

Міністерство освіти і науки України
Державний заклад
«Луганський національний університет імені Тараса Шевченка»

Навчально-науковий інститут математики та інформаційних технологій

Кафедра математики та інформатики

Мамчур Сергій Вікторович

**ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ВЕБ ОРІЄНТОВАНОЇ
ІНФОРМАЦІЙНОЇ СИСТЕМИ ПОШУКУ ТА БРОНЮВАННЯ
ПАРКОМІСЦЬ**

кваліфікаційна робота

**здобувача вищої освіти другого (магістерського) рівня
освітньої програми «Комп'ютерні науки та інформаційні технології»
за спеціальністю 122 Комп'ютерні науки**

Особистий підпис _____  _____ Сергій МАМЧУР

Науковий керівник _____ Владислав КОЗУБ,
доктор філософії, старший викладач
кафедри інформаційних технологій
та систем

В.о. завідувача кафедри _____ Юрій КОЗУБ,
доктор технічних наук, професор
кафедри математики та інформатики

АНОТАЦІЯ

Мамчур С. В.

Тема: Проектування та реалізація веб орієнтованої інформаційної системи пошуку та бронювання паркомісць.

Спеціальність: 122 «Комп'ютерні науки».

Установа: ЛНУ імені Тараса Шевченка, 2026р.

Магістерська робота містить: 88 с., 41 рис., 2 табл., 26 джерел.

Об'єкт дослідження: веб орієнтовна система пошуку та бронювання паркомісць.

Предмет дослідження: процес швидкого пошуку та бронювання паркомісць.

Мета дослідження: проектування та реалізація веб орієнтованої інформаційної системи пошуку та бронювання паркомісць.

Результати роботи. Проаналізовано підходи до побудови веб орієнтованих додатків. Розглянуто та проаналізовано існуючі додатки для пошуку та бронювання паркомісць які працюють на території України. Сформульовано змістову та математичну постановки задачі.

Проведено моделювання та аналіз веб орієнтованої інформаційної системи пошуку та бронювання паркомісць. Сформовано вимоги до технічного забезпечення та побудована архітектура системи.

Розроблено веб орієнтовану інформаційну систему для пошуку та бронювання паркомісць. У системі доступний пошук паркомісць та їх подальше бронювання, а також керування власним кабінетом. Серверна частина реалізована на мові програмування Java 8, а клієнтська – TypeScript, фреймворк Angular 8. Для збереження даних використовується реляційна база даних PostgreSQL.

Ключові слова: ІНФОРМАЦІЙНА СИСТЕМА, БАЗА ДАНИХ, ФОРМУЛА ГАВЕРСИНУСА, АЛГОРИТМ ДЕЙКСТРИ, СИСТЕМА, БРОНЮВАННЯ, ШТРАФ, ПАРКОМІСЦЕ, POSTGRESQL, ANGULAR 8, JAVA.

ANNOTATION

Mamchur Serhii

Theme: Design and implementation of a web-based information system for searching and booking parking spaces.

Speciality: 122 "Computer Science".

Institution: Luhansk Taras Shevchenko National University (LTSNU), 2026 year.

Master's work of: 88 p., 41 im, 26 sources.

A research object of: Web-based parking space search and reservation system..

The article of research- Quick search and reservation process for parking spaces.

An aim of research is Design and implementation of a web-based information system for searching and booking parking spaces.

Job performanes. The approaches to building web-oriented applications were analyzed. Existing applications for searching and booking parking spaces operating in Ukraine were considered and analyzed. The content and mathematical formulation of the problem was formulated.

The modeling and analysis of the web-oriented information system for searching and booking parking spaces were carried out. The requirements for technical support were formed and the system architecture was built.

A web-oriented information system for searching and booking parking spaces was developed. The system allows searching for parking spaces and their subsequent booking, as well as managing your own account. The server part is implemented in the Java 8 programming language, and the client part is implemented in TypeScript, the Angular 8 framework. The PostgreSQL relational database is used to store data.

Keywords: INFORMATION SYSTEM, DATABASE, HAVERSINUS FORMULA, Dijkstra's ALGORITHM, SYSTEM, RESERVATION, FINE, PARKING SPACE, POSTGRESQL, ANGULAR 8, JAVA.

| | |
|--|-----------|
| ЗМІСТ | |
| ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ | 6 |
| ВСТУП | 7 |
| РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОГО СЕРЕДОВИЩА ТА МАТЕМАТИЧНОГО ЗАБЕЗПЕЧЕННЯ | 10 |
| 1.1. Огляд і аналіз сучасних підходів до проєктування інформаційних систем паркування | 10 |
| 1.2. Огляд аналогів..... | 15 |
| 1.3. Аналіз предметного середовища | 25 |
| 1.3.1. Вимоги до функціональних характеристик..... | 25 |
| 1.3.2. Вхідні дані..... | 26 |
| 1.3.3. Вихідні дані | 27 |
| 1.3.4. Опис процесу діяльності | 28 |
| 1.3.5. Опис функціональної моделі | 30 |
| 1.4. Дослідження математичного забезпечення..... | 31 |
| 1.4.1. Змістова постановка задачі | 31 |
| 1.4.2. Математична постановка задачі | 32 |
| 1.4.3. Обґрунтування методу розв’язання..... | 32 |
| 1.4.4. Опис методу розв’язання..... | 33 |
| Висновки до розділу | 38 |
| РОЗДІЛ 2. МОДЕЛЮВАННЯ ТА АНАЛІЗ ВЕБ ОРІЄНТОВАНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ ПОШУКУ ТА БРОНЮВАННЯ ПАРКОМІСЦЬ..... | 39 |
| 2.1. Вимоги до технічного забезпечення | 39 |
| 2.2. Архітектура програмного забезпечення | 39 |
| 2.2.1. Логічне представлення статичної моделі структури програмного забезпечення | 40 |
| 2.2.2. Діаграма компонентів | 42 |
| 2.2.3. Діаграма послідовності | 42 |
| 2.2.4. Діаграма станів | 44 |

| | |
|--|------------|
| 2.3. Опис структури бази даних..... | 44 |
| Висновки до розділу | 52 |
| РОЗДІЛ 3. РОЗРОБКА ВЕБ ОРІЄНТОВАНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ ПОШУКУ ТА БРОНЮВАННЯ ПАРКОМІСЦЬ | 54 |
| 3.1. Обґрунтування вибору засобів розробки | 54 |
| 3.2. Специфікація функцій | 57 |
| 3.3. Розробка інтерфейсу | 70 |
| 3.4. Керівництво користувача | 72 |
| Висновки до розділу | 82 |
| ВИСНОВКИ..... | 84 |
| СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ..... | 86 |
| ДОДАТОК А | 89 |
| ДОДАТОК Б | 107 |

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

| | |
|------|------------------------------------|
| HQL | Hibernate Query Language |
| JPA | Java Persistence API |
| СКБД | Система керування базою даних |
| IDE | Integrated Development Environment |
| MVC | Model-View-Controller |
| ПЗ | Програмне забезпечення |
| UML | Unified Model Language |

ВСТУП

Парковка є невід’ємною частиною міської інфраструктури. Раціональне використання паркувального простору важливе для офісних, житлових, торгово-розважальних і адміністративних об’єктів. Водночас спостерігається дисбаланс між темпами автомобілізації та розвитком дорожньої мережі, що призводить до дефіциту місць для паркування. Як свідчить статистика, близько 20% трафіку в центрах міст складається з водіїв, які шукають вільне місце [1, 2]. Це робить управління паркувальним простором однією з ключових задач.

Сучасні системи управління парковками сприяють підвищенню комфорту користувачів і ефективності для власників. Такі системи забезпечують автоматичний контроль в’їзду, виїзду, розрахунок оплати й адаптивну тарифікацію. Вони знижують експлуатаційні витрати, мінімізують зловживання, покращують заповнюваність і збільшують пропускну здатність [3]. Автоматизовані паркінги мають значні переваги: вони економлять простір, кошти, час, а також забезпечують високий рівень безпеки. Дослідження показують, що такі системи зменшують трафік на 9%, викиди газів — на 45%, відстань, яку долають автомобілі до паркування, — на 35%, а час, витрачений на пошук місця, — на 41% [4].

Інтелектуальні системи паркування (SmartParking) дозволяють користувачам шукати та бронювати вільні місця, перевіряти залишковий час сесії, а також спрощують реєстрацію транспортних засобів. Веб додаток, інтегрований з такими системами, стане зручним рішенням для водіїв, забезпечуючи ефективний контроль оплати, бронювання і запобігання порушенням [5].

Кожне місто України, зокрема Київ, має великі проблеми з парковками, а саме загальної кількості паркомісць просто недостатньо. Окрім того, що кількість паркомісць недостатня, існує проблема ведення паркобізнесу. Для того, щоб ефективно здавати в оренду паркомісця, необхідно мати онлайн систему для бронювання. Розробка та подальша реклама такої системи досить

дорога. Рішенням цієї проблеми є розробка єдиного маркетплейсу парковок, адже даний тип системи буде працювати не з одним власником парковки. Будь-хто може додати власні парковки і отримувати дохід від їх оренди. Звісно, така система має передбачувати перевірку права на ведення даного типу бізнесу аби уникнути його неправомірного ведення і як наслідок ухилянь від податків.

Даний дипломний проєкт націлений на побудову системи, яка буде забезпечувати потреби клієнтів, тобто водіїв.

Об'єкт дослідження: веб орієнтовна система пошуку та бронювання паркомісць.

Предмет дослідження: процес швидкого пошуку та бронювання паркомісць.

Мета дослідження: проєктування та реалізація веб орієнтованої інформаційної системи пошуку та бронювання паркомісць.

Методи дослідження: порівняння та аналіз.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

- розробка власного кабінету для клієнтів, який дозволить керувати кредитними картками та переглядати історію замовлень і штрафів;
- розробка пошуку найближчих паркувальних майданчиків до обраної адреси;
- розробка пошуку вільних паркомісць на обраній парковці на вказаний проміжок часу та їх бронювання.

Практичною цінністю роботи є розробка веб орієнтованої інформаційної системи для пошуку та подальшого бронювання знайдених паркомісць.

В першому розділі проаналізовано предметне середовище та виконано порівняння системи з існуючими аналогами. Проведено дослідження математичного забезпечення інформаційної системи.

В другому розділі виконано моделювання та аналіз веб-орієнтованої інформаційної системи пошуку та бронювання паркомісць. Сформовано вимоги до технічного забезпечення системи та описано її архітектуру.

У третьому розділі аргументується вибір засобів розробки. Наведено специфікацію функцій серверної та клієнтської частин застосунку. Спроектовано веб-інтерфейс системи з описанням наявного функціоналу і побудовано керівництво користувача.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОГО СЕРЕДОВИЩА ТА МАТЕМАТИЧНОГО ЗАБЕЗПЕЧЕННЯ

1.1. Огляд і аналіз сучасних підходів до проєктування інформаційних систем паркування

Автоматизовані паркінгові системи стають дедалі більш затребуваними в умовах постійного зростання кількості транспортних засобів у міських агломераціях. Найчастіше такі парковки функціонують поблизу великих торговельно-розважальних центрів, спортивних комплексів, багатофункціональних будівель та інших об'єктів із високою інтенсивністю відвідування. Попри те, що значна частина водіїв звикла користуватися традиційними паркувальними майданчиками, активне впровадження сучасних технологій сприяє підвищенню популярності автоматизованих паркінгів.

Зростання інтересу до автоматизованих парковок зумовлене тим, що автоматизація основних процесів паркування забезпечує оперативний і зручний в'їзд та виїзд транспортних засобів, підвищує пропускну здатність об'єктів і загалом покращує комфорт користування паркінгом. Окрім цього, для керуючих компаній впровадження автоматизованих систем є економічно доцільним, оскільки дозволяє мінімізувати ризики зловживань і помилок з боку обслуговуючого персоналу.

Сучасні автоматизовані парковки зазвичай складаються з комплексу взаємопов'язаних технічних засобів, до яких належать стійки в'їзду та виїзду, автоматичні шлагбауми, інформаційні табло, засоби візуальної навігації, платіжні термінали, касові модулі та системи відеоспостереження. Усі зазначені компоненти інтегруються в єдину інформаційну систему, що забезпечує їх узгоджену взаємодію та ефективне функціонування [6].

Автоматизовані парковки спрямовані на розв'язання низки ключових завдань, серед яких основними є автоматизація процесів паркування, забезпечення зручних і сучасних способів оплати послуг, а також формування

аналітичних звітів щодо кількості транспортних засобів, тривалості їх перебування на території паркінгу та обсягів здійснених платежів [6].

До основних переваг автоматизованих паркінгових систем належить передусім економічна доцільність їх упровадження. Застосування автоматизації у багатьох випадках дозволяє повністю відмовитися від утримання операторів, які на неавтоматизованих парковках здійснюють ручне управління процесами в'їзду, виїзду та оплати. Це суттєво зменшує експлуатаційні витрати та підвищує рентабельність об'єкта.

Важливою перевагою є також підвищення конкурентоспроможності паркінгу. Можливість швидко та безперешкодно припаркувати автомобіль у безпосередній близькості до місця призначення стає суттєвим чинником вибору для користувачів, особливо в періоди несприятливих погодних умов, коли тривалі піші переходи є небажаними.

Автоматизовані системи паркування сприяють більш ефективному використанню паркувального простору. На відміну від неавтоматизованих майданчиків, які часто зайняті таксі, працівниками прилеглих офісів або мешканцями навколишніх будинків, автоматизовані паркінги дозволяють забезпечити пріоритетне обслуговування цільових відвідувачів. Це зменшує нераціональне використання місць і підвищує загальну ефективність експлуатації паркінгу.

Ще однією суттєвою перевагою є підвищений рівень контролю та прозорості фінансових операцій. За статистичними даними, на платних, але не автоматизованих парковках за участі операторів власники можуть не доотримувати до 35 % виручки. Використання автоматизованих систем мінімізує людський фактор, знижує ризики зловживань та забезпечує точний облік платежів [6].

Крім того, автоматизовані парковки підвищують рівень безпеки. В умовах соціально-економічної нестабільності, коли спостерігається зростання рівня злочинності, важливим завданням паркінгових систем є запобігання угону

транспортних засобів та несанкціонованому доступу. Інтеграція систем відеоспостереження, контролю доступу та ідентифікації транспортних засобів дозволяє суттєво зменшити відповідні ризики та підвищити довіру користувачів [6].

Сучасні системи управління паркуванням умовно поділяються на дві основні категорії: напіваавтоматичні та автоматичні. Напіваавтоматичні паркінги, залежно від організації процесів, можуть бути реалізовані з однією або двома стійками обслуговування.

Напіваавтоматична парковка з однією стійкою є найбільш поширеним і економічно доступним варіантом. Така система передбачає обов'язкову присутність обслуговуючого персоналу, зокрема касира, який зазвичай розміщується безпосередньо на виїзді з території паркінгу. Принцип її функціонування полягає в тому, що під час в'їзду водій отримує паркувальний талон зі штрих-кодом, після чого автоматично відкривається в'їзний шлагбаум і система реєструє факт заїзду транспортного засобу. Під час виїзду водій передає талон касиру, який за допомогою сканера зчитує інформацію, а система автоматично обчислює вартість користування паркінгом. Після здійснення оплати касир надає дозвіл на виїзд, відкриваючи шлагбаум [6].

Напіваавтоматичний варіант із двома стійками застосовується у випадках, коли касове обслуговування організоване поза зоною виїзду. У такій системі після оплати послуг водій отримує спеціальний виїзний талон, який прикладається до зчитувального пристрою, встановленого біля виїзного шлагбаума. Лише після підтвердження оплати система автоматично дозволяє виїзд транспортного засобу.

Автоматична парковка, на відміну від напіваавтоматичних рішень, є повністю автономною системою управління паркінгом. Усі основні операції — в'їзд, оплата та виїзд — виконуються водіями самостійно без участі обслуговуючого персоналу. Така система включає в'їзні та виїзні стійки, а також спеціальні термінали самообслуговування, що забезпечують можливість

незалежної оплати послуг паркування. Автоматизована система в режимі реального часу фіксує всі події, пов'язані з рухом транспортних засобів, та формує детальну фінансову і статистичну звітність [7].

В інформаційних системах управління паркуванням застосовуються різні способи ідентифікації транспортних засобів. Найпростішим і найбільш поширеним методом є ідентифікація за паркувальним талоном зі штрих-кодом, який водій отримує під час в'їзду на територію паркінгу. Відповідна інформація зі штрих-коду реєструється в базі даних системи, а під час виїзду талон зчитується сканером, на основі чого автоматично визначається вартість користування послугами паркування.

Більш функціональним підходом є використання безконтактних RFID-карток стандарту Mifare. На відміну від ідентифікації за паперовими талонами, такий метод дозволяє реалізувати додаткові можливості, зокрема впровадження гнучких тарифних і дисконтних програм, що можуть діяти на території паркінгу або торговельно-розважального комплексу.

Найбільш технологічно розвиненим варіантом є ідентифікація транспортного засобу за його державним номерним знаком. Для цього в'їзні та виїзні стійки обладнуються відеокамерами, які здійснюють автоматичне розпізнавання номерів. У разі, якщо зчитування номерного знаку з певних причин є неможливим, система в автоматичному режимі видає водієві паркувальний талон зі штрих-кодом як резервний спосіб ідентифікації [7].

Жоден із розглянутих підходів до побудови систем управління паркінгом не забезпечує можливості попереднього інформування водіїв про наявність вільних паркувальних місць. Унаслідок цього автомобілісти змушені витратити значний час на пошук місця для стоянки, здійснюючи додаткові поїздки та долаючи зайві відстані. Така ситуація призводить до підвищеного навантаження на міську дорожню інфраструктуру, що негативно позначається на пропускній здатності вулично-дорожньої мережі, безпеці дорожнього руху та загальній швидкості пересування транспортних потоків.

З огляду на зазначене, актуальним технічним завданням є розширення функціональних можливостей інформаційної системи обслуговування автопаркінгу шляхом упровадження механізмів попереднього визначення та бронювання паркувальних місць. Для реалізації такого підходу доцільним є інтегрування веб орієнтованого додатку до складу інформаційної системи паркінгу. У результаті водій отримує можливість заздалегідь переглянути інформацію про наявність вільних місць за обраною адресою, що дозволяє зекономити час і зменшити витрати пального. За умови наявності вільного місця система надає можливість його тимчасового бронювання з подальшим використанням у визначений проміжок часу [7].

Системи автоматичного паркування являють собою спеціалізовані інженерні комплекси, призначені для багаторівневого вертикального розміщення транспортних засобів з максимально раціональним використанням доступного простору. Завдяки особливостям конструкції та застосуванню автоматизованих механізмів такі системи забезпечують переміщення автомобіля від зони в'їзду до паркувального місця без безпосередньої участі водія. У науковій та технічній літературі подібні рішення зустрічаються також під різними найменуваннями, зокрема:

- роботизовані гаражі;
- системи автоматизованого паркування транспортних засобів;
- автоматичні паркінги;
- автоматизовані системи зберігання та пошуку автомобілів;
- механізовані стоянки [7].

Функціонування системи автоматичного паркування здійснюється у декілька етапів. На початковому етапі водій під'їжджає до зони приймання автомобіля, після чого пасажир залишає транспортний засіб. Далі спеціалізовані механізми системи в автоматичному або напівавтоматичному режимі виконують транспортування автомобіля до вільного паркувального місця.

Застосування таких технологій є доцільним насамперед в умовах обмежених міських територій, оскільки вони сприяють підвищенню щільності розміщення транспортних засобів, зростанню ефективності використання простору та покращенню рівня комфорту для користувачів [7].

1.2. Огляд аналогів

Одним із аналогів проєктованої системи є автоматизована система паркування **LOT PARKING**. Дане програмно-апаратне рішення орієнтоване на забезпечення автоматичного контролю процесів в'їзду та виїзду транспортних засобів, а також на автоматизацію розрахунків за надані паркувальні послуги. Система дозволяє мінімізувати участь персоналу в обслуговуванні паркінгу та підвищити ефективність управління паркувальним простором.

До основних переваг автоматизованої системи **LOT PARKING** належать:

- автоматизований механізм приймання та обробки платежів;
- забезпечення фінансового контролю та обліку операцій;
- можливість моніторингу подій у режимі реального часу;
- підвищений рівень безпеки експлуатації паркінгу;
- зручність і комфорт для користувачів паркувальних послуг.

Автоматизована система паркування **LOT PARKING** призначена для розв'язання комплексу функціональних завдань, пов'язаних з ефективним управлінням паркувальним простором. Зокрема, система забезпечує автоматизований контроль процесів в'їзду та виїзду транспортних засобів на територію стоянки, а також реалізує автоматизацію розрахунків за паркувальні послуги із застосуванням квитків зі штрих-кодом і безконтактних карт доступу.

Крім того, **LOT PARKING** надає можливість здійснювати зручний фінансовий контроль та аналіз економічних показників діяльності паркувального комплексу, забезпечує безперервний моніторинг подій у режимі реального часу та сприяє підвищенню рівня комфорту користування паркувальними послугами для клієнтів [8].

Автоматизована система паркування **LOT PARKING** має модульну архітектуру та включає такі основні компоненти: в'їзну та виїзну стійки, платіжний термінал, серверну частину системи, автоматизоване робоче місце адміністратора, робоче місце касира, а також комплекс спеціалізованого програмного забезпечення (рис. 1.1). За потреби до складу системи можуть додатково входити в'їзні та виїзні шлагбауми, що розширює функціональні можливості паркувального комплексу.



Рис. 1.1. Розроблення проєкту паркувального комплексу на основі системи **LOT PARKING**

Під час в'їзду на територію паркінгу транспортний засіб потрапляє в зону дії першої в'їзної індукційної петлі, після чого система ідентифікує наявність автомобіля та надає дозвіл на в'їзд. Для отримання паркувального квитка водій натискає кнопку на в'їзній стійці. Після отримання картки або квитка зі штрих-кодом система автоматично відкриває шлагбаум. У процесі руху автомобіль послідовно перетинає першу та другу індукційні петлі, а після виходу з зони дії другої петлі формується команда на закриття шлагбаума. На дисплеї в'їзної стійки відображаються інформаційні повідомлення та підказки для водія. У разі виникнення запитань користувач має можливість скористатися голосовим зв'язком із адміністратором системи [8].

Оплата послуг паркування може здійснюватися як через автоматичний платіжний термінал, так і безпосередньо через касира. Для цього клієнт пред'являє картку або квиток зі штрих-кодом, отриманий під час в'їзду. На основі зафіксованого часу заїзду система автоматично обчислює вартість паркування відповідно до встановлених тарифів. Після внесення оплати інформація про платіж реєструється в системі, а користувачу надається визначений часовий інтервал для виїзду з території паркінгу [8].

Під час виїзду з території паркінгу транспортний засіб потрапляє в зону дії першої виїзної індукційної петлі, що ініціює процедуру завершення паркування. Для підтвердження права на виїзд користувач вставляє картку або квиток зі штрих-кодом у виїзну стійку. Система здійснює перевірку факту оплати на підставі поданого ідентифікатора, і у разі позитивного результату автоматично подає команду на відкриття виїзного шлагбаума. Після того як автомобіль залишає зону дії другої виїзної індукційної петлі, система формує сигнал на закриття шлагбаума. Функціонування системи передбачає періодичний обмін даними із сервером, що зумовлює необхідність наявності стабільного та безперервного каналу зв'язку між усіма компонентами системи [8].

Ще одним аналогом розглядуваної системи є автоматизація паркувальних місць із використанням технології RFID. RFID-системи є ефективним інструментом автоматизованого управління процесами паркування, однак на сучасному ринку такі рішення представлені обмежено, оскільки перебувають на етапі активного розвитку. Водночас актуальність застосування RFID-технологій у сфері паркування постійно зростає, адже вони спрямовані на розв'язання низки актуальних проблем, з якими стикаються власники та користувачі паркувальних комплексів [9].

До основних проблем, характерних для організації паркувальних процесів, належать надмірна тривалість процедур в'їзду та виїзду транспортних засобів, відсутність ефективного контролю кількості доступних паркувальних місць, а також помилки й затримки, зумовлені впливом людського фактору. Окрім цього,

суттєвим недоліком є відсутність точного обліку подій, пов'язаних із заїздом та виїздом автомобілів. Застосування RFID-обладнання на основі технології UHF дозволяє оптимізувати контрольно-пропускні процеси та підвищити загальну ефективність функціонування паркінгу [9].

Для реалізації автоматизованого контролю з використанням RFID-технологій застосовується комплекс із чотирьох основних компонентів, а саме: UHF-зчитувач дальньої дії, контролер системи контролю та управління доступом, UHF-мітки для ідентифікації транспортних засобів, а також спеціалізоване програмне забезпечення, яке забезпечує обробку та зберігання даних [9].

Функціонування RFID-системи на паркінгу ґрунтується на використанні спеціальних ідентифікаційних міток, які закріплюються на транспортному засобі. Як RFID-мітки можуть застосовуватися наклейки, що встановлюються під лобове скло, корпусні мітки, розміщені поблизу номерного знаку, або пластикові карти з вбудованим чіпом дальньої дії. Під час наближення автомобіля до контрольно-пропускного пункту мітка потрапляє в зону дії зчитувача на відстані до 10 метрів, після чого відбувається автоматичне зчитування унікального ідентифікаційного коду. Отримані дані передаються до контролера, який функціонує під керуванням спеціалізованого програмного забезпечення.

Програмне забезпечення RFID-системи забезпечує адміністрування доступу, збір та обробку аналітичної інформації про переміщення транспортних засобів через контрольно-пропускні пункти, а також дозволяє оперативно керувати правами доступу для кожного користувача системи та виконувати інші функції управління [9]. Впровадження автоматизованої RFID-системи в'їзду на паркінг сприяє скороченню чисельності обслуговуючого персоналу, зокрема охоронців, та суттєво зменшує вплив людського фактору на роботу паркувального комплексу.

Крім того, система може бути доповнена функцією «антидубль», яка унеможливує одночасний доступ на територію паркінгу транспортних засобів з однаковими унікальними ідентифікаторами RFID-міток. Повторний в'їзд автомобіля з відповідною міткою стає можливим лише після фіксації виїзду шляхом зчитування ідентифікатора на виїзному зчитувачі. Оскільки RFID-мітки, що встановлюються на лобове скло, мають властивість саморуйнування при спробі демонтажу, фальсифікація факту виїзду без фактичного переміщення автомобіля є неможливою [9].

До основних переваг застосування RFID-технології в системах паркування належить можливість швидкого та безконтактного розпізнавання транспортних засобів. Ідентифікація автомобіля може здійснюватися без зупинки або виходу водія з салону, зокрема й у разі використання пластикових карт з RFID-чіпом. Така технологія забезпечує точну фіксацію часу в'їзду та виїзду автомобілів, що дозволяє здійснювати коректний облік паркувальних подій.

Крім того, RFID-системи гарантують високий рівень захисту від несанкціонованого доступу шляхом автоматичного блокування проїзду транспортних засобів, які не мають відповідних прав доступу. Суттєвими перевагами є також простота використання, надійність та безпека функціонування системи, а також легкість встановлення й експлуатації, що мінімізує потребу у залученні людських ресурсів [9].

Третім аналогом розглядуваної системи є автоматизація процесів паркування з використанням технології UHF. Організація автоматизованого паркінгу сьогодні є важливою складовою інфраструктури сучасних житлових комплексів. Основні вимоги користувачів до паркування зводяться до забезпечення збереження транспортного засобу, швидкого та зручного проїзду з метою економії особистого часу, а також гарантії наявності вільних паркувальних місць.

Традиційні системи доступу до паркінгу часто потребують виконання додаткових дій, які можуть створювати незручності для водіїв, зокрема

необхідність відкривати вікно автомобіля або очікувати дозволу від охоронця. На зміну таким рішенням, що характеризуються високою вартістю радіо брелків, потребою в регулярній заміні елементів живлення та недостатнім рівнем безпеки карт доступу, приходять автоматизовані системи в'їзду, побудовані на основі технології ультрависокочастотної ідентифікації (UHF) (рис. 1.2) [10].

Система на основі технології UHF складається з наступних основних компонентів:

- **мітки ALN-9662**, які виконують роль унікального ідентифікатора транспортного засобу;
- **зчитувачі**, що фіксують присутність мітки та передають дані до контролера [10].

Принцип роботи системи полягає у використанні спеціальної мітки, яку можна реалізувати у різних форматах: наклеєною під лобове скло автомобіля, встановленою під номерний знак або інтегрованою в стандартну пластикову картку.



Рис. 1.2. Проектування автоматизованого паркінгу з використанням RFID

Монтаж мітки безпосередньо на автомобіль є найбільш ефективним рішенням, оскільки в такому випадку ідентифікація транспортного засобу відбувається практично без участі водія. Коли мітка потрапляє в зону дії зчитувача, останній передає її унікальний ідентифікаційний номер на контролер,

який функціонує під керуванням спеціалізованого програмного забезпечення. Контролер, обробивши отриманий номер, приймає рішення про надання або відмову в доступі до паркувальної зони [10]. Дальність зчитування міток за допомогою цієї технології становить до 7 метрів (рис. 1.3).



Рис. 1.3. Мітка ALN-9662

За допомогою програмного забезпечення системи можна відстежувати час в'їзду та виїзду кожного транспортного засобу, обмежувати доступ лише до певних зон паркінгу, а також реалізовувати інші функції адміністрування. У випадках, коли один користувач має декілька ідентифікаторів для одного паркувального місця (наприклад, пластикові картки), застосовується функція «Анти-дубль». Вона запобігає доступу на територію паркінгу міткам з однаковим ідентифікаційним номером до тих пір, поки цей номер не буде зафіксований на виїзному зчитувачі. При цьому контроль забезпечує факт фактичного виїзду автомобіля через шлагбаум, тобто неможливо пройти пішки з картою через виїзд та використати її повторно для іншого автомобіля [10].

ParkMyCarNow – це програмно-апаратна платформа, призначена для ефективного управління паркувальною інфраструктурою. Система надає користувачам широкий вибір методів доступу до паркінгу, що дозволяє водіям обирати найбільш зручний для себе спосіб:

- через мобільний додаток;
- за допомогою дзвінка на номер шлагбаума;
- із застосуванням RFID-карт доступу;
- за технологією розпізнавання номерних знаків;
- через централізовану диспетчеризацію.

Система здатна визначати наявність вільних паркувальних місць у заданому регіоні та рекомендувати їх розташування на інтерактивній схемі паркінгу в режимі online. Для гостей готелів вона надає можливість використовувати місця біля готелю, а також на інших платних автостоянках і міських паркінгах. Крім того, система контролює дотримання правил користування паркувальними місцями.

Для ефективного та компактного розміщення автомобілів на території майбутньої парковки необхідно ретельно продумати розташування всього комплексу обладнання. Оптимальна організація паркінгу можлива за умови застосування сучасних технологій. Компанія **Intelpol** пропонує систему автоматизованого паркування **ParkMyCarNow**, яка підтримує декілька способів доступу на паркінг: технологію RFID UHF, мобільний додаток, систему розпізнавання номерних знаків та інші. Інтеграція спеціалізованого обладнання з програмним забезпеченням дозволяє максимально ефективно використовувати паркувальні місця, підвищуючи комфорт для водіїв і створюючи додаткову перевагу в іміджі підприємств та забудовників.

У поєднанні з автоматизованими паркувальними системами обладнання «**RFID паркінг**» забезпечує надійний, безпечний та практично безпомилковий комплекс для ідентифікації, обліку та контролю транспортних засобів. Воно дозволяє вирішувати ці завдання значно ефективніше, вигідніше та зручніше, ніж традиційні системи паркування, оснащені відеокамерами на в'їзді. Відеоспостереження при цьому виконує допоміжні функції, наприклад, забезпечує безпеку території або фіксує протиправні дії.

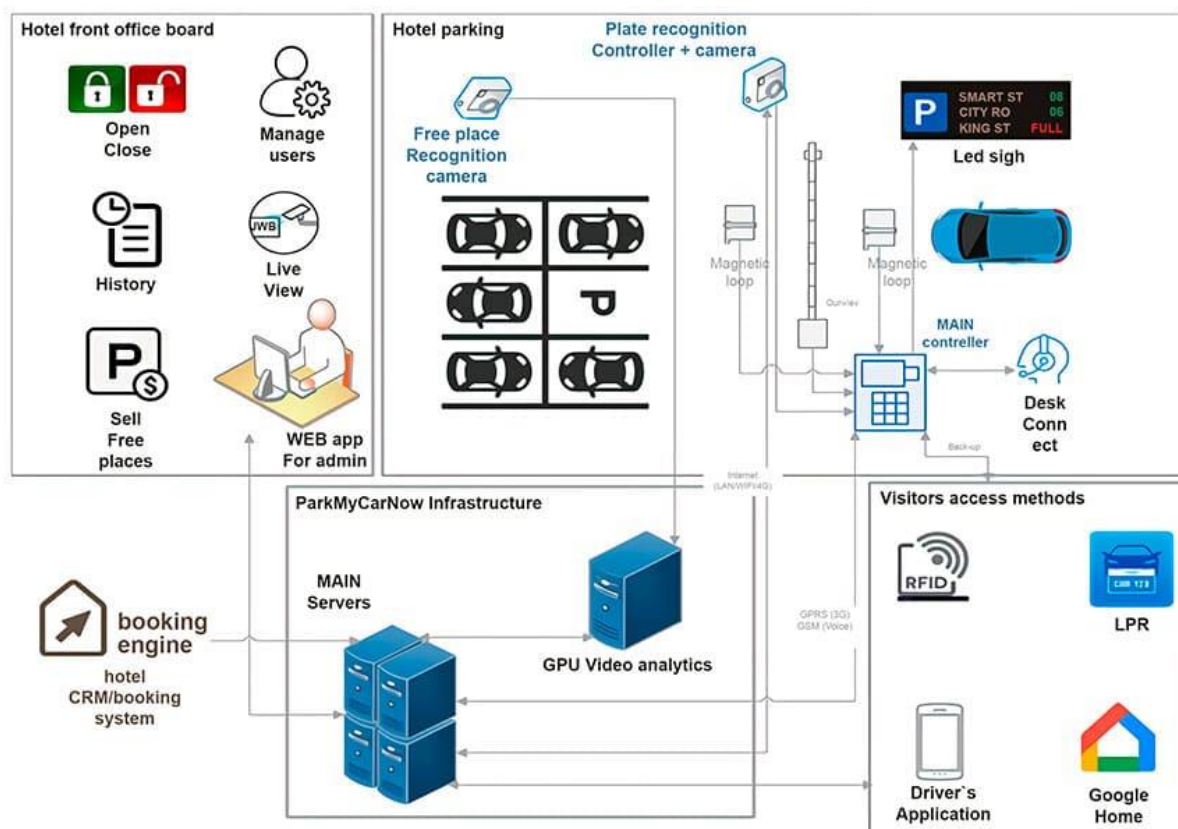


Рис. 1.4. Архітектура рішення ParkMyCarNow

Автоматизовані платні парковки являють собою спеціалізоване обладнання, яке функціонує за наступним алгоритмом:

1. На в'їзді транспортний засіб відвідувача ідентифікується, а його дані перевіряються з базою інформації.
2. У разі збігу визначених параметрів (номер телефону, RFID-мітка, номерний знак автомобіля) відбувається спрацювання реле та автоматичне відкриття шлагбаума.
3. Ця подія фіксується в журналі обліку подій.

Обладнання встановлюється на в'їзді та інтегрується із шлагбаумом, воротами або ролетами, налаштовується відповідно до технічних стандартів і забезпечує надійну роботу, створюючи комфортні умови для водіїв. Контроль доступу здійснюється через центральний контролер, який порівнює дані потенційного відвідувача, передані через GSM-канал, з інформацією на хмарному сервері. У системі передбачено реєстрацію користувачів, об'єднання

їх у групи та ведення обліку для кожного транспортного засобу. Крім того, автоматизована система паркування підтримує журнал подій, у якому фіксуються всі в'їзди, виїзди та час перебування автомобілів на території паркінгу.

Під час проведення пошуку аналогів були виявлені наступні системи зі схожим функціоналом: **KyivSmartCity, Barking.**

Розглянемо кожну з них більш детально. **KyivSmartCity** – це велика електронно-цифрова система, яка об'єднує в одному місці: рахунки, штрафи, транспортні новини, оголошення і багато інших послуг. Раніше коли потрібно було кудись йти по адміністративних послугах, зараз можна зробити в декілька дії на телефоні. Серед широкого спектру функціоналу є можливість бронювати паркомісця. Пошук вільних паркомісць відбувається способом введення номера парковки, а сама система пропонує їх з наявних, виводячи, як перелік адрес. Бронювання відбувається почасово. Також для бронювання необхідно вказувати номерний знак автомобіля. Дана система має функцію онлайн оплати. Так як програма орієнтована не тільки на бронювання паркомісць, то вона має обмежений функціонал: не можна здійснювати пошук найближчих парковок з доступним паркомісцями та система не веде облік зайнятих паркомісць. Найбільшим недоліком даної програми є те, що вона має локальний характер і покриває тільки парковки міста Києва.

Barking на відмінно від KyivSmartCity орієнтований тільки на здачу і оренду паркомісць, відповідно має набагато більший функціонал, і обсяг парковок. Дана програма дозволяє не тільки бронювати паркомісця, але й здавати в оренду власні, на той час коли вони вільні. Будь-яка особа може здати в оренду паркомісце, якщо вона є її власником.

Порівняльна характеристика аналогів з наявною системою

| | Дана робота | KyivSmartCity | Barking |
|--|--------------------|----------------------|----------------|
| Онлайн оплата | Так | Так | Так |
| В'їзд на парковку за QR-кодом | Так | Ні | Так |
| Пошук найближчих парковок в певному радіусі | Так | Ні | Ні |

1.3. Аналіз предметного середовища**1.3.1. Вимоги до функціональних характеристик**

Інформаційна система повинна виконувати роль комунікатора між клієнтом та власником парковки.

Система має виконувати наступні функції для всіх типів користувачів:

- реєстрація в системі:
 1. система повинна надати клієнту створити приватний акаунт, який дозволить виконувати бронювання паркомісць;
 2. система повинна надати власнику парковки створити комерційний акаунт, котрий дозволить здавати в оренду паркомісця;
- авторизація в системі;
- редагування персональної інформації.

Для типу користувача «Клієнт» система повинна виконувати наступні функції:

1. ведення власними банківськими картками;
2. перегляд історії замовлень за вказаний період;
3. генерація QR-коду для кожного замовлення, котрий використовується при в'їзді на парковку;
4. перегляд історії штрафів по статусам;
5. оплата штрафів;

6. пошук вільних місць на обраній парковці в заданий проміжок часу;
7. пошук найближчих парковок з вільними місцями;
8. бронювання паркомісць.

Система повинна мати функцію автоматичного розсилання електронних листів власникам парковок, про необхідність сплати відсотку від доходу, котрий отриманий за допомогою даної системи. Дані листи повинні надсилатися першого числа кожного місяця всім власникам парковок, дохід яких перевищує нуль.

1.3.2. Вхідні дані

Вхідні дані для роботи системи надходять з декількох джерел, а саме від:

- клієнтів;
- власників парковок;
- адміністраторів.

Варто розглянути детально, які саме дані надходять від вищезгаданих джерел.

Дані, що надходять від клієнтів. При реєстрації клієнт повинен вказати наступні параметри:

- ім'я;
- прізвище;
- електронну пошту;
- номер телефону.

Вищезгадані дані, окрім електронної пошти, можуть бути змінені у власному кабінеті при необхідності.

Для того, щоб виконати пошук вільних паркомісць, клієнт мусить вказати наступні дані:

- парковку, на якій він хоче зупинитися;
- дату та час в'їзду на парковку;
- дату та час виїзду з парковки.

Безпосередньо для здійснення бронювання, клієнт повинен додати принаймні одну банківську карту вказавши наступні параметри:

- номер;
- термін дії;
- CVV2 код.

Дані, що надходять від власників парковок. Під час реєстрації власник парковок має вказати наступний перелік параметрів:

- назву компанії чи фізичної-особи підприємця;
- адресу реєстрації;
- індивідуальний податковий номер;
- електронну пошту;
- корпоративний номер телефону.

Після виконання реєстрації, власник парковок повинен завантажити наступний перелік документів:

- копію свідоцтва про державну реєстрацію;
- копію паспорта (тільки для фізичних осіб);
- копію ідентифікаційного коду.

Для того, щоб додати нову парковку в систему, необхідно вказати такі характеристики:

- адреса розташування;
- тариф на паркування;
- розмір штрафу;
- копію технічного паспорта парковки;
- перелік номерів паркомісць.

Дані, що надходять від адміністраторів. Адміністратор виконує перевірку документів на парковки і по її результатам змінює статус парковки, після чого клієнти можуть бронювати на ній паркомісця.

1.3.3. Вихідні дані

Вихідними даними роботи системи є бронювання та штрафи.

Бронювання представлені наступним переліком параметрів:

- адреса парковки;
- статус;
- номер паркомісця;
- час в'їзду на парковку;
- час виїзду з парковки;
- фактичний час виїзду з парковки;
- загальна сума за перебування на парковці.

Штрафи формуються в тому випадку, коли клієнт виїхав з парковки пізніше, ніж було заплановано. Вони мають такі параметри:

- адреса парковки;
- номер паркомісця;
- запланований час виїзду з парковки;
- фактичний час виїзду з парковки;
- коментар.

1.3.4. Опис процесу діяльності

В даній роботі розглядається автоматизація процесу бронювання паркомісць.

До автоматизації, процес пошуку вільних паркомісць відбувається досить незручно для водіїв. Адже коли вони планують кудись поїхати, їм спершу необхідно дізнатися, де вони зможуть припаркувати власний транспортний засіб. В момент, коли визначилися, то приїжджають на парковку, але може виявитися, що там не має вільних місць і необхідно знову повторювати дії з пошуку іншого місця. При успішному знаходженні парковки і на ній є місця, водій отримує талон на паркування і займає вільне місце. По завершенню стоянки, необхідно оплатити час паркування. На рис. 1.5. наведена діаграма діяльності процесу пошуку парковок до автоматизації.

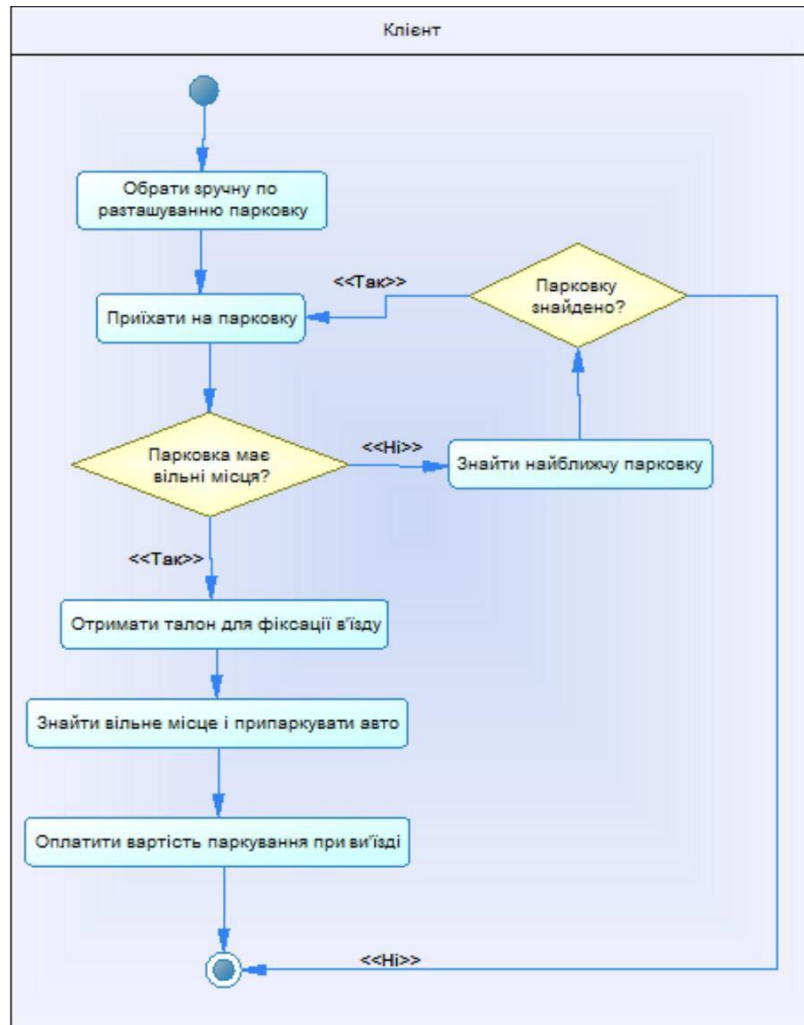


Рис. 1.5. Діаграма діяльності процесу пошуку парковок до автоматизації

Після автоматизації, процес пошуку вільних паркомісць буде простішим і не потребуватиме зайвих переїзді в їх пошуку. Також, можна буде здійснювати бронювання паркомісць. Водій має бути зареєстрованим в системі. На головній сторінці він матиме можливість вибрати парковку, за вказаною адресою, і вказати проміжок часу, на який хоче забронювати паркомісце. Система виконає пошук вільного місця за вказаними параметрами і відбудеться перехід на сторінку оформлення бронювання. В разі, якщо не буде знайдено вільного паркомісця за вказаними параметрами, система запропонує здійснити пошук вільних місць на сусідніх парковках. На сторінці оформлення бронювання, клієнту необхідно буде обрати банківську картку і підтвердити бронювання.

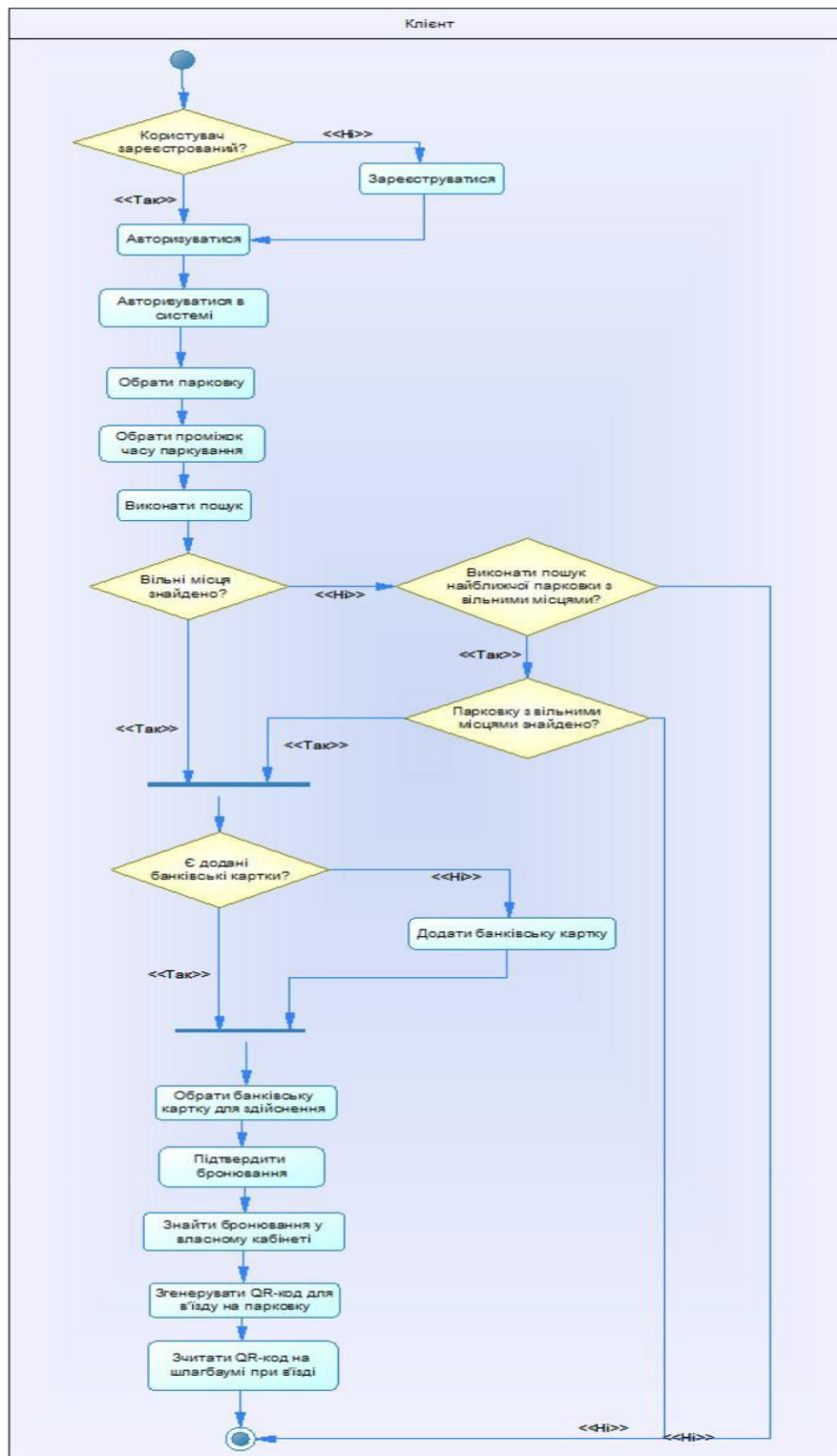


Рис. 1.6. Діаграма діяльності процесу пошуку парковок після автоматизації

1.3.5. Опис функціональної моделі

Перед початком проєктування діаграми використання потрібно визначити перелік акторів, що дозволить нам визначити перелік дій, які вони будуть виконувати. Нижче наведено список акторів з їх описом:

Клієнт. Зареєстрований користувач веб-сервісу, котрий використовує його для бронювання паркомісць заздалегідь.

Тепер визначимо дії, які можуть виконувати актори системи та відобразимо їх за допомогою діаграми варіантів використання (рис. 1.7).

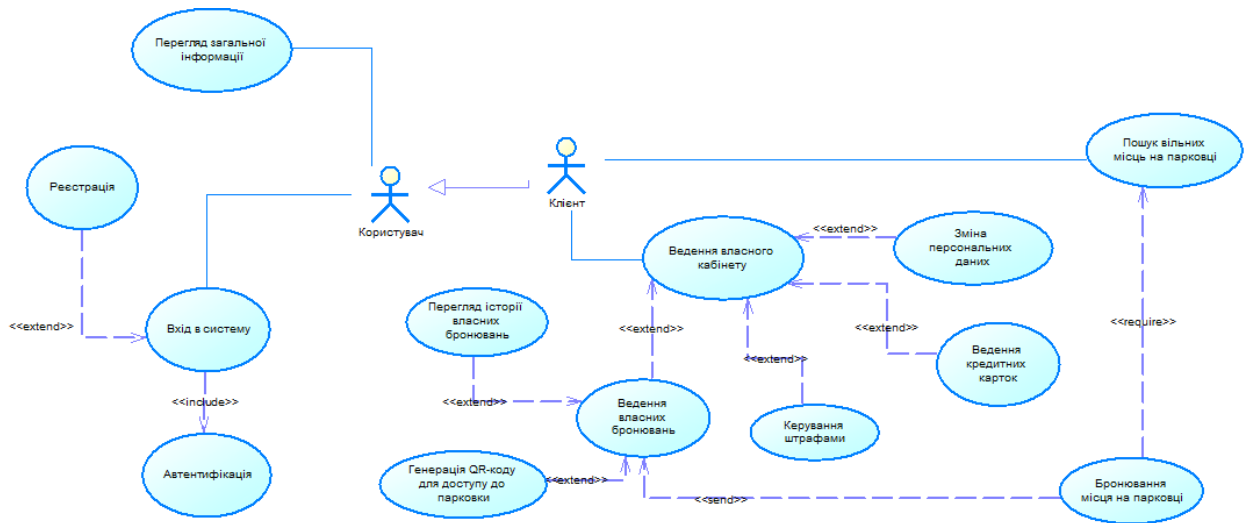


Рис. 1.7. Діаграма варіантів використання

1.4. Дослідження математичного забезпечення

1.4.1. Змістова постановка задачі

Процес бронювання паркомісць клієнтом можна розділити на декілька підзадач:

- пошук вільних місць на обраній парковці на вказаний проміжок часу;
- пошук вільних місць на найближчих парковках, в тому випадку, коли на обраній парковці немає вільних паркомісць;
- створення бронювання.

Виходячи зі списку вищезгаданих підзадач, можемо сформулювати низку математичних задач, котрі забезпечать якісну роботу системи.

Задача пошуку найближчих парковок умовно розділяється на дві підзадачі. Спочатку необхідно знайти всі парковки, котрі знаходяться в радіусі 2000 метрів до обраної. Після того, як знайдено парковки в заданому радіусі, необхідно знайти найкоротший шлях між обраною та кожною з тих, котрі були знайдені.

1.4.2. Математична постановка задачі

1.4.2.1. Задача пошуку відстані між точками по географічним координатам

Призначенням цієї задачі є знаходження прямої відстані між двома парковками. Кожна парковка має адресу розташування за допомогою якої можна з'ясувати географічні координати. Вхідними даними є:

- адреса з визначеними географічними координатами обраної клієнтом парковки;
- список адрес з визначеними географічними координатами всіх парковок системи.

Ця задача переслідує ціль знаходження переліку парковок, що знаходяться в радіусі 2000 метрів від тієї, де клієнт хотів би припаркувати власний транспортний засіб.

1.4.2.2. Задача пошуку найкоротшого шляху

Призначенням даної задачі є знаходження найкоротшого шляху між двома парковками за допомогою алгоритму Дейкстри. На вхід нам відомо:

- географічні координати розташування обраної клієнтом парковки;
- географічні координати парковок, що знаходяться в радіусі 2000 метрів.

Ціллю задачі є знаходження найближчої до обраної клієнтом парковки з вільними місцями на вказаний проміжок часу.

1.4.3. Обґрунтування методу розв'язання

Для розв'язку задачі 1.4.2.1 буде використовуватись формула гаверсинуса. Мінусом даної формули є те, що в її основі лежить припущення, що Земля має форму сфери. Таке припущення призводить до того, що похибка розрахунків становить 0.5%. В нашому випадку нас це задовольняє, адже більш важливим критерієм для нас є швидкодія, а за цим критерієм він найкращий.

Для розв'язку задачі 14.2.2 використовується класичний алгоритм Дейкстри. Існує велика кількість алгоритмів для знаходження найкоротшого шляху. Для знаходження шляху між двома вершинами графу найбільше підходить алгоритм Дейкстри, адже це його пряме призначення. Він є дуже простим в реалізації і має прийнятну швидкодію.

1.4.4. Опис методу розв'язання

1.4.4.1. Задача пошуку відстані між точками по географічним координатам

Формула гаверсинуса – дуже важлива у навігації, адже обчислити відстань між точками на сфері, за їхніми довготою, та широтою. Вона є окремим випадком загальної формули сферичної тригонометрії, закону гаверсінусів, відносно сторін та кутів сферичних трикутників зображених на рис. 1.8, де v і w це дві вказані точки, u – Північний полюс, d – шукана відстань, C - відстань за вказаною довготою, та c — це бажаний d/R (R – радіус Землі).

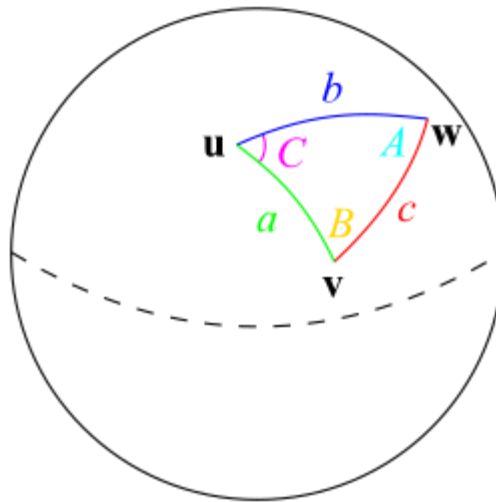


Рис. 1.8. Сферичний трикутник розв'язаний за теоремою гаверсинуса

Формула має вигляд:

$$\text{haversin}\left(\frac{d}{r}\right) = \text{haversin}(\varphi_2 - \varphi_1) + \cos(\varphi_1) \cos(\varphi_2) \text{haversin}(\lambda_2 - \lambda_1), \quad (1.1.)$$

де

$$\text{haversin}(\theta) = \sin^2\left(\frac{\theta}{2}\right) = \frac{1 - \cos(\theta)}{2}; \quad (1.2)$$

– d -відстань між двома точками;

- r – радіус сфери;
- φ_1, φ_2 – широта першої і другої точки відповідно;
- λ_1, λ_2 – довгота першої та другої точки відповідно.

Ліва частина рівняння $\frac{d}{r}$ – це центральний кут в радіанах, обов'язково широту і довготу потрібно перевести в радіани. Знайти d можна через застосування зворотнього гаверсінуса, або через арксинус:

$$d = r \operatorname{haversin}^{-1}(h) = 2r \arcsin(\sqrt{h}), \quad (1.3)$$

$$\text{де } h = \operatorname{haversin}\left(\frac{d}{r}\right). \quad (1.4)$$

Напишемо формули більш розгорнуто:

$$d = 2r \arcsin \left(\sqrt{\operatorname{haversin}(\varphi_2 - \varphi_1) + \cos(\varphi_1) \cos(\varphi_2) \operatorname{haversin}(\lambda_2 - \lambda_1)} \right) \quad (1.5)$$

$$d = 2r \arcsin \left(\sqrt{\sin^2 \left(\frac{\varphi_2 - \varphi_1}{2} \right) + \cos(\varphi_1) \cos(\varphi_2) \sin^2 \left(\frac{\lambda_2 - \lambda_1}{2} \right)} \right) \quad (1.6)$$

Обидві формули дають лише наближену відстань, адже Земля не ідеальна сфера. Також радіус Землі R варіюється в певних межах, на екваторі він більший ніж на полюсі на 1%, отже формула гаверсінуса не може бути гарантовано точнішою ніж 0.5%.

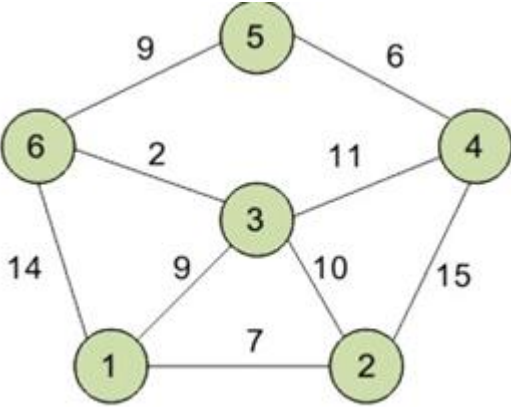
1.4.4.2. Пошук найкоротшого шляху за допомогою алгоритму Дейкстри

Алгоритм Дейкстри [2] – алгоритм на графах, розроблений нідерландським вченим Едсгером Дейкстрой в 1959 році. Знаходить найкоротший маршрут від однієї вершини до всіх інших. У нашому випадку, достатньо знаходити маршрут між двома заданими вершинами. Обов'язковою вимогою для роботи алгоритму, є те що відстань між вершинами (вага ребра) не має бути від'ємною.

Широкого застосування алгоритм набув при побудові маршрутів на картах.

У таблиці 1.2 наведено представлення графу в різному вигляді.

Форми представлення графу

| Маркований граф | Матриця відстаней |
|---|--|
|  | $\begin{pmatrix} 0 & 7 & 9 & 0 & 0 & 14 \\ 7 & 0 & 10 & 15 & 0 & 0 \\ 9 & 10 & 0 & 11 & 0 & 2 \\ 0 & 15 & 11 & 0 & 6 & 0 \\ 0 & 0 & 0 & 6 & 0 & 9 \\ 14 & 0 & 2 & 0 & 9 & 0 \end{pmatrix}$ |

У матричному вигляді, це буде квадратна матриця $n \times n$, де n – кількість вершин графу, а на перетині двох вершин стоїть відстань між ними, тобто вага ребра.

Розглянемо по кроках роботу алгоритму.

Ініціалізація.

Спочатку необхідно ініціалізувати всі відстані від стартової вершини. Для стартової вершини, відстань встановлюємо в 0, а для всіх інших встановити достатньо велике значення, адже поки що відстані до них невідомі. Також всі вершини позначаємо як невідвідані. На рисунку 1.9 зображено приклад вхідного ініціалізованого графу.

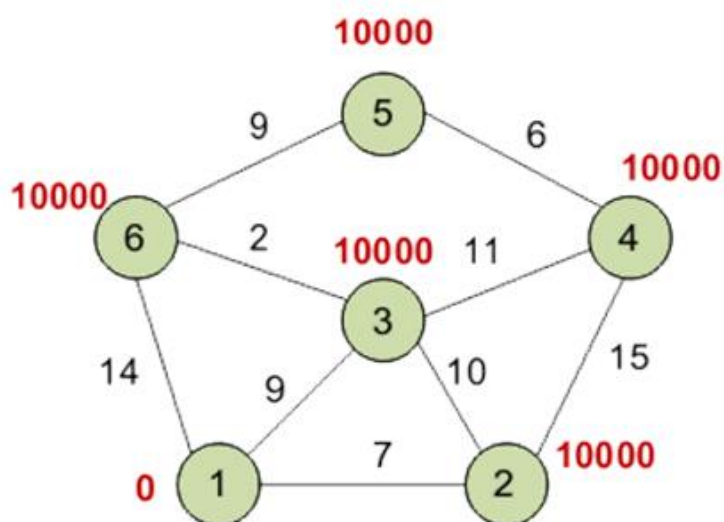


Рис. 1.9. Ініціалізований граф

Крок 1

Знайдемо відстань між вершинами 1 та 5. Для цього потрібно до мітки поточної вершини додати вагу ребра до відповідної вершини. Відстань від 1-ої вершини до 2-гої рівна $0 + 7 = 7$. Далі необхідно порівняти відстань до 2-гої вершини з його міткою і взяти найменше значення $\min(7, 10000) = 7$. Відстань до 2-гої вершини рівна 7 і це значення позначаємо, як мітка.

Аналогічно, необхідно знайти відстані до інших сусідніх вершин (вершини 3 і 6). Після знаходження відстаней, позначаємо поточну вершину відвіданою, як показано на рисунку 1.10.

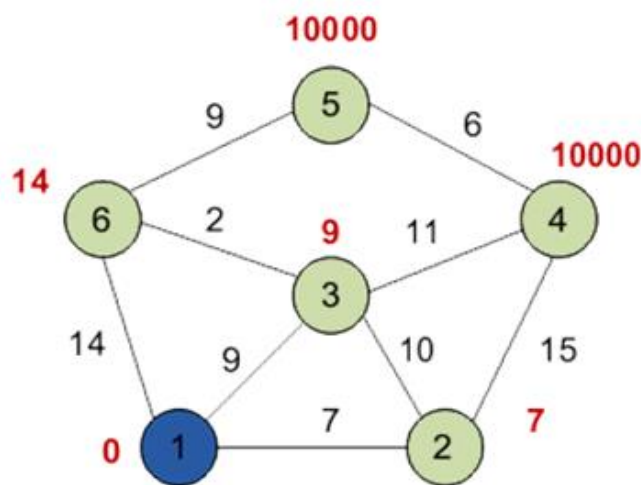


Рис. 1.10. Позначення вершину 1 відвіданою

Крок 2

Для другої вершини необхідно обчислити відстані до суміжних вершин 3 та 4, адже вершина 1 вже була відвідана на попередньому кроці.

Просумуємо відстань до 3-ої вершини $7 + 10 = 17$. Результируюча відстань буде рівна $\min(17, 9) = 9$. Маршрут до 3-ої вершини через другу буде довшим порівнянні з прямим переходом. Отже, беремо менше значення, тобто беремо прямий перехід з 1-ої до 3-ої і ставимо відповідну мітку, як показано на рисунку 1.11.

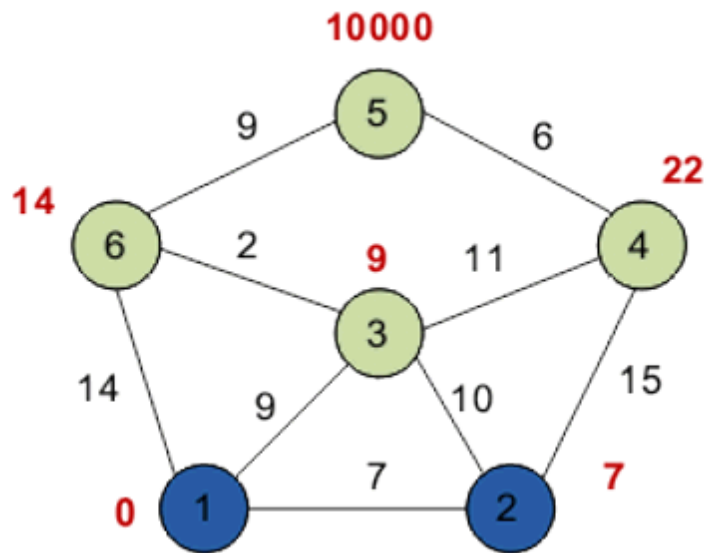


Рис. 1.11. Позначення вершини 2 відвіданою

Аналогічно шукаємо відстань до 4-ої вершини і позначаємо вершину 2 відвіданою.

Кроки 3 – 6

Кроки 3 – 6 аналогічні до 2-го кроку. Коли відвідаємо усі вершини, отримаємо граф, котрий зображено на рисунку 1.12.

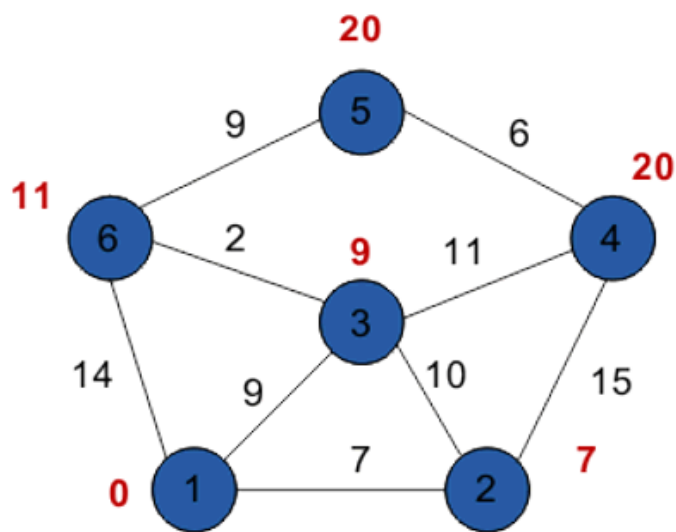


Рис. 1.12. Результуючий граф

Відстань від 1-ої вершини до вершини 5 рівна 20. Залишилося тільки визначити сам маршрут між даними вершинами. Маршрут шукатимемо з кінця. Він відстані будемо віднімати вагу ребра і якщо це значення буде співпадати з міткою тієї вершини, то її включаємо в маршрут.

Вершина 5 має сусідні вершини 4 і 6. Визначимо, яка з них належить найкоротшому маршруту. Для вершин 4 отримаємо $20 - 6 = 14$ і значення не співпадає з міткою 20, а для вершини 6, $20 - 9 = 11$, що відповідає мітці. Отже, найкоротший маршрут включає в себе вершину 6. Аналогічно шукаємо інші вершини, що входять в шлях, доки не дійдемо до вершини 1.

Результуючий маршрут буде наступним: 1 – 3 – 6 – 5. Його довжина дорівнює 20 одиниць.

Складність алгоритму рівна $O(n^2)$, адже на вхід подається матриця розмірності n^2 і необхідно перебрати всі елементи матриці.

Висновки до розділу

У цьому розділі було обґрунтовано доцільність розробки інформаційної системи пошуку та бронювання паркомісць. Було описано процеси діяльності до та після автоматизації, які були відображені в діаграмах діяльності. Були висунуті функціональні вимоги до системи, які наведені в діаграмі варіантів використання.

Проведено порівняльний аналіз розроблюваної системи та вже існуючих аналогів. Як результат порівняння, бачимо певні переваги розроблюваної системи над аналогами.

Також було сформульовано змістову та математичну постановки задачі. Задача пошуку найближчих парковок розділена на 2 підзадачі.

Було обґрунтовано методи розв'язку задачі. Також, в рамках даного розділу було наведено детальний опис використаних методів.

РОЗДІЛ 2. МОДЕЛЮВАННЯ ТА АНАЛІЗ ВЕБ ОРІЄНТОВАНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ ПОШУКУ ТА БРОНЮВАННЯ ПАРКОМІСЦЬ

2.1. Вимоги до технічного забезпечення

Сервер, на якому буде розгорнуто клієнтську частину застосунку повинен мати вказані параметри:

- процесор з частотою не менше 1.5 ГГц;
- RAM об'ємом 1 ГБ;
- 1 ГБ доступного місця на жорсткому диску.

Сервер, на якому буде розгорнуто API частину застосунку повинен мати вказані параметри:

- процесор з частотою не менше 1.5 ГГц;
- RAM об'ємом 1 ГБ;
- 200 Мб доступного місця на жорсткому диску.

Сервер, на якому буде розгорнуто базу даних повинен мати вказані параметри:

- процесор з частотою не менше 1.5 ГГц;
- RAM об'ємом 1 ГБ;
- 2 ГБ доступного місця на жорсткому диску.

Клієнт повинен мати встановлений один з веб-браузерів:

- Google Chrome 23+
- IE10+/Edge
- Firefox 21+
- Safari 6+
- Opera 15+

2.2. Архітектура програмного забезпечення

Програмний продукт було спроектовано на основі архітектурного шаблону Model-View-Controller (далі MVC). Цей шаблон має на меті розділення застосунку на три окремих частини: дані, користувацький інтерфейс

та бізнес логіка. Він широко використовується для проєктування веб-застосунків. На рис. 2.1. наведена схема взаємодії компонентів в шаблоні MVC.

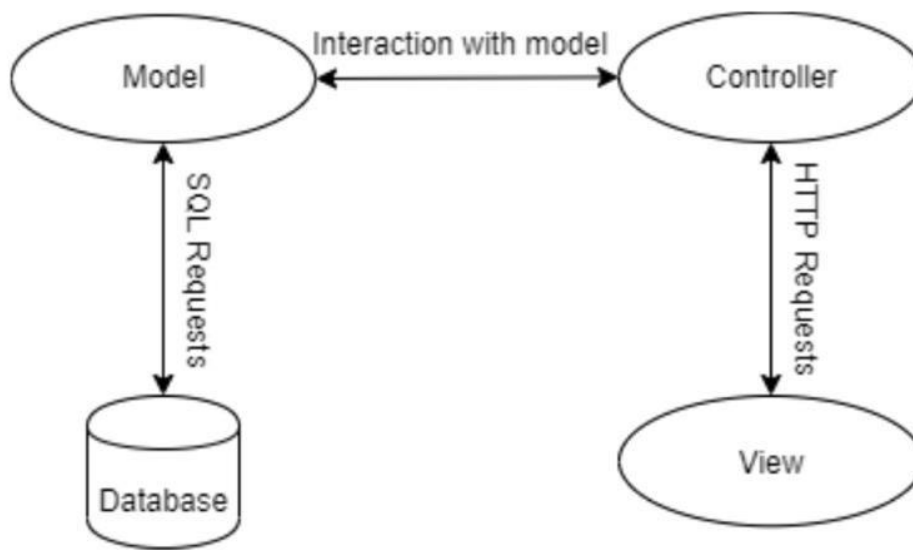


Рис. 2.1. Схема роботи шаблону MVC

Використання шаблону MVC дозволило розділити застосунок на 3 окремих частини, що в свою чергу дозволило працювати з ними не завдаючи шкоди іншим. Основними перевагами цього шаблону є:

- дотримання єдиної концепції системи;
- спрощення процесу відладки застосунку.

2.2.1. Логічне представлення статичної моделі структури програмного забезпечення

У мові UML для статичного представлення моделей систем використовуються діаграми класів та пакетів.

В уніфікованій мові моделювання (UML) діаграма пакетів показує залежності між пакетами, з яких складається певна модель. На рис. 2.2 представлена діаграма пакетів, яка описує структуру кодової бази серверної частини застосунку. Ця діаграма показує взаємодію різних пакетів проєкту, що дає розуміння про його структуру. Можна помітити, що багато пакетів мають однакові імена, це свідчить про те, що при розробці програмного продукту дотримувалася конвенція іменування.

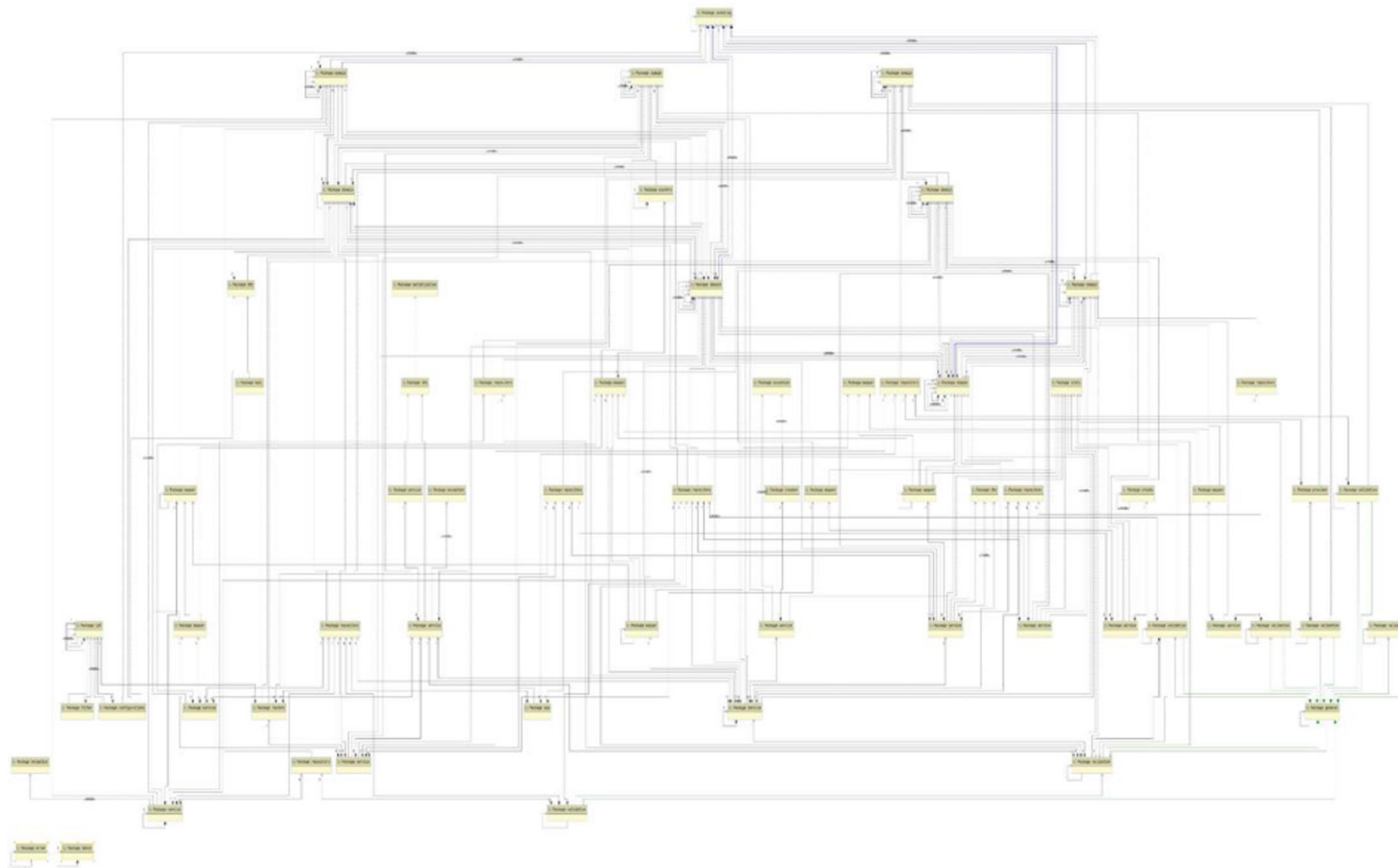


Рис. 2.2. Діаграма пакетів веб орієнтованої інформаційної системи пошуку та бронювання паркомісць

2.2.2. Діаграма компонентів

Діаграма компонентів – це UML діаграма, за допомогою якої відображаються компоненти, залежності та зв'язки між ними. Цей тип діаграми відображає залежності між компонентами ПЗ використовуючи структурні зв'язки. Дана діаграма представлена на рис. 2.3. З неї явно видно архітектурний шаблон MVC із зв'язками між компонентами. Також на ній наведено сторонні сервіси з якими взаємодіє система – це OpenRouteServer та OpenStreetMap, які необхідні для роботи з географічними координатами адрес реєстрації парковок.

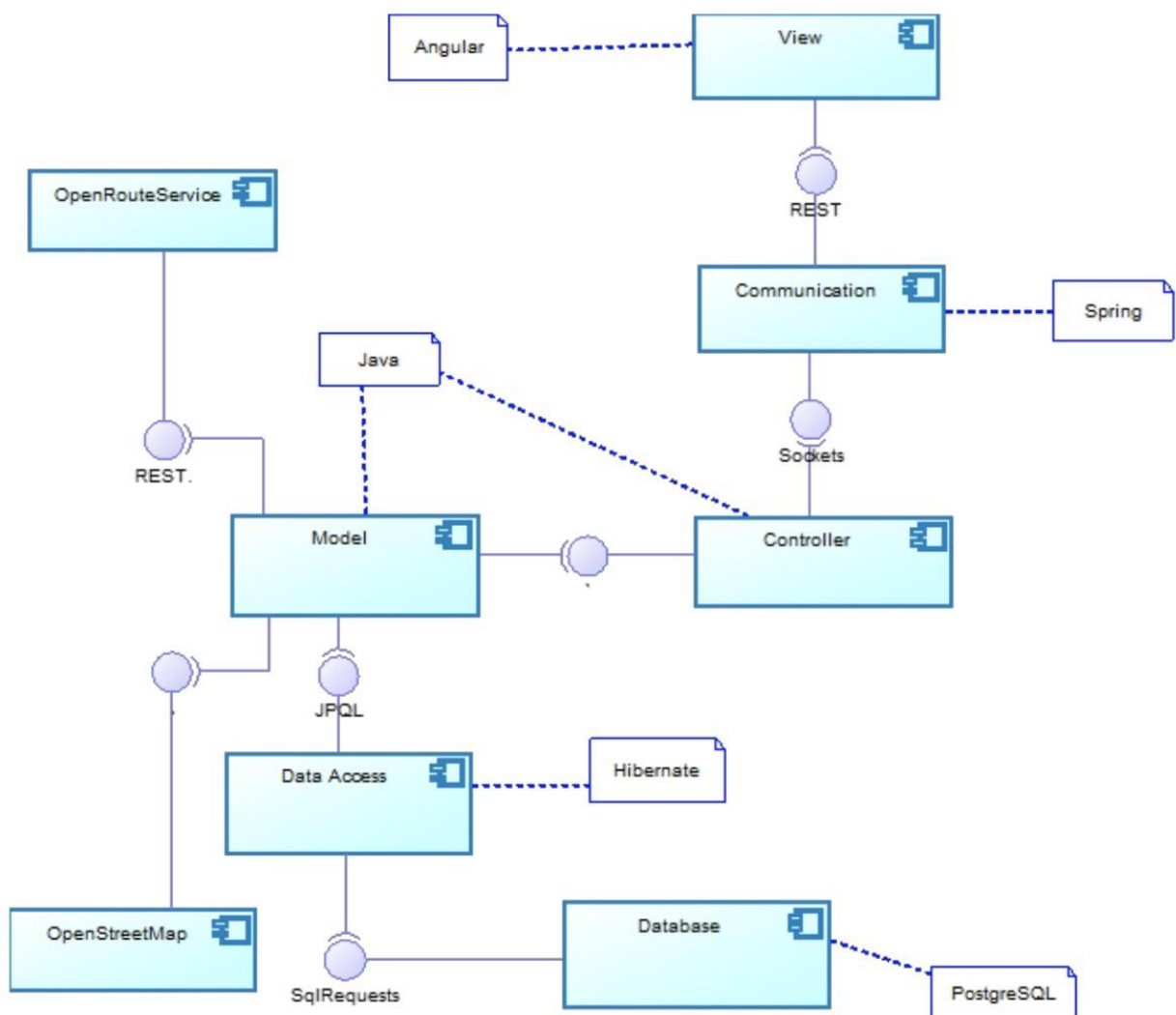


Рис. 2.3. Діаграма компонентів веб орієнтованої інформаційної системи пошуку та бронювання паркомісць

2.2.3. Діаграма послідовності

Діаграма послідовності – це UML діаграма, яка показує взаємодію об'єктів системи впорядкованих за часом. Ця діаграма відображає послідовність

виконання пошуку та бронювання паркомісць в системі. Вона представлена на рис. 2.4.

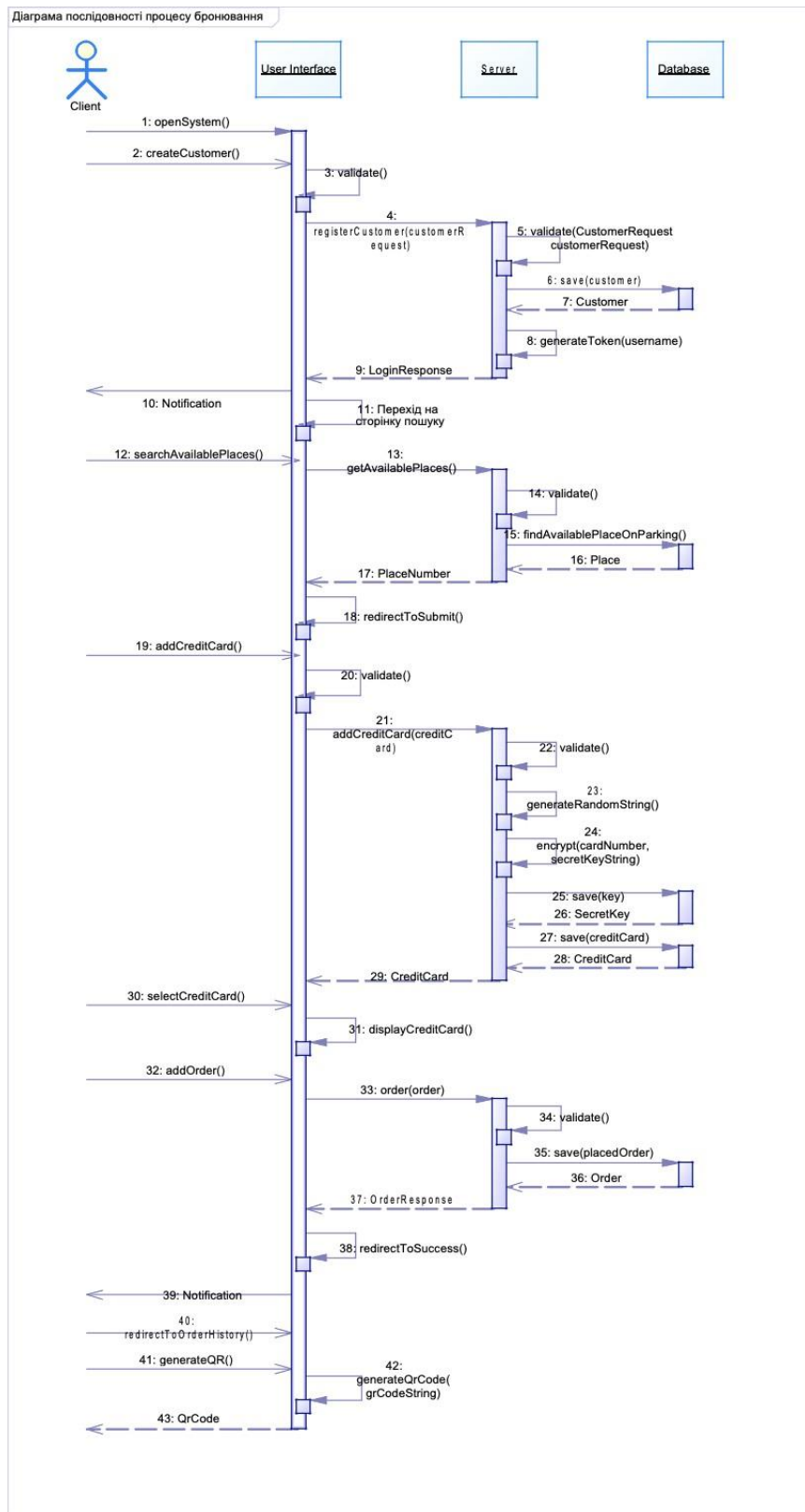


Рис. 2.4. Діаграма послідовності веб орієнтованої інформаційної системи пошуку та бронювання паркомісць

2.2.4. Діаграма станів

Діаграма станів використовується для відображення різних станів об'єктів в залежності від подій протягом всього життєвого циклу. Дана діаграма дуже добре описує динамічність системи і за допомогою неї можна легко зрозуміти роботу процесів. На рис. 2.5. представлена діаграма станів, яка описує процес пошуку та бронювання паркомісць в системі.

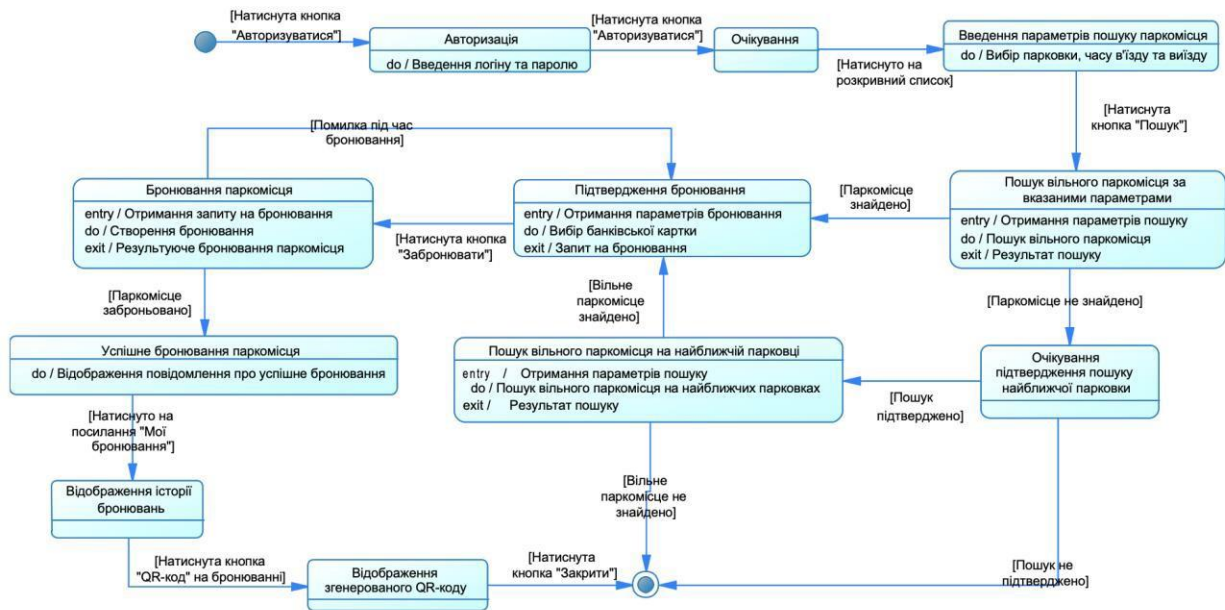


Рис. 2.5. Діаграма станів веб-орієнтованої інформаційної системи пошуку та бронювання паркомісць

2.3. Опис структури бази даних

Під час розробки системи використовувалась реляційна база даних PostgreSQL. Сервер бази даних розташований на безкоштовному хостингу “Heroku”.

Загальний розмір бази даних становить 26 таблиць. Деякі з таблиць, а саме 7, носять службовий характер. Вони використовуються для зберігання метаданих про роботу системи. Отже, тільки 19 з таблиць використовуються в бізнес-логіці

Наведемо перелік сутностей системи в таблиці 2.1.

Перелік сутностей

| № | Назва таблиці | Сутність |
|----|---------------|-------------------------|
| 1 | Address | Адреса |
| 2 | Country | Країна |
| 3 | Credit_card | Кредитна картка клієнта |
| 4 | Customer | Клієнт |
| 5 | Document | Метадані документа |
| 6 | Document_type | Тип документа |
| 7 | Order_item | Бронювання паркомісця |
| 8 | Orders | Бронювання |
| 9 | Parking | Парковка |
| 10 | Penalty | Штраф |
| 11 | Place | Паркомісце |
| 12 | Price_history | Історія тарифів |
| 13 | Role | Роль |
| 14 | Secret_key | Секретний ключ |
| 15 | Supplier | Власник парковок |
| 16 | Users | Користувач |

У таблиці 2.2 наведено детальний опис таблиць бази даних.

Таблиця 2.2

Детальний опис таблиць бази даних

| Назва таблиці | Назва стовпця | Тип даних | Опис поля |
|---------------|---------------|--------------|---------------------------|
| Address | id | bigint | Первинний ключ |
| | country_iso3 | varchar(3) | Код країни в форматі ISO3 |
| | city | varchar(255) | Місто |
| | street | varchar(255) | Вулиця |
| | street_number | varchar(255) | Номер будинку |

| Назва таблиці | Назва стовпця | Тип даних | Опис поля |
|---------------|-----------------|------------------|--|
| | latitude | Double precision | Широта |
| | longitude | double precision | Довжина |
| Country | iso3 | varchar(3) | Код країни в форматі ISO3 |
| | name | varchar(255) | Назва країни |
| Credit_card | id | bigint | Первинний ключ |
| | reference | varchar(255) | Унікальне посилання |
| | card_number | varchar(255) | Номер картки |
| | expiration_date | date | Терім закінчення дії |
| | cvv | varchar(4) | Захисний код |
| | status | varchar(50) | Статус |
| | owner | varchar(255) | Ім'я власника клієнта |
| | secret_key_id | bigint | Посилання на секретний ключ, за допомогою якого зашифровано номер картки |
| Customer | id | bigint | Первинний ключ |
| | first_name | varchar(255) | Ім'я клієнта |
| | last_name | varchar(255) | Прізвище клієнта |
| | phone_number | varchar(255) | Номер телефону |
| | user_id | bigint | Посилання на користувача |
| Document | id | bigint | Первинний ключ |

| Назва таблиці | Назва стовпця | Тип даних | Опис поля |
|---------------|---------------|--------------|--|
| | name | varchar(255) | Назва |
| | size | integer | Розмір |
| | path | varchar(255) | Шлях до документу |
| | status | varchar(255) | Статус |
| | reference | varchar(255) | Унікальне посилання |
| Document_type | code | varchar(50) | Код документа |
| | name | varchar(255) | Повна назва типу документа |
| | supplier | boolean | Тип документа належить власнику парковок |
| | parking | boolean | Тип документа належить парковці |
| Order_item | id | bigint | Первинний ключ |
| | reference | varchar(255) | Унікальне посилання |
| | place_number | varchar(50) | Номер місця |
| | datetime_from | timestamp | Запланований час заїзду на парковку |
| | datetime_to | timestamp | Запланований час виїзду з парковки |
| | order_id | bigint | Посилання на бронювання |
| Orders | id | bigint | Первинний ключ |

| Назва таблиці | Назва стовпця | Тип даних | Опис поля |
|---------------|---------------|---------------|------------------------------------|
| | reference | varchar(255) | Унікальне посилання |
| | total_price | numeric(10,2) | Загальна ціна за бронювання |
| | status | varchar(50) | Статус |
| | quantity | integer | Кількість заброньованих місць |
| | parking_id | bigint | Посилання на парковку |
| | customer_id | bigint | Посилання на клієнта |
| | created_date | timestamp | Дата та час створення бронювання |
| Parking | id | bigint | Первинний ключ |
| | reference | varchar(255) | Унікальне посилання |
| | price | numeric(10,2) | Тариф за годину паркування |
| | penalty | numeric(10,2) | Штраф за хвилину запізнення виїзду |
| | status | varchar(50) | Статус |
| | size | integer | Кількість паркомісць |
| | address_id | bigint | Посилання на адресу |

| Назва таблиці | Назва стовпця | Тип даних | Опис поля |
|------------------|---------------|---------------|------------------------------------|
| | supplier_id | bigint | Посилання на власника парковки |
| Parking_document | parking_id | bigint | Посилання на парковку |
| | document_id | bigint | Посилання на документ |
| Penalty | id | bigint | Первинний ключ |
| | reference | varchar(255) | Унікальне посилання |
| | total | numeric(10,2) | Загальна сума штрафу |
| | status | varchar(50) | Статус |
| | comment | text | Коментар |
| | order_item_id | bigint | Посилання на бронювання паркомісця |
| | customer_id | bigint | Посилання на клієнта |
| | created_date | timestamp | Дата та час створення штрафу |
| Place | id | bigint | Первинний ключ |
| | reference | varchar(255) | Унікальне посилання |
| | place_number | varchar(255) | Номер паркомісця |
| | status | varchar(50) | Статус |

| Назва таблиці | Назва стовпця | Тип даних | Опис поля |
|---------------|---------------|---------------|------------------------------------|
| | parking_id | bigint | Посилання на парковку |
| Price_history | id | bigint | Первинний ключ |
| | price | numeric(10,2) | Тариф за годину паркування |
| | penalty | numeric(10,2) | Штраф за хвилину запізнення виїзду |
| | start_date | timestamp | Початок дії тарифу |
| | end_date | timestamp | Закінчення дії тарифу |
| | parking_id | bigint | Посилання на парковку |
| Role | id | bigint | Первинний ключ |
| | name | varchar(50) | Назва ролі |
| | description | varchar(255) | Опис |
| | admin | boolean | Приналежність адміністратору |
| | supplier | boolean | Приналежність власнику парковок |
| | customer | boolean | Приналежність клієнта |
| Secret_key | id | bigint | Первинний ключ |
| | secret_key | varchar(255) | Секретний ключ |
| Supplier | id | bigint | Первинний ключ |

| Назва таблиці | Назва стовпця | Тип даних | Опис поля |
|-------------------|---------------|--------------|--------------------------------|
| | name | varchar(255) | Назва власника парковки |
| | vat_number | varchar(10) | Ідентифікаційний код |
| | address_id | bigint | Посилання на адресу |
| | user_id | bigint | Посилання на користувача |
| Supplier_document | supplier_id | bigint | Посилання на власника парковок |
| | document_id | bigint | Посилання на документ |
| User_role | user_id | bigint | Посилання на користувача |
| | role_id | bigint | Посилання на роль |
| Users | id | bigint | Первинний ключ |
| | email | varchar(255) | Електронна пошта користувача |
| | password | varchar(255) | Пароль |
| | type | varchar(50) | Тип користувача |

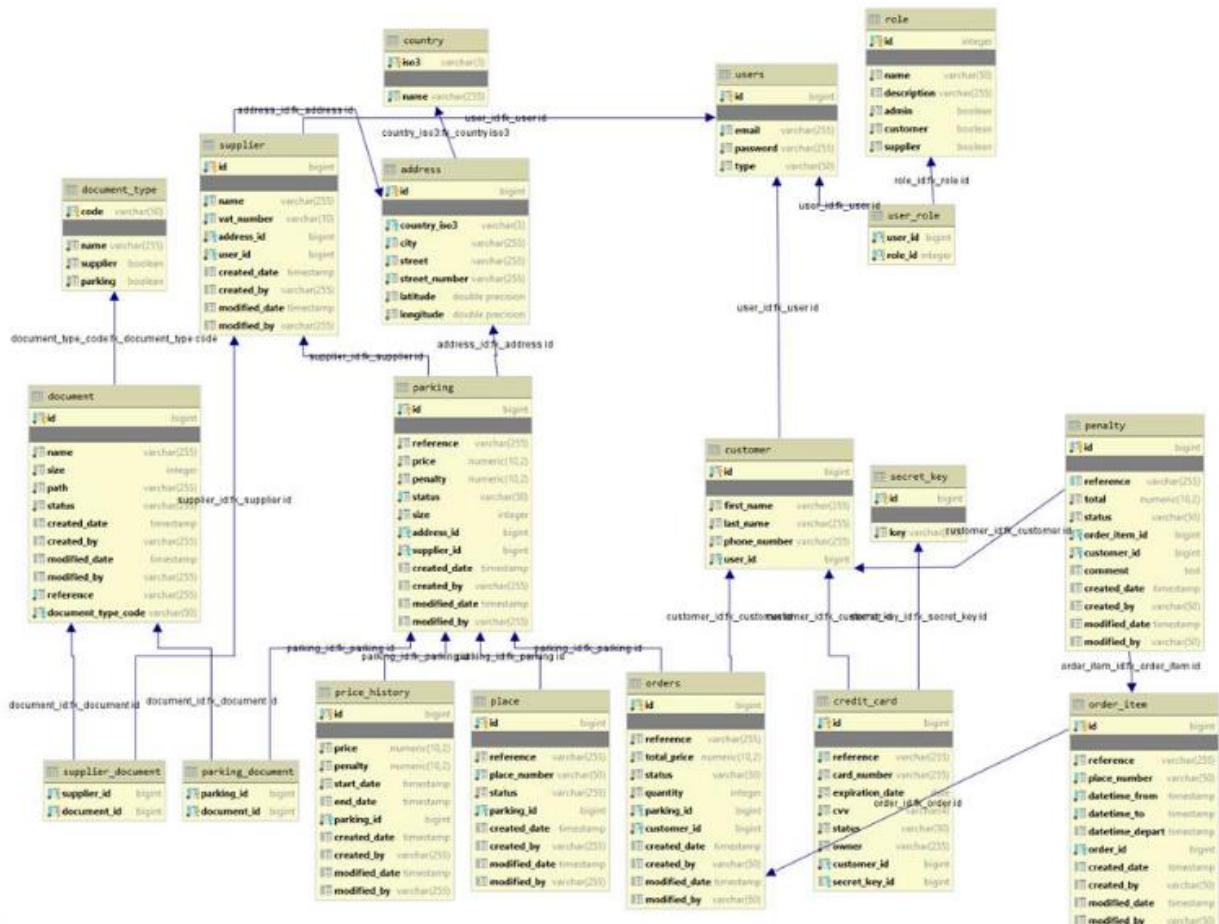


Рис. 2.6. Схема бази даних

Висновки до розділу

В даному розділі була описана архітектура застосунку і наведено її схематичне зображення. Побудовано діаграми пакетів, компонентів послідовності та станів. Висунуто вимоги до серверів, на яких буде розгортатися застосунок.

Система складається з 2-х компонент: серверна частина (API) та клієнтська. Кожна з них є незалежною і розміщені на різних серверах в одній мережі хостингу Нероки. Вищезгадані компоненти обмінюються повідомленнями використовуючи протокол HTTPS та архітектурний стиль REST.

В даному розділі було визначено декілька джерел вхідної інформації. Для кожного з них було визначено перелік вхідних та вихідних даних. Також, було описано структуру бази даних. Вона налічує 26 таблиць, з них 19 забезпечують роботу бізнес логіки системи, а 7 – зберігання метаданих про роботу системи.

Наведено перелік сутностей, котрі використовуються в системі, з вказанням таблиць, до яких вони відносяться. Також, побудовано таблицю, в котрій наведено перелік таблиць бази даних, які забезпечують роботу бізнес логіки системи, з детальним описом всіх параметрів.

РОЗДІЛ 3. РОЗРОБКА ВЕБ ОРІЄНТОВАНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ ПОШУКУ ТА БРОНЮВАННЯ ПАРКОМІСЦЬ

3.1. Обґрунтування вибору засобів розробки

Під час розробки даного програмного продукту використовувались велика кількість технічних засобів. Основні з них перераховані в таблиці 3.1.

Таблиця 3.1

Технічні засоби використанні під час розробки

| Тип | Назва | Версія |
|-------------------------|-----------------|----------|
| Мови програмування | Java/TypeScript | 11/3.5.3 |
| Фреймворки | Spring/Angular | 5/8 |
| ORM | Hibernate | 5.3 |
| IDE | IntelliJ Idea | 2021.2.3 |
| RDBMS | PostgreSQL | 12 |
| Засіб проектування | PowerDesigner | 16.5 |
| Засіб керування базами | DataGrid | 2019.3.2 |
| Система контролю версій | Git | 2.16.1 |
| Репозиторій | GitHub | — |
| Хостинг | Heroku | — |

Java – це високорівнева, кросплатформена мова програмування, яка широко використовується у різних сферах розробки програмного забезпечення. Головною ціллю даної мови було забезпечити роботу програм на пристроях з обмеженими ресурсами, наприклад телефони. Наразі, дана мова програмування набула широкої популярності для розробки серверної частини програм. Перевагами мови програмування Java є: кросплатформеність, масштабованість, велика кількість готових бібліотек.

Наразі, мова програмування Java набула неабиякої популярності для розробки веб-застосунків. Зазвичай, ця мова програмування обирається для тих систем, де очікується високе навантаження. Так як інформаційна система пошуку

та бронювання паркомісць має перспективи розвитку, значить навантаження на систему може рости, а Java допоможе з ним впоратись.

TypeScript [6] мова програмування, яка була розроблена компанією Microsoft. Ця мова програмування є зворотно сумісною з JavaScript. Використовується для розробки веб-застосунків і розширює можливості мови програмування JavaScript.

Мова програмування TypeScript дуже сильно спрощує процес написання коду. Це відбувається за допомогою так названого «синтаксичного цукру», який був доданий до TypeScript розробниками.

Фреймворк Spring [3] розроблений для мови програмування Java з метою спрощення та прискорення процесу написання веб-застосунків. Даний фреймворк містить велику кількість модулів, які мають різноманітні призначення. Один з базових модулів цього фреймворку дає підтримку інверсії управління, яка значно спрощує написання коду та його подальше тестування. Наразі, Spring нараховує близько 20 модулів, кожен з яких містить ще велику кількість компонентів. Такі цифри дійсно вражають, а головне, що цей фреймворк дозволяє зробити виконання великої кількості задач простішим на стільки, на скільки це можливо. Більшість сучасних систем, які пишуться на мові програмування Java, використовують фреймворк Spring і ми не виключення.

Angular – це front-end фреймворк, написаний на мові програмування TypeScript, з відкритим вихідним кодом. Він розробляється під керівництвом Angular Team в компанії Google. Цей фреймворк дуже сильно спрощує процес написання front-end частини систем, тому ми використовували його в даному проєкті.

Hibernate – даний фреймворк використовується для того, щоб відображати реляційні структури на об'єкти мови програмування Java. Він є однією з найпопулярніших реалізацій специфікації JPA. Метою даного фреймворку є звільнення розробника від ручного зв'язування таблиць бази даних та результатів виконання запитів. Він надає зручний інтерфейс для побудови SQL запитів за допомогою власної специфікації Hibernate Query Language

(HQL). Перевагою цієї специфікації є побудова запитів в об'єктно орієнтованому стилі, що є більш зрозумілим для розробників.

Для зберігання даних використовується об'єктно-реляційна система керування базами даних (далі СКБД) PostgreSQL [5]. Вихідний код цієї СКБД є відкритим і за рахунок цього вона має широкий функціонал і велику кількість оптимізації, що дозволяє їй скласти конкуренцію таким комерційним СКБД як Oracle, MsSQL і т.д [7]. Вона є абсолютно безкоштовною, що також надає велику перевагу.

Під час розробки використовувались два типи інтегрованих середовищ розробки (далі IDE), для роботи з базою та для написання коду. Вони розроблені однією компанією JetBrains. Для роботи з базою використовувалась програма DataGrid, а для написання коду – IDE IntelliJ Idea. Ці IDE мають велику кількість функціоналу, що дозволяє спростити роботу розробника.

Git – система контролю версій, що була розроблена на базі ядра операційної системи Linux. Він дозволяє відстежувати всі зміни, що відбувалися в проекті. За допомогою даного програмного забезпечення вирішується проблема роботи в команді, адже Git дозволяє працювати окремо, а в необхідний момент часу об'єднувати всю роботу.

GitHub – це одне з найбільших віддалених сховищ git-репозиторіїв. Воно дозволяє керувати доступом до проектів, обмінюватися вже виконаною роботою, відстежувати помилки в програмному забезпеченні. Це сховище має безкоштовний тарифний план який і використовувався під час розробки системи. Дане сховище дозволяє створювати приватні репозиторії, щоб код не був доступний для всіх охочих, має великий перелік доступного функціоналу і дуже простий у використанні.

Для розгортання розробленого продукту використовувався безкоштовний хостинг Heroku. Він дозволяє безкоштовно розгортати свої навчальні проекти. Кожен з використовуваних серверів має обмежені ресурси, але їх достатньо для навчальних проектів. Даний хостинг дозволяє з легкістю конфігурувати всі

сервери і навіть обирати їх розташування, для забезпечення швидкодії застосунків.

3.2. Специфікація функцій

В таблицях 3.2-3.26 наведено перелік методів серверної частини підсистеми з їх описом.

Таблиця 3.2

Методи класу **OpenStreetMapService**

| Сигнатура | Опис |
|--|---|
| getAddress(String country, String city, String street, String houseNumber) | Знаходження адреси з її географічними координатами за допомогою сервісу OpenStreetMap |
| buildURL(String country, String city, String street, String houseNumber) | Побудова веб шляху до сервісу OpenStreetMap |

Таблиця 3.3

Методи класу **CreditCardService**

| Сигнатура | Опис |
|--|---|
| addCreditCard(CreditCardRequest creditCardRequest) | Додавання кредитної картки клієнта |
| getCreditCards() | Знаходження всіх кредитних карток клієнта |
| deleteCreditCard(String cardReference) | Видалення кредитної картки |

Таблиця 3.4

Методи класу **CustomerService**

| Сигнатура | Опис |
|---|--|
| getCustomer(@NotBlank String email) | Знаходження клієнта по електронній пошті |
| updateCustomerInfo(CustomerBase customerBase) | Оновлення інформації про клієнта |

Методи класу CreditCardNumberValidator

| Сигнатура | Опис |
|--|---|
| isValid(String creditCardNumber, ConstraintValidatorContext | Перевірка формату номера кредитної картки |

Таблиця 3.6

Методи класу CvvCodeValidator

| Сигнатура | Опис |
|---|---|
| isValid(String cvv, ConstraintValidatorContext | Перевірка формату захисного коду кредитної картки |

Таблиця 3.7

Методи класу ExpirationDateValidator

| Сигнатура | Опис |
|---|--|
| isValid(LocalDate expirationDate, ConstraintValidatorContext | Перевірка терміну дії кредитної картки |

Таблиця 3.8

Методи класу CreditCardController

| Сигнатура | Опис |
|--|--|
| addCreditCard(@Valid @RequestBody CreditCardRequest creditCard) | Запит на додавання кредитної картки |
| getCreditCards() | Запит на знаходження всіх кредитних карток |
| deletedCreditCard(@PathVariable("reference") String cardReference) | Запит на видалення картки |

Таблиця 3.9

Методи класу CustomerManagementController

| Сигнатура | Опис |
|------------------|---|
| getCustomer() | Запит на знаходження інформації про клієнта |

| Сигнатура | Опис |
|---|---|
| updateCustomerInfo(CustomerBase customerBase) | Запит на оновлення інформації про клієнта |

Таблиця 3.10

Методи класу OpenRouteServiceClient

| Сигнатура | Опис |
|--|---|
| getDirection(List<Pair<BigDecimal, BigDecimal>> coordinates) | Знаходження шляху між точками, що задані географічними координатами |
| buildRequest(List<Pair<BigDecimal, BigDecimal>> coordinates) | Побудова запиту на знаходження шляху |
| callApi(Uri uri, HttpEntity<?> request) | Виклик зовнішнього сервісу |

Таблиця 3.11

Методи класу DistanceService

| Сигнатура | Опис |
|---|---|
| double calculateDistanceBetweenPoints(Pair<Double, Double> startLatLonPair, Pair<Double, Double> endLatLonPair) | Знаходження прямої відстані між точками, що задані географічними координатами |
| calculateDistanceBetweenPoints(Address startAddress, Address endAddress) | Знаходження прямої відстані між двома адресами |
| findShortestDistanceBetweenAddresses(Address startAddress, Address endAddress) | Знаходження найкоротшого шляху між адресами |

Методи класу OrderRepository

| Сигнатура | Опис |
|---|--|
| findAllByCustomerAndCreatedDateBetween(@Param("email") String email, @Param("from") LocalDateTime from, @Param("to") LocalDateTime to, Pageable pageable) | Знаходження сторінки замовлень клієнта за вказаний проміжок часу |

Таблиця 3.13

Методи класу PlacesAreAvailableValidator

| Сигнатура | Опис |
|--|----------------------------------|
| isValid(Order order, ConstraintValidatorContext) | Перевірка доступності паркомісць |

Таблиця 3.14

Методи класу OrderRepositoryImpl

| Сигнатура | Опис |
|--|---|
| findByReferenceAndPlaceNumberAndFromToDates(String reference, String placeNumber, LocalDateTime from, LocalDateTime to) | Пошук замовлення за унікальним посиланням, номер паркомісця та періодом перебування |
| existsOrderByReferenceAndPlaceNumberAndFromToDates(String reference, String placeNumber, LocalDateTime from, LocalDateTime to) | Перевірка існування замовлення за унікальним посиланням, номер паркомісця та періодом перебування |

Методи класу PlacesAreKnownValidator

| Сигнатура | Опис |
|--|--------------------------------|
| isValid(Order order, ConstraintValidatorContext | Перевірка існування паркомісць |

Таблиця 3.16

Методи класу TimestampToIsAfterTimestampFromValidator

| Сигнатура | Опис |
|---|--|
| isValid(OrderItemRequest orderItemRequest, ConstraintValidatorContext | Перевірка коректності проміжку часу бронювання паркомісця |

Таблиця 3.17

Методи класу OrderService

| Сигнатура | Опис |
|--|---|
| getCustomerOrders(Integer pageNumber, Integer pageSize, LocalDate fromDate, LocalDate toDate) | Знаходження списку бронювань клієнта |
| findOrders(String identifier, Pageable pageable, LocalDate fromDate, LocalDate toDate, Function<OrderSearchRequest, Page<Order>> orderSearchFunction) | Знаходження писку бронювань |
| orderParkingPlace(com.kpi.parking.model.Order order) | Бронювання паркомісця |
| enrichOrder(Order order, Parking parking) | Розширення замовлення необхідною інформацією |
| calculateTotalPrice(Set<OrderItem> orderItems, BigDecimal pricePerHour) | Розрахунок загальної вартості паркування |

Методи класу PenaltyRepository

| Сигнатура | Опис |
|--|---------------------------------------|
| <code>findPageByUsernameAndStatus(@Param("username") String username, @Param("status") PenaltyStatus status, Pageable pageable)</code> | Пошук штрафів клієнта за статусом |
| <code>findByReference(String reference)</code> | Пошук штрафу за унікальним посиланням |
| <code>findActivePenaltiesByUsername(@Param("username") String username)</code> | Знаходження активних штрафів клієнта |

Таблиця 3.19

Методи класу OrderController

| Сигнатура | Опис |
|--|---------------------------------------|
| <code>getCustomerOrders(@RequestParam(value = "pageNumber") Integer pageNumber, @RequestParam(value = "pageSize") Integer pageSize, @RequestParam(value = "fromDate", required = false) @DateTimeFormat(iso = DateTimeFormat.ISO.DATE) LocalDate fromDate, @RequestParam(value = "toDate", required = false) @DateTimeFormat(iso = DateTimeFormat.ISO.DATE) LocalDate toDate)</code> | Запит на знаходження списку замовлень |
| <code>order(@Valid @RequestBody Order order)</code> | Запит на бронювання паркомісця |

Методи класу DateTimeFromIsStartedValidator

| Сигнатура | Опис |
|--|-----------------------------------|
| isValid(VisitRequest visitRequest, ConstraintValidatorContext) | Перевірка дати в'їзду на парковку |

Таблиця 3.21

Методи класу PenaltyService

| Сигнатура | Опис |
|---|---|
| getPenalties(int pageNumber, int pageSize, PenaltyStatus penaltyStatus) | Знаходження та обробка штрафів за статусом |
| payPenalty(String penaltyReference, Payment payment) | Сплата штрафу |
| isTotalPenaltyExceedsLimit() | Перевірка перевищення ліміту штрафів клієнтом |

Таблиця 3.22

Методи класу PenaltyController

| Сигнатура | Опис |
|--|--|
| getCustomerPenalties(@RequestParam(value = "status") String status, @RequestParam(value = "pageNumber") Integer pageNumber, @RequestParam(value = "pageSize")) | Запит на пошук штрафів клієнта |
| payFine(@PathVariable("reference") String reference, @Valid Payment payment) | Запит на сплату штрафу |
| getPenaltyStatuses() | Запит на отримання списку штрафів |
| isTotalPenaltyExceedsLimit() | Перевірка перевищення суми активних штрафів клієнтом |

Методи класу ParkingController

| Сигнатура | Опис |
|--|--|
| <code>getAvailableParkings()</code> | Запит на отримання |
| <code>getAvailablePlaces(@ParkingKnownAndActive(groups = ParkingKnownAndActive.class) @PathVariable("reference") String reference, @TimestampIsCorrect(groups = TimestampIsCorrect.class) @RequestParam("from") String from, @RequestParam("to") String to)</code> | Запит на отримання списку вільних місць |
| <code>findNearestParkingWithPlaces(@ParkingKnownAndActive(groups = ParkingKnownAndActive.class) @PathVariable("reference") String reference, @TimestampIsCorrect(groups = TimestampIsCorrect.class) @RequestParam("from") String from, @RequestParam("to") String to)</code> | Запит на пошук найближчої парковки з вільними паркомісцями |

Таблиця 3.24

Методи класу VisitService

| Сигнатура | Опис |
|--|--|
| <code>enterParking(VisitRequest visitRequest)</code> | Реєстрація в'їзду на парковку |
| <code>leaveParking(VisitRequest visitRequest)</code> | Реєстрація виїзду з парковки |
| <code>calculatePenalty(OrderItem orderItem, Customer customer, Price price, LocalDateTime departDateTime)</code> | Розрахунок штрафу за несвоєчасне звільнення паркомісця |
| <code>updateStatusOfParkingPlaces(Parking parking, PlaceStatus newStatus, @NotBlank String)</code> | Оновлення статусу паркомісця |

Методи класу VisitController

| Сигнатура | Опис |
|---|----------------------------|
| enterParking(@Valid @DateTimeFromIsStarted(groups = DateTimeFromIsStarted.class) @DateTimeHasNotPassed(groups = DateTimeHasNotPassed.class) VisitRequest enterRequest) | Запит на в'їзд на парковку |
| leaveParking(@Valid VisitRequest leaveRequest) | Запит на виїзд за парковки |

Таблиця 3.26

Методи класу ParkingService

| Сигнатура | Опис |
|---|---|
| getAvailableParkings() | Пошук доступних парковок |
| findAvailablePlace(@NotBlank String parkingReference, @NotBlank String from, @NotBlank String to) | Пошук вільних паркомісць |
| findNearestParking(@NotBlank String reference, @NotBlank String from, @NotBlank String to) | Пошук найближчої парковки з вільними місцями |

У таблицях 3.27-3.36 наведено перелік методів клієнтської частини підсистеми з їх описом.

Таблиця 3.27

Методи класу CardService

| Сигнатура | Опис |
|------------|---|
| getCards() | Отримання списку банківських карток з серверу |

| Сигнатура | Опис |
|-----------------------|---|
| addCard(creditCard) | Відправка запиту до серверу на додавання банківської картки |
| deleteCard(reference) | Відправка запиту до серверу на видалення картки |

Таблиця 3.28

Методи класу CusOrderComponent

| Сигнатура | Опис |
|--------------|--------------------------|
| ngOnInit() | Ініціалізація компонента |
| getCountry() | Отримання списку країн |

Таблиця 3.29

Методи класу CustomerService

| Сигнатура | Опис |
|--------------------------|--|
| getInfo() | Відправка запиту до серверу на отримання інформації про клієнта |
| updateInfo(customerInfo) | Відправка запиту до серверу на оновлення персональної інформації клієнта |

Таблиця 3.30

Методи класу OrderService

| Сигнатура | Опис |
|---|---|
| getCustomerOrders(filters) | Відправка запиту до серверу на отримання списку бронювань клієнта |
| addOrder(parkingReference, placeNumber, from, to) | Відправка запиту до серверу на створення бронювання |

Методи класу ParkingService

| Сигнатура | Опис |
|---|---|
| getAvailableParking() | Відправка запиту до серверу на отримання списку доступних парковок |
| getAvailablePlaces(parkingReference, from, to) | Відправка запиту до серверу на отримання списку вільних паркомісць |
| getAvailableNearestPlaces(parkingReference, from, to) | Відправка запиту до серверу на пошук паркомісць на найближчих парковках |

Таблиця 3.32

Методи класу PenaltyService

| Сигнатура | Опис |
|---|--|
| getPenaltyStatuses() | Відправка запиту до серверу на отримання списку статусів для штрафів |
| getPenalties(filters) | Відправка запиту до серверу на отримання списку штрафів клієнта |
| getTotalPenalty() | Відправка запиту до серверу на перевірку загальної суми активних штрафів клієнта |
| payPenalty(penaltyReference, cardReference) | Відправка запиту до серверу на сплату штрафу |

Таблиця 3.33

Методи класу CheckoutComponent

| Сигнатура | Опис |
|--------------------|---------------------------------|
| ngOnInit() | Ініціалізація компонента |
| processOrderInfo() | Форматування деталей бронювання |

| Сигнатура | Опис |
|------------------------------|--------------------------------------|
| calculateTotalPrice() | Розрахунок загальної суми бронювання |
| getCreditCards() | Отримання списку кредитних карт |
| selectCreditCard(creditCard) | Вибір кредитної картки |
| addOrder() | Створення бронювання |
| showLoader() | Відображення динамічних прелоадерів |

Таблиця 3.34

Методи класу CusCardComponent

| Сигнатура | Опис |
|-------------------------------------|---|
| ngOnInit() | Ініціалізація компонента |
| initialiseCard() | Ініціалізація відображення кредитних карт |
| getProcessCard() | Форматування даних кредитних карток |
| addCard() | Додавання кредитної картки |
| cleanCard() | Очищення введеної інформації |
| deleteCard(reference, index) | Видалення кредитної картки |
| validation() | Перевірка даних кредитної картки |
| setNotification(type, action, code) | Відображення повідомлення |
| showLoader() | Відображення динамічних прелоадерів |
| filterOrders() | Фільтрація бронювань |
| getOrders() | Отримання списку бронювань |
| processOrders(orders) | Форматування даних про бронювання |
| generateQrCode(grCodeString) | Генерація QR-коду |
| createPagination() | Створення блоку пагінації |
| changePage(pageNumber) | Перехід на іншу сторінку |
| changeSize(size) | Зміна розміру сторінки |
| showLoader() | Відображення динамічних прелоадерів |

Методи класу **CusAccountComponent**

| Сигнатура | Опис |
|-------------------------------------|---|
| ngOnInit() | Ініціалізація компонента |
| getInfo() | Отримання інформації про клієнта |
| updateInfo() | Оновлення персональної інформації про клієнта |
| editable() | Визначення редагованості інформації |
| validation() | Перевірка персональної інформації |
| setNotification(type, action, code) | Відображення повідомлення |
| showLoader() | Відображення динамічних прелоадерів |

Таблиця 3.36

Методи класу **PenaltyComponent**

| Сигнатура | Опис |
|-------------------------------------|---|
| ngOnInit() | Ініціалізація компонента |
| getCreditCards() | Отримання списку кредитних карток клієнта |
| setSelectPenalty(penalty, index) | Обрати штраф для сплати |
| payPenalty(creditCard) | Сплатити штраф |
| getPenaltyStatuses() | Отримати список можливих статусів штрафів |
| filterPenalties() | Фільтрація штрафів |
| getPenalties() | Отримання списку штрафів |
| processPenalties(penalties) | Форматування інформації про штрафи |
| createPagination() | Створення блоку пагінації |
| changePage(pageNumber) | Перехід на іншу сторінку |
| changeSize(size) | Зміна розміру сторінки |
| showLoader() | Відображення динамічних прелоадерів |
| setNotification(type, action, code) | Відображення повідомлення |

3.3. Розробка інтерфейсу

Особливо важливим елементом програмного продукту є інтерфейс користувача. Правильно спроектований інтерфейс повинен бути зручним у використанні і простим в його освоєнні. Занадто складний та перевантажений інтерфейс буде відлякувати потенційних користувачів, тому інформаційна система стане абсолютно непотрібною кінцевим юзерам.

Розроблена система надає повний функціонал для виконання пошуку паркомісць з подальшим їх бронюванням. Інтерфейс програмного продукту з пошуку та бронювання паркомісць надає наступний перелік функціоналу:

- Пошук паркомісць;
- Бронювання паркомісць;
- Перегляд та редагування персональної інформації;
- Керування банківськими картками;
- Перегляд деталей власних бронювань;
- Керування власними штрафами;
- Вихід з облікового запису.

Веб орієнтована система пошуку та бронювання паркомісць умовно поділена на 2 частини:

- Сторінка пошуку та бронювання паркомісць;
- Кабінет користувача.

Спочатку пропонуємо розглянути першу частину системи, сторінки пошуку та бронювання, так як це є головним функціоналом системи.

Після автентифікації користувача в системі, він має бути перенаправлений на головну сторінку системи, яка дозволить виконати пошук вільних паркомісць за вказаними параметрами (рис. 3.1.).

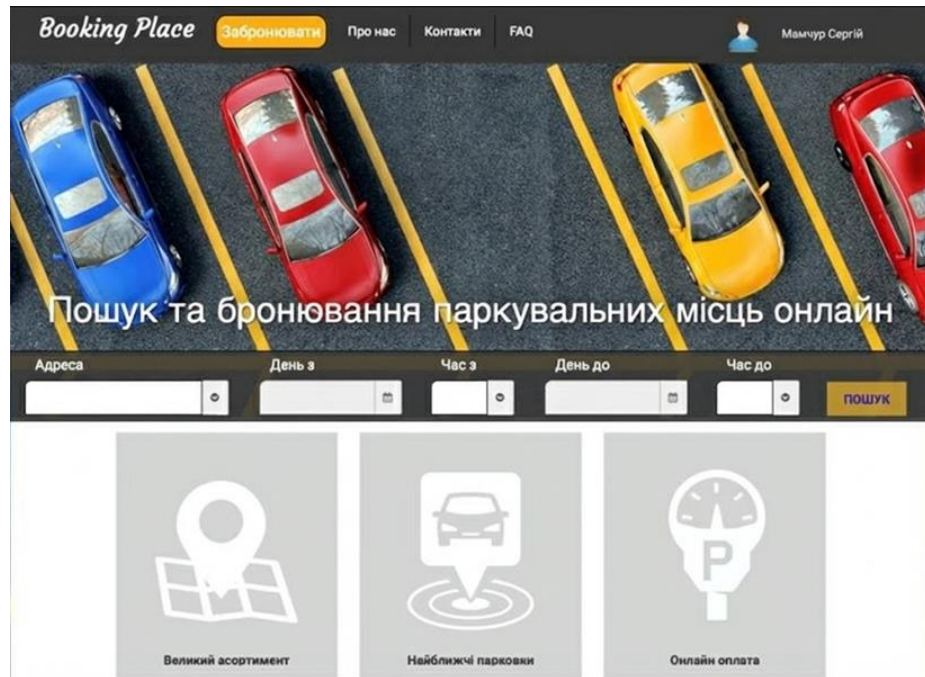


Рис. 3.1. Головна сторінка розроблюваної системи

Головна сторінка дозволяє ввести наступні параметри для виконання пошуку вільного паркомісця:

- Адреса;
- День та час початку бронювання;
- День та час закінчення бронювання.

Після введення необхідних даних і натискання кнопки «Пошук», користувача буде перенаправлений на сторінку оформлення бронювання.

Тепер пропоную перейти до другої частини системи і розглянути як буде виглядати кабінет користувача і який функціонал має містити. Перш за все, варто сказати яким чином можна потрапити до кабінету користувача. Для переходу в особистий кабінет, потрібно натиснути на ім'я користувача в правому кутку веб-сторінки і у випадяючому списку обрати один з пунктів (окрім «Вихід») як показано на рис. 3.2.

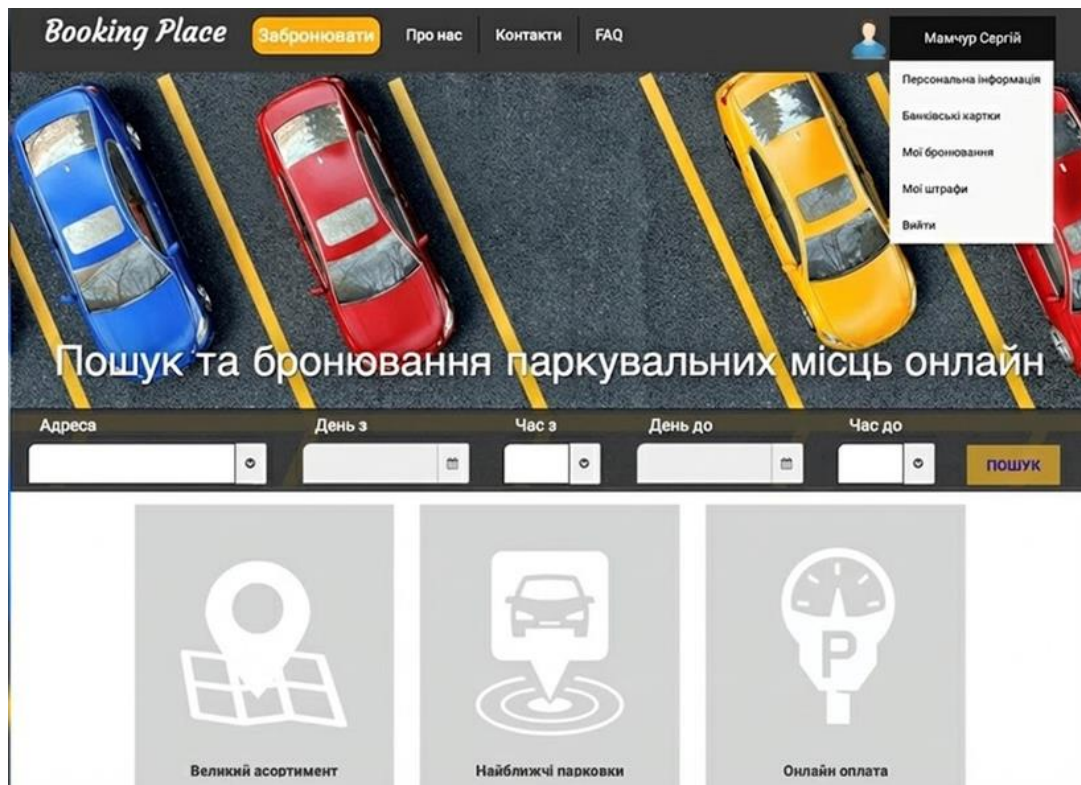


Рис. 3.2. Огляд випадаючого списку для переходу до власного кабінету

За допомогою кабінету користувача можна виконати такі дії:

- Редагування персональної інформації;
- Додати або видалити кредитну картку;
- Виконати пошук списку бронювань користувача;
- Переглянути деталі конкретного бронювання;
- Згенерувати QR-коду для в'їзду на паркінг;
- Виконати пошук штрафів по статусу;
- Здійснити оплату штрафу;
- Переглянути деталі штрафу.

Детальний огляд того, як користуватись описаним функціоналом за допомогою інтерфейсу користувача буде описано у розділі «Керівництво користувача».

3.4. Керівництво користувача

Після того як клієнт зареєструвався та авторизувався в системі, він може змінити персональні дані, якщо це необхідно. Для цього необхідно перейти на сторінку з персональними даними, як показано на рисунку 3.3.

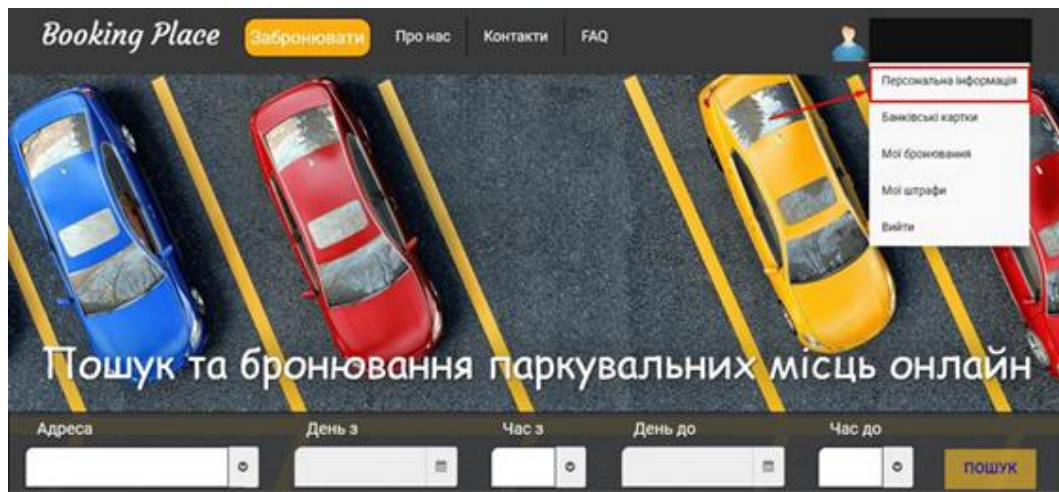


Рис. 3.3. Перехід на сторінку з персональними даними

Для того, щоб відредагувати персональні дані, необхідно натиснути на олівець, як показано на рисунку 3.4.

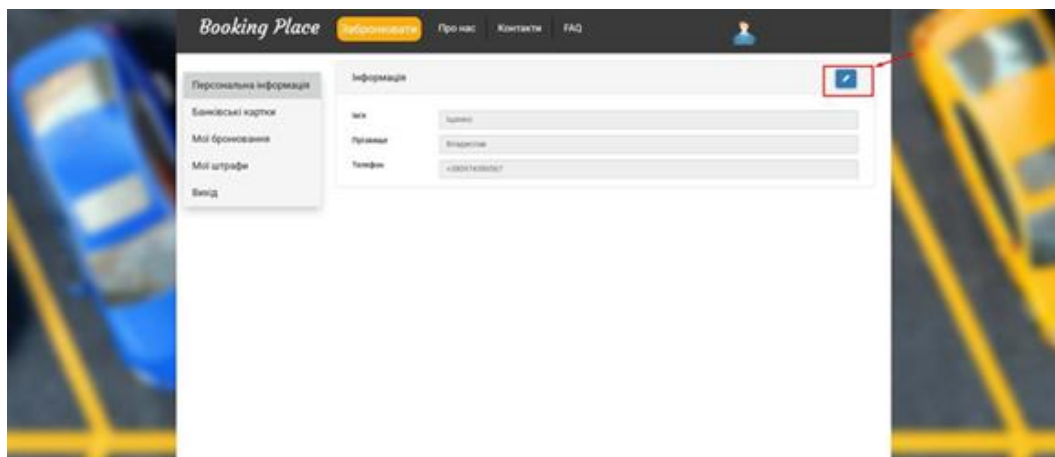


Рис. 3.4. Дозвіл редагування персональних даних

Наступним кроком необхідно відредагувати дані та зберегти їх, як показано на рисунку 3.5.

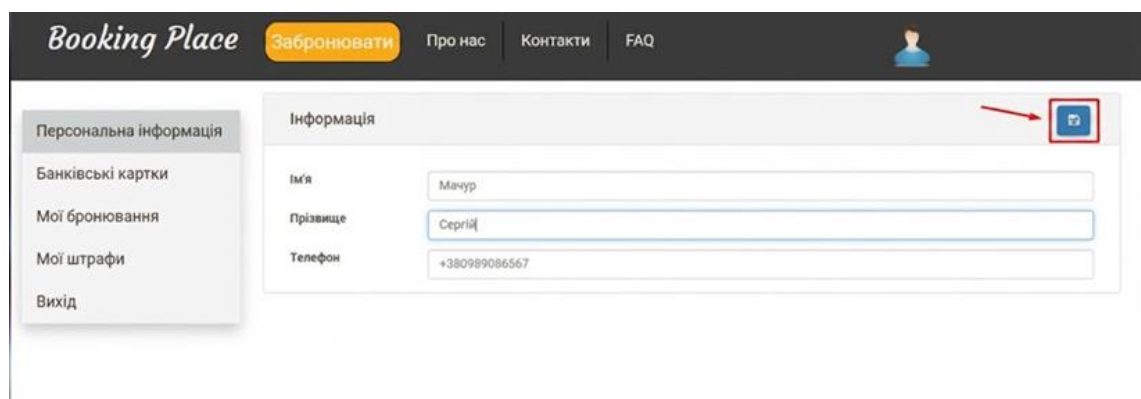


Рис. 3.5. Редагування та збереження персональних даних

Для того, щоб виконувати бронювання, клієнт має додати принаймні одну кредитну картку. Перейдемо на сторінку «Банківські картки» персонального кабінету, як показано на рисунку 3.6.

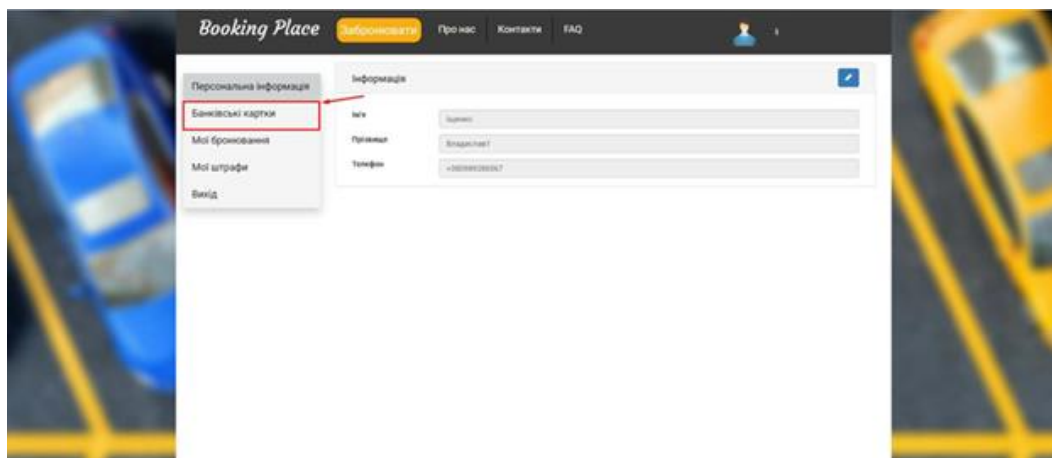


Рис. 3.6. Перехід на сторінку «Банківські картки»

Після того, як відкриється сторінка керування банківськими картками, необхідно ввести дані кредитної картки, яку клієнт хоче використовувати для здійснення оплати, і натиснути кнопку «Додати картку». На рисунку 3.7 показано як виконати додавання картки.

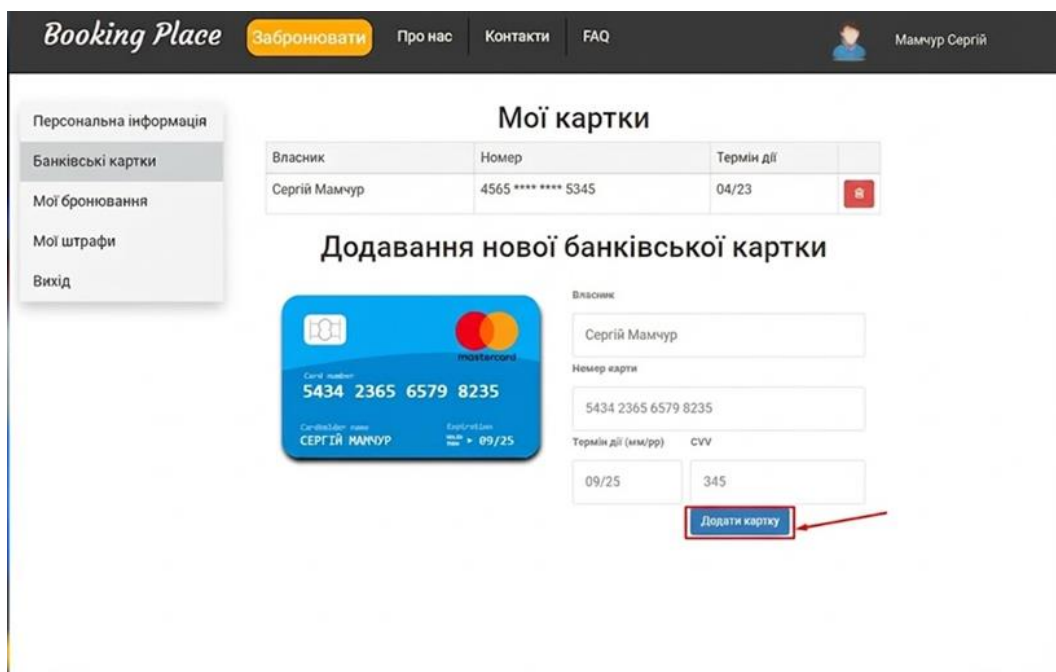


Рис. 3.7. Додавання кредитної картки клієнта

Клієнт може видалити банківську картку в будь-який момент часу, натиснувши на відповідну кнопку, як показано на рисунку 3.8.

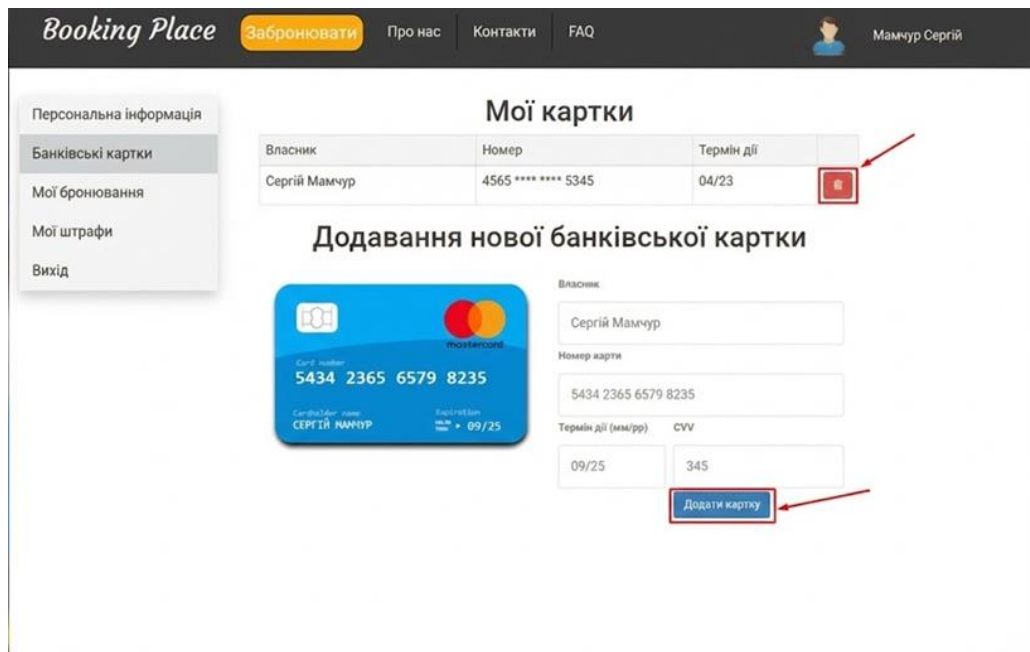


Рис. 3.8. Видалення кредитної картки

Перейдемо до головного функціоналу для клієнтів, а саме пошук та бронювання паркомісць. На рисунку 3.9 показано, як виконати перехід на сторінку пошуку вільних паркомісць.

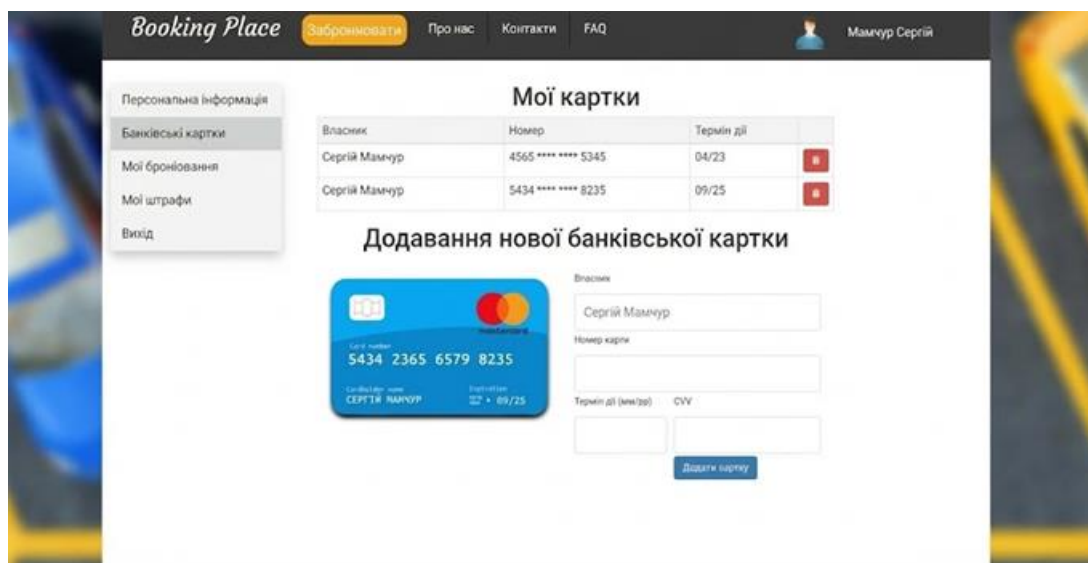


Рис. 3.9. Перехід на сторінку пошуку вільних паркомісць

Перед пошуком вільних паркомісць, клієнт має вказати наступні дані:

- адресу парковки, обравши зі списку;
- дату заїзду;
- час заїзду;
- дату виїзду;

– час виїзду.

Після вказання всіх необхідних даних, необхідно натиснути на кнопку «Пошук» і система виконає пошук вільного паркомісця на обраній парковці на вказаний проміжок часу. На рисунку 3.10 показано приклад пошуку вільних паркомісць.

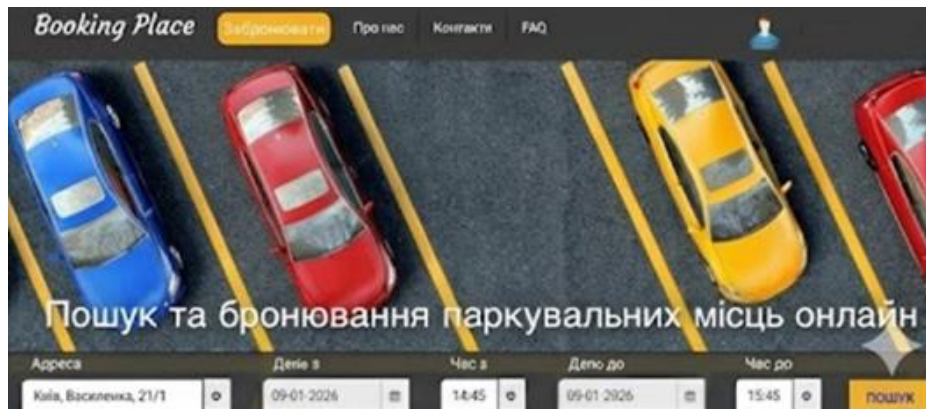


Рис. 3.10. Пошук вільних паркомісць

Необхідно розглянути два випадки, коли є вільні паркомісця і коли їх немає. У випадку наявності вільного паркомісця на обраній парковці, система виконає перенаправлення на сторінку підтвердження. Більш цікавим є випадок, коли на парковці немає вільних місць, в такому разі система запропонує пошук вільних паркомісць на найближчій парковці, як показано на рисунку 3.11.

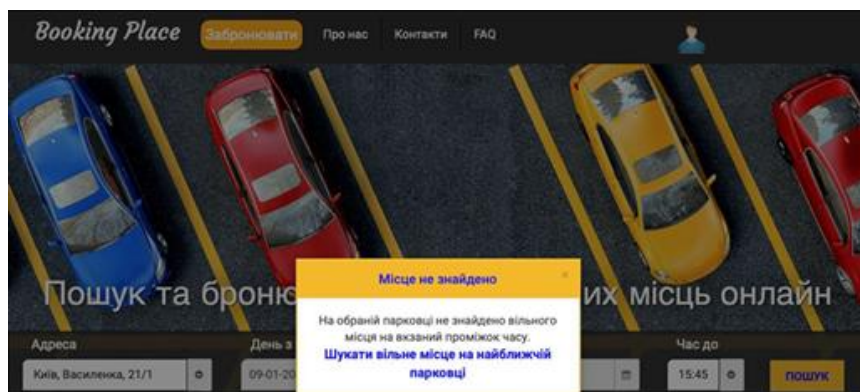


Рис. 3.11. Пропозиція про пошук паркомісць на найближчих парковках


Якщо клієнт зацікавлений в пошуку паркомісць по найближчим парковкам, йому необхідно підтвердити цю дію, як показано на рисунку 3.12.

Після підтвердження пошуку паркомісць на найближчих парковках, система обере ту парковку, шлях до якої найкоротший і виконає перенаправлення на сторінку підтвердження, як показано на рисунку 3.12.

Оформлення бронювання

| | | | |
|---------------------|------------------------------|------------------|-------------|
| Парковка: | Київ, Космонавта Комарова, 1 | Вартість: | 10 грн/год. |
| Місце: | 1 | Штраф Ө: | 19 грн/хв. |
| Час заїзду: | 9 Січня 2026р 14:45 | | |
| Час виїзду: | 9 Січня 2026р 15:45 | | |
| Час стоянки: | 1 год. | | |

ОБРАТИ БАНКІВСЬКУ КАРТКУ



Загальна сума: 10 грн

ЗАБРОНІЮВАТИ

Рис. 3.12. Сторінка підтвердження бронювання

На сторінці підтвердження бронювання, клієнт може побачити всю інформацію про бронювання, включаючи ціну за годину та штраф, який буде накладено у випадку несвоєчасного звільнення парковки. Також, клієнт може побачити місце на карті, де знаходиться парковка і суму до сплати. Перед тим як підтвердити бронювання, клієнт має обрати банківську картку, з якої буде проведена оплата. Для вибору картки необхідно натиснути на кнопку «Обрати банківську картку», після чого відкриється спливаюче вікно, як показано на рисунку 3.13.

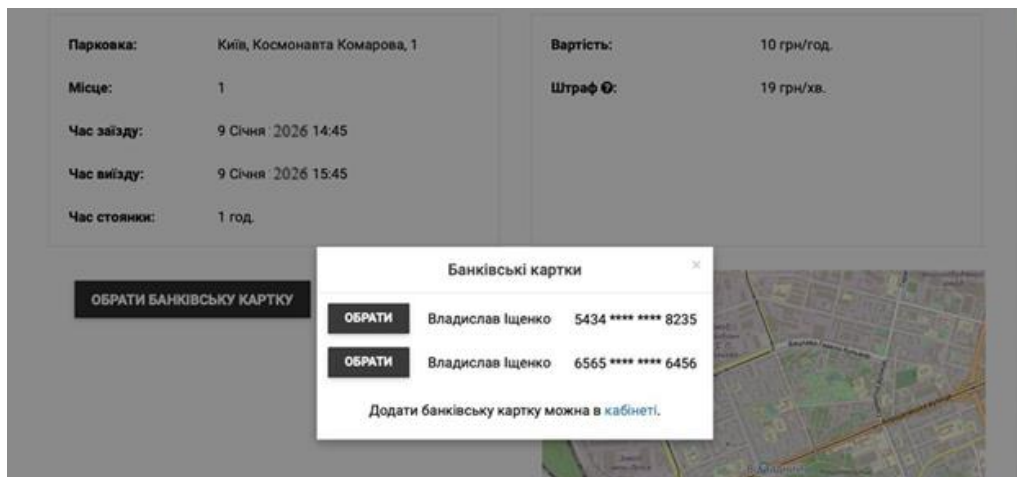


Рис. 3.13. Вікно вибору банківської картки

Після того, як банківську картку обрано, необхідно натиснути на кнопку «Забронювати». У разі успішного бронювання паркомісця, система виконає перенаправлення на сторінку успішного бронювання, яка показана на рисунку 3.14.

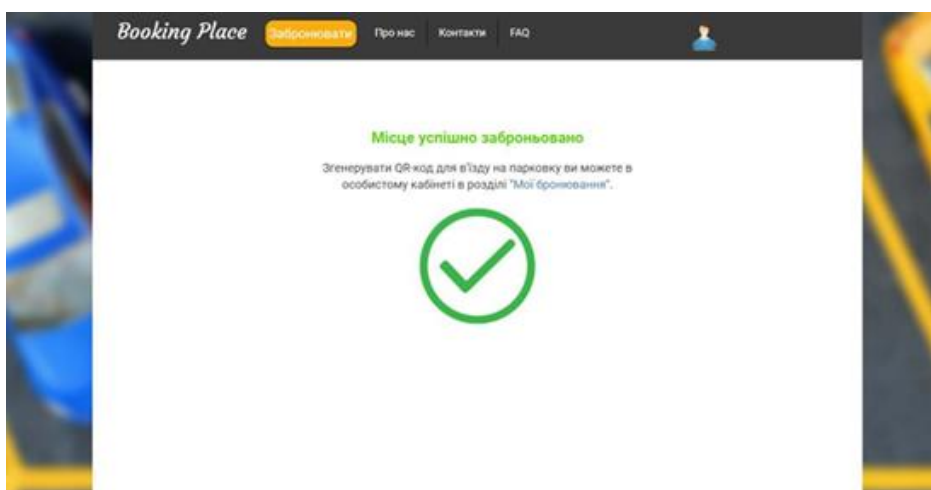


Рис. 3.14. Сторінка успішного бронювання

Натиснувши на посилання «Мої бронювання» на сторінці успішного бронювання, відбудеться перехід на сторінку бронювань користувача. На сторінці бронювань клієнт може побачити всю історію своїх бронювань і виконати фільтрацію по даті створення. Приклад цієї сторінки наведено на рисунку 3.15.

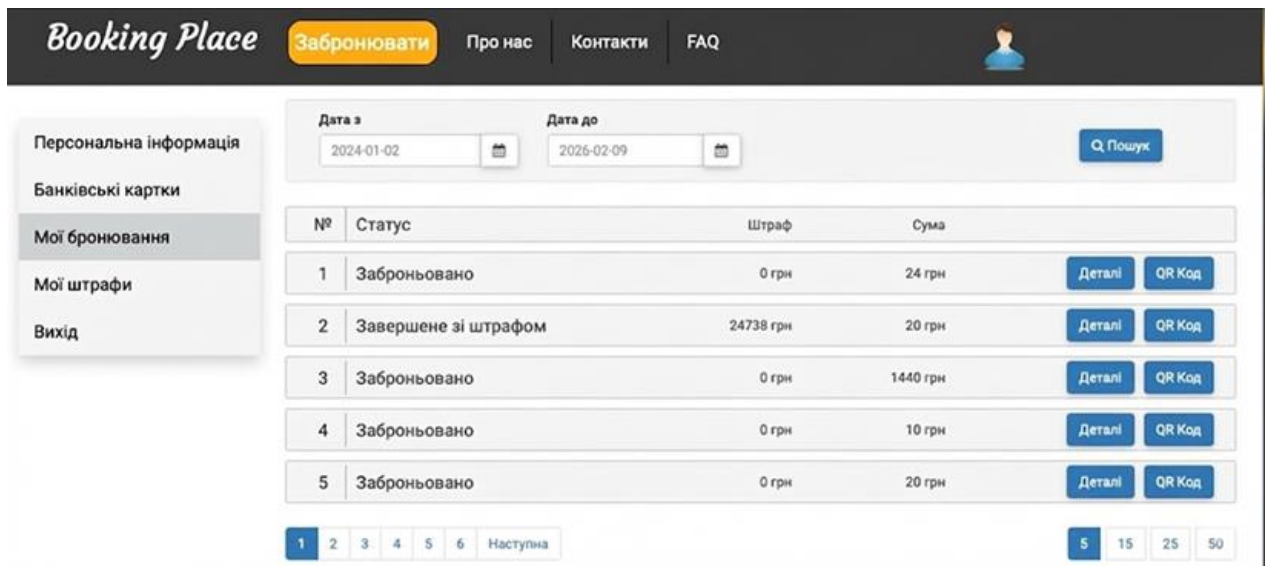


Рис. 3.15. Сторінка історії бронювань

Клієнт має можливість переглянути деталі кожного з замовлень натиснувши на кнопку «Деталі» відповідного бронювання. Результат наведено на рисунку 3.16.

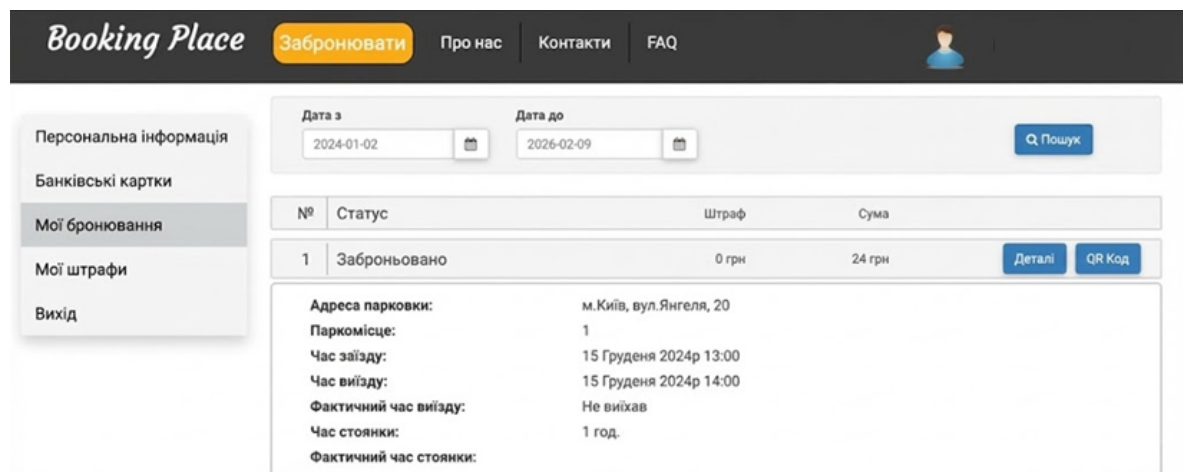


Рис. 3.16. Деталі бронювання

При в'їзді на парковку, клієнту необхідно буде відсканувати QR-код на шлагбаумі, який необхідно згенерувати натиснувши на кнопку «QR Код». Результат генерації показано на рисунку 3.17.



Рис. 3.17. Згенерований QR-код

Після того, як клієнт виїде з парковки, статус замовлення буде змінено на «Завершено» у випадку своєчасного звільнення парковки, або «Завершено зі штрафом» в тому випадку, якщо клієнт звільнив паркомісце не своєчасно.

Якщо клієнт звільнив паркомісце не своєчасно, то йому нарахується штраф за кожен хвилину простою відповідно до встановленого тарифу. Переглянути всі штрафи можна перейшовши на сторінку «Мої штрафи», як показано на рисунку 3.18.

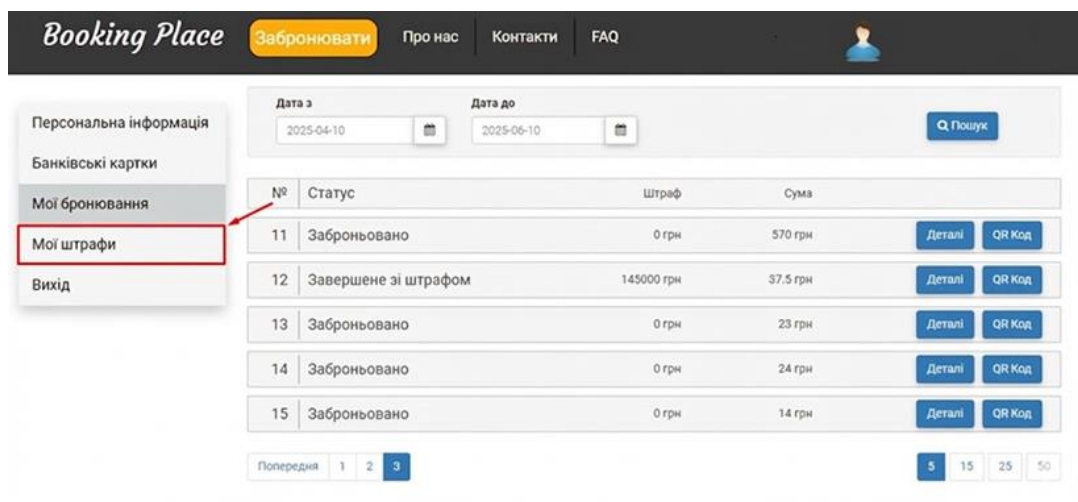


Рис. 3.18. Перехід на сторінку «Мої штрафи»

На сторінці «Мої штрафи» можна переглянути штрафи, які вже були сплачені і ті, які ще не сплачені, обравши відповідний статус в фільтрах. На рисунку 3.19 зображена сторінка зі списком не сплачених штрафів.

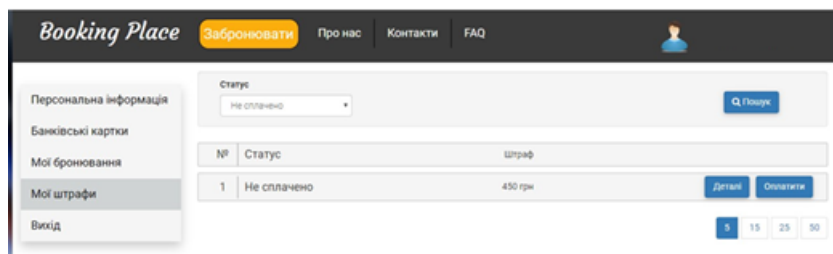


Рис. 3.19. Список не сплачених штрафів

Для того, щоб переглянути деталі штрафу, де відображається час перевищення парковки і коментар, необхідно натиснути на кнопку «Деталі» відповідного штрафу. На рисунку 3.20 показано деталі не сплаченого штрафу.

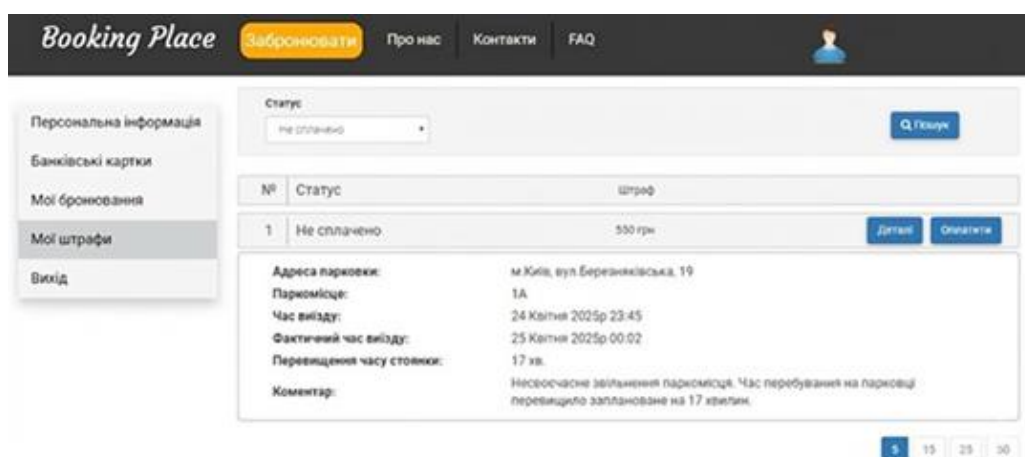


Рис. 3.20. Деталі не сплаченого штрафу

Клієнт мусить сплачувати свої активні справи, адже в разі накопичення штрафів на суму більш ніж 500 гривень, він не зможе бронювати паркомісця. На рисунку 3.21 показано приклад поведінки системи, якщо клієнт має загальну суму штрафів більше ніж 500 грн.



Рис. 3.21. Повідомлення про перевищення ліміту штрафів

Щоб виконати оплату штрафу, клієнт має натиснути на кнопку «Оплатити» після чого має обрати картку, з якої він бажає виконати оплату і натиснути

кнопку «Оплатити» напроти неї. На рисунку 3.22 наведено приклад оплати штрафу.

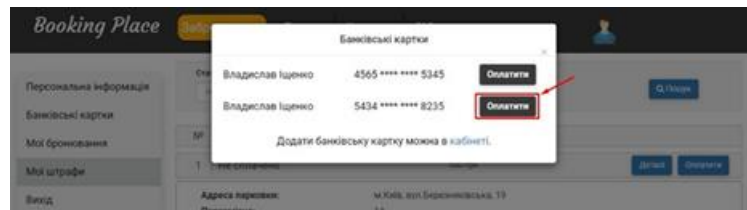


Рис. 3.22. Оплата штрафу

Після того, як штраф буде успішно оплаченим його статус буде змінено на «Сплачено» і його можна буде побачити в історичних штрафах. На рисунку 3.23 показано список сплачених штрафів.

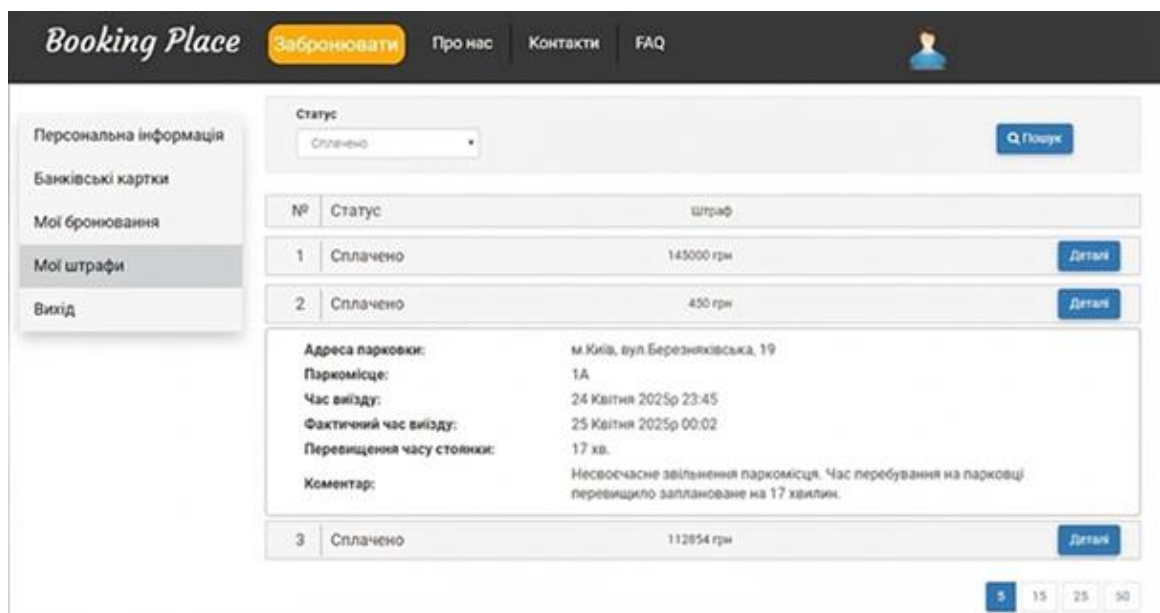


Рис. 3.23. Список сплачених штрафів

Варто зазначити, що сесія авторизованого клієнта триває 2 години, після плинку яких виконається автоматичний перехід на головну сторінку системи.

Висновки до розділу

В даному розділі наведено опис засобів розробки, котрі використовувались під час реалізації програмного продукту. Наведено опис кожного з використаних засобів розробки і вказано їх призначення.

Також, наведено опис методів для серверної та клієнтської частин продукту.

Розроблено інтерфейс з описом функціоналу, котрий має бути доступний кінцевому користувачу.

Було розглянуто користувацький інтерфейс для клієнта. Побудовано керівництво користувача для системи з пошуку та бронювання паркомісць. Розглянуто процес пошуку паркомісць на найближчих парковках.

ВИСНОВКИ

У магістерській роботі було розглянуто проектування та реалізацію веб орієнтованої інформаційної системи пошуку та бронювання паркомісць.

Для досягнення поставленої мети були вирішені наступні задачі:

- розроблено власний кабінет для клієнтів, який дозволив керувати кредитними картками та переглядати історію замовлень і штрафів;
- розроблено пошук найближчих паркувальних майданчиків до обраної адреси;
- розроблено пошук вільних паркомісць на обраній парковці на вказаний проміжок часу та їх бронювання.

В першому розділі проаналізовано предметне серидовище інформаційної системи. Було виконано порівняння розробленої системи з вже існуючими аналогами. Також, було проведено дослідження математичного забезпечення і сформовано задачі. Головними математичними задачами виявились: задача з пошуку відстані між двома географічними точками і задача з пошуку найкоротшого шляху між парковками. Було описано методи за допомогою яких вищеперераховані задачі розв'язувались з їх детальним описом і обґрунтуванням даного вибору.

В другому розділі змодельована та проаналізована веб орієнтована інформаційна система пошуку та бронювання паркомісць. В рамках цього розділу було сформовано вимоги до технічного забезпечення інформаційної системи. Також, даний розділ містить UML-діаграми, які описують процес пошуку та бронювання паркомісць. Даний розділ містить загальний опис архітектури застосунку.

В третьому розділі обґрунтовано вибір засобів розробки для програмного продукту. В цьому розділі наведено специфікацію функцій застосунку з їх коротким описанням. Також, третій розділ містить опис процесу проектування інтерфейсу і керівництво користувача, в якому описано всі функції системи.

На цьому етапі розробки, система є готовою до бета-тестування кінцевими користувачами. Звісно, застосунок має елементи, які потребують

вдосконалення. Наприклад, наразі система не має інтеграції з реальною платіжною системою (на зразок LiqPay), тому кошти не будуть списуватись з рахунків, що не дозволить запустити систему в реальному світі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. About PostgreSQL [Електронний ресурс] // Режим доступу:
2. Bloch J., Effective Java. Boston: Pearson Education (US), 2018. 416 p.
3. Fain Y., Moiseev A. Angular Development with TypeScript. New York: Manning Publications, 2019. 560 p.
4. Fain Y., Moiseev A. Angular Development with TypeScript. New York: Manning Publications, 2019. 560 p.
5. Git [Електронний ресурс] // Режим доступу: <https://git-scm.com/about>
6. Martin R., Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston: Pearson Education (US), 2017. 432 p.
7. MVC Architecture [Електронний ресурс] // Режим доступу: <https://towardsdatascience.com/everything-you-need-to-know-about-mvc-architecture-3c827930b4c1>
8. Oaks S., Java Performance: In-depth Advice for Tuning and Programming Java 8, 11, and Beyond. Sebastopol: O'Reilly Media, Inc, USA, 2020. 450 p.
20. Wilken J., Aden D., Aden J. Angular in Action. New York: Manning Publications, 2019. 520 p.
9. Obe R., Hsu L. PostgreSQL: Up and Running, 3e: A Practical Guide to the Advanced Open-Source Database. Sebastopol: O'Reilly Media, Inc, USA, 2017. 250 p.
10. Oracle vs PostgreSQL [Електронний ресурс] // Режим доступу: <https://www.datavail.com/blog/postgresql-vs-oracle-lets-compare/#:~:text=Oracle%20database%20management%20systems%2C%20the,developed%20by%20volunteer%20developers%20worldwide.>
11. Schoeing H., Mastering PostgreSQL 12: Advanced techniques to build and administer scalable and reliable PostgreSQL database applications, 3rd Edition. Birmingham: Packt Publishing Limited, 2019. 470 p.
12. Spring Framework Overview [Електронний ресурс] // Режим доступу: <https://spring.io/>
13. TypeScript [Електронний ресурс] // Режим доступу: typescriptlang.org

14. Urma R., Fusco M., Mycroft A. Modern Java in Action: Lambdas, streams, functional and reactive programming. New York: Manning Publications, 2018. 592 p.
15. Walls C., Spring Boot in Action. New York: Manning Publications, 2016. 264 p.
16. Walls C., Spring in Action, Fifth Edition. New York: Manning Publications, 2019. 520 p.
17. Інновації у парковках: які технології можуть допомогти міській інфраструктурі? [Електронний ресурс]. Режим доступу: <https://www.forbes.ua/tehnologii/340561-innovacii-v-parkovkah-kakie-tehnologiimogut-pomoch-gorodskoy-infrastrukture>.
18. Опис роботи алгоритму Дейкстри [Електронний ресурс] // Режим доступу: <https://prog-cpp.ru/deikstra>
19. Парковка [Електронний ресурс] // Режим доступу: <https://slovotvir.org.ua/words/parkovka>
20. Розумна парковка та її переваги [Електронний ресурс]. Режим доступу: <https://ula.lantec.ua/statti/rozumna-parkovka-ta-jiji-perevagi>
21. Список країн за кількістю автомобілів на 1000 осіб [Електронний ресурс] // Режим доступу: <http://autonews-ua.info/more.html?id=5545>
22. Сучасні паркувальні системи. [Електронний ресурс]. Режим доступу: <https://dentaauto.ua/raznoe/sovremennyye-parkovochnye-sistemy.html>.
23. Формула гаверсінуса [Електронний ресурс] // Режим доступу: https://uk.wikipedia.org/wiki/%D0%A4%D0%BE%D1%80%D0%BC%D1%83%D0%BB%D0%B0_%D0%B3%D0%B0%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D0%BD%D1%83%D1%81%D0%B0
24. Хо К., Харроп Р., Шефер К. Spring 5 для професіоналів / пер з англ. І. Берштейн. Москва: Вільямс, 2016. 1120 с.
25. Цифра дня: скільки автомобілів на планеті? [Електронний ресурс] // Режим доступу: <http://mmr.net.ua/autoworld/news/94527>
26. Ягузинська І. Ю., Типушова І. О. Сучасні автоматизовані системи паркування автомобілів // Науково-методичний електронний журнал

“Концепт” – 2020. – Т. 35. – С. 156–160. – URL:
<http://ekoncept.ua/2015/95585.htm>.

ДОДАТОК А

Лістинг коду

Вихідні коди серверної частини інформаційної системи

```
public class StringToDoubleConverter extends StdConverter<String, Double> {
    @Override
    public Double convert(String s) {
        return Double.valueOf(s);
    }
}

@Service
public class OpenStreetMapService {
    private static final Logger LOGGER = LoggerFactory.getLogger(OpenStreetAddress.class);
    @Value("${openstreetmap.basPath}")
    private String basePath;
    @Autowired
    private RestTemplate restTemplate;
    public OpenStreetAddress getAddress(String country, String city, String street, String
houseNumber) {
        OpenStreetAddress[] openStreetAddresses = null; try {
            openStreetAddresses = restTemplate
                .getForObject(buildURL(country, city, street, houseNumber),
                    OpenStreetAddress[].class);
        } catch (RestClientException ex) {
            LOGGER.error("Error occurs while reading of coordinates from OpenStreetMap
service.");
        }
        return openStreetAddresses != null && openStreetAddresses.length > 0 ?
            openStreetAddresses[0] : null;
    }
    private String buildURL(String country, String city, String street, String houseNumber) {
        String address = String.format("%s+%s+%s+%s", country, city, street, houseNumber);
        return String.format("%s?q=%s&format=json", basePath, address);
    }
}

public interface CreditCardRepository extends JpaRepository<CreditCard, Long>,
QuerydslPredicateExecutor<CreditCard> {
    @Query("SELECT cc FROM CreditCard cc " + "INNER JOIN FETCH cc.secretKey " +
        "INNER JOIN cc.customer c " +
        "INNER JOIN c.user u " +
        "WHERE u.email = :email AND cc.status <> 'DELETED'")
    List<CreditCard> findActiveByEmail(@Param("email") String email);
    @Query("SELECT cc FROM CreditCard cc " +
        "INNER JOIN FETCH cc.secretKey " +
        "WHERE cc.reference = :cardReference")
    CreditCard findByReference(@Param("cardReference") String cardReference);
}

public interface CustomerRepository extends JpaRepository<Customer, Long>,
QuerydslPredicateExecutor<Customer> {
    Customer findByUserEmail(String email);
    boolean existsByUserEmail(String email);
}
```

```

    }
    public interface SecretKeyRepository extends JpaRepository<SecretKey, Long> {
    }
    @Service
    public class CreditCardService {
private static final Logger LOGGER = LoggerFactory.getLogger(CreditCardService.class);
        @Autowired
private CreditCardRepository creditCardRepository;
        @Autowired
private CustomerRepository customerRepository;
        @Autowired
private SecretKeyRepository secretKeyRepository;
        @Autowired
private CryptoService aesCryptoService;
        @Transactional
private UserContext userContext;
    public CreditCard addCreditCard(CreditCardRequest creditCardRequest) {
        String secretKeyString = RandomUtils.generateRandomString(32);
        String encryptedCardNumber =
            aesCryptoService.encrypt(creditCardRequest.getCardNumber(),
secretKeyString);
        LOGGER.debug("Credit card number successfully encrypted.");
        Customer customer =
            customerRepository.findByUserEmail(userContext.getCurrentUsername());
        CreditCard creditCard = new CreditCard(creditCardRequest.getOwner(),
            encryptedCardNumber, creditCardRequest.getExpirationDate(),
            creditCardRequest.getCvv(), customer, new
SecretKey(secretKeyString));
        creditCard = creditCardRepository.save(creditCard);
        LOGGER.debug("Credit card saved to database.");
        return creditCard;
    }
    @Transactional(readOnly = true)
    public List<CreditCard> getCreditCards() {
        String customerEmail = userContext.getCurrentUsername();
        LOGGER.info("Request to retrieve all cards for user: {}", customerEmail);
        List<CreditCard> creditCards =
            creditCardRepository.findActiveByEmail(customerEmail);
        creditCards.forEach(cc ->
            cc.setCardNumber(aesCryptoService.decrypt(cc.getCardNumber(),
cc.getSecretKey().getKey())));
        return creditCards;
    }
    @Transactional
    public void deleteCreditCard(String cardReference) {
        CreditCard creditCard = creditCardRepository.findByReference(cardReference);
        SecretKey secretKey = creditCard.getSecretKey();
        creditCard.setStatus(CreditCardStatus.DELETED); creditCard.setSecretKey(null);
        creditCardRepository.save(creditCard); secretKeyRepository.delete(secretKey);
    }
    }
}

```

```

@Service
public class CustomerService {
    @Autowired
    private CustomerRepository customerRepository;
    @Autowired
    private UserContext userContext;
    @Autowired
    private CustomerMapper customerMapper;
    public Customer getCustomer(@NotBlank String email) {
        return customerRepository.findOne(QCustomer.customer.user.email.eq(email))
            .orElseThrow(IllegalStateException::new);
    }
    public void updateCustomerInfo(CustomerBase customerBase) { Customer customer =
        getCustomer(userContext.getCurrentUsername()); customerMapper.map(customer,
        customerBase);
        customerRepository.save(customer);
    }
}
@Target(ElementType.PARAMETER)
@Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy = CreditCardNumberValidator.class)
public @interface CreditCardNumberValid {
    String message() default "{EPS019}";
    Class<?>[] groups() default {};
    Class<? extends Payload>[] payload() default {};
}
public class CreditCardNumberValidator implements
    StatelessValidator<CreditCardNumberValid, String> {
    private Pattern pattern = Pattern.compile("\\d{16}");
    @Override
    public boolean isValid(String creditCardNumber, ConstraintValidatorContext
    constraintValidatorContext) {
        return pattern.matcher(creditCardNumber.replaceAll("\\s",
        StringUtils.EMPTY)).matches();
    }
}
@GroupSequence({ CreditCardNumberValid.class, CvvValid.class,
    ExpirationDateValid.class, Default.class
})
public interface CreditCardValidationSequence {
}
public class CvvCodeValidator implements StatelessValidator<CvvValid, String> {
    private final Pattern pattern = Pattern.compile("\\d{3}");
    @Override
    public boolean isValid(String cvv, ConstraintValidatorContext constraintValidatorContext) {
        return pattern.matcher(cvv).matches();
    }
}
@Target(ElementType.PARAMETER)
@Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy = CvvCodeValidator.class)
public @interface CvvValid {

```

```

String message() default "{EPS020}";
    Class<?>[] groups() default {};
    Class<? extends Payload>[] payload() default {};
}
@Target(ElementType.PARAMETER)
@Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy = ExpirationDateValidator.class)
public @interface ExpirationDateValid {
    String message() default "{EPS021}";
    Class<?>[] groups() default {};
    Class<? extends Payload>[] payload() default {};
}
public class ExpirationDateValidator implements StatelessValidator<ExpirationDateValid,
    LocalDate> {
    @Override
    public boolean isValid(LocalDate expirationDate, ConstraintValidatorContext
    constraintValidatorContext) {
        return expirationDate.getDayOfMonth() == 1;
    }
}
@RestController
@Validated(CreditCardValidationSequence.class)
@Scope(proxyMode = ScopedProxyMode.TARGET_CLASS)
public class CreditCardController implements CreditcardsApi {
    private static final Logger LOGGER = LoggerFactory.getLogger(CreditCardController.class);
    @Autowired
    private CreditCardService creditCardService;
    @Autowired
    private CreditCardMapper creditCardMapper;
    @Override
    public ResponseEntity<CreditCardResponse> addCreditCard(@Valid @RequestBody
    CreditCardRequest creditCard) {
        LOGGER.info("Request to add new credit card.");
        CreditCard savedCard = creditCardService.addCreditCard(creditCard);
        savedCard.setCardNumber(creditCard.getCardNumber());
        return ResponseEntity.ok(creditCardMapper.map(savedCard));
    }
    @Override
    public ResponseEntity<List<CreditCardResponse>> getCreditCards() {
        List<CreditCard> creditCards = creditCardService.getCreditCards();
        return ResponseEntity.ok(creditCardMapper.map(creditCards));
    }
    @Override
    public ResponseEntity<Void> deletedCreditCard(@PathVariable("reference") String
    cardReference) { LOGGER.info("Request to delete credit card with reference { }",
    cardReference); creditCardService.deleteCreditCard(cardReference);
        LOGGER.info("Card successfully deleted.");
        return ResponseEntity.ok().build();
    }
}
}
@RestController
@Scope(proxyMode = ScopedProxyMode.TARGET_CLASS)

```

```

    public class CustomerManagementController implements CustomersApi {
    private static final Logger LOGGER =
    LoggerFactory.getLogger(CustomerManagementController.class);
        @Autowired
    private UserContext userContext;
        @Autowired
    private CustomerService customerService;
        @Autowired
    private CustomerMapper customerMapper;
        @Override
    public ResponseEntity<CustomerBase> getCustomer() {
        LOGGER.info("Request to retrieve info about customer.");
        Customer customer =
        customerService.getCustomer(userContext.getCurrentUsername());
        return    ResponseEntity.ok(customerMapper.map(customer));
    }

    @Override
    public ResponseEntity<Void> updateCustomerInfo(CustomerBase customer) {
        LOGGER.info("Request to update customer info.");
        customerService.updateCustomerInfo(customer);
        return    ResponseEntity.ok().build();
    }
    }

    public class OpenRouteServiceException extends RuntimeException {
    }

    public class RouteNotFoundException extends RuntimeException {
    }

    @Component
    public class OpenRouteServiceClient {
    private static final Logger LOGGER =
    LoggerFactory.getLogger(OpenRouteServiceClient.class);
        @Value("${openrouteservice.basePath}")
    private String basePath;
        @Value("${openrouteservice.api_key}")
    private String apiKey;
        @Autowired
    private RestTemplate restTemplate;
    public DirectionResponse getDirection(List<Pair<BigDecimal, BigDecimal>> coordinates) {
        URI uri = UriComponentsBuilder.fromHttpUrl(basePath)
        .path("/directions/driving-car")
        .build().toUri();
        HttpHeaders headers = new HttpHeaders();
        headers.add("Authorization", apiKey);
        ResponseEntity<DirectionResponse> directionResponseEntity = callApi(uri,
        new HttpEntity<>(buildRequest(coordinates), headers)); DirectionResponse
        directionResponse = directionResponseEntity.getBody();
        if (directionResponse == null) {
            LOGGER.error("Route between points isn't found.");
            throw new OpenRouteServiceException();
        }
        return directionResponse;
    }
    }

```

```

private DirectionRequest buildRequest(List<Pair<BigDecimal, BigDecimal>> coordinates) {
    DirectionRequest request = new DirectionRequest()
        .language(DirectionRequest.LanguageEnum.RU)
        .units(DirectionRequest.UnitsEnum.M);
    coordinates.forEach(c -> {
        request.addCoordinatesItem(Arrays.asList( c.getKey().doubleValue(),
            c.getValue().doubleValue()));
        request.addRadiusesItem(-1);
    });
    return request;
}

private ResponseEntity<DirectionResponse> callApi(URI uri, HttpEntity<?> request) {
    try {
        return restTemplate.exchange(uri, HttpMethod.POST, request,
            new ParameterizedTypeReference<DirectionResponse>(){});
    } catch (Exception ex) {
        LOGGER.error("Exception occurs during searching of routes in OpenRouteService:
            {} ", ex.toString());
        throw new OpenRouteServiceException();
    }
}

@Service
public class DistanceService {
    private static final Logger LOGGER = LoggerFactory.getLogger(DistanceService.class);
    @Autowired
    private OpenRouteServiceClient orsClient;

    public double calculateDistanceBetweenPoints(
        Pair<Double, Double> startLatLonPair, Pair<Double, Double> endLatLonPair) {
        double latitudeDistance = Math.toRadians(endLatLonPair.getKey() -
            startLatLonPair.getKey()); double longitudeDistance =
            Math.toRadians(endLatLonPair.getValue() - startLatLonPair.getValue()); double
            startLatitudeInRad = Math.toRadians(startLatLonPair.getKey());
            double endLatitudeInRad = Math.toRadians(endLatLonPair.getKey());
            double a = Math.pow(Math.sin(latitudeDistance / 2), 2) + Math.cos(startLatitudeInRad)
                * Math.cos(endLatitudeInRad) * Math.pow(Math.sin(longitudeDistance / 2), 2);
            double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));

        return Constants.EARTH_RADIUS * c;
    }

    public double calculateDistanceBetweenPoints(Address startAddress, Address endAddress) {
        Pair<Double, Double> startPoint = Pair.of(startAddress.getLatitude().doubleValue(),
            startAddress.getLongitude().doubleValue());
        Pair<Double, Double> endPoint = Pair.of(endAddress.getLatitude().doubleValue(),
            endAddress.getLongitude().doubleValue());
        return calculateDistanceBetweenPoints(startPoint, endPoint);
    }

    public double findShortestDistanceBetweenAddresses(Address startAddress, Address
        endAddress) {
        Pair<BigDecimal, BigDecimal> startPoint = Pair.of(startAddress.getLongitude(),
            startAddress.getLatitude());

```



```

Pair<BigDecimal, BigDecimal> endPoint = Pair.of(endAddress.getLongitude(),
endAddress.getLatitude()); DirectionResponse directionResponse =
orsClient.getDirection(Arrays.asList(startPoint, endPoint)); Iterator<Route>
routeIterator = directionResponse.getRoutes().iterator();
Route route = routeIterator.hasNext() ? routeIterator.next() : null;
if (route == null) {
    LOGGER.error("Route between points {} and {} is not found.", startPoint, endPoint);
    throw new RouteNotFoundException();
}
LOGGER.debug("Direction between addresses found: {}", directionResponse);
return route.getSummary().getDistance();
}
}
public interface OrderItemRepository extends JpaRepository<Order, Long>,
QuerydslPredicateExecutor<Order> {
}
public interface OrderRepository extends JpaRepository<Order, Long>,
QuerydslPredicateExecutor<Order>, PagingAndSortingRepository<Order, Long>,
OrderRepositoryCustom {
    @Query(value = "SELECT o FROM Order o " + "INNER JOIN FETCH o.parking p " +
        "INNER JOIN FETCH o.orderItems oi " + "LEFT JOIN FETCH oi.penalty " +
        "WHERE p.reference = :reference AND o.createdDate BETWEEN :from AND :to " +
        "ORDER BY o.id DESC",
        countQuery = "SELECT count(o) FROM Order o " +
            "INNER JOIN o.parking p " +
            "WHERE p.reference = :reference AND o.createdDate BETWEEN :from AND :to ")
    Page<Order> findAllByParkingReferenceAndCreatedDateBetween(@Param("reference")
String
        parkingReference,
                                @Param("from") LocalDateTime from,
                                @Param("to") LocalDateTime to, Pageable pageable);
    @Query(value = "SELECT o FROM Order o " +
        "INNER JOIN FETCH o.parking p " + "INNER JOIN o.customer c " + "INNER
JOIN c.user u " +
        "INNER JOIN FETCH o.orderItems oi " +
        "LEFT JOIN FETCH oi.penalty " +
        "WHERE u.email = :email AND o.createdDate BETWEEN :from AND :to " +
        "ORDER BY o.id DESC",
        countQuery = "SELECT count(o) FROM Order o " +
            "INNER JOIN o.customer c " +
            "INNER JOIN c.user u " +
            "WHERE u.email = :email AND o.createdDate BETWEEN :from AND :to ")
    Page<Order> findAllByCustomerAndCreatedDateBetween(@Param("email") String email,
                                @Param("from") LocalDateTime from,
                                @Param("to") LocalDateTime to, Pageable pageable);
}
public interface OrderRepositoryCustom {
    List<Order> findAllForParkingsInPeriod(List<String> references, LocalDateTime
startDate, LocalDateTime
endDate);
    Order findByReferenceAndPlaceNumberAndFromToDates(String reference, String
placeNumber,

```

```

        LocalDateTime from, LocalDateTime to);
boolean existsOrderByReferenceAndPlaceNumberAndFromToDates(String reference, String
placeNumber,
        LocalDateTime from, LocalDateTime to);
    }
    public class OrderRepositoryImpl implements OrderRepositoryCustom {
        @PersistenceContext
private EntityManager em;
        @Override
public List<Order> findAllForParkingsInPeriod(List<String> references, LocalDateTime
startDate,
                LocalDateTime endDate) {
                    return new JPAQuery<>(em)
                        .select(order)
                        .from(order)
                        .join(order.parking, parking)
                        .join(parking.address)
                        .where(order.createdDate.between(startDate,
endDate).and(parking.reference.in(references)))
                        .fetch();
                }
        @Override
public Order findByReferenceAndPlaceNumberAndFromToDates(String reference, String
placeNumber,
        LocalDateTime from, LocalDateTime to) {
            return new JPAQuery<>(em)
                .select(order)
                .from(order)
                .join(order.customer).fetchJoin()
                .join(order.orderItems, orderItem).fetchJoin()
                .join(order.parking).fetchJoin()
                .where(order.reference.eq(reference)
                    .and(orderItem.placeNumber.eq(placeNumber))
                    .and(orderItem.dateTimeFrom.eq(from))
                    .and(orderItem.dateTimeTo.eq(to)))
                .fetchFirst();
        }
        @Override
public boolean existsOrderByReferenceAndPlaceNumberAndFromToDates(String reference,
String
        placeNumber, LocalDateTime from, LocalDateTime to) {
            return new JPAQuery<>(em)
                .select(order)
                .from(order)
                .join(order.orderItems, orderItem)
                .where(order.reference.eq(reference)
                    .and(orderItem.placeNumber.eq(placeNumber))
                    .and(orderItem.dateTimeFrom.eq(from))
                    .and(orderItem.dateTimeTo.eq(to)))
                .fetchCount() > 0;
        }
    }

```



```

@Service
public class OrderService {
private static final Logger LOGGER = LoggerFactory.getLogger(OrderService.class);
    @Autowired
private UserContext userContext;
    @Autowired
private OrderRepository orderRepository;
    @Autowired
private OrderMapper orderMapper;
    @Autowired
private ParkingRepository parkingRepository;
    @Autowired
private CustomerRepository customerRepository;
public PageableOrderResponse getCustomerOrders(Integer pageNumber, Integer pageSize,
        LocalDate fromDate, LocalDate toDate) {
    LOGGER.info("Request to retrieve orders page={ }, pageSize={ } for customer { }.",
        pageNumber, pageSize,
        userContext.getCurrentUsername());
    Function<OrderSearchRequest, Page<Order>> orderSearchFunction = request ->
        orderRepository
            .findAllByCustomerAndCreatedDateBetween(request.getIdentifier(),
                request.getFromDateTime(), request.getToDateTime(), request.getPageable());
    return findOrders(userContext.getCurrentUsername(), PageRequest.of(pageNumber,
        pageSize), fromDate, toDate,
        orderSearchFunction);
}
public PageableOrderResponse findOrders(String identifier, Pageable pageable, LocalDate
fromDate,
    LocalDate toDate,
        Function<OrderSearchRequest, Page<Order>> orderSearchFunction) {
    LocalDateTime fromDateTime =
Optional.ofNullable(fromDate).orElse(OrderUtils.defaultFromDate()).atStartOfDay(); LocalDateTime
    toDateTime =
Optional.ofNullable(toDate).orElse(OrderUtils.defaultToDate()).atTime(LocalTime.now());
    OrderSearchRequest request = new OrderSearchRequest(identifier, fromDateTime,
        toDateTime, pageable);
    LOGGER.info("Searching of orders by request { }", request); Page<Order> orderPage =
        orderSearchFunction.apply(request);
    if (!orderPage.hasContent()) {
        LOGGER.debug("No orders are found.");
        return new PageableOrderResponse().total(BigDecimal.ZERO);
    }
    PageableOrderResponse response = new PageableOrderResponse()
        .total(BigDecimal.valueOf(orderPage.getTotalElements()))
        .orders(orderMapper.map(orderPage.getContent()));
    LOGGER.debug("Found orders: { }", response);
    return response;
}
@Transactional
public OrderResponse orderParkingPlace(com.kpi.parking.model.Order order) {
    Parking parking = parkingRepository.findOneWithPlaces(order.getParkingReference());
    Order placedOrder = orderMapper.map(order); enrichOrder(placedOrder, parking);
}

```

```

        orderRepository.save(placedOrder);
        LOGGER.debug("Order successfully placed {}", placedOrder);
        return orderMapper.map(placedOrder);
    }
    private void enrichOrder(Order order, Parking parking) {
        Customer customer =
            customerRepository.findByUserEmail(userContext.getCurrentUsername());
        order.setStatus(OrderStatus.PLACED);
        order.setCustomer(customer);
        order.setParking(parking);
        order.setQuantity(order.getOrderItems().size());
        order.setTotalPrice(calculateTotalPrice(order.getOrderItems(),
            parking.getPrice().getPrice())); order.getOrderItems().forEach(oi -> oi.setOrder(order));
    }
    private BigDecimal calculateTotalPrice(Set<OrderItem> orderItems, BigDecimal
    pricePerHour) {
        long totalBookingTime = orderItems.stream()
            .map(oir -> Duration.between(oir.getDateTimeFrom(), oir.getDateTimeTo()))
            .mapToLong(Duration::toMinutes)
            .sum();
        double hours = ((double) totalBookingTime) / 60;
        return pricePerHour.multiply(BigDecimal.valueOf(hours));
    }
}

@GroupSequence({ ParkingKnownAndActive.class, TimestampIsCorrect.class,
    TimestampToIsAfterTimestampFrom.class, PlacesAreKnown.class,
    PlacesAreAvailable.class,
    Default.class
})
public interface OrderValidationSequence {
}
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy = PlacesAreAvailableValidator.class)
public @interface PlacesAreAvailable { String message() default "{EPS029}"; Class<?>[]
    groups() default {};
    Class<? extends Payload>[] payload() default {};
}
public class PlacesAreAvailableValidator implements
    StatelessValidator<PlacesAreAvailable, Order> {
    @Autowired
    private PlaceRepository placeRepository;
    @Override
    public boolean isValid(Order order, ConstraintValidatorContext
    constraintValidatorContext) { Iterator<OrderItemRequest> iterator =
        order.getOrderItems().iterator(); OrderItemRequest orderItem = iterator.hasNext() ?
        iterator.next() : null;
        if (orderItem == null) {
            return false;
        }
        return placeRepository.isPlaceAvailable(order.getParkingReference(),
            orderItem.getPlaceNumber(),

```

```

        ParserUtils.parseToLocalDateTime(orderItem.getFrom()),
        ParserUtils.parseToLocalDateTime(orderItem.getTo()));
    }
}
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy = PlacesAreKnownValidator.class)
public @interface PlacesAreKnown {
String message() default "{EPS028}";
Class<?>[] groups() default {};
Class<? extends Payload>[] payload() default {};
}
public class PlacesAreKnownValidator implements StatelessValidator<PlacesAreKnown,
Order> {
    @Autowired
private PlaceRepository placeRepository;
    @Override
public boolean isValid(Order order, ConstraintValidatorContext constraintValidatorContext) {
    List<String> placeNumbers = order.getOrderItems().stream()
        .map(OrderItemRequest::getPlaceNumber)
        .collect(Collectors.toList());
    return placeRepository.existsAvailablePlacesInParking(order.getParkingReference(),
        placeNumbers);
}
}
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy = TimestampToIsAfterTimestampFromValidator.class)
public @interface TimestampToIsAfterTimestampFrom {
String message() default "{EPS027}";
Class<?>[] groups() default {};
Class<? extends Payload>[] payload() default {};
}
public class TimestampToIsAfterTimestampFromValidator implements
StatelessValidator<TimestampToIsAfterTimestampFrom, OrderItemRequest> {
    @Autowired
private com.kpi.parking.parking.validation.TimestampToIsAfterTimestampFromValidator
validator;
    @Override
public boolean isValid(OrderItemRequest orderItemRequest, ConstraintValidatorContext
constraintValidatorContext) {
    return validator.isValid(new Object[]{orderItemRequest.getPlaceNumber(),
orderItemRequest.getFrom(), orderItemRequest.getTo()},
        constraintValidatorContext);
}
}
}
@RestController
@Validated(OrderValidationSequence.class)
@Scope(proxyMode = ScopedProxyMode.TARGET_CLASS)
public class OrdersController implements OrdersApi {
private static final Logger LOGGER = LoggerFactory.getLogger(OrdersController.class);
    @Autowired

```

```

private OrderService orderService;
    @Override
    @CheckCustomer
public ResponseEntity<PageableOrderResponse> getCustomerOrders(
    @RequestParam(value = "pageNumber") Integer pageNumber,
    @RequestParam(value = "pageSize") Integer pageSize,
    @RequestParam(value = "fromDate", required = false)
    @DateTimeFormat(iso = DateTimeFormat.ISO.DATE) LocalDate fromDate,
    @RequestParam(value = "toDate", required = false)
    @DateTimeFormat(iso = DateTimeFormat.ISO.DATE) LocalDate toDate) {
    return ResponseEntity.ok(orderService.getCustomerOrders(pageNumber, pageSize,
        fromDate, toDate));
}

    @Override
    @CheckCustomer
public ResponseEntity<OrderResponse> order(@Valid @RequestBody Order order) {
    LOGGER.info("Request to place order {}", order);
    return ResponseEntity.ok(orderService.orderParkingPlace(order));
}

}
@RestController
@Validated(ParkingSequence.class)
@Scope(proxyMode = ScopedProxyMode.TARGET_CLASS)
public class ParkingController implements ParkingsApi {
private static final Logger LOGGER = LoggerFactory.getLogger(ParkingController.class);
    @Autowired
private ParkingService parkingService;
    @Override
    @CheckCustomer
public ResponseEntity<List<ParkingResponse>> getAvailableParkings() {
    LOGGER.info("Request to retrieve list of available parkings.");
    return ResponseEntity.ok(parkingService.getAvailableParkings());
}

    @Override
    @CheckCustomer
    @TimestampToIsAfterTimestampFrom(groups =
TimestampToIsAfterTimestampFrom.class)
        public ResponseEntity<List<String>> getAvailablePlaces(
            @ParkingKnownAndActive(groups = ParkingKnownAndActive.class)
            @PathVariable("reference") String reference,
            @TimestampIsCorrect(groups = TimestampIsCorrect.class)
            @RequestParam("from") String from,
            @RequestParam("to") String to) {
            LOGGER.info("Request to find available parking place");
            return ResponseEntity.ok(parkingService.findAvailablePlace(reference, from, to));
        }

    @Override
    @CheckCustomer
    @TimestampToIsAfterTimestampFrom(groups =
TimestampToIsAfterTimestampFrom.class)
public ResponseEntity<NearestParking> findNearestParkingWithPlaces(
    @ParkingKnownAndActive(groups = ParkingKnownAndActive.class)

```

```

        @PathVariable("reference") String reference,
        @TimestampIsCorrect(groups = TimestampIsCorrect.class)
        @RequestParam("from") String from,
        @RequestParam("to") String to) {
    return ResponseEntity.ok(parkingService.findNearestParking(reference, from, to));
}
}
@Service
public class ParkingService {
private static final Logger LOGGER = LoggerFactory.getLogger(ParkingService.class);
private static final Comparator<com.kpi.parking.model.ParkingResponse>
    PARKING_RESPONSE_COMPARATOR = Comparator
        .comparing(com.kpi.parking.model.ParkingResponse::getAddress,
        Comparator.comparing(com.kpi.parking.model.Address::getCountry))
        .thenComparing(com.kpi.parking.model.ParkingResponse::getAddress,
        Comparator.comparing(com.kpi.parking.model.Address::getCity))
        .thenComparing(com.kpi.parking.model.ParkingResponse::getAddress,
        Comparator.comparing(com.kpi.parking.model.Address::getStreet))
        .thenComparing(com.kpi.parking.model.ParkingResponse::getAddress,
        Comparator.comparing(com.kpi.parking.model.Address::getStreetNumber));
    @Value("${order.datetime.to.offset}")
private Long dateTimeToOffset;
    @Value("${parkings.radius}")
private Double searchRadius;
    @Autowired
private ParkingRepository parkingRepository;
    @Autowired
private PlaceRepository placeRepository;
    @Autowired
private ParkingProvider parkingProvider;
    @Autowired
private ParkingMapper parkingMapper;
    @Autowired
private AddressMapper addressMapper;
    @Autowired
private OrderRepository orderRepository;
    @Autowired
private UserContext userContext;
    @Autowired
private AddressProvider addressProvider;
    @Autowired
private AddressService addressService;
    @Autowired
private SupplierRepository supplierRepository;
    @Autowired
private DistanceService distanceService;
    @Autowired
private OrderService orderService;
public List<com.kpi.parking.model.ParkingResponse> getAvailableParkings() {
    List<Parking> parkings = parkingRepository.findAvailableParkings();
    return parkings.stream()
        .map(parkingMapper::mapWithoutDetails)

```

```

        .sorted(PARKING_RESPONSE_COMPARATOR)
        .collect(Collectors.toList());
    }
    public List<String> findAvailablePlace(@NotBlank String parkingReference,
                                          @NotBlank String from,
                                          @NotBlank String to) {
        Place place = placeRepository.findAvailablePlaceOnParking(
            parkingReference, ParserUtils.parseToLocalDateTime(from), parseToDateTime(to));
        if (place == null) {
            return Collections.emptyList();
        }
        return Collections.singletonList(place.getPlaceNumber());
    }
    public com.kpi.parking.model.NearestParking findNearestParking(@NotBlank String
reference,
                                          @NotBlank String from,
                                          @NotBlank String to) {
        LocalDateTime fromDateTime = ParserUtils.parseToLocalDateTime(from);
        LocalDateTime toDateTime = parseToDateTime(to);
        Address selectedParkingAddress =
            parkingRepository.findOneWithAddress(reference).getAddress();
        List<Parking> availableParkings = parkingRepository.findAvailableParkings();
        // search parkings in radius with available place
        Map<Parking, Place> parkingPlaceMap = availableParkings.stream()
            .filter(p -> !p.getReference().equals(reference)
                && isParkingInRadius(selectedParkingAddress, p.getAddress()))
            .collect(HashMap::new, (m, v) -> m.put(v, placeRepository
                .findAvailablePlaceOnParking(v.getReference(), fromDateTime, toDateTime)),
                HashMap::putAll);
        parkingPlaceMap = parkingPlaceMap.entrySet().stream()
            .filter(e -> e.getValue() != null)
            .collect(Collectors.toMap(Map.Entry::getKey, Map.Entry::getValue));
        // search nearest parking
        Map.Entry<Parking, Place> nearestParkingWithPlace =
            parkingPlaceMap.entrySet().stream()
                .map(p -> Pair.of(p, distanceService.findShortestDistanceBetweenAddresses(
                    selectedParkingAddress, p.getKey().getAddress())))
                .reduce(BinaryOperator.minBy(Comparator.comparingDouble(Pair::getValue)))
                .map(Pair::getKey)
                .orElse(null);
        if (nearestParkingWithPlace == null) {
            return null;
        }
        return parkingMapper.map(nearestParkingWithPlace.getKey(),
            nearestParkingWithPlace.getValue());
    }
    private boolean isParkingInRadius(Address startAddress, Address endAddress) {
        return distanceService.calculateDistanceBetweenPoints(startAddress, endAddress) <=
            searchRadius;
    }
    private LocalDateTime parseToDateTime(String dateTime) {
        return ParserUtils.parseToLocalDateTime(dateTime).plusMinutes(dateTimeToOffset);
    }
}

```



```

    }
    public interface PenaltyRepository extends JpaRepository<Penalty, Long>,
        QuerydslPredicateExecutor<Penalty> {
        @Query(value = "SELECT p FROM Penalty p " + "INNER JOIN FETCH p.orderItem oi " +
            " + "INNER JOIN FETCH oi.order o " +
            "INNER JOIN FETCH o.parking " + "INNER JOIN p.customer c " + "INNER JOIN " +
            "c.user u " +
            "WHERE u.email = :username AND p.status = :status " +
            "ORDER BY p.id DESC",
            countQuery = "SELECT count(p) FROM Penalty p " +
                "INNER JOIN p.orderItem " + "INNER JOIN p.customer c " + "INNER JOIN " +
                "c.user u " +
                "WHERE u.email = :username AND p.status = :status")
        Page<Penalty> findPageByUsernameAndStatus(@Param("username") String username,
            @Param("status") PenaltyStatus status, Pageable pageable);
        Penalty findByReference(String reference);
        @Query("SELECT p FROM Penalty p " + "INNER JOIN p.customer c " + "INNER " +
            "JOIN c.user u " +
            "WHERE u.email = :username AND p.status = 'ACTIVE'")
        List<Penalty> findActivePenaltiesByUsername(@Param("username") String username);
    }
    @Service
    public class PenaltyService {
    private static final Logger LOGGER = LoggerFactory.getLogger(PenaltyService.class);
        @Value("${penalty.total.boundary}")
        private double boundaryPenalty;
        @Autowired
        private PenaltyRepository penaltyRepository;
        @Autowired
        private UserContext userContext;
        @Autowired
        private PenaltyMapper penaltyMapper;
        @Transactional(readOnly = true)
        public PageablePenaltyResponse getPenalties(int pageNumber, int pageSize, PenaltyStatus
            penaltyStatus) {
            LOGGER.info("Request to retrieve penalties in status { } for user { }.", penaltyStatus,
                userContext.getCurrentUsername());
            Pageable pageable = PageRequest.of(pageNumber, pageSize);
            Page<Penalty> penaltyPage = penaltyRepository.findPageByUsernameAndStatus(
                userContext.getCurrentUsername(), penaltyStatus, pageable);
            if (!penaltyPage.hasContent()) {
                LOGGER.debug("No penalties are found.");
                return new PageablePenaltyResponse().total(BigDecimal.ZERO);
            }
            PageablePenaltyResponse response = new PageablePenaltyResponse()
                .total(BigDecimal.valueOf(penaltyPage.getTotalElements()))
                .penalties(penaltyMapper.map(penaltyPage.getContent()));
            LOGGER.debug("Found penalties: { }", response);
            return response;
        }
    }
    @Transactional

```

```

        public void payPenalty(String penaltyReference, Payment payment) { Penalty penalty =
            penaltyRepository.findByReference(penaltyReference);
            penalty.setStatus(PenaltyStatus.CLOSED);
            penaltyRepository.save(penalty);
        }

        @Transactional(readOnly = true)
        public boolean isTotalPenaltyExceedsLimit() {
            String username = userContext.getCurrentUsername();
            LOGGER.info("Request to check amount of penalties for customer { }.", username);
            List<Penalty> penalties =
                penaltyRepository.findActivePenaltiesByUsername(username);
            BigDecimal totalPenalty = penalties.stream()
                .map(Penalty::getTotal)
                .reduce(BigDecimal.ZERO, BigDecimal::add);
            return totalPenalty.doubleValue() > boundaryPenalty;
        }
    }

    @RestController
    public class PenaltyController implements PenaltiesApi {
        private static final Logger LOGGER = LoggerFactory.getLogger(PenaltyController.class);

        @Autowired
        private PenaltyService penaltyService;

        @Override
        @CheckCustomer
        public ResponseEntity<PageablePenaltyResponse>
            getCustomerPenalties(@RequestParam(value =
                "status") String status,
                                @RequestParam(value = "pageNumber") Integer
                                    pageNumber,
                                @RequestParam(value = "pageSize") Integer pageSize) {
            return ResponseEntity.ok(penaltyService.getPenalties(pageNumber, pageSize,
                PenaltyStatus.fromValue(status)));
        }

        @Override
        @CheckCustomer
        public ResponseEntity<Void> payFine(@PathVariable("reference") String reference,
            @Valid Payment payment) {
            LOGGER.info("Request to pay penalty { } using card { }.", reference, payment);
            penaltyService.payPenalty(reference, payment);
            return ResponseEntity.ok().build();
        }

        @Override
        public ResponseEntity<List<String>> getPenaltyStatuses() {
            return ResponseEntity.ok(Stream.of(PenaltyStatus.values())
                .map(PenaltyStatus::getValue)
                .collect(Collectors.toList()));
        }

        @Override
        @CheckCustomer
        public ResponseEntity<Boolean> isTotalPenaltyExceedsLimit() {
            return ResponseEntity.ok(penaltyService.isTotalPenaltyExceedsLimit());
        }
    }

```



```

    }
    @Service
    public class VisitService {
private static final Logger LOGGER = LoggerFactory.getLogger(VisitService.class);
        private static final String PENALTY_COMMENT_TEMPLATE = "Несвоєчасне звільнення паркомісця. Час перебування на парковці перевищило заплановане на %s хвилин.";
        @Autowired
        public OrderRepository orderRepository;
        @Autowired
        private ParkingRepository parkingRepository;
        @Transactional
        public void enterParking(VisitRequest visitRequest) {
            Order order =
                orderRepository.findByReferenceAndPlaceNumberAndFromToDates(visitRequest.getOrderRef
                    erence(), visitRequest.getPlaceNumber(),
                    ParserUtils.parseToLocalDateTime(visitRequest.getFrom()),
                    ParserUtils.parseToLocalDateTime(visitRequest.getTo()));
            Parking parking = order.getParking();
            order.setStatus(OrderStatus.ACTIVE);
            order.getOrderItems().forEach(oi -> updateStatusOfParkingPlaces(parking,
                PlaceStatus.OCCUPIED, oi.getPlaceNumber()));
            LOGGER.debug("Order successfully update according to request.");
            orderRepository.save(order); parkingRepository.save(parking);
        }

        @Transactional
        public void leaveParking(VisitRequest visitRequest) {
            Order order =
                orderRepository.findByReferenceAndPlaceNumberAndFromToDates(visitRequest.getOrderRef
                    erence(), visitRequest.getPlaceNumber(),
                    ParserUtils.parseToLocalDateTime(visitRequest.getFrom()),
                    ParserUtils.parseToLocalDateTime(visitRequest.getTo()));
            Parking parking = order.getParking();
            LocalDateTime now = LocalDateTime.now();
            LocalDateTime departDateTime =
                now.truncatedTo(ChronoUnit.MINUTES).plusMinutes(1L); Price price =
                order.getParking().getPrice();
            Customer customer = order.getCustomer();
            order.setStatus(OrderStatus.CLOSED_SUCCESSFULLY); order.getOrderItems().stream()
                .peek(oi -> oi.setDateTimeDepart(departDateTime))
                .filter(oi -> now.isAfter(oi.getDateTimeTo()))
                .forEach(oi -> {
                    order.setStatus(OrderStatus.CLOSED_WITH_PENALTY);
                    oi.setPenalty(calculatePenalty(oi, customer, price, departDateTime));
                    updateStatusOfParkingPlaces(parking, PlaceStatus.AVAILABLE,
                        oi.getPlaceNumber());
                });
            LOGGER.debug("Order successfully update according to request.");
            orderRepository.save(order); parkingRepository.save(parking);
        }
        private Penalty calculatePenalty(OrderItem orderItem, Customer customer, Price price,
            LocalDateTime
            departDateTime) {

```

```

        BigDecimal latenessTime = BigDecimal.valueOf(Duration
            .between(orderItem.getDateTimeTo(),    departDateTime).getSeconds())
            .divide(BigDecimal.valueOf(60L),    RoundingMode.CEILING); BigDecimal penalty =
            price.getPenalty().multiply(latenessTime);
        return new Penalty(penalty, customer, orderItem, String.format(PENALTY_COMMENT_TEMPLATE,
            latenessTime));
    }
    private void updateStatusOfParkingPlaces(Parking parking, PlaceStatus newStatus,
        @NotBlank String
            placeNumber) { parking.getPlaces().stream()
                .filter(p -> placeNumber.equals(p.getPlaceNumber()))
                .forEach(p -> p.setStatus(newStatus));
    }
}
    @RestController
    @Validated(VisitParkingValidationSequence.class)
    @Scope(proxyMode = ScopedProxyMode.TARGET_CLASS)
    public class VisitController implements ParkingApi {
    private static final Logger LOGGER = LoggerFactory.getLogger(VisitController.class);
        @Autowired
    private VisitService visitService;
        @Override
    public ResponseEntity<Void> enterParking(@Valid
        @DateTimeFromIsStarted(groups = DateTimeFromIsStarted.class)
        @DateTimeHasNotPassed(groups = DateTimeHasNotPassed.class)
        VisitRequest
            enterRequest) {
        LOGGER.info("Request to enter on parking {}", enterRequest);
        visitService.enterParking(enterRequest);
        return    ResponseEntity.ok().build();
    }
        @Override
    public ResponseEntity<Void> leaveParking(@Valid VisitRequest leaveRequest) {
        LOGGER.info("Request to leave parking {}", leaveRequest);
        visitService.leaveParking(leaveRequest);
        return    ResponseEntity.ok().build();
    }
    }
}

```

ДОДАТОК Б

Лістинг коду

Вихідні коди клієнтської частини інформаційної системи

```
import { Injectable } from '@angular/core';
import { RequestService } from './request.service';

@Injectable({
  providedIn: 'root'
})
export class CardService {
  constructor(private request: RequestService) {
  }
  getCards() {
return this.request.get('creditcards', true);
  }
  addCard(creditCard) {
    let expirationDate = creditCard.expirationDate.value.split('/');
    const body = {
      owner: creditCard.creditName.value,
      cardNumber: creditCard.cardNumber.value.replace(/\s/g,
        ''), expirationDate: `20${expirationDate[1]}-${
        ${expirationDate[0]}-01`, cvv: creditCard.cvv.value
    };
return this.request.post('creditcards', body);
  }
  deleteCard(reference){
    return this.request.delete('creditcards/' + reference);
  }
}

import { Injectable } from '@angular/core';
import { RequestService } from './request.service';
@Injectable({
  providedIn: 'root'
})
export class CustomerService {
  constructor(private request: RequestService) {
  }
  getInfo() {
return this.request.get('customers', true);
  }
  updateInfo(customerInfo) {
    const body = {
      lastName: customerInfo.lastName.value,
      firstName: customerInfo.firstName.value,
      phoneNumber:
        customerInfo.phoneNumber.value
    };
return this.request.put('customers', body);
  }
}

import { Injectable } from '@angular/core';
```

```

import {RequestService} from "../request.service";
@Injectable({
  providedIn: 'root'
})
export class OrderService {
  constructor(private requestService: RequestService) {
  }
  getOrders(filters) {
    const query = new Map([
      ['reference', filters.reference.toString()],
      ['fromDate', (filters.fromDate.year + '-' + ((filters.fromDate.month < 10) ? '0' +
filters.fromDate.month : filters.fromDate.month) + '-' + ((filters.fromDate.day < 10) ? '0' +
filters.fromDate.day : filters.fromDate.day)).toString()],
      ['toDate', (filters.toDate.year + '-' + ((filters.toDate.month < 10) ? '0' + filters.toDate.month :
filters.toDate.month) + '-' + ((filters.toDate.day < 10) ? '0' + filters.toDate.day :
filters.toDate.day)).toString()],
      ['pageNumber', filters.pageNumber],
      ['pageSize', filters.pageSize]
    ]);
    return this.requestService.get('parkings/' + filters.reference + '/orders', true, query);
  }
  getCustomerOrders(filters){
    const query = new Map([
      ['fromDate', (filters.fromDate.year + '-' + ((filters.fromDate.month < 10) ? '0' +
filters.fromDate.month : filters.fromDate.month) + '-' + ((filters.fromDate.day < 10) ? '0' +
filters.fromDate.day : filters.fromDate.day)).toString()],
      ['toDate', (filters.toDate.year + '-' + ((filters.toDate.month < 10) ? '0' + filters.toDate.month :
filters.toDate.month) + '-' + ((filters.toDate.day < 10) ? '0' + filters.toDate.day :
filters.toDate.day)).toString()], ['pageNumber', filters.pageNumber],
      ['pageSize', filters.pageSize]
    ]);
    return this.requestService.get('orders/', true, query);
  }
  addOrder(parkingReference, placeNumber, from, to){
    const body = {
      parkingReference:
      parkingReference, orderItems:[{
        placeNumber:
        placeNumber, from: from,
        to: to
      }]
    };
    return this.requestService.post('orders', body);
  }
}

import {Injectable} from '@angular/core';
import {RequestService} from "../request.service";
@Injectable({
  providedIn: 'root'
})
export class ParkingService {
  constructor(private request: RequestService) {
  }
  getAvailableParking(){

```

```

return this.request.get('parkings/available', true);
    }
    getAvailablePlaces(parkingReference, from, to){
        const query = new Map([
            ['from', from],
            ['to', to]
        ]);
return this.request.get('parkings/'+parkingReference+'/places/available', true, query);
    }
    getAvailableNearestPlaces(parkingReference, from, to){
        const query = new Map([
            ['from', from],
            ['to', to]
        ]);
return this.request.get('parkings/'+parkingReference+'/nearest', true, query);
    }
}

import { Injectable } from
'@angular/core';
import { RequestService } from
"./request.service"; import
{ CookieService } from "ngx-cookie-
service"; import { Router } from
"@angular/router";
import { environment } from
"../environments/environment";
@Injectable
e({
    providedIn:
    'root'
})
export class
PenaltyService {
    construct
    or(
private requestService: RequestService,
private cookie: CookieService,
private router: Router) {
    }
    public getPenaltyStatuses() {
return this.requestService.get('penalties/statuses', true);
    }
    public
getPenalties(filters){
        const query = new
        Map([ ['status',
            filters.status],
            ['pageNumber', filters.pageNumber],
            ['pageSize', filters.pageSize]
        ]);
return this.requestService.get('penalties/customer', true, query);

```

```

    }
    getTotalPenalty(){
return this.requestService.get('penalties/total/check', true);
    }
    public payPenalty(penaltyReference, cardReference){
    const body = {
        cardReference: cardReference
    };
return this.requestService.patch('penalties/' + penaltyReference + '/customer',body);
    }
}

import { Injectable } from '@angular/core';
import { RequestService } from './request.service';

@Injectable({
    providedIn: 'root'
})
export class TimeService {
    constructor(private request: RequestService) {
    }
    private minutes = ['00', '15', '30', '45'];
    private months = [
        'Січня',
        'Лютого',
        'Березня',
        'Квітня',
        'Травня',
        'Червня',
        'Липня',
        'Серпня',
        'Вересня',
        'Жовтня',
        'Листопада',
        'Груденя',
    ];
    getTimesItems() {
        const items: string[] = [];
        let hour;
for (let i = 0; i < 24; i++) {
        hour = this.convertTime(i);
        this.minutes.forEach(minute => items.push(hour + ':' + minute));
    }
        console.log(items);
return items;
    }

    getMonth(number) {
return this.months[number - 1];
    }
    convertTime(time) {
return time < 10 ? '0' + time : time;
    }
    calculateTimeDuration(from, to) {
        let duration = new Date(to).valueOf() - new Date(from).valueOf();

```

```

    let hours = duration / 1000 / 3600;
    let intHours = Math.floor(hours);
    let minutes = ((duration - (intHours * 3600 * 1000)) / (1000 * 60)).toString();
    let stringDate =
        "";
    console.log(intHours);
    if (intHours > 24) {
        let days = Math.floor(intHours / 24);
        intHours = intHours - days * 24;
        stringDate = days + " дн ";
    }
    if(intHours){
        stringDate += intHours + " год. ";
    }
    if(minutes != '0') {
        console.log(minutes);
        stringDate += this.convertTime(minutes) + " хв.";
    }
    return stringDate;
}

dateToString(date) {
    if (date != null && date.length > 0) { date = new Date(date);
        return date.getDate() + ' ' + this.getMonth(date.getMonth() + 1) + ' ' + date.getFullYear() + 'р '
            +
            this.convertTime(date.getHours()) + ':' + this.convertTime(date.getMinutes());
    }
    return "";
}

}

import { Component, OnInit } from '@angular/core';
import { ReceiverService } from "../_services/receiver.service";
import { CardService } from
    "../_services/card.service"; import { Title } from
    "@angular/platform-browser"; import
    { TimeService } from "../_services/time.service";
import { OrderService } from
    "../_services/order.service";
import { environment } from "../../environments/environment";
import { CookieService } from "ngx-cookie-service";
import { Router } from "@angular/router";
declare var $: any;
@Component({
    selector: 'app-checkout',
    templateUrl: './checkout.component.html',
    styleUrls: ['./checkout.component.less', './card.less']
})
export class CheckoutComponent implements OnInit {
    creditCards = [];
    selectedCreditCard = null;
    buttonCardText = 'Обрати банківську картку';

```

```

    orderInfo;
    dateFrom: Date;
    dateTo: Date;
    orderTotal;
    timeStaying = "";
    dateString = {
from: "",
to: ""
    };
    constructor(
private receiverService: ReceiverService,
private cardService: CardService,
private title: Title,
private timeService: TimeService,
private orderService: OrderService,
private cookie: CookieService,
private router: Router
    ) {
    }

    ngOnInit() {
this.title.setTitle('Бронювання');
this.orderInfo = this.receiverService.getOrderInfo();
if (!this.orderInfo || !this.cookie.check('token')) {
    window.location.href = environment.siteUrl;
} else {
    this.processOrderInfo();
    this.calculateTotalPrice();
    ;
    this.getCreditCards();
}
    }

    processOrderInfo() {
        let dateFrom = this.orderInfo.dateFrom.split('-')
            .reverse().join('-'); let dateTo =
            this.orderInfo.dateTo.split('-').reverse().join('-');
this.dateFrom = new Date(dateFrom + 'T' + this.orderInfo.timeFromGroup + ':00');
this.dateTo = new Date(dateTo + 'T' + this.orderInfo.timeToGroup + ':00');
this.dateString.from = this.timeService.dateToString(dateFrom);
this.dateString.to = this.timeService.dateToString(dateTo);
    }

    calculateTotalPrice() {
        this.timeStaying = this.timeService.calculateTimeDuration(this.dateFrom,
            this.dateTo); let duration = this.dateTo.valueOf() - this.dateFrom.valueOf();
        let hours = duration / 1000 / 3600;
this.orderTotal = hours * this.orderInfo.stateGroup.object.price;
    }

    getCreditCards() {
        const promise = new Promise((resolve, reject) => {
            this.cardService.getCards().toPromise()
                .then((res: any) => {
                    let cards = [];
                    res.forEach(function (card)
                        {

```



```

    let datePasts = card.expirationDate.split('-');
    console.log(datePasts);
    cards.push({
      owner: card.owner,
      cardNumber: card.cardNumber,
      expirationDate: datePasts[1] + '/' + datePasts[0].toString().substr(-2),
      reference: card.reference
    });
  });
  this.creditCards = cards;
  resolve();
},
err => {
  // Error
  reject(err);
}
).catch();
});
}

selectCreditCard(creditCard) {
  this.selectedCreditCard = creditCard;
  this.buttonCardText = 'ЗМІНИТИ БАНКІВСЬКУ
  картку';
}

addOrder() {
  if (this.selectedCreditCard !== null) {
    this.showLoader();
    const promise = new Promise((resolve,
    reject) => { this.orderService.addOrder(
    this.orderInfo.stateGroup.object.reference,
    this.orderInfo.place,
    this.orderInfo.dateFrom + " " + this.orderInfo.timeFromGroup,
    this.orderInfo.dateTo + " " + this.orderInfo.timeToGroup
    ).toPromise()
    .then((res: any) => {
      if(res !== null){
        this.router.navigate(['succes
        s']);
      }
      resolve();
    },
    err => {
      // Error
      reject(err);
    }
    ).catch();
  });
} else {
  $('#select-card').click();
}

}

showLoader(){
  $('#book_button').hide();
  $('.block-loader').show();
}

```

```

    }
  }
import { Component, ComponentFactoryResolver, OnInit, ViewChild, ViewContainerRef } from
'@angular/core';
import { environment } from "../../environments/environment";
import { CookieService } from "ngx-cookie-service";
import { CustomerService } from "../_services/customer.service";
import { Title } from "@angular/platform-browser";
import * as $ from 'jquery';
import { NotificationComponent } from "../../layout/notification/notification.component";
@Component({
  selector: 'app-cus-account',
  templateUrl: './cus-account.component.html',
  styleUrls: ['./cus-account.component.less']
})
export class CusAccountComponent implements OnInit {
  typePage: string =
'customer/account'; typeClient:
string = 'customer'; editableInfo:
boolean = false;
  info = {
firstName: { value: "", error: false },
  lastName: { value: "", error: false },
  email: { value: "", error: false },
phoneNumber: { value: "", error: false }
  };
  componentRef: any;
  @ViewChild('viewContainerRefCustomerAccount', { read: ViewContainerRef, static: false })
  entry:
  ViewContainerRef;
  constructor(
private cookie: CookieService,
private customerService: CustomerService,
private title: Title,
private resolver: ComponentFactoryResolver,
  ) {
  }
  ngOnInit() {
if (!this.cookie.check('token')) {
  window.location.href = environment.siteUrl
}
this.title.setTitle("Мій кабінет");
this.getInfo();
  }
  getInfo(): void {
const promise = new Promise((resolve, reject) => {
  this.customerService.getInfo().toPromise()
    .then((res: any) => {
      console.log(res);
      this.info.firstName.value =
res.firstName;
      this.info.lastName.value =

```

```

        res.lastName; this.info.email.value
        = res.email;
        this.info.phoneNumber.value =
        res.phoneNumber; resolve();
    },
    err => {
        // Error
        reject(err);
    }
    );
});
}

updateInfo():
    void {
if (this.validation()) {
    this.showLoader();
    const promise = new Promise((resolve, reject) => {
        this.customerService.updateInfo(this.info).toPromise()
        .then((res: any) => {
            this.setNotification('success', 'updateAccountCust', '107');
            this.hideLoader();
            this.editable();
            resolve();
        },
        err => {
            // Error
            this.setNotification('error', 'updateAccountCust', err['error'][0].code);
            this.hideLoader();
            reject(err);
        }
        ).catch();
    });
}

    console.log(this.info);
}

    editable(): void {
this.editableInfo = !this.editableInfo;
}

    validation(): boolean {
        const regexp = new RegExp(/^(^<>()\[\]\.\,\;\s@""]+(\.^[^<>()\[\]\.\,\;\s@""]+)*)([".+"])?@((\[[0-9]{1,3}\. [0-9]{1,3}\. [0-9]{1,3}\. [0-9]{1,3}\]|([a-zA-Z-0-9]+\.)+[a-zA-Z]{2,}))$/);
        let error = false;
if (this.info.firstName.value.length < 2) {
    this.info.firstName.error = true;
    error = true;
} else {
    this.info.firstName.error = false;
}
if (this.info.lastName.value.length < 2) {
    this.info.lastName.error = true;
    error = true;
} else {
    this.info.lastName.error = false;
}

```

```

    }
    if (this.info.phoneNumber.value.length !== 13) {
        this.info.phoneNumber.error = true;
        error = true;
    } else {
        this.info.phoneNumber.error = false;
    }
    console.log(this.info);
    return !error;
}

setNotification(type, action, code) {
this.entry.clear();
const factory = this.resolver.resolveComponentFactory(NotificationComponent);
this.componentRef = this.entry.createComponent(factory);
this.componentRef.instance.action = action;
this.componentRef.instance.code = code;
    this.componentRef.instance.type = type;
    setTimeout(() => {
        this.componentRef.destroy();
    }, 5000);
}

showLoader(){
$('.button-save').hide();
$('.loader').show();
}

hideLoader(){
$('.loader').hide();
$('.button-save').show();
}
}

import { Component, ComponentFactoryResolver, OnInit, ViewChild, ViewContainerRef } from
'@angular/core';
declare function initialiseCard(): any;

import * as $ from 'jquery';

import { CustomerService } from "../_services/customer.service";
import { CardService } from "../_services/card.service";
import { environment } from "../environments/environment";
import { CookieService } from "ngx-cookie-service";
import { Title } from "@angular/platform-browser";
import { NotificationComponent } from "../layout/notification/notification.component";
@Component({
    selector: 'app-cus-card',
    templateUrl: './cus-card.component.html',
    styleUrls: ['./cus-card.component.less', './card.less']
})
export class CusCardComponent implements OnInit {
    typePage: string = 'customer/card';
    typeClient: string = 'customer';
    cardForm = {
        creditName: { value: "", error: false },
        cardNumber: { value: "", error:

```

```

        false}, expirationDate: {value: "",
        error: false}, cvv: {value: "", error:
        false}
    };
    creditCards = [];
    componentRef: any;
    @ViewChild('viewContainerRefCustomerCard', {read: ViewContainerRef, static: false}) entry:
    ViewContainerRef;
    constructor(
    private customerService: CustomerService,
    private cardService: CardService,
    private cookie: CookieService,
    private title: Title,
    private resolver: ComponentFactoryResolver
    ) {
    }
    ngOnInit() {
    if (!this.cookie.check('token')) {
        window.location.href = environment.siteUrl
    }
    this.title.setTitle("Мої картки");
    this.initialiseCard();
    this.getCreditCards();
    }
    initialiseCard() {
    $(document).ready(function () {
        initialiseCard();
    });
    }
    getCreditCards(){
    const promise = new Promise((resolve, reject) => {
        this.cardService.getCards().toPromise()
        .then((res: any) => {
            this.creditCards = res;
            resolve();
        },
        err => {
            // Error
            reject(err);
        }
        ).catch();
    });
    }
    getProcessCard():any{
    let cards = [];
    this.creditCards.forEach(function (card) {
        let datePasts = card.expirationDate.split('-');
        cards.push({
            owner: card.owner,
            cardNumber: card.cardNumber,
            expirationDate: datePasts[1] + '/' + datePasts[0].toString().substr(-2),
            reference: card.reference
        });
    });

```

```

});
return cards;
}
addCard() {
if (this.validation()) {
    this.showLoader();
    const promise = new Promise((resolve, reject) => {
        this.cardService.addCard(this.cardForm).toPromise()
            .then((res: any) => {
                this.creditCards.push(res);
                this.setNotification('success', 'addCard', '108');
                this.cleanCard();
                this.hideLoader();
                resolve();
            },
            err => {
                // Error
                this.setNotification('error', 'addCard', err['error'][0].code);
                this.hideLoader();
                reject(err);
            })
            .catch();
    });
}
}

cleanCard(){
    this.cardForm.creditName.value = "";
    this.cardForm.cardNumber.value = "";
    this.cardForm.expirationDate.value = "";
    this.cardForm.cvv.value = "";
    $('#name').click();
}

deleteCard(reference, index){
    const promise = new Promise((resolve, reject) => {
        this.cardService.deleteCard(reference).toPromise()
            .then((res: any) => {
                this.setNotification('success', 'deleteCard', '109');
                this.creditCards.splice(index, 1);
                resolve();
            },
            err => {
                // Error
                this.setNotification('error', 'deleteCard', err['error'][0].code);
                reject(err);
            })
            .catch();
    });
}

validation(): boolean {
    let error = false;
    if (this.cardForm.creditName.value.length < 8) {
        this.cardForm.creditName.error =
            true; error = true;
    } else {

```

```

        this.cardForm.creditName.error = false;
    }
    if (this.cardForm.cardNumber.value.replace(/\s/g, "").length !== 16) {
        this.cardForm.cardNumber.error =
            true; error = true;
    } else {
        this.cardForm.cardNumber.error = false;
    }
    if (this.cardForm.expirationDate.value.length !== 5) {
        this.cardForm.expirationDate.error =
            true; error = true;
    } else {
        let currentYear = new Date().getFullYear().toString().substr(-2);
        let currentMoth = new Date().getMonth().toString();
        let cardDate =
            this.cardForm.expirationDate.value.split('/'); if
            (cardDate[1] < currentYear) {
                this.cardForm.expirationDate.error = true;
                error = true;
            } else {
                if (cardDate[1] === currentYear && Number(currentMoth) > Number(cardDate[0])) {
                    this.cardForm.expirationDate.error =
                        true; error = true;
                } else {
                    this.cardForm.expirationDate.error = false;
                }
            }
    }
}
if (this.cardForm.cvv.value.length < 2) {
    this.cardForm.cvv.error =
        true; error = true;
} else {
    this.cardForm.cvv.error = false;
}
return !error;
}

showLoader(){
$('#add_card_button').hide();
$('.loader').show();
}
hideLoader(){
$('.loader').hide();
$('#add_card_button').show();
}
setNotification(type, action, code) {
this.entry.clear();
const factory = this.resolver.resolveComponentFactory(NotificationComponent);
this.componentRef = this.entry.createComponent(factory);
this.componentRef.instance.action = action;
this.componentRef.instance.code = code;
    this.componentRef.instance.type
        = type; setTimeout(() => {
            this.componentRef.destroy();
        }, 5000);
}

```

```

    }
  }
  import { Component, OnInit } from '@angular/core';
  import * as $ from 'jquery';
  import { Title } from "@angular/platform-browser";
  import { environment } from "../../environments/environment";
  import { CookieService } from "ngx-cookie-service";
  import { SupplierService } from
  "../../_services/supplier.service"; import
  { CountryService } from "../../_services/country.service";
  import { OrderService } from
  "../../_services/order.service"; import { TimeService }
  from "../../_services/time.service";
  @Component({
    selector: 'app-cus-order',
    templateUrl: './cus-order.component.html',
    styleUrls: ['./cus-order.component.less']
  })
  export class CusOrderComponent implements OnInit {
    typePage: string = 'customer/orders';
    typeClient: string = 'customer';
    arrayCountries = new Map<string,
    string>(); countries;
    currentDate: Date = new Date(Date.now());
    pageSizes = [5, 15, 25, 50];
    filters = {
  fromDate: {
    year:
      this.currentDate.getFullYear(),
    month:
      this.currentDate.getMonth(),
    day: this.currentDate.getDate()
  },
  toDate: {
    year:
      this.currentDate.getFullYear(),
    month: this.currentDate.getMonth()
      + 2, day: this.currentDate.getDate()
  },
  pageNumber: 0,
  pageSize: this.pageSizes[0]
  };
    allParking: [];
    orders;
    paginations: object[] = [];
    QRCodeString: string = null;
    constructor(
  private supplierService: SupplierService,
  private countryService: CountryService,
  private cookie: CookieService,
  private title: Title,
  private orderService: OrderService,
  private timeService: TimeService

```



```

    ) {
this.title.setTitle('Бронювання');
    }
    ngOnInit() {
this.orders = null;
    if (!this.cookie.check('token')) {
        window.location.href =
            environment.siteUrl;
    }
this.getCountry();
this.getOrders();
    }

getCountry() {
    const promise = new Promise((resolve, reject) => {
        this.countryService.getCountries().toPromise()
            .then((res: any) => {
                console.log(res);
                this.countries = res;
                this.countries.forEach(country => {
                    this.arrayCountries.set(country.code, country.name);
                });
                resolve();
            },
            err => {
                // Error
                reject(err);
            }
        );
    });
}

filterOrders() {
    this.orders = null;
    this.showLoader();
    this.getOrders();
    }

getOrders() {
    const promise = new Promise((resolve, reject) => {
        this.orderService.getCustomerOrders(this.filters).toPromise()
            .then((res: any) => {
                this.processOrders(res);
                this.createPagination();
                this.hideLoader();
                resolve();
            },
            err => {
                // Error
                reject(err);
            }
        );
    });
}

processOrders(orders) {
    console.log(orders);
}

```

```

if (orders.total > 0) {
  for (let i = 0; i < orders.orders.length; i++) {
    for (let placeNumber = 0; placeNumber < orders.orders[i].orderItems.length;
        placeNumber++) { orders.orders[i].orderItems[placeNumber].qrCodeString =
      orders.orders[i].reference + ";" + orders.orders[i].orderItems[placeNumber].placeNumber +
      ";" + orders.orders[i].orderItems[placeNumber].from + ";" +
      orders.orders[i].orderItems[placeNumber].to;
      orders.orders[i].orderItems[placeNumber].duration =
this.timeService.calculateTimeDuration(orders.orders[i].orderItems[placeNumber].from,
orders.orders[i].orderItems[placeNumber].to);
      if (orders.orders[i].orderItems[placeNumber].depart !== null) {
        orders.orders[i].orderItems[placeNumber].realDuration =
this.timeService.calculateTimeDuration(orders.orders[i].orderItems[placeNumber].from,
orders.orders[i].orderItems[placeNumber].depart);
        orders.orders[i].orderItems[placeNumber].depart =
this.timeService.dateToString(orders.orders[i].orderItems[placeNumber].depart);
      } else {
        orders.orders[i].orderItems[placeNumber].depart = 'Не виїхав';
        orders.orders[i].orderItems[placeNumber].realDuration = "";
      }
      orders.orders[i].orderItems[placeNumber].from =
this.timeService.dateToString(orders.orders[i].orderItems[placeNumber].from);
      orders.orders[i].orderItems[placeNumber].to =
this.timeService.dateToString(orders.orders[i].orderItems[placeNumber].to);
    }
  }
}
this.orders = orders;
}
generateQrCode(grCodeString) {
this.QRCodeString = grCodeString;
}
createPagination() {
this.paginations = [];
if (this.orders.total >
this.filters.pageSize) {
  console.log(this.orders.total);
  console.log(this.filters.pageSize);
  let pages = Math.ceil(this.orders.total / this.filters.pageSize);
  console.log(pages);
  let activePage = this.filters.pageNumber;
  if (activePage > 0) {
    this.paginations.push(
      h({ number:
        activePage - 1,
        name: 'Попередня',
        class: "
      }));
  }
  for (let i = 0; i < pages; i++) {
    this.paginations.push({
      number: i,
      name: i + 1,
      class: activePage == i ? 'active' : "

```

```

    });
  }
  if (activePage !== (pages -
  1)) {
    this.paginations.push({
      number: activePage + 1,
      name: 'Наступна',
      class: "
    });
  }
}

}

changePage(pageNumber) {
  this.filters.pageNumber = pageNumber;
  this.filterOrders();
  console.log(this.filters.pageNumber);
}

changeSize(size) {
  this.filters.pageNumber = 0;
  this.filters.pageSize = size;
  this.filterOrders();
}

showLoader() {
  console.log('loader show');
}

$(".loader").show();
}

hideLoader() {
}

$(".loader").hide();
}

}

import { Component, ComponentFactoryResolver, OnInit, ViewChild, ViewContainerRef } from
'@angular/core';
import { PenaltyService } from "../_services/penalty.service";
import { Title } from "@angular/platform-browser";
import { environment } from "../_environments/environment";
import * as $ from 'jquery';
import { CookieService } from "ngx-cookie-
service"; import { TimeService } from
"../_services/time.service"; import
{ CardService } from
"../_services/card.service";
import { NotificationComponent } from "../_layout/notification/notification.component";
@Component({
  selector: 'app-penalty',
  templateUrl: './penalty.component.html',
  styleUrls: ['./penalty.component.less']
})
export class PenaltyComponent implements OnInit {
  typePage: string = 'customer/penalty';
  typeClient: string = 'customer';
  penaltyStatuses: string[];
  penalties;

```

```

    totalPenalties = 0;
    creditCards;
    selectedPenalty = null;
    selectedPenaltyIndex =
    null;
    pageSizes = [5, 15, 25, 50];
    filters = {
        status: 'Не сплачено',
    };
    pageNumber: 0,
    pageSize: this.pageSizes[0]
    };
    paginations: object[] = [];
    componentRef: any;
    @ViewChild('viewContainerRefCustomerPenalty', {read: ViewContainerRef, static: false})
    entry:
    ViewContainerRef;
    constructor(
    private penaltyService: PenaltyService,
    private title: Title,
    private cookie: CookieService,
    private timeService: TimeService,
    private cardService: CardService,
    private resolver: ComponentFactoryResolver
    ) { }
    ngOnInit() {
        this.title.setTitle('Мої штрафи');
        if (!this.cookie.check('token')) {
            window.location.href = environment.siteUrl;
        }
        this.getPenaltyStatuses();
        this.filterPenalties();
        this.getCreditCards();
    }

    getCreditCards() {
        const promise = new Promise((resolve, reject) => {
            this.cardService.getCards().toPromise()
                .then((res: any) => {
                    // this.creditCards = res;
                    let cards = [];
                    res.forEach(function (card)
                    {
                        let datePasts = card.expirationDate.split('-');
                        console.log(datePasts);
                        cards.push({
                            owner: card.owner,
                            cardNumber: card.cardNumber,
                            expirationDate: datePasts[1] + '/' + datePasts[0].toString().substr(-2),
                            reference: card.reference
                        });
                    });
                    this.creditCards = cards;
                    resolve();
                },
                err => {

```

```

        // Error
        reject(err);
    }
    ).catch();
});
}

setSelectPenalty(penalty, index){
    this.selectedPenalty = penalty;
    this.selectedPenaltyIndex = index;
}

payPenalty(creditCard) {
    const promise = new Promise((resolve, reject) => {
        this.penaltyService.payPenalty(this.selectedPenalty.reference,
            creditCard.reference).toPromise()
            .then((res: any) => {
                this.setNotification('success','penaltyPay','110');
                this.penalties.splice(this.selectedPenaltyIndex, 1);
                resolve();
            },
            err => {
                // Error
                reject(err);
            }
        );
    });
}

private getPenaltyStatuses(){
    const promise = new Promise((resolve, reject) => {
        this.penaltyService.getPenaltyStatuses().toPromise()
            .then((res: any) => {
                console.log(res);
                this.penaltyStatuses = res;
                resolve();
            },
            err => {
                // Error
                reject(err);
            }
        );
    });
}

public filterPenalties(){
    this.penalties = null;
    this.totalPenalties = 0;
    this.showLoader();
    this.getPenalties();
}

private getPenalties(){
    const promise = new Promise((resolve, reject) => {
        this.penaltyService.getPenalties(this.filters).toPromise()
            .then((res: any) => {
                console.log(res);
                this.processPenalties(res.penalties);
            }
        );
    });
}

```

```

        this.totalPenalties = res.total;
        this.createPagnation();
        this.hideLoader();
        resolve();
    },
    err => {
        // Error
        reject(err);
    }
);
});
}
processPenalties(penalties)
{ console.log(penalties);
  if(penalties != null) {
    penalties.forEach(penalty
=> {
      console.log(penalty);
      penalty.duration = this.timeService.calculateTimeDuration(penalty.to,
      penalty.depart); penalty.to = this.timeService.dateToString(penalty.to);
      penalty.depart = this.timeService.dateToString(penalty.depart);
    });
  }
  this.penalties = penalties;
  console.log(this.penalties);
}
createPagnation() {
this.paginations = [];
if (this.totalPenalties > this.filters.pageSize) {
  console.log(this.totalPenalties);
  console.log(this.filters.pageSize);
  let pages = Math.ceil(this.totalPenalties /
  this.filters.pageSize); console.log(pages);
  let activePage = this.filters.pageNumber;
  if (activePage > 0) {
    this.paginations.push
    h({ number:
    activePage - 1,
    name: 'Попередня',
    class: "
    });
  }
  for (let i = 0; i < pages;
  i++) {
    this.paginations.push(
    { number: i,
    name: i + 1,
    class: activePage == i ? 'active' : "
    });
  }
  if (activePage != (pages -
  1)) {
    this.paginations.push({

```

```

        number: activePage + 1,
        name: 'Наступна',
        class: "
    });
    }
}

changePage(pageNumber) {
    this.filters.pageNumber = pageNumber;
    this.filterPenalties();
}

changeSize(size) {
    this.filters.pageNumber = 0;
    this.filters.pageSize = size;
    this.filterPenalties();
}

showLoader() {
    console.log('loader show');
    $(".loader").show();
}

hideLoader() {
    $(".loader").hide();
}

setNotification(type, action, code) {
    this.entry.clear();
    const factory = this.resolver.resolveComponentFactory(NotificationComponent);
    this.componentRef = this.entry.createComponent(factory);
    this.componentRef.instance.action = action;
    this.componentRef.instance.code = code;
    this.componentRef.instance.type = type;
    setTimeout(() => {
        this.componentRef.destroy();
    }, 5000);
}

import { Component, OnInit } from '@angular/core';
import { Title } from "@angular/platform-browser";
@Component({
    selector: 'app-success',
    templateUrl: './success.component.html',
    styleUrls: ['./success.component.less']
})
export class SuccessComponent implements OnInit {
    constructor( private
    title: Title
    ) { }
    ngOnInit() {
        this.title.setTitle("Успішне бронювання");
    }
}

```