МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук,
управління та адміністрування
Кафедра інформаційних технологій

Кваліфікаційна робота бакалавра

на тему: Розробка веб-застосунку для пошуку зарядних станцій для електромобілів

Виконав студент групи **К-19б**
спеціальності 122 Комп'ютерні науки
Вітвицький Микола Олександрович

Керівник **Ткач Т.Б., к.ф.-м.н.**

Консультант

Рецензент **Перелигін Б.В., к.т.н., доцент**

Одеса 2023

ЗМІСТ

ТЕРМІНИ І СКОРОЧЕННЯ

АЗС – це комплекс обладнання на придорожній території, призначений для заправки паливом транспортних засобів.

UML – уніфікована мова моделювання, використовується у парадигмі об'єктно-орієнтованого програмування.

GPS – система глобального позиціонування.

GUI – тип інтерфейсу, який дає змогу користувачам взаємодіяти з електронними пристроями через графічні зображення.

DTO (Data transfer object) – один із шаблонів проєктування, який використовують для передачі даних між підсистемами програми.

БД – база даних.

SOLID – це набір принципів об'єктно-орієнтованого програмування.

JVM (Java Virtual Machine) – віртуальна машина для виконання байт-коду Java.

MySQL – це система управління базами даних, яка використовується для підтримки реляційних баз даних.

ОС – операційна система.

ВСТУП

Автотранспорт на сьогоднішній день є самим поширеним і зручним способом пересування. Але, незважаючи на свої переваги, він несе основну долю екологічного збитку навколишньому середовищу – до 63%. Забруднення відбувається і на стадіях виробництва, і в процесі експлуатації, і при утилізації автомобілів, масел і палива.

У світі спостерігається глобальна тенденція зі зниження людського впливу на навколишнє середовище. І одним із засобів зменшення негативного впливу є перехід на електромобілі. Все це привело до того, що прогресивні країни стали проектувати і створювати електромобілі.

Так, уряди європейських країн декларують плани на заборону продажів дизельних та бензинових авто на найближчі десятиліття. Зокрема, до 2025 року Норвегія планує повністю припинити продаж машин з двигунами внутрішнього згорання, а до 2030 до неї приєднаються Швеція та Данія.

Але проблемою електромобілів досі залишається інфраструктура. Так кількість зарядних станцій у Німеччині становить близько 21 000 зарядних станцій, у той час як звичайних АЗС налічується близько 15 000. Однак, варто зауважити, що АЗС мають більшу кількість паливних насосів й процес заправки значно швидший [1].

Отже, дуже корисно мати можливість перегляду усіх зарядних станцій, їх стан, характеристики та можливість бронювання.

Тому, головною метою роботи є розробка застосунку для пошуку зарядної станції для електромобілів, яка аналізує інформацію про географічне місце користувача, станцій та їхній стан, шукає найближчу від нього вільну станцію та будує маршрут до неї.

Для успішної реалізації мети роботи є необхідним розв’язання наступних задач:

– проведення аналізу конкурентних аналогів системи з метою вивчення їх переваг та недоліків;

– формування вимог до системи для пошуку зарядних станцій для електромобілів;
– проєктування архітектури інформаційної системи та побудова основних UML-діаграм, необхідних для її розробки;
– аналіз доступних інструментів для розробки інформаційних систем, обґрунтований вибір програмних засобів та технологій;
– складання та виконання тестів для перевірки роботи інформаційної системи для пошуку зарядних станцій.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Дослідження аналогів

Для огляду та аналізу аналогічних додатків було розглянуто веб-системи для пошуку зарядних станцій з різних місць, таких як Google Play, AppStore, Windows Market. Для кращого порівняння було обрано декілька готових програмних продуктів, що користуються найбільшою популярністю у водіїв електромобілів.

Під час вибору аналогів основними критеріями їх оцінки були:

– кількість завантажень;

– загальний рейтинг;

– відгуки користувачів;

– реалізований функціонал.

Розглянемо більш детально декілька систем.

Перша система – PlugShare – це найбільша у світі спільнота водіїв електромобілів, реалізована як навігаційна мапа (рис.1.1).

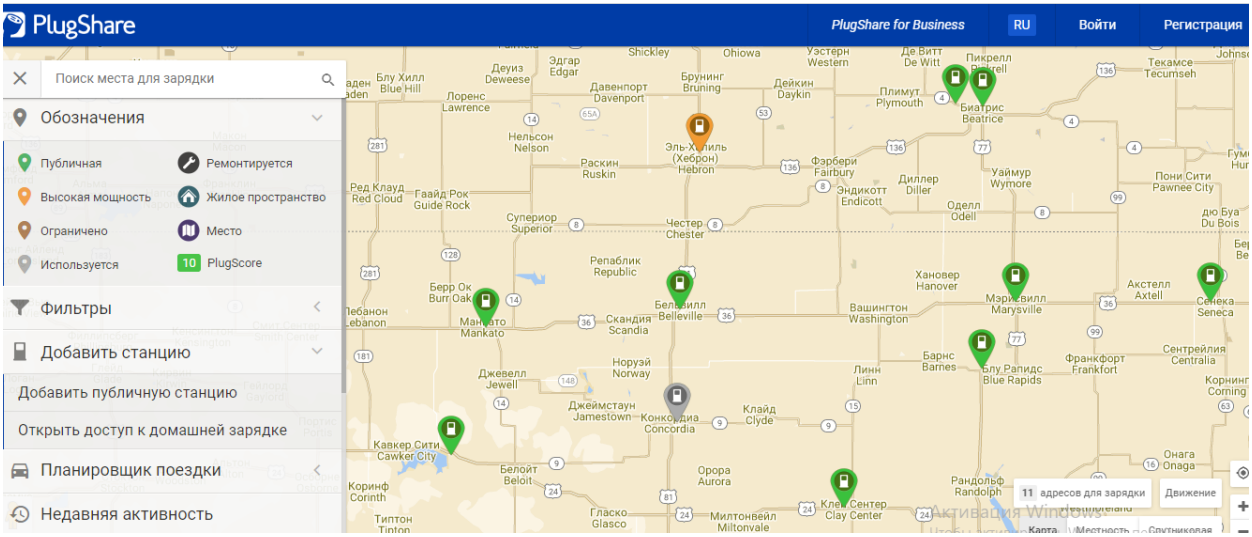

Рисунок 1.1 – Інформаційна система PlugShare

Загальна кількість завантажень – 1.500.000+.

Загальний рейтинг – 4.75.

Загальна кількість відгуків – 80.000.

У системи є наступні функції:

– пошук зарядних станцій;

– фільтрування за критеріями автомобіля;

– перевірка доступності;

– можливість поставити статус, щоб інші водії бачили, чи доступна станція;

– можливість залишити відгук;

– рейтинг станцій [2].

Наступна система – GO TO-U – мобільний додаток реалізований у вигляді навігаційної мапи, яка містить інформацію про зарядні станції для електромобілів (рис.1.2).

Рисунок 1.2 – Інформаційна система GO TO-U

Загальна кількість завантажень – 30.000+.

Загальний рейтинг – 3.9.

Загальна кількість відгуків – 500+.

У системи є наступні функції:

– отримати інформацію про статус зарядної станції;
– пошук зарядної станції;
– можливість бронювання;
– планування поїздки;
– можливість отримувати бонуси та подарунки;
– можливість оплачувати платні зарядні станції у додатку (на партнерських станціях);
– можливість отримувати оповіщення про сеанс зарядки, кількість отриманих кВт та іншу потрібну інформацію [3].

Ще одна система – 2Chargers – мапа зарядних станцій для електромобілів.

Загальна кількість завантажень – 1.000+.

Загальний рейтинг – 4.8.

Загальна кількість відгуків – 300+.

Виконує наступні функції:

– фільтрація по роз'ємам та типу станції;
– перегляд детальної інформації про станцію;
– можливість знайти станцію по карті;
– можливість залишити відгук [4].

Наведемо порівняльну характеристику (табл.1) веб-систем для пошуку зарядних станцій для електромобілів.

Із таблиці можна побачити, що не у всіх додатках є підтримка таких важливих функцій як, наприклад, бронювання станції або перевірка доступності.

Проаналізувавши аналогічні системи, можна зрозуміти чому PlugShare має таку популярність відносно таких конкурентів як 2Chargers та GO TO-U. Його функціонал направлений на полегшення процесу пошуку зарядних станцій електромобілів, зокрема сам пошук станцій та побудову маршруту до них. Але, в той самий час один з недоліків PlugShare зі слів користувачів є

проблема зайнятості зарядної станції під час їхнього приїзду до неї. Для розв'язання цієї проблеми інформаційна система, яка розроблюється, повинна містити інформацію про стан зарядної станції та можливість її бронювання на вказаний час.

Таблиця 1 – Порівняльна характеристика веб-систем для пошуку зарядних станцій для електромобілів

Існуючі рішення	Функція пошуку	Фільтрація за критеріями	Перевірка доступності станції	Планування поїздки	Можливість залишити відгук	Рейтинг	Оплата всередині програми	Бронювання зарядної станції	Підтримка операційних систем
PlugShare	✓	✓	–	–	✓	✓	–	–	iOS, Android
2Chargers	✓	✓	–	–	✓	✓	–	–	iOS
GO TO-U	✓	✓	–	✓	✓	–	✓	✓	iOS, Android
CarCharger	✓	✓	✓	✓	✓	✓	✓	✓	Windows, Android, iOS, MacOS

Відомим є факт, чим більше є навантаженим програмний функціонал, тим більше часу необхідно на виконання основних функцій, бо це спричиняє необхідність виконання більшої кількості кроків (вибір відповідного розділу меню, його команд, натискання кнопок, вибір перемикачів, заповнення форм тощо). Саме тому розробка власної інформаційної системи є актуальною.

1.2 Постановка задачі

Проаналізувавши існуючі аналоги, дослідивши їх переваги та недоліки, було сформульовано завдання дипломної роботи.

Необхідно розробити веб-застосунок для пошуку зарядних станцій для електромобілів.

У системі повинно бути два типу користувачів: власник електромобілю й адміністратор.

Функції власника електромобілю:

– додавання інформації про себе й своє авто;
– пошук зарядної станції;
– бронь місця для зарядки;
– побудова маршруту до станції;
– оплата зарядки.

Функції адміністратора системи:

– можливість зміни інформації про зарядні станції;
– можливість переглядання статистики;
– можливість модерування власників.

Система має бути інтуїтивно зрозумілою для користувача, тобто для користування системою не має бути необхідною спеціальна підготовка.

Система повинна мати чіткий, стриманий інтерфейс.

Середній час відповіді на транзакцію має бути не більший ніж 1 секунда.

Система має бути портативною, тобто вона повинна зберегти всі свої основні функції незалежно від типу пристрою, на якому вона використовується.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Визначення вимог до системи

Визначимо функціональні вимоги до системи, що розроблюється.

У системи є 2 групи користувачів: власник електромобіля та адміністратор. Власник електромобіля – основний користувач системи, тобто людина, яка бажає знайти зарядну станцію. Адміністратор – людина, яка відстежує стан станцій, може додавати нові та оновлювати інформацію про станції, які вже існують. В системі також використовується Google Maps API. Його основна задача – відповідати за відображення карти в системі та прокладати маршрут від користувача до зарядної станції.

Власник електромобіля може реєструватись, авторизуватись, додавати інформацію про власне авто, переглядати перелік станцій та бронювати їх, а також залишати коментар та оцінити станцію, який буде відображатись для всіх користувачів.

Адміністратор також матиме можливість авторизації, та реєстрації інших адміністраторів, додавати нові станції, та редагувати наявні, модерувати повідомлення користувачів.

Щоб система була конкурентоспроможною, необхідне зменшення часу на пошук зарядної станції для електромобілів. Цього можна досягти шляхом чіткого визначення системи, її швидкодії та зручності використання. Таким чином, сформовано наступні нефункціональні вимоги.

Продуктивність:

– середній час відповіді на транзакцію має бути не більший ніж 1 секунда;
– система повинна підтримувати пропускну здатність не менше 1000 одночасних користувачів.

Можливість обслуговування: система має підтримувати можливість модифікування для додавання нових функцій і покращення наявного функціонала.

Зручність використання:

– система має бути інтуїтивно зрозумілою для користувача;
– для користування системою не має бути необхідною спеціальна підготовка;
– система повинна мати чіткий, стриманий інтерфейс, щоб не відвертати увагу користувача від виконання основних функцій.

Сумісність: система має бути портативною, тобто вона повинна зберегти всі свої основні функції незалежно від типу пристрою, на якому вона використовується.

В системі присутні форми, тож обов'язковою нефункціональною вимогою є контроль можливих помилок під час їхнього заповнення, а також захист системи від людського фактора. Система повинна перевіряти всі отримані дані на коректність перед тим, як обробляти їх чи звертатись до сховища даних.

2.2 Проєктування UML-діаграм

Для формалізації функціональних вимог до системи, було створено діаграму варіантів використання (рис. 2.1), що є графічним зображенням можливих взаємодій між користувачем та системою. У ній відображаються різні сценарії використання системи та різні типи користувачів. Варіанти використання позначають колами або еліпсами, а користувачі зображуються у вигляді паличкових фігур [5].

Розглянемо кожний варіант використання більш детально, а саме визначимо успішний сценарій, його розширення, а також зобразимо діаграму послідовності.

Діаграма послідовності – це вид UML-діаграми, на якій зображено життєвий цикл об'єкта та взаємодія акторів інформаційної системи на єдиній часовій осі в рамках одного прецеденту. Їх використовують для ілюстрації виконання певних задач між користувачем та системою [6].

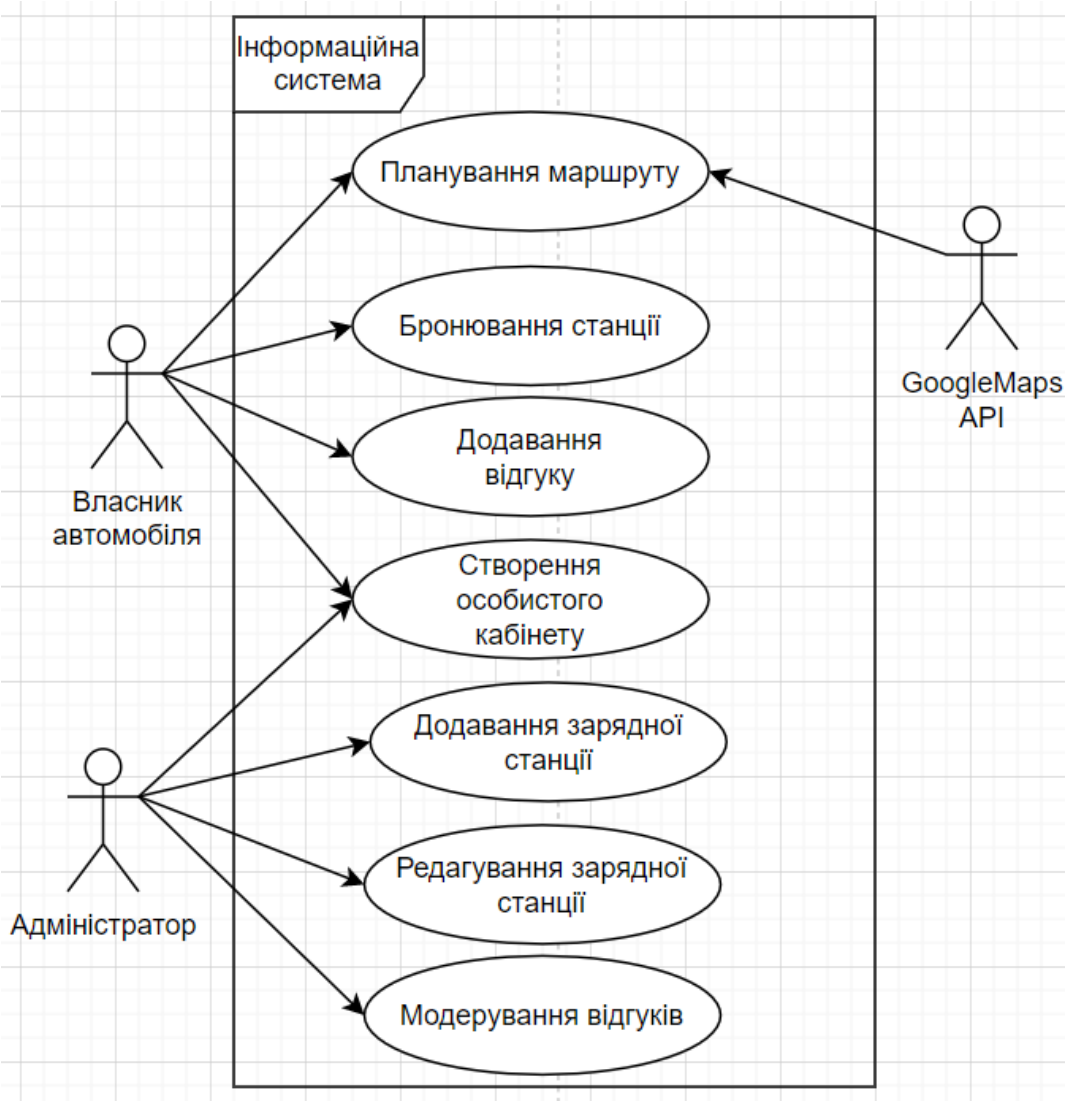

Рисунок 2.1 – Діаграма варіантів використання

В деяких сценаріях замість “Власник автомобіля” або “Адміністратор” використовується позначення “Користувач”, оскільки для даних сценаріїв не важливо хто саме це буде.

а) Сценарій — Планування маршруту (рис.2.2).

Дата створення: 01.06.2023.

Учасники й інтереси:

Власник – бажає отримати маршрут до обраної станції.

Гарант успіху – відображення карти з прокладеним маршрутом.

Мінімальна умова – користувач обрав зарядну станцію.

Передумова – Власник авторизований та відкрив карту.

1) Власник обирає зарядну станцію. Система відображає дані про станцію.
2) Власник запитує в системи маршрут до зарядної станції. Система відображає маршрут на карті.

Розширення:

а*) Не був встановлений інтернет-зв'язок:

а1) система відображає повідомлення із помилкою;

а2)власник налагоджує інтернет-зв'язок. Система відображає дані про станцію. Перехід до П2.

б*) Не був встановлений GPS зв’язок:

б1) система відображає повідомлення із помилкою;

б2) власник налагоджує GPS зв’язок. Перехід до П2.

в*) Відсутній зв’язок з Google Maps API:

в1) система відображає повідомлення з помилкою.

в2) система пропонує користувачу повторити запит на побудову маршруту. Перехід до П2

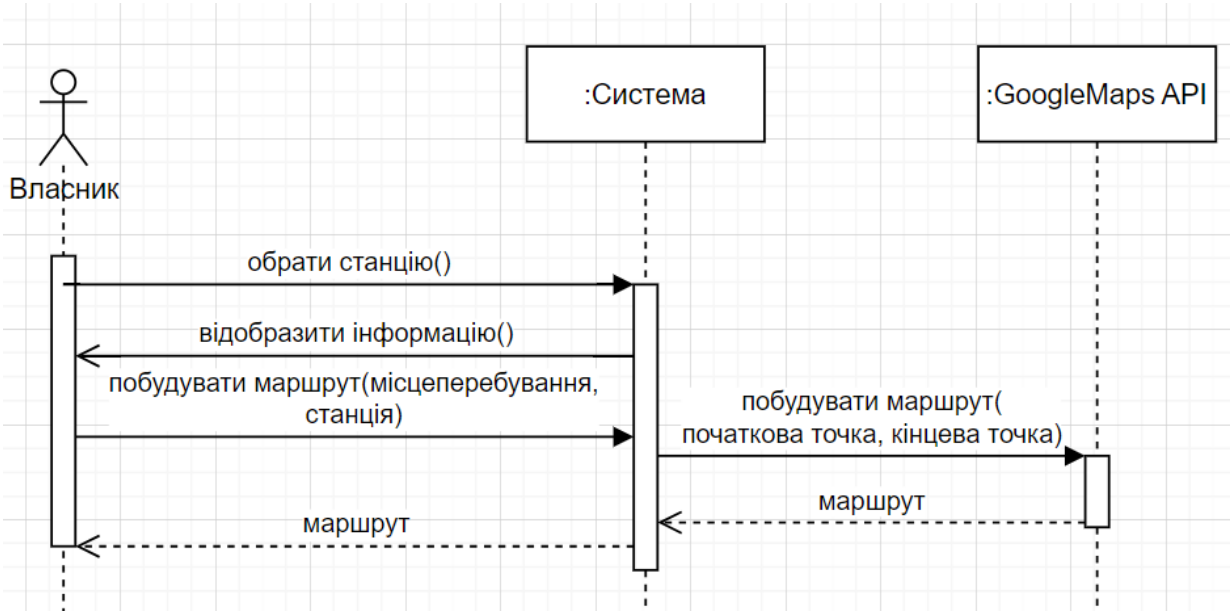

Рисунок 2.2 – Діаграма послідовності “Побудова маршруту”

b) Сценарій – Бронювання зарядної станції (рис.2.3)

Дата створення: 02.06.2023.

Учасники й інтереси:

Власник – бажає забронювати обрану станцію на вказаний час.

Гарант успіху – створення бронювання.

Мінімальна умова – користувач відкрив форму бронювання станції.

1) Власник обирає зарядну станцію. Система відображає дані про станцію.
2) Власник натискає «Забронювати». Система відображає форму для заповнення даних.
3) Власник заповнює форму. Система запитує чи потрібно зберігати дані. Користувач підтверджує.
4) Система перевіряє можливість бронювання на вказаний час. Система повертає повідомлення “Бронювання створено”.

Розширення:

а*) Не був встановлений інтернет-зв'язок:

а1) система відображає повідомлення із помилкою;

а2)власник налагоджує інтернет-зв'язок. Система відображає дані про станцію. Перехід до П2.

б*) Була допущена помилка під час заповнення форми:

б1) система відображає повідомлення із помилкою.

б2)власник виправляє помилку. Система запитує чи потрібно зберігати дані. Користувач підтверджує.

в*) Станція заброньована на даний час:

в1) система відображає повідомлення з помилкою.

в2) власник обирає інший час. Перехід до П3.

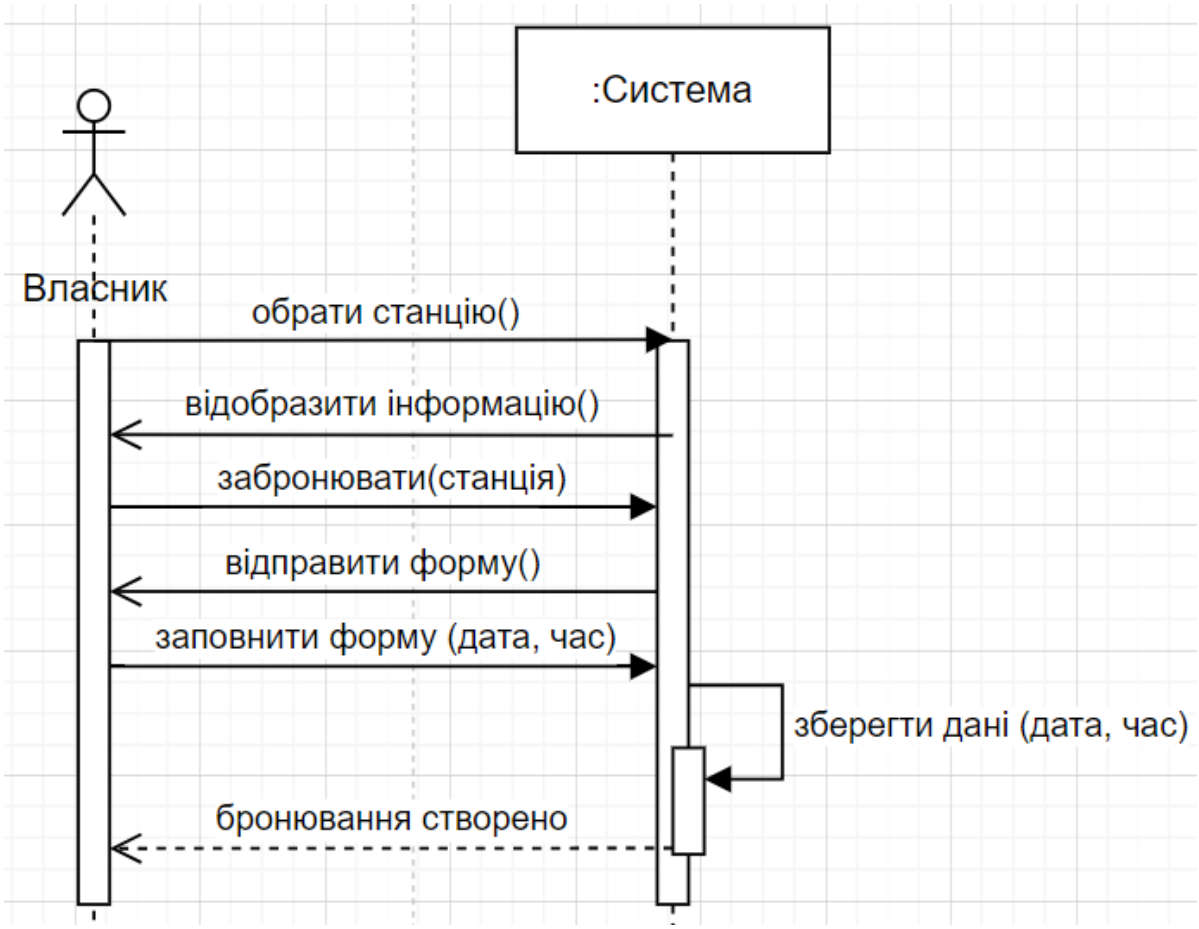

Рисунок. 2.3 – Діаграма послідовності “Бронювання станції”

с) Сценарій – Додавання відгуку (рис.2.4).

Дата створення: 03.06.2023.

Учасники й інтереси:

Власник – бажає поділитись з іншими користувачами досвідом користування станцією.

Гарант успіху – створений відгук у профілі станції.

Мінімальна умова – Власник перейшов відкрив історію зарядок.

Передумова – Власник автомобіля здійснив бронювання зарядної станції.

1) Власник відкриває історію зарядок. Система відображає список станцій.

2) Власник обирає необхідну станцію. Система відображає сторінку станції.

3) Власник додає новий коментар. Система зберігає дані.

Розширення:

а*) Не був встановлений інтернет-зв'язок:

а1) система відображає повідомлення із помилкою;

а2)власник налагоджує інтернет-зв'язок. Система відображає станції. Перехід до П2.

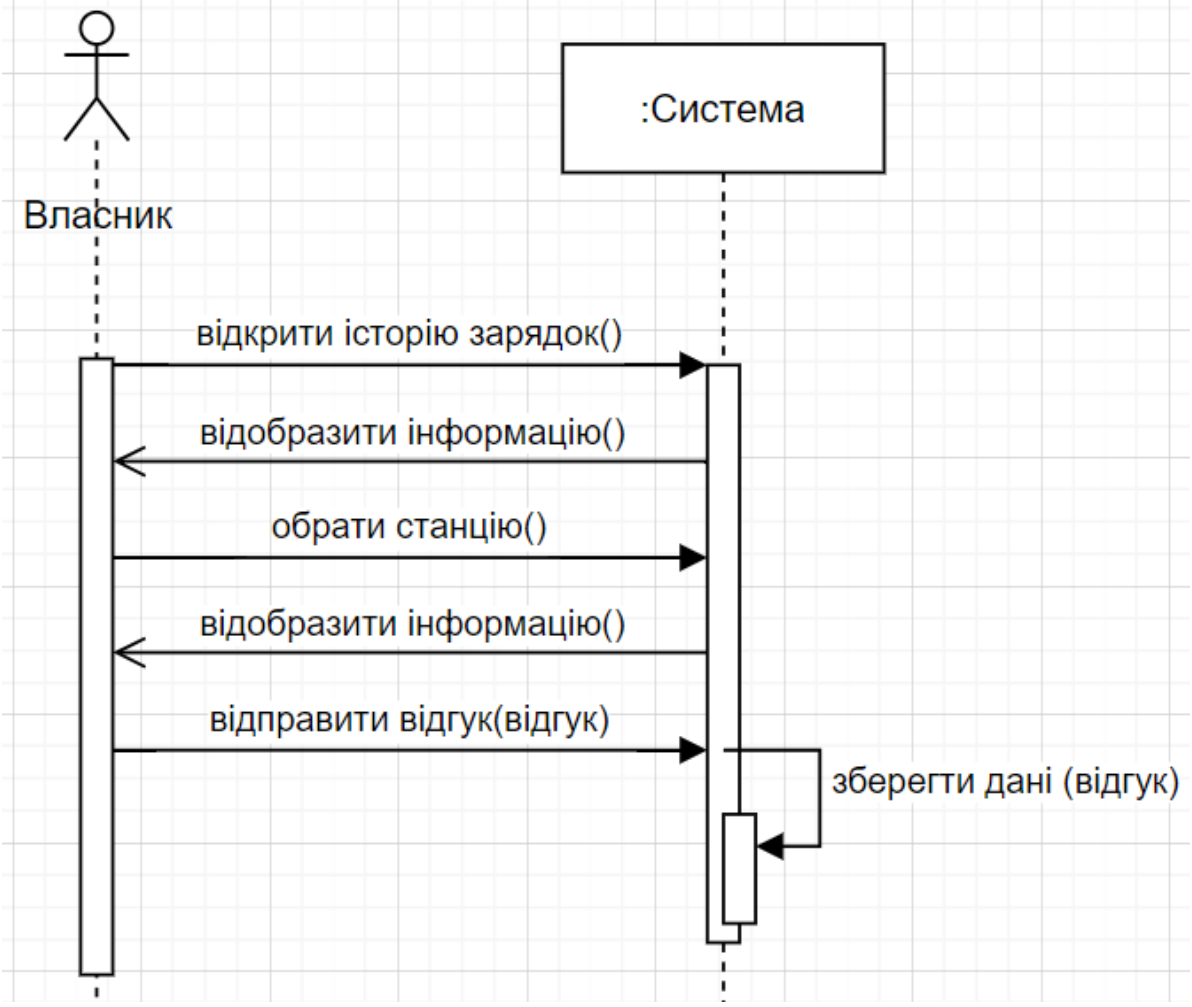

Рисунок 2.4 – Діаграма послідовності “Додавання відгуку”

d) Сценарій – Створення особистого кабінету (рис.2.5).

Дата створення: 04.06.2023

Учасники й інтереси:

Користувач – бажає створити особистий кабінет.

Гарант успіху – створення особистого кабінету.

Мінімальна умова – користувач відкрив форму створення особистого кабінету.

1) Користувач відкриває сторінку «Особистий кабінет». Система відображає форму для заповнення даних.

2)Користувач заповнює форму. Користувач підтверджує заповнення форми. Система зберігає інформацію.

Розширення:

а*) Не був встановлений інтернет-зв'язок:

а1) система відображає повідомлення із помилкою;

а2) користувач налагоджує інтернет-зв'язок. Система відображає форму для заповнення даних. Перехід до П2.

б*) Була допущена помилка під час заповнення форми:

б1) система відображає повідомлення із помилкою;

б2) користувач виправляє помилку. Система запитує чи потрібно зберегти інформацію. Користувач підтверджує.

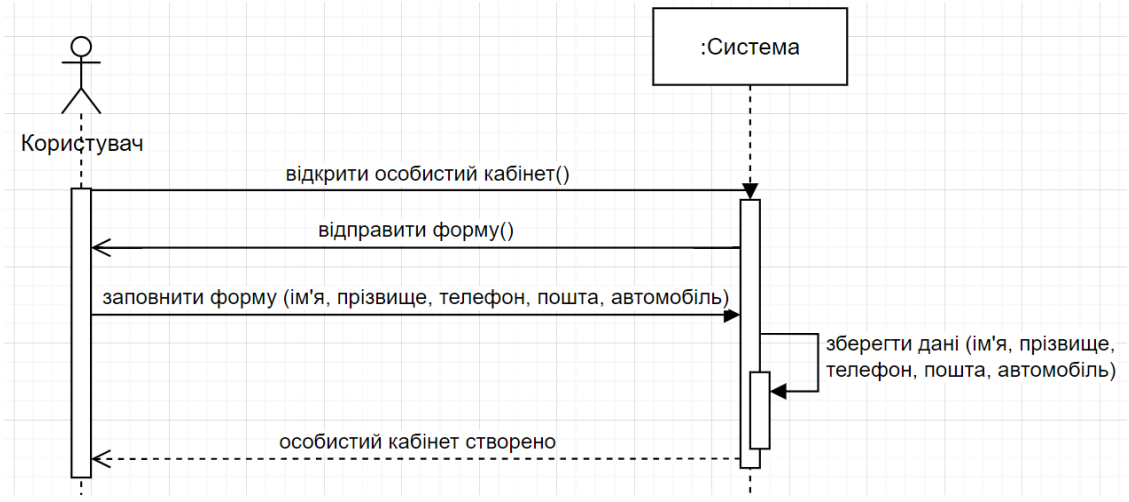

Рисунок 2.5 – Діаграма послідовності “Створення особистого кабінету”

2.3 Проєктування архітектури системи

Визначимо архітектурне рішення, яке найкраще застосувати для нашої системи на основі визначених вимог до системи. В результаті аналізу сценаріїв використання складається загальна картина взаємодії користувачів з системою. Так, користувач відкриває застосунок та спілкується з системою через GUI. Наприклад, користувач хоче забронювати станцію. Для цього він обирає станцію, дату та час на який хоче створити бронювання. Після цього ці дані повинні якось опрацюватись, щоб інший користувач не зміг дублювати бронювання. Також, станція може вийти з ладу й адміністратор через GUI має оновити інформацію про станцію. Тобто інформація повинна десь зберігатись, щоб у користувачів завжди був доступ до неї й вони могли спостерігати її зміну.

Таким чином, можна дійти до висновку, що система повинна мати як мінімум три модулі, відповідальні за її повноцінну роботу:

– модуль – взаємодія з користувачем;
– модуль – робота бізнес-логіки;
– модуль – збереження інформації.

Головними нефункціональними вимогами до системи є надійність і продуктивність. У системі кожну хвилину може відбуватися бронювання та побудова маршруту до зарядної станції, тобто навантаження може бути доволі великим. Також вона має бути доступною не менше 90% робочого часу. Отже, необхідно, щоб система була масштабованою з можливістю бути доступною для багатьох людей.

На основі зазначених вище тверджень було вирішено обрати архітектурний шаблон "Клієнт-Сервер”. Головною причиною вибору даного шаблону є вимоги до швидкодії та портативності системи. Клієнт є доволі мало потужним і дозволяє адаптувати систему під будь-яке середовище та зробити її доступною з будь-якого місця де є інтернет. Вся бізнес-логіка відбувається на сервері, який повинен бути доволі потужним. Також такий

шаблон ідеально підходить під сформовані вище три модулі, з яких складається система.

Таким чином, визначимо в системі місце кожного модуля:

– модуль, який відповідає за взаємодію з користувачем – клієнт;

– модуль, який відповідає за бізнес-логіку – сервер;

– модуль, який відповідає за збереження інформації – сховище даних.

Рішення зберігати дані не на сервері прийняте з багатьох причин. Так, при раптових проблемах з сервером, які можуть спричинити його відключення чи перезавантаження, дані не постраждають, бо винесені в окремий модуль. Додатковою перевагою даного шаблону є відсутність дублювання коду, бо основний код зберігається на сервері, а клієнт відповідає лише за взаємодію користувача з системою та відображення результату взаємодії. Схему даного шаблону можна побачити на рис. 2.6.

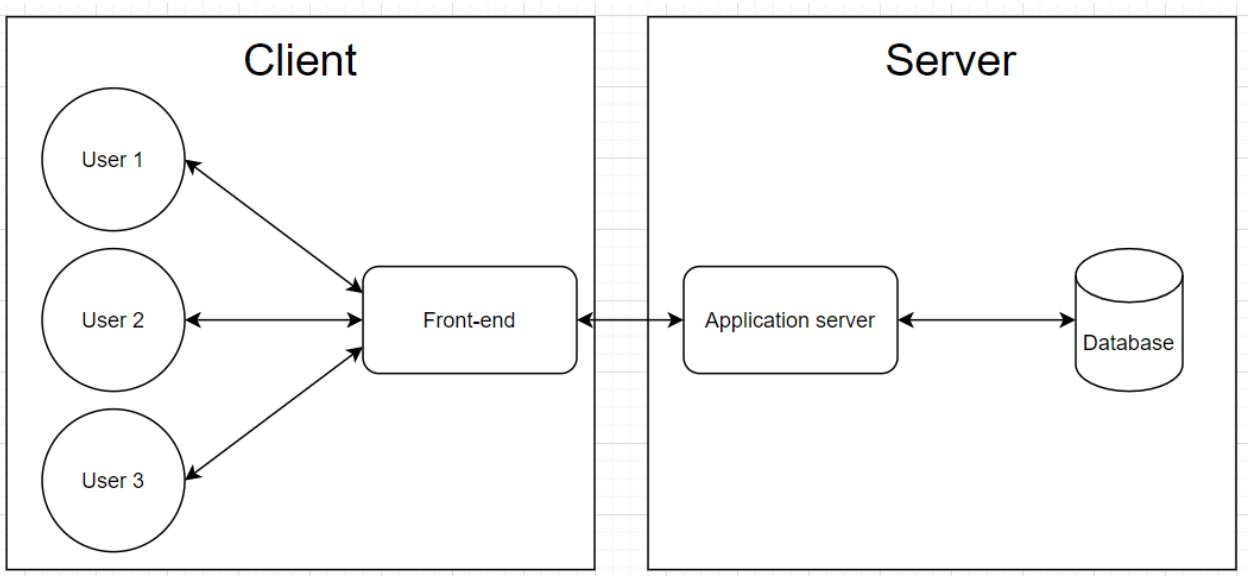

Рисунок 2.6 – Загальна архітектура веб-застосунку

Обрана архітектура системи поділяє її на дві підсистеми: клієнт та сервер. В основі архітектури сервера використовуються деякі GRASP патерни: Contoller, Service та Repository.

Contoller – це такий клас, який відповідає за взаємодію користувача з системою, в цьому випадку – взаємодію клієнта з сервером. Його задача – отримати від користувача подію й запустити відповідні бізнес-процеси. В ідеальному випадку Controller не повинен нічого знати про користувача, що дозволяє використовувати його для різних користувачів.

Щоб мати можливість повторно використовувати логіку між компонентами є сенс винести її в окремий рівень – Service. Service – рівень, який відповідає за бізнес-логіку. Якщо Controller'у необхідно отримати, опрацювати або відправити якісь дані, то він робить це через Service. Так само якщо декільком контролерам знадобилась однакова логіка, вони працюють із Service, але сам рівень Service не повинен нічого знати ані про клієнта, ані про Controller.

Для виконання бізнес-логіки може знадобитись інформація про стан системи та її зміна, тобто необхідна взаємодія зі сховищем даних. Винесемо таку взаємодію в окремий рівень – Repository. Завдяки цьому достатньо один раз реалізувати запит даних й будь-який сервіс зможе повторно використати цей запит.

Система повинна розуміти структуру даних з якими вона працює. Класи, що описують таку інформацію, відокремлені в окрему групу – Models. Моделі в залежності від типу використання поділяються на декілька типів. Entities – модель, що відповідає за роботу зі сховищем даних, повністю відтворює структуру сутності в сховищі даних. DTO (Data transfer object) – моделі, які служать для перенесення даних між ярусами системи. Одним з призначень DTO є перешкоджання отримання доступу до чутливої інформації. Наприклад, в системі існує сутність студент, в якій міститься ПІБ студента та його номер телефону. Клієнт бажає отримати лише прізвище студента, тож передавати йому додаткову інформацію про студента не є безпечним. Для розв'язання цієї проблеми й існують DTO, які передають лише мінімум необхідної інформації між рівнями, необхідної для повноцінного функціонування системи [7].

Отже, на основі зазначених вище тез було вирішено також обрати рівневу архітектуру для сервера. Загальна діаграма рівнів сервісу зображена на рис. 2.7.

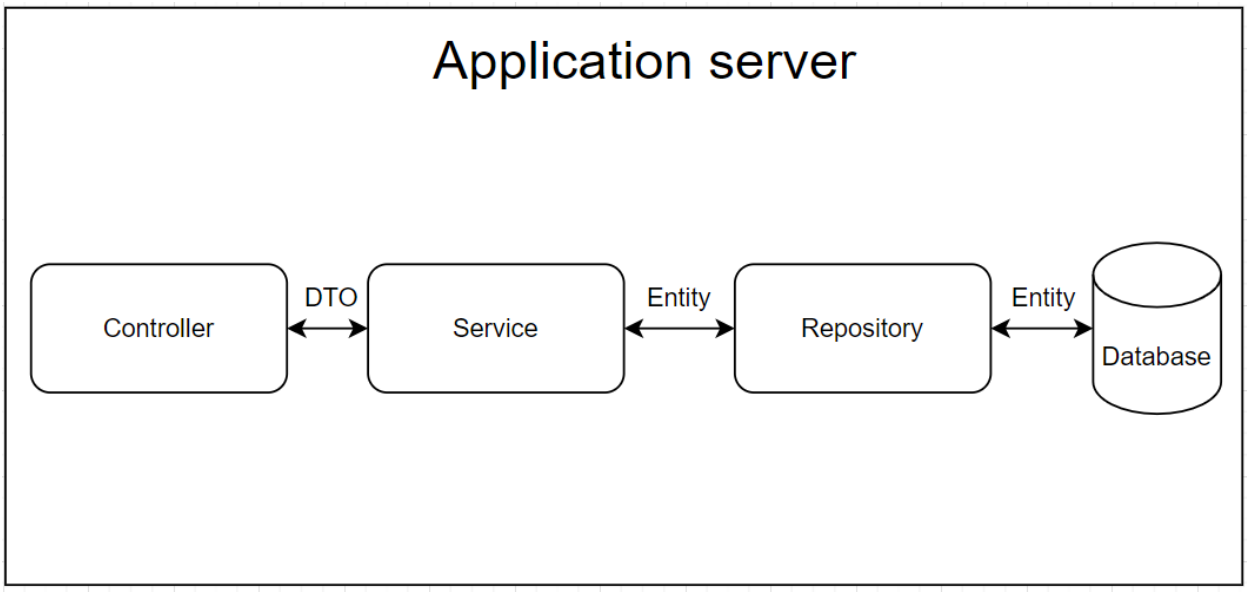

Рисунок 2.7 – Архітектура сервера

2.4 Проєктування сховища даних

Для повноцінної роботи інформаційної системи необхідно зберігати й працювати з великою кількістю даних, які зручно представити у вигляді реляційної моделі.

Таке представлення дозволяє легко виділити сутності, наявні в предметній області, та їхні атрибути. В реляційній моделі сутності представляються в вигляді таблиці бази даних, атрибути – поля таблиць, відповідно. Кожна таблиця містить атрибут первинний ключ, що ідентифікує кожний запис таблиці. Це значення повинне бути унікальним в межах таблиці. Для встановлення та забезпечення зв’язку між даними у двох таблицях для контролю даних, які можуть зберігатися в таблиці існує додатковий атрибут – зовнішній ключ. Якщо поле таблиці не обов’язково

повинно мати значення, то воно позначається як N – nullable. Якщо значення поля повинне бути унікальним, то воно позначається як U – unique [8].

В якості таблиць будуть виступати наступні сутності:

– користувач;

– роль;

– станція;

– бронювання;

– відгук.

Всі таблиці, які існують в даній БД, мають відношення один-до-багатьох, тобто нормалізовані, що дозволяє виключити аномалії, підвищити зручність роботи з даними та продуктивність.

На рис. 2.8 представлена реляційна схема бази даних для інформаційної системи, що розробляється в рамках даної кваліфікаційної роботи.

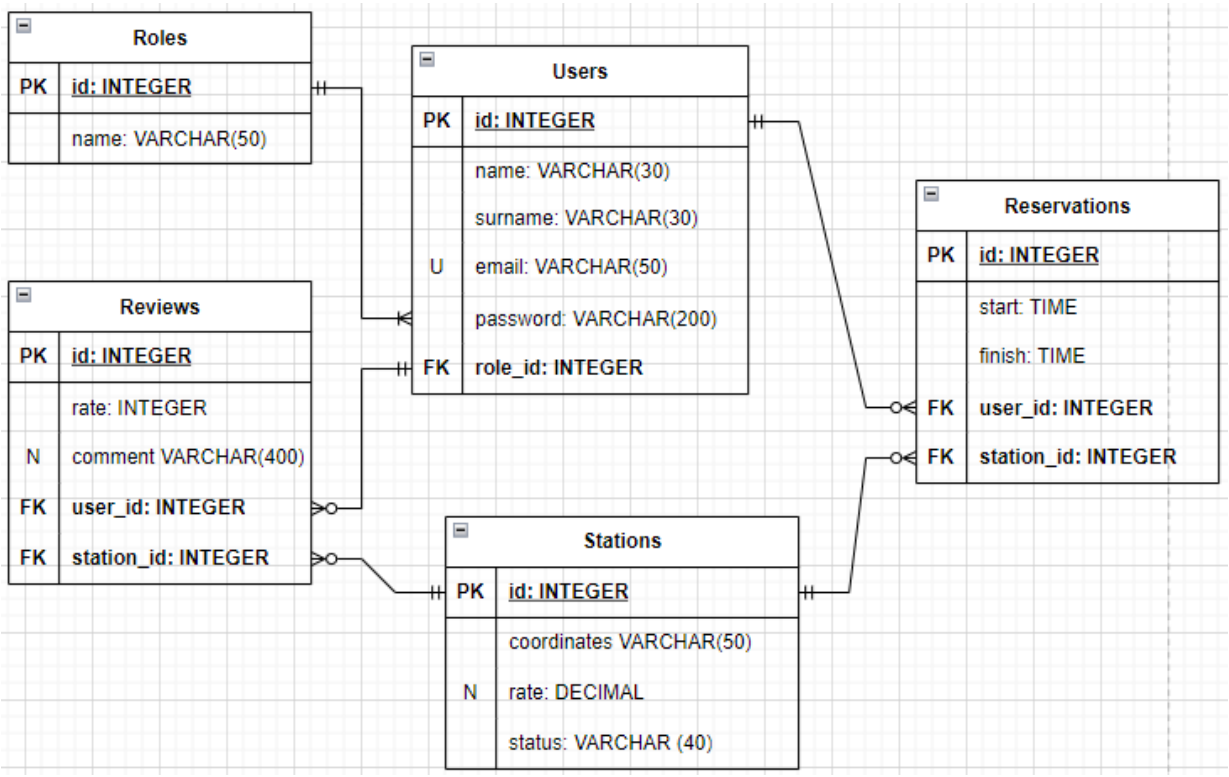

Рисунок 2.8 – Реляційна схема сховища даних системи

Таблиця Users зберігає дані про користувачів системи (табл.2.1). Система може мати будь-яку необхідну кількість користувачів. Кожен користувач може мати будь-яку кількість бронювань й відгуків, але для кожного бронювання й відгуку передбачений лише один користувач. Ключовим полем таблиці є ідентифікатор користувача. Для зберігання даних адміністратора окремої таблиці не передбачено, оскільки для цього існує окрема сутність роль, яка відповідальна за визначення типу користувача.

Таблиця 2.1 – Структура таблиці Users

Атрибут	Тип	Ключ	Опис атрибута
id	INTEGER	Primary key	Ідентифікатор користувача
name	VARCHAR(30)		Ім’я користувача
surname	VARCHAR(30)		Прізвище
email	VARCHAR(50)		користувача
password	VARCHAR(200)		Пароль користувача.
role_id	INTEGER	Foreign key	Ідентифікатор ролі

Таблиця Roles (табл.2.2) зберігає інформацію про ролі, наявні в системі. Вона містить первинний ключ – ідентифікатор ролі. В одного користувача може бути лише одна роль, але одна роль може бути в багатьох користувачів.

Таблиця 2.2 – Структура таблиці Roles

Атрибут	Тип	Ключ	Опис атрибута
id	INTEGER	Primary key	Ідентифікатор користувача
name	VARCHAR(50)		Назва ролі

Таблиця Stations зберігає інформацію про станції (табл.2.3). Вона містить первинний ключ – ідентифікатор станції Кожна станція може мати

багато бронювань та відгуків, але в кожному бронюванні й відгуку може бути присутня лише одна станція. Таблиця містить одне nullable поле: рейтинг станції може бути порожнім, оскільки може не бути жодного відгуку для даної станції

Таблиця 2.3 – Структура таблиці Stations

Атрибут	Тип	Ключ	Опис атрибута
id	INTEGER	Primary key	Ідентифікатор станції
coordinates	VARCHAR(50)		Географічне місце станції
rate	DECIMAL		Середня оцінка станції на основі оцінки в відгуках
status	VARCHAR(40)		Статус станції. Можливі статуси: справна, зайнята та не функціонує

Таблиця Reservations містить інформацію про бронювання зарядних станцій (табл.2.4). Користувач може створити багато бронювань різних станцій на різний час. Але за в одному бронюванні можна забронювати лише одну станцію.

Таблиця 2.4 – Структура таблиці Reservations

Атрибут	Тип	Ключ	Опис атрибута
id	INTEGER	Primary key	Ідентифікатор бронювання
start	DATE		Дата та час початку бронювання
finish	DATE		Дата та час закінчення бронювання
user_id	INTEGER		Ідентифікатор користувача
station_id	INTEGER		Ідентифікатор станції

Таблиця Reviews містить інформацію про відгуки (табл.2.5). Після використання зарядної станції користувач може оцінити роботу станції та залишити коментар. Таблиця містить первинний ключ – ідентифікатор відгуку. Атрибут коментар може бути порожнім, оскільки необов’язково залишати коментар.

Таблиця 2.5 – Структура таблиці Reviews

Атрибут	Тип	Ключ	Опис атрибута
id	INTEGER	Primary key	Ідентифікатор бронювання
rate	INTEGER		Оцінка роботи станції від одного до п’яти
comment	VARCHAR(400)		Коментар. Зауваження до роботи станції
user_id	INTEGER		Ідентифікатор користувача
station_id	INTEGER		Ідентифікатор станції

2.5 Проєктування програмних класів

Таким чином, в системі існують наступні оперуючі сутності:

– User – сутність, яка відображає користувача системи;
– Role – сутність, яка відображає роль користувача. Можливі ролі: власник автомобіля та адміністратор;
– Station – сутність, яка відображає станцію;
– Reservation – сутність, яка відображає бронювання;
– Rate - сутність, яка відображає відгук.

Відповідно до першого SOLID принципу єдиного обов’язку (Self responsibility) всі ці сутності повинні мати власне відображення на кожному рівні системи [9].

Побудуємо програмні класи для кожної сутності.

Розглянемо більш детально діаграму програмних класів для сутності користувач (рис.2.9). Клас User є відображенням відповідної таблиці в базі даних, тобто цей клас має ті ж самі атрибути, які зазначені в базі даних. Такими атрибутами є:

– id – унікальний ідентифікатор користувача;
– name – ім’я користувача;
– surname – прізвище користувача;
– email – електронна пошта користувача, яка додатково є логіном;
– password – пароль користувача.

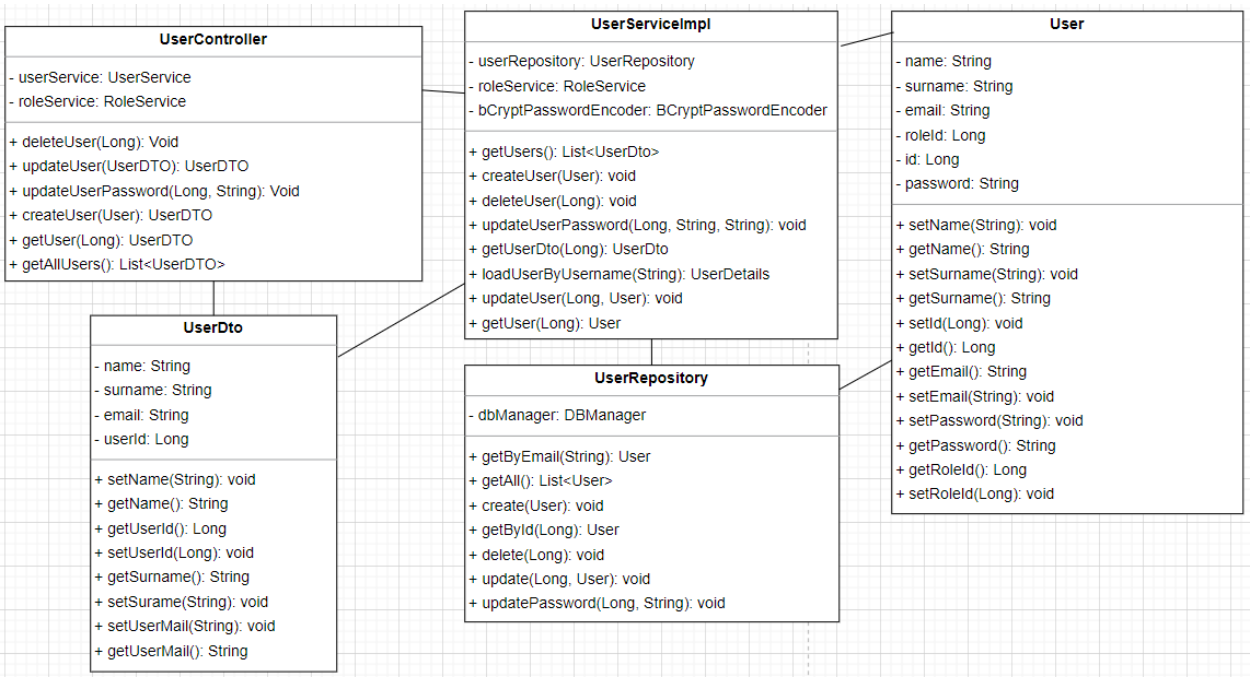

Рисунок 2.9 – Діаграма програмних класів сутності користувач

Також клас User має атрибут roleId, який необхідний для створення зв’язку користувача з його роллю. Завдяки цьому атрибуту можемо дізнатись роль користувача. Клас містить так звані методи гетери та сетери. Основна задача цих методів захистити дані всередині об’єкту. Гетери відповідають за отримання значень атрибутів об'єкта, сетери – зміна значень атрибутів.

Клас UserDto є реалізацію вище зазначеного патерну DTO. Він містить основну інформацію про користувача, а саме:

– id – унікальний ідентифікатор користувача;
– name – ім’я користувача;
– surname – прізвище користувача;
– email – електронна пошта користувача, яка додатково є логіном.

Так само як і User клас, клас UserDto містить гетери й сетери для всіх атрибутів. Ці два класи необхідні для зберігання та отримання стану об’єктів користувача.

Клас UserRepository реалізує патерн Repository. Відповідно до своєї мети він містить один атрибут DbManager – спеціальний інтерфейс, за допомогою якого відбуватиметься спілкування зі сховищем даних. Клас містить наступні методи:

– отримати користувача за його емейлом – getUserByEmail(String). В системі присутнє обмеження на унікальність емейлу, тож за ним можна отримати користувача;
– отримання усіх користувачів системи – getAll();
– створення користувача – create(User);
– отримання користувача за його унікальним ідентифікатором – getById(Long);
– видалити користувача з системи – delete (Long). Пошук необхідного для видалення користувача відбуватиметься за його унікальним ідентифікатором, який є параметром функції;
– оновити інформацію про користувача – update (Long, user);
– оновити пароль користувача – updatePassword (Long, String).

Клас UserService реалізує патерн Service. Він містить в собі усю бізнес-логіку, пов’язану з сутністю користувач. Містить в собі наступні атрибути:

– userRepository – згідно з обраною архітектурою сервіс може спілкуватись зі сховищем даних лише через прошарок Repository.

– roleService – сервіс, який містить бізнес-логіку, пов’язану з сутністю роль. Через те, що існує зв’язок між даними сутностями, то для повноцінної реалізації бізнес-логіки, пов’язаної з користувачем, необхідно мати доступ до бізнес-логіки, пов’язаної з роллю.
– bCryptPasswordEncoder – енкодер, за допомогою якого відбуватиметься шифрування пароля користувача.

Клас містить наступні методи:

– отримання усіх користувачів – getUsers();
– створити користувача – createUser(UserDto);
– видалити користувача – deleteUser(Long);
– оновити пароль користувача – updateUserPassword(Long, String);
– отримання інформації про користувача – getUserDto(Long);
– отримання користувача за емейлом – loadUserByEmail(String);
– оновити інформацію про користувача – updateUser(Long, User);
– отримати усього користувача – getUser(Long).

Клас UserController реалізує патерн Controller. В нього лише один атрибут UserService, необхідний для виклику виконання необхідної бізнес-логіки. Клас має наступні функції:

– видалити користувача – deleteUser(Long);
– оновити інформацію про користувача – updateUser(UserDto);
– оновити пароль користувача – updateUserPassword(Long, String);
– отримати користувача за ідентифікатором – getUser(Long);
– отримати усіх користувачів – getAllUsers().

Розберемо програмні класи для сутності роль (рис.2.10).

Клас роль є відображенням відповідної таблиці в сховищі даних.

Клас має наступні атрибути, для кожного з яких реалізовані гетери й сетери:

– унікальний ідентифікатор – id;
– назва ролі – name.

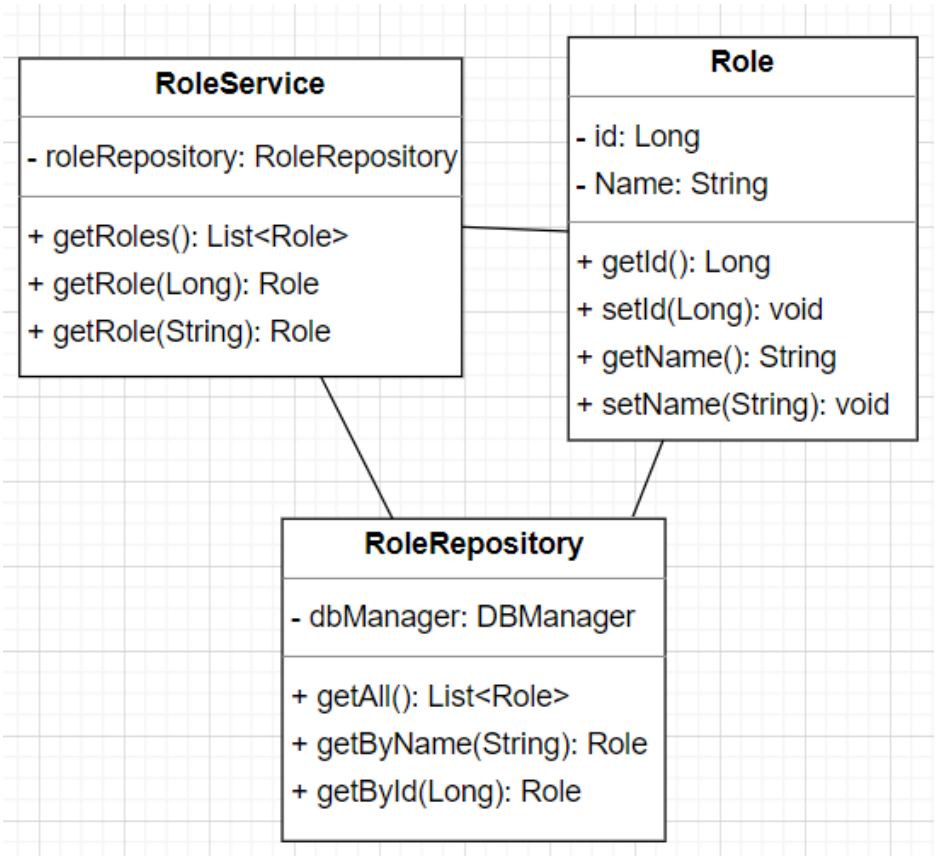

Рисунок 2.10 – Діаграма програмних класів сутності роль

Клас RoleRepository відповідає за роботу з даними та роботою зі сховищем даних. Для цього він має атрибут DbManager. Клас RoleRepository реалізує наступні методи:

– отримати список ролей – getAll();
– отримати роль за назвою – getByName(String);
– отримати роль за унікальним ідентифікатором – getById(Long).

Клас RoleService відповідає за бізнес-логіку пов’язану з сутністю роль. Щоб реалізувати своє призначення він повинен мати можливість працювати з даними, які зберігаються в сховищі даних. Для цього клас містить атрибут RoleRepository. Клас реалізує наступні методи:

– отримати список ролей – getRoles();
– отримати роль за назвою – getRole(String);
– отримати роль за унікальним ідентифікатором – getRole(Long).

Розглянемо програмні класи для сутності відгук (рис.2.11).

Клас Rate є відображення відповідної таблиці в базі даних. Цей клас описує структуру сутності відгук. Має наступні атрибути, для кожного з яких реалізовані гетери й сетери:

– унікальний ідентифікатор – id;
– текстовий коментар - comment;
– обраний рейтинг – rate;
– унікальний ідентифікатор користувача – userId. Оскільки відгук залишає користувач, тобто ці сутності мають зв’язок, необхідно зберігати ідентифікатор користувача;
– унікальний ідентифікатор станції – stationId. Відгук залишають про якусь станцію. Тож, щоб підтримати даний зв'язок необхідна присутність даного ідентифікатору.

Клас RateDto необхідний для передачі даних між рівнями системи. Має наступні атрибути, для кожного з яких реалізовані гетери та сетери:

– унікальний ідентифікатор – rateId;
– ім’я користувача, який залишив відгук – fullName;
– обрана оцінка – rate;
– текстовий коментар – comment;
– унікальний ідентифікатор користувача – userId.

Клас RateRepository – відповідає за спілкування зі сховищем даних й роботу з даними, пов’язаними з сутністю відгук. Для виконання своїх обов’язків він містить атрибут DbManager.

Клас RateRepository реалізує наступні методи:

– отримати список відгуків для станції – getByStationId(Long). Метод необхідний для пошуку всіх відгуків, залишених для конкретної станції;
– Отримати усі відгуки – getAll();
– оновити відгук – update(Rate);
– видалити відгук – delete(id);

– отримати відгук за унікальним ідентифікатором – getById(Long);

– створити відгук – create(Rate).

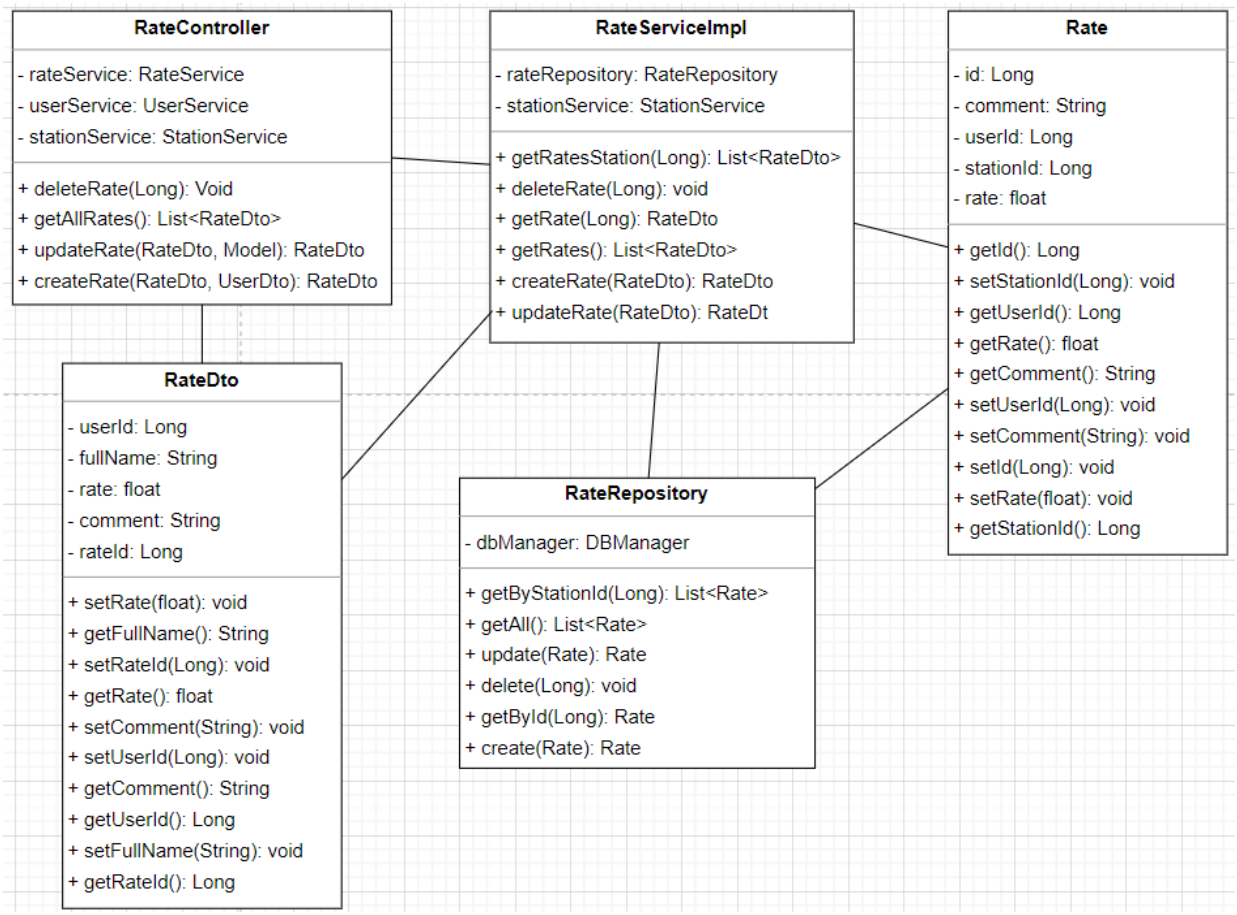

Рисунок 2.11 – Діаграма програмних класів сутності відгук

Клас RateService реалізує усю необхідну бізнес-логіку, пов’язану з сутністю відгук.

Для виконання своїх обов’язків він містить наступні атрибути:

– RateRepository необхідний для роботи зі сховищем даних;

– StationService необхідний для виклику підрахунку рейтингу станції після додавання або редагування відгуку.

Клас RateController є зовнішньою точкою спілкування з системою, тобто є реалізацією патерну Controller. Він має наступні атрибути:

– UserService – необхідний для отримання списку користувачів;

– StationService – необхідний для отримання списку станцій;

– RateService – необхідний для виклику необхідної бізнес-логіки.

Для виклику бізнес-логіки клас RateController реалізовує наступні методи:

– видалити рейтинг – deleteRate(Long);

– отримати всі відгуки – getAllRates();

– оновити відгук – updateRate(RateDto);

– створити відгук – createRate(RateDto, UserDto).

Для виконання своїх обов’язків клас RateService реалізує наступні методи:

– отримання відгуків для конкретної станції за її унікальним ідентифікатором – getRatesStation(Long);

– видалити відгук за його унікальним ідентифікатором – delete(Long);

– отримати відгук за його унікальним ідентифікатором – getRate(Long);

– отримання усіх відгуків – getRates();

– додавання нового відгуку – createRate(RateDto);

– оновлення відгуку – updateRate(RateDto).

Розглянемо програмні класи для сутності станція.

Клас Station є відображенням відповідної таблиці в сховищі даних, тобто даний клас повністю описує сутність станції в системі. Даний клас має наступні атрибути, для кожного з яких реалізовані гетери та сетери:

– унікальний ідентифікатор – id;

– координати станції – coordinates;

– загальний рейтинг станції – rate;

– стан станції – status.

Клас StationDto необхідний для трансферу даних станції між ярусами системи. Даний клас має наступні атрибути, для кожного з яких реалізовані гетери та сетери:

– унікальний ідентифікатор – id;

– координати станції – coordinates;

– загальний рейтинг станції – rate;

– стан станції – status.

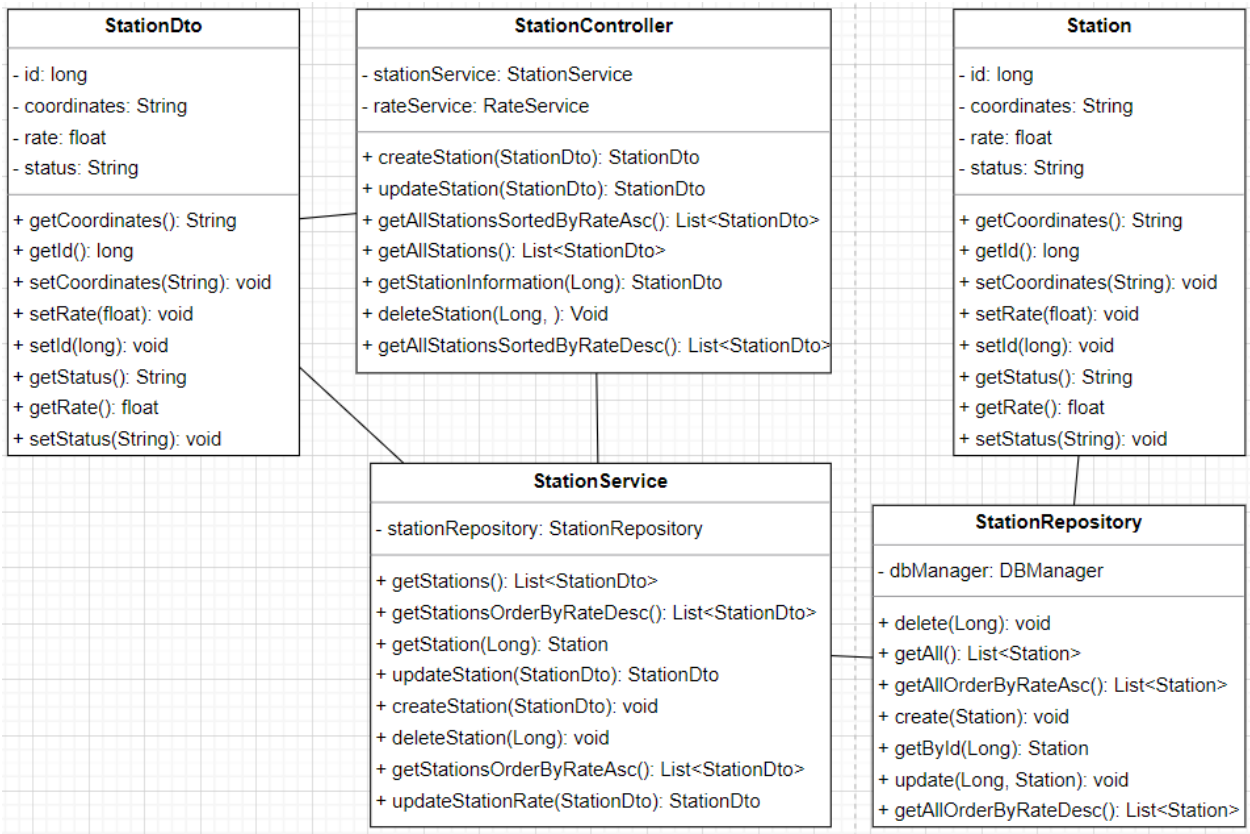

Рисунок 2.12 – Діаграма програмних класів сутності станція

Клас StationRepository має обов’язки роботи зі сховищем даних, а саме безпосередньо з сутністю станція. Для виконання своїх обов'язків має відповідний атрибут DbManager. Сам клас реалізує наступні методи:

– видалити станцію за унікальним ідентифікатором – delete(Long);

– отримати всі станцій – getAll();

– отримати всі станції, відсортовані за рейтингом за зростанням – getAllOrderByRateAsc();

– отримати всі станції, відсортовані за рейтингом за зменшенням – getAllOrderByRateDesc();

– створити станцію – create(Station);
– отримати станцію за її унікальним ідентифікатором – getById(Long);
– оновити інформацію про станцію – update(Station).

Клас StationService реалізовую усю бізнес-логіку в системі, пов’язану з сутністю станції. В ньому присутній один атрибут StationRepository, який необхідний для роботи з даними. Клас реалізує наступні методи:

– отримати всі станції – getStations();
– отримати всі станції, відсортовані за рейтингом за зростанням – getAllOrderByRateAsc();
– отримати всі станції, відсортовані за рейтингом за зменшенням – getAllOrderByRateDesc();
– отримати станцію за її унікальним ідентифікатором – getStation(Long);
– оновити інформацію про станцію – updateStation(StationDto);
– створити станцію – createStation(StationDto);
– видалити станцію за її унікальним ідентифікатором – deleteStation(Long);
– перерахунок загального рейтингу станції після додавання або оновлення відгуку – updateStationRate(StationDto).

Клас StationController є точкою виклику необхідної бізнес-логіки, пов’язаної зі станцією, тобто є реалізацією патерну Contoller. Для цього він має атрибут StationService. Клас реалізує наступні методи:

– створити станцію – crateStation(StationDto);
– оновити інформацію про станцію updateStation(StationDto);
– отримати всі станції, відсортовані за рейтингом за зростанням – getAllStationsOrderByRateAsc();
– отримати всі станції, відсортовані за рейтингом за зменшенням – getAllStationsOrderByRateDesc();
– отримати всі станції – getAllStation();

– отримати інформацію про станцію за її унікальним ідентифікатором getStationInformation(Long);

– видалити станцію – deleteStation(StationDto).

Розглянемо програмні класи, пов’язані з сутністю бронювання.

Клас Reservation (рис.2.13) є відображенням сутності бронювання в сховищі даних, тобто він повністю повторює структуру й має наступні атрибути, для кожного з яких реалізовані гетери та сетери:

– унікальний ідентифікатор – id;

– дата та час початку бронювання – start;

– дата та час закінчення бронювання – finish;

– унікальний ідентифікатор користувача – userId.

– унікальний ідентифікатор станції – stationId.

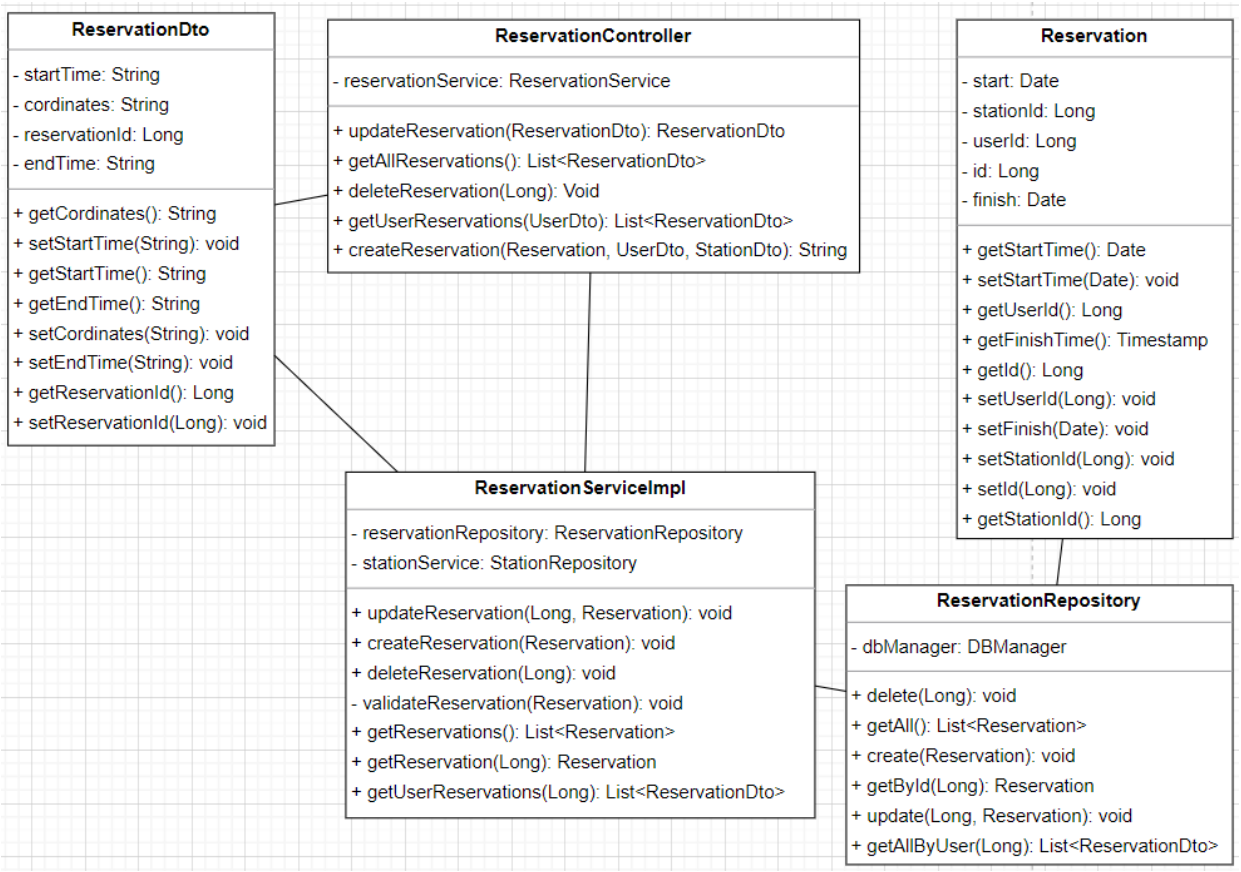

Рисунок 2.13 – Діаграма програмних класів сутності станція

Клас ReservationDto необхідний для трансферу даних між ярусами системи. Клас має наступні атрибути, для кожного з яких реалізовані гетери та сетери:

– унікальний ідентифікатор бронювання – reservationId;
– координати станції, яка була заброньована – coordinates. Необхідно для того, щоб користувач міг легко зі сторінки бронювання побудувати маршрут до обраної станції;
– дата та час початку бронювання – startTime;
– дата та час закінчення бронювання – finishTime.

Клас ReservationRepository є реалізацією патерну Repository для сутності бронювання. Тобто призначенням цього класу є спілкування зі сховищем даних та передача або отримання даних пов’язаних з даною сутністю. Для цього він містить один атрибут DbManager. Клас реалізує наступні методи:

– видалити бронювання за його унікальним ідентифікатором – delete(Long);
– отримати лист усіх бронювань – getAll();
– створити бронювання – create(Reservation);
– отримати бронювання за його унікальним ідентифікатором – getById(Long);
– оновити інформацію про бронювання – update(Reservation);
– отримати усі бронювання для конкретного користувача за його унікальним ідентифікатором – getAllByUser(Long).

Клас ReservationService є реалізацією патерну Service для сутності бронювання. Призначенням цього класу є виконання всієї бізнес-логіки, яка стосується даної сутності. Для виконання свого призначення він містить наступні атрибути:

– reservationRepository необхідний для оновлення або отримання даних зі сховища даних;

– stationService необхідний для перевірки стану станцію, щоб було неможливо забронювати таку станцію, яка вийшла з ладу.

Клас ReservationService реалізує наступні методи:

– оновити інформацію про бронювання – updateReservation(ReservationDto);
– створити бронювання – createReservation(ReservationDto);
– видалити бронювання за його унікальним ідентифікатором – deleteReservation(Long);
– перевірити чи можливе бронювання на вказаний час – validateReservation(ReservationDto);
– отримати усі бронювання – getReservations();
– отримати бронювання за його унікальним ідентифікатором – getReservationById(Long);
– отримати усі бронювання користувача за його унікальним ідентифікатором – getUserReservations(Long).

Клас ReservationController є точкою виклику бізнес-логіки, яка стосується бронювання в системі, тобто є реалізацією патерну Controller для даної сутності. Щоб виконувати свої обов’язки він має атрибут reservationService необхідний безпосередньо для виклику бізнес-логіки, пов’язаної з бронюванням. Клас має наступні методи:

– оновити інформацію про бронювання – updateResertvation(ReservationDto);
– отримати усі бронювання – getReservations();
– видалити бронювання за його унікальним ідентифікатором – deleteReservation(Long);
– отримати усі бронювання користувача – getUserReservations(UserDto);
– створити бронювання – createReservation(Reservation, UserDto, StationDto).

2.6 Аналіз ризиків

Аналіз ризиків є необхідним, для завчасного виявлення ризиків й шляхів їх мінімізації. Перш за все необхідно їх визначити. Для цієї мети було проведено мозковий штурм, під час якого було згенеровані ризики, проаналізована їхня ймовірність та способи усунення (табл.2.6).

Таблиця 2.6 – Аналіз ризиків

Ризик	Ймовірність	Ступінь впливу	Шляхи мінімізації
Відсутність електроенергії	Середній	Середній	Створення автономного робочого місця шляхом придбання електрогенераторів та проведення оптоволокна. Використання автономних коворкінгів.
Виникнення нових вимог до системи	Низький	Високий	Визначення вимог на етапі аналізу. Аналіз конкурентів. Консультації з дипломним керівником
Недостатні навички	Низький	Високий	Консультації з дипломним керівником. Проходження тренінгів та курсів.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ

3.1 Опис використаних технологій

Java. Ключовою особливістю мови Java є те, що код спочатку транслюється в спеціальний байт-код, незалежний від платформи. Потім цей байт-код виконується віртуальною машиною JVM. У цьому плані Java відрізняється від стандартних інтерпретованих мов як PHP або Perl, код яких відразу виконується інтерпретатором. Водночас Java не є і мовою, що чисто компілюється, як C або C++ [10].

Подібна архітектура забезпечує кросплатформеність та апаратну переносимість програм на Java, завдяки чому подібні програми без перекомпіляції можуть виконуватись на різних платформах – Windows, Linux, Mac OS тощо. Для кожної з платформ може бути своя реалізація віртуальної машини JVM, але кожна з них може виконувати той самий код.

Ще однією ключовою особливістю Java є те, що вона підтримує автоматичне складання сміття. А це означає, що вам не треба звільняти вручну пам'ять від об'єктів, що раніше використовувалися, як в C++, оскільки збирач сміття це зробить автоматично за вас. Java є об'єктноорієнтованою мовою. Вона підтримує поліморфізм, наслідування, статичну типізацію. Об'єктноорієнтований підхід дозволяє вирішити завдання з побудови великих, але в той самий час гнучких, масштабованих і додатків, що розширюються.”

MySQL. “MySQL представляє систему управління реляційними базами даних. На сьогодні це одна з найпопулярніших систем керування базами даних [11].

MySQL має кросплатформеність, є дистрибутиви під різні ОС, у тому числі найбільш популярні версії Linux, Windows, MacOS.”

Spring Framework. “Spring Framework (або коротко Spring) – універсальний фреймворк з відкритим вихідним кодом для Java-платформи.

Хоч, Spring не забезпечував будь-яку конкретну модель програмування, він став широко поширеним у Java-спільноті головним чином як альтернатива та заміна моделі Enterprise JavaBeans. Spring надає більшу свободу Java-розробникам у проєктуванні; крім того, він надає добре документовані та легкі у використанні засоби розв'язання проблем, що виникають при створенні програм корпоративного масштабу.

Тим часом особливості ядра Spring застосовні в будь-якому Java-додатку, і існує безліч розширень та удосконалень для побудови вебдодатків на Java Enterprise платформі. З цих причин Spring набув великої популярності та визнається розробниками як стратегічно важливий фреймворк.”

“Термін «Spring» означає різні речі в різних контекстах. Його можна використовувати для позначення самого проєкту Spring Framework, з якого все почалося. Згодом інші проєкти Spring були створені на основі Spring Framework. Найчастіше, коли говорять «Spring» , мають на увазі всю сімейку проєктів.

Spring Framework розділений на модулі. Програми можуть вибирати, які модулі їм потрібні. В основі – модулі основного контейнера, включаючи модель конфігурації та механізм впровадження залежностей. Крім того, Spring Framework забезпечує базову підтримку для різних архітектур додатків, включаючи обмін повідомленнями, транзакційні дані та постійність, а також веб. Він також включає вебфреймворк Spring MVC на основі Servlet і, паралельно, реактивний вебфреймворк Spring WebFlux [12].

Maven. “Це інструмент автоматизації збірки, який використовується переважно для проєктів Java.

Maven розглядає два аспекти створення програмного забезпечення: як створюється програмне забезпечення та його залежності. XML-файл описує проєкт програмного забезпечення, що створюється, його залежності від інших зовнішніх модулів і компонентів, порядок збирання, каталоги та необхідні плагіни. Він поставляється з попередньо визначеними цілями для виконання певних чітко визначених завдань, таких як компіляція коду та

його пакування. Maven динамічно завантажує бібліотеки Java і плагіни Maven з одного або кількох репозиторіїв, таких як Центральний репозиторій Maven 2, і зберігає їх у локальному кеші [13].

Http протокол. “Hypertext Transfer Protocol (HTTP) – це протокол прикладного рівня для розподілених спільних гіпермедійних інформаційних систем. Це основа для передачі даних у Всесвітній павутині (тобто в Інтернеті) з 1990 року. HTTP є загальним протоколом без збереження стану, який можна використовувати для інших цілей, використовуючи розширення методів запиту, кодів помилок і заголовків. По суті, HTTP – це протокол зв’язку на основі TCP/IP, який використовується для доставлення даних (файлів HTML, файлів зображень, результатів запитів тощо) у Всесвітній павутині. Стандартним портом є TCP 80, але можна використовувати й інші порти. Він забезпечує стандартизований спосіб для комп’ютерів спілкуватися один з одним. Специфікація HTTP вказує, як дані запитів клієнтів будуть створюватися та надсилатися на сервер, а також як сервери відповідають на ці запити [14].

“Hibernate – засіб відображення між об'єктами та реляційними структурами (object-relational mapping, ORM) для платформи Java. Hibernate надає легкий для використання каркас (фреймворк) для відображення між об'єктноорієнтованою моделлю даних і традиційною реляційною базою даних.”

3.2 Опис програмної реалізації

Система поділена на 2 окремі частини: клієнт та сервер. Файлову структуру сервера зображено на рис. 3.1.

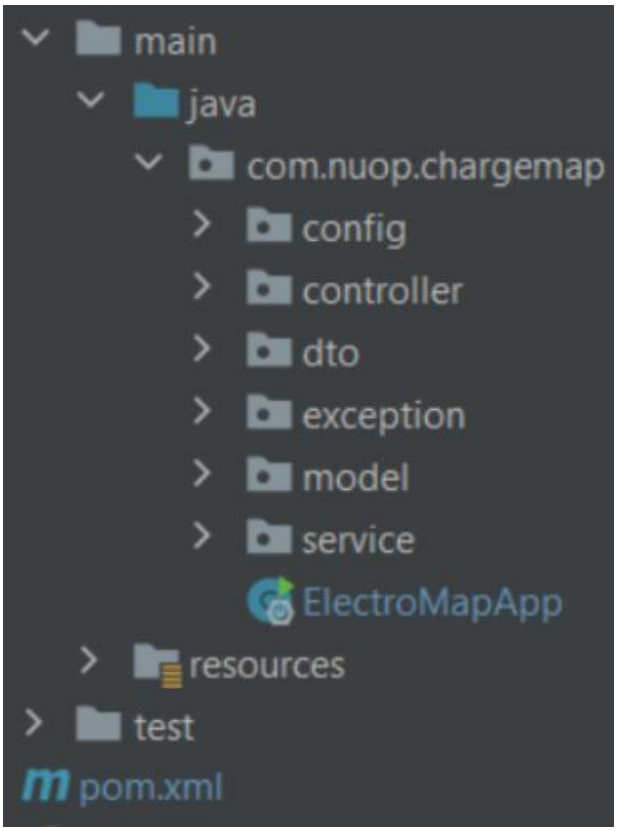

Рисунок 3.1 – Файлова структура сервера

Сервер має стандартний набір директорій для будь-якого Java застосунку. Корнева папка має таку ж назву як і сам проєкт – chargemap. Всередині неї знаходиться папка src, яка містить увесь код серверної частини та тести до нього в папках main та test відповідно. Також в цій папці міститься pom.xml файл, необхідний для налаштування автоматизатора збірки проєкту Maven.

Папка main містить 2 підпапки java та resources. В папці java зберігається суто код серверної частини, в resources – файл application.properties. Це такий файл, що містить усі необхідні змінні для роботи сервера. Наприклад, інформацію з підключення до бази даних, вид діалекту SQL, який використовується, спеціальні конфігурації Spring Boot.

Папки, які знаходяться в java заведено називати пакетами. За прийнятими стандартами оформлення пакетів на мові Java необхідно, щоб усі

класи було згруповані за пакетами, які дозволяються легко знайти необхідні класи. Так кореневим пакетом є com.nuop.chargemap. Ці три вкладені пакети відображають домен системи.

Розглянемо детально вміст пакета chargemap. Одним з пакетів є config. В цьому пакеті зберігається налаштування сервера, а саме налаштування безпеки, зокрема налаштування захисту від міжсайтового скриптинга. “Міжсайтовий скриптинг – тип вразливості інтерактивних інформаційних систем у вебі. Виникає, коли на сторінки, які були згенеровані сервером, з якоїсь причини потрапляють користувацькі скрипти. Специфіка подібних атак полягає в тому, що замість безпосередньої атаки сервера зловмисники використовують вразливий сервер для атаки на користувача.”

Далі йде пакет controller, який зберігає всі класи системи, що реалізують патерн Controller. Також в цьому пакеті міститься спеціальний контролер ErrorHandlerController.class, який дозволяє кастомізувати обробку помилок, які повертає сервер клієнту.

Наступними йдуть два пакети dto, model та exception. Пакет dto містить всі dto-класи системи. Пакет model – всі model-класи. Пакет exception – кастомні помилки, які можуть виникати під час роботи системи. Створення кастомних помилок допомагає відокремити причину помилку, та за допомогою ErrorHandlerController.class повернути клієнту пояснення причини виникнення помилки, що дозволить легко її виправити.

Останнім пакетом є service. Цей пакет містить всі класи, що реалізують бізнес-логіку системи. В ньому зберігаються інтерфейси всіх цих класів, а також в пакеті impl – реалізація інтерфейсів. Такий підхід обумовлений двома дизайн-патернами, на яких заснована архітектура вебсистем, написаних за допомогою Spring: Принцип інверсії залежностей та впровадження залежностей.

“Принцип інверсії залежностей (Dependency inversion) – один з п'яти SOLID-принципів об'єктноорієнтованого проєктування програм, суть якого

полягає у розриві зв'язності між програмними модулями вищого та нижчого рівнів за допомогою спільних абстракцій.”

“Впровадження залежностей (Dependency injection) – шаблон проєктування програмного забезпечення, що передбачає надання зовнішньої залежності програмному компоненту, використовуючи «інверсію керування» (англ. Inversion of control, IoC) для розв'язання (отримання) залежностей.”

Фреймворк Spring дозволяє в атрибутах класу вказувати лише інтерфейс, який потрібен для виклику всередині методів. Під час створення такого класу він знайде відповідну реалізацію даного інтерфейсу й підмінить її в цей атрибут. Такий підхід дозволяє легко підміняти реалізацію в залежності від потреб й дозволяє не переробляти кожний раз верхні рівні системі при необхідності зміни нижніх.

Розглянемо систему, яка була розроблена, з позиції користувача. Перше, куди потрапить користувач – головна сторінка (рис.3.2). У навігаційному меню розташовані дві кнопки: “Вхід” та “Реєстрація”, натиснувши на які користувач може авторизуватись в системі або зареєструватись відповідно.

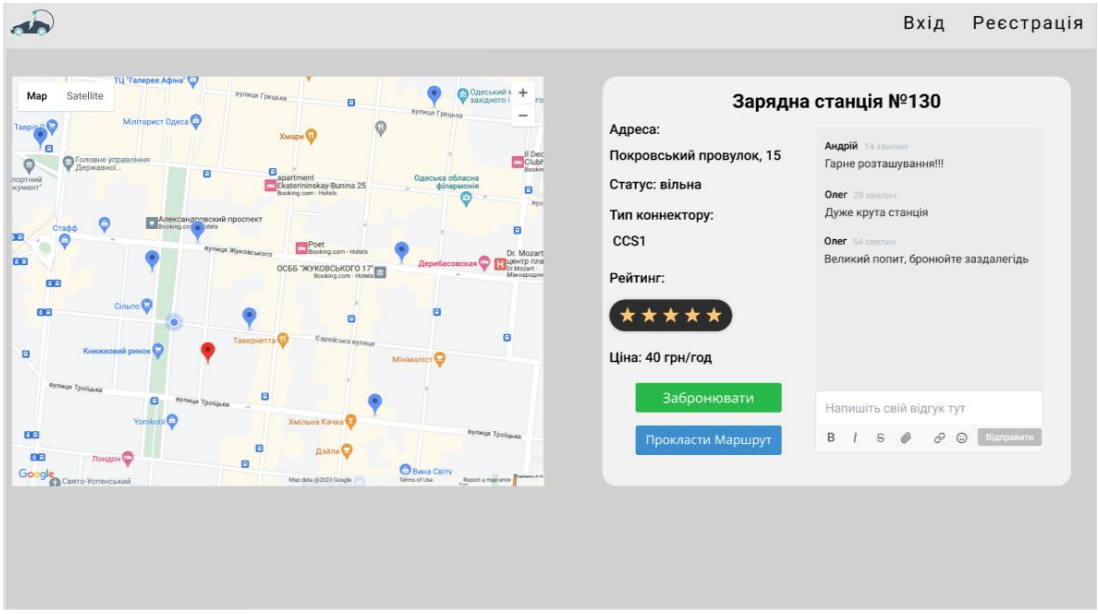

Рисунок 3.2 – Головна сторінка вебзастосунку неавторизованого користувача

Після проходження авторизації пункти навігаційного меню зміняться на “Мої бронювання” та “Мій профіль” (рис.3.3). Обравши “Мої бронювання”, користувач може переглянути історію та всі актуальні його бронювання. Натиснувши “Мій профіль”, користувач перейде до сторінки, яка відображає інформацію про нього. Також він зможе редагувати інформацію.

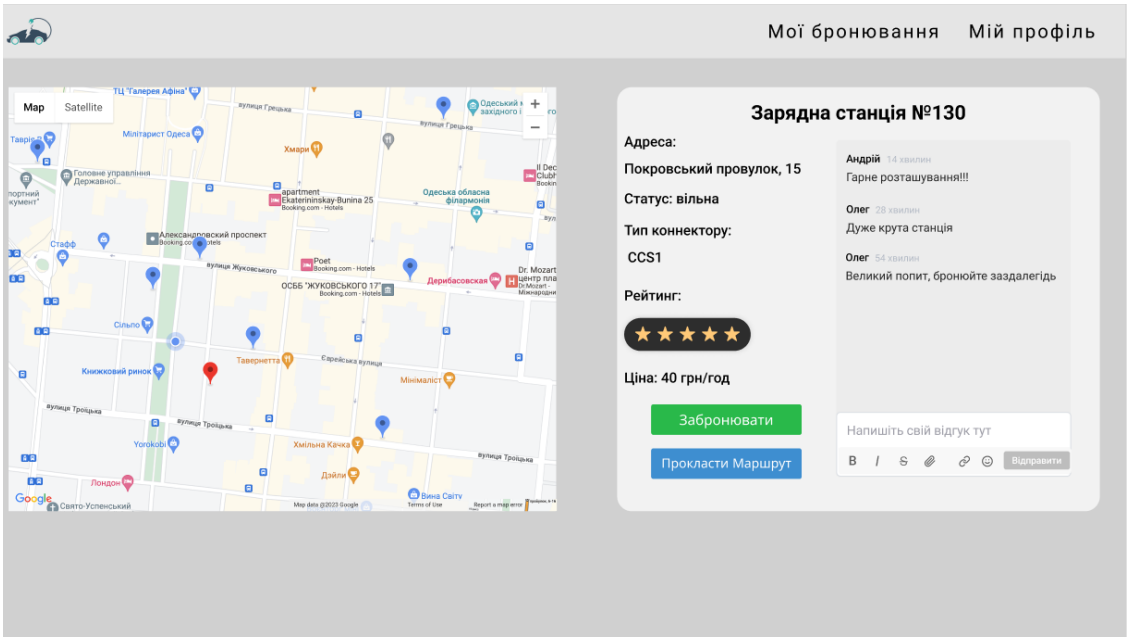

Рисунок 3.3 – Головна сторінка веб-застосунку авторизованого користувача

На головній сторінці користувачу відображається карта, на якій відмічені місцеперебування користувача (синє коло) та зарядні станції, які наявні в даних системи (сині маркери). Користувач зможе натиснути на будь-який синій маркер, щоб дізнатись детальну інформацію про станцію (маркер стане червоним). В правій частині екрана відобразиться інформація про станцію.

Щоб прокласти маршрут до станції, користувач має натиснути відповідну кнопку. Система побудує маршрут та відобразить його зеленим кольором на мапі (рис.3.4).

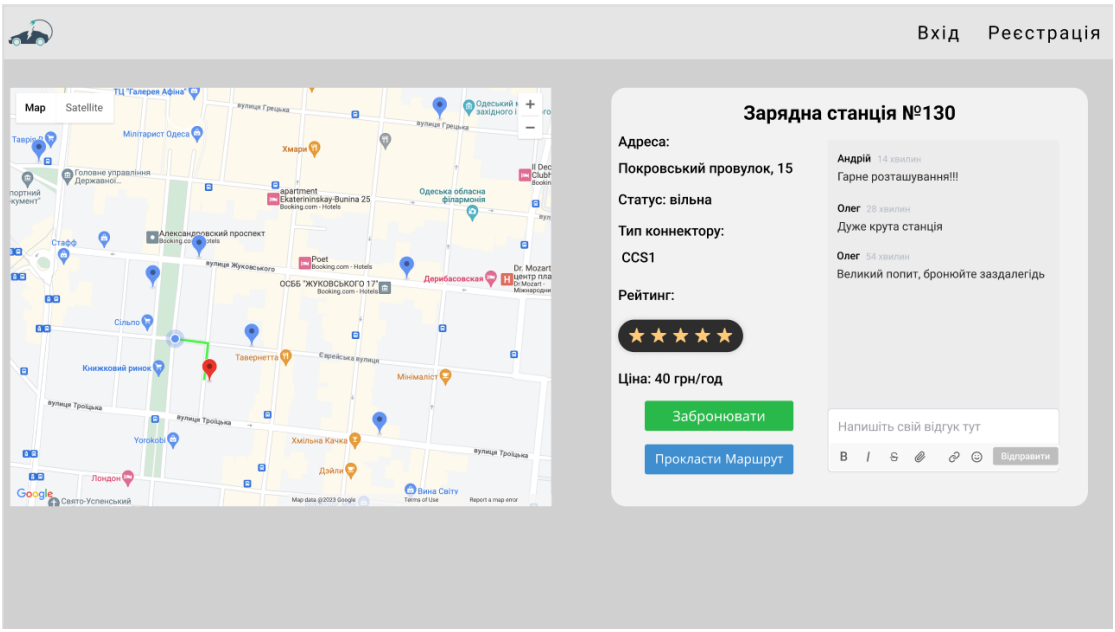

Рисунок 3.4 – Відображення прокладеного маршруту від користувача до обраної зарядної станції

Якщо користувач бажає забронювати обрану зарядну станцію, то він натисне відповідну кнопку “Забронювати”. Система відобразить необхідну форму для створення бронювання (рис.3.5).

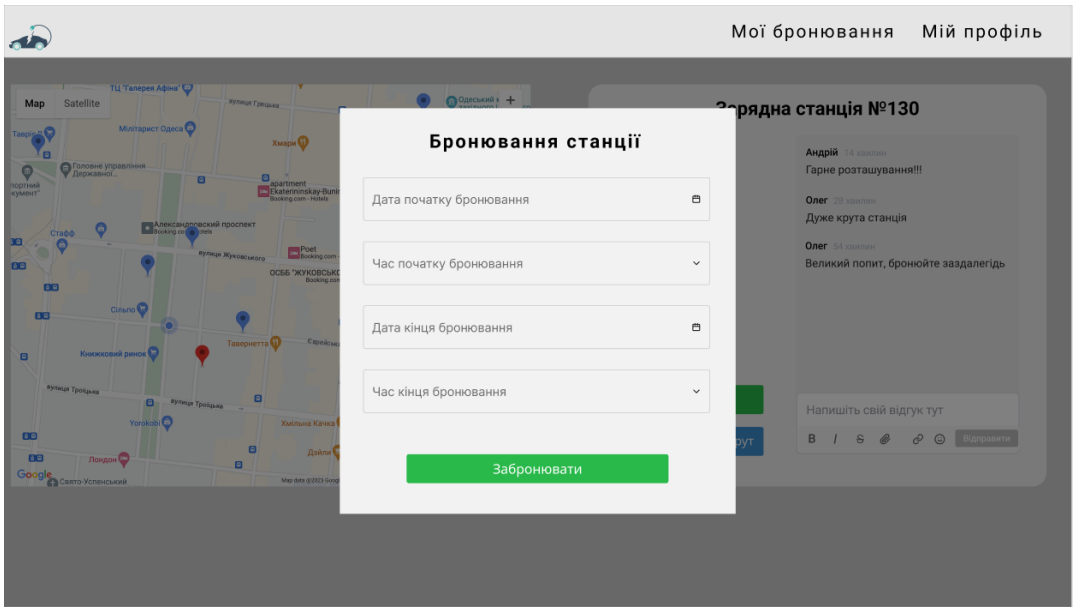

Рисунок 3.5 – Форма для створення бронювання обраної зарядної станції

Користувач заповнить дату та час, на яку він бажає забронювати зарядну станцію та натисне кнопку “Забронювати”. Система перевірить, чи не забронькована станція на вказаний час, та, якщо все гаразд й бронювання створено, відобразить повідомлення про успішність дії (рис.3.6).

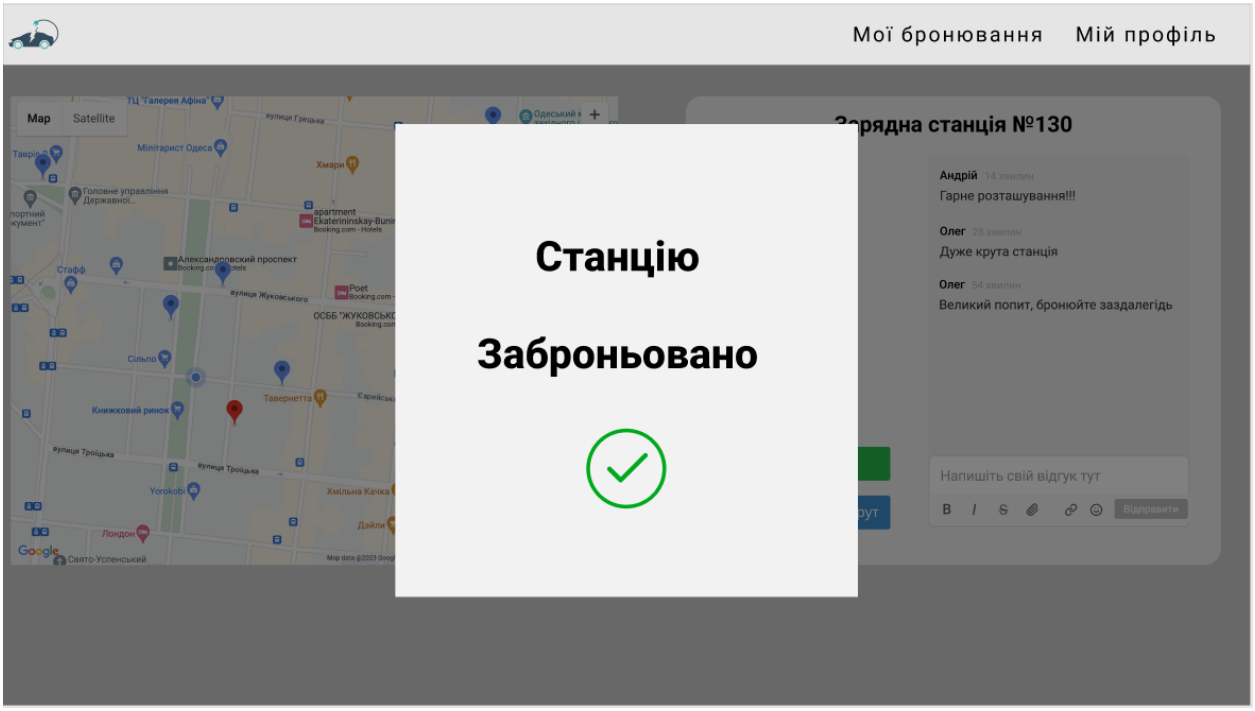

Рисунок 3.6 – Повідомлення про успішність створення бронювання

ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи бакалавра спроєктован та розроблен веб-застосунок для пошуку зарядних станцій.

У роботі реалізовані наступні задачі:

– визначення найближчих конкурентів та проведення аналізу їх функціонування;
– визначення функціональних і нефункціональних вимог до інформаційної системи;
– виконання проєктування інформаційної системи з використання шаблона проєктування Клієнт-Сервер;
– розробка та тестування застосунку.

Практична цінність розробленої системи полягає в тому, що автоматизовано процес пошуку зарядної станції. Для уникнення проблем, пов’язаних з пошуком, система дозволяє бронювати станцію на вказаний час, будувати маршрут до станції та переглядати в якому статусі вона перебуває в цей час.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Майбутнє автомобілів: чи врятують міста електрокари та гібридні автівки? URL: https://hmarochos. kiev.ua/2020/01/13/majbutnye-avtomobiliv-chy-vryatuyut-mista-elektrokary-ta-gibrydni-avtivky/. (дата звернення 01.05.2023)
2. Plugshare.com. URL: https://www.plugshare.com/ru. (дата звернення 01.05.2023)
3. Go-tou. URL: https://go-tou.com/ua/map. (дата звернення 02.05.2023)
4. 2chargers. URL: https://2chargers.net/. (дата звернення 02.05.2023)
5. Діаграма варіантів використання. URL:https://uk.wikipedia.org/wiki/%D0%94%D1%96%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%B0_%D0%BF%D1%80%D0%B5%D1%86%D0%B5%D0%B4%D0%B5%D0%BD%D1%82%D1%96%D0%B2. (дата звернення 03.05.2023)
6. Діаграма Послідовності. URL:https://uk.wikipedia.org/wiki/%D0%94%D1%96%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%B0_%D0%BF%D0%BE%D1%81%D0%BB%D1%96%D0%B4%D0%BE%D0%B2%D0%BD%D0%BE%D1%81%D1%82%D1%96(дата звернення 03.05.2023).
7. Архітектура вебзастосунків. URL: https://habr.com/ ru/articles/493430/. (дата звернення 03.05.2023)
8. Що таке Java й для чого вона потрібна? URL: https://www.java.com/en/download/help/whatis_java.html. (дата звернення 03.05.2023)
9. Мова програмування Java. URL: https://metanit.com/ java/tutorial/1.1.php. (дата звернення 04.05.2023)
10. Що таке Mysql? URL: https://metanit.com/sql/mysql/1.1.php. (дата звернення 04.05.2023)

11. Spring Framework [Електронний ресурс]. – Режим доступу: URL: https://uk.wikipedia.org/wiki/Spring_Framework. (дата звернення 05.05.2023)
12. Spring framework overview [Електронний ресурс]. – Режим доступу: URL: https://docs.spring.io/spring-framework/docs/current/reference/html/overview.htm. (дата звернення 05.05.2023)
13. Apache Maven. [Електронний ресурс]. – Режим доступу: URL: https://en.wikipedia.org/wiki/Apache_Maven. (дата звернення 06.05.2023)
14. Http Overview. [Електронний ресурс]. – Режим доступу: URL: https://www.tutorialspoint.com/http/http_overview.htm. (дата звернення 06.05.2023)

ДОДАТОК А

Лістинг програми

```
package  chargemap.model;

public class Rate {
    private Long id;
    private Long stationId;
    private Long userId;
    private float rate;
    private String commentary;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public Long getStationId() {
        return stationId;
    }

    public void setStationId(Long stationId) {
        this.stationId = stationId;
    }

    public Long getUserId() {
        return userId;
    }

    public void setUserId(Long userId) {
        this.userId = userId;
    }

    public float getRate() {
        return rate;
    }

    public void setRate(float rate) {
        this.rate = rate;
    }

    public String getCommentary() {
        return commentary;
    }

    public void setCommentary(String commentary) {
        this.commentary = commentary;
    }
}

package  chargemap.model;

import java.sql.Timestamp;
```

```
public class Reservation {
    private Long id;
    private Timestamp time;
    private Timestamp endTime;
    private String code;
    private Long userId;
    private Long stationId;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public Timestamp getTime() {
        return time;
    }

    public void setTime(Timestamp time) {
        this.time = time;
    }

    public String getCode() {
        return code;
    }

    public void setCode(String code) {
        this.code = code;
    }

    public Long getUserId() {
        return userId;
    }

    public void setUserId(Long userId) {
        this.userId = userId;
    }

    public Long getStationId() {
        return stationId;
    }

    public void setStationId(Long stationId) {
        this.stationId = stationId;
    }

    public Timestamp getEndTime() {
        return endTime;
    }

    public void setEndTime(Timestamp endTime) {
        this.endTime = endTime;
    }
}

package  chargemap.model;
```

```
public class Role {
    private Long id;
    private String Name;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return Name;
    }

    public void setName(String name) {
        Name = name;
    }
}

package  chargeamap.model;

public class Station {
    private long id;
    private String cordinates;
    private float rate;
    private float price;
    private String status;

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getCordinates() {
        return cordinates;
    }

    public void setCordinates(String cordinates) {
        this.cordinates = cordinates;
    }

    public float getRate() {
        return rate;
    }

    public void setRate(float rate) {
        this.rate = rate;
    }

    public float getPrice() {
        return price;
    }
```

```
    public void setPrice(float price) {
        this.price = price;
    }

    public String getStatus() {
        return status;
    }

    public void setStatus(String status) {
        this.status = status;
    }
}

package  chargemap.model;

import org.springframework.security.core.GrantedAuthority;
import
org.springframework.security.core.userdetails.UserDetails;

import java.util.Collection;

public class User implements UserDetails {
    private Long id;
    private String lastName;
    private String firstName;
    private String midName;
    private String mail;
    private String password;
    private Long autoId;
    private Long roleId;

    private Collection<? extends GrantedAuthority> authorities;
    private boolean isAccountNonExpired;
    private boolean isAccountNonLocked;
    private boolean isCredentialsNonExpired;
    private boolean isEnabled;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
```

```
    }

    public String getMidName() {
        return midName;
    }

    public void setMidName(String midName) {
        this.midName = midName;
    }

    public String getMail() {
        return mail;
    }

    public void setMail(String mail) {
        this.mail = mail;
    }

    @Override
    public      Collection<?      extends      GrantedAuthority>
getAuthorities() {
        return authorities;
    }

    public     void     setAuthorities(Collection<?     extends
GrantedAuthority> authorities) {
        this.authorities = authorities;
    }

    public String getPassword() {
        return password;
    }

    @Override
    public String getUsername() {
        return getMail();
    }

    @Override
    public boolean isAccountNonExpired() {
        return isAccountNonExpired;
    }

    public  void  setAccountNonExpired(boolean  accountNonExpired)
{
        isAccountNonExpired = accountNonExpired;
    }

    public void setAccountNonLocked(boolean accountNonLocked) {
        isAccountNonLocked = accountNonLocked;
    }

    public         void         setCredentialsNonExpired(boolean
credentialsNonExpired) {
        isCredentialsNonExpired = credentialsNonExpired;
    }

    public void setEnabled(boolean enabled) {
        isEnabled = enabled;
```

```
    }

    @Override
    public boolean isAccountNonLocked() {
        return isAccountNonLocked;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return isCredentialsNonExpired;
    }

    @Override
    public boolean isEnabled() {
        return isEnabled;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public Long getAutoId() {
        return autoId;
    }

    public void setAutoId(Long autoId) {
        this.autoId = autoId;
    }

    public Long getRoleId() {
        return roleId;
    }

    public void setRoleId(Long roleId) {
        this.roleId = roleId;
    }
}

package  chargemap.dto.repository;

import java.sql.*;

public class DBManager {
    private static final String USER = "root";
    private static final String PASSWORD = "1111";
    private      static      final      String      URL      =
"jdbc:mysql://localhost:3306/chargemap";

    public Connection getConnection() {
        Connection connection = null;
        try {
            Class.forName("com.mysql.jdbc.Driver");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        try {
            connection  =  DriverManager.getConnection(URL,  USER,
PASSWORD);
        } catch (SQLException throwables) {
```

```
            throwables.printStackTrace();
        }
        return connection;
    }

    public void closeResource(Connection connection, Statement
statement,      ResultSet      resultSet,      PreparedStatement
preparedStatement) {
        try {
            if (connection != null) {
                connection.close();
            }
            if (statement != null) {
                statement.close();
            }
            if (resultSet != null) {
                resultSet.close();
            }
            if (preparedStatement != null) {
                preparedStatement.close();
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public void closeResource(Connection connection, Statement
statement, ResultSet resultSet) {
        closeResource(connection, statement, resultSet, null);
    }

    public     void     closeResource(Connection     connection,
PreparedStatement preparedStatement, ResultSet resultSet) {
        closeResource(connection,        null,        resultSet,
preparedStatement);
    }

    public     void     closeResource(Connection     connection,
PreparedStatement preparedStatement) {
        closeResource(connection,           null,           null,
preparedStatement);
    }
}

package  chargemap.dto.repository;

import  chargemap.dto.RateDto;
import  chargemap.model.Rate;

import java.util.List;

public interface RateDAO {
    List<Rate> getAll();

    Rate getById(Long id);

    List<RateDto> getByStationId(Long id);

    Boolean isRateExist(Long userId, Long stationId);
```

```
    void create(Rate rate);

    void delete(Long id);

    void update(Long id, Rate rate);
}

package  chargemap.dto.repository;

import  chargemap.dto.ReservationDto;
import  chargemap.model.Reservation;

import java.util.List;

public interface ReservationDAO {
    List<Reservation> getAll();

    List<ReservationDto> getAllUser(Long id);

    Reservation getById(Long id);

    Boolean isReservationExist(Long id, String startTimestamp,
String endTimestamp);

    void create(Reservation reservation);

    void delete(Long id);

    void update(Long id, Reservation reservation);
}

package  chargemap.dto.repository;

import  chargemap.model.Role;

import java.util.List;

public interface RoleDAO {
    List<Role> getAll();

    Role getById(Long id);

    Role getByName(String name);
}

package  chargemap.dto.repository;

import  chargemap.model.Station;

import java.util.List;

public interface StationDAO {
    List<Station> getAll();

    List<Station> getAllOrderByPriceAsc();

    List<Station> getAllOrderByPriceDesc();
```

```
    List<Station> getAllOrderByRateAsc();

    List<Station> getAllOrderByRateDesc();

    List<Station> getAllFavouriteOrderByPriceAsc(Long id);

    List<Station> getAllFavouriteOrderByPriceDesc(Long id);

    List<Station> getAllFavouriteOrderByRateAsc(Long id);

    List<Station> getAllFavouriteOrderByRateDesc(Long id);

    List<Station> getAllUserFavourite(Long id);

    List<Station> getAllByUserAuto(Long id);

    Station getById(Long id);

    void create(Station station);

    void delete(Long id);

    void update(Long id, Station station);

    void updateRate(Long id);
}

package  chargecracker.dto.repository;

import  chargecracker.model.User;

import java.util.List;

public interface UserDAO {
    List<User> getAll();

    User getById(Long id);

    User getByMail(String mail);

    void create(User user);

    void createFullUser(User user);

    void delete(Long id);

    void update(Long id, User user);

    void updateFullUser(Long id, User user);

    void updateAuto(Long userId, Long autoId);

    void updatePassword(Long id, String password);
}

package  chargemap.dto.repository.impl;

import  chargemap.dto.repository.DBManager;
import  chargemap.dto.repository.RateDAO;
```

```
import  chargemap.dto.RateDto;
import  chargemap.model.Rate;
import org.springframework.stereotype.Repository;

import java.sql.*;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;

@Repository
public class RateDAOImpl implements RateDAO {
    private DBManager dbManager = new DBManager();

    @Override
    public List<Rate> getAll() {
        List<Rate> rates = new LinkedList<>();
        Connection connection = null;
        Statement statement = null;
        ResultSet resultSet = null;
        try {
            connection = dbManager.getConnection();
            statement = connection.createStatement();
            resultSet  =  statement.executeQuery("SELECT  *  FROM
RATES");

            while (resultSet.next()) {
                rates.add(fillRate(resultSet));
            }

            return rates;
        } catch (SQLException exc) {
            exc.printStackTrace();
        } finally {
            dbManager.closeResource(connection,       statement,
resultSet);
        }

        return null;
    }

    @Override
    public Rate getById(Long id) {
        Connection connection = null;
        PreparedStatement preparedStatement = null;
        ResultSet resultSet = null;
        try {
            connection = dbManager.getConnection();
            preparedStatement = connection.prepareStatement(
                    "SELECT * FROM RATES WHERE RATE_ID = ?");
            preparedStatement.setLong(1, id);
            resultSet = preparedStatement.executeQuery();
            if (!resultSet.next()) {
                return null;
            }

            return fillRate(resultSet);
        } catch (SQLException exc) {
            exc.printStackTrace();
        } finally {
```

```
            dbManager.closeResource(connection,
preparedStatement, resultSet);
        }

        return null;
    }

    @Override
    public List<RateDto> getByStationId(Long id) {
        List<RateDto> rateList = new ArrayList<>();
        Connection connection = null;
        PreparedStatement preparedStatement = null;
        ResultSet resultSet = null;
        try {
            connection = dbManager.getConnection();
            preparedStatement = connection.prepareStatement("" +
                    "SELECT         U.USER_ID,         R.RATE_ID,
CONCAT(U.LASTNAME,  '  ',  U.FIRSTNAME)      FULL_NAME,  R.RATE,
R.COMMENTARY " +
                    "FROM RATES R, USERS U " +
                    "WHERE  STATION_ID  =  ?  AND  U.USER_ID  =
R.USER_ID");
            preparedStatement.setLong(1, id);
            resultSet = preparedStatement.executeQuery();
            while (resultSet.next()) {
                RateDto rateDto = new RateDto();
                rateDto.setRateId(resultSet.getLong("RATE_ID"));
                rateDto.setUserId(resultSet.getLong("USER_ID"));
                rateDto.setRate(resultSet.getFloat("RATE"));

rateDto.setFullName(resultSet.getString("FULL_NAME"));

rateDto.setCommentary(resultSet.getString("COMMENTARY"));
                rateList.add(rateDto);
            }

            return rateList;
        } catch (SQLException exc) {
            exc.printStackTrace();
        } finally {
            dbManager.closeResource(connection,
preparedStatement, resultSet);
        }

        return null;
    }

    @Override
    public Boolean isRateExist(Long userId, Long stationId) {
//TODO CHECK
        Connection connection = null;
        PreparedStatement preparedStatement = null;
        ResultSet resultSet = null;
        try {
            connection = dbManager.getConnection();
            preparedStatement = connection.prepareStatement("" +
                    "SELECT  IF(RATE_ID  IS  NOT  NULL,  'TRUE',
'FALSE') RATE_EXIST " +
                    "FROM RATES " +
```

```
                    "WHERE USER_ID = ? AND STATION_ID = ? ");
            preparedStatement.setLong(1, userId);
            preparedStatement.setLong(2, stationId);
            resultSet = preparedStatement.executeQuery();

            return resultSet.next();
        } catch (SQLException exc) {
            exc.printStackTrace();
        } finally {
            dbManager.closeResource(connection,
preparedStatement, resultSet);
        }

        return null;
    }

    private    Rate    fillRate(ResultSet    resultSet)    throws
SQLException {
        Rate rate = new Rate();

        rate.setId(resultSet.getLong("RATE_ID"));
        rate.setStationId(resultSet.getLong("STATION_ID"));
        rate.setUserId(resultSet.getLong("USER_ID"));
        rate.setRate(resultSet.getFloat("RATE"));
        rate.setCommentary(resultSet.getString("COMMENTARY"));

        return rate;
    }

    @Override
    public void create(Rate rate) {
        Connection connection = null;
        PreparedStatement preparedStatement = null;
        try {
            connection = dbManager.getConnection();
            preparedStatement = connection.prepareStatement(
                    "INSERT  INTO  RATES  (STATION_ID,  USER_ID,
RATE, COMMENTARY) VALUES(?, ?, ?, ?)");
            preparedStatement.setLong(1, rate.getStationId());
            preparedStatement.setLong(2, rate.getUserId());
            preparedStatement.setFloat(3, rate.getRate());
            preparedStatement.setString(4,
rate.getCommentary());
            preparedStatement.executeUpdate();
        } catch (SQLException exc) {
            exc.printStackTrace();
        } finally {
            dbManager.closeResource(connection,
preparedStatement);
        }
    }

    @Override
    public void delete(Long id) {
        Connection connection = null;
        PreparedStatement preparedStatement = null;
        try {
            connection = dbManager.getConnection();
```

```
            preparedStatement                                    =
connection.prepareStatement("DELETE FROM RATES WHERE RATE_ID =
?");
            preparedStatement.setLong(1, id);
            preparedStatement.executeUpdate();
        } catch (SQLException exc) {
            exc.printStackTrace();
        } finally {
            dbManager.closeResource(connection,
preparedStatement);
        }
    }

    @Override
    public void update(Long id, Rate rate) {
        Connection connection = null;
        PreparedStatement preparedStatement = null;
        try {
            connection = dbManager.getConnection();
            preparedStatement                                    =
connection.prepareStatement("UPDATE RATES SET " +
                    "USER_ID = ?, " +
                    "STATION_ID = ?," +
                    "RATE = ?," +
                    "COMMENTARY = ? " +
                    "WHERE RATE_ID = ?");
            preparedStatement.setLong(1, rate.getUserId());
            preparedStatement.setLong(2, rate.getStationId());
            preparedStatement.setFloat(3, rate.getRate());
            preparedStatement.setString(4,
rate.getCommentary());
            preparedStatement.setLong(5, id);
            preparedStatement.executeUpdate();
        } catch (SQLException exc) {
            exc.printStackTrace();
        } finally {
            dbManager.closeResource(connection,
preparedStatement);
        }
    }
}

package  chargemap.dto.repository.impl;

import  chargemap.dto.repository.DBManager;
import  chargemap.dto.repository.ReservationDAO;
import  chargemap.dto.ReservationDto;
import  chargemap.model.Reservation;
import org.springframework.stereotype.Repository;

import java.sql.*;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.LinkedList;
import java.util.List;

@Repository
public class ReservationDAOImpl implements ReservationDAO {
```

```
    private DBManager dbManager = new DBManager();

    @Override
    public List<Reservation> getAll() {
        List<Reservation> reservations = new LinkedList<>();
        Connection connection = null;
        Statement statement = null;
        ResultSet resultSet = null;
        try {
            connection = dbManager.getConnection();
            statement = connection.createStatement();
            resultSet  =  statement.executeQuery("SELECT  *  FROM
RESERVATIONS");

            while (resultSet.next()) {
                reservations.add(fillReservation(resultSet));
            }
            return reservations;
        } catch (SQLException exc) {
            exc.printStackTrace();
        } finally {
            dbManager.closeResource(connection,       statement,
resultSet);
        }

        return null;
    }

    @Override
    public List<ReservationDto> getAllUser(Long id) {
        List<ReservationDto> result = new ArrayList<>();
        Connection connection = null;
        PreparedStatement preparedStatement = null;
        ResultSet resultSet = null;
        try {
            connection = dbManager.getConnection();
            preparedStatement = connection.prepareStatement("" +
                    "SELECT R.RESERVATION_ID," +
                    "       R.CODE," +
                    "         DATE_FORMAT(R.TIME, '%d/%m/%y %T')
START_TIME," +
                    "          DATE_FORMAT(R.TIME_END, '%d/%m/%y
%T') END_TIME," +
                    "       S.CORDINATES\n" +
                    "FROM  RESERVATIONS  R  JOIN  STATIONS  S  on
R.STATION_ID = S.STATION_ID\n" +
                    "WHERE R.USER_ID = ?" +
                    "  AND R.TIME_END > NOW()" +
                    "ORDER BY START_TIME, END_TIME;");
            preparedStatement.setLong(1, id);
            resultSet = preparedStatement.executeQuery();
            while (resultSet.next()) {
                ReservationDto     reservationDto     =     new
ReservationDto();

reservationDto.setCode(resultSet.getString("CODE"));

reservationDto.setCordinates(resultSet.getString("CORDINATES"));
```

```
reservationDto.setReservationId(resultSet.getLong("RESERVATION_I
D"));

reservationDto.setStartTime(resultSet.getString("START_TIME"));

reservationDto.setEndTime(resultSet.getString("END_TIME"));
                result.add(reservationDto);
            }

            if (result.size() == 0) {
                return null;
            }

            return result;
        } catch (SQLException exc) {
            exc.printStackTrace();
        } finally {
            dbManager.closeResource(connection,
preparedStatement, resultSet);
        }

        return null;
    }

    @Override
    public Reservation getById(Long id) {
        Connection connection = null;
        PreparedStatement preparedStatement = null;
        ResultSet resultSet = null;
        try {
            connection = dbManager.getConnection();
            preparedStatement = connection.prepareStatement(
                    "SELECT    *    FROM    RESERVATIONS    WHERE
RESERVATION_ID = ?");
            preparedStatement.setLong(1, id);
            resultSet = preparedStatement.executeQuery();
            if (!resultSet.next()) {
                return null;
            }

            return fillReservation(resultSet);
        } catch (SQLException exc) {
            exc.printStackTrace();
        } finally {
            dbManager.closeResource(connection,
preparedStatement, resultSet);
        }

        return null;
    }
//TODO CHECK
    @Override
    public    Boolean    isReservationExist(Long    id,    String
startTimestamp, String endTimestamp) {
        Connection connection = null;
        PreparedStatement preparedStatement = null;
        ResultSet resultSet = null;
        try {
```

```
            connection = dbManager.getConnection();
            preparedStatement = connection.prepareStatement("" +
                    "SELECT * " +
                    "FROM RESERVATIONS " +
                    "WHERE  STATION_ID  =  ?  AND  (STR_TO_DATE(?,
'dd.mm.yyyy HH24:MI:SS') BETWEEN TIME AND TIME_END " +
                    "OR  STR_TO_DATE(?,  'dd.mm.yyyy  HH24:MI:SS')
BETWEEN TIME AND TIME_END)");
            preparedStatement.setLong(1, id);
            preparedStatement.setString(2, startTimestamp);
            preparedStatement.setString(3, endTimestamp);
            resultSet = preparedStatement.executeQuery();

            return resultSet.next();
        } catch (SQLException exc) {
            exc.printStackTrace();
        } finally {
            dbManager.closeResource(connection,
preparedStatement, resultSet);
        }

        return null;
    }

    private  Reservation  fillReservation(ResultSet  resultSet)
throws SQLException {
        Reservation reservation = new Reservation();

        reservation.setId(resultSet.getLong("RESERVATION_ID"));
        reservation.setTime(resultSet.getTimestamp("TIME"));

reservation.setEndTime(resultSet.getTimestamp("TIME_END"));
        reservation.setCode(resultSet.getString("CODE"));

reservation.setStationId(resultSet.getLong("STATION_ID"));
        reservation.setUserId(resultSet.getLong("USER_ID"));

        return reservation;
    }

    @Override
    public void create(Reservation reservation) {
        Connection connection = null;
        PreparedStatement preparedStatement = null;
        try {
            connection = dbManager.getConnection();
            preparedStatement = connection.prepareStatement(
                    "INSERT  INTO  RESERVATIONS  (TIME,  TIME_END,
CODE  ,STATION_ID,  USER_ID,  CREATED_AT)  VALUES(?,  ?,  ?,  ?,  ?,
STR_TO_DATE(?, '%d/%m/%Y'))");
            preparedStatement.setTimestamp(1,
reservation.getTime());
            preparedStatement.setTimestamp(2,
reservation.getEndTime());
            preparedStatement.setString(3,
reservation.getCode());
            preparedStatement.setLong(4,
reservation.getStationId());
```

```
            preparedStatement.setLong(5,
reservation.getUserId());
            SimpleDateFormat        formatter        =        new
SimpleDateFormat("dd/MM/yyyy");
            preparedStatement.setString(6,   formatter.format(new
Date()));
            preparedStatement.executeUpdate();
        } catch (SQLException exc) {
            exc.printStackTrace();
        } finally {
            dbManager.closeResource(connection,
preparedStatement);
        }
    }

    @Override
    public void delete(Long id) {
        Connection connection = null;
        PreparedStatement preparedStatement = null;
        try {
            connection = dbManager.getConnection();
            preparedStatement                                    =
connection.prepareStatement("DELETE   FROM   RESERVATIONS   WHERE
RESERVATION_ID = ?");
            preparedStatement.setLong(1, id);
            preparedStatement.executeUpdate();
        } catch (SQLException exc) {
            exc.printStackTrace();
        } finally {
            dbManager.closeResource(connection,
preparedStatement);
        }
    }
    //TODO CHECK TIME
    @Override
    public void update(Long id, Reservation reservation) {
        Connection connection = null;
        PreparedStatement preparedStatement = null;
        try {
            connection = dbManager.getConnection();
            preparedStatement                                    =
connection.prepareStatement("UPDATE RESERVATIONS SET " +
                    "TIME = ?," +
                    "TIME_END = ?," +
                    "CODE = ?, " +
                    "USER_ID = ?, " +
                    "STATION_ID = ? " +
                    "WHERE RESERVATION_ID = ?");
            preparedStatement.setTimestamp(1,
reservation.getTime());
            preparedStatement.setTimestamp(2,
reservation.getEndTime());
            preparedStatement.setString(3,
reservation.getCode());
            preparedStatement.setLong(4,
reservation.getUserId());
            preparedStatement.setLong(5,
reservation.getStationId());
            preparedStatement.setLong(6, id);
```

```
            preparedStatement.executeUpdate();
        } catch (SQLException exc) {
            exc.printStackTrace();
        } finally {
            dbManager.closeResource(connection,
preparedStatement);
        }
    }
}
```