**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**Чорноморський національний університет**
**імені Петра Могили**
**Факультет комп'ютерних наук**
**Кафедра інтелектуальних інформаційних систем**

**ДОПУЩЕНО ДО ЗАХИСТУ**

Завідувач кафедри інтелектуальних
інформаційних систем, д-р техн. наук, проф.

_____Ю. П. Кондратенко

«____»_____2022 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

# СЕРВІС ПОШУКУ ДЛЯ АВТОМОБІЛЬНИХ ПОПУТНИКІВ

Спеціальність 122 «Комп'ютерні наук»

**122 – БКР – 401.21810305**

***Виконав студент 4-го курсу, групи 402***
***_____М. В. Бочков***
«___» червня 2022 р.

***Керівник: ст. викладач***
***_____С. Ю. Боровльова***
«___» червня 2022 р.

**Миколаїв – 2022**

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**Чорноморський національний університет ім. Петра Могили**
**Факультет комп'ютерних наук**
**Кафедра інтелектуальних інформаційних систем**

Рівень вищої освіти **бакалавр**
Спеціальність **122 «Комп'ютерні науки»**
*(шифр і назва)*
Галузь знань **12 «Інформаційні технології»**
*(шифр і назва)*

**ЗАТВЕРДЖУЮ**

Завідувач кафедри інтелектуальних інформаційних систем, д-р техн. наук, проф.
_____ Ю. П. Кондратенко
«____» _____2022 р.

# З А В Д А Н Н Я
**на виконання кваліфікаційної роботи**

Видано студенту групи 402 факультету комп'ютерних наук Бочкову Максиму Володимировичу.

1. Тема кваліфікаційної роботи «Сервіс пошуку для автомобільних попутників».

Керівник роботи Боровльова Світлана Юріївна, старший викладач.

Затв. наказом Ректора ЧНУ ім. Петра Могили від «___» _____ 2022 р. № ____

2. Строк представлення кваліфікаційної роботи студентом «___» _____ 2022 р.

3. Вхідні (початкові) дані до роботи: функціональні та нефункціональні вимоги до сервісу для пошуку автомобільних попутників.

Очікуваний результат: сервіс пошуку для автомобільних попутників.

4. Перелік питань, що підлягають розробці (зміст пояснювальної записки):

– дослідження та аналіз предметної теми;

– огляд існуючих аналогів;

– аналіз вимог до ПЗ;

– проектування, кодування (розробка сервісу для пошуку автомобільних попутників), тестування;

5. Перелік графічного матеріалу: 32 таблиці, 23 рисунків, 31 літературних джерел та 2 додатки.

6. Завдання до спеціальної частини: аналіз основних вимог пов'язаних з роботою на підприємстві.

7. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис |
|---|---|---|
| Спеціальна частина з охорони праці | Алексєєва А. О., канд. техн. наук, доцент кафедри екології Медичного інституту ЧНУ імені Петра Могили | |

Керівник роботи __ст. викладача Боровльова С. Ю._____
*(наук. ступінь, вчене звання, прізвище та ініціали)*

_____
*(підпис)*

Завдання прийнято до виконання ___Бочков М. В._____
*(прізвище та ініціали)*

_____
*(підпис)*

Дата видачі завдання «_23_» _____листопада____ 2021 р.

**КАЛЕНДАРНИЙ ПЛАН**
**виконання кваліфікаційної роботи**

Тема: «Сервіс пошуку для автомобільних попутників».

| № | Найменування роботи | Початок | Закінчення | Примітки |
|---|---|---|---|---|
| **1** | Подання заяви на затвердження теми та керівників БКР | 23.11.2021 | 23.11.2021 | виконано |
| **2** | Складання календарного плану роботи на весь період | 20.04.2022 | 21.04.2022 | виконано |
| **3** | Отримання завдання на переддипломну практику | 25.04.2022 | 25.04.2022 | виконано |
| **4** | Збір та аналіз та аналіз матеріалів до БКР | 26.04.2022 | 30.04.2022 | виконано |
| **5** | Виконання БКР: аналіз предметної сфери, аналіз аналогів, побудова моделі, розробка, тестування | 01.05.2022 | 28.05.2022 | виконано |
| **6** | Розробка звіту з переддипломної практики | 28.05.2022 | 30.05.2022 | виконано |
| **7** | Передзахист БКР на засіданні комісії кафедри | 30.05.2022 | 30.05.2022 | виконано |
| **8** | Оформлення БКР | 31.05.2022 | 16.06.2022 | виконано |
| **9** | Подання БКР рецензенту | 22.06.2022 | 22.06.2022 | |
| **10** | Подання БКР до захисту | 26.06.2022 | 26.06.2022 | |
| **11** | Захист БКР перед екзаменаційною комісією | 29.06.2022 | 29.06.2022 | |

Розробив студент __Бочков Максим Володимирович__ _____
*(прізвище та ініціали)* (*підпис*)
Керівник роботи __ст. викладач Боровльова Світлана Юрієвна_ _____
*(наук. ступінь, вчене звання, прізвище та ініціали)* *(підпис)*

«____» _____ 2022 р.

# АНОТАЦІЯ

**бакалаврської кваліфікаційної роботи студента групи 402 ЧНУ ім. Петра Могили**

**Бочкова Максима Володимировича**

**Тема: «Сервіс пошуку для автомобільних попутників»**


У даній кваліфікаційній роботі завдання полягає у тому щоб розробити сервіс, який буде надавати можливість користувачам знаходити автомобільних попутників. Завдання роботи полягає у тому, щоб розробити сервіс, який буде надавали можливість користувачам знаходити автомобільних попутників.

Об’єктом є процес пошуку автомобільних попутників.

Предметом є технології для створення сервісу для пошуку автомобільних попутників.

Метою роботи є покращення процесу пошуку автомобільних попутників та аналіз даних під час поїздок за рахунок автоматизації.

У першому розділі досліджено предметну область, проаналізовано існуючі аналоги, визначено основні вимоги до проєкту та сформовано технічне завдання. У другому розділі описано технології та інструменти розробки для програмного забезпечення. У третьому розділі описано моделювання системи з використанням UML діаграм. У четвертому розділі представлено огляд роботи системи та користувацького інтерфейсу та описано базу даних. У спеціальній частині охорони праці описані основні вимоги та небезпеки під час праці на підприємстві.

Робота містить 32 таблиці, 23 рисунків, 31 літературних джерел та 2 додатків.

Ключові слова: спільні поїздки, безпека, база даних, UML діаграми, застосунок.

**ABSTRACT**

**to the bachelor's qualification work by the student of group 402 of Petro Mohyla Black Sea National University**

**Bochkov Maksym Volodymyrovych**

**Topic: "Search service for car passengers"**

In this qualification work, the task is to develop a service that will allow users to find car passengers. The task of the work is to develop a service that will enable users to find car passengers.

The object is the process of finding car passengers.

The subject is technologies for creating a service for finding car passengers.

The aim of the work is to improve the process of searching for car passengers and data analysis during travel through automation.

In the first section the subject area is investigated, the existing analogues are analyzed, the basic requirements to the project are defined and the technical task is formed. The second section describes software development technologies and tools. The third section describes system simulation using UML charts. The fourth section provides an overview of the system and user interface and describes the database. The special part of labor protection describes the main requirements and hazards during work at the enterprise.

The work contains 32 tables, 23 figures, 31 references and 2 appendices.

Keywords: joint trips, security, database, UML charts, application.

# ЗМІСТ

# ПЕРЕЛІК СКОРОЧЕНЬ

БД – база даних

СКБД – система керування базами даних

API – application programming interface

CSS – cascading style sheets

DI – dependency injection

DOM – document object model

GPS – global position system

HTTP – HyperText transfer protocol

HTML – HyperText markup language

IDE – integrated development environment

IMAP – Internet message access protocol

IoC – inversion of control

IP – Internet protocol

JRE – java runtime environment

JSON – JavaScript object notation

JVM – java virtual machine

ORM – object-relational mapping

POP – post office protocol

SDK – software development kit

SQL – structured query language

SMTP – Simple Mail Transfer Protocol

UI – user interface

UX – user experience

# Пояснювальна записка

**до кваліфікаційної роботи**

на тему:

## «Сервіс пошуку для автомобільних попутників»

Спеціальність 122 «Комп’ютерні науки»

**122 – БКР – 402.21810305**


***Виконав студент 4-го курсу, групи 402***
***_____М. В. Бочков***
*(підпис, ініціали та прізвище)*
«___» _____ 2022 р.

***Керівник: ст. викладач***
*(наук. ступінь, вчене звання)*
***_____С. Ю. Боровльова***
*(підпис, ініціали та прізвище)*
«___» _____ 2022 р.


**Миколаїв – 2022**

# ВСТУП

За останні декілька років завдяки мобільному телефону райдшеринг, або спільні поїздки з попутниками, набули неабиякої популярності. Автоматизація процесу пошуку водіїв та потрібних маршрутів зробила цей процес простим та швидким.

Поїзди або автобуси їдуть за великим маршрутом задля того щоб везти якомога більшу кількість людей, а водії, які користуються сервісами для пошуку автомобільних попутників, вибирають найкоротшу відстань щоб доїхати до необхідного населеного пункту. Саме цією зручністю, комфортом, економією часу та грошей такі сервіси приваблюють користувачів. Водії, в свою чергу, заощаджують гроші на пальному, а інколи навіть заробляють на цьому.

Актуальні такі системи і для невеликих міст і селищ, де немає прямого автобусного або залізничного сполучення і які знаходяться на великій відстані від великих транспортних вузлів і магістралей. В наш скрутний час ця актуальність притаманна містам де залізничне сполучення було знищено під час війни.

Проте всі сервіси не гарантують абсолютної безпеки. Існує багато випадків шахрайства, грабежів та летальних кінців під час поїздок. Тому корисно буде мати застосунок, який протягом всіє поїздки збирає сигнали з телефону, аналізує їх, та в випадку правопорушення матиме можливість допомогти пасажиру.

Об’єктом є процес пошуку автомобільних попутників.

Предметом є технології для створення сервісу для пошуку автомобільних попутників.

Метою роботи є покращення процесу пошуку автомобільних попутників та аналіз даних під час поїздок за рахунок автоматизації.

Для досягнення визначеної мети необхідно вирішити такі завдання:

– проаналізувати існуючі аналоги;

– розробити технічне завдання;

– обрати основні та допоміжні засоби реалізації сервісу для пошуку автомобільних попутників;

– реалізувати відповідний сервіс пошуку для автомобільних попутників.

Для створення системи збору та аналізу інформації було використано мову програмування Java. Для створення мобільного додатку було використано Ionic Framework. В якості СКБД було використано PostgreSQL. В якості ORM використано Spring Data. Для взаємодії з клієнтською частиною використано Spring Web. Розгортання серверу, підключення до БД, створення connection pool, налаштування логування, підключення бібліотек для роботи з JSON відбулося за допомогою Spring Boot.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

## 1.1 Опис предметної сфери

Райдшерінгом (від англійського ridesharing – “розділити поїздку”) називається спосіб здійснення поїздок, коли водій та пасажир, яким по дорозі, знаходять один одного на спеціалізованому сервісі та ділять між собою транспортні витрати.

Настільки швидке зростання популярності райдшерингу можна пояснити тим, що він зручний і вигідний як для водіїв, так і для пасажирів. Перші можуть частково чи повністю компенсувати вартість палива на дорогу. Другі – здійснювати міжміські поїздки простіше (не потрібно стояти в касу, переживати про нестачу квитків), зручніше (водій часто підвозить від «двері до дверей», знайти попутника можна за 3 хвилини за допомогою мобільного телефону), вигідніше (залежно від регіону та виду громадського транспорту можна заощадити від 10 до 60% вартості квитка [1].

У кожному сервісі для пошуку автомобільних попутників впроваджена система рейтингів та оцінок. Працює вона наступним чином: після кожної поїздки водій та пасажири залишають відгук про поїздку та оцінюють один одного. Відповідно, чим вищий у користувача рейтинг і більше позитивних відгуків, тим простіше йому знаходити собі супутників. Якщо з якоїсь причини користувач не довіряєте своєму потенційному супутнику (у нього немає відгуків, він поки не завантажив фотографію), тобто можливість знайти іншого супутника вже перевіреного та схваленого іншими користувачами.

Сучасні райдшерингові сервіси допомагають користувачам знайти максимального «зручного» для себе попутника, за рахунок вказівки своїх уподобань та побажань у дорозі. Наприклад, можна дізнатися чи можна курити в

машині чи ні, чи можна провозити свійських тварин, вказати ставлення до музики під час поїздки та ступінь своєї балакучості [2].

Основними задачами сервісів для пошуку автомобільних попутників – є:

- Об'єднання людей, які шукають попутників, подорожуючи автомобілем, і таким чином заповнювати порожні місця у транспорті.
- Поділ витрат на паливо між пасажирами та власником авто.
- Зниження виділення вуглекислого газу за рахунок спільних поїздок.
- Підвищення рівню безпеки за рахунок збільшення уваги водіїв на дорозі. У дослідженні про безпеку на дорогах, яке проводилося одночасно в 11 країнах, 85% українських водіїв відповіли, що присутність попутників в автомобілі сприяє уважнішому водінню і дотриманню правил дорожнього руху. Також 65% водіїв відповіли, що під час поїздок з пасажирами частіше роблять зупинки на відпочинок, 55% дотримуються швидкісного режиму, коли їдуть з попутниками [3].
- Гарантія безпеки протягом всієї поїздки.

## 1.2 Огляд та аналіз наявних аналогів

### 1.2.1 Огляд сервісу BlaBlaCar

Одним з найбільших сервісів для пошуку автомобільних попутників є BlaBlaCar. BlaBlaCar – провідна світова спільнота мандрівників, завдяки якій понад 90 мільйонів її учасників у 22 країнах можуть подорожувати разом. Технологічні рішення дають змогу BlaBlaCar об’єднувати людей, які шукають попутників, подорожуючи автомобілем або автобусом, і таким чином заповнювати порожні місця в транспорті. BlaBlaCar робить поїздки доступнішими, зручнішими та веселішими. Щорічно транспортна мережа

BlaBlaCar знижує викиди вуглекислого газу на 1,6 мільйона тон і об’єднує 120 мільйонів людей.

BlaBlaCar дозволяє водіям самостійно встановлювати місця посадки та висадки, а також маршрути. Це робить його чудовим вибором для мандрівників, які хочуть дістатися до необхідних місць.

Комбінація різних маршрутів і варіантів може призвести до різних витрат на той самий шлях. Але можливість вибрати водія дає пасажирам більше важелів впливу на те, скільки вони витрачають, а також тип транспортного засобу, на якому вони хотіли б дістатися місця призначення. Різні параметри вибору є однією з ключових цінностей BlaBlaCar. Ви можете вибрати водія залежно від його «балакучості» та рівня музики, яка грає в машині, а також можете відфільтрувати водіїв залежно від того, чи вони курять або дозволяють домашнім тваринам їздити у своїх транспортних засобах. Все це досягається за рахунок увімкнення простих значків при пошуку. У подібних ситуаціях важливим є також попереднє спілкування з водієм, і BlaBlaCar надає необхідні для цього інструменти.

Створення облікового запису дає багато можливостей. Можна спілкуватися через інтерфейс чату в додатку, а також зв'язатися з ними по телефону. Всі ці опції безкоштовні, додатково платити за них не потрібно [4].

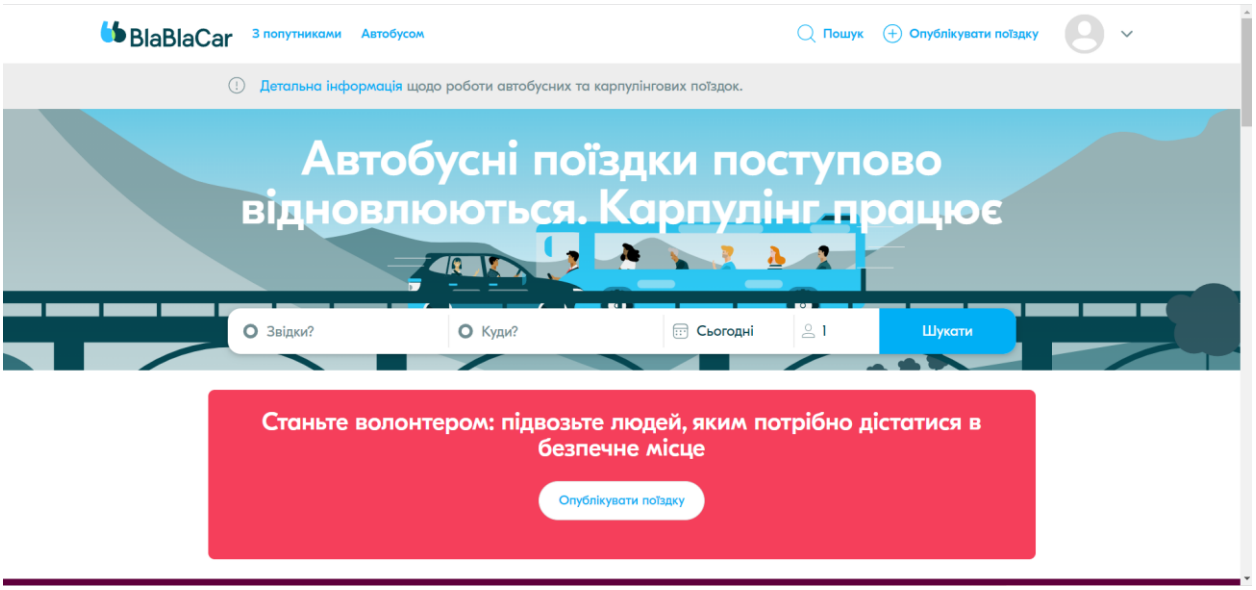


Рисунок 1.1 – Інтерфейс сервісу BlaBlaCar

До недоліків даного сервісу можна віднести:

– Відсутність гарантування безпеки для пасажирів під час поїздки. Сервіс не збирає та не аналізує дані GPS-сигналів та їх локацій.
– Шахрайство зі сторони водіїв. Через відсутність платіжної системи сервісу, багато псевдо-водіїв пропонують оплатити поїздку перед її початком на їх власні рахунки а згодом видаляють поїздки.
– Відсутність перевірки документів водіїв.

### 1.2.2 Огляд сервісу inDriver

inDriver — міжнародний сервіс пасажирських, вантажних та міжміських перевезень, послуг з кур'єрської доставки, вирішення побутових та бізнес-завдань. Компанія входить до топ-3 найбільших райдхейлінгових додатків у світі згідно з даними дослідницької агенції Sensor Tower [5].

Сервіс дає змогу робити міжміські поїздки. Для цього необхідно опублікувати свої заявки та самим призначити вартість послуг. Водії, у свою чергу, можуть шукати попутників та подорожувати дешево.

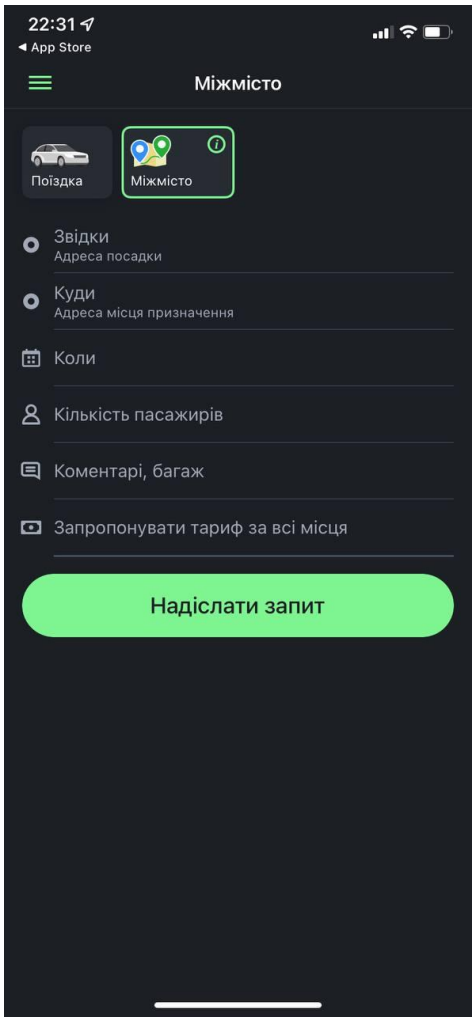


Рисунок 1.2 – Інтерфейс сервісу inDriver

До недоліків даного сервісу можна віднести:

– Дуже маленька кількість водіїв, які хочуть їхати в друге місто. Більшість водіїв використовують додаток як сервіс таксі.
– Через те, що ціну поїздки вказують пасажири, водіям не завжди вона підходить.

– Маленька кількість зайнятих місць в салоні через те, що головний ініціатор поїздки – пасажир, водій не може взяти інших пасажирів в автомобіль.

## 1.3 Постановка задачі

### 1.3.1 Основні вимоги до проєкту

Основна вимога проєкту – створення повноцінного та безпечного сервісу для пошуку автомобільних попутників. Процес пошуку зі сторони водія полягає у створенні так званих оголошень, які містять інформацію про маршрут, час поїздки та ціну за поїздку і кількість вільних місць. Процес пошуку водія для пасажира полягає в пошуку існуючих поїздок з використанням фільтрів в мобільному додатку.

Сервіс має надавати:

- можливість створювати оголошення водіям;
- можливість пошуку поїздок з використанням фільтрів для пасажирів;
- повний контроль над поїздками та аналіз локацій отриманих з мобільного телефону пасажирів;
- систему сповіщень для диспетчерів в разі небезпеки пасажирів.

Визначившись із стислим переліком вимог, які передують детальному технічному завданню, можна побачити, що завдання потребує реалізації складного та цікавого функціоналу, а також потребує високого рівня виконавця із впевненими знаннями у сфері комп’ютерних наук.

### 1.3.2 Технічне завдання

Технічне завдання повинне містити в собі чіткий опис усіх елементів системи з точки зору користувацького досвіду, технічних аспектів та наявності потрібного функціоналу.

*Назва продукту:* GoWithMe (з англійської – поїхали зі мною).

*Мета розробки:* створення сервісу для пошуку автомобільних попутників.

*Технічні вимоги:* до технічних вимог можна віднести архітектурні підходи до проєктування системи, реалізацію необхідного функціоналу, вимоги до зручності користувацького інтерфейсу UI/UX.

Архітектурний підхід, який має бути застосований при розробці – трирівнева архітектура [6]. Це архітектурний шаблон проєктування, основа створення веб-додатків. Ця архітектура складається з трьох компонентів, які прийнято називати шарами:

- Клієнтський шар – інтерфейс користувача. Це може бути веб-браузер, який надсилає HTML-сторінки. Головне, щоб за його допомогою користувач міг надсилати запити на сервер та обробляти його відповіді.
- Шар логіки – сервер, на якому відбувається обробка запитів/відповідей. Часто його ще називають серверним шаром. Також тут відбуваються всі логічні операції: математичні розрахунки, операції з даними, звернення до інших сервісів або сховищ даних.
- Шар даних – сервер баз даних: до нього звертається сервер. У цьому шарі зберігається вся необхідна інформація, якою користується програма під час роботи.

Таким чином, сервер бере на себе всі зобов'язання щодо звернення до даних, не даючи можливості користувачеві звернутися до них безпосередньо.

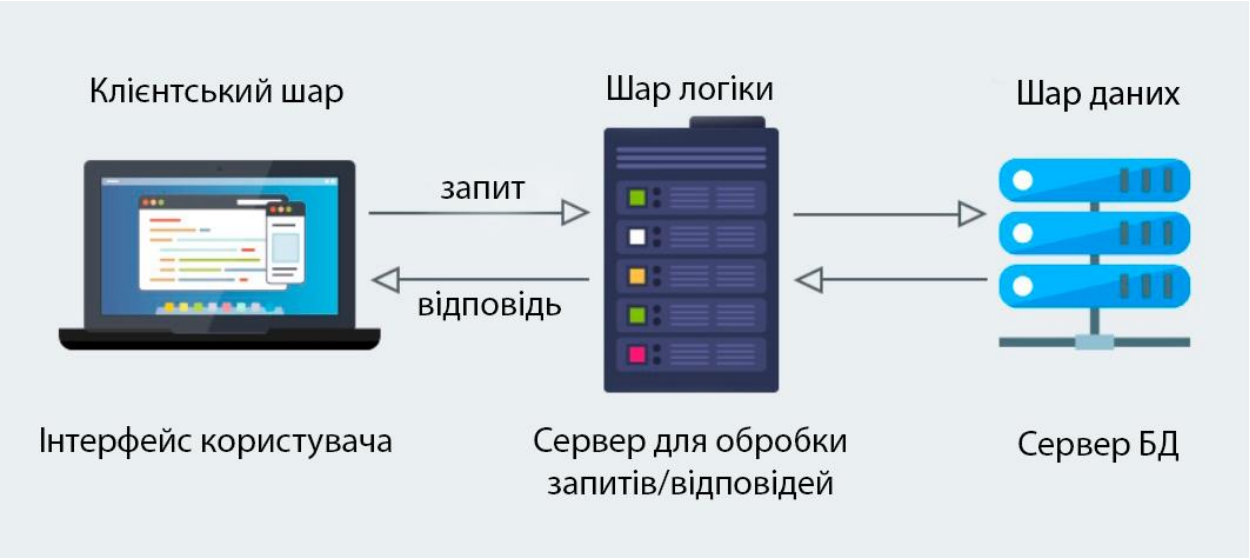


Рисунок 1.3 – Схема трирівневої архітектури

Список обов’язкового функціоналу сервісу:

1) Реєстрація користувача та можливість подальшої авторизації.
2) Можливість створювати оголошення водіям.
3) Можливість пошуку поїздок з використанням фільтрів для пасажирів.
4) Перегляд існуючих поїздок.
5) Перегляд інформації водіїв та їх автомобілів.
6) Перегляд історії поїздок.
7) Система відгуків.
8) Бронювання поїздок та прийняття або неприйняття водіями.
9) Отримання локацій з мобільного телефону та їх аналіз.
10) Тривожна кнопка, яка надає інформацію диспетчеру, посилаючи листа за допомогою Google SMTP Server.

*Вимоги до програмного забезпечення:*

Клієнтська частина даного проєкту представляє собою мобільний додаток на платформі IOS та Android. Для його розробки має використовуватись Ionic

Framework. Серверна частина має бути розроблена за допомогою мови програмування Java з використанням Spring Framework.

*Вимоги до технічного забезпечення:*

Мінімальна апаратна конфігурація комп’ютера, на якому може бути запущене середовище:

- процесор: Intel i3;
- оперативна пам’ять: 8 ГБ;
- вільного місця на SSD: 650 МБ;
- стабільне інтернет з’єднання від 20 МБ/с.

**Висновки до розділу 1**

За результатами першого розділу була детально розглянута предметна сфера. Було визначено поняття райдшерінгу та описано функціональну модель сервісу із описом можливостей користувача.

Було розглянуто найпопулярніші сервіси аналоги. Аналіз показав, що сервіси для пошуку автомобільних попутників стали досить популярними за останні кілька років. Однак ні один аналог не реалізує аналізу зібраних даних під час поїздки та не гарантує безпеки пасажирам.

В заключній частині розділу були описані основні вимоги до системи, побудовано технічне завдання, технічні вимоги до пристроїв, була визначена та описана архітектура застосунку.

# 2 ВИБІР ЗАСОБІВ РЕАЛІЗАЦІЇ ЗАСТОСУНКУ ДЛЯ ПОШУКУ АВТОМОБІЛЬНИХ ПОПУТНИКІВ

## 2.1 Мова програмування Java

Java — це об'єктно-орієнтована мова програмування, розроблена компанією Sun Microsystems (надалі придбаною компанією Oracle). Дата офіційного випуску – 23 травня 1995 року.

Програми Java транслюються в байт-код, який потім виконується віртуальною машиною Java (JVM). JVM – це програма, яка обробляє байтовий код та передає інструкції обладнання як інтерпретатор. Перевагою подібної реалізації є незалежність байт-коду від операційної системи та обладнання, що дозволяє виконувати Java-програми на будь-якому пристрої, для якого існує JVM.

Іншою важливою особливістю технології Java є гнучка система безпеки завдяки тому, що виконання програми повністю контролюється віртуальною машиною. Будь-які операції, які перевищують встановлені повноваження програми (наприклад, спроба несанкціонованого доступу до даних або з'єднання з іншим комп'ютером), викликають негайне переривання.

Часто до недоліків концепції віртуальної машини відносять те, що виконання байт-коду віртуальною машиною може знижувати продуктивність програм та алгоритмів, реалізованих мовою Java. Програми, написані на Java, мають репутацію повільніших і більше оперативної пам'яті, ніж написані мовою C. Однак, якщо порівнювати Java з мовами, що интерпретируются, які найчастіше використовуються у Web-розробці, то продуктивність Java зазвичай помітно вище [7].

У Java реалізований механізм управління пам'яттю, який називається збирачем сміття або garbage collector. Розробник створює об'єкти, а JRE за

допомогою збирача сміття очищає пам'ять, коли об'єкти перестають використовуватися.

Також Java популярна в космічній галузі. Наприклад, її активно використовують в NASA. Як зазначає інженер Шон Хенлі, Java дозволяє швидко і без значних витрат реалізовувати кросплатформені рішення. Зокрема, мовою Java написані інструменти для візуалізації даних з телескопа імені Джеймса Вебба. Набір API World Wind NASA також створений мовою Java. Він допомагає будувати інтерактивні 3D-карти земної кулі. Дані для цього система бере із супутникових знімків Landsat і SRTM [8].

Синтаксис мови Java схожий на синтаксис інших сі-подібних мов. Ось його деякі особливості:

- Чутливість до регістру - ідентифікатори User і user в Java є різними сутностями.
- Для іменування методів використовується lowerCamelCase. Якщо назва методу складається з одного слова, воно має починатися з малої літери. Приклад: firstMethodName().
- Для найменування класів використовується UpperCamelCase. Якщо назва складається з одного слова, вона повинна починатися з великої літери. Приклад: FirstClassName.
- Назва файлів програми має точно співпадати з назвою класу з урахуванням чутливості до регістру. Наприклад, якщо клас називається FirstClassName, файл має називатися FirstClassName.java.
- Ідентифікатори завжди починаються з літери (A-Z, a-z), знаку $ або нижнього підкреслення _.

Переваги Java:

- Кросплатформність. Java має спеціальну програму, яка виконує код, — віртуальну машину Java або Java Virtual Machine. Написаний програмний код один раз працює з будь-якою апаратною платформою або операційною системою: від смарткарт до додатків для розумних будинків.
- Ком'юніті. Java - досить поширена мова: ним користується велика кількість розробників, і вирішення практично будь-якої проблеми, яка може виникнути при роботі з Java вже хтось придумав. Завдяки тисячам бібліотек та форумів можна знайти готове рішення майже в будь-якій ситуації. На GitHub, наприклад, є відкриті проєкти та документація, а на форумі Stack Overflow можна звернутися за допомогою до ком'юніті.
- Надійність. Мова Java суворо типізована. Тобто будь-яка змінна або вираз має певний тип на момент компіляції, що спрощує виявлення будь-яких проблем. Компілятор сам підказує програмісту, де той припускається помилки, і не дає її зробити.
- Об'єктно-орієнтованість. Усі бібліотеки, написані будь-коли для Java, – це класи, які відповідають за функціональність мови. Будь-яка програма Java — набір класів, що описують різні об'єкти. Це добре, тому що дозволяє створювати складні програми, але прості у підтримці. І загалом Java — мультипарадигменна мова, тобто підтримує безліч принципів програмування, що дозволяє ефективно вирішувати різні завдання.
- Відносна простота. Функціональність мови оновлюється повільно, тому можна легко переходити на нові версії - наново вивчати не

доведеться. Java - строго типізована мова, а значить, у новачка завжди буде можливість побачити помилку в коді при компіляції.

– Гнучкість. На Java можна розробити програму будь-якої складності: інтернет-магазин, банківські програми, високонавантажені системи та навіть штучний інтелект.

## 2.2 Spring Framework

Spring Framework, або просто Spring - один з найпопулярніших фреймворків для створення веб-програм на Java. Фреймворк це щось схоже на бібліотеку, але є один момент. Грубо кажучи, використовуючи бібліотеку, просто створюються об'єкти класів, які в ній є, викликаються потрібні методи і таким чином виходить потрібний результат. Тобто, тут більш імперативний підхід: чітко вказується у програмі, у який конкретний момент треба створити якийсь об'єкт, у який момент викликати конкретний метод, ітд. З фреймворками справи трохи інакше. Просто пишуться якісь класи, прописуються там якась частина логіки, а створює об'єкти класів і викликає методи вже сам фреймворк. Найчастіше класи імплементують якісь інтерфейси з фреймворку або успадковують якісь класи з нього, таким чином отримуючи частину вже написаної функціональності. Але не обов'язково саме так. У Spring, наприклад, намагаються максимально відійти від такої жорсткої зв'язності (коли класи безпосередньо залежать від якихось класів/інтерфейсів з цього фреймворку), і використовують для цієї мети анотації.

Spring - це не один якийсь конкретний фреймворк. Це скоріше загальна назва для цілого ряду невеликих фреймворків, кожен з яких виконує якусь свою роботу.

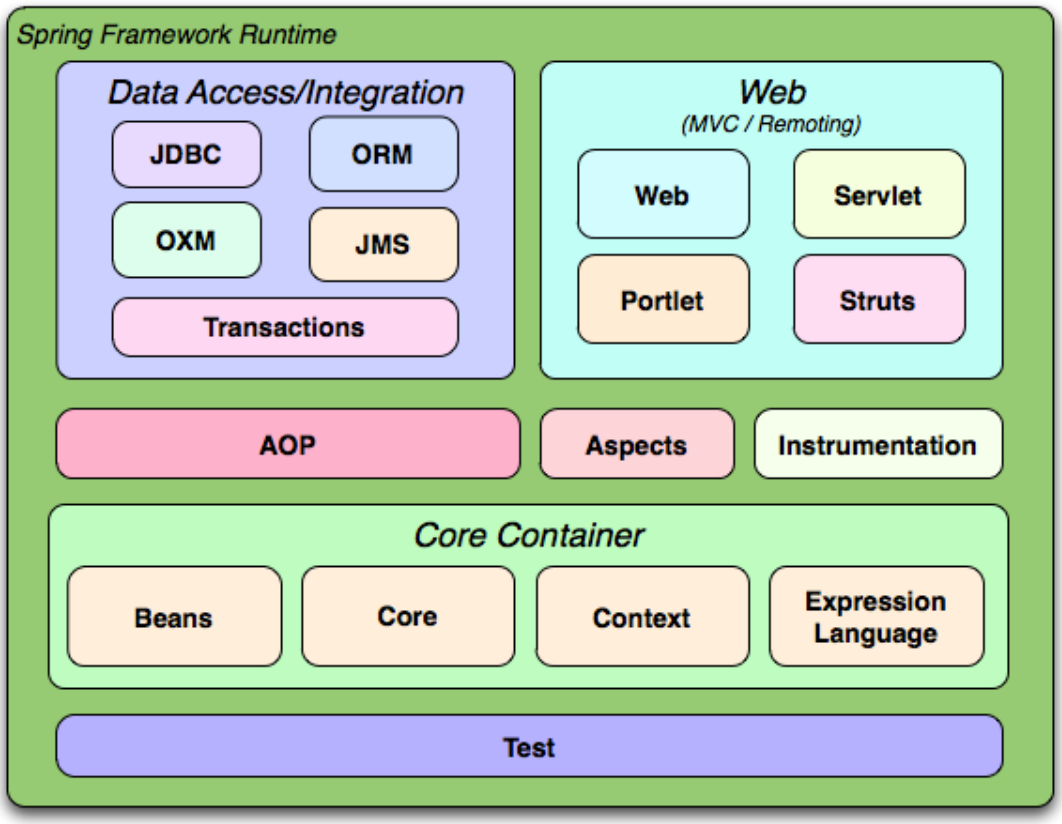


Рисунок 2.1 – Модульна структура Spring Framework

Як видно з рис 2.1, Spring модульна структура. Це дозволяє підключати тільки ті модулі, які потрібні для програми і не підключати ті, які не будуть використовуватися.

Модуль data access містить у собі засоби для роботи з даними, web - для роботи в мережі (у тому числі і для створення веб-додатків), core - для реалізації Dependency Injection.

Крім того, є ще так-звана ціла Spring-інфраструктура: безліч інших проєктів, які не входять у сам фреймворк офіційно, але при цьому безшовно інтегруються в проєкт на Spring (наприклад Spring Security для роботи з авторизацією користувачів на сайті) [9].

Незважаючи на те, що Spring містить в собі багато корисних модулів, головна його ціль – реалізація Inversion of Control та Dependency Injection.

Інверсія управління (IoC) є принципом проєктування. Як випливає з назви, він використовується для інвертування різних видів елементів керування в об’єктно-орієнтованому дизайні для досягнення слабкого зв’язку. Тут елементи керування відносяться до будь-яких додаткових обов’язків, які має клас, крім його основної відповідальності. Сюди входить контроль над ходом роботи програми, а також контроль над ходом створення об’єкта або створення та зв’язування залежного об’єкта [10].

Впровадження залежностей (DI) — це шаблон проєктування, який використовується для реалізації IoC. Він дозволяє створювати залежні об’єкти поза класом і надає ці об’єкти класу різними способами. Використовуючи DI, ми переміщуємо створення та прив’язування залежних об’єктів за межі класу, який залежить від них.

Шаблон Dependency Injection включає 3 типи класів:

- клас клієнта (залежний клас): клас, який залежить від класу сервісу;
- клас сервіс: клас, який надає обслуговування класу клієнта;
- клас інжектора: клас інжектора вводить об'єкт класу сервісу в клас клієнта.

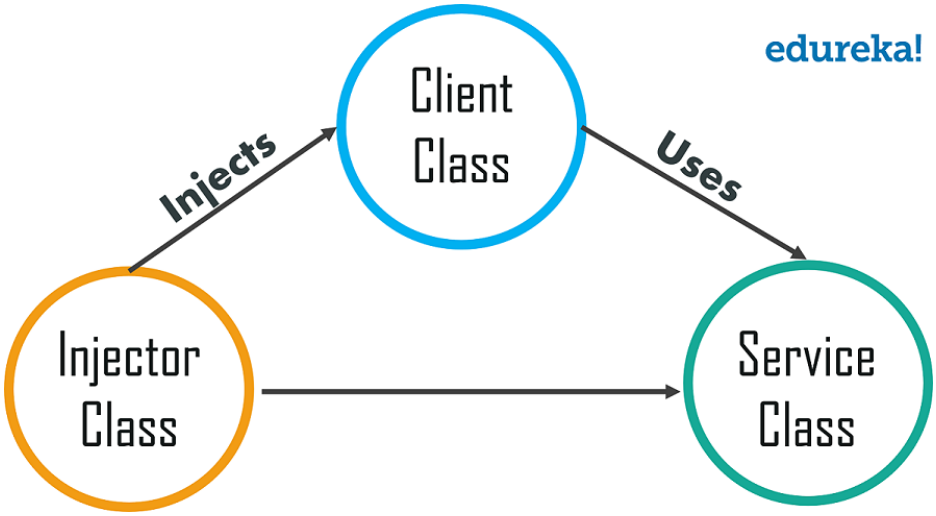


Рисунок 2.2 – Залежності класів в шаблоні Dependency Injection

Як видно з рис. 2.2, клас інжектор створює об’єкт класу сервісу та впроваджує цей об’єкт до об’єкта клієнта. Таким чином, шаблон DI відокремлює відповідальність за створення об’єкта класу сервісу від класу клієнта [11].

Існує три основні способи конфігурації Spring-програми (тобто, вказівки спрингу які саме об'єкти потрібні для роботи):

– за допомогою xml файлів/конфігів;
– за допомогою java-конфігів;
– автоматична конфігурація.

Ключовим елементом Spring є інфраструктурна підтримка на рівні програми: Spring зосереджується на «підготовці» корпоративних додатків, щоб команди могли зосередитися на бізнес-логіці на рівні програми без зайвих зв’язків із конкретними середовищами розгортання.

### 2.3 Ionic Framework

Ionic — це набір інструментів інтерфейсу користувача з відкритим вихідним кодом для створення продуктивних, високоякісних мобільних і

настільних додатків із використанням веб-технологій HTML, CSS та JavaScript з інтеграцією для популярних фреймворків, таких як Angular, React та Vue.

Ionic зосереджується на взаємодії інтерфейсу користувача та інтерфейсу програми — елементів керування інтерфейсом користувача, взаємодії, жестів, анімації. Його легко освоїти, він інтегрується з іншими бібліотеками або фреймворками. Крім того, його можна використовувати окремо без будь-якої інтерфейсної рамки за допомогою простого сценарію.

Ionic емулює рекомендації щодо користувальницького інтерфейсу нативного додатка та використовує нативні пакети SDK, об’єднуючи стандарти користувальницького інтерфейсу та функції нативних програм разом із повною потужністю та гнучкістю відкритої мережі. Ionic використовує Capacitor (або Cordova) для внутрішнього розгортання або працює у браузері як прогресивний веб-додаток.

Переваги Ionic Framework:

- Кросплатформенність. Створення та розгортання програм, які працюють на кількох платформах, від iOS та Android до десктоп та веб-додатків – і все це з однією базою коду.
- Використання веб-стандартів. Ionic побудовано на основі надійних стандартизованих веб-технологій: HTML, CSS і JavaScript, використовуючи сучасні веб-API, такі як Custom Elements і Shadow DOM. Через це компоненти Ionic мають стабільний API, і вони не є примхою одного постачальника платформи.
- Гарний дизайн. Чисто, просто та функціонально. Ionic розроблено для роботи та коректного відображення на всіх платформах.

– Простота. Ionic створено з урахуванням простоти, тому створення додатків є приємним, легким у освоєнні та доступним практично будь-кому, хто має навички веб-розробки.

Ionic має ще один великий плюс — швидкість розробки. Оскільки він заснований на Angular, проєкт на Ionic можна запускати у браузері та бачити, як виглядатиме програма під час розробки. Для того щоб побачити таке прев'ю, не обов'язково встановлювати програму на смартфон або емулятор. Це істотно допомагає заощаджувати час під час зміни UI. А під час роботи над функціями, що вимагають перевірки на смартфоні (наприклад, зробити фото), складання білда та встановлення його на смартфон займе лише кілька хвилин. На Android встановлювати програми можна прямо з командного рядка, а на iOS білд потрібно відкрити Xcode.

Для деплою в Google Play Store завершеної програми, необхідно її спочатку зібрати: ionic cordova build android --prod --release. Потім його потрібно підписати за допомогою jarsigner (інструмент для підпису APK), використовуючи JKS-ключ і zipalign (інструмент для перетворення підписаного додатка в APK на готове для завантаження на Google Play Store). При цьому важливо використовувати той же JKS-ключ, який завантажений у консоль розробника Play Store [12].

### 2.4 Google SMTP server

Simple Mail Transfer Protocol (SMTP) — простий протокол зв'язку, застосовуваний із метою пересилання електронних листів із сервера відправника на сервер одержувача. Цей протокол не розрахований на обробку вхідних повідомлень, його використовують для надсилання та подальшої доставки листів адресату. Переважно за допомогою SMTP відправляють масові та транзакційні розсилки.

Іноді протокол SMTP плутають із IMAP або POP. Але між ними є суттєва різниця. SMTP застосовують для надсилання листів, тоді як POP або IMAP використовують для обробки цих листів після отримання.

SMTP-сервер - сервер, який працює за протоколом SMTP. Його головне завдання - виступати ретранслятором (передавачем) між серверами відправника та адресата. Кожен SMTP-сервер має власну адресу у форматі smtp.serveraddress.com. Це дозволяє безпомилково визначати потрібний сервер при надсиланні пошти.

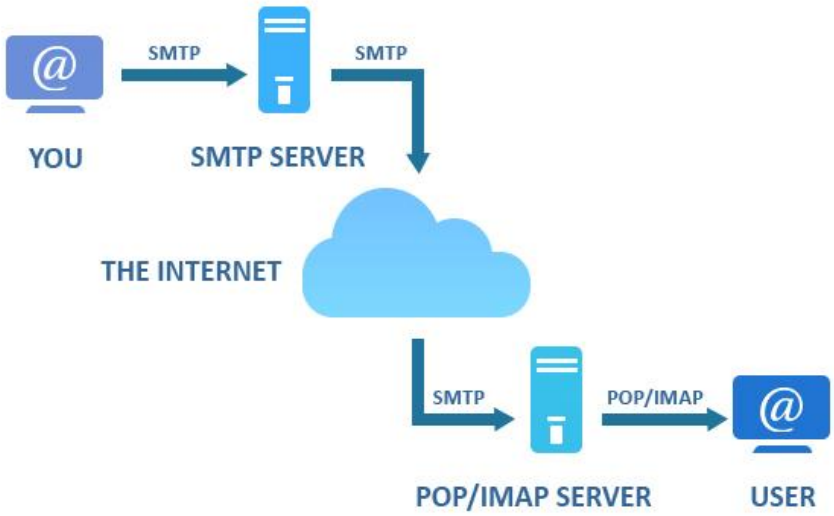


Рисунок 2.3 - Схема надсилання листа за протоколом SMTP

Протокол SMTP працює за нескладним набором правил:

- при надсиланні листа вказується відправник та одержувач;
- сервер надсилає запити обом сторонам, щоб у їх існування;
- повідомлення надсилається за вказаною адресою.

Функціонал SMTP обмежується доставкою листів. На стороні адресата електронний лист виймається за допомогою POP та IMAP.

Надсилання листів по SMTP відбувається через отримання відповідей на команди:

- MAIL FROM - адреса відправника;
- RCPT TO - адреса одержувача;
- DATA — зміст повідомлення.

Покрокове надсилання листа по SMTP виглядає так:

1) сервер відправника отримує необхідну інформацію та шукає сервер на стороні одержувача, щоб передати йому листа: по email адресата визначає поштового провайдера та запитує IP-адресу SMTP-сервера одержувача;
2) сервер виконує з'єднання порту 25 і передає лист серверу одержувача;
3) за відсутності відповіді сервера з боку одержувача, відбувається ще кілька спроб встановити з'єднання. Якщо відповіді немає, сервер відправника поверне помилку надсилання.

Для роботи з протоколом SMTP стандартно застосовують 25 портів. Але щоб уникнути спам-розсилок, провайдери можуть закрити до нього доступ. Тоді його замінюють додатковими портами:

- 465 - застосовують для створення захищеного SSL з'єднання;
- 587 - використовують для запобігання вихідному спаму за рахунок обов'язкової автентифікації відправника.

Усі SMTP-сервери умовно поділяють на два типи:

- Звичайні, призначені для надсилання особистої пошти. Такі сервери найчастіше належать провайдерам інтернет-мереж, веб-хостингів або електронної пошти. Відмінна риса - обмеження за обсягом листів. Зазвичай кожен провайдер має встановлений денний ліміт.

– Виділені сервери ретрансляції SMTP, які застосовуються для масових розсилок та надсилання транзакційних листів. Через ці сервери можна розсилати мільйони листів, не побоюючись блокування IP.

Також SMTP сервери можна класифікувати в залежності від джерела. Сервер для надсилання листів може надати:

– Інтернет-провайдери. Такий сервер не доведеться настроювати. Однак провайдери зазвичай обмежують кількість листів та швидкість відправлення. Ліміт листів, зазвичай, визначається тарифом.
– Безкоштовні поштові послуги. Більшість поштових сервісів дозволяють надсилати повідомлення по SMTP - Gmail, Yahoo. Для експлуатації сервісу знадобиться авторизація з паролем та логіном акаунта. У налаштуваннях поштового клієнта необхідно вказати порт та тип шифрування. Цим способом вдасться настроїти сервер навіть на мобільному телефоні. Проте ліміти досить малі для масових розсилок – у середньому 150-500 листів на добу.
– Хостинг-провайдери. Маючи в своєму розпорядженні сайт, можна скористатися сервером хостингу. Однак якщо IP, який може використовувати кілька сотень сайтів, буде йти спам, поштові служби заблокують адресу. Якщо потрібно отримати повну керованість, захист від блокування та виділений IP, необхідно орендувати у провайдера віртуальний SMTP-сервер (VPS).
– Транзакційні email-сервіси. Дані сервіси дозволяють надсилати листи через SMTP із сайту, із CRM-системи або іншого веб-додатка. Достатньо зареєструвати обліковий запис і вказати в потрібній системі сервіс як відправник.

Вибір відповідного сервера обумовлений цілями компанії та масштабами діяльності. При невеликій кількості листів можна обійтися безкоштовними варіантами чи функціоналом свого сайту. Для масових розсилок, коли кількість листів досягає кількох сотень, тисяч або мільйонів, варто звернути увагу на віртуальні сервери та комерційні послуги транзакційних розсилок.

Переваги SMTP для email-маркетингу

- Високий показник доставки листів. Щоб уникнути спаму, провайдери можуть блокувати надсилання пошти або обмежувати кількість листів з однієї поштової скриньки. При використанні SMTP можливість блокування знижується за рахунок обов'язкової авторизації користувача. Зазвичай для авторизації застосовують логін та пароль від облікового запису користувача у конкретному сервісі або акаунті.
- Висока безпека взаємодії. Особисті дані передплатників зберігаються на сервері відправника. SMTP-сервер лише пересилає листа, не зберігаючи дані.
- Не потрібно встановлювати додаткове програмне забезпечення. Для запуску достатньо розуміти принцип роботи протоколу та знати необхідний набір команд.
- Детальні звіти про доставку листів і помилки, що виникають. Наприклад, помилка на етапі передачі даних MAIL FROM означає, що ваша зворотна адреса не сподобалася серверу одержувача;
- Висока швидкість доставки листів. При цьому відправник точно знає, чи адресат отримав повідомлення.
- Низька вартість. Надсилання листів через SMTP обійдеться в десятки разів дешевше порівняно з відправкою повідомлень через послуги

email-розсилок. Сервіси email-розсилок надають додаткові можливості для роботи з листами: конструктори, шаблони, автоматичні ланцюжки, детальну аналітику. SMTP-сервер має лише одне завдання — доставити повідомлення адресату. Коли вам потрібно максимально оперативно доставити велику кількість листів, наприклад транзакційних, краще використовувати SMTP.

Незважаючи на всі плюси SMTP-сервера, він має свої особливості, які потрібно враховувати:

- Обмеження з боку провайдера Провайдер може заблокувати порт 25 з метою запобігання спам розсилок зі своєї мережі.
- Проблеми з доставкою через грейлістинг. У процесі використання SMTP відбувається множинний обмін запитами між серверами. Іноді сервер одержувача перестає відгукуватися, підозрюючи спам. Включається автоматичний фільтр спаму (грейлістинг) і сервер відправника припиняє спроби надсилання. Можливість грейлістингу необхідно передбачити при налаштуванні сервера. Наприклад, у сервісах розсилок настроюють повторні відправки.
- Потреба у доопрацюванні. Для надсилання розсилок через SMTP-сервер його здебільшого потрібно налаштовувати. Моніторинг відкриттів та переходів, налаштування заголовків та трек-пікселів, оперативне виправлення помилок – все це вимагатиме участі розробників [13].

## 2.5 База даних PostgreSQL

PostgreSQL – це потужна система об’єктно-реляційних баз даних з відкритим вихідним кодом, яка використовує та розширює мову SQL у поєднанні

з багатьма функціями, які безпечно зберігають і масштабують найскладніші робочі навантаження даних. Витоки PostgreSQL сягають 1986 року в рамках проєкту POSTGRES в Каліфорнійському університеті в Берклі і має понад 30 років активної розробки на базовій платформі.

PostgreSQL заслужив міцну репутацію завдяки своїй перевіреній архітектурі, надійності, цілісності даних, надійному набору функцій, розширюваності та відкритих вихідних кодів, які стоять за програмним забезпеченням, щоб постійно надавати продуктивні та інноваційні рішення. PostgreSQL працює на всіх основних операційних системах, має ACID-сумісність і має потужні доповнення, такі як популярний розширювач геопросторової бази даних PostGIS. Не дивно, що PostgreSQL став реляційною базою даних з відкритим кодом, яку вибирають багато людей та організацій.

PostgreSQL намагається відповідати стандарту SQL, якщо така відповідність не суперечить традиційним функціям або може призвести до неправильних архітектурних рішень. Багато функцій, необхідних стандартом SQL, підтримуються, хоча іноді вони мають дещо відмінний синтаксис або функції [14].

Нижче наведено список різноманітних функцій, які можна знайти в PostgreSQL:

- складні запити;
- зовнішні ключі;
- тригери;
- змінні уявлення;
- транзакційна цілісність;
- багатоверсійність.

Крім того, користувачі можуть всіляко розширювати можливості PostgreSQL, наприклад, створюючи свої:

- типи даних;
- функції;
- оператори;
- агрегатні функції;
- методи індексування;
- процедурні мови.

А завдяки вільній ліцензії, PostgreSQL дозволяється безкоштовно використовувати, змінювати та розповсюджувати всім і для будь-яких цілей – особистих, комерційних чи навчальних [15].

PostgreSQL має високу масштабованість як за величезною кількістю даних, якими він може керувати, так і за кількістю одночасних користувачів, які він може вмістити. У виробничих середовищах є активні кластери PostgreSQL, які керують багатьма терабайтами даних, і спеціалізовані системи, які керують петабайтами.

**Висновки до розділу 2**

В другому розділі було описано основні технології, що використовуються в проєкті:

- мова програмування Java;
- Spring Framework;
- Ionic Framework;
- Google SMTP Server;
- база даних PostgreSQL.

Для написання коду використовувалась потужне середовище розробки IntelliJ IDEA. IDE дає змогу використовувати та поєднувати всі перераховані технології.

# 3 МОДЕЛЮВАННЯ ТА ПРОЄКТУВАННЯ СИСТЕМИ

## 3.1 Діаграма використання

Діаграма використання описує функціональні вимоги системи з точки зору випадків використання. Це модель передбачуваної функціональності системи (випадки використання) та її середовища (актори). Приклади використання дозволяють вам співвідносити те, що вам потрібно від системи, з тим, як система задовольняє ці потреби [16].

Оскільки це дуже потужний інструмент планування, модель використання зазвичай використовується на всіх фазах циклу розробки усіма членами команди.

Діаграма випадків використання зазвичай проста. Вона не відображає деталей випадків використання:

- вона лише узагальнює деякі взаємозв'язки між випадками використання, суб'єктами та системами;
- вона не відображає порядок, у якому виконуються кроки для досягнення цілей кожного випадку використання.

Діаграми використання зазвичай розробляються на ранній стадії розробки, і люди часто застосовують моделювання випадків використання для таких цілей:

- вказання контексту системи;
- охоплення вимог системи;
- перевірка архітектури системи;
- генерація тестові кейси;
- розробка аналітиків спільно з експертами доменів.

Нижче наведено діаграму використання (рис. 3.1), що відображає основних акторів системи та їх основні функції:

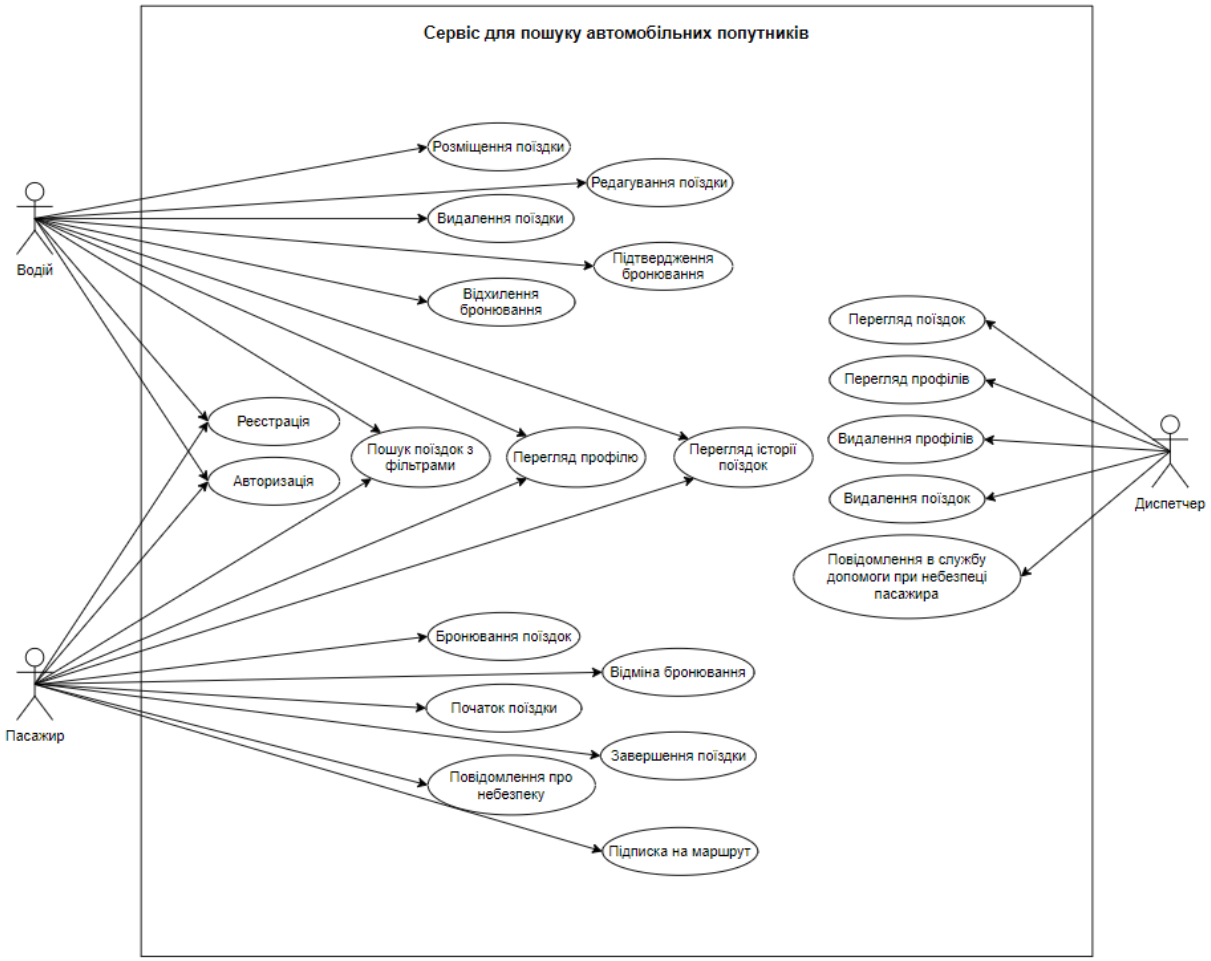


Рисунок 3.1 – Діаграма використання

### 3.2 Діаграма розгортання

Діаграма розгортання моделює фізичне розгортання артефактів на вузлах. Наприклад, для опису веб-сайту діаграма розгортання показує, які апаратні компоненти (вузли) існують (наприклад, веб-сервер, сервер додатків та сервер баз даних), на яких програмних компонентах (артефакти) працює кожен вузол (наприклад, вебзастосунок, база даних), а також те, як зв’язані різні частини [17].

Вузли відображаються у вигляді вікон, а артефакти, виділені кожному вузлу, у вигляді прямокутників у вікнах. Вузли можуть мати підвузли, які відображаються у вигляді вкладених вікон. Один вузол на схемі розгортання

може концептуально представляти кілька фізичних вузлів, таких як кластер серверів баз даних [18].

Діаграма розгортання допомагає моделювати фізичний аспект об'єктно-орієнтованої програмної системи. Це структурна схема, яка показує архітектуру системи як розгортання (розподіл) програмних артефактів для розгортання цілей. Артефакти представляють конкретні елементи у фізичному світі, які є результатом процесу розвитку. Він моделює конфігурацію часу виконання у статичному поданні та візуалізує розподіл артефактів у програмі.

Нижче наведено діаграму розгортання системи, що розробляється (рис. 3.2). На ній відображено основні компоненти системи та їх взаємозв’язки один між одним. Такий вид діаграм дозволяє раціонально організувати компоненти та відобразити їх.

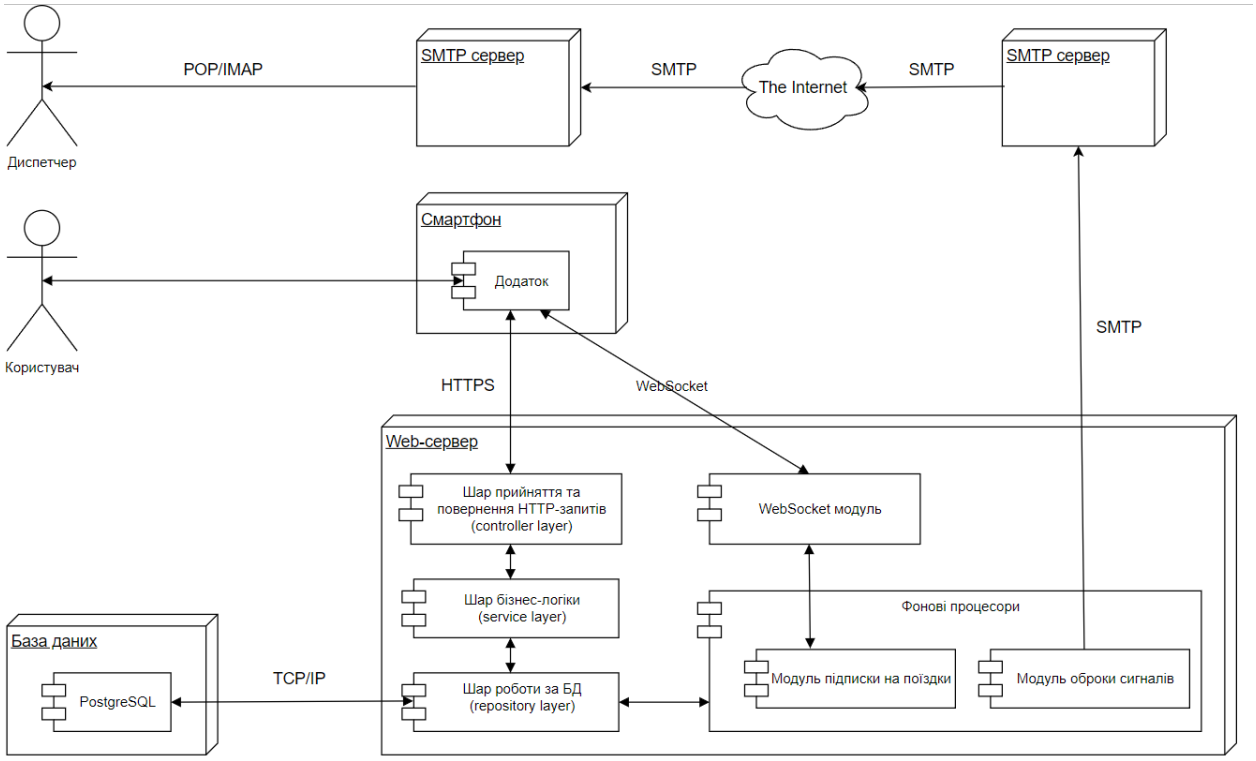


Рисунок 3.2 – Діаграма розгортання

### 3.3 Створення сценаріїв використання

**Use Case (сценарій користування)** – це перелік дій, сценарій, за яким користувач взаємодіє з додатком або програмою для виконання будь-якої дії та досягнення конкретної мети [19].

Тестування за юзкейсами проводиться для того, щоб виявити додаткові логічні прогалини та баги у web-додатку, які складно знайти під час тестування окремих індивідуальних модулів або частин цього web-додатку.

У більшості випадків Use Case описує, що робить система, а не як. Власне, цього правила і варто дотримуватися, створюючи такі сценарії.

За допомогою юзкейсів можна описувати взаємодію двох або більшої кількості учасників, що мають конкретну мету.

Розглянемо детально кожен з варіантів використання в таблицях 3.1 – 3.17.

Таблиця 3.1 – Сценарій використання UC-01

| | |
|---|---|
| Назва | Реєстрація користувача. |
| Опис | Користувач має можливість створити новий профіль в системі використавши унікальну адресу електронної пошти, пароль, вік, ім’я та прізвище. |
| Учасники | Пасажир, водій. |
| Передумови | Користувач з введеними даними не повинен існувати в системі. |
| Постумови | В системі створюється профіль з даними користувача. |
| Основний сценарій | Користувач обирає реєстрацію. Заповнює у формі електронну пошту, номер телефону, пароль, ім’я, прізвище, вік. На клієнтській стороні перевіряється чи всі поля |

Продовження таблиці 3.1

| | |
|---|---|
| | заповнені та чи введена інформація коректна. Клієнтська сторона відправляє HTTP POST запит до сервера. Сервер робить повторну валідацію введених даних, перевіряє чи немає вже існуючого користувача і зберігає його в БД. Клієнтській стороні надсилається вся інформація про користувача. |
| Розширений сценарій | Клієнтська частина перевіряє коректність електронної пошти (наявність символів «@», «.»), імені та прізвища (мають бути більше 1 літери та не містити числових значень) та електронну пошту (має мітити тільки цифри та символ «+»). Серверна частина дублює перевірку клієнтської частини а також перевіряє наявність введених пошти та паролю в системі. В разі існування даного користувача, сервер повертає помилку і клієнтська частина показує повідомлення з текстом «Такий користувач вже існує». |

Таблиця 3.2 – Сценарій використання UC-02

| | |
|---|---|
| Назва | Авторизація користувача. |
| Опис | Користувач має можливість увійти в свій профіль, використовуючи пошту та пароль. |
| Учасники | Пасажир, водій. |
| Передумови | Користувач існує в системі. |

Продовження таблиці 3.2

| | |
|---|---|
| Постумови | Користувача ідентифіковано, на клієнтську частину посилається вся інформація про нього, в БД записується дата останнього входу в профіль. |
| Основний сценарій | Користувач обирає авторизацію. Заповнює у формі електронну пошту та пароль. Клієнтська частина перевіряє наявність введених даних та їх коректність і відправляє HTTP POST запит до сервера. Сервер робить аналогічну валідацію, перевіряє наявність користувача в БД та повертає клієнтській частині успішну відповідь та всю інформацію про користувача. |
| Розширений сценарій | Якщо сервер не знайшов користувача в БД, на клієнтську частину повертається відповідь з помилкою. Клієнтська частина показує повідомлення з текстом «Такого користувача не існує». |

Таблиця 3.3 – Сценарій використання UC-03

| | |
|---|---|
| Назва | Вихід з профілю. |
| Опис | Користувач має можливість вийти з профілю. |
| Учасники | Пасажир, водій. |
| Передумови | Користувач має активну сесію. |
| Постумови | Користувач не має можливості користуватись клієнтською частиною (за виключенням реєстрації та авторизації). В БД записується дата виходу з профілю для цього користувача. |

Таблиця 3.3 – Сценарій використання UC-03

| | |
|---|---|
| Основний сценарій | Користувач нажимає на кнопку «Вийти» на головній сторінці додатку. Клієнтська частина відправляє HTTP POST запит до сервера. Сервер перевіряє чи є активна сесія для цього користувача і записує в БД дату виходу користувача з додатку. |
| Розширений сценарій | Після збереження дати виходу користувача з профілю, клієнтська частина видаляє з пам’яті всю інформацію про користувача та переходить на сторінку авторизації. |

Таблиця 3.4 – Сценарій використання UC-04

| | |
|---|---|
| Назва | Перегляд профілю користувача. |
| Опис | Користувач має можливість переглядати інформацію про себе. |
| Учасники | Пасажир, водій. |
| Передумови | Користувач авторизований в системі. |
| Постумови | Дані користувача відображаються у додатку. |
| Основний сценарій | Користувач обирає вкладку «Профіль». Клієнтська частина відображає вже існуючу в ній інформацію користувача. |
| Розширений сценарій | При натисканні на вкладку «Профіль», клієнтська частина відображає аватар, ім’я, прізвище, електрону адресу та номер телефону користувача. |

Таблиця 3.5 – Сценарій використання UC-05

| | |
|---|---|
| Назва | Пошук поїздок з фільтрами. |

Продовження таблиці 3.5

| | |
|---|---|
| Опис | Користувач має можливість шукати поїздки використовуючи фільтри. |
| Учасники | Пасажир, водій. |
| Передумови | Користувач авторизований в системі. |
| Постумови | Поїздки, дані яких відповідають заповненим фільтрам відображаються в вкладці «Пошук». |
| Основний сценарій | Користувач заповнює фільтри (місто з якого треба виїжджати, місто в яке потрібно їхати, дата, кількість місць в автомобілі), нажимає кнопку «Знайти поїздки». Клієнтська частина відправляє HTTP POST запит до сервера. Сервер дістає всі поїздки з БД, які відповідають вказаним фільтрам та повертає їх клієнтській частині. Клієнтська частина відображає їх. |
| Розширений сценарій | Сервер формує SQL-запит, в який підставляє данні з фільтрів. Після передачі поїздок до клієнтської частини, поїздки відображаються списком та кожен елемент містить в собі стислу інформацію про поїздку (година виїзду, вартість, відстань в кілометрах, аватар та ім’я водія, місто виїзду та місто прибуття). Після натискання на кнопку «Знайти поїздки» з’являється кнопка «Підписатися на маршрут». |

Таблиця 3.6 – Сценарій використання UC-06

| | |
|---|---|
| Назва | Перегляд поїздки. |
| Опис | Користувач має можливість переглянути поїздку. |
| Учасники | Пасажир. |

Продовження таблиці 3.6

| | |
|---|---|
| Передумови | Авторизованій в системі користувач переходить на поїздку з вкладки «Пошук» або «Поїздки». |
| Постумови | Клієнтська частина надає розширену інформацію про вибрану поїздку користувачу. |
| Основний сценарій | Користувач обирає поїздку. Клієнтська частина відправляє HTTP GET запит до сервера. Сервер збирає всю інформацію по даній поїздці та відправляє клієнтській частині. Клієнтська частина відкриває нове вікно з відображенням детальної інформації про поїздку. |
| Розширений сценарій | Клієнтська частина відображає таку інформацію:<br>− список проміжних пунктів (міст та селищ), які можна переглянути на карті натиснувши на кнопку «Маршрут на карті»;<br>− кнопка, при натисканні на яку, відкривається профіль водія;<br>− інформація про автомобіль;<br>− кнопки «Забронювати поїздку», «Відмінити бронювання», «Почати поїздку», «Повідомити про небезпеку», «Завершити поїздку» в залежності від стану поїздки;<br>список інших пасажирів. |

Таблиця 3.7 – Сценарій використання UC-07

| | |
|---|---|
| Назва | Перегляд профіля водія. |

Продовження таблиці 3.7

| | |
|---|---|
| Опис | Користувач має можливість переглядати профіль водія. |
| Учасники | Пасажир. |
| Передумови | Авторизований користувач знаходиться на вікні перегляду поїздки. |
| Постумови | Клієнтська частина надає розширену інформацію про водія користувачу. |
| Основний сценарій | Користувач, який знаходиться у вікні перегляду поїздки, нажимає на профіль водія. Клієнтська частина відкриває вікно перегляду інформації водія та відображає дані водія якого дістає з поїздки. |
| Розширений сценарій | Клієнтська частина відображає аватар, ім’я, прізвище, вік, рейтинг, відгуки інших пасажирів, вимоги, кількість опублікованих поїздок та дату реєстрації водія. |

Таблиця 3.8 – Сценарій використання UC-08

| | |
|---|---|
| Назва | Бронювання поїздки пасажиром. |
| Опис | Користувач має можливість бронювати поїздку. |
| Учасники | Пасажир, водій. |
| Передумови | Авторизований користувач знаходиться на вікні перегляду поїздки та натискає кнопку «Забронювати поїздку». |
| Постумови | На клієнтській частині зникає кнопка «Забронювати поїздку» та з’являється кнопка «Відмінити бронювання», в БД записується інформація про бронювання, на клієнтській |

Продовження таблиці 3.8

| | |
|---|---|
| | частині водія на поїздці з’являється форма для прийняття та відхилення бронювання пасажиру. |
| Основний сценарій | Клієнтська частина пасажиру відправляє HTTP POST запит до сервера. Сервер додає інформацію про бронювання в БД. На клієнтській частині водія з’являється форма для прийняття та відхилення бронювання пасажиру. |
| Розширений сценарій | На клієнтській частині водія на певній поїздці з’являється ім’я та аватар пасажира, кількість місць для бронювання та зелена кнопка «Прийняти бронювання» і червона кнопка «Відхилити бронювання». |

Таблиця 3.9 – Сценарій використання UC-09

| | |
|---|---|
| Назва | Відміна бронювання поїздки пасажиром |
| Опис | Користувач має можливість відмінити бронювання. |
| Учасники | Пасажир, водій. |
| Передумови | Авторизований користувач знаходиться на вікні перегляду поїздки та натискає кнопку «Відмінити бронювання». |
| Постумови | На клієнтській частині зникає кнопка «Відмінити бронювання» та з’являється кнопка «Забронювати поїздку», в БД записується інформація про відмінну бронювання, на клієнтській частини водія на поїздці зникає форма для прийняття та відхилення бронювання пасажиру. |
| Основний сценарій | Клієнтська частина пасажиру відправляє HTTP PUT запит до сервера. Сервер змінює інформацію про бронювання в |

Продовження таблиці 3.9

| | |
|---|---|
| | БД. На клієнтській частині водія зникає форма для прийняття та відхилення бронювання пасажиру. |
| Розширений сценарій | Сервер перевіряє чи існує вже бронювання для пасажиру та поїздки. Якщо немає, на клієнтську частину відправляється повідомлення про помилку. |

Таблиця 3.10 – Сценарій використання UC-10

| | |
|---|---|
| Назва | Прийняття бронювання водієм. |
| Опис | Водій має можливість прийняти бронювання пасажира. |
| Учасники | Водій, пасажир. |
| Передумови | Пасажир натиснув на кнопку «Забронювати поїздку» у вікні перегляду поїздки. |
| Постумови | Пасажир додається в список пасажирів на перегляді поїздки, з’являється кнопка «Почати поїздку». |
| Основний сценарій | Клієнтська частина відправляє HTTP PUT запит до сервера. Сервер робить зміни в БД стосовно поїздки та пасажиру та відправляє успішну відповідь до клієнтської частини водія. На клієнтській частині пасажиру у вікні перегляду поїздки з’являється кнопка «Почати поїздку» та пасажир заноситься в загальний список пасажирів. |
| Розширений сценарій | Сервер перевіряє чи існує бронювання для пасажиру на поїздку. Якщо немає, на клієнтську частину водія відправляється повідомлення про помилку. Якщо є, на клієнтську частину водія відправляється успішна відповідь, |

Продовження таблиці 3.10

| | |
|---|---|
| | пасажир заноситься до загального списку пасажирів і зникає форма прийняття та відхилення бронювання. |

Таблиця 3.11 – Сценарій використання UC-11

| | |
|---|---|
| Назва | Відхилення бронювання водієм. |
| Опис | Водій має можливість відхилити бронювання пасажира. |
| Учасники | Водій, пасажир. |
| Передумови | Пасажир натиснув на кнопку «Забронювати поїздку» у вікні перегляду поїздки. |
| Постумови | У пасажира видаляється бронювання. На клієнтській частині перегляду поїздки зникає конка «Відмінити бронювання». |
| Основний сценарій | Клієнтська частина відправляє HTTP DELETE запит до сервера. Сервер видаляє бронювання в БД та відправляє успішну відповідь до клієнтської частини водія. На клієнтській частині пасажиру у вікні перегляду поїздки зникає кнопка «Відмінити бронювання» та з’являється текст «Водій відхилив Ваш запит на бронювання». |
| Розширений сценарій | Сервер перевіряє чи існує бронювання для пасажиру на поїздку. Якщо немає, на клієнтську частину водія відправляється повідомлення про помилку. Якщо є, на клієнтську частину водія відправляється успішна відповідь, зникає форма прийняття та відхилення бронювання. |

Таблиця 3.12 – Сценарій використання UC-12

| | |
|---|---|
| Назва | Початок поїздки. |
| Опис | Пасажир має можливість почати поїздку. |
| Учасники | Пасажир. |
| Передумови | Бронювання пасажира підтвердив водій. Пасажир авторизований та знаходиться у вікні перегляду поїздки. Час натискання на кнопку «Почати поїздку» такий самий або більший ніж час виїзду поїздки, який вказаний в інформації поїздки. |
| Постумови | Зміни в БД стосовно поїздки. Початок збору та аналізу сигналів з мобільного телефону пасажиру. Зникнення кнопки «Почати поїздку» та відображення кнопок «Завершити поїздку» та «Повідомити про небезпеку» у вікні перегляду поїздки. |
| Основний сценарій | Клієнтська частина пасажира надсилає HTTP PUT запит до сервера. Сервер вносить зміни стосовно даної поїздки в БД, починає збір та аналіз сигналів з мобільного телефону пасажира та надсилає успішну відповідь клієнтській частині пасажира. Клієнтська частина пасажира ховає кнопку «Почати поїздку» та відображає кнопки «Завершити поїздку» та «Повідомити про небезпеку» у вікні перегляду поїздки. |
| Розширений сценарій | Сервер перевіряє час натискання на кнопку «Почати поїздку» на те, що він такий самий або більший ніж час виїзду поїздки, який вказаний в інформації поїздки. Якщо він менший, на клієнтську частину пасажира відправляється |

Продовження таблиці 3.12

| | |
|---|---|
| | відповідь з помилкою, на клієнтській частині пасажира з’являється повідомлення «Час поїздки ще не наступив». |

Таблиця 3.13 – Сценарій використання UC-13

| | |
|---|---|
| Назва | Повідомлення про небезпеку |
| Опис | Пасажир має можливість повідомити про небезпеку під час поїздки. |
| Учасники | Пасажир. |
| Передумови | Авторизований пасажир знаходиться в стані поїздки та натискає на кнопку «Повідомити про небезпеку». |
| Постумови | Повідомлення про небезпеку з інформацією про пасажира, водія та місцезнаходження пасажиру надсилається на пошту диспетчеру. |
| Основний сценарій | Клієнтська частина пасажира відправляє HTTP POST запит до сервера. Сервер формує лист з інформацією про пасажира, водія та місцезнаходження пасажиру і відправляє його диспетчеру на електрону пошту. |
| Розширений сценарій | Сервер перевіряє чи є пасажир в стані поїздки. Якщо ні, на клієнтську частину пасажира відправляється відповідь з помилкою, на клієнтській частині пасажира з’являється повідомлення «Ви не знаходитись в стані поїздки». |

Таблиця 3.14 – Сценарій використання UC-14

| | |
|---|---|
| Назва | Завершення поїздки. |
| Опис | Пасажир має можливість завершити поїздку. |
| Учасники | Пасажир. |
| Передумови | Авторизований пасажир знаходиться в стані поїздки та натискає на кнопку «Завершити поїздку». |
| Постумови | У вікні перегляду поїздки для пасажира зникають кнопки «Завершити поїздку» та «Повідомити про небезпеку». Поїздка потрапляє в список архівних поїздок. З’являється можливість оцінити водія та написати відгук. |
| Основний сценарій | Клієнтська частина пасажира надсилає HTTP PUT запит до сервера. Сервер вносить зміни стосовно даної поїздки в БД та надсилає успішну відповідь клієнтській частині пасажира. Клієнтська частина пасажира приховує кнопки «Завершити поїздку» та «Повідомити про небезпеку» у вікні перегляду поїздки та пропонує написати відгук про поїздку та оцінити водія. |
| Розширений сценарій | Сервер перевіряє чи є пасажир в стані поїздки. Якщо ні, на клієнтську частину пасажира відправляється відповідь з помилкою, на клієнтській частині пасажира з’являється повідомлення «Ви не знаходитись в стані поїздки». Якщо так, поїздка позначається як «архівна» та додається в список архівних поїздок в вікні перегляду власних поїздок. |

Таблиця 3.15 – Сценарій використання UC-15

| | |
|---|---|
| Назва | Написання відгуків та оцінювання поїздок. |
| Опис | Пасажир має можливість написати відгук про поїздку та оцінити водія. |
| Учасники | Пасажир, водій. |
| Передумови | Авторизований пасажир завершив поїздку. |
| Постумови | До загального списку відгуків водія додається новий відгук, а його оцінка перераховується з урахуванням нової оцінки від пасажиру. |
| Основний сценарій | Клієнтська частина пасажира надсилає HTTP POST запит до сервера. Сервер додає інформацію про новий відгук в БД та надсилає успішну відповідь клієнтській частині пасажира. |
| Розширений сценарій | Сервер перевіряє чи зв’язаний пасажир з поїздкою, до якої пасажир пише відгук. Якщо ні, на клієнтську частину пасажира відправляється відповідь з помилкою, на клієнтській частині пасажира з’являється повідомлення «Ви не можете писати відгук до даної поїздки». |

Таблиця 3.16 – Сценарій використання UC-16

| | |
|---|---|
| Назва | Перегляд історії поїздок |
| Опис | Користувач має можливість переглядати історію власних поїздок. |
| Учасники | Пасажир, водій. |
| Передумови | Користувач авторизований. |

Продовження таблиці 3.16

| | |
|---|---|
| Постумови | Дані про власні поїздки відображаються в додатку в вкладці «Поїздки». |
| Основний сценарій | Клієнтська частина надсилає HTTP GET запит до сервера. Сервер збирає інформацію та відправляє її на клієнтську частину. |
| Розширений сценарій | При переході на вкладку «Поїздки», клієнтська частина спочатку має відображати активні (ті які ще не наступили) та архівні (які вже завершились) поїздки. |

Таблиця 3.17 – Сценарій використання UC-17

| | |
|---|---|
| Назва | Підписка на маршрут |
| Опис | Користувач має можливість підписуватись на маршрут та отримувати повідомлення про нові поїздки на підписаному маршруті. |
| Учасники | Пасажир, водій. |
| Передумови | Авторизований користувач на вкладці «Пошук» має заповнити фільтри та натиснути кнопку «Підписатися на маршрут». |
| Постумови | Користувач має отримувати листи про нові поїздки на підписаному маршруті. |
| Основний сценарій | Клієнтська частина надсилає HTTP POST запит до сервера. Сервер додає інформацію про підписку в БД та надсилає успішну відповідь клієнтській частині. При появі нових |

Продовження таблиці 3.17

| | |
|---|---|
| | поїздок на маршруті, формується лист та відправляється користувачу на електрону пошту. |
| Розширений сценарій | Сервер перевіряє коректність даних в фільтрах. При помилці в даних, на клієнтську частину надсилається відповідь з помилкою. На клієнтській частині з'являється повідомлення «Неправильно вказані фільтри». |

**Висновки до розділу 3**

В третьому розділі було виконано моделювання бізнес-логіки системи з використанням UML-діаграм. Уніфікована мова моделювання (UML) – це загальноприйнята, розробницька мова моделювання в галузі програмної інженерії, яка покликана забезпечити стандартний спосіб візуалізації проєкту системи.

На початку моделювання було розроблену діаграму використання. Діаграма використання описує функціональні вимоги системи з точки зору випадків використання. Це модель передбачуваної функціональності системи (випадки використання) та її середовища (актори).

Наступним кроком було розроблено діаграму розгортання. Діаграма розгортання моделює фізичне розгортання артефактів на вузлах.

Останнім кроком було описано сценарії використання системи. Варіант використання – це перелік дій або кроків подій, що зазвичай визначають взаємодію між роллю та системою для досягнення мети.

# 4 ПРОЄКТУВАННЯ БАЗИ ДАНИХ І ДЕМОНСТРАЦІЯ РОБОТИ СИСТЕМИ

## 4.1 Проектування бази даних системи

База даних (БД) — це організована структура, призначена для зберігання, зміни й обробки взаємопов’язаної інформації, як малих так і великих обсягів. Бази даних активно використовуються для сайтів зі значними обсягами даних — часто це інтернет-магазини, портали та корпоративні сайти. Такі сайти зазвичай розроблені за допомогою серверної мови програмування або на базі CMS і не мають готових сторінок з даними за аналогією з HTML-сайтами. Сторінки динамічних сайтів формуються в результаті взаємодії скриптів і баз даних після відповідного запиту клієнта до веб-сервера [20].

В контексті баз даних варто розглянути поняття СКБД — система керування базами даних. СКБД — це комплекс програмних засобів, необхідних для створення структури нової бази, її наповнення, редагування вмісту і відображення інформації. Найбільш поширеними СКБД є MySQL, PostgreSQL, Oracle, Microsoft SQL Server.

Особливості СКБД типу клієнт-сервер:

- розташування СКБД на сервері з базами даних;
- безпосередній доступ до БД;
- централізована обробка клієнтських запитів на обробку даних;
- високий рівень надійності, доступності та безпеки.

Для зручності роботи з СКБД використовують спеціальні веб-додатки, які дозволяють за допомогою графічного інтерфейса виконувати адміністрування сервера баз даних, запускати спеціальні команди, а також працювати з контентом таблиць і баз даних — дії, які при відсутності веб-додатка виконують за

допомогою консолі. Приклади: phpMyAdmin використовують для адміністрування СКБД MySQL, pgAdmin — для PostgreSQL.

Для даної системи було обрано базу даних PostgreSQL. Було виділено 16 сутностей для зберігання а базі даних. У таблицях 4.1-4.15 наведено їх опис.

Таблиця 4.1 – Сутність Trip

| Атрибут | Тип | Опис |
|---|---|---|
| trip_id | bigint | Унікальний ідентифікатор поїздки |
| create_date | timestamp | Дата створення |
| distance_in_meters | bigint | Дистанція в метрах |
| duration_in_seconds | bigint | Тривалість в секундах |
| trip_number | varchar | Номер поїздки |
| update_date | timestamp | Дата редагування |
| driver_id | bigint | Зовнішній ключ для зв’язку із сутністю водія (сутність Driver) |
| is_main_trip | boolean | Флаг, чи являється поїздка головною |

Таблиця 4.2 – Сутність Waypoint

| Атрибут | Тип | Опис |
|---|---|---|
| waypoint_id | bigint | Унікальний ідентифікатор маршрутної точки |
| trip_id | bigint | Зовнішній ключ для зв’язку із сутністю поїздки (сутність Trip) |
| date_time | timestamp | Запланований час прибуття в маршрутну точку |

Таблиця 4.3 – Сутність Place

| Атрибут | Тип | Опис |
|---|---|---|
| place_id | bigint | Унікальний ідентифікатор місця |
| waypoint_id | bigint | Зовнішній ключ для зв’язку із сутністю маршрутної точки (сутність Waypoint) |
| longitude | varchar | Довгота |
| latitude | varchar | Широта |
| country_code | varchar | Код країни |
| city | varchar | Назва міста |
| address | varchar | Адреса |

Таблиця 4.4 – Сутність Vehicle

| Атрибут | Тип | Опис |
|---|---|---|
| vehicle_id | bigint | Унікальний ідентифікатор транспорту |
| trip_id | bigint | Зовнішній ключ для зв’язку із сутністю поїздки (сутність Trip) |
| model | varchar | Модель транспортного засобу |
| make | varchar | Марка транспортного засобу |

Таблиця 4.5 – Сутність Price

| Атрибут | Тип | Опис |
|---|---|---|
| price_id | bigint | Унікальний ідентифікатор ціни |
| trip_id | bigint | Зовнішній ключ для зв’язку із сутністю поїздки (сутність Trip) |
| currency | varchar | Назва валюти |

Продовження таблиці 4.5

| | | |
|---|---|---|
| amount | double | Кількість грошей |

Таблиця 4.6 – Сутність Driver

| **Атрибут** | **Тип** | **Опис** |
|---|---|---|
| driver_id | bigint | Унікальний ідентифікатор водія |
| user_id | bigint | Зовнішній ключ для зв’язку із сутністю користувача (сутність User profile) |
| years | integer | Кількість років |
| user_picture | varchar | Фотографія водія |
| registration_date | timestamp | Дата реєстраціїї |
| rating | real | Рейтинг водія |
| phone_number | varchar | Номер телефону |
| firstname | varchar | Ім’я |
| lastname | varchar | Прізвище |
| email | varchar | Електрона пошта |
| count_of_trips | integer | Кількість поїздок |

Таблиця 4.7 – Сутність Driver requirement

| **Атрибут** | **Тип** | **Опис** |
|---|---|---|
| driver_requirement_id | bigint | Унікальний ідентифікатор вимоги водія |
| driver_id | bigint | Зовнішній ключ для зв’язку із сутністю водія (сутність Driver) |
| requirement | requirement_type | Опис вимоги водія |

Таблиця 4.8 – Сутність Driver feedback

| Атрибут | Тип | Опис |
|---|---|---|
| driver_feedback_id | bigint | Унікальний ідентифікатор відгука водія |
| trip_id | bigint | Зовнішній ключ для зв’язку із сутністю поїздки (сутність Trip) |
| user_id | bigint | Зовнішній ключ для зв’язку із сутністю користувача (сутність User profile) |
| driver_id | bigint | Зовнішній ключ для зв’язку із сутністю водія (сутність Driver) |
| grade | real | Оцінка водія за одну поїздку |
| create_date | timestamp | Дата створення |
| comment | varchar | Коментар до відгуку |

Таблиця 4.9 – Сутність User profile

| Атрибут | Тип | Опис |
|---|---|---|
| user_id | bigint | Унікальний ідентифікатор користувача |
| password | varchar | Пароль |
| last_log_in_date | timestamp | Остання дата авторизації |
| last_log_out_date | timestamp | Остання дата виходу з профілю |
| firstname | varchar | Ім’я |
| lastname | varchar | Прізвище |
| email | varchar | Електрона пошта |
| trip_details_id | bigint | Зовнішній ключ для зв’язку із сутністю деталей поїздки (сутність Trip details) |
| user_role | user_role_type | Роль користувача |

Продовження таблиці 4.9

| | | |
|---|---|---|
| user_picture | varchar | Фотографія користувача |
| phone_number | varchar | Номер телефону |

Таблиця 4.10 – Сутність Stop over

| **Атрибут** | **Тип** | **Опис** |
|---|---|---|
| stop_over_id | bigint | Унікальний ідентифікатор зупинки |
| trip_details_id | bigint | Зовнішній ключ для зв'язку із сутністю деталей поїздки (сутність Trip details) |
| longitude | varchar | Довгота |
| latitude | varchar | Широта |
| country_code | varchar | Код країни |
| city_name | varchar | Назва міста |
| address | varchar | Адреса зупинки |

Таблиця 4.11 – Сутність Trip details

| **Атрибут** | **Тип** | **Опис** |
|---|---|---|
| trip_details_id | bigint | Унікальний ідентифікатор деталей поїздки |
| comment | varchar | Коментар водія до поїздки |
| create_date | timestamp | Дата створення |
| departure_date | timestamp | Дата прибуття |
| departure_city | varchar | Місто відправлення |
| arrival_city | varchar | Місто прибуття |
| is_archived | boolean | Флаг, чи поїздка вже завершена |
| is_passed | boolean | Флаг, чи поїздка вже почалась |

Продовження таблиці 4.11

| | | |
|---|---|---|
| is_women_trip | boolean | Флаг, чи поїздка тільки для жінок |
| main_permanent_id | bigint | Зовнішній ключ для зв’язку із сутністю головної поїздки (сутність Trip) |
| seats | integer | Кількість всіх місць в автомобілі |
| seats_left | integer | Кількість занятих місць |
| update_date | timestamp | Дата редагування |
| vehicle_picture | varchar | Фотографія транспортного засобу |
| permanent_id | bigint | Зовнішній ключ для зв’язку із сутністю поїздки (сутність Trip) |

Таблиця 4.12 – Сутність User trip

| **Атрибут** | **Тип** | **Опис** |
|---|---|---|
| user_trip_id | bigint | Унікальний ідентифікатор зв’язки поїздки і пасажиру |
| lat | varchar | Широта |
| lng | varchar | Довгота |
| trip_start_time | timestamp | Дата початку поїздки |
| trip_and_time | timestamp | Дата завершення поїздки |
| trip_id | bigint | Зовнішній ключ для зв’язку із сутністю поїздки (сутність Trip) |
| user_id | bigint | Зовнішній ключ для зв’язку із сутністю користувача (сутність User profile) |
| user_trip_status | user_trip_status_type | Статус поїздки |

Продовження таблиці 4.12

| | | |
|---|---|---|
| departure_city | varchar | Місто відправлення |
| arrival_city | varchar | Місто прибуття |
| seats | integer | Кількість зайнятих місць в автомобілі пасажиром |
| need_help | char | Флаг, чи потрібна пасажиру допомога |

Таблиця 4.13 – Сутність User trip history

| **Атрибут** | **Тип** | **Опис** |
|---|---|---|
| user_trip_history_id | bigint | Унікальний ідентифікатор історії зв’язки поїздки і пасажиру |
| lat | varchar | Широта |
| lng | varchar | Довгота |
| trip_start_time | timestamp | Дата початку поїздки |
| trip_and_time | timestamp | Дата завершення поїздки |
| trip_id | bigint | Зовнішній ключ для зв’язку із сутністю поїздки (сутність Trip) |
| user_id | bigint | Зовнішній ключ для зв’язку із сутністю користувача (сутність User profile) |
| user_trip_status | user_trip_status_type | Статус поїздки |
| departure_city | varchar | Місто відправлення |
| arrival_city | varchar | Місто прибуття |
| seats | integer | Кількість зайнятих місць в автомобілі пасажиром |

Продовження таблиці 4.13

| | | |
|---|---|---|
| need_help | char | Флаг, чи потрібна пасажиру допомога |

Таблиця 4.14 – Сутність Subscription

| Атрибут | Тип | Опис |
|---|---|---|
| subscriprion_id | bigint | Унікальний ідентифікатор підписки |
| user_id | bigint | Зовнішній ключ для зв’язку із сутністю користувача (сутність User profile) |
| notified_trips | varchar | Список нових поїздок для відправлення пасажиру |
| number_of_seats | integer | Кількість необхідних місць для пасажиру |
| create_date | timestamp | Дата створення |
| departure_date | timestamp | Дата відправлення |
| city_from | varchar | Місто відправлення |
| city_to | varchar | Місто прибуття |

Таблиця 4.15 – Сутність Subscription notification

| Атрибут | Тип | Опис |
|---|---|---|
| subscription_notification_id | bigint | Унікальний ідентифікатор повідомлення |
| notification_date | timestamp | Дата повідомлення |
| subscriprion_id | timestamp | Зовнішній ключ для зв’язку із сутністю підписки (сутність Subscription) |

Опис змісту даних для кожної сутності:

1) Trip – таблиця поїздок, які публікують водії.
2) Waypoint – таблиця маршрутних точок поїздок.
3) Place – таблиця яка містить опис про місця маршрутних точок.
4) Vehicle – таблиця транспортних засобів.
5) Price – таблиця цін за поїздки.
6) Driver – таблиця водіїв.
7) Driver requirement – таблиця з описом вимог від водіїв.
8) Driver feedback – таблиця відгуків.
9) User profile – таблиця користувачів.
10) Stop over – таблиця зупинок під час поїздок.
11) Trip details – таблиця з детальним описом поїздок.
12) User trip – таблиця яка зв’язує пасажирів та поїздки та містить інформацію про поточний стан поїздки.
13) User trip history – таблиця в якій зберігається вже не актуальна інформація про поїздки. Дублю поля сутності User trip.
14) Subscription – таблиця підписок на поїздку.
15) Subscription notification – таблиця повідомлень для поїздок, на які зробили підписку.

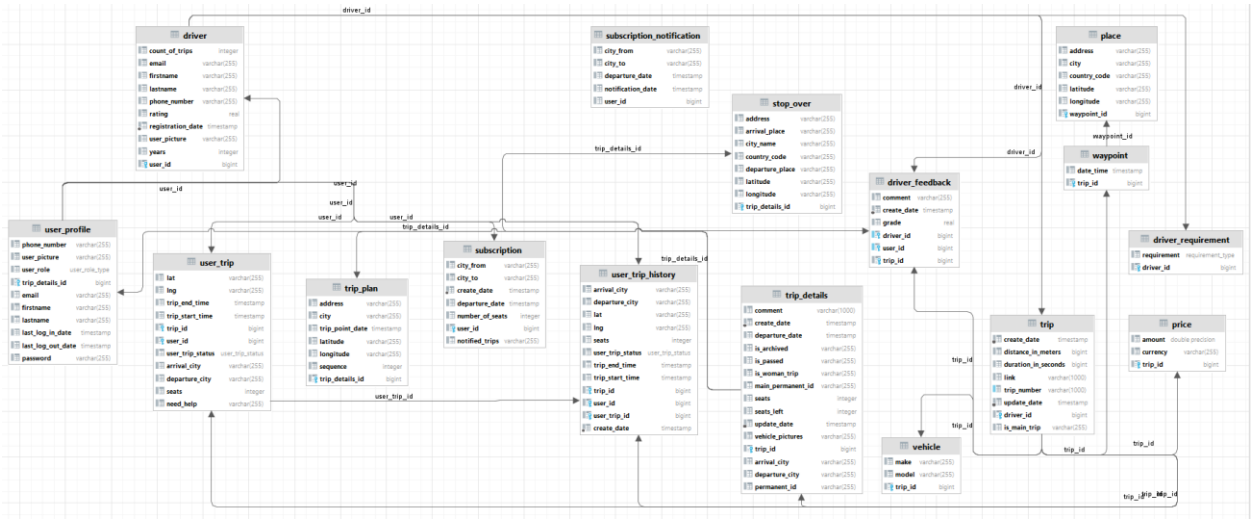


Рисунок 4.1 – Діаграма зв’язків між таблицями

## 4.2 Демонстрація роботи системи

При запуску застосунку система намагається авторизувати користувача. Якщо користувача не ідентифіковано йому надається можливість або зареєструватися, або авторизуватися до профілю. На рис. 4.2 зображено форму авторизації.

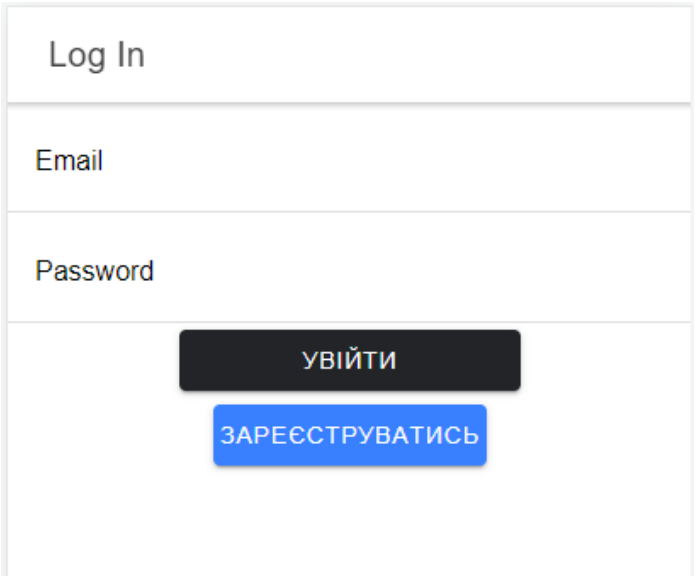


Рисунок 4.2 – Форма авторизації

Обравши опцію реєстрації користувачу надається можливість створити профіль заповнивши обов'язкові поля. На рис. 4.3 зображено форму реєстрації користувача.

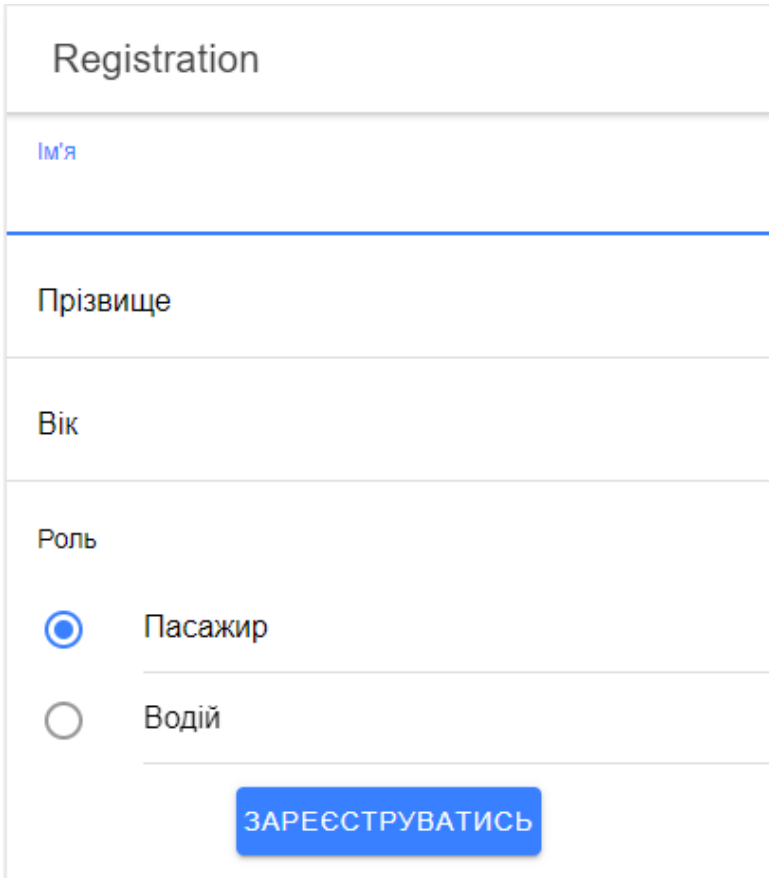


Рисунок 4.3 – Форма реєстрації користувача

Якщо на першому кроці користувач обирає авторизацію – йому надається можливість зробити це, заповнивши поля пошти та паролю.

Після успішної авторизації користувач потрапляє на сторінку пошуку поїздок за фільтрами. Заповнивши всі необхідні поля та натиснувши на кнопку «Знайти поїздки», користувачу надається список активних поїздок, які відповідають заповненим фільтрам та відкривається доступ до кнопки «Підписатися на маршрут. При натисненні на яку в системі створюється підписка та при появі нових поїздок, які будуть відповідати пасажирським фільтрам, буде

повідомлено про них користувачу через електрону пошту та повідомлення в додатку. На рис. 4.3 зображено сторінку пошуку поїздок і на рис. 4.4 зображено повідомлення про знайдену нову поїздку.

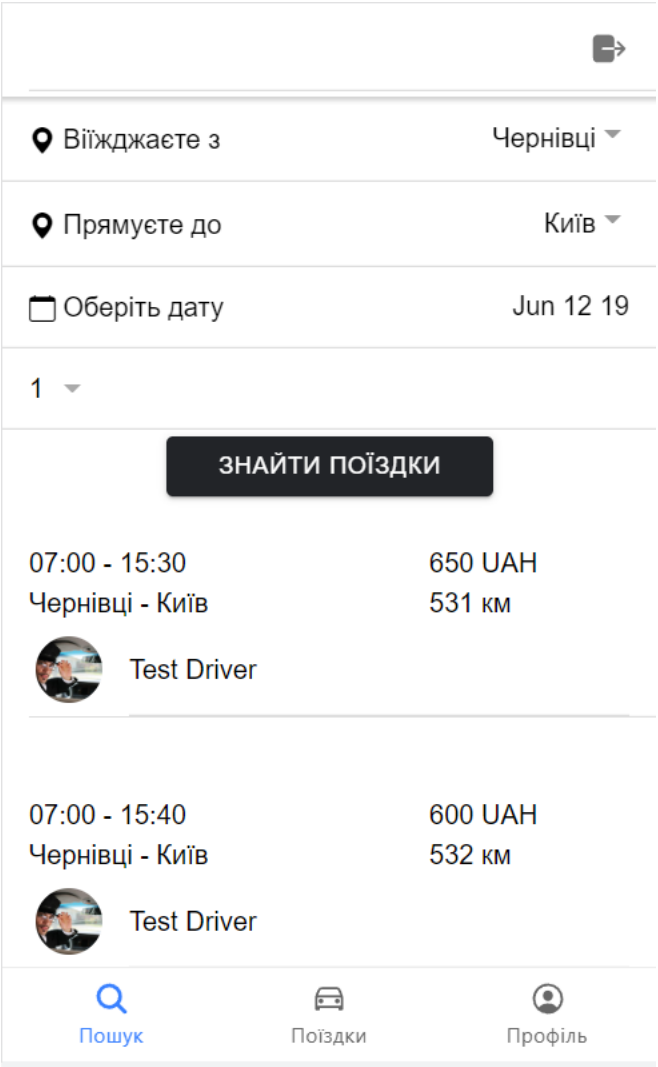


Рисунок 4.3 – Сторінка пошуку поїздок

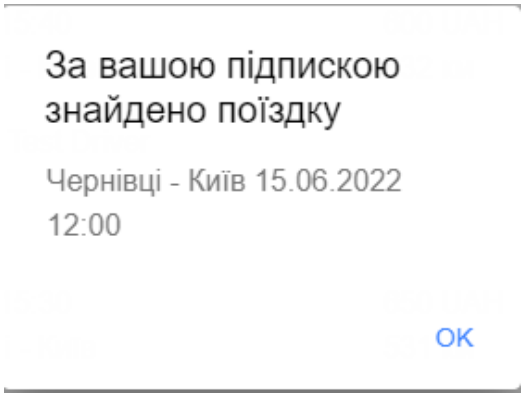


Рисунок 4.4 – Повідомлення про нову поїздку через повідомлення в додатку

При переході на поїздку, відображається список маршрутних точок та їх розташування на карті, кнопка для перегляду профілю водія, кнопка «Забронювати поїздку» (якщо користувач ще цього не зробив) і список пасажирів, які теж їдуть в автомобілі.

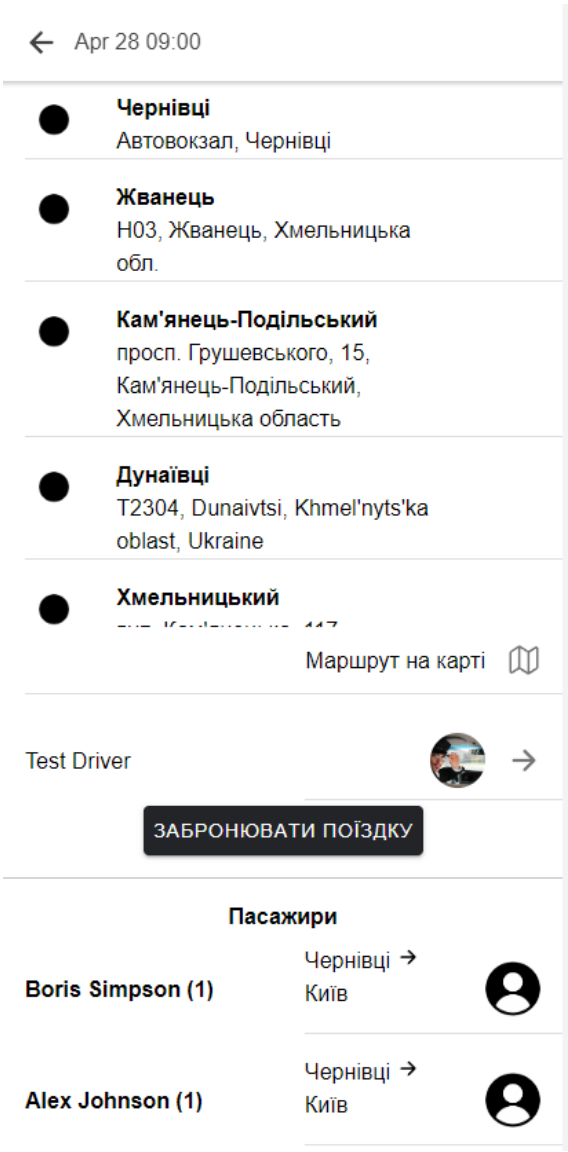


Рисунок 4.5 – Сторінка перегляду поїздки

При натисканні на кнопку «Маршрут на карті», відкривається відображення маршрутних точок на карті та шлях між ними.

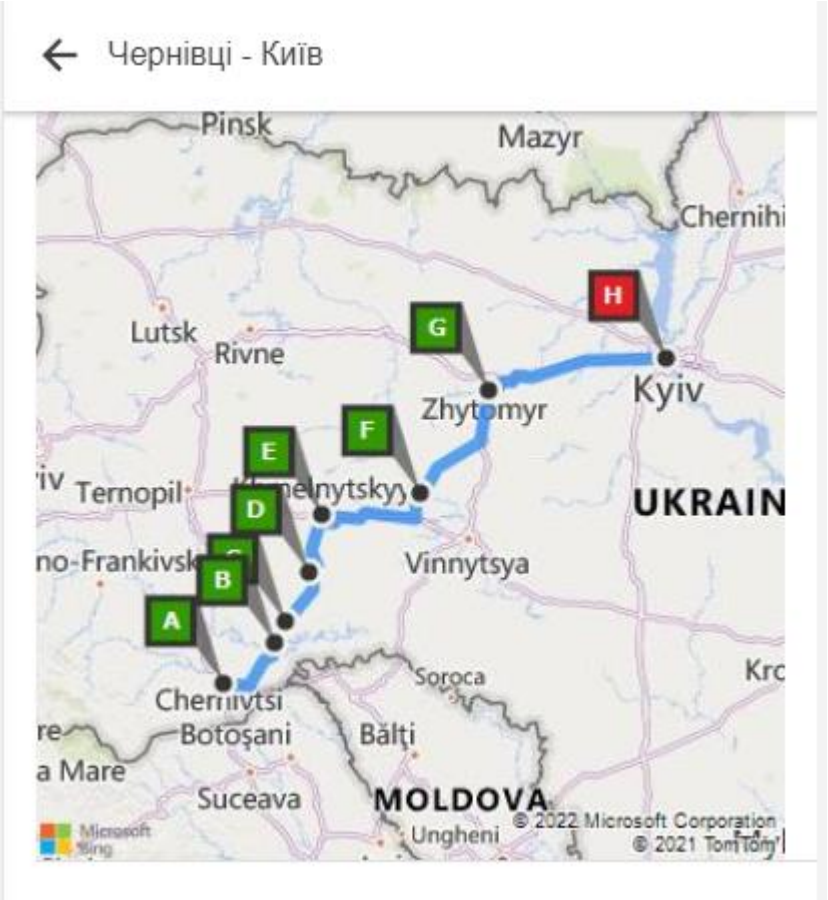


Рисунок 4.6 – Відображення маршрутних точок на карті

При натисканні на ім’я або фотографію водія, відкривається відображення профіля водія. На сторінці міститься інформація про вік водія, його оцінка та кнопка для перегляду відгуків, вимоги, кількість опублікованих поїздок та дата реєстрації на сервісі.

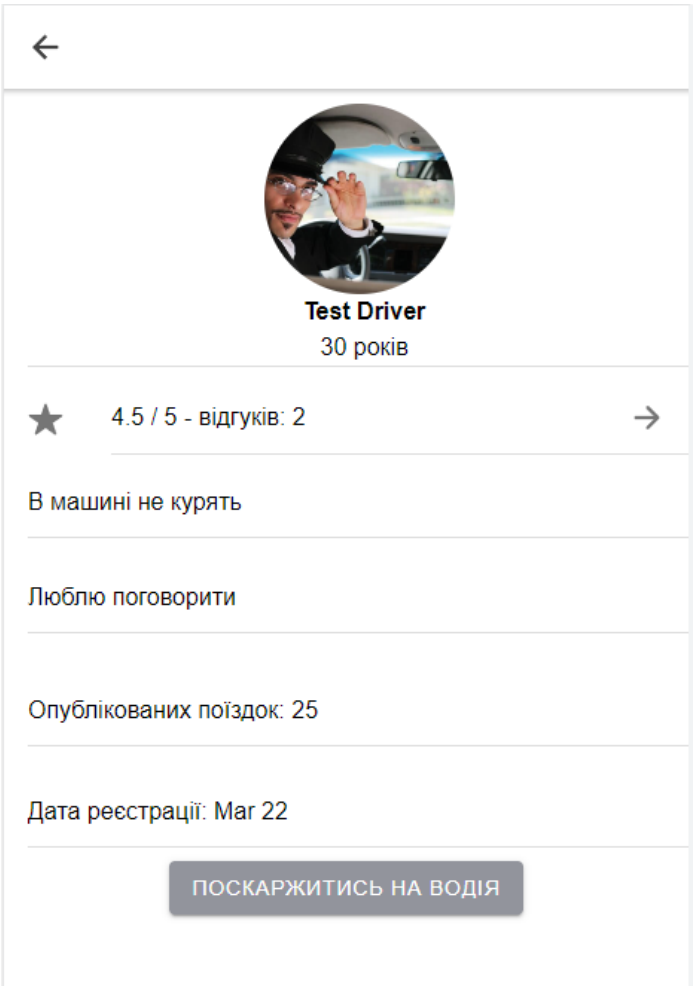


Рисунок 4.7 – Сторінка перегляду профіля водія

При натисканні на кнопку відгуків в профілі водія, відкривається сторінка перегляду відгуків яка містить загальну оцінку та перелік відгуків інших пасажирів.

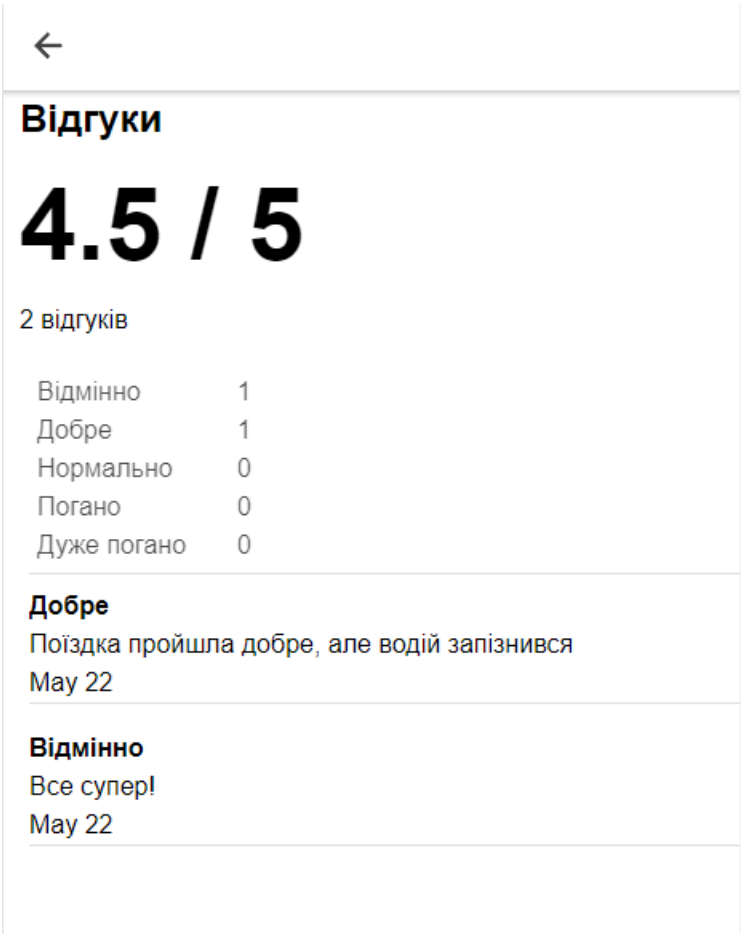


Рисунок 4.8 – Сторінка перегляду відгуків водія

При переході на сторінку «Поїздки», відображається список активних (поїздки, які ще не закінчились) та архівних (поїздки, які вже закінчились) поїздок. Також для водіїв на цій сторінці відображаються пасажири, які намагаються забронювати поїздки.

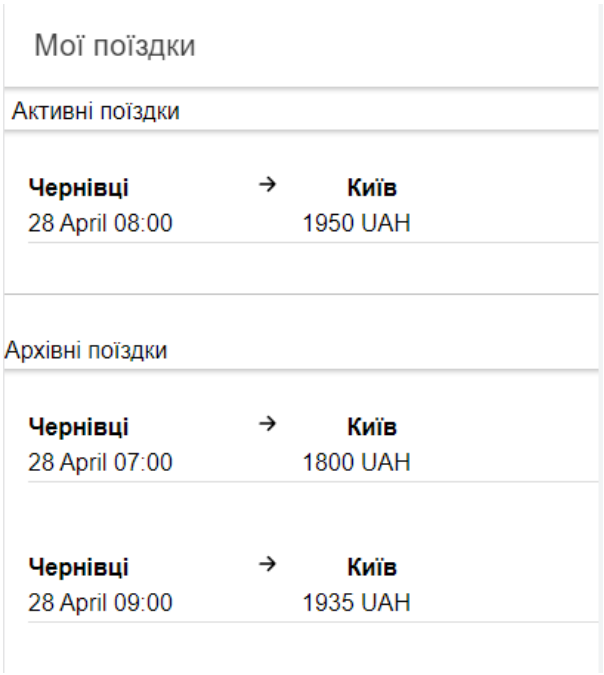


Рисунок 4.9 – Сторінка «Поїздки»

При переході на сторінку «Профіль», відображається профіль користувача.

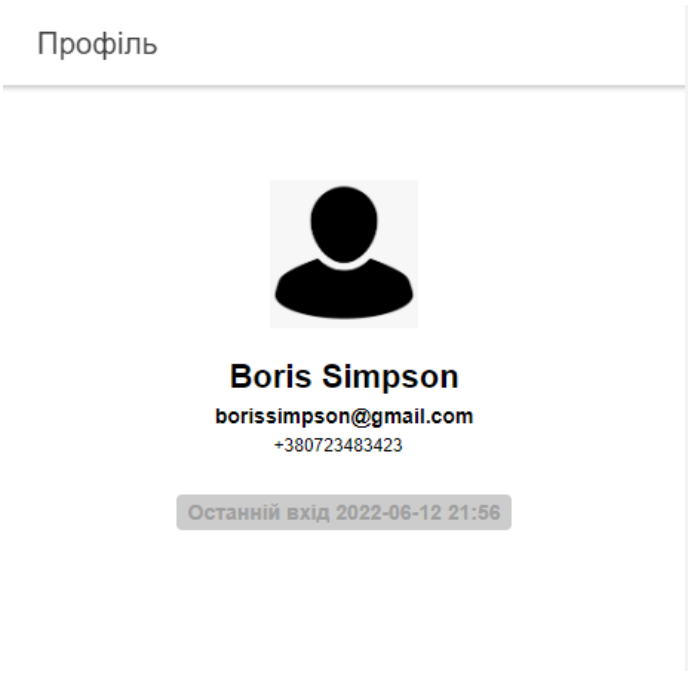


Рисунок 4.10 – Сторінка «Профіль»

При натисканні кнопки «Забронювати» на сторінці перегляду поїздки, водію надсилається запит де він може прийняти чи відхилити бронювання.

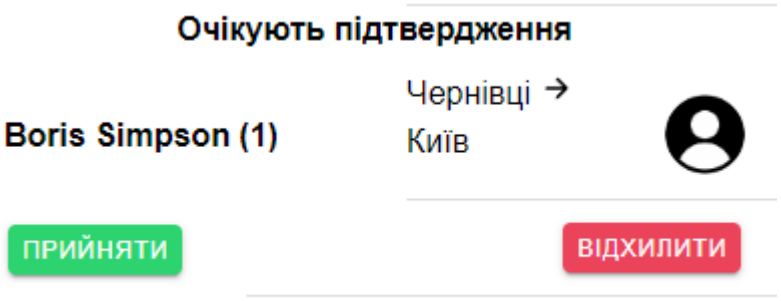


Рисунок 4.11 – Форма прийняття або відхилення бронювання

Після підтвердження бронювання водієм, є можливість почати поїздку. Для цього необхідно натиснути кнопку «Почати поїздку». Саме в цей момент система починає збирати та оброблювати локації з додатку. Також на цьому кроці з’являється індикатор того що поїздка почалась та кнопки «Завершити поїздку» і «Повідомити про небезпеку».

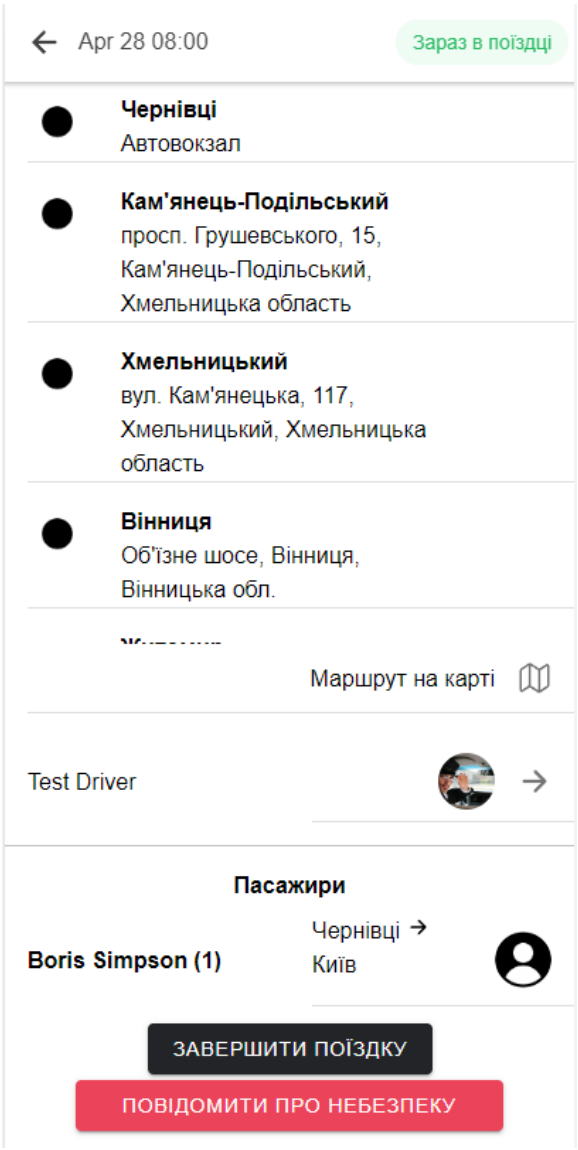


Рисунок 4.12 – Сторінка перегляду поїздки після її початку

При натисканні кнопки «Повідомити про небезпеку», диспетчеру буде надіслано листа на електрону пошту. Теж саме буде зроблено, якщо система зафіксує, що автомобіль рухається не за маршрутом. Лист складається з інформації про пасажира, водія та локації автомобіля.


Рисунок 4.13 – Приклад листа, який приходить диспетчеру при небезпеці пасажира

Після успішного завершення поїздки є можливість завершити поїздку натиснувши на кнопку «Завершити поїздку».

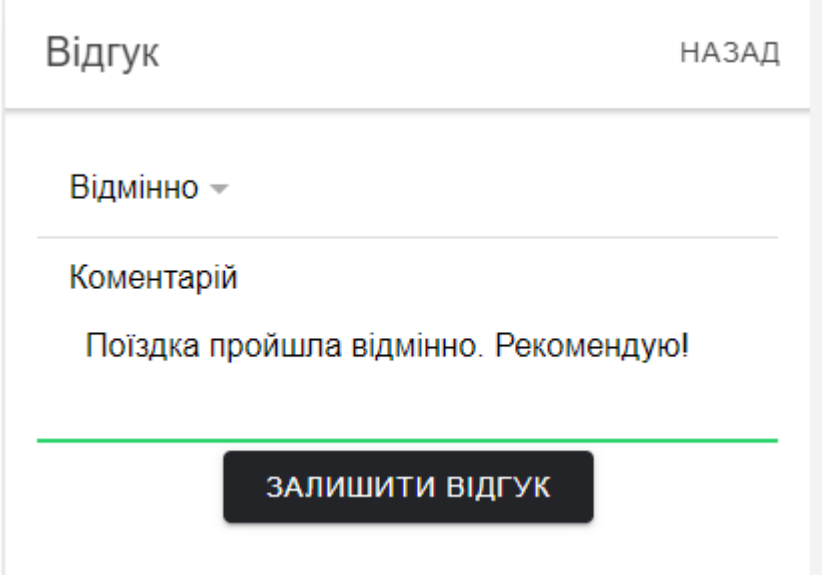


Рисунок 4.14 – Форма заповнення відгуку

При цьому поїздка потрапляє у список архівних поїздок на вкладці «Поїздки» і у пасажира з’являється можливість оцінити поїздку та залишити відгук до неї.

## Висновки до розділу 4

В першому підрозділу четвертого розділу було висвітлено поняття бази даних та системи керування базами даних, описані сутності системи та розроблено діаграму бази даних сервісу. Було обрано СКБД – PostgreSQL. База даних містить 15 таблиць, в котрих зберігаються всі дані, необхідні для коректної роботи системи.

В другому підрозділі було описано користувацький інтерфейс та продемонстровано роботу системи за допомогою знімків екрану. Було продемонстровано роботу всіх основних функцій та можливостей.

Сервіс пошуку для автомобільних попутників представлена у вигляді мобільного застосунку та повинен бути розміщеним на платформі Google Play та Apple Store.

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**Чорноморський національний університет імені Петра Могили**
**Факультет комп'ютерних наук**
**Кафедра інтелектуальних інформаційних систем**

## Спеціальний розділ

# ОХОРОНА ПРАЦІ

**до кваліфікаційної роботи**

на тему:

# «СЕРВІС ПОШУКУ ДЛЯ АВТОМОБІЛЬНИХ ПОПУТНИКІВ»

Спеціальність 122 «Комп'ютерні науки»

**122 – БКР – 402.21810305**

***Виконав студент 4-го курсу, групи 402***
_____***М. В. Бочков***
*(підпис, ініціали та прізвище)*
«___» _____ 2022 р.

***Консультант канд. техн. наук, доцент***
*(наук. ступінь, вчене звання)*
_____***А. О. Алексєєва***
*(підпис, ініціали та прізвище)*
«___» _____ 2022 р.

**Миколаїв – 2022**

# 5 ОХОРОНА ПРАЦІ

За останні декілька років завдяки мобільному телефону райдшеринг, або спільні поїздки з попутниками, набули неабиякої популярності. Автоматизація процесу пошуку водіїв та потрібних маршрутів зробила цей процес простим та швидким.

Поїзди або автобуси їдуть за великим маршрутом задля того щоб везти якомога більшу кількість людей, а водії, які користуються сервісами для пошуку автомобільних попутників, вибирають найкоротшу відстань щоб доїхати до необхідного населеного пункту. Саме цією зручністю, комфортом, економією часу та грошей такі сервіси приваблюють користувачів. Водії, в свою чергу, заощаджують гроші на пальному, а інколи навіть заробляють на цьому.

В даному розділі будуть розглянуті заходи, щодо створення необхідних умов праці для диспетчерів, при роботі з персональним комп'ютером і мережним устаткуванням, які забезпечують повну безпеку цієї роботи.

Охорона праці користувача комп'ютерної техніки та мережевого обладнання включає в себе соціально-економічні, організаційні, технічні, гігієнічні, лікувально-профілактичні та інші заходи, спрямовані на збереження життя і здоров'я людини. Основні документи, які регламентують ці заходи – це Закон України з охорони праці, Конституція України, Кодекс законів про працю, відомчі інструкції з техніки безпеки. Нагляд і контроль за дотриманням правил і норм техніки безпеки на підприємстві здійснює відділ з охорони праці підприємства.

## 5.1 Аналіз умов праці диспетчерів за персональним комп’ютером

На даний час питання охорони праці є одним з найактуальніших, тому що кожен робітник використовує комп’ютер, який впливає на здоров’я людини, тому

виникає потреба у виконання всіх норм і умов охорони праці під час роботи з комп’ютерною технікою. Всі фактори, які впливають під час роботи поділяються на небезпечні і шкідливі.

До небезпечних і шкідливих факторів виробництва належать виробничі фактори, які можуть завдати шкоди здоров'ю та життю людини. До таких факторів відносяться шкідливі для здоров'я електричні і магнітні коливання, інфрачервоне та іонізуюче випромінювання; шум та вібрації, статична електрика.

Крім того, існують вторинні фактори, які посилюють цю ситуацію, до них можна віднести наступні: тісні приміщення, в яких сконцентровано безліч комп’ютерів, заниження або завищення рівнів штучного та природного освітлення, порушення режиму провітрювання, використання подовженого часу роботи за комп’ютером, відсутність умов працювати за комп’ютером на необхідній від користувача відстані, вимушена статична поза тощо [21].

В даному випадку розрізняють такі шкідливі фактори:

1. Небезпека ураження електричним струмом від електронного обладнання.
2. Вплив шкідливих випромінювань, що виникають в електронно-променевих трубках і пристроях бездротового зв'язку.
3. Небезпека, яка криється в рухомих частинах електронно-механічного обладнання.
4. Небезпека опіку від нагрітих елементів електронної техніки (головки принтерів і т.д.).
5. Велика концентрація електрообладнання підвищує можливість електричного замикання що, в свою чергу, підвищує ймовірність пожеж.

Отже, комп'ютерна техніка може негативно впливати на здоров'я і фізичний стан людини. Тому, при облаштуванні й обладнанні комп’ютерних кабінетів, нормуванні тривалості роботи, необхідно неухильно дотримуватися санітарних, ергономічних, гігієнічних норм та проводити певні фізкультурно-оздоровчі заходи. Це дозволить усім працюючим за комп'ютерами значно зменшити їх вплив на здоров’я, фізичний стан та психіку людини.

Серед основних заходів профілактики захворювань, пов'язаних з використанням комп'ютера, виділятимемо відповідний технічний стан електронно-обчислювальних засобів, правильне розміщення комп'ютерів і периферійного обладнання, раціональну організацію робочого місця, дотримання норм освітлення та мікроклімату приміщення, додержання необхідного режиму роботи за ПК, наявність спеціалізованих комп'ютерних меблів [22].

Таким чином, виявлені небезпечні і шкідливі виробничі фактори викликають необхідність проведення технічних, технологічних, організаційних та протипожежних заходів, які і будуть розроблені в даному розділі для створення здорових і безпечних умов праці.

## 5.2 Техніка безпеки для диспетчерів на підприємстві

### 5.2.1 Загальні вимоги безпеки при роботі за ПК на підприємстві

До роботи з персональним комп’ютером допускаються особи, які пройшли медичний огляд, вступний інструктаж з охорони праці, первинний інструктаж з охорони праці на робочому місці та мають навички з даного фаху. В подальшому вони проходять повторні інструктажі з охорони праці на робочому місці один раз на півріччя та медичні огляди один раз на два роки.

Робочі місця мають бути розташовані на відстані не менше 1,5 м від стіни з вікнами, від інших стін на відстані 1м, між собою на відстані не менше 1,5 м. Також необхідним є те, щоб джерело природнього світла розташовувалось ліворуч від робочого місця для того щоб уникнути попадання в очі прямого світла.

Для уникнення світлових відблисків екрану, клавіатури в напрямку очей користувача, від світильників загального освітлення або сонячних променів, необхідно використовувати антиполискові сітки (фільтри з металевої або нейлонової сітки використовувати не рекомендується, тому що сітка спотворює зображення через інтерференцію світла), спеціальні фільтри для екранів, захисні козирки, жалюзі на вікнах. Найкращу якість зображення забезпечують скляні поляризаційні фільтри, які усувають майже всі відблиски, роблять зображення чітким і контрастним.

При роботі з текстовою інформацією найбільш фізіологічним правильним є зображення чорних знаків на світлому фоні.

Монітор повинен бути розташований на робочому місці так, щоб поверхня екрана знаходилася в центрі поля зору на відстані 40 - 70 см від очей користувача. Рекомендується розміщувати елементи робочого місця так, щоб була однакова відстань очей від екрана та клавіатури.

Раціональною робочою позою може вважатися таке положення, при якому ступні працівника розташовані горизонтально на підлозі, стегна зорієнтовані у горизонтальній площині, верхні частини рук - вертикальні. Кут ліктьового суглоба коливається в межах 70-90°, зап'ястя зігнуті під кутом не більше ніж 20°, нахил голови 15-20°.

Для нейтралізації зарядів статичної електрики в приміщенні, де виконується робота на комп'ютерах, в тому числі на лазерних та світлодіодних

принтерах, рекомендується збільшувати вологість повітря за допомогою кімнатних зволожувачів [23]. Повинна бути ефективна вентиляція і підтримуватися відносна вологість повітря на рівні 40-60%.

Для забезпечення оптимальної працездатності і збереження здоров'я професійних користувачів протягом робочої зміни повинні встановлюватися регламентовані перерви. Тривалість безперервної роботи з персональним комп'ютером без регламентованої перерви не повинна перевищувати 2 годин.

### 5.2.2 Вимоги безпеки перед початком роботи диспетчерів

1. Впевнитись, що на робочому місці немає сторонніх предметів, все обладнання і блоки ПК з’єднані з системним блоком.
2. Повернути монітор так, щоб він був під прямим кутом; при цьому екран має бути трохи нахиленим – нижній край ближче до користувача; відрегулювати яскравість та контрастність монітора.
3. Перевірити загальний стан комп’ютерної техніки, справність електропроводки, з’єднувальних проводів, штепсельних вилок, розеток, заземлення захисного екрана.
4. Відрегулювати висоту та нахил спинки крісла за потребою користувача [24].

### 5.2.3 Вимоги безпеки під час роботи диспетчерів

1. Не залишати працюючий комп’ютер і додаткові пристрої без нагляду.
2. Підключати і відключати кабелі пристроїв ПК тільки при відключеній напрузі.

3. Подавати напругу на пристрої і окремі блоки ПК тільки після перевірки надійності і правильності кріплення провідників заземлення, справності кабелів і роз'ємів мережі електроживлення.
4. При виявленні несправності в пристроях ПК необхідно вимкнути апаратуру та звернутися до спеціаліста з технічного обслуговування ПК.
5. Для профілактики порушень і підтримання працездатності користувача ПК власником повинні бути введені додаткові регламентовані перерви для відпочинку.
6. У період роботи за дисплеєм необхідно передбачити через кожні 45 хвилин п'ятихвилинні перерви для відпочинку [25].

### 5.2.4 Вимоги безпеки для диспетчерів в аварійних ситуаціях

1. При припиненні подавання електроенергії, вимкнути апаратуру.
2. Якщо на металевих частинах апаратури виявлено напругу, заземлюючий провід обірваний - необхідно вимкнути обладнання, доповісти керівникові про несправності електрообладнання і не приступати до роботи.
3. При появі запаху паленого, незвичного звуку - припинити роботу і повідомити керівнику.
4. При виникненні пожежі негайно вимкнути обладнання, знеструмити електромережу, повідомити про пожежу всім співробітникам, приступити до гасіння осередку пожежі наявними засобами пожежогасіння та викликати пожежну службу.
5. При нещасному випадку необхідно звільнити потерпілого від травмуючого фактора, звернутися до медпункту за допомогою, по

можливості зберегти місце травмування в тому стані, в якому воно було на момент травмування [26].

### 5.2.5 Вимоги безпеки для диспетчерів після закінчення роботи

1. Виключити ПК в послідовності згідно з інструкцією з експлуатації.
2. Доповісти керівнику за технічний стан комп’ютера, про всі зауваження і несправності в роботі комп’ютера, якщо вони були.
3. Привести в порядок робоче місце, прибрати сміття та зайві предмети.

## 5.3 Виробнича санітарія та гігієна праці диспетчерів

Регулярна робота за комп'ютером без застосування відповідних захисних засобів призводить до зниження імунітету, захворювання органів зору, серцево-судинної системи та шлунково-кишкового тракту. Спілкування з комп’ютером супроводжується нервовим напруженням, оскільки вимагає швидкої реакції. Короткочасна робота з комп'ютером, встановленим з грубим порушенням гігієнічних норм і правил, призводить до підвищеного стомлення [27]. Праця людини, що протікає в умовах надмірного нервово-емоційного напруження, довготривалих статичних навантажень, обмеженої рухової активності призводить до неврозів, відхилень у психіці, захворювань опорно-рухового апарату, серцево-судинної системи тощо [28].

Недостатнє і нерівномірне освітлення робочого місця призводить до перенапруження органу зору, перевтоми організму, виникнення нервової дратівливості, зниження уваги. Доведено, що у половини працюючих робітників в таких умовах спостерігаються дефекти зору – короткозорість, зниження гостроти зору та інше, а в 3-8% чоловіків виникає частковий дальтонізм.

Захист від електромагнітного випромінювання комп'ютера [29]:

1. По можливості придбати рідкокристалічний монітор, оскільки його випромінювання значно менше, ніж в поширених ЕЛТ моніторів (монітор з електронно-променевою трубкою).
2. Системний блок і монітор повинні знаходитися на відстані 40-70 см від користувача.
3. Не залишати комп'ютер включеним на тривалий час якщо він не використовується.
4. У зв'язку з тим, що електромагнітне випромінювання від стінок монітора набагато більше, необхідно поставити монітор в кут, так щоб випромінювання поглиналося стінами. Особливу увагу варто звернути на розстановку моніторів в адміністративному приміщенні.
5. По можливості скоротити час роботи за комп'ютером і частіше робити перерви.
6. Комп'ютер та додаткова апаратура повинні бути заземленими.

Додатково, для збереження належного рівня здоров’я та професійної придатності робітників, рекомендується виділити на підприємстві окреме побутове приміщення для перепочинку працівників і зняття ними нервово-емоційного напруження, що виникає при роботі з комп’ютером.

### 5.4 Психофізіологія і ергономіка праці диспетчерів

Психофізіологічні причини – це помилкові дії внаслідок втоми працівника через надмірну важкість і напруженість роботи, монотонність праці, хворобливий стан працівника, необережність, невідповідність психофізіологічних чи антропометричних даних працівника.

До небезпечних та шкідливих психофізіологічних виробничих чинників належать фізичні (статичні, динамічні) і нервово-психічні перевантаження

(розумова перенапруга, монотонність праці, перенапруження зорового, слухового, тактильного аналізаторів, емоційні перевантаження).

Надмірні фізичні та нервово-психічні перевантаження зумовлюють зміни у фізіологічному та психічному станах працівника, призводять до розвитку втоми та перевтоми [30].

Залежно від функціональних порушень в організмі робітників під впливом трудових навантажень розрізняють чотири ступеня втоми:

1. Втома першого ступеня (маловиражена) проявляється різними порушеннями при виконанні точних рухів з незначними м'язовими зусиллями в зв'язку з невідповідністю силових дій з боку працівника. При цьому робота з помірними і максимальними зусиллями виконується без істотних змін.
2. Втома другого ступеня (помірна) характеризується незначним зниженням працездатності і витривалості. Порушення виявляються в збільшенні кількості помилок при виконанні дій, які вимагають незначних або максимальних м'язових зусиль.
3. Втома третього ступеня (виражена) характеризується відчутним зменшенням працездатності і витривалості рухового апарату. Час реакцій збільшується, швидкість оптимальних і максимальних робочих реакцій сповільнюється, м'язова сила при виконанні максимальних зусиль зменшується. Мінімальні м'язові зусилля виконуються з надмірною силою в 2–2,5 рази. Загальна працездатність зменшується.
4. Втома четвертого ступеня (сильновиражена) супроводжується ультрапарадоксальними реакціями. Всі позитивні сигнали

працівником не сприймаються, а негативні викликають позитивні реакції, що призводить до помилок, аварій тощо.

Втома може бути фізичною (м'язовою) або нервово-психічною (центральною). Обидві форми втоми поєднуються при важкій роботі, їх не можна строго відокремити одну від одної. Важка фізична робота призводить до м'язового стомлення, а посилена розумова або монотонна робота викликає втому центрального походження. Слід чітко розмежовувати стомлення і втому, обумовленою потребою в сні.

Основною відмінністю втоми від перевтоми є зворотність зрушень при втомі і неповна зворотність їх при перевтомі. Втома негативно не впливає на здоров'я і часто справляє позитивний вплив на організм людини, в той час як перевтома має негативний вплив. Разом з тим критерії перевтоми не розроблені. Проявами перевтоми є головний біль, підвищена стомлюваність, дратівливість, нервозність, порушення сну і такі захворювання, як вегетативно-судинна дистонія, артеріальна гіпертензія, виразкова хвороба, ішемічна хвороба серця, інші професійні захворювання. Перевтома може бути гострою як результат одноразової напруженої діяльності і хронічною – як результат тривалої повторної діяльності [31].

Ергономіка - наука, що вивчає людину і її діяльність в умовах сучасного виробництва з метою оптимізації знарядь, умов і процесу праці. Головними цілями ергономіки є безпека праці, підвищення ефективності системи «людина – техніка – середовище»; комфортність – задоволеність людини результатами своєї праці і забезпечення умов для розвитку особистості людини в процесі праці.

Ергономіку формують такі групи показників:

1. Антропометричні - регламентують відповідність машини розмірам і форми тіла працюючої людини, рухливості частин тіла і іншим параметрам.
2. Гігієнічні - характеризують метеорологічні умови мікроклімату та обмеження впливу шкідливих факторів зовнішнього середовища.
3. Фізіологічні і психофізіологічні - характеризують ті ергономічні вимоги, які визначають відповідність силовим, швидкісним, енергетичним, зоровим, слуховим, тактильним можливостям і особливостям людини.
4. Психологічні - відображають відповідність машини можливостям і особливостям сприйняття пам'яті, мислення, психомоторики людини.

**Висновки до розділу 5**

Охорона праці – це система збереження життя, здоров'я і працездатності працівників у процесі трудової діяльності, що включає в себе правові, соціально-економічні, організаційні, технічні, санітарно-гігієнічні, лікувально-профілактичні, реабілітаційні та інші заходи.

Під час виконання спеціальної частини до кваліфікаційної бакалаврської роботи, яка присвячена питанням охорони праці було розглянуто та проаналізовано умови праці та техніки безпеки для диспетчерів на підприємстві. Також були описані загальні вимоги безпеки, вимоги безпеки перед початком роботи, під час роботи та після закінчення роботи. Важливе значення мав опис вимог безпеки в аварійних ситуаціях.

Також, не менш важливим, був опис виробничої санітарії та гігієни праці диспетчерів на виробництві, психофізіології і ергономіки праці. В даних

підрозділах було розглянуто поняття втоми і перевтоми та їх негативного впливу на організм.

Для зменшення негативного впливу на організм диспетчера допоможе дотримання всіх вищеописаних умов та дотримання режиму праці і відпочинку.

Розділ по охороні праці був розроблений заради того, щоб показати основні вимоги безпеки під час праці диспетчерів.

# ВИСНОВКИ

Перед виконанням кваліфікаційної роботи бакалавра оглянуто предметну сферу і розглянуто популярні існуючі аналоги, описано їх призначення та наведено їх недоліки. На основі отриманої інформації під час аналізу сфери та аналогів були висвітленні основні вимоги до проєкту та створено технічне завдання.

Наступним кроком був описаний перелік засобів для реалізації поставленого завдання. В якості серверної мови програмування було використано мову Java, для написання клієнтської частини обрано Ionic Framework, а всю інформацію збережено в базі даних PostgreSQL.

В третьому розділі було створено діаграму використання та діаграму розгортання і наведено перелік дій (сценаріїв), за яким користувач може взаємодіяти з сервісом.

В четвертому розділі були описані сутності системи та розроблено діаграму бази даних сервісу. Також описано користувацький інтерфейс та продемонстровано роботу всіх основних функцій та можливостей системи за допомогою знімків екрану.

За результатами спеціального розділу з охорони праці були створені безпечні умови праці на робочих місцях для диспетчерів. Досягнуто це було за допомогою опрацювання питань умов праці, техніки безпеки, виробничої санітарії, психофізіології та ергономіки праці.

Мету роботи, що полягала в покращенні процесу пошуку автомобільних попутників та аналізу даних під час поїздок за рахунок автоматизації було досягнуто.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Карпулінг і райдшеринг: веб-сайт. URL: https://interesnye-istorii.in.ua/carpooling-ridesharing/ (дата звернення: 25.05.2022).

2) Райдшеринг в Україні: економні подорожі із попутниками: веб-сайт. URL: https://autogeek.com.ua/ridesharing-v-ukraine-ekonomnyie-puteshestviya-s-poputchikami/ (дата звернення: 25.05.2022).

3) Попутники сприяють дорожній дисципліні (результати дослідження та інфографіка) – веб-сайт. URL: https://blog.blablacar.com.ua/newsroom/novini/poputniki-spriiaiut-dorozhnii-distciplini-rezultati-doslidzhennia-ta-infografika (дата звернення: 25.05.2022).

4) Сервіси, альтернативні BlaBlaCar: огляд варіантів: веб-сервер. URL: https://zagraninfo.com/analogi-blablacar.html (дата звернення: 25.05.2022).

5) Застосунок аналог: inDriver: веб-сайт. URL: https://habr.com/ru/company/indriver/profile/ (дата звернення: 25.05.2022).

6) Стаття стосовно трирівневої архітектури: веб-сайт. URL: https://javarush.ru/groups/posts/2519-chastjh-2-pogovorim-nemnogo-ob-arkhitekture-po (дата звернення: 25.05.2022).

7) Мова програмування Java: веб-сайт. URL: https://web-creator.ru/articles/java (дата звернення: 26.05.2022).

8) Чому Java – найпопулярніша мова програмування у світі: веб-сайт. URL: https://java.lviv.ua/chomu-java-najpopulyarnisha-mova-programuvannya-u-sviti (дата звернення: 26.05.2022).

9) Стаття відносно Spring Framework Java: веб-сайт. URL: https://javarush.ru/groups/posts/spring-framework-java-1 (дата звернення: 26.05.2022).

10) Стаття відносно Inversion of Control: веб-сайт. URL: https://www.tutorialsteacher.com/ioc/inversion-of-control (дата звернення: 26.05.2022).

11) Стаття відносно Dependency Injection: веб-сайт. URL: https://www.tutorialsteacher.com/ioc/dependency-injection (дата звернення: 26.05.2022).

12) Плюси та мінуси розробки додатків на Ionic: веб-сайт. URL: https://dou.ua/lenta/articles/ionic-development/ (дата звернення: 26.05.2022).

13) Стаття відносно SMTP-серверу : веб-сервер. URL: https://www.unisender.com/ru/support/about/glossary/chto-takoe-smtp/ (дата звернення: 26.05.2022).

14) Навіщо використовувати PostgreSQL?: веб-сайт. URL: https://www.postgresql.org/about/ (дата звернення: 26.05.2022).

15) PostgreSQL: Документація: 9.6: веб-сайт. URL:- https://postgrespro.ru/docs/postgresql/9.6/intro-whatis (дата звернення: 26.05.2022).

16) Мова UML. Діаграма використання: веб-сайт. URL: http://p4ilka.blogspot.com/2018/12/uml.html (дата звернення: 27.05.2022).

17) Стаття відносно діаграми розгортання: веб-сайт. URL: https://studfile.net/preview/5010027/page:6/ (дата звернення: 27.05.2022).

18) Стаття відносно схеми розгортання: веб-сайт. URL: https://uk.upwiki.one/wiki/Deployment_diagram (дата звернення: 27.05.2022).

19) Що таке use case та для чого вони потрібні: веб-сайт. URL: https://training.qatestlab.com/blog/technical-articles/what-is-a-use-case-and-what-are-they-for/ (дата звернення: 27.05.2022).

20) Стаття відносно бази даних та системи керування базами даних: веб-сайт. URL: https://hostiq.ua/wiki/ukr/database/ (дата звернення: 27.05.2022).

21) Стаття відносно комп’ютера і здоров’я людини: веб-сайт. URL: https://lugynska-gromada.gov.ua/news/1604391805 (дата звернення: 02.06.2022).

22) Стаття відносно впливу комп’ютера на здоров’я користувача: веб-сайт. URL: http://mirgorod-gorono.at.ua/publ/metodob_39_ednannja_vchiteliv/inform/vpliv_komp_jutera_na_zdorov_ja_koristuvacha/26-1-0-238 (дата звернення: 02.06.2022).

23) Стаття відносно техніки безпеки при роботі з комп’ютером: веб-сайт. URL: https://studopedia.com.ua/1_43654_tehnika-bezpeki-pri-roboti-z-pk.html (дата звернення: 02.06.2022).

24) Стаття відносно інструкції з охорони праці при роботі з персональним комп’ютером: веб-сайт. URL: https://www.sop.com.ua/article/485-nstruktsya-z-ohoroni-prats-pri-robot-na-personalnomu-kompyuter (дата звернення: 02.06.2022).

25) Стаття відносно вимог безпеки під час роботи на ПК: веб-сайт. URL: https://sites.google.com/site/ohoronapraci44/33-vimogi-bezpeki-pid-cas-roboti-na-pk (дата звернення: 02.06.2022).

26) Стаття відносно інструкції з охорони праці при роботі з комп'ютером, принтером, ксероксом та іншою оргтехнікою: веб-сайт. URL: https://osvita-docs.com/node/41 (дата звернення: 02.06.2022).

27) Стаття відносно санітарно-гігієнічної вимоги роботи на комп'ютері в навчальних закладах: веб-сайт. URL: https://city-adm.lviv.ua/news/science-and-health/medicine/219680-sanitarno-hihiienichni-vymohy-roboty-na-komp-iuteri-v-navchalnykh-zakladakh (дата звернення: 02.06.2022).

28) Стаття відносно основи фізіології, гігієни праці та виробничої санітарії, санітарно-гігієнічних вимог та їх реалізації в технологічних процесах,

факторів, що впливають на функціональний стан користувача: веб-сайт. URL: https://infopedia.su/8xfa0c.html (дата звернення: 02.06.2022).

29) ГОСТ 12.2.007.0-75. ССБТ. Изделия электротехнические. Общие требования безопасности. М.: Издательство стандартов, 1981. 38с

30) Стаття відносно психофізіологічного перевантаження на роботі – як чинника виникнення травматизму: веб-сайт. URL: https://oppb.com.ua/news/psyhofiziologichne-perevantazhennya-na-roboti-yak-chynnyk-vynyknennya-travmatyzmu (дата звернення: 02.06.2022).

31) Стаття відносно втоми з точки зору фізіології: веб-сайт. URL: https://uk.wikipedia.org/wiki/Втома_(фізіологія) (дата звернення: 02.06.2022).

# ДОДАТОК А
## Клас контролер для взаємодії пасажира з поїздкою

```
@RestController
@RequestMapping("/user-trip")
public class UserTripController {
    final Logger logger = LoggerFactory.getLogger(UserTripController.class);
    private final UserTripService userTripService;
    private final DriverService driverService;

    public UserTripController(UserTripService userTripService,
                              DriverService driverService) {
        this.userTripService = userTripService;
        this.driverService = driverService;
    }

    @PostMapping("/getExistingUserTrip")
    public ResponseEntity getExistingUserTrip(@RequestBody UserTrip userTrip) {
        try {
            userTrip =
userTripService.getUserTripByUserAndTrip(userTrip.getUser(),
userTrip.getTrip());

            return ResponseEntity.ok(userTrip);
        } catch (Exception e) {
            logger.error("Exception in getExistingUserTrip:" +
Arrays.toString(e.getStackTrace()));
            return ResponseEntity.badRequest().body("Error: " + e.getMessage());
        }
    }

    @GetMapping("/getUserTripListByTripId")
    public ResponseEntity getUserTripListByTripId(@RequestParam Long tripId) {
        try {
            List<UserTrip> userTripList =
userTripService.getUserTripListByTripId(tripId);
            List<UserTripDTO> userTripDTOList =
UserTripFactory.createUserTripDTOList(userTripList);
            return ResponseEntity.ok(userTripDTOList);
        } catch (Exception e) {
            logger.error("Exception in bookUserOnTrip:" +
Arrays.toString(e.getStackTrace()));
            return ResponseEntity.badRequest().body("Error: " + e.getMessage());
        }
    }

    @PostMapping("/book")
    public ResponseEntity bookUserOnTrip(@RequestBody UserTripDTO userTripDTO) {
        try {
            UserTrip userTrip = UserTripFactory.createVOFromDTO(userTripDTO);
            userTripDTO =
UserTripFactory.createDTOFromVO(userTripService.bookUserOnTrip(userTrip));
            return ResponseEntity.ok(userTripDTO);
        } catch (Exception e) {
```

```
            logger.error("Exception in bookUserOnTrip:" + e);
            return ResponseEntity.badRequest().body("Error: " + e.getMessage());
        }
    }

    @PostMapping("/acceptBookingByDriver")
    public ResponseEntity acceptBookingByDriver(@RequestBody UserTripDTO
userTripDTO) {
        try {
            UserTrip userTrip = UserTripFactory.createVOFromDTO(userTripDTO);
            userTripService.acceptBooking(userTrip);
            return ResponseEntity.ok().build();
        } catch (Exception e) {
            logger.error("Exception in acceptBookingByDriver:" + e);
            return ResponseEntity.badRequest().body("Error: " + e.getMessage());
        }
    }

    @PostMapping("/deleteBookingByUser")
    public ResponseEntity deleteBookingByUser(@RequestBody UserTripDTO
userTripDTO) {
        try {
            UserTrip userTrip = UserTripFactory.createVOFromDTO(userTripDTO);
            userTripDTO =
UserTripFactory.createDTOFromVO(userTripService.deleteBookingByUser(userTrip));
            return ResponseEntity.ok(userTripDTO);
        } catch (Exception e) {
            logger.error("Exception in deleteBookingByUser:" + e);
            return ResponseEntity.badRequest().body("Error: " + e.getMessage());
        }
    }

    @PostMapping("/deleteBookingByDriver")
    public ResponseEntity deleteBookingByDriver(@RequestBody UserTripDTO
userTripDTO) {
        try {
            UserTrip userTrip = UserTripFactory.createVOFromDTO(userTripDTO);
            userTripDTO =
UserTripFactory.createDTOFromVO(userTripService.deleteBookingByDriver(userTrip))
;

            return ResponseEntity.ok(userTripDTO);
        } catch (Exception e) {
            logger.error("Exception in deleteBookingByDriver:" + e);
            return ResponseEntity.badRequest().body("Error: " + e.getMessage());
        }
    }

    @PostMapping("/startTrip")
    public ResponseEntity startTrip(@RequestBody UserTripDTO userTripDTO) {
        try {
            UserTrip userTrip = UserTripFactory.createVOFromDTO(userTripDTO);
            userTripDTO =
UserTripFactory.createDTOFromVO(userTripService.startTrip(userTrip));
            return ResponseEntity.ok(userTripDTO);
```

```
        } catch (Exception e) {
            logger.error("Exception in startTrip:" + e);
            return ResponseEntity.badRequest().body("Error: " + e.getMessage());
        }
    }

    @PostMapping("/endTrip")
    public ResponseEntity endTrip(@RequestBody UserTrip userTrip) {
        try {
            userTrip = userTripService.endTrip(userTrip);
            return ResponseEntity.ok(userTrip);
        } catch (Exception e) {
            logger.error("Exception in endTrip:" + e);
            return ResponseEntity.badRequest().body("Error: " + e.getMessage());
        }
    }

    @PostMapping("/reportAboutDanger")
    public ResponseEntity reportAboutDanger(@RequestBody UserTripDTO
userTripDTO) {
        try {
            UserTrip userTrip = UserTripFactory.createVOFromDTO(userTripDTO);
            userTripDTO =
UserTripFactory.createDTOFromVO(userTripService.reportAboutDanger(userTrip));

            return ResponseEntity.ok(userTripDTO);
        } catch (Exception e) {
            logger.error("Exception in reportAboutDanger:" + e);
            return ResponseEntity.badRequest().body("Error: " + e.getMessage());
        }
    }

    @PostMapping("/addFeedback")
    public ResponseEntity addFeedback(@RequestBody DriverFeedback
driverFeedback) {
        try {
            driverService.addDriverFeedback(driverFeedback);
            return ResponseEntity.ok().build();
        } catch (Exception e) {
            logger.error("Exception in addFeedback:" + e);
            return ResponseEntity.badRequest().body("Error: " + e.getMessage());
        }
    }
}
```

# ДОДАТОК Б
# Клас обробки локацій пасажиру

```
@Component
public class LocationSocketHandler extends TextWebSocketHandler {

    Logger logger = LoggerFactory.getLogger(LocationSocketHandler.class);
    private final TripService tripService;
    private final UserTripService userTripService;
    private final BingMapAPIImpl bingMapAPIImpl;
    private static int INDEX = 0;

    public LocationSocketHandler(TripService tripService,
                                 UserTripService userTripService,
                                 BingMapAPIImpl bingMapAPIImpl) {
        this.tripService = tripService;
        this.userTripService = userTripService;
        this.bingMapAPIImpl = bingMapAPIImpl;
    }

    @Override
    protected void handleTextMessage(WebSocketSession session, TextMessage
message) throws Exception {
        super.handleTextMessage(session, message);

        try {
            String location = getLocationFromMessage(message);
            Long userId = Long.parseLong(message.getPayload().split(" ")[1]);
            double minDistance = 1000D;
            location = Objects.requireNonNull(location);

            UserTrip userTrip =
userTripService.getUserTripByUserId(userId).orElseThrow();

            Optional<List<String>> locationPointsOpt =
tripService.getLocationPointsForTripByUserId(userId);
            if (locationPointsOpt.isPresent()) {
                List<String> locationPoints = locationPointsOpt.get();
                for (int i = 0; i < locationPoints.size() - 1; i++) {
                    Optional<List<String>>
locationPointsBetweenTwoCoordinatesOpt
                            =
bingMapAPIImpl.getLocationPointsBetweenTwoCoordinates(locationPoints.get(i),
locationPoints.get(i));

                    if (locationPointsBetweenTwoCoordinatesOpt.isPresent()) {
                        for (String locationPoint :
locationPointsBetweenTwoCoordinatesOpt.get()) {
                            double distance = calculateDistance(location,
locationPoint);

                            if (distance < minDistance) minDistance = distance;
                        }
                    }
```

```
                }
            }

            if (minDistance > 50D && minDistance != 1000D) {
                session.sendMessage(new TextMessage(generatePayload()));
            }

            userTrip.setLat(location.split(",")[0]);
            userTrip.setLng(location.split(",")[1]);

            userTripService.addOrUpdateUserTrip(userTrip);
        } catch (Exception e) {
            logger.error("Exception in LocationSocketHandler", e);
        }
    }

    private String getLocationFromMessage(TextMessage message) {
        if (message.getPayload().contains("undefined")) {
            if (INDEX < locations.size()) return locations.get(INDEX++);
        } else {
            return message.getPayload().split(" ")[0];
        }
        return null;
    }

    public double calculateDistance(String locationPoint1, String
locationPoint2) {
        double lat1 = Double.parseDouble(locationPoint1.split(",")[0]);
        double lat2 = Double.parseDouble(locationPoint2.split(",")[0]);
        double lng1 = Double.parseDouble(locationPoint1.split(",")[1]);
        double lng2 = Double.parseDouble(locationPoint2.split(",")[1]);

        return calculateDistance(lat1, lat2, lng1, lng2);
    }

    public double calculateDistance(double lat1, double lat2, double lon1,
double lon2) {
        // The math module contains a function named toRadians which converts
from  degrees to radians.
        lon1 = Math.toRadians(lon1);
        lon2 = Math.toRadians(lon2);
        lat1 = Math.toRadians(lat1);
        lat2 = Math.toRadians(lat2);

        // Haversine formula
        double dlon = lon2 - lon1;
        double dlat = lat2 - lat1;
        double a = Math.pow(Math.sin(dlat / 2), 2)
                + Math.cos(lat1) * Math.cos(lat2)
                * Math.pow(Math.sin(dlon / 2), 2);
        double c = 2 * Math.asin(Math.sqrt(a));
        // Radius of earth in kilometers. Use 3956 for miles
        double r = 6371;

        // calculate the result
```

```
        return (c * r);
    }

    private String generatePayload() {
        final String messageStr = "Водій змінює маршрут";

        return new JSONObject().put("message", messageStr).toString();
    }
}
```