as fullname, x.blocked as blocked")
            return user_profile
```

config.py

```
        from datetime import timedelta
        from flask_cors import CORS
        import yaml
        from app.server_controll import bcolors
        with open("config.yml", "r") as app_config:
            cfg = yaml.load(app_config, Loader=yaml.FullLoader)
            print(f"{bcolors.HEADER}[Config loaded!]{bcolors.ENDC}", flush=True)
        from init import app, cfg

        app_cors = CORS(app, resources={r"/*": {"origins": "*"}},
        supports_credentials=True)
        app.config['SECRET_KEY'] = (cfg['app']['SECRET_KEY'])
        app.config['JWT_SECRET_KEY'] = (cfg['app']['JWT_SECRET_KEY'])
        app.config['JWT_BLACKLIST_ENABLED'] = (cfg['app']['JWT_BLACKLIST_ENABLED'])
        app.config['JWT_BLACKLIST_TOKEN_CHECKS'] = ['access', 'refresh']
        app.config["JWT_ACCESS_TOKEN_EXPIRES"] =
        timedelta(seconds=(cfg['app']['JWT_ACCESS_TOKEN_EXPIRES']))
        app.config['GOOGLEMAPS_KEY'] = (cfg['google_api']['key'])
        if (cfg['app']['enable_cookies']) is True:
        app.config["JWT_TOKEN_LOCATION"] = ["cookies"]
        app.config["JWT_COOKIE_CSRF_PROTECT"] = (cfg['app']['JWT_COOKIE_CSRF_PROTECT'])
        app.config["JWT_COOKIE_SECURE"] = (cfg['app']['JWT_COOKIE_SECURE'])
```

webports.py

```
        from app.app_func import *
        from datetime import datetime
        from datetime import timedelta
        from datetime import timezone
        from init import app, bcrypt
        from app.database.database import *
        from aioflask import jsonify, request
        from flask_jwt_extended import (jwt_required, create_access_token,
get_jwt_identity, create_refresh_token, get_jwt, set_access_cookies,
set_refresh_cookies, unset_jwt_cookies)
        from app.saferequest import SafeJson
```

```
import uuid

@app.after_request
def refresh_expiring_jwts(response):
    try:
        exp_timestamp = get_jwt()["exp"]
        now = datetime.now(timezone.utc)
        target_timestamp = datetime.timestamp(now + timedelta(seconds=1800))
        if target_timestamp > exp_timestamp:
            access_token = create_access_token(identity=get_jwt_identity())
            print(f"{bcolors.WARNING}[Token refreshed]{bcolors.ENDC}")
            set_access_cookies(response, access_token)
        return response
    except (RuntimeError, KeyError):
        # Case where there is not a valid JWT. Just return the original respone
        return response

@app.route("/", methods=["GET"])
async def main_page():
    return jsonify(msg="Welcome to Bitroller API"), 200

@app.route('/api/v1/refresh', methods=['POST'])
@jwt_required(refresh=True)
def refresh():
    current_user = get_jwt_identity()
    access_token = create_access_token(identity=current_user, fresh=False)
    refresh_token = create_refresh_token(identity=current_user)
    response = jsonify(access_token=access_token, refresh_token=refresh_token)
    set_access_cookies(response, access_token)
    set_refresh_cookies(response, refresh_token)
    return response

@app.route("/google/auth", methods=["POST"])
def google_auth():
    return

@app.route("/api/v1/sign-in", methods=["POST"])
def sign_in():
    if request.method == 'POST':
        login_json = SafeJson(request.get_json())
        message = login_json.init_keys(['email', str], ['password', str])
```

```
            if message is None:
                email, password = login_json.get_values()
                data_email = find_email(email)
                block_user = if_block(email)
                if block_user is False:
                    if data_email is not None and data_email == email:
                        data_password =
bcrypt.check_password_hash(get_password(email), password)
                        if data_password is True:
                            user_uuid = find_uuid(email)
                            access_token = create_access_token(identity=user_uuid)
                            refresh_token = create_refresh_token(identity=user_uuid)
                            response = jsonify(access_token=access_token,
refresh_token=refresh_token)
                            set_access_cookies(response, access_token)
                            set_refresh_cookies(response, refresh_token)
                            print(f"{bcolors.OKGREEN}[User with email ({email}) and
uuid  ({user_uuid}) login in!]{bcolors.ENDC}", flush=True)
                            return response
                        else:
                            return jsonify(msg="Bad Password!"), 401
                    else:
                        return jsonify(msg="User not found!"), 401
                else:
                    return jsonify(msg="User is blocked!"), 401
            else:
                return jsonify(msg="Fill all Fields!"), 400
        else:
            return jsonify(msg="Bad request!"), 400


    @app.route("/api/v1/sign-up", methods=["POST"])
    def sign_up():
        if request.method == 'POST':
            register_json = SafeJson(request.get_json())
            message = register_json.init_keys(['email', str], ['fullname', str],
['password', str])
            if not message:
                email, fullname, password = register_json.get_values()
                password_hash = bcrypt.generate_password_hash(password)
                reg_time = datetime.utcnow()
                user_key = str(email) + "reg" + str(reg_time)
```

```
                create_uuid = uuid.uuid5(uuid.NAMESPACE_URL, user_key)
                user_uuid = str(create_uuid)
                user = create_user(user_uuid, email, fullname, password_hash)
                if not user:
                    return jsonify(msg="A user with that email already exists!"), 400
                access_token = create_access_token(identity=user_uuid)
                refresh_token = create_refresh_token(identity=user_uuid)
                print(f"{bcolors.OKGREEN}[User with email ({email}) and uuid
({user_uuid}) created!]{bcolors.ENDC}", flush=True)
                response = jsonify(access_token=access_token,
refresh_token=refresh_token)
                set_access_cookies(response, access_token)
                set_refresh_cookies(response, refresh_token)
                return response
            else:
                return jsonify(msg="Fill all Fields!"), 400
        else:
            return jsonify(msg="Bad request!"), 400

    @app.route("/api/v1/sign-out", methods=["DELETE"])
    @jwt_required()
    def sign_out():
        response = jsonify({"msg": "Logout successful"})
        uuid = get_jwt_identity()
        print(f"{bcolors.WARNING}[User with uuid  ({uuid}) sign out!]{bcolors.ENDC}",
flush=True)
        unset_jwt_cookies(response)
        return response

    @app.route("/api/v1/profile", methods=["GET", "POST"])
    @jwt_required()
    def profile():
        if request.method == 'GET':
            current_user = get_jwt_identity()
            profile_data = get_user(current_user)
            email = profile_data['email']
            fullname = profile_data['fullname']
            current_trip = profile_data['current_trip']
            uuid = profile_data['uuid']
            phoneNumber = profile_data['phoneNumber']
            profileImage = profile_data['profileImage']
```

```
                current_Vehicle = profile_data['current_Vehicle']
                return jsonify(email=email, fullname=fullname, id=uuid,
profileImage=profileImage, phoneNumber=phoneNumber, current_Vehicle=current_Vehicle,
trip=current_trip)
            if request.method == 'POST':
                get_email = request.json.get("email", None)
                if get_email is not None:
                    data_email = find_email(get_email)
                    if data_email is not None:
                        return jsonify(msg="Can not change this email"), 400
                    else:
                        current_user = get_jwt_identity()
                        edit_user_email(current_user, get_email)
                    return jsonify(msg="Email changed!"), 200
                get_phone_number = request.json.get("phoneNumber", None)
                if get_phone_number is not None:
                    data_number = find_number(get_phone_number)
                    if data_number is not None:
                        return jsonify(msg="Can not change this Phone"), 400
                    else:
                        current_user = get_jwt_identity()
                        edit_user_number(current_user, get_phone_number)
                    return jsonify(msg="Phone changed!"), 200

        @app.route("/api/v1/trips", methods=["GET"])
        @jwt_required()
        def trips():
            if request.method == 'GET':
                current_user = get_jwt_identity()
                trips_list = find_user_trips(current_user)
                pers_bike, pers_hoverboard, pers_scooter =
find_user_trips_data(current_user)
                if pers_bike and pers_hoverboard and pers_scooter is not None:
                    vehicle_pecentage = {"bike": pers_bike, "scooter": pers_scooter,
"hoverboard": pers_hoverboard}
                    total_price = 100
                    total_distance = 12312
                    favourite_district = "LUTSK CITY"
                    trips_list = [vehicle_location_to_dict_in_trips(trip) for trip in
trips_list]
                    return jsonify(totalPrice=total_price, totalDistance=total_distance,
```

```
favouriteDistrict=favourite_district, vehiclePecentage=vehicle_pecentage,
trips=trips_list)
            return jsonify(trips=[])

    @app.route("/api/v1/trip-details", methods=["GET"])
    def trip_details():
        return

    @app.route("/api/v1/stats", methods=["GET"])
    def trip_stats():
        data_stats = (
            {
                "monthly":[{
                    "name" : "April",
                    "tripsNumber" : 15},
                    {
                        "name": "May",
                        "tripsNumber": 8},
                    {
                        "name": "June",
                        "tripsNumber": 6},
                    {
                        "name": "July",
                        "tripsNumber": 23},
                    {
                        "name": "August",
                        "tripsNumber": 53}
                ],
                "vehicle":[{
    "name":"bike",
    "numberOfTrips": 14},
    {"name":"scooter", "numberOfTrips": 45},
    {"name":"hoverboard",
    "numberOfTrips": 12}
                ]
            }
        )
        return jsonify(data_stats), 200

    @app.route("/api/v1/vehicle", methods=["GET"])
    @jwt_required()
```

```
def vehicle(id=None, code=None):
    vehicle_uuid = request.args.get('id', None)
    vehicle_code = request.args.get('code', None)
    if vehicle_uuid or vehicle_code is not None:
        vehicle_data = find_vehicle_by_args(vehicle_code, vehicle_uuid)
        if vehicle_data is None:
            return jsonify(msg="This vehicle is not exist!"), 400
        vehicle_data = vehicle_location_to_dict(vehicle_data)
        return jsonify(vehicle_data), 200
    return jsonify(msg="Bad data!"), 400

@app.route("/api/v1/payment", methods=["GET"])
def payment():
    fake_cadrs = ([{
        "cardNumber": '3455385923650364',
        "cardType": "visa",
        "isCurrent": True
    },
        {
            "cardNumber": '4567212367534213',
            "cardType": "mastercard",
            "isCurrent": False
        }

    ])
    return jsonify(cards=fake_cadrs), 200

@app.route("/api/v1/payment-add", methods=["POST"])
def payment_add():
    return

@app.route('/api/v1/vehicle-connect', methods=['GET'])
@jwt_required()
def get_vehicle(id=None, code=None):
    if request.method == 'GET':
        vehicle_uuid = request.args.get('id', None)
        vehicle_code = request.args.get('code', None)
        if vehicle_uuid or vehicle_code is not None:
            vehicle_data = find_vehicle_by_args(vehicle_code, vehicle_uuid)
            if vehicle_data is not None:
                vehicle_uuid = vehicle_data['uuid']
```

```
                    start_location = vehicle_data['current_position']
                    user_uuid = get_jwt_identity()
                    user_data = get_user(user_uuid)
                    user_in_ride = user_data['current_Vehicle']
                    if user_in_ride is None:
                        is_active = vehicle_data['is_active']
                        if is_active:
                            return jsonify(msg="This vehicle is already active!"), 400
                        activate_vehicle(vehicle_uuid, user_uuid)
                        now = datetime.now()
                        reg_time = datetime.timestamp(now)
                        trip_key = str(vehicle_data) + "reg" + str(reg_time)
                        vehicle = vehicle_data['Vehicle_type']
                        create_trip_uuid = uuid.uuid5(uuid.NAMESPACE_URL, trip_key)
                        trip_uuid = str(create_trip_uuid)
                        time = int(reg_time)
                        create_trip(user_uuid, vehicle_uuid, trip_uuid, vehicle, time,
start_location)
                        vehicle_data = vehicle_location_to_dict(vehicle_data)
                        return jsonify(vehicle_uuid=vehicle_uuid, user_uuid=user_uuid,
vehicle_data=vehicle_data), 200
                    return jsonify(msg="You have active vehicle!"), 400
                return jsonify(msg="This vehicle is not exist!"), 400
            return jsonify(msg="Bad data!"), 400
        return jsonify(msg="Bad request!"), 400

    @app.route('/api/v1/vehicles', methods=['GET'])
    @jwt_required()
    def vehicles():
        vehicle_list = get_vehicles()
        if vehicle_list is not None:
            result_list = []
            for vehicle in vehicle_list:
                result_list.append(vehicle_location_to_dict(vehicle))
            return jsonify(vehicles=result_list), 200
        return jsonify("There are no available vehicles!"), 400

    @app.route('/api/v1/end-trip', methods=['POST'])
    @jwt_required()
    def clear_vehicle_data():
        end_trip_json = SafeJson(request.get_json())
```

```
        if end_trip_json.my_dict is not None:
            message = end_trip_json.init_keys(['lat', float], ['lng', float])
            if not message:
                user_uuid = get_jwt_identity()
                user_data = get_user(user_uuid)
                current_vehicle = user_data['current_Vehicle']
                current_trip = user_data['current_trip']
                now = datetime.now()
                end_time = datetime.timestamp(now)
                if current_vehicle is None:
                    return jsonify(msg="You Have no active vehicle!"), 400
                else:
                    lat, lng = end_trip_json.get_values()
                    end_location = json.dumps(dict({'lat': lat, 'lng': lng}))
                    trip_info = find_user_current_trip(current_trip)
                    start_time = trip_info['start_data']
                    price_gen, duration = set_trip_price(start_time, end_time)
                    time = int(end_time)
                    end_trip(current_vehicle, user_uuid, time, current_trip,
end_location, price_gen)
                    return jsonify(msg="Trip ended!", price=price_gen,
duration=duration), 200
            return jsonify(msg="No location data!"), 400
        return jsonify(msg="No location data!"), 400
```

ДОДАТОК Б

Лістинг коду, використаного в frontend частині додатку.

App.jsx (компонента «App» надає можливість переключатись між різними сторінками додатку не перезавантажуючи сторінку.)

```
import React from 'react';
import { Switch, Route, Redirect } from "react-router-dom";
import { ROUTES } from "./config";
import GuestPage from "./guest-page";
import RentPage from './rent-page';
import SignIn from './sign-in-page';
import SignUp from './sign-up-page';
import ProfilePage from './profile-page';
import HistoryPage from './history-page';
import MapPage from './map-page';
import '../styles/index.scss'

const { root, guest, signIn, rent, history, tourist, map, signUp, profile } =
ROUTES;

const App = () => {
  return (
    <Switch>
      <Route exact path={root} component={() => {
        return <Redirect to={guest} />
      }} />
      <Route exact path={signIn} component={SignIn} />
      <Route exact path={signUp} component={SignUp} />
      <Route exact path={guest} component={GuestPage} />
      <Route exact path={rent} component={RentPage} />
      <Route exact path={profile} component={ProfilePage} />
      <Route exact path={history} component={HistoryPage} />
      <Route exact path={map} component={MapPage} />
    </Switch>
  );
};

export default App;
```

index.js

```
import React from "react";
import ReactDOM from "react-dom";
import App from "./modules/App";
import "./styles/index.scss";
import { BrowserRouter as Router } from "react-router-dom";
import { Provider } from "react-redux";
import store from "./redux/store";

ReactDOM.render(
  <Provider store={store}>
    <Router>
      <App />
    </Router>
  </Provider>,
  document.getElementById("root")
);
```

Лістинг програмного коду для сторінки «map-page»:

index.jsx (html + js для сторінки map-page)

```
import React, { useEffect, useState, memo } from 'react';
import { useDispatch, useSelector } from 'react-redux';
import { GoogleMap, useJsApiLoader, Marker, MarkerClusterer, DirectionsService,
DirectionsRenderer, StandaloneSearchBox } from '@react-google-maps/api';
import Header from '../../components/header';
import Footer from '../../components/footer';
import MapPageInfo from './subcomponents/MapPageInfo';
import MarkerModal from './subcomponents/MarkerModal';
import MapStartTrip from './subcomponents/MapStartTrip';
import scooterMarker from '../../assets/images/scooterMarker.svg';
import bikeMarker from '../../assets/images/bikeMarker.svg';
import hoverboardMarker from '../../assets/images/hoverboardMarker.svg';
import userMarker from '../../assets/images/userMarker.svg';
import { receiveVehicles } from './actions/mapActions';


import './index.scss';

const MapPage = () => {
```

```
const dispatch = useDispatch();

useEffect(() => {
  dispatch(
    receiveVehicles(),
  );
}, []);

const [lat, setLat] = useState();
const [lng, setLng] = useState();
const [direction, setDirection] = useState({
  destination: null
});
const [response, setResponse] = useState(null);

const showModal = (vehicleId) => {
  const modal = document.getElementById(vehicleId);
  modal.style.display = "block";
};

const nearest = useSelector((state) => state.map.nearest);
const vehicle = useSelector((state) => state.map.vehicle);
const vehicles = useSelector((state) => state.map.vehicles);
const district = useSelector((state) => state.map.district);
const allVehicle = useSelector((state) => state.map.allVehicle);
const currentVehicle = useSelector((state) => state.trip.vehicle);
const vehicleId = useSelector((state) => state.trip.vehicle_id);

const vehicleMarker = (vehicle) => {
  if (vehicle === 'bike') {
    return bikeMarker;
  } else if (vehicle === 'hoverboard') {
    return hoverboardMarker;
  } else {
    return scooterMarker;
  }
};

const containerStyle = {
  width: '100%',
  height: '100%',
```

```
  position: 'relative'
};

const { isLoaded } = useJsApiLoader({
  id: 'google-map-script',
  googleMapsApiKey: "AIzaSyCOpbx_dezg9g6_LWAiLW0RmU5fDqHtlyw",
  libraries: ["places"]
});

const [map, setMap] = React.useState(null);

const onLoad = React.useCallback(function callback(map) {
  navigator.geolocation.getCurrentPosition(function (position) {
    setLat(position.coords.latitude);
    setLng(position.coords.longitude);
  });
}, []);

const onUnmount = React.useCallback(function callback(map) {
  setMap(null)
}, []);

function createKey(location) {
  return location.lat + location.lng
};

function directionsCallback(response) {
  console.log(response)

  if (response !== null) {
    if (response.status === 'OK') {
      setResponse(response)
    } else {
      console.log('response: ', response)
    }
  }
}

const mapDesign = [
  {
    "featureType": "administrative",
```

```
  "elementType": "labels",
  "stylers": [
    {
      "color": "#FFFFFF"
    },
    {
      "visibility": "simplified"
    }
  ]
},
{
  "featureType": "landscape.man_made",
  "elementType": "all",
  "stylers": [
    {
      "visibility": "simplified"
    },
    {
      "color": "#303030"
    }
  ]
},
{
  "featureType": "landscape.natural",
  "elementType": "geometry",
  "stylers": [
    {
      "color": "#000000"
    },
    {
      "visibility": "simplified"
    }
  ]
},
{
  "featureType": "poi",
  "stylers": [
    {
      "visibility": "off"
    }
  ]
```

```
},
{
  "featureType": "road",
  "elementType": "geometry",
  "stylers": [
    {
      "visibility": "simplified"
    },
    {
      "color": "#808080"
    }
  ]
},
{
  "featureType": "road",
  "elementType": "labels.text",
  "stylers": [
    {
      "color": "#FFFFFF"
    },
    {
      "visibility": "simplified"
    }
  ]
},
{
  "featureType": "road",
  "elementType": "labels.icon",
  "stylers": [
    {
      "visibility": "off"
    }
  ]
},
{
  "featureType": "water",
  "elementType": "all",
  "stylers": [
    {
      "color": "#303030"
    }
```

```
        ]
      }
    ]

    return isLoaded ? (
      <div className='map-page'>
        <Header />
        <div className='map-page-content'>
          <div className='map-page-google-map'>
            <GoogleMap
              mapContainerStyle={containerStyle}
              center={{
                lat: lat,
                lng: lng
              }}
              zoom={15}
              onLoad={onLoad}
              onUnmount={onUnmount}
              options={{
                styles: mapDesign,
              }}
            >
              <MarkerClusterer>
                {(clusterer) =>
                  vehicles.map((vehicle, index) => (
                    <Marker
                      onClick={() => showModal(vehicle.vehicle_code)}
                      title={`${vehicle.vehicle_charge}%`}
                      icon={vehicleMarker(vehicle.Vehicle_type)}
                      key={index}
                      position={vehicle.current_position}
                      clusterer={clusterer}
                    >
                      <div id={vehicle.vehicle_code} className='handle-marker-
modal'>
                        <MarkerModal
                          setDirection={setDirection}
                          position={vehicle.current_position}
                          vehicleType={vehicle.Vehicle_type}
                          vehicleId={vehicle.vehicle_code}
                          energy={vehicle.vehicle_charge}
```

```
                    />
                  </div>
                </Marker>
              ))
            }
          </MarkerClusterer>
          <Marker
            title='You'
            icon={userMarker}
            position={{
              lat: lat,
              lng: lng
            }}
          />
          {
            (
              direction.destination !== '' &&
              direction.origin !== ''
            ) && (
              <DirectionsService
                options={{
                  destination: direction.destination,
                  origin: {
                    lat: lat,
                    lng: lng
                  },
                  travelMode: 'WALKING'
                }}
                callback={directionsCallback}
                onLoad={directionsService => {
                  console.log('DirectionsService onLoad directionsService: ',
directionsService)
                }}
                onUnmount={directionsService => {
                  console.log('DirectionsService onUnmount directionsService: ',
directionsService)
                }}
              />
            )
          }
```

```
                {
                  response !== null && (
                    <DirectionsRenderer
                      options={{
                        directions: response
                      }}
                      onLoad={directionsRenderer => {
                        console.log('DirectionsRenderer onLoad directionsRenderer: ',
directionsRenderer)
                      }}
                      onUnmount={directionsRenderer => {
                        console.log('DirectionsRenderer onUnmount directionsRenderer:
', directionsRenderer)
                      }}
                    />
                  )
                }
              </GoogleMap>
            </div>
            {currentVehicle
              ? <MapStartTrip
                vehicle={currentVehicle}
                position={{
                  lat: lat,
                  lng: lng
                }}
                vehicleId={vehicleId}
              />
              : <MapPageInfo
                nearest={nearest}
                vehicle={vehicle}
                district={district}
                allVehicle={allVehicle}
              />}
          </div>
          <Footer />
        </div >
      ) : <></>
    }

    export default memo(MapPage);
```

index.scss (опис стилів для цієї сторінки)

```
@import "../../styles/index.scss";
.map-page {
  background: linear-gradient(
      180deg,
      rgba(0, 0, 0, 0.8) 0%,
      rgba(0, 0, 0, 0.8) 100%
    ),
    url("../../assets/images/history-background.png") fixed;
  background-repeat: round;
  &-content {
    display: flex;
    flex-direction: column;
    @include breakpoint(mobile_landscape) {
      display: flex;
      flex-direction: row;
    }
  }
  .finish-trip {
    padding: getRem(15) getRem(45);
    font-weight: bold;
    font-size: getRem(24);
    @include breakpoint(mobile_landscape) {
      font-size: getRem(18);
      padding: getRem(6) getRem(33);
    }
    line-height: getRem(23);
    color: #1b1b1b;
    border: none;
    background: #ffc107;
    border-radius: getRem(4);
    cursor: pointer;
    &:focus {
      outline: none;
      background-color: rgba(255, 193, 7, 0.6);
    }
  }
  .vehicle-type-in-start-trip {
    font-size: getRem(25);
```

```
  text-transform: uppercase;
  color: #ffc107;
  letter-spacing: getRem(1);
}

.handle-marker-modal {
  display: none;
  width: 100%;
  height: 100%;
  margin-top: getRem(-100);
  @include breakpoint(mobile_landscape) {
    margin-left: getRem(150);
    width: fit-content;
    height: fit-content;
  }
  box-shadow: getRem(4) getRem(4) getRem(15) getRem(10) rgba(34, 33, 33, 0.75);
  border-radius: getRem(25);
  background: #1d1d1d;
  position: fixed;
  z-index: 10;

  .marker-modal {
    display: flex;
    flex-direction: column;
    align-items: center;
    text-align: center;
    padding: getRem(30) getRem(50);

    &-close {
      cursor: pointer;
      width: getRem(50);
      position: relative;
      margin-left: getRem(400);
      margin-bottom: getRem(-30);
      z-index: 10;
      @include breakpoint(mobile_landscape) {
        width: getRem(30);
        margin-left: getRem(300);
      }
    }
```

```
  &-actions {
    margin-top: getRem(30);
    display: flex;
    justify-content: space-around;
    & button {
      padding: getRem(15) getRem(45);
      font-weight: bold;
      font-size: getRem(24);
      @include breakpoint(mobile_landscape) {
        font-size: getRem(18);
        padding: getRem(6) getRem(33);
      }
      line-height: getRem(23);
      color: #1b1b1b;
      border: none;
      background: #ffc107;
      border-radius: getRem(4);
      cursor: pointer;
      &:focus {
        outline: none;
        background-color: rgba(255, 193, 7, 0.6);
      }
    }
    &-reserve {
      margin-right: getRem(30);
    }
  }
}
.marker-modal-info {
  & h3 {
    font-size: getRem(38);
    @include breakpoint(mobile_landscape) {
      font-size: getRem(30);
    }
    font-weight: bold;
    color: white;
  }
  & p {
    font-size: getRem(30);
    @include breakpoint(mobile_landscape) {
      font-size: getRem(24);
```

```
      }
      color: #ffc107;
    }
  }
  .marker-modal-info-image {
    width: getRem(200);
    height: getRem(200);
    @include breakpoint(mobile_landscape) {
      width: getRem(123);
      height: getRem(123);
    }
    border-radius: 50%;
    background-color: rgba(255, 193, 7, 0.65);
    display: flex;
    justify-content: center;
    align-items: center;
    margin-left: getRem(20);
    & img {
      height: getRem(130);
      @include breakpoint(mobile_landscape) {
        height: getRem(92);
      }
    }
  }
}
.file {
  opacity: 0;
  width: 0.1px;
  height: 0.1px;
  position: absolute;
}
.file-input label {
  display: block;
  position: relative;
  width: getRem(230);
  height: getRem(60);
  font-size: getRem(24);
  @include breakpoint(mobile_landscape) {
    width: getRem(200);
    height: getRem(50);
    font-size: getRem(18);
```

```
  }
  border-radius: getRem(25);
  background: linear-gradient(40deg, #ffee00, #8b6c03);
  box-shadow: 0 getRem(4) getRem(7) rgba(0, 0, 0, 0.4);
  display: flex;
  align-items: center;
  justify-content: center;
  color: #fff;
  font-weight: bold;
  cursor: pointer;
  transition: transform 0.2s ease-out;
}
.map-page-google-map {
  width: 100%;
  height: getRem(800);
  margin-top: getRem(68);
  @include breakpoint(mobile_landscape) {
    width: getRem(638);
    height: getRem(569);
    margin-left: getRem(159);
  }
  .map-page-google-map-info-window {
    margin-bottom: 300px;
    &-content {
      background: #fff;
      display: flex;
      width: 100px;
      height: 100px;
    }
  }
}

.map-page-info {
  margin-top: getRem(80);
  margin-left: getRem(50);
  & h2 {
    color: #fff;
    font-weight: 700;
    font-size: getRem(36);
  }
  &-percentage {
```

```
      width: getRem(330);
      margin-top: getRem(38);
      display: flex;
      justify-content: space-between;
      & h3 {
        text-transform: uppercase;
        color: rgb(255, 193, 7);
        font-size: getRem(14);
        font-weight: 500;
      }
      & p {
        margin-top: getRem(-5);
        color: #fff;
        font-size: getRem(24);
        font-weight: 700;
      }
      .vehicle-icon {
        display: flex;
        flex-direction: column;
        align-items: center;
      }
    }
    &-totals {
      margin-top: getRem(-10);
      &-item {
        margin-top: getRem(35);
        color: rgb(255, 193, 7);
        display: flex;
        text-decoration: none;
        font-size: getRem(24);
        font-weight: 500;
        & span {
          font-weight: 700;
          margin-left: getRem(15);
          color: #fff;
        }
      }
    }
  }
}
```