

Міністерство освіти і науки України

Державний заклад
«Луганський національний університет імені Тараса Шевченка»

Навчально-науковий інститут математики та інформаційних технологій

Кафедра інформаційних технологій та систем

Зюбін Володимир Миколайович

**ВЕБ-ОРІЄНТОВАНА ІНФОРМАЦІЙНА СИСТЕМА
ОБСЛУГОВУВАННЯ АВТОПАРКІНГУ**

кваліфікаційна робота

здобувача вищої освіти другого (магістерського) рівня

освітньої програми «Мультимедійні системи»

за спеціальністю 121 Інженерія програмного забезпечення

Особистий підпис _____ Володимир ЗЮБІН

Науковий керівник _____ Світлана ДОНЧЕНКО,
асистент кафедри інформаційних
технологій та систем

Завідувач кафедри _____ Микола СЕМЕНОВ,
кандидат педагогічних наук, доцент
кафедри інформаційних технологій
та систем

Полтава – 2025

АНОТАЦІЯ

Зюбін В. М.

Тема: Веб-орієнтована інформаційна система обслуговування автопаркінгу.

Спеціальність: 121 «Інженерія програмного забезпечення».

Установа: ЛНУ імені Тараса Шевченка, 2025 р.

Магістерська робота містить: 80 с., 40 рис., 22 табл., 1 додат., 27 джерел.

Об'єкт дослідження – керування кабінетом власника майданчиків автомобільних паркувань.

Предмет дослідження – спрощення керування кабінетом власника майданчиків автомобільних паркувань.

Мета роботи – реалізація системи керування кабінетом власника майданчиків атомобільних паркувань з дотриманням стандартів захисту та обробки персональних даних.

Результати роботи. Була розроблена система з керування кабінетом власника майданчиків автомобільних паркувань. Для цього було розроблено власний кабінет з можливістю його ведення, а саме: редагування персональної інформації, можливість додавати, редагувати і видаляти власні парковки, переглядом інформації про бронювання і динаміку зміни вартості оренди паркомісць.

Приділено увагу захисту персональних даних. Для цього було проаналізовано алгоритми шифрування і вибрано AES. Даний вибір обґрунтовано і наведено детальний опис його роботи і рівень захищеності даних.

Розроблено вед-додаток керування кабінетом власника майданчиків автомобільних паркувань. Серверна частина реалізована на мові програмування Java 8, а клієнтська – TypeScript, фреймворк Angular 8. Для збереження даних використовується реляційна база даних PostgreSQL.

Ключові слова: ШИФРУВАННЯ, AES, GDPR, СИСТЕМА, ПАРКОВКА, БРОНЮВАННЯ.

ABSTRACT

Ziubin Volodymyr

Theme: Web-based parking information system.

Speciality: 121 Software Engineering

Institution: Luhansk Taras Shevchenko National University (LTSNU), 2025.

The masters work contains: 80 pages, 40 Fig., 22 Table, 1 adj., 27 source.

A research object is management of the office of the owner of car parking lots.

The article of research is simplifying the management of the parking lot owner's office.

An aim of work is implementation of a management system for the owner of a car parking lot in compliance with personal data protection and processing standards.

Job performances. Was developed a system for managing the cabinet of the owner of cars parking. To do this, we developed our cabinet with the ability to maintain it, namely: editing personal information, the ability to add, edit and delete their parking, viewing information about reservations, and the dynamics of changes in the cost of renting parking spaces.

Attention was paid to the protection of personal data. To do this, we analyzed the encryption algorithms and chose AES. This choice is justified and provides a detailed description of its work and the level of data protection.

A management application for managing the office of the owner of car parking lots has been done. The server part was written with Java 8 programming language, and the client part was written with TypeScript, the Angular 8 framework. The Postgres relational database is used to store data.

Keywords: ENCRYPTION, AES, GDPR, SYSTEM, PARKING, BOOKING.

ЗМІСТ

| | |
|--|-----------|
| ВСТУП..... | 6 |
| РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОГО СЕРЕДОВИЩА ТА МАТЕМАТИЧНОГО ЗАБЕЗПЕЧЕННЯ | 9 |
| 1.1. Аналіз сучасних підходів до побудови інформаційних систем паркінгу | 9 |
| 1.2. Огляд аналогів..... | 12 |
| 1.3. Аналіз предметного середовища | 19 |
| 1.3.1. Опис процесу діяльності | 19 |
| 1.3.2. Опис функціональної моделі | 21 |
| 1.4. Дослідження математичного забезпечення..... | 22 |
| 1.4.1. Змістова постановка задачі | 22 |
| 1.4.2. Математична задача..... | 22 |
| 1.4.3. Обґрунтування методу розв'язання | 23 |
| 1.4.4. Опис методу розв'язання | 23 |
| Висновки до розділу | 27 |
| РОЗДІЛ 2. МОДЕЛЮВАННЯ ТА АНАЛІЗ ВЕБ ОРІЄНТОВАНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ КЕРУВАННЯ КАБІНЕТОМ ВЛАСНИКА МАЙДАНЧИКІВ АВТОМОБІЛЬНИХ ПАРКУВАНЬ | 28 |
| 2.1. Вимоги до технічного забезпечення | 28 |
| 2.1.1. Вимоги до серверної частини веб-додатку | 28 |
| 2.1.2. Вимоги до клієнтської частини веб-додатку..... | 28 |
| 2.1.3. Вимоги до серверу бази даних | 28 |
| 2.1.4. Вимоги до клієнта..... | 28 |
| 2.2. Архітектура програмного забезпечення | 29 |
| 2.2.1. Логічне представлення статичної моделі структури програмного забезпечення | 29 |
| 2.2.2. Діаграма розгортання | 31 |
| 2.2.3. Діаграма послідовності | 32 |
| 2.2.4. Діаграма станів..... | 33 |
| Висновки до розділу | 33 |

| | |
|---|-----------|
| РОЗДІЛ 3. РОЗРОБКА ВЕБ-ОРІЄНТОВАНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ З КЕРУВАННЯ КАБІНЕТОМ ВЛАСНИКА МАЙДАНЧИКІВ АВТОМОБІЛЬНИХ ПАРКУВАНЬ | 34 |
| 3.1. Обґрунтування вибору засобів розробки | 34 |
| 3.2. Специфікація функцій | 36 |
| 3.3. Розробка інтерфейсу | 45 |
| 3.4. Керівництво користувача | 47 |
| Висновки до розділу | 58 |
| ВИСНОВКИ | 59 |
| СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ | 61 |
| ДОДАТОК А..... | 64 |

ВСТУП

Парковка є невід'ємною частиною міської інфраструктури. Рациональне 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. Вебдодаток, інтегрований з такими системами, стане зручним рішенням для водіїв, забезпечуючи ефективний контроль оплати, бронювання і запобігання порушенням [5].

Процес введення парковки в експлуатацію – не такий вже і простий. Побудувати, чи облаштувати парковку, цього недостатньо. Її потрібно популяризувати і про неї мають знати водії. Проста реклама це питання частково вирішить, але як далі вести облік по парковці!? Дивитися по завантаженню – цей варіант підходить тільки для автомобілів, які наявні в

даний момент. Можливий варіант, що хтось хоче забронювати місце на завтра в певний проміжок часу. Для цього необхідно розробити функціонал для власників паркувальних майданчиків, де вони можуть керувати власними парковками. Водії будуть оформлювати бронювання паркомісць на вибраній парковці. Але це розробляється не в межах даної дипломної роботи. Але для того щоб водії вибрали парковку, необхідно її спочатку додати в систему, чим і буде займатися кабінет власника майданчиків автомобільних паркувань.

Даний дипломний проєкт націлений на побудову системи, яка буде забезпечувати власників паркувальних майданчиків, необхідним функціоналом для ведення їхнього парко-бізнесу.

Об'єкт дослідження – керування кабінетом власника майданчиків автомобільних паркувань.

Предмет дослідження – спрощення керування кабінетом власника майданчиків автомобільних паркувань.

Мета роботи – реалізація системи керування кабінетом власника майданчиків атомобільних паркувань з дотриманням стандартів захисту та обробки персональних даних.

Методи дослідження: методи аналізу існуючих рішень на ринку.

Цілями розробки даної системи є:

- спрощення ведення паркобізнесу в умовах діджиталізації за рахунок створення загального майданчику для паркувань з великим асортиментом парковок.

Для досягнення поставлених цілей необхідно розв'язати наступні задачі:

- ведення власного кабінету для власників парковок, де можливо буде здійснювати керування парковками;
- керування паркувальними майданчиками та наявною кількістю паркомісць;
- перегляд динаміки змін цін на паркувальних майданчиках і бронювань по парковках.

Практичною цінністю роботи є розробка веб-орієнтованого додатку для керування автомобільними майданчиками в реальному часі.

В першому розділі було проаналізовано предметне середовище і існуючі аналоги. Досліджено математичне забезпечення інформаційної системи.

В другому розділі виконано аналіз та моделювання веб-орієнтованої системи керування кабінетом власника майданчиків автомобільних паркувань і сформовано мінімальні вимоги для технічного забезпечення, де буде розгорнуто дану систему. Також було описано архітектуру застосунку.

У третьому розділі наведено засоби розробки і аргументи, чому саме вони були вибрані. Описано методи класів серверної і клієнтської частити застосунку, спроектовано веб-інтерфейс системи з описом наявного функціоналу і побудовано керівництво користувача.

РОЗДІЛ 1.

АНАЛІЗ ПРЕДМЕТНОГО СЕРЕДОВИЩА ТА МАТЕМАТИЧНОГО ЗАБЕЗПЕЧЕННЯ

1.1. Аналіз сучасних підходів до побудови інформаційних систем паркінгу

Автоматизовані парковки набувають дедалі більшої актуальності, оскільки кількість автомобілів у містах постійно зростає. Зазвичай такі парковки розташовуються поруч із великими торговими центрами, спортивними залами, багатофункціональними комплексами та іншими популярними об'єктами. Хоча багато водіїв звикли до звичайних паркувальних майданчиків, впровадження нових технологій сприяє зростанню популярності автоматизованих парковок.

Це пояснюється тим, що автоматизація паркувальних процесів забезпечує зручний і швидкий в'їзд і виїзд автомобілів, підвищуючи пропускну здатність і роблячи паркінг більш привабливим для користувачів. Водночас для керуючих компаній автоматизовані системи є економічно вигідними, оскільки знижують ризики зловживань із боку персоналу.

Сучасні автоматизовані парковки включають такі ключові компоненти: стійки для в'їзду та виїзду, автоматичні шлагбауми, інформаційні табло, елементи візуальної навігації, термінали для оплати, каси та системи відеоспостереження. Усі ці елементи інтегруються в єдину інформаційну систему, що забезпечує їх ефективну взаємодію та узгоджену роботу [6].

Автоматизовані парковки вирішують низку важливих завдань:

- автоматизація процесів паркування;
- здійснення оплати послуг у зручний спосіб;
- формування звітів про кількість автомобілів, тривалість перебування та здійснені оплати [6].

Основними перевагами автоматизаційних парковок є економія грошей, адже впровадження автоматизованого паркування в багатьох випадках дозволяє повністю позбутися витрат на операторів, які на не автоматизованих

паркінгах вручну управляють і регулюють процесами на паркування. Другою перевагою можна зазначити конкурентоспроможність, наявність можливості вільно запаркуватися в безпосередній близькості від мети подорожі стає значною перевагою. Особливо в той час року коли клімат не схильний до піших прогулянок. Ефективність паркування — нераціональне використання паркувального простору. Чи не автоматизовані парковки заповнюють таксиста, офісні клерки з найближчих бізнес-центрів, мешканці прилеглих будинків тощо. Це заважає заповненню паркінгу цільовими відвідувачами. Наступною перевагою є контроль. За різними даними на платних, але не автоматизованих парковках, за участю операторів у процесах оплати, власники паркінгів не отримують до 35% виручки. Останньою з переваг це безпека. У кризових умовах зазвичай зростає кількість злочинів. Важливим завданням з яким вам допоможе впоратися система - це запобігання угонам транспортних засобів [6].

Сучасні системи управління парковкою поділяються на дві основні групи: напівавтоматичні та автоматичні. Напівавтоматичні парковки бувають з однією стійкою або з двома. Напівавтоматична парковка з однією стійкою є найбільш бюджетним та поширеним варіантом. Вона передбачає постійну присутність персоналу (касира), який розміщується на місці виїзду з території парковки. Принцип роботи напівавтоматичної парковки з однією стійкою полягає в тому, що водій перед в'їзним шлагбаумом отримує в'їзний квиток зі штрих-кодом, стріла шлагбаума піднімається і система фіксує проїзд автомобіля. При виїзді водій віддає талон касиру, який за допомогою сканера штрих-кодів зчитує інформацію з талона після чого система автоматично розраховує вартість перебування даного автомобіля на парковці. Після отримання оплати, касир відкриває шлагбаум для виїзду[6].

Варіант з двома стійками використовується тоді, коли місце розташування касиру знаходиться не на виїзді із зони паркінгу. На відміну від попереднього варіанту після оплатити послуги паркінгу, водію надається виїзний талон, який прикладається до зчитувача на стійці при виїзному

шлагбаумі. Автоматична парковка є повністю автоматизованим видом паркінгу. В'їзд, оплата та виїзд із зони паркінгу при такому варіанті здійснюється водіями самостійно без допомоги обслуговуючого персоналу. В системі автоматичної парковки використовуються в'їзна та виїзна стійки, спеціальний термінал самообслуговування, який надає можливість оплатити послуги паркінгу самостійно. Така система в автоматичному режимі фіксує усі в'їзди, виїзди, а також формує фінансову звітність [7].

В інформаційних системах парковок можуть застосовуватися кілька видів ідентифікації. Найпростішим методом є ідентифікація за квитком зі штрих-кодом, який водій отримує при в'їзді. Штрих-код фіксується в базі даних. При виїзді квиток зчитується сканером штрих-кодів, на підставі чого розраховується сума для оплати. Наступним варіантом є використання безконтактних RFID карток стандарту Mifare. На відміну від попереднього при такому підході з'являється можливість впровадження різних дисконтних програм на території комплексу. Найбільш просунутим варіантом є варіант, при якому ідентифікація автомобіля здійснюється за його номерним знаком. Для реалізації такого варіанту в'їзна та виїзна стійки обладнуються відеокамерами. У випадку, коли номерний знак зафіксувати не вдалося, водію автоматично видається квиток з штрих-кодом [7].

Усі з розглянутих варіантів побудови систем керування паркінгом не дозволяють водію заздалегідь дізнатись чи є вільні місця на паркувальному майданчику. Це створює передумови для того, щоб автомобілісти витрачали час на пошук вільного місця для паркування, проїзжаючи зайві кілометри. Наслідком цього є додаткове навантаження на дорожню мережу міста, що негативно впливає на безпеку руху та швидкість просування транспортних потоків. Відповідно до цього розширення функціональних можливостей інформаційної системи обслуговування автопаркінгу, спрямоване на надання можливості водію заздалегідь визначати та бронювати місце для паркування є актуальним технічним завданням. Для його вирішення пропонується інтегрувати в інформаційну систему автопаркінгу веб-додаток. У результаті

водій отримує можливість переглянути наявність вільних місць за потрібною йому адресою, економлячи при цьому свій час та бензин. Якщо на парковці є вільне місце, водію надається можливість його забронювати на деякий час та згодом зайняти його [7].

Системи автоматичного паркування — це спеціальні конструкції, які дозволяють вертикально розміщувати автомобілі, максимально ефективно використовуючи простір. Завдяки унікальному дизайну ці системи автоматично переміщують транспортні засоби від місця в'їзду до паркувального місця без участі водія. Такі системи також відомі під іншими назвами, зокрема:

- роботизований гараж;
- система паркування автомобілів;
- автоматична парковка;
- автоматизована система зберігання та пошуку транспортних засобів;
- механічна стоянка [7].

Принцип роботи:

1. Подача автомобіля

Водій під'їжджає до входу системи, де всі пасажери виходять з автомобіля.

2. Автоматичне транспортування

Механізми системи переміщують автомобіль до доступного місця для паркування. Процес може бути повністю автоматизованим або виконуватися за участю супроводу.

Ця технологія є ідеальним рішенням для умов обмеженого простору, підвищуючи ефективність паркування та комфорт для користувачів.

1.2. Огляд аналогів

Одним з аналогів розглядуваної системи є автоматична система LOT PARKING. LOT PARKING є сучасним рішенням, призначеним для організації автоматизованого контролю в'їзду та виїзду транспортних засобів та

автоматизації розрахунків за користування паркувальними послугами.

Переваги автоматизованої системи паркування LOT PARKING:

- автоматизація системи оплати;
- фінансовий контроль та облік;
- моніторинг подій у режимі реального часу;
- безпека;
- зручність для клієнтів.

Автоматизована система паркування LOT PARKING вирішує такі завдання:

- автоматизація контролю в'їзду (виїзду) автомобілів на стоянку;
- автоматизація системи оплати на основі квитків зі штрих-кодом та безконтактних карт доступу;
- зручний контроль та аналіз фінансової діяльності паркувального комплексу;
- моніторинг подій системи LOT PARKING у режимі реального часу;
- комфортність користування послугами паркувального комплексу для клієнтів [8].

Система автоматизованого паркування LOT PARKING складається з в'їзної стійки, виїзної стійки, автомата оплати, сервера системи, автоматизованого робочого місця адміністратора, робочого місця касира та комплекту програмного забезпечення (рис. 1.1). За бажанням в комплект постачання можуть входити в'їзний та виїзний шлагбауми.



Рис. 1.1. Проектування парковки за системою LOT PARKING

При в'їзді на паркування автомобіль потрапляє в зону дії першої в'їзної індукційної петлі, і система дозволяє в'їзд автомобіля на паркування. Водій повинен натиснути кнопку в'їзної стійки. Після того як водій забирає картку/білет зі штрих-кодом у в'їзної стійки, відкривається шлагбаум. Автомобіль перетинає першу та другу петлі, і в момент залишення другої петлі система подає команду на закриття шлагбауму. На екрані стійки з'являються підказки для водія. Якщо у водія виникли питання, він може скористатися голосовим зв'язком із адміністратором системи [8]. Оплата послуг паркування. Для оплати послуг паркування покупець може користуватися автоматично оплати або звернутися до касира. Клієнт повинен надати картку/квиток зі штрих-кодом, отриману на в'їзді. Система визначає час в'їзду та розраховує вартість послуги залежно від чинних тарифів. Клієнт зобов'язаний сплатити вартість паркування в автоматі оплати або у касира. В системі реєструється інформація про оплату та клієнту надається час для того, щоб залишити паркову [8].

При виїзді автомобіль потрапляє до зони першої виїзної індукційної петлі та запускається процедура виїзду. При виїзді клієнт вставляє картку у виїзну стійку. Система перевіряє факт оплати за пред'явленим документом (карта/квиток зі штрих-кодом). У разі успішного підтвердження оплати

система відкриває виїзний шлагбаум. Клієнт отримує право виїзду і після залишення другої виїзної індукційної петлі система закриває шлагбаум. Система працює в режимі періодичного зв'язку з сервером, для нормального функціонування системи потрібен канал зв'язку між пристроями, що постійно працює [8].

Другим аналогом є автоматизація паркувальних місць на основі RFID. Ефективним рішенням для автоматизації паркування є RFID-система. Ринок не дуже багатий на подібні рішення, тому що знаходиться ще на стадії формування. Актуальність таких систем неухильно зростає, оскільки вони покликані вирішити ряд проблем, з якими стикаються сьогодні власники та користувачі паркувань [9].

Основні проблеми паркування:

- в'їзд/виїзд транспорту відбувається надто довго;
- відсутність контролю кількості вільних паркувальних місць;
- помилки та затримки через вплив людського фактора;
- відсутність точного обліку подій в'їзду та виїзду;
- RFID-обладнання на основі технології UHF дозволить оптимізувати контрольно-пропускні процеси [9].

Для цього застосовуються 4 основні компоненти автоматизації.

- UHF зчитувач дальньої дії;
- контролер системи контролю та управління доступом;
- UHF етикетці для ідентифікації автомобіля;
- програмне забезпечення [9].

Як працює система RFID на паркові? На автомобіль встановлюється RFID-мітка (наклейка під лобове скло або корпусна мітка під номерний знак; пластикова карта з чіпом дальньої дії) мітка потрапляє в поле видимості зчитувача (до 10 метрів) і зчитується, як результат унікальний код ідентифікатора потрапляє до контролера керованого спеціальним програмним забезпеченням (ПЗ). ПЗ дозволяє проводити адміністрування системи та отримувати аналітичні дані про пересування кожного автомобіля через КПП,

а також оперативно управляти правами доступу для кожного користувача системи та багато іншого [9]. Впровадження системи автоматизованого в'їзду на паркінг дозволить скоротити кількість персоналу, який обслуговує паркінг, наприклад охоронців, а також мінімізувати вплив людського фактора на роботу системи. Є можливість застосувати систему «Анти-дубль», яка запобігатиме доступу на паркування автомобілів з однаковими унікальними номерами міток. Повторний в'їзд автомобіля з цим унікальним номером мітки можливий лише після виїзду зчитування мітки на виїзному зчитувачі. Оскільки мітка нанесена на лобове скло має функцію саморуйнування, зафіксувати факт виїзду без фактичного проїзду на автомобілі неможливо [9].

Переваги RFID-технології на парковці

- швидкісне безконтактне розпізнавання автомобіля (навіть при використанні спеціальної пластикової картки можна просто пред'явити її прямо з салону авто);
- відстеження час в'їзду/виїзду автомобілів;
- заборона проїзду транспорту без дозволу доступу та високий рівень захисту від несанкціонованого доступу;
- простота використання, надійність та безпека;
- легка установка та експлуатація практично без залучення людського ресурсу [9].

Третім аналогом є автоматизація паркування на основі технології UHF. Організація автоматичного паркінгу є невід'ємною частиною сучасного житлового комплексу. Вимоги користувачів паркування прості: забезпечення безпеки автомобіля, зручний та швидкий проїзд для економії особистого часу, наявність вільних місць. Класична система в'їзду на паркування вимагає докласти певних зусиль, які можуть бути вкрай незручними. Наприклад, висовуватись з вікна або чекати реакції охоронця [10]. На зміну високої вартості радіобрелків, необхідності часткої заміни елементів живлення, низькому рівню безпеки карток приходить автоматизована система в'їзду на

основі технології ультра високочастотної ідентифікації (технології UHF) (рис 1.2).

З чого складається UHF-система:

- мітки ALN-9662, які є ідентифікатором автомобіля;
- зчитувачі [10].

Принцип дії системи полягає у встановленні спеціальної мітки, яка може бути інтегрована у різних формах: наклеєна під лобове скло, встановлена під номерний знак автомобіля або вбудована у звичайну пластикову картку.

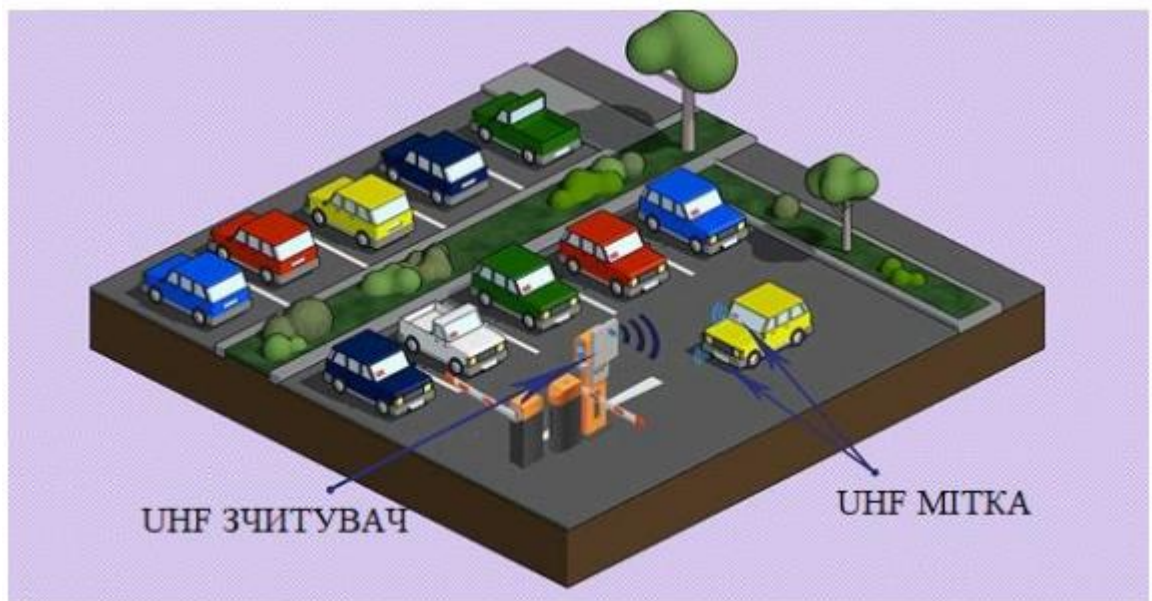


Рис. 1.2. Проекція парковки за системою на основі RFID

Установка мітки саме на автомобіль є найбільш раціональною, оскільки процес ідентифікації транспортного засобу у такому разі відбувається фактично без водія. Коли мітка потрапляє в поле видимості зчитувача, останній надсилає унікальний номер мітки для погодження на контролер, який керується спеціальним програмним забезпеченням. І вже контролер, проаналізувавши номер, приймає рішення про дозвіл чи заборону доступу [10]. Дальність зчитування міток при використанні цієї технології відбувається на відстані до 7 м (рис. 1.3).



Рис. 1.3. Мітка ALN-9662

У ПЗ можна відслідковувати час в'їзду/виїзду кожного автомобіля, призначати доступ для нього лише у певні зони паркінгу та багато інших функцій. Для випадків, коли у власника одного місця для паркування є кілька ідентифікаторів для доступу на це місце (наприклад, у вигляді пластикових карт), можна застосувати систему «Анти-дубль». Вона дозволить забороняти доступ на паркування міткам з однаковими номерами, поки цей номер не пройде через виїзний зчитувач. При цьому має бути зафіксовано реальний факт виїзду автомобіля через шлагбаум. Тобто пройти пішки з картою через виїзд і запустити ще одну машину за тим самим номером не вийде [10].

Перед початком розробки системи, необхідно дослідити існуючі аналоги. В процесі були знайдені наступні системи зі схожим функціоналом: **Parking UA** та **Privat24**.

Розглянемо кожну з них більш детально. **Parking UA** пропонує оптимальні варіанти паркування згідно локації, та прокладає найшвидший маршрут до них. Вибір парковок відбувається на інтерактивній карті з інформацією про них. Також можна керувати своїм паркуванням, тобто можна продовжити час у разі необхідності. Під час сканування QR-коду застосунок сам вибере на якій парковці ви перебуваєте і залишиться тільки вказати номер авто і метод оплати. Даний застосунок є дуже комфортним у використанні для

водіїв і має великий обсяг парковок, так як працює в межах цілої країни. Незважаючи на всі переваги, дана система має великий мінус, а саме відсутність можливості здавати в оренду власні паркомісця, що нас найбільше цікавить.

Privat24 – одна з найбільших систем мобільного банкінгу в Україні власником якої є державний банк ПриватБанк. Вже згаданий банк додав можливість купувати абонементи на парковки різних тарифів. Пошук парковок відбувається шляхом вказування на карті адреси, де необхідно залишити автомобіль, а система пропонує найближчі парковки. В даній системі, як і в більшості попередніх, немає можливості здавати в оренду власні парковки.

Таблиця 1.1

Порівняльна характеристика аналогів з наявною системою

| | Дана робота | ParkingUA | Privat24 |
|--|-------------|-----------|----------|
| Доступ до парковки по QR-коду | Так | Ні | Ні |
| Можливість здачі в оренду власних паркомісць | Так | Ні | Ні |

1.3. Аналіз предметного середовища

1.3.1. Опис процесу діяльності

В даній роботі розглядається автоматизація здачі паркомісць в оренду.

До автоматизації процесу здачі паркомісць в оренду власнику необхідно встановити шлагбаум і відповідний автомат для видачі і оплати талонів. Для контролю необхідно найняти людину, навіть дві, для по змінної роботи, а це все кошти. На рисунку 1.4 наведена діаграма діяльності процесу облаштування парковки до автоматизації.

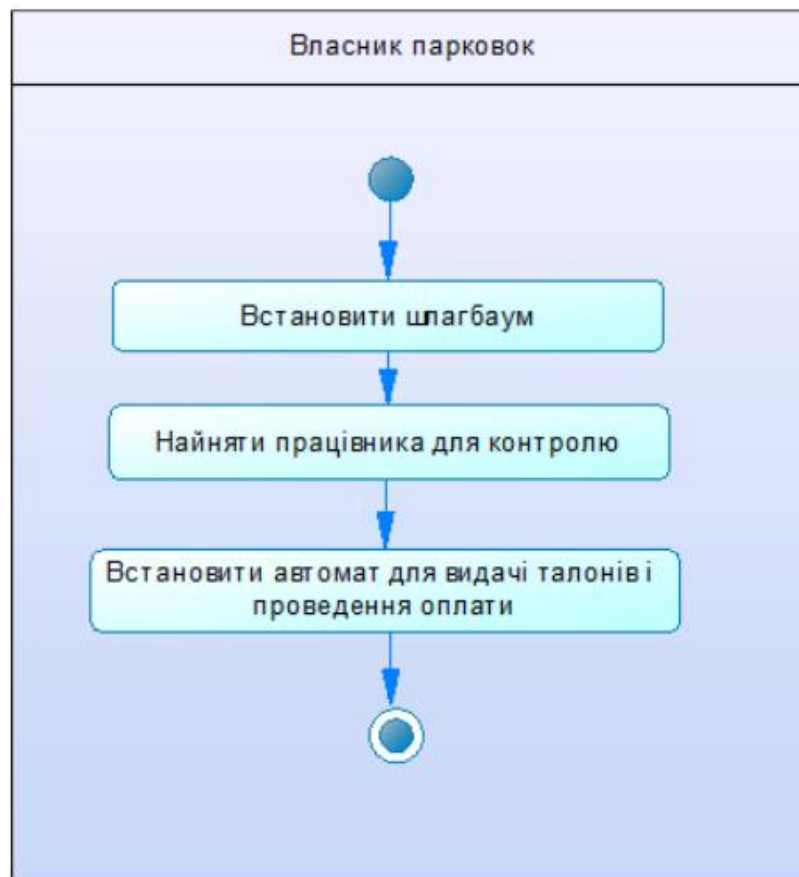


Рис. 1.4. Діаграма діяльності процесу облаштування парковки до оптимізації

Після оптимізації власник парковок має авторизуватися у власному кабінеті, якщо незареєстрований, то необхідно заповнити реєстраційну форму. Після авторизації він має завантажити документи, які засвідчують його право на ведення підприємницької діяльності. Наступним кроком буде додавання власних парковок в систему, які також потрібно підтверджувати відповідним документом. Після того, як парковки пройдуть підтвердження адміністратором системи, вони будуть доступні для водіїв, які зможуть на них бронювати паркомісця. Для фіксації заїзду і виїзду з парковки необхідно буде встановити шлагбаум зі сканером QR-коду. Таким чином, система буде працювати в автоматичному режимі, без залучення допоміжних працівників.

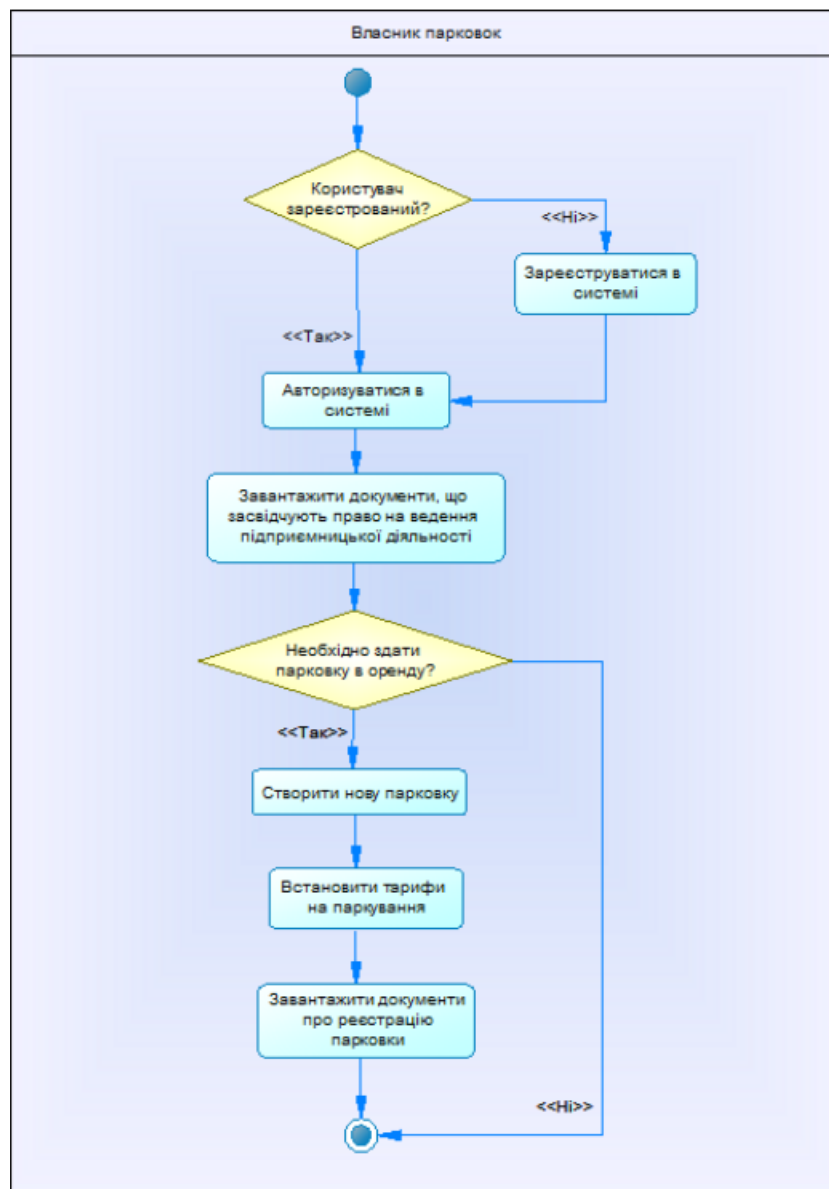


Рис 1.5. Діаграма діяльності процесу облаштування парковки після оптимізації

1.3.2. Опис функціональної моделі

Перед тим як починати проектувати діаграми використання необхідно визначити перелік акторів, що у свою чергу дозволить визначити перелік дій, які вони будуть виконувати. Нижче наведено список акторів з їх описом:

- **Адміністратор.** Співробітник компанії, який виконує перевірку документів власників парковок та перевірку технічного забезпечення парковок, котрі здаються в оренду
- **Власник парковок.** Зареєстрований користувач, котрий має на меті здачу в оренду власних паркомісць за допомогою даної системи

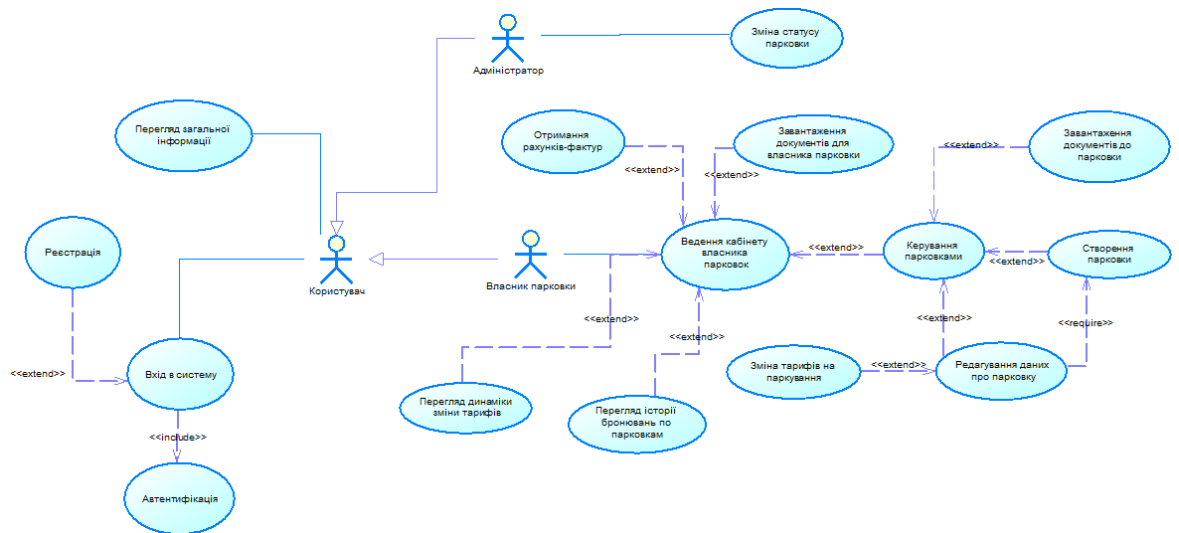


Рис. 1.6. Діаграма варіантів використання

1.4. Дослідження математичного забезпечення

1.4.1. Змістова постановка задачі

Захист персональних даних є одним із важливих моментів коли зберігаємо у системі дані про власників парковок і клієнтів. Відповідно до Закону України «Про захист персональних даних» власники системи відповідають перед законом про цілісність і конфіденційність інформації. Для цього існує стандарт GDPR (General Data Protection Regulation) – нормативно-правовий акт Європейського Союзу, призначений для захисту персональних даних. Недотримання стандарту GDPR може призвести до витоку інформації та втрати коштів клієнтів, за що власники системи мають відповідальність перед законом.

1.4.2. Математична задача

Задача шифрування даних за допомогою алгоритму AES.

Призначенням цієї задачі є дотримання вимог GDPR. Ці вимоги встановлюють правила для захисту персональних даних користувачів, котрі забезпечують їх безпеку. На вхід нам подається рядок даних, котрі мають бути зашифрованими.

Ціллю задачі є збереження конфіденційної інформації клієнта в зашифрованому вигляду.

1.4.3 Обґрунтування методу розв'язання

Для розв'язку задачі буде використовуватись алгоритм шифрування AES, адже він вже досить довго є стандартом в галузі криптографії. Звісно, існують інші алгоритми шифрування і їхньої стійкості достатньо в більшості випадків, але AES можна назвати кращим серед них. Дешифрувати дані, які зашифровані за допомогою алгоритму AES, неможливо за прийнятний час. Ми будемо використовувати ключ довжиною 256 біт, що забезпечить більшу стійкість алгоритму.

1.4.4. Опис методу розв'язання

Задача шифрування даних за допомогою алгоритму AES

Advanced Encryption Standard (AES) симетричний алгоритм шифрування (розмір блока 128 біт, ключ 128/192/256 біт) прийнятий в якості стандарту шифрування правлінням США.

Надійність роботи даного алгоритму залежить від розмірності ключа. Він працює з ключами довжиною в 128/192/256 біт. В залежності від його довжини, версії шифрування позначають AES-128, AES-192, або AES-256. Також від довжини ключа залежить кількість раундів шифрування: для 128 – це 10 раундів, а для 192 і 256 необхідно 12 і 14 раундів відповідно. На рисунку 1.4 наведено залежність часу, необхідного для дешифрування тексту, від довжини тексту.

| Довжина ключа | Час взлому |
|---------------|------------------------------|
| 56 біт | 399 секунд |
| 128 біт | 1.02×10^{18} років |
| 192 біта | 1.872×10^{37} років |
| 256 біт | 3.31×10^{56} років |

Рис. 1.7. Залежність часу взлому від розміру ключа

Можна зробити висновок, що розмірності ключа в 128 біт достатньо для безпечного зберігання інформації.

Розглянемо принцип роботи алгоритму AES з ключем довжиною 128 біт. Ключ розділяється на 16 частин k_0, k_1, \dots, k_{15} і записується в стовпчики матриці

InputKey. Кожен стовпець створює ключ шифра – це 4 слова w_0, w_1, w_2, w_3 , де $w_0 = k_0 k_1 k_2 k_3$, $w_1 = k_4 k_5 k_6 k_7$ і т.д. Розбиття ключа показано на рисунку 1.8.

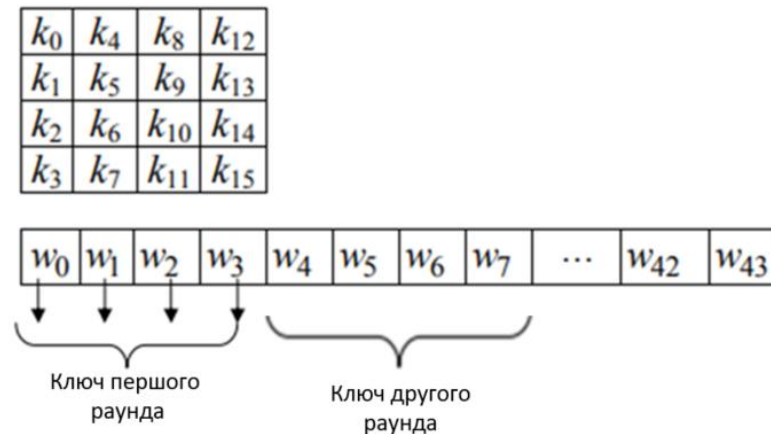


Рис. 1.8. Розбиття ключа

Кількість раундів роботи алгоритму залежить від довжини ключа. Для ключа довжиною 128 біт, кількість раундів становить 10. На рисунку 1.9 зображено схему перетворення даних в 10 раундів.

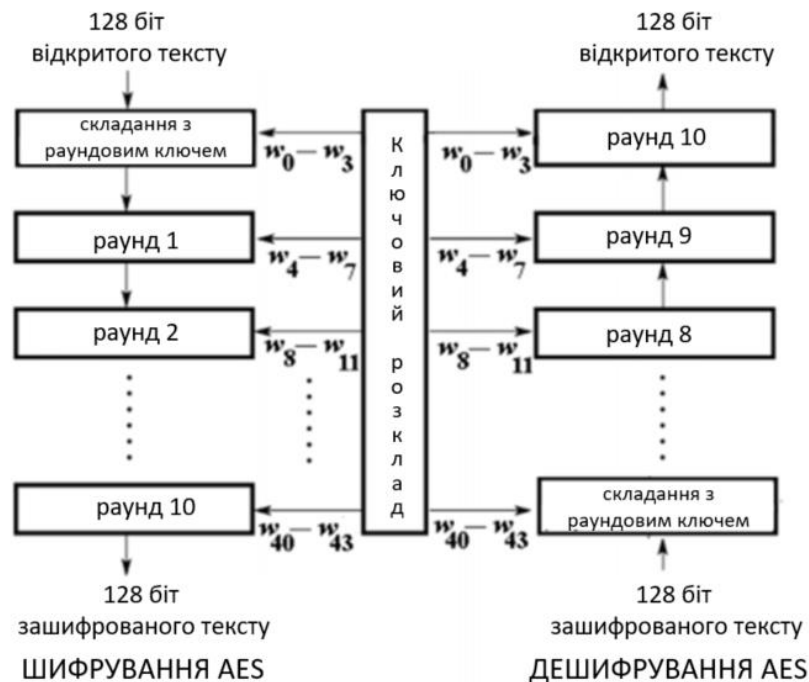


Рис. 1.9. Схема перетворення даних алгоритму AES

Опишемо схему роботи кожного раунду. Перед першим раундом виконується операція *AddRoundKey* (сумування по модулю 2 з початковим ключом шифру). Раунд складається з 4 перетворень:

- *SubBytes* – побайтова підстановка в S-боксі з фіксованою таблицею замін. Графічне зображення цього перетворення наведено на рисунку 1.10.

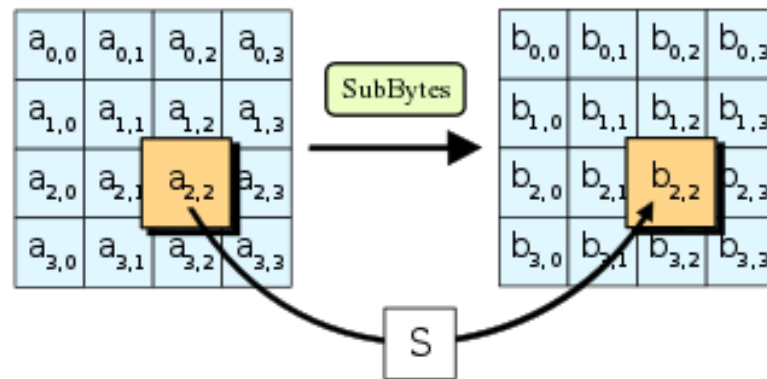


Рис. 1.10. Процедура *SubBytes*

- *ShiftRows* – побайтовий зсув рядків матриці *State* на різну кількість байт, де *State* – матриця, що описує дані на вході та виході раунду. На рисунку 1.11 показано дане перетворення.

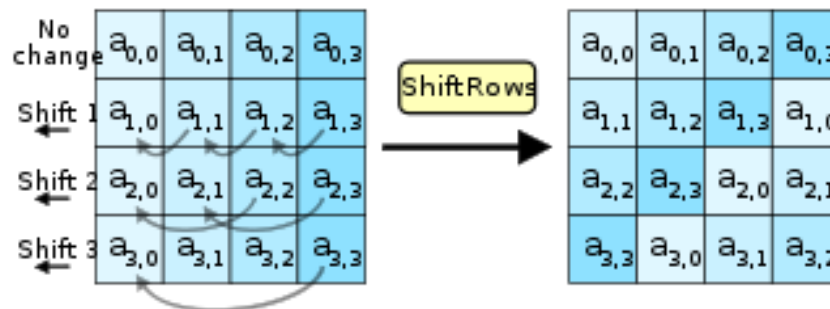


Рис. 1.11. Процедура *ShiftRows*

- *MixColumns* – перемішування байт в стовпцях. На рисунку 1.12 показано схематичне зображення цього перетворення.

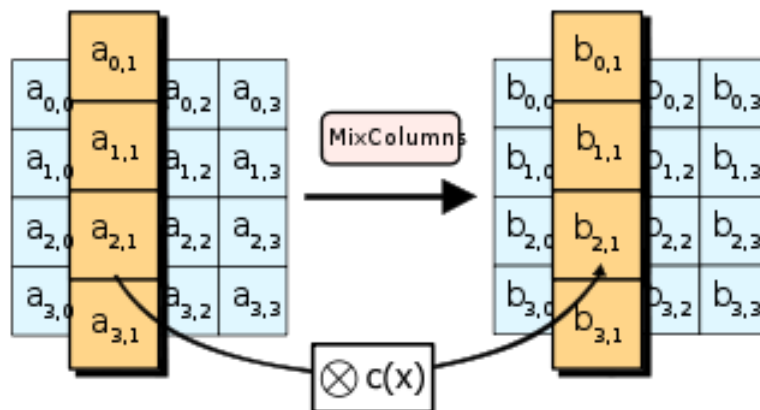


Рис. 1.12. Процедура *MixColumns*

- *AddRoundKey* – складання з раундовим ключем (операція XOR).
Графічне представлення даного перетворення показано на рисунку 1.13.

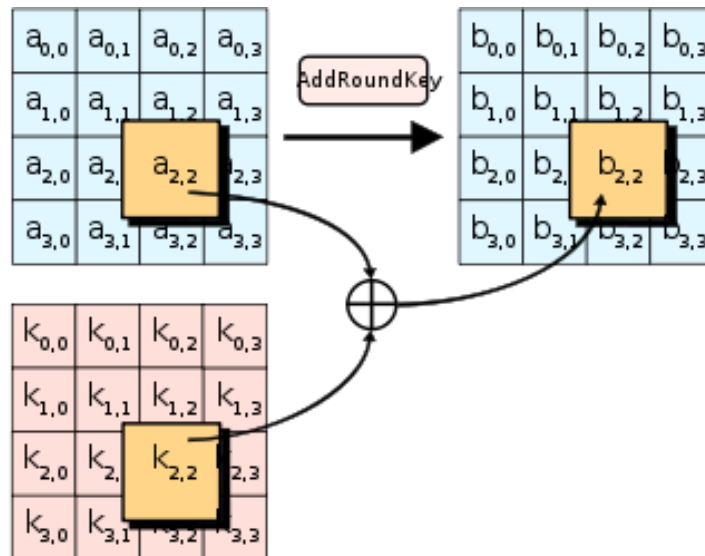


Рис. 1.13. Процедура *AddRoundKey*

Варто згадати, що останній раунд відрізняється від попередніх, адже в ньому не виконується перетворення *MixColumns*.

Для того щоб підібрати ключ довжиною 128 байт для алгоритму необхідно виконати 2^{127} операцій, а для 192 і 256 байт, необхідно 2^{191} і 2^{255} операцій відповідно. Кількість операцій дуже велика, відповідно ніхто не було пробувати знайти ключ методом перебору. Алгоритм AES стійких до наступних видів атак:

- диференціальний криптоаналіз;
- лінійний криптоаналіз;
- криптоаналіз на основі зв'язаних ключів.

Єдиним можливим способом злому шифру є атаки по побічним каналам. Але дані атаки не пов'язані з математичною частиною алгоритму, а більше залежать від особливостей реалізації системи, яка використовує його.

Даний алгоритм шифрування має велику швидкість роботи, але для нас даний показник не є основним, на першому місці безпека.

Висновки до розділу

В даному розділі було визначено змістовну постановку задачі для шифрування даних. Відповідно до неї було визначено сформульовано математичну постановку задачі, яку необхідно дослідити і розв'язати в межах даної системи. Було обґрунтовано чому взято саме симетричний алгоритм AES, а не інший. Також наведено його детальний опис роботи по кроках і вказані розміри секретних ключів і їх надійність. Також вказано переваги і недоліки, які в рамках даної системи є несуттєвими.

РОЗДІЛ 2.

МОДЕЛЮВАННЯ ТА АНАЛІЗ ВЕБ ОРІЄНТОВАНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ КЕРУВАННЯ КАБІНЕТОМ ВЛАСНИКА МАЙДАНЧИКІВ АВТОМОБІЛЬНИХ ПАРКУВАНЬ

2.1. Вимоги до технічного забезпечення

2.1.1. Вимоги до серверної частини веб-додатку

Сервер на якому буде розгорнуто серверну частину, має мати наступні мінімальні технічні характеристики:

- процесор з частотою 1 ГГц;
- RAM об'ємом 1 ГБ;
- 1 ГБ доступного місця на SSD.

2.1.2. Вимоги до клієнтської частини веб-додатку

Сервер на якому буде розгорнуто клієнтську частину, має мати наступні мінімальні технічні характеристики:

- процесор з частотою 1 ГГц;
- RAM об'ємом 0,5 ГБ;
- 1 ГБ доступного місця на SSD.

2.1.3. Вимоги до серверу бази даних

Сервер на якому буде розгорнуто базу даних, має мати наступні мінімальні технічні характеристики:

- процесор з частотою 1 ГГц;
- RAM об'ємом 0,5 ГБ;
- 3 ГБ доступного місця на SSD.

2.1.4. Вимоги до клієнта

У клієнта має бути встановлений веб-браузер зі списку відповідної версії, чи новішої:

- Opera 14+
- Google Chrome 23+;
- Firefox 21+;

– Safari 6+;

2.2. Архітектура програмного забезпечення

В основу програмного продукту закладено архітектурний шаблон MVC, який розшифровується, як Model-View-Controller. Цей шаблон поділяє систему на три взаємопов'язані частини: модель даних, вигляд який виступає інтерфейсом користувача, та модуль керування. На рисунку 2.1 показано взаємодію всіх трьох компонентів архітектурного шаблону.

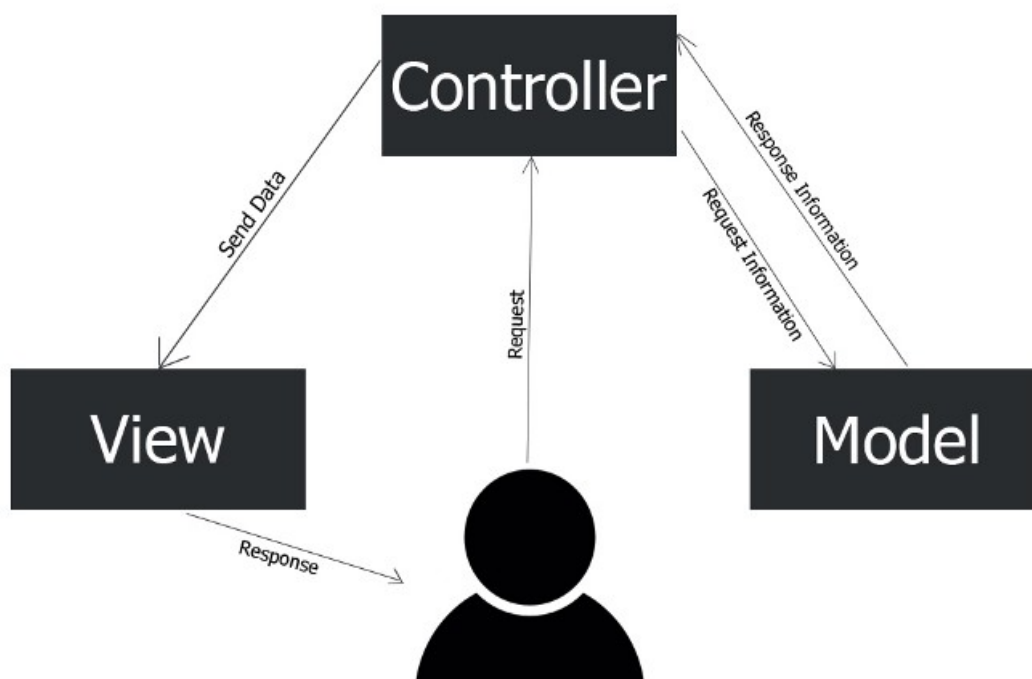


Рис. 2.1. Схема взаємодії частин MVC

Даний архітектурний шаблон значно полегшує програмування і майбутню підтримку коду, адже модель інкапсулює ядро даних і основний функціонал їхньої обробки і не залежить від процесу вводу чи виводу даних.

2.2.1. Логічне представлення статичної моделі структури програмного забезпечення

У мові UML для статичного представлення моделей систем використовуються діаграми класів та пакетів. Вона відображає залежності між пакетами, з яких і складається модель. На рисунку 2.2 представлена діаграма пакетів для серверної частини частини застосунку. Пакети особливо корисні при тестуванні. Кожен пакет при тестуванні може містити один або кілька тестових класів, за допомогою яких перевіряється поведінка пакета.

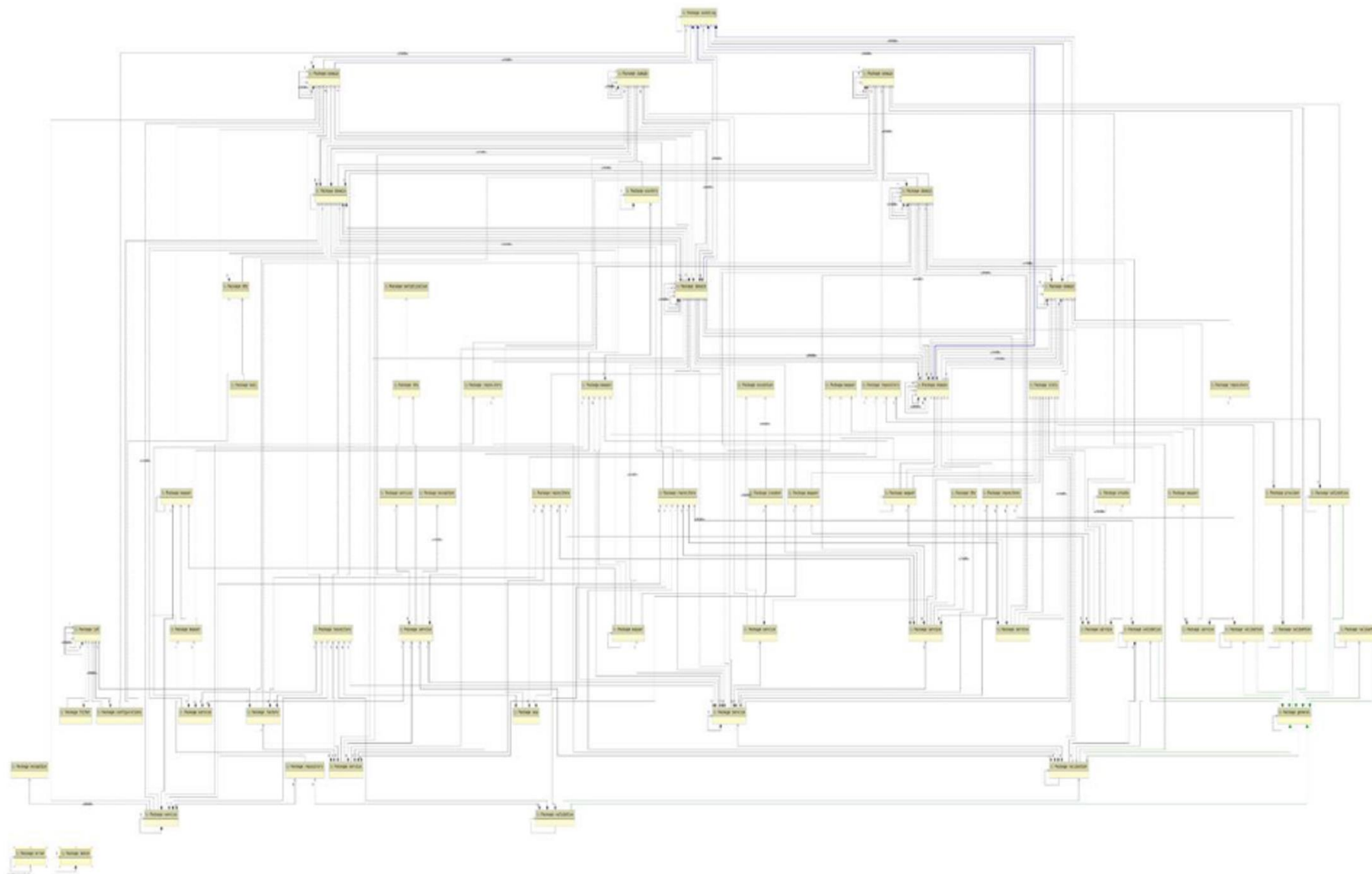


Рис. 2.2 Діаграма пакетів для серверної частини частини застосунку

2.2.2. Діаграма розгортання

Діаграма розгортання – це UML діаграма, яка відображає обчислювальні вузли, а також об’єкти, які виконуються на них. Даний тип діаграми дозволяє побачити ту частину системи, яку не видно під час її роботи. Кожна з вказаних компонент відповідає робочим екземплярам одиниць коду. Діаграма показує інфраструктуру на якій працює програмний продукт. На рисунку 2.3 предасталена діаграма розгортання системи.

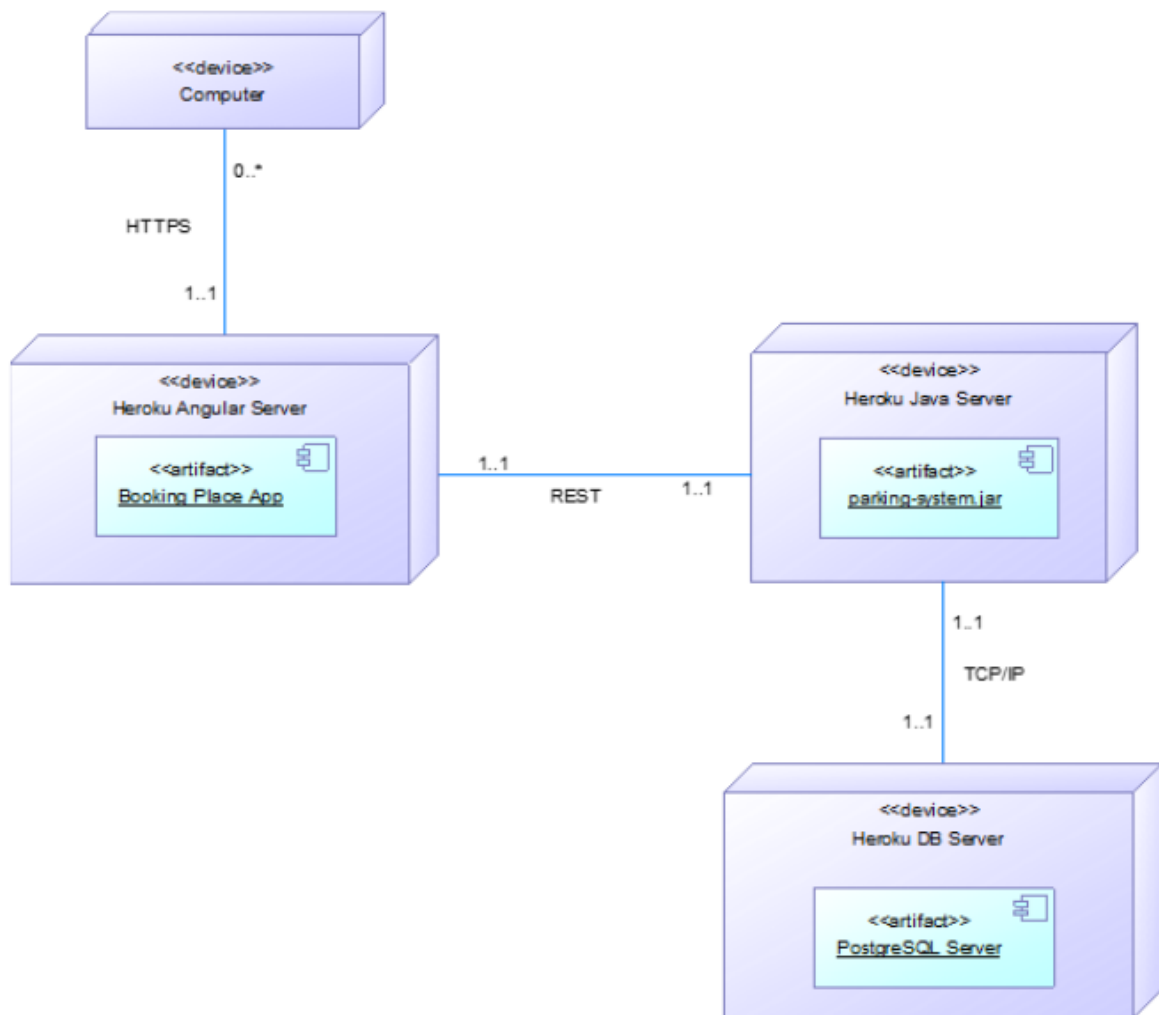


Рис. 2.3. Діаграма розгортання системи керування кабінетом власника майданчиків автомобільних паркувань

На діаграмі видно, що система розділена на три окремі компоненти, які розміщені на сервері «Heroku». Також показано, як вони між собою взаємодіють: клієнтська частина з серверною взаємодіють через REST (Representational state transfer), а серверна – з базою даних, через протокол TCP/IP, адже розміщені на одному сервері. В даній системі може одночасно працювати декілька девайсів

(телефон, комп'ютер, планшет) і всі вони будуть взаємодіяти з клієтською частиною, як це показано на даній діаграмі.

2.2.3. Діаграма послідовності

Діаграма послідовності – це UML діаграма, яка відображає взаємодію об'єктів системи впорядкованих за часом. На якій вертикальні лінії, це об'єкти системи, які взаємодіють між собою і зв'язок показаний у формі повідомлень між ними. На рисунку 2.4 предстала діаграма послідовності процесу додавання парковки.

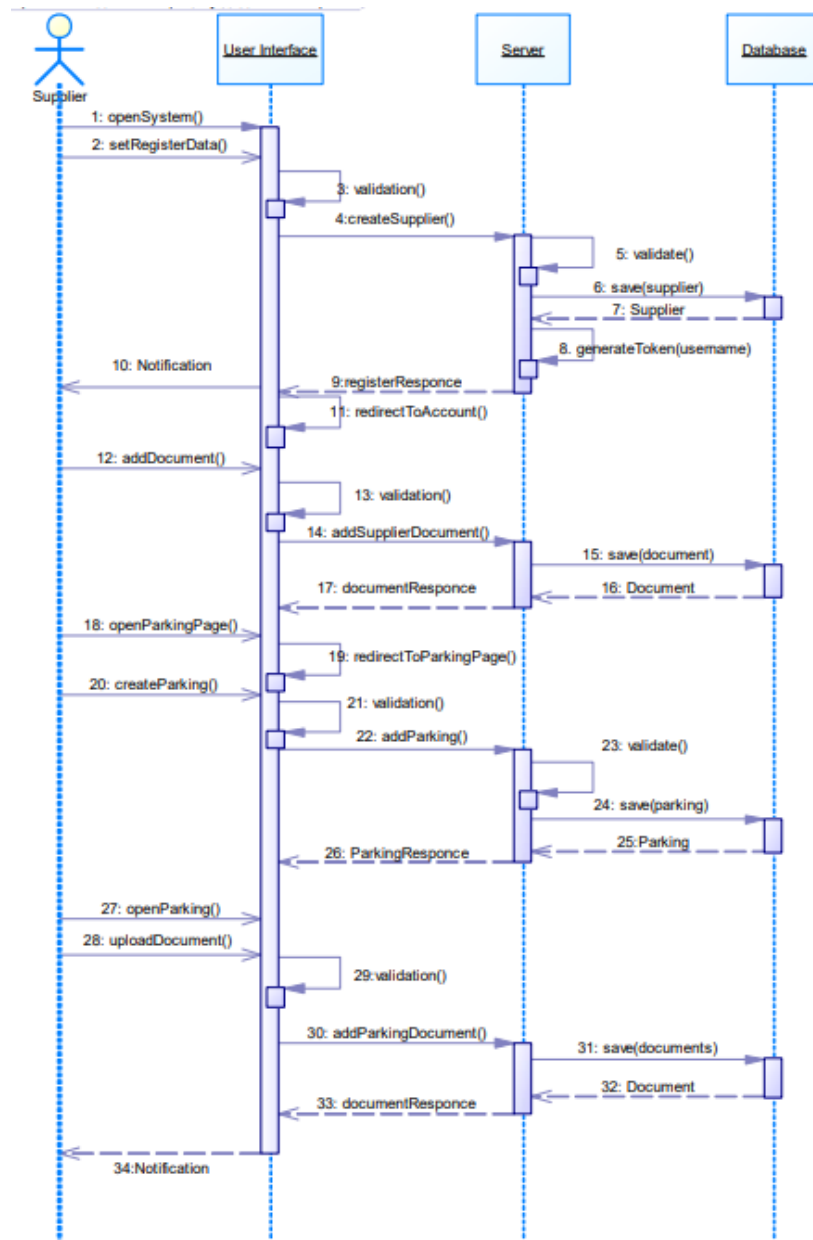


Рис. 2.4. Діаграма послідовності процесу додавання парковки

Було виділено чотири об'єкти системи – це власник парковок, користувацький інтерфейс(фронтальна частина системи), сервер і також база даних. На діаграмі чітко видно в якому порядку взаємодіють об'єкти, які виконують операції і що повертають, як результат.

2.2.4. Діаграма станів

Діаграма станів – це UML діаграма, яка показує зміну станів об'єктів системи в часі. Дана діаграма найкраще пояснює, як працює система, адже на ній показані ключові стани системи, які відбуваються для створення нової парковки, починаючи від входу на сайт і реєстрації власника парковок. Після успішної реєстрації і, за потреби, редагування персональної інформації, відображаються стани додавання парковки – це саме додавання парковки і додавання документу до парковки. На рисунку 2.5 представлена діаграма станів створення нової парковки.



Рис. 2.5. Діаграма станів створення нової парковки

Висновки до розділу

В даному розділі була описана архітектура застосунку по керування кабінетом власника майданчиків автомобільних паркувань. Прописано мінімальні вимоги до технічного забезпечення для розгортання застосунку. Побудовано необхідні діаграми, серед яких: діаграми послідовності та станів з коротким описом. На двох діаграмах показаний процес створення нової парковки починаючи від реєстрації власника парковок в системі.

РОЗДІЛ 3.

РОЗРОБКА ВЕБ-ОРІЄНТОВАНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ З КЕРУВАННЯ КАБІНЕТОМ ВЛАСНИКА МАЙДАНЧИКІВ АВТОМОБІЛЬНИХ ПАРКУВАНЬ

3.1. Обґрунтування вибору засобів розробки

Для розробки застосунку використовувалося чимало різних технічних засобів, адже система поділена на серверну і клієнтську частини.

Серверна частина була написана на мові програмування Java і фреймворки Spring і Hibernate

Java – це скомпільована мова програмування, яка працює на більшості відомих операційних систем, включаючи Windows, macOS та Linux. Адаптація C / C ++, Java пропонує простий синтаксис кодування разом із вдосконаленим управлінням пам'яттю та збиранням сміття. Java належить і підтримується Oracle, що є великою перевагою перед іншими мовами програмування. Перевагами мови програмування Java є: кросплатформеність, масштабованість, велика кількість готових бібліотек. На даний момент дана мова програмування набирає неабиякої популярності, то був сенс вибрати саме її для розробки системи. Також у майбутньому не буде проблем з масштабуванням.

Spring Framework — це програмний каркас (фреймворк) з відкритим кодом та контейнери з підтримкою інверсії управління для платформи Java.

Основні особливості Spring Framework можуть бути використані будь-яким додатком Java, але є розширення для створення веб-додатків на платформі Java EE. Незважаючи на це, Spring Framework не нав'язує якоїсь конкретної моделі програмування, Spring Framework став популярним в спільноті Java як альтернатива, або навіть доповнення моделі Enterprise JavaBean (EJB). Цей фреймворк значно полегшить написання коду, адже не потрібно буде багато писати чистого коду на Java.

Hibernate — засіб відображення між об'єктами та реляційними структурами (object-relational mapping, ORM) для платформи Java. Hibernate є вільним програмним забезпеченням, яке поширюється на умовах GNU Lesser

General Public License. Hibernate надає легкий для використання каркас (фреймворк) для відображення між об'єктно-орієнтованою моделлю даних і традиційною реляційною базою даних, що значно нам полегшить взаємодію з базою даних.

Для зберігання даних використовується об'єктно-реляційна система керування базами даних (СКБД) PostgreSQL. PostgreSQL має відкритий код. Крім того, він має міцний характер з високою продуктивністю і багатозадачністю. Ця СКБД також містить безліч функцій, які роблять її одним із хороших варіантів вибору з усіх баз даних. PostgreSQL став кращим з кожним його випуском. Також вона є безкоштовною, що є великим плюсом її вибору для використання в проекті.

Клієнт застосунку написаний мовою програмування Typescript на фреймворку Angular 8.

Typescript – це мова, заснована на JavaScript. Ключова різниця між Javascript та TypeScript це те, що JavaScript є мовою скриптування на стороні клієнта, а TypeScript - об'єктно-орієнтована мова, складена. Об'єктно-орієнтована парадигма програмування орієнтована на абстрагування даних, а не на алгоритм обробки даних. Він заснований на двох основних поняттях; предмети та класи, що значно полегшує програмування, адже вже мова програмування стає типізованою.

Angular – один з найпопулярніших і сучасних фреймворків (бібліотек) JavaScript, що розвивається Google. Простими словами, це набір готових рішень, що дозволяють швидко створювати складні й динамічні інтерфейси на сайтах. Angular можна розглядати як більш сучасну і потужну альтернативу широко відомої JavaScript бібліотеці jQuery.

Для контролю версій використовується GIT. Він дозволяє зберігати декілька створених версій файлу, проекту чи його складових, а також бачити зміни, які були створені розробником під час роботи. Для зберігання проекту використовується середовище GitHub, яке дозволяє керувати доступом до репозиторіїв. Дане середовище полегшує програмування у команді і дає

можливість перегляти і зберігати всі зміни про проєкті, що є дуже важливим в плані безпеки продукту.

Проект розгорнуто на безкоштовному хостингу Heroku, який призначений для навчальних проєктів. Також можна з легкістю конфігурувати сервери, де розміщується застосунок.

Для розробки системи, а саме написати код було вирішено в IDE IntelliJ Idea, адже вона оптимізована під мову програмування Java, але також можна писати код і на Typescript, що є великим плюсом. Також з даної IDE можна підключитися до БД. Вся розробка відбувається в межах однієї IDE.

3.2. Специфікація функцій

В таблицях 3.1 – 3.12 наведено перелік методів серверної частини системи з їх описом.

Таблиця 3.1

Специфікація методів класу SupplierManagementController

| Сигнатура | Опис |
|---|--|
| getSupplier() | Отримання інформації про власника парковок |
| updateSupplierInfo(SupplierBase supplier) | Оновлення інформації про власника парковок |

Таблиця 3.2

Специфікація методів класу SupplierRepositoryImpl

| Сигнатура | Опис |
|---|--|
| existWithDocument(String documentReference, String email) | Перевірка на існування документа у власника парковок |

Таблиця 3.3

Специфікація методів класу **SupplierService**

| Сигнатура | Опис |
|--|--|
| <code>getSupplierWithAllInfo(String username)</code> | Отримання детальної інформації про власника парковок |
| <code>updateSupplierInfo(SupplierBase supplierBase)</code> | Оновлення інформації про власника парковок |

Таблиця 3.4

Специфікація методів класу **Supplier**

| Сигнатура | Опис |
|---------------------------------|--------------------------|
| <code>equals(Object obj)</code> | Порівняння двох об'єктів |
| <code>hashCode()</code> | Генерація Hash коду |

Таблиця 3.5

Специфікація методів класу **ParkingController**

| Сигнатура | Опис |
|---|---|
| <code>getSupplierParkings()</code> | Отримання парковок власника парковок |
| <code>getParkingByReference(String reference)</code> | Отримання парковки |
| <code>createParking(ParkingRequest parking)</code> | Створення парковки |
| <code>updateParking(String reference, ParkingRequest parking)</code> | Оновлення інформації про парковку |
| <code>deleteParking(String reference)</code> | Видалення парковки |
| <code>deleteParkingPlace(String placeReference)</code> | Видалення паркомісць на парковці |
| <code>getParkingsPriceHistory(String reference)</code> | Отримання історичних даних цін парковок |
| <code>getParkingOrders(String reference, Integer pageNumber, Integer pageSize, LocalDate fromDate, LocalDate toDate)</code> | Отримання замовлень для парковки |

Таблиця 3.6

Специфікація методів класу **ParkingProvider**

| Сигнатура | Опис |
|---|--------------------|
| getParkingByReference(String reference) | Отримання парковки |

Таблиця 3.7

Специфікація методів класу **ParkingService**

| Сигнатура | Опис |
|---|--|
| getSupplierParkings() | Отримання парковок власника парковок |
| getParkingInfoByReference(String reference) | Отримання інформації про парковку по її коду |
| getParkingPriceHistory(String reference) | Отримання історичних даних цін парковок |
| createParking(ParkingRequest parkingRequest) | Створення парковки |
| updateParking(String reference, ParkingRequest parkingRequest) | Оновлення інформації про парковку |
| addNewPlaces(Parking parking, List<String> places) | Додавання нових паркомісць до парковки |
| updatePriceHistory(Parking parking, double newPrice, double newPenalty) | Оновлення історії цін для парковки |
| deleteParkingByReference(String reference) | Видалення парковки |
| deleteParkingPlace(String parkingReference, String placeReference) | Видалення паркомісць для парковки |
| getOrderPage(String parkingReference, Integer pageNumber, Integer pageSize, LocalDate fromDate, LocalDate toDate) | Отримання певної сторінки із замовленнями |

Таблиця 3.8

Специфікація методів класу **DocumentController**

| Сигнатура | Опис |
|---|--|
| getParkingDocumentTypes() | Отримання типів документів для парковок |
| getSupplierDocumentTypes() | Отримання типів документів для власника парковок |
| uploadParkingDocument(String reference, MultipartFile parkingDocument, String documentTypeCode) | Завантаження документу для парковки |
| deleteParkingDocument(String documentReference, String parkingReference) | Видалення документу парковки |
| uploadSupplierDocument(MultipartFile supplierDocument, String documentTypeCode) | Завантаження документу для власника парковок |
| deleteSupplierDocument(String documentReference) | Видалення документу власника парковок |

Таблиця 3.9

Специфікація методів класу **FileManager**

| Сигнатура | Опис |
|--|------------------|
| saveFile(String path, MultipartFile multipartFile) | Збереження файлу |

Таблиця 3.10

Специфікація методів класу **DocumentService**

| Сигнатура | Опис |
|---|--|
| <code>getParkingDocumentTypes()</code> | Отримання типів документів для парковок |
| <code>getSupplierDocumentTypes()</code> | Отримання типів документів для власника парковок |
| <code>getDocumentTypesByPredicate(Predicate where)</code> | Отримання документів за певною умовою |
| <code>uploadParkingDocument(String parkingReference, MultipartFile parkingDocument, String documentTypeCode)</code> | Завантаження документу для парковки |
| <code>uploadSupplierDocument(MultipartFile supplierDocument, String documentTypeCode)</code> | Завантаження документу для власника парковок |
| <code>uploadDocument(Document document, MultipartFile multipartFile)</code> | Завантаження документу |
| <code>createDocument(String uniqueId, MultipartFile document, String documentTypeCode)</code> | Створення документу |
| <code>deleteDocumentByReference(String documentReference)</code> | Видалення документу |

Таблиця 3.11

Специфікація методів класу **AddressProvider**

| Сигнатура | Опис |
|------------------------------------|--|
| <code>AddressRepositoryImpl</code> | <code>isAddressLinkedToActiveParking(String country, String city, String street, String streetNumber)</code> |

Таблиця 3.12

Специфікація методів класу **AddressRepositoryImpl**

| Сигнатура | Опис |
|---|--|
| existWithDocument(String documentReference, String email) | Перевірка на існування документа у власника парковок |

В таблицях 3.13 – 3.21 наведено перелік методів клієнтської частини системи з їх описом.

Таблиця 3.13

Специфікація методів класу **CountryService**

| Сигнатура | Опис |
|----------------|------------------------|
| getCountries() | Отримання списку країн |

Таблиця 3.14

Специфікація методів класу **SupplierService**

| Сигнатура | Опис |
|---|--|
| getInfo() | Отримання інформації про власника парковок |
| updateInfo(data) | Оновлення інформації про власника парковок |
| updateParking(data) | Оновлення інформації про парковку |
| getAllParkings() | Отримання короткої інформації про всі парковки |
| getParking(reference) | Отримання детальної інформації про парковку |
| deleteParking(reference) | Видалення парковки з системи |
| deletePlace(parkingReference, placeReference) | Видалення паркомісця з парковки |
| getInfoPrices(reference) | Отримання історичних даних про зміну цін оренди парковок |

Таблиця 3.15

Специфікація методів класу **DocumentService**

| Сигнатура | Опис |
|---|--|
| <code>getSupplierDocumentTypes()</code> | Отримання типів документів для підтвердження акаунта власника парковок |
| <code>getParkingDocumentTypes()</code> | Отримання типів документів для підтвердження парковки |
| <code>addParkingDocument(formData, reference)</code> | Додавання до парковки документу |
| <code>addSupplierDocument(formData)</code> | Додавання документа до акаунта власника парковок |
| <code>deleteSupplierDocument(reference)</code> | Видалення документа для підтвердження акаунта власника парковок |
| <code>deleteParkingDocument(parkingReference, documentReference)</code> | Видалення документа для парковки |

Таблиця 3.16

Специфікація методів класу **OrderService**

| Сигнатура | Опис |
|---------------------------------|----------------------------|
| <code>getOrders(filters)</code> | Отримання списку бронювань |

Таблиця 3.17

Специфікація методів класу **RequestService**

| Сигнатура | Опис |
|--|---------------------------------|
| <code>post(url, body)</code> | Запит POST |
| <code>postFileUpload(url, body)</code> | Запит POST для надсилення файлу |
| <code>get(url, isheaders = false, query = null)</code> | Запит GET |
| <code>put(url, body)</code> | Запит PUT |
| <code>delete(url)</code> | Запит DELETE |

Специфікація методів класу **TimeService**

| Сигнатура | Опис |
|---------------------------------|-------------------------|
| calculateTimeDuration(from, to) | Обчислення різниці часу |
| dateToString(date) | Форматування дати |

Специфікація методів класу **AccountComponent**

| Сигнатура | Опис |
|-------------------------------------|--|
| ngOnInit() | Ініціалізації даних сторінки власника парковок |
| getInfo() | Отримання інформації про власника парковок |
| getDocumentTypes() | Отримання типів документів |
| setDocumentType(documentTypes) | Задання типу документа до завантаженого файлу |
| setStatusTypeDocument(code, status) | Зміна статусу типу документа |
| getCountry() | Отримання списку країн |
| selectFile(event) | Вибір файлу для завантаження |
| addDocument() | Збереження документа |
| deleteDocument(reference, index) | Видалення документа |
| updateInfo() | Оновлення інформації про власника парковок |
| editable() | Зміна статуту для редагування даних |
| validation() | Перевірка введених даних на коректність |
| startLoader() | Запуск анімації на час завантаження даних |
| endLoader() | Зупинка анімації |

Таблиця 3.20

Специфікація методів класу **MyParkingComponent**

| Сигнатура | Опис |
|--|---|
| getCountry() | Отримання списку країн |
| getAllParkings() | Отримання короткої інформації про всі парковки власника парковок |
| getDocumentTypes() | Отримання типів документів для парковок |
| getParking(reference) | Отримання детальної інформації про парковку |
| checkChangesForm() | Перевірка даних на наявність змін |
| addPlace() | Додавання паркомісця до парковки |
| selectFile(event) | Вибір файлу для завантаження |
| addDocument(reference) | Збереження документа до парковки |
| deleteDocument(parkingReference, documentReference, index) | Видалення документа для парковки |
| updateParkingInfo() | Оновлення інформації про парковку |
| deleteParking(reference) | Видалення парковки |
| deletePlace(parkingReference, placeReference) | Видалення паркомісця для парковки |
| deletePlaceFront(placeRef) | Видалення паркомісця для парковки з списку на клієнтській частині |
| validation() | Перевірка введених даних |
| clearParkingForm() | Очищення форми для парковки |

Таблиця 3.21

Специфікація методів класу **OrderComponent**

| Сигнатура | Опис |
|--------------|------------------------|
| getCountry() | Отримання списку країн |

| Сигнатура | Опис |
|------------------------|--|
| filterOrders() | Фільтрація бронювань за вказаними параметрами |
| getAllParkings() | Отримання короткої інформації про всі парковки власника парковок |
| getOrders() | Отримання списку бронювань |
| processOrders(orders) | Обробка даних отриманих бронювань |
| createPagination() | Генерація пагінації для бронювань |
| changePage(pageNumber) | Перехід на вибрану сторінку бронювань |
| changeSize(size) | Зміна відображення кількості бронювань на сторінці |
| showLoader() | Запуск анімації на час завантаження даних |
| hideLoader() | Зупинка анімації |

3.3. Розробка інтерфейсу

Одним із важливих елементів програмного продукту є інтерфейс, адже від нього буде залежати, наскільки зручним буде користування і чи взагалі з ним будуть працювати. Інтерфейс має бути одночасно простим і інформативним і неперевантажувати користувача інформацією, як це часто буває. І відповідно таким сервісом ніхто не буде користуватися.

Розроблена система надає швидкий доступ до основних елементів кабінету власника паркувальних майданчиків. Реалізовано зручне меню до основних сторінок:

- Персональна інформація;
- Мої парковки;
- Керування цінами;
- Бронювання.

Перед початком використання системи, власник парковок повинен створити комерційний акаунт. Для того, щоб перейти до заповнення форми реєстрації, необхідно відкрити вікно авторизації і натиснути на посилання «Зареєструватися». У відкритому вікні необхідно обрати тип акаунта «Комерційний». Після того, як форма буде заповнена, потрібно натиснути кнопку «Зареєструватися». На рисунку 3.1 зображено вікно реєстрації для власників парковок.

The image shows a web application window titled "Реєстрація" (Registration) with a yellow header. At the top left, there are links for "Контакти" (Contacts) and "FAQ". Below the header, there are two radio buttons for account type: "Приватний" (Private) and "Комерційний" (Commercial), with the latter being selected. The form contains several input fields: "НАЗВА ФОП/ЮОП" (Company Name), "КРАЇНА" (Country), "МІСТО" (City), "ВУЛИЦЯ" (Street), "№ БУДИНКУ" (Building Number), "ІНН" (Tax ID), "EMAIL" (Email), "ТЕЛЕФОН" (Phone), "ПАРОЛЬ" (Password), and "ПІДТВЕРДЖЕННЯ ПАРОЛЮ" (Confirm Password). A green button labeled "ЗАРЕЄСТРУВАТИСЯ" (Register) is at the bottom. Below the button, there is a link "Вже зареєстровані? Війти" (Already registered? Log in). The window also has a close button (X) in the top left corner.

Рис. 3.1. Вікно реєстрації для власника парковок

Після успішного залогінення користувач переходить на сторінку персональної інформації, яка показана на рисунку 3.2

| Тип | Назва | Статус | |
|---|---|--------------|--|
| Витиска з державного реєстру | 27.pdf | Завантажений | |
| Свідчення про державну реєстрацію юридичної особи | main-merged.pdf | Завантажений | |
| Ідентифікаційний код | covid-vaccination-certificate_VOLODYMYR_LEVCHUK.pdf | Завантажений | |
| Свідчення про державну реєстрацію фізичної особи | 27.pdf | Завантажений | |

Рис. 3.2. Сторінка персональної інформації власника паркувальний майданчиків

У власному кабінеті, користувач може виконувати різноманітні дії, а саме:

- Редагувати персональну інформацію;
- Додавати, або видаляти документи, що підтверджують його компанію;
- Додавати, редагувати і видаляти власні паркувальні майданчики;
- Прикріплювати документи до паркувальних майданчиків
- Вказувати місця і ціну на парковці;
- Переглядати історію змін цін на парковках;
- Переглядати всі бронювання на власних парковаках;
- Переглядати детальну інформацію по кожному бронюванню.

Детальний опис функціоналу кабінету власника паркувальних майданчиків описаний в розділі «Керівництво користувача».

3.4. Керівництво користувача

Після успішної авторизації, чи реєстрації власник парковок автоматично переходить на сторінку з його персональною інформацією, яка показана на рисунку 3.3.

The screenshot shows the 'Booking Place' user interface. At the top, there is a navigation bar with 'Про нас', 'Контакти', and 'FAQ' links, along with a user profile icon and the text 'КиївПасТранс'. On the left, a sidebar menu contains 'Персональна інформація' (selected), 'Мої парковки', 'Керування цінами', 'Бронювання', and 'Вихід'. The main content area is divided into three sections: 'Інформація', 'Адреса', and 'Документи'. The 'Інформація' section contains fields for 'Назва компанії' (КиївПасТранс) and 'Код ЄДРПОУ' (53416532). The 'Адреса' section contains fields for 'Код країни' (Україна), 'Місто' (Київ), 'Вулиця' (Павлівська), and 'Номер будинку' (23). The 'Документи' section contains a table with columns 'Тип', 'Назва', 'Статус', and an action column. The table lists two documents: 'Паспорт' (place.png) and 'Свідоцтво про державну реєстрацію фізичної особи' (16520423.pdf), both with a status of 'Завантажений'. Below the table, there is a dropdown menu, a 'Вибрати файл' button, and a 'Завантажити' button.

Рис. 3.3. Сторінка персональної інформації власника парковок

Для можливості редагування інформації необхідно натиснути на кнопку редагування, яка знаходиться у правому верхньому куті сторінки і поля для введення даних стануть доступними для редагування, як це показано на рисунку 3.4

The screenshot shows the 'Booking Place' user interface in edit mode. The 'Інформація' section has a red border around the 'Назва компанії' and 'Код ЄДРПОУ' fields. The 'Адреса' section has a red border around the 'Код країни', 'Місто', 'Вулиця', and 'Номер будинку' fields. The 'Документи' section is the same as in the previous screenshot. The 'Вибрати файл' button is now labeled 'Вибрати файл' and the 'Завантажити' button is now labeled 'Завантажити'.

Рис. 3.4. Редагування персональної інформації

Якщо під час редагування даних була незаповнена, чи неправильно заповнена інформації, то біля відповідно поля, буде виведено повідомлення про некоретність заповнення даних, як це показано рисунку 3.5.

Рис. 3.5. Помилки під час неправильно заповнених даних

Після успішного редагування і збереження інформації відображається на певний час повідомлення результат збереження даних. На рисунку 3.6 зображено повідомлення про успішне редагування даних.

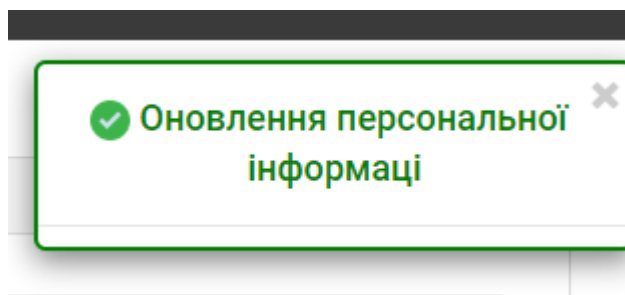


Рис. 3.6. Повідомлення про успішне редагування персональної інформації

Для додавання документів до кабінету, необхідно вибрати тип документу і завантажити сам файл, як це показано на рисунку 3.7

Рис. 3.7. Завантаження документів до кабінету власника парковок

Документи можна завантажувати тільки по одному на кожен з доступних типів, також розмір документу не має бути більшим за 5MB. На рисунку 3.8

зображено повідомлення у випадку спроби завантаження документу некоректного розміру.

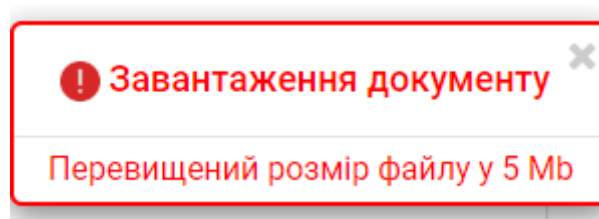


Рис. 3.8. Додавання файлу з перевищеним розміром

При успішному додаванні документу буде відповідне повідомлення. Для видалення документів, необхідно натиснути на кнопку видалення у вибраному документі, які виділені на рисунку 3.9.




| Документи | | | |
|--|---|--|--|
| Тип | Назва | Статус | |
| Свідоцтво про державну реєстрацію фізичної особи | 16520423.pdf | Завантажений |  |
| Виписка з державного реєстру | ПРИКЛАД-ПріоритетиАльтернатив.doc | Завантажений |  |
| Паспорт | place.png | Завантажений |  |
| <input type="text"/> | <input type="button" value="Вибрати файл"/> Файл не вибрано | <input type="button" value="Завантажити"/> | |


Рис. 3.9. Кнопки видалення документів

При видаленні документа буде відповідне повідомлення про помилку, чи успішне видалення і тип даного документу, знову буде можливим для вибору при додаванні нового документу.

При переході на сторінку «Мої замовлення», буде виведений список усіх парковок власника парковок, як це зображено на рисунку 3.10.

Booking Place

Про насКонтактиFAQ

КиївПасТранс

Персональна інформація

Мої парковки



Керування цінами

Бронювання

Вихід



1

Україна, м.Київ, вул.Чоколівський бульвар, 42/1





2

Україна, м.Київ, вул.Кудряшова, 16/1





3

Україна, м.Київ, вул.Березняківська, 19





4

Україна, м.Київ, вул.Леваневського, 2/3





5

Україна, м.Київ, вул.Генерала Шаповала, 5г





6

Україна, м.Київ, вул.Янгеля, 20





7

Україна, м.Київ, вул.Машинобудівна, 24а





8

Україна, м.Київ, вул.Василенка, 21/1





9

Україна, м.Київ, вул.Янгеля, 4





10

Україна, м.Київ, вул.Борщагівська, 154



11

Україна, м.Київ, вул.Борщагівська, 144



12

Україна, м.Київ, вул.Борщагівська, 121





Рис. 3.10. Список парковок власника парковок

Для додавання нової парковки необхідно натиснути на кнопку «+» і відкриється форма, як на рисунку 3.11

Парковка

Країна

Місто

Вулиця

Номер будинку

Ціна грн/год

Штраф грн/хв

Статус

Паркомісця

№

Статус

Номер місця

+

ЗБЕРЕГТИ

Рис. 3.11. Форма введення даних про нову парковку

У випадку, якщо незаповнені деякі поля, чи з помилками, то при збереженні даних, будуть наступні повідомлення про помилки, які зображені на рисунку 3.12

The screenshot shows a web form titled "Парковка" (Parking) with a close button (X) in the top right corner. The form contains several input fields, each with a red border indicating a validation error. The fields and their corresponding error messages are:

- Країна** (Country): A dropdown menu showing "Україна" (Ukraine).
- Місто** (City): A text input field with the placeholder "Місто". Below it, the error message "Місто має бути не менше 2 символів" (City must be at least 2 characters) is displayed.
- Вулиця** (Street): A text input field with the placeholder "Вулиця". Below it, the error message "Вулиця має бути не менше 2 символів" (Street must be at least 2 characters) is displayed.
- Номер будинку** (House number): A text input field with the placeholder "Номер будинку". Below it, the error message "Номер будинку має бути не менше одного символу" (House number must be at least one character) is displayed.
- Ціна грн/год** (Price UAH/hour): A text input field with the placeholder "Ціна". Below it, the error message "Ціна має бути більше нуля" (Price must be greater than zero) is displayed.
- Штраф грн/хв** (Fine UAH/hour): A text input field with the placeholder "Штраф". Below it, the error message "Штраф має бути більше нуля" (Fine must be greater than zero) is displayed.
- Статус** (Status): A dropdown menu showing "Новий" (New).

Below the form fields, there is a section labeled "Паркомісця" (Parking spaces) with a table containing two columns: "№" (Number) and "Статус" (Status). Below the table, there is a text input field for "Номер місця" (Space number) and a blue button with a "+" sign. At the bottom right, there is a blue button labeled "ЗБЕРЕГТИ" (Save).

Рис. 3.12. Виведення помилок для некоректно заповнених полів

Якщо буде введено невідому адресу, то виведеться відповідне повідомлення, як на рисунку 3.13.

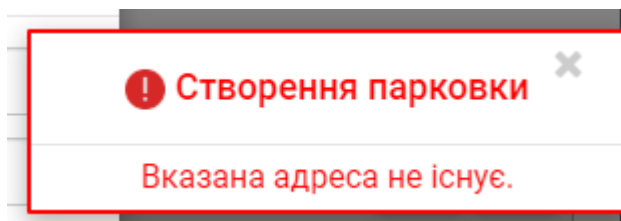


Рис. 3.13. Помилка додавання парковки на неіснуючу адресу

Після успішного додавання парковки необхідно відкрити створену парковку і завантажити документ, що засвідчує право на власність. Потрібно завантажувати тільки один документ, вимогу до самого файлу аналогічні, як і при завантаженні для власника парковок. На рисунку 3.14 виділено функціонал додавання документу.

Парковка

Країна: Україна

Місто: Київ

Вулиця: Борщагівська

Номер будинку: 121

Ціна грн/год: 25

Штраф грн/хв: 40

Статус: Інформація перевіряється

Документи

| Тип | Назва | Статус |
|----------------------------|--------------------------------|-------------|
| Технічний паспорт парковки | Вибрати файл Файл не вибрано | Завантажити |

Паркомісця

| № | Статус |
|---|--------|
| 1 | Вільне |
| 2 | Вільне |

Номер місця:

ЗБЕРЕГТИ

Рис. 3.14. Додавання документа до парковки

Після того як був доданий документ до парковки, адміністратор його перевіряє і змінює статус парковки на «Активна» і тепер вона активна для бронювання. Змінений статус показаний на рисунку 3.15

Статус

Активна

Рис. 3.15. Статус затвердженої парковки

Для редагування інформації про парковку, необхідно натиснути на відповідну кнопку певної вибраної парковки, кнопки редагування виділені на рисунку 3.16.

























| | | | |
|----|---|--|--|
| | | | + |
| 1 | Україна, м.Київ, вул.Чоколівський бульвар, 42/1 |  |  |
| 2 | Україна, м.Київ, вул.Кудряшова, 16/1 |  |  |
| 3 | Україна, м.Київ, вул.Березняківська, 19 |  |  |
| 4 | Україна, м.Київ, вул.Леваневського, 2/3 |  |  |
| 5 | Україна, м.Київ, вул.Генерала Шаповала, 5г |  |  |
| 6 | Україна, м.Київ, вул.Янгеля, 20 |  |  |
| 7 | Україна, м.Київ, вул.Машинобудівна, 24а |  |  |
| 8 | Україна, м.Київ, вул.Василенка, 21/1 |  |  |
| 9 | Україна, м.Київ, вул.Янгеля, 4 |  |  |
| 10 | Україна, м.Київ, вул.Борщагівська, 154 |  |  |
| 11 | Україна, м.Київ, вул.Борщагівська, 144 |  |  |
| 12 | Україна, м.Київ, вул.Борщагівська, 121 |  |  |

Рис. 3.16. Кнопки редагування парковок

При відкритті парковки на редагування доступними до редагування будуть наступні дані, які показані на рисунку 3.17. Поки не виконано жодних змін даних, то кнопка збереження неактивна. Можна додавати і видаляти паркомісця на парковці. Результат даних дій буде супроводжуватися відповідними повідомленнями. Важливим критерієм для видалення паркомісця, є те що воно має бути вільним, інакше система заборонить дану операцію.

Парковка

Країна

Україна

Місто

Київ

Вулиця

Березняківська

Номер будинку

19

Ціна грн/год

7

Штраф грн/хв

25

Статус

Активна

Документи

| Тип | Назва | Статус |
|----------------------------|--|------------------------|
| Технічний паспорт парковки | <div>Вибрати файл</div> <div>Файл не вибрано</div> | <div>Завантажити</div> |

Паркомісця

| № | Статус | |
|----|---------|--|
| 1A | Вільне | |
| 1B | Зайняте | |
| 2A | Вільне | |
| 2B | Вільне | |
| 3A | Вільне | |
| 3B | Вільне | |
| 4A | Вільне | |

Рис. 3.17. Форма редагування даних парковки

Парковки також можна видаляти, але за умови що на ній не має активних бронювань, інакше буде виведено повідомлення, як на рисунку 3.18 і система не видалить вибрану парковку.

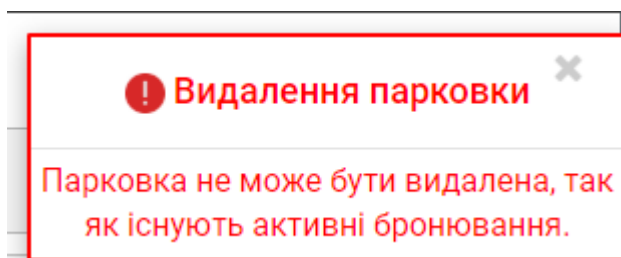


Рисунок 3.18. Повідомлення про неможливість видалення парковки

Переглянути історію змін вартості оренди паркомісць можна на сторінці «Керування цінами» у власному кабінеті. При переході на дану сторінку

виводиться список усіх парковок власника парковок, як це показано на рисунку 3.19

| Booking Place | | | Про нас | Контакти | FAQ | КиївПасТранс | |
|------------------------|----|---|-------------|----------|-----|--------------|--|
| Персональна інформація | 1 | Україна, м.Київ, вул.Чоколівський бульвар, 42/1 | 24 грн/год. | | | | |
| Мої парковки | 2 | Україна, м.Київ, вул.Кудряшова, 16/1 | 8 грн/год. | | | | |
| Керування цінами | 3 | Україна, м.Київ, вул.Березняківська, 19 | 7 грн/год. | | | | |
| Бронювання | 4 | Україна, м.Київ, вул.Леваневського, 2/3 | 16 грн/год. | | | | |
| Вихід | 5 | Україна, м.Київ, вул.Генерала Шаповала, 5г | 15 грн/год. | | | | |
| | 6 | Україна, м.Київ, вул.Янгеля, 20 | 23 грн/год. | | | | |
| | 7 | Україна, м.Київ, вул.Машинобудівна, 24а | 31 грн/год. | | | | |
| | 8 | Україна, м.Київ, вул.Василенка, 21/1 | 11 грн/год. | | | | |
| | 9 | Україна, м.Київ, вул.Янгеля, 4 | 14 грн/год. | | | | |
| | 10 | Україна, м.Київ, вул.Борщагівська, 154 | 10 грн/год. | | | | |
| | 11 | Україна, м.Київ, вул.Борщагівська, 144 | 20 грн/год. | | | | |
| | 12 | Україна, м.Київ, вул.Борщагівська, 121 | 25 грн/год. | | | | |

Рис. 3.19. Список парковок власника парковок

При кліку на певну парковку відкривається з графік змін цін, де показано динаміку змін з часом. На рисунку 3.20 наведено графік змін для обраної парковки.

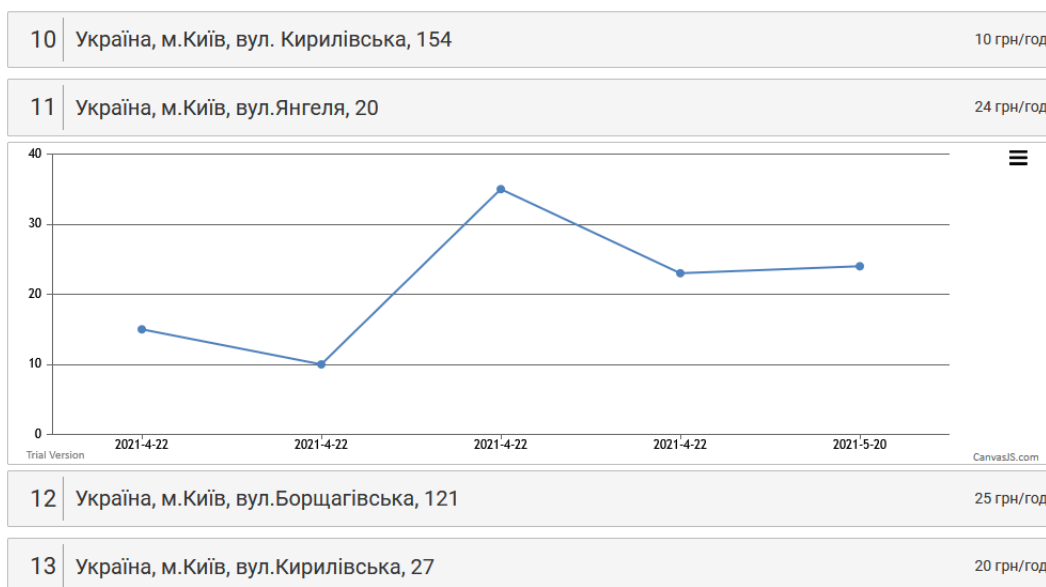


Рис. 3.20. Графік змін ціни на оренду паркомісця

При виділенні певної області графіку він буде деталізуватися, також можна його роздрукувати і зберегти, як зображення у різних форматах, тільки для цього

необхідно натиснути на відповідну кнопку і вибрати опцію, як це показано на рисунку 3.21.

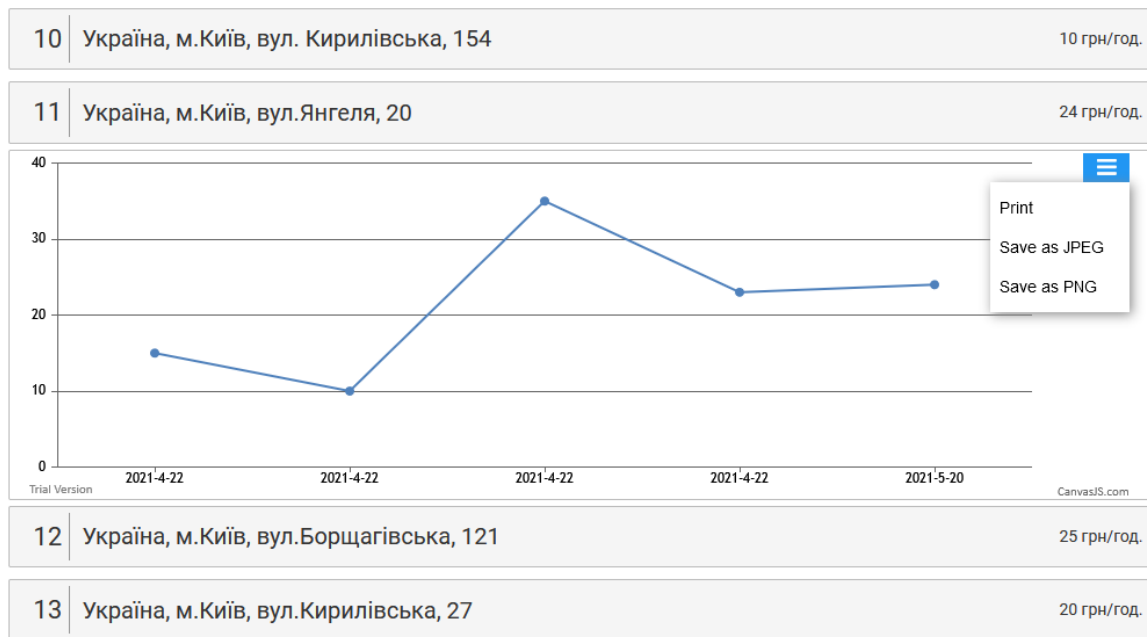


Рис. 3.21. Додаткові опції для графіку

Власник парковок має мати можливість переглядати бронювання по власних парковках. Для цього було зроблено окрему сторінку, де виводяться всі бронювання на його парковках. Так як бронювань буде дуже багато, було реалізовано їх фільтрацію, де можна вибрати парковку і вказати часовий діапазон і виконати пошук бронювань, як це показано на рисунку 3.22.

The screenshot shows the search results for parking reservations. The search criteria are:

- Парковка: Україна, м.Київ, вул.Янгеля, 20
- Дата з: [empty]
- Дата до: [empty]
- Пошук: [button]

The results table is as follows:

| № | Статус | Штраф | Сума |
|---|--------------|-------|--------|
| 1 | Заброньовано | 0 грн | 24 грн |

Below the table, the details of the reservation are shown:

- Паркомісце: 1
- Час заїзду: 15 грудня 2024 13:00
- Час виїзду: 15 грудня 2024 14:00
- Фактичний час виїзду: Не виїхав
- Час стоянки: 1 год.
- Фактичний час стоянки: [empty]

At the bottom right, there are buttons for pagination: 5, 15, 25, and 50.

Рис. 3.22. Результат пошуку бронювань по вказаним параметрам

Було додано пагінацію, щоб не виводити всі дані на одній сторінці і також можна вибирати кількість бронювань на одній сторінці. На рисунку 3.23 виділено пагінацію і вибір кількість відображення бронювань на сторінці.

The screenshot shows a web interface for parking reservations. At the top, there is a search bar with fields for 'Парковка' (Parking) with a dropdown menu showing 'Україна, м.Київ, вул.Янгеля, 20', 'Дата з' (Date from), and 'Дата до' (Date to). A blue 'Пошук' (Search) button is on the right. Below the search bar is a table with columns: '№' (Number), 'Статус' (Status), 'Штраф' (Fine), and 'Сума' (Sum). The table contains one row with '1' in the first column, 'Заброньовано' (Reserved) in the second, '0 грн' in the third, and '24 грн' in the fourth. Below the table is a detailed view of the reservation for item 1, showing fields for 'Паркомісце' (Parking space), 'Час заїзду' (Arrival time), 'Час виїзду' (Departure time), 'Фактичний час виїзду' (Actual departure time), 'Час стоянки' (Parking time), and 'Фактичний час стоянки' (Actual parking time). At the bottom right, there is a pagination control with four buttons: '5', '15', '25', and '50'. The '5' button is highlighted with a red border.

| № | Статус | Штраф | Сума |
|---|--------------|-------|--------|
| 1 | Заброньовано | 0 грн | 24 грн |

Детальна інформація про бронювання №1:

- Паркомісце: 1
- Час заїзду: 15 грудня 2024 13:00
- Час виїзду: 15 грудня 2024 14:00
- Фактичний час виїзду: Не виїхав
- Час стоянки: 1 год.
- Фактичний час стоянки:

Пагінація: 5 15 25 50

Рис. 3.23. Пагінація і вибір кількість відображення бронювань на сторінці

При кліку на відповідне бронювання відкривається блок з детальною інформацією, яка є доступною для власника парковок.

Висновки до розділу

В даному розділі було наведено детальну інструкцію користувача для роботи з підсистемою з керування кабінетом власника майданчиків автомобільних паркувань, де описано весь доступний його функціонал. Велику увагу привернуто функціоналу по роботі з парковками.

Також було описано методи класів серверної і клієнтської частини застосунку.

Розроблено інтерфейс користувача і детально описано керівництво користувача, де пройдено по всіх головних аспектах функціоналу по керуванню кабінетом власника майданчиків автомобільних паркувань.

ВИСНОВКИ

У рамках магістерського дослідження було розроблено систему з керування кабінетом власника майданчиків автомобільних паркувань.

Для досягнення поставленої мети було виконано наступні задачі:

- розроблено систему, де можливо здійснювати керування парковками;
- розроблено керування паркувальними майданчиками та наявною кількістю паркомісць;
- розроблено можливість перегляду динаміки змін цін на паркувальних майданчиках і бронювань по парковках.

У першому розділі було проаналізовано майбутню систему і виконано порівняння з існуючими аналогами. Також було визначено змістовну постановку задачі, на основі якої побудовано математичну постановку задачі. Основною задачею даної індивідуальної частини є шифрування даних. Для цього було проаналізовано алгоритми шифрування і вибрано AES. Даний вибір обґрунтовано і наведено детальний опис його роботи і рівень захищеності даних.

В другому розділі була описана архітектура застосунку, мінімальні вимоги до технічного забезпечення для розгортання застосунку. Також було побудовано необхідні діаграми, серед яких: діаграми послідовності та станів з коротким описом. На діаграмах показаний процес створення нової парковки починаючи від реєстрації власника парковок в системі.

В третьому розділі описано вибрані засоби розробки і обґрунтовано, чому вибрані саме вони. Також в розділі наведено специфікації методів класів серверної і клієнтської частини застосунку. У даному розділі опис процес проектування інтерфейсу користувача і наведено керівництво користувача з описом всіх важливих функціональних елементів підсистеми.

На даному етапі розробки, система готова до бета-тестування реальними власниками парковок.

Майбутньому планується зробити розширення збору статистичних даних для власників парковок. На разі статистика є тільки для тарифу паркування. По аналогії можна реалізувати для штрафів. Але більш перспективною буде

аналітика про пошукові запити водії і збір даних, де найбільше бронювань. Дана статистика надасть інформацію для власників парковок про подальший розвиток власного паркобізнесу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. 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>.
2. 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>.
3. Сучасні паркувальні системи. [Електронний ресурс]. Режим доступу: <https://dentaauto.ua/raznoe/sovremennye-parkovochnye-sistemy.html>.
4. Розумна парковка та її переваги [Електронний ресурс]. Режим доступу: <https://ula.lantec.ua/statti/rozumna-parkovka-ta-jiji-perevagi>
5. Игнатьев Ю.В. Возведение автомобильных стоянок и парковок в крупных городах //К: 2019. № 17. С. 68–72. 6. Автоматизовані системи платного паркування [Електронний ресурс]. <https://parkmag./news/obzor-avtomatizirovannyh-sistem-platnoyparkovki.html>
6. Системи управління парковками [Електронний ресурс] – режим доступу: <https://ia.ua/solutions-ru/infrastructure-object/building-automation/199-parking-control-systems/>
7. Автоматизована система парковки [Електронний ресурс] – режим доступу: <https://lotgroup.ua/product/smart-city/automated-parking-system/>
8. Автоматизована система парковки [Електронний ресурс] – режим доступу: https://idcard.com.ua/solutions/parking_rfid/
9. Автоматизована система парковки [Електронний ресурс] – режим доступу: <http://rfidukraine.com.ua/automation-of-parking-based-on-uhftechnology/>
10. ЗАКОН УКРАЇНИ Про захист персональних даних [Електронний ресурс] // Режим доступу: <https://zakon.rada.gov.ua/laws/show/2297-17>

11. GDPR [Электронный ресурс] // Режим доступа: <https://dictionary.cambridge.org/ru/словарь/английский/gdpr>
12. Advanced Encryption Standard [Электронный ресурс] // Режим доступа: https://ru.wikipedia.org/wiki/Advanced_Encryption_Standard
13. Цифра дня: статистика автомобілів на планеті? [Электронный ресурс] // Режим доступа: <http://mmr.net.ua/autoworld/news/94527>
14. Паркувальний майданчик(парковка) [Электронный ресурс] // Режим доступа: <https://slovotvir.org.ua/words/parkovka>
15. Why Spring? [Электронный ресурс] // Режим доступа: <https://spring.io/why-spring>
16. TypeScript Documentation [Электронный ресурс] // Режим доступа: <https://www.typescriptlang.org/docs/>
17. PostgreSQL: The World's Most Advanced Open Source Relational Database [Электронный ресурс] // Режим доступа: <https://www.postgresql.org/>
18. Git Documentation [Электронный ресурс] // Режим доступа: <https://git-scm.com/doc>
19. The Model View Controller Pattern – MVC Architecture and Frameworks Explained [Электронный ресурс] // Режим доступа: <https://www.freecodecamp.org/news/the-model-view-controller-pattern-mvc-architecture-and-frameworks-explained/>
20. Angular [Электронный ресурс] // Режим доступа: <https://angular.io/>
21. Gosling and McGilton (May 1996). The Java Language Environment.
22. J. Gosling, B. Joy, G. Steele, G. Brachda. The Java Language Specification, 2nd Edition
23. James Gosling; Bill Joy, Guy Steele, Gilad Bracha (2005). The Java Language Specification, Third Edition. Addison-Wesley.
24. Кей С. Хорстманн (2014). Java SE 8. Вводный курс.
25. Fain Y., Moiseev A. Angular Development with TypeScript. New York: Manning Publications, 2019. 560 p.

26. 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.
27. Стаття 13. Інформація, яку необхідно надати у разі збирання персональних даних від суб'єкта даних | GDPR-Text.com. GDPR Text. 2019-10-14

ДОДАТОК А

Лістинг коду серверної частини системи

```
@Service
public class DocumentService {

    private static final Logger LOGGER =
LoggerFactory.getLogger(DocumentService.class);

    @Autowired
    private DocumentTypeRepository documentTypeRepository;

    @Autowired
    private DocumentTypeMapper documentTypeMapper;

    @Autowired
    private DocumentMapper documentMapper;

    @Autowired
    private DocumentRepository documentRepository;

    @Autowired
    private ParkingRepository parkingRepository;

    @Autowired
    private SupplierRepository supplierRepository;

    @Autowired
    private FileManager fileManager;

    @Autowired
    private UserContext userContext;

    @Value("${document.basePath}")
    private String documentBasePath;

    @Value("${document.upload.allowed}")
    private Boolean uploadAllowed;

    public List<com.kpi.parking.model.DocumentType> getParkingDocumentTypes() {
        return getDocumentTypesByPredicate(documentType.parking.isTrue());
    }

    public List<com.kpi.parking.model.DocumentType> getSupplierDocumentTypes() {
        return getDocumentTypesByPredicate(documentType.supplier.isTrue());
    }

    public List<com.kpi.parking.model.DocumentType>
getDocumentTypesByPredicate(Predicate where) {

        Iterable<DocumentType> documentTypes =
documentTypeRepository.findAll(where);

        LOGGER.debug("Retrieved document types: {}", documentTypes);

        return StreamSupport.stream(documentTypes.spliterator(), false)
            .map(documentTypeMapper::map)
            .collect(Collectors.toList());
    }

    @Transactional
    public DocumentMetadata uploadParkingDocument(
        String parkingReference, MultipartFile parkingDocument, String
```



```

documentTypeCode) {

    Parking parking =
parkingRepository.findOne(QParking.parking.reference.eq(parkingReference)).get();

    Document document = createDocument(parkingReference, parkingDocument,
documentTypeCode);

    document = uploadDocument(document, parkingDocument);
parkingRepository.save(parking.addDocument(document));

    LOGGER.info("Parking document successfully uploaded.");

    return documentMapper.map(document);
}

@Transactional
public DocumentMetadata uploadSupplierDocument(MultipartFile
supplierDocument, String documentTypeCode) {

    String username = userContext.getCurrentUsername();
    Supplier supplier = supplierRepository.findByUserEmail(username);

    Document document = createDocument(username, supplierDocument,
documentTypeCode);

    document = uploadDocument(document, supplierDocument);
supplierRepository.save(supplier.addDocument(document));

    LOGGER.info("Supplier document successfully uploaded.");

    return documentMapper.map(document);
}

private Document uploadDocument(Document document, MultipartFile
multipartFile) {
    if (uploadAllowed && !fileManager.saveFile(document.getPath(),
multipartFile)) {
        LOGGER.error("Error occurs during uploading of the document.");
        throw new DocumentUploadException();
    }

    return documentRepository.save(document);
}

private Document createDocument(String uniqueId, MultipartFile document,
String documentTypeCode) {
    DocumentType documentType =
documentTypeRepository.findById(documentTypeCode)
        .orElseThrow(IllegalStateException::new);

    String originalFilename = document.getOriginalFilename();
    DateTimeFormatter dateTimeFormatter =
DateTimeFormatter.ofPattern("ddMMyyyyHHmmss");

    String fileName = String.format("%s_%s.%s",
StringUtils.substringBeforeLast(originalFilename, "."),
        dateTimeFormatter.format(LocalDateTime.now()),
StringUtils.substringAfterLast(originalFilename, "."));
    String path = String.format("%s/%s/%s", documentBasePath, uniqueId,
fileName);

    return new Document(originalFilename, document.getSize(), path,
DocumentStatus.SUBMITTED, documentType);
}

```

```

    @Transactional
    public void deleteDocumentByReference(@NotBlank String documentReference) {

        Document persistedDocument =
documentRepository.findOne(document.reference.eq(documentReference))
                .orElseThrow(IllegalStateException::new);

        persistedDocument.setStatus(DocumentStatus.DELETED);

        documentRepository.save(persistedDocument);
    }
}

public class FileManager {

    private static final Logger LOGGER =
LoggerFactory.getLogger(FileManager.class);

    public boolean saveFile(@NotBlank String path, @NotNull MultipartFile
multipartFile) {

        LOGGER.debug("Saving file {}. ", path);

        try {
            FileUtils.writeByteArrayToFile(new File(path),
multipartFile.getBytes());
        } catch (IOException ex) {
            LOGGER.error(String.format("Error occurs during saving of file %s",
path));
            return false;
        }

        return true;
    }
}

@RestController
@Validated(DocumentUploadSequence.class)
@Scope(proxyMode = ScopedProxyMode.TARGET_CLASS)
public class DocumentController implements DocumentsApi {

    private static final Logger LOGGER =
LoggerFactory.getLogger(DocumentController.class);

    @Autowired
    private DocumentService documentService;

    @Override
    public ResponseEntity<List<DocumentType>> getParkingDocumentTypes() {

        LOGGER.info("Request to retrieve list of document types specific for
parkings.");

        return ResponseEntity.ok(documentService.getParkingDocumentTypes());
    }

    @Override
    public ResponseEntity<List<DocumentType>> getSupplierDocumentTypes() {

        LOGGER.info("Request to retrieve list of document types specific for
suppliers.");

        return ResponseEntity.ok(documentService.getSupplierDocumentTypes());
    }
}

```

```

        @Override
        public ResponseEntity<DocumentMetadata> uploadParkingDocument(
            @ParkingKnownAndBelongsToSupplier(groups =
ParkingKnownAndBelongsToSupplier.class)
            @PathVariable("reference") String reference,
            @ExtensionIsAllowed(groups = ExtensionIsAllowed.class)
            @RequestPart(value = "parkingDocument") MultipartFile
parkingDocument,
            @KnownDocumentType(groups = KnownDocumentType.class)
            @RequestPart(value = "documentTypeCode") String documentTypeCode) {

            LOGGER.info("Request to upload document for parking with code: {}",
documentTypeCode);

            return
ResponseEntity.ok(documentService.uploadParkingDocument(reference,
parkingDocument, documentTypeCode));
        }

        @Override
        @DocumentExistForParking(groups = DocumentExistForParking.class)
        public ResponseEntity<Void> deleteParkingDocument(
            @PathVariable("documentReference") String documentReference,
            @ParkingKnownAndBelongsToSupplier(groups =
ParkingKnownAndBelongsToSupplier.class)
            @PathVariable("parkingReference") String parkingReference) {

            documentService.deleteDocumentByReference(documentReference);

            return ResponseEntity.ok().build();
        }

        @Override
        public ResponseEntity<DocumentMetadata> uploadSupplierDocument(
            @ExtensionIsAllowed(groups = ExtensionIsAllowed.class)
            @RequestPart(value = "supplierDocument") MultipartFile
supplierDocument,
            @KnownDocumentType(groups = KnownDocumentType.class)
            @RequestPart(value = "documentTypeCode") String documentTypeCode) {

            LOGGER.info("Request to upload document for supplier with code: {}",
documentTypeCode);

            return
ResponseEntity.ok(documentService.uploadSupplierDocument(supplierDocument,
documentTypeCode));
        }

        @Override
        public ResponseEntity<Void> deleteSupplierDocument(
            @DocumentExistForSupplier(groups = DocumentExistForSupplier.class)
            @PathVariable("documentReference") String documentReference) {

            documentService.deleteDocumentByReference(documentReference);

            return ResponseEntity.ok().build();
        }
    }

    @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> getSupplierParkings() {

        List<Parking> parkings =
parkingRepository.findAllWithAddressByEmail(userContext.getCurrentUsername());

        LOGGER.debug("Found {} active parkings: {}", parkings.size(),
parkings);

        return parkings.stream()
            .map(parkingMapper::map)
            .collect(Collectors.toList());
    }

```

```

    public com.kpi.parking.model.ParkingDetailsResponse
getParkingInfoByReference(@NotBlank String reference) {

    Parking parking = parkingProvider.getParkingByReference(reference);
    parking.setPlaces(parking.getPlaces().stream()
        .filter(p -> p.getStatus() != PlaceStatus.DELETED)
        .collect(Collectors.toSet()));

    LOGGER.debug("Found parking: {}", parking);

    return parkingMapper.mapToParkingDetails(parking);
}

    public com.kpi.parking.model.ParkingPriceResponse
getParkingPriceHistory(String reference) {

    Parking parking = parkingRepository.findOneWithPrices(reference);

    LOGGER.debug("Found active parking: {}.", parking);

    com.kpi.parking.model.ParkingPriceResponse parkingPriceResponse =
parkingMapper.mapToParkingPrice(parking);
    parkingPriceResponse.setPrices(parkingPriceResponse.getPrices().stream()
        .sorted(Comparator.comparing(Price::getStartDate))
        .collect(Collectors.toList()));

    return parkingPriceResponse;
}

    public com.kpi.parking.model.ParkingDetailsResponse
createParking(com.kpi.parking.model.ParkingRequest parkingRequest) {

    Supplier supplier =
supplierRepository.findByUserEmail(userContext.getCurrentUser());

    Parking parking = parkingMapper.map(parkingRequest, supplier);
    parking.setStatus(ParkingStatus.UNDER_VALIDATION);

    addressService.enrichWithCoordinates(parking.getAddress());

    parkingRepository.save(parking);

    return parkingMapper.mapToParkingDetails(parking);
}

    @Transactional
    public void updateParking(@NotBlank String reference,
com.kpi.parking.model.ParkingRequest parkingRequest) {

    Parking parking = parkingProvider.getParkingByReference(reference);

    Address address =
addressProvider.getAddress(parkingRequest.getAddress());

    if (address != null) {
        parking.setAddress(address);
    } else {
        parking.setAddress(addressMapper.map(parkingRequest.getAddress()));
    }

    if (parking.getSize() < parkingRequest.getPlaces().size()) {
        addNewPlaces(parking, parkingRequest.getPlaces());
    }
}

```

```

        com.kpi.parking.parking.domain.Price price = parking.getPrice();

        if (price.getPrice().doubleValue() != parkingRequest.getPrice()
            || price.getPenalty().doubleValue() !=
parkingRequest.getPenalty()) {
            updatePriceHistory(parking, parkingRequest.getPrice(),
parkingRequest.getPenalty());
        }

        addressService.enrichWithCoordinates(parking.getAddress());

        parkingRepository.save(parking);
    }

    private void addNewPlaces(Parking parking, List<String> places) {
        parking.setSize(places.size());

        places.removeAll(parking.getPlaces().stream()
            .map(Place::getPlaceNumber)
            .collect(Collectors.toList()));

        LOGGER.debug("Adding of {} new places to parking {}.", places.size(),
parking);

        parking.getPlaces().addAll(places.stream()
            .map(Place::new)
            .peek(p -> p.setParking(parking))
            .collect(Collectors.toSet()));
    }

    private void updatePriceHistory(Parking parking, double newPrice, double
newPenalty) {
        BigDecimal price = BigDecimal.valueOf(newPrice);
        BigDecimal penalty = BigDecimal.valueOf(newPenalty);

        parking.setPrice(new
com.kpi.parking.parking.domain.Price().price(price).penalty(penalty));

        List<PriceHistory> prices = parking.getPriceHistories();
        PriceHistory currentPrice = prices.stream()
            .filter(p -> p.getEndDate() == null)
            .findFirst()
            .orElse(null);

        if (currentPrice == null) {
            LOGGER.error("Parking {} has no active price.", parking);
            throw new IllegalStateException(
                String.format("Parking with reference - %s has no current
price.", parking.getReference()));
        }

        LocalDateTime currentDateTime = LocalDateTime.now();

        currentPrice.setEndDate(currentDateTime);
        prices.add(new PriceHistory(price, penalty, currentDateTime, parking));
    }

    @Transactional
    public void deleteParkingByReference(@NotBlank String reference) {

        Parking persistedParking = parkingRepository.findByReference(reference);
        persistedParking.setStatus(ParkingStatus.DELETED);

        parkingRepository.save(persistedParking);
    }

```

```

    }

    @Transactional
    public void deleteParkingPlace(@NotBlank String parkingReference, @NotBlank
String placeReference) {

        Parking parking =
parkingProvider.getParkingByReference(parkingReference);
        parking.setSize(parking.getSize() - 1);

        Place place = placeRepository.findByReference(placeReference);
        place.setStatus(PlaceStatus.DELETED);

        parkingRepository.save(parking);
        placeRepository.save(place);
    }

    @Transactional(readonly = true)
    public com.kpi.parking.model.PageableOrderResponse getOrderPage(String
parkingReference, Integer pageNumber, Integer pageSize,
                                                                    LocalDate
fromDate, LocalDate toDate) {

        Function<OrderSearchRequest, Page<Order>> orderSearchFunction = request
-> orderRepository

.findAllByParkingReferenceAndCreatedDateBetween(request.getIdentifier(),
request.getFromDateTime(),
            request.getToDateTime(), request.getPageable());

        return orderService.findOrders(parkingReference,
PageRequest.of(pageNumber, pageSize), fromDate, toDate,
            orderSearchFunction);
    }

    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());
    }

    private boolean isParkingInRadius(Address startAddress, Address endAddress)
{
        return distanceService.calculateDistanceBetweenPoints(startAddress,
endAddress) <= searchRadius;
    }

    private LocalDateTime parseToDateTime(String dateTime) {
        return
ParserUtils.parseToLocalDateTime(dateTime).plusMinutes(dateTimeToOffset);
    }
}

@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
public ResponseEntity<List<ParkingResponse>> getSupplierParkings() {

    LOGGER.info("Retrieving of all active parking with available places.");

    return ResponseEntity.ok(parkingService.getSupplierParkings());
}

@Override
public ResponseEntity<ParkingDetailsResponse> getParkingByReference(
    @ParkingKnownAndBelongsToSupplier(groups =
ParkingKnownAndBelongsToSupplier.class)
    @PathVariable("reference") String reference) {

    LOGGER.info("Retrieving of parking by reference: {}.", reference);

    return
ResponseEntity.ok(parkingService.getParkingInfoByReference(reference));
}

@Override
@CheckSupplier
public ResponseEntity<ParkingDetailsResponse> createParking(
    @RequestBody ParkingRequest parking) {

    LOGGER.info("Request to create new parking.");

    return ResponseEntity.ok(parkingService.createParking(parking));
}

@Override
@UniqueAddressPerParking(groups = UniqueAddressPerParking.class)
public ResponseEntity<Void> updateParking(
    @ParkingKnownAndBelongsToSupplier(groups =
ParkingKnownAndBelongsToSupplier.class)
    @PathVariable("reference") String reference,
    @RequestBody ParkingRequest parking) {

    LOGGER.info("Request to update parking with reference: {}.", reference);

    parkingService.updateParking(reference, parking);

    return ResponseEntity.ok().build();
}

@Override
public ResponseEntity<Void> deleteParking(
    @ParkingKnownAndBelongsToSupplier(groups =
ParkingKnownAndBelongsToSupplier.class)
    @ParkingIsDeletable(groups = ParkingIsDeletable.class)
    @PathVariable("reference") String reference) {

    LOGGER.info("Request to delete parking with reference: {}.", reference);

    parkingService.deleteParkingByReference(reference);

    return ResponseEntity.ok().build();
}

@Override
@PlaceBelongsToParking(groups = PlaceBelongsToParking.class)
@ParkingPlaceIsDeletable(groups = ParkingPlaceIsDeletable.class)
public ResponseEntity<Void> deleteParkingPlace(

```



```

        @ParkingKnownAndBelongsToSupplier(groups =
ParkingKnownAndBelongsToSupplier.class)
        @PathVariable("parkingreference") String parkingReference,
        @PathVariable("placereference") String placeReference) {

            LOGGER.info("Request to delete parking place with reference: {}",
placeReference);

            parkingService.deleteParkingPlace(parkingReference, placeReference);

            return ResponseEntity.ok().build();
        }

        @Override
        public ResponseEntity<ParkingPriceResponse> getParkingsPriceHistory(
            @ParkingKnownAndBelongsToSupplier(groups =
ParkingKnownAndBelongsToSupplier.class)
            @PathVariable("reference") String reference) {

            LOGGER.info("Request to retrieve price history.");

            return
ResponseEntity.ok(parkingService.getParkingPriceHistory(reference));
        }

        @Override
        public ResponseEntity<PageableOrderResponse> getParkingOrders(
            @ParkingKnownAndBelongsToSupplier(groups =
ParkingKnownAndBelongsToSupplier.class)
            @PathVariable("reference") String reference,
            @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) {

            LOGGER.info("Request to retrieve all order for parking with reference:
{}, which were placed from {} to {}",
reference, fromDate, toDate);

            return ResponseEntity.ok(parkingService.getOrderPage(reference,
pageNumber, pageSize, fromDate, toDate));
        }
    }

    public class SupplierRepositoryImpl implements SupplierRepositoryCustom {

        @PersistenceContext
        private EntityManager em;

        @Override
        public Supplier findWithAllInformationByEmail(@NotBlank String email) {
            Supplier foundSupplier = new JPAQuery<Supplier>(em)
                .select(supplier)
                .from(supplier)
                .join(supplier.user, user).fetchJoin()
                .join(supplier.address).fetchJoin()
                .leftJoin(supplier.documents).fetchJoin()
                .where(user.email.eq(email))
                .fetchFirst();

            foundSupplier.getDocuments().removeIf(d -> d.getStatus() ==
DocumentStatus.DELETED);

            return foundSupplier;
        }
    }

```

```

    }

    @Override
    public boolean existWithDocument(String documentReference, String email) {
        return new JPAQuery<Supplier>(em)
            .select(supplier)
            .from(supplier)
            .join(supplier.user, user)
            .join(supplier.documents, document)
            .where(user.email.eq(email)
                .and(document.reference.eq(documentReference))
                .and(document.status.ne(DocumentStatus.DELETED)))
            .fetchCount() > 0;
    }
}

@Service
public class SupplierService {

    private static final Logger LOGGER =
LoggerFactory.getLogger(SupplierService.class);

    @Autowired
    private SupplierRepository supplierRepository;

    @Autowired
    private SupplierMapper supplierMapper;

    @Autowired
    private AddressMapper addressMapper;

    @Autowired
    private AddressService addressService;

    @Autowired
    private UserContext userContext;

    public Supplier getSupplierWithAllInfo(String username) {
        return supplierRepository.findWithAllInformationByEmail(username);
    }

    @Transactional
    public void updateSupplierInfo(SupplierBase supplierBase) {

        Supplier supplier =
getSupplierWithAllInfo(userContext.getCurrentUsername());

        supplierMapper.map(supplier, supplierBase);
        addressMapper.map(supplier.getAddress(), supplierBase.getAddress());

        addressService.enrichWithCoordinates(supplier.getAddress());

        supplierRepository.save(supplier);
    }
}

@RestController
@Scope(proxyMode = ScopedProxyMode.TARGET_CLASS)
public class SupplierManagementController implements SupplierApi {

    private static final Logger LOGGER =
LoggerFactory.getLogger(SupplierManagementController.class);

    @Autowired
    private SupplierService supplierService;

    @Autowired

```

```

private UserContext userContext;

@Autowired
private SupplierMapper supplierMapper;

@Override
public ResponseEntity<SupplierResponse> getSupplier() {

    LOGGER.info("Request to retrieve info about customer.");

    Supplier supplier =
supplierService.getSupplierWithAllInfo(userContext.getCurrentUsername());

    return ResponseEntity.ok(supplierMapper.map(supplier));
}

@Override
public ResponseEntity<Void> updateSupplierInfo(@Valid SupplierBase supplier)
{

    LOGGER.info("Request to update info about supplier.");

    supplierService.updateSupplierInfo(supplier);

    return ResponseEntity.ok().build();
}

@Override
public boolean isParkingLinkedToActiveOrder(String reference) {
    return new JPAQuery<Boolean>(em)
        .select(order)
        .from(order)
        .innerJoin(order.parking, parking)
        .where(order.status.notIn(OrderStatus.CLOSED_SUCCESSFULLY,
OrderStatus.CLOSED_WITH_PENALTY)
            .and(parking.reference.eq(reference)))
        .fetchCount() > 0;
}

@Override
public Parking findWithDependenciesByReference(String reference) {
    Parking result = new JPAQuery<>(em)
        .select(parking)
        .from(parking)
        .innerJoin(parking.address).fetchJoin()
        .innerJoin(parking.supplier, supplier).fetchJoin()
        .innerJoin(supplier.user).fetchJoin()
        .leftJoin(parking.documents).fetchJoin()
        .leftJoin(parking.places, place).fetchJoin()
        .where(parking.reference.eq(reference))
        .fetchFirst();

    result.getPlaces().removeIf(pl -> pl.getStatus() == PlaceStatus.DELETED);
    result.getDocuments().removeIf(d -> d.getStatus() ==
DocumentStatus.DELETED);

    return result;
}

@Override
public boolean existWithDocument(String parkingReference, String
documentReference) {
    return new JPAQuery<>(em)
        .select(parking)

```

```

        .from(parking)
        .join(parking.documents, document)
        .where(parking.reference.eq(parkingReference)
            .and(document.reference.eq(documentReference))
            .and(document.status.ne(DocumentStatus.DELETED)))
        .fetchCount() > 0;
    }
}

public interface PlaceRepository extends JpaRepository<Place, Long>,
PlaceRepositoryCustom {

    Place findByReference(String reference);

    @Query("SELECT CASE WHEN count(o) > 0 THEN true ELSE false END FROM Order o
" +
        "INNER JOIN o.orderItems oi " +
        "INNER JOIN o.parking p " +
        "INNER JOIN p.places pl " +
        "WHERE p.reference = :parkingReference AND o.status <> 'CLOSED' " +
        "AND pl.reference = :placeReference AND oi.placeNumber =
pl.placeNumber")
    boolean isPlaceLinkedToActiveOrder(@Param("parkingReference") String
parkingReference,
                                        @Param("placeReference") String
placeReference);
}

@Service
public class AESCryptoService implements CryptoService {

    private static final Logger LOGGER =
LoggerFactory.getLogger(AESCryptoService.class);

    @Override
    public String encrypt(@NotBlank String stringToEncrypt, String
secretKeyString) throws CryptoException {

        try {
            Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
            cipher.init(Cipher.ENCRYPT_MODE, buildSecretKey(secretKeyString));

            return
Base64.getEncoder().encodeToString(cipher.doFinal(stringToEncrypt.getBytes(Stand
ardCharsets.UTF_8)));
        } catch (Exception ex) {
            LOGGER.error("Error occurs during encryption of data.");
            throw new CryptoException();
        }
    }

    @Override
    public String decrypt(@NotBlank String stringToDecrypt, String
secretKeyString) throws CryptoException {

        try {
            Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5PADDING");
            cipher.init(Cipher.DECRYPT_MODE, buildSecretKey(secretKeyString));

            return new
String(cipher.doFinal(Base64.getDecoder().decode(stringToDecrypt)));
        } catch (Exception ex) {
            LOGGER.error("Error occurs during decryption of data.");
            throw new CryptoException();
        }
    }

    private SecretKey buildSecretKey(String secretKeyString) throws

```

```

NoSuchAlgorithmException {
    byte[] key = secretKeyString.getBytes(StandardCharsets.UTF_8);
    MessageDigest sha = MessageDigest.getInstance("SHA-256");
    key = sha.digest(key);
    key = Arrays.copyOf(key, 16);

    return new SecretKeySpec(key, "AES");
}
}
public interface CryptoService {

    String encrypt(String stringToEncrypt, String secretKeyString) throws
    CryptoException;

    String decrypt(String stringToDecrypt, String secretKeyString) throws
    CryptoException;
}

```

Лістинг коду клієнтської частини системи

```

export class CountryService {
    constructor(private request: RequestService) {}

    getCountries() {
        return this.request.get('countries');
    }
}
export class DocumentService {
    constructor(private request: RequestService) {

    }

    getSupplierDocumentTypes() {
        return this.request.get('documents/suppliers/documenttypes', true);
    }

    getParkingDocumentTypes() {
        return this.request.get('documents/parkings/documenttypes', true);
    }

    addParkingDocument(formData, reference) {
        return this.request.postFileUpload('documents/parkings/' + reference,
        formData);
    }

    addSupplierDocument(formData) {
        return this.request.postFileUpload('documents/supplier', formData);
    }

    deleteSupplierDocument(reference) {
        return this.request.delete('documents/' + reference + '/supplier');
    }

    deleteParkingDocument(parkingReference, documentReference) {
        return this.request.delete('documents/' + documentReference + '/parkings/' +
        parkingReference);
    }
}
export class NotificationService {
    actions = new Map([
        ['login', 'Авторизація'],
        ['register', 'Реєстрація'],
        ['updateAccountSupp', 'Оновлення персональної інформації'],
        ['createParking', 'Створення парковки'],
        ['updateParking', 'Оновлення інформації парковки'],
        ['deleteParking', 'Видалення парковки'],
    ])
}

```

```

        ['deleteParkingPlace', 'Видалення паркомісця'],
        ['uploadFile', 'Завантаження документа'],
        ['deleteSupplierDocument', 'Видалення документа'],
        ['updateAccountCust', 'Оновлення персональної інформації'],
        ['addCard', 'Додавання карти'],
        ['deleteCard', 'Видалення карти'],
        ['penaltyPay', 'Оплата штрафу']
    ]);
    messagesError = new Map([
        ['EPS003', 'Вказана пошта має некоректний формат.'],
        ['EPS004', 'Вказана пошта вже використовується.'],
        ['EPS005', 'Введений номер телефону має некоректний формат.'],
        ['EPS006', 'Вказаного користувача не існує.'],
        ['EPS007', 'Неправильний логін чи пароль.'],
        ['EPS008', 'Вказаної парковки не існує.'],
        ['EPS009', 'Паркомісце вже існує. Парковка повинна містити унікальні паркомісця.'],
        ['EPS010', 'За вказаною адресою парковка вже зареєстрована.'],
        ['EPS012', 'Парковка не може бути видалена, так як існують активні бронювання.'],
        ['EPS013', 'Паркомісце не може бути видалено, так як існують активні бронювання.'],
        ['EPS014', 'Документ не завантажено.'],
        ['EPS015', 'Обраний тип документа не існує.'],
        ['EPS018', 'Сервіс тимчасово недоступний.'],
        ['EPS019', 'Номер кредитної карти має неправильний формат.'],
        ['EPS020', 'Cvv код некоректний.'],
        ['EPS020', 'Cvv код некоректний.'],
        ['EPS022', 'Вказана адреса не існує.'],
        ['EPS023', 'Документ не існує.'],
        ['EPS024', 'Документ не існує.'],
        ['EPS029', 'Обране місце було заброньовано. Будь-ласка, повторіть ваші дії.'],
        ['EPS031', 'Замовлення не існує.'],
        ['100', 'Перевищений розмір файлу у 5 Mb']
    ]);
    messageSuccess = new Map([
        ['100', 'Парковка успішно створена'],
        ['101', 'Інформація про парковку успішно оновлена'],
        ['102', 'Парковка успішно видалена'],
        ['103', 'Паркомісце успішно видалено'],
        ['104', 'Інформацію успішно оновлено'],
        ['105', 'Документ успішно завантажено'],
        ['106', 'Документ успішно видалено'],
        ['107', 'Персональна інформація успішно оновлена'],
        ['108', 'Карту успішно додано'],
        ['109', 'Карту успішно видалено'],
        ['110', 'Штраф оплачений успішно']
    ]);

    constructor() {
    }

    getTitle(action): string {
        return this.actions.get(action);
    }

    getMessage(code, type): string {
        return (type === 'error') ? this.messagesError.get(code) :
        this.messageSuccess.get(code);
    }
}

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);
}
}
export class RequestService {

  constructor(private http: HttpClient, private cookie: CookieService) { }

  post(url, body) {
    const headers = new HttpHeaders().set('Authorization', `Bearer
${this.cookie.get('token')}`);
    return this.http.post(`${environment.apiUrl}${url}`, body, {headers});
  }

  postFileUpload(url, body){
    const headers = new HttpHeaders().set('Authorization', `Bearer
${this.cookie.get('token')}`);
    const request = new HttpRequest('POST', `${environment.apiUrl}${url}`, body,
{
      headers: headers,
      reportProgress: true,
      responseType: 'json'
    });
    return this.http.request(request);
  }

  get(url, isheaders = false, query = null) {
    let headers = null;

    if (isheaders && this.cookie.check('token')) {
      headers = new HttpHeaders().set('Authorization', `Bearer
${this.cookie.get('token')}`)
    }

    if (query) {
      url += '?';
      for (let [key, value] of query) {
        url += '&' + key + '=' + value;
      }
    }

    return this.http.get(`${environment.apiUrl}${url}`, {headers});
  }

  put(url, body) {
    const headers = new HttpHeaders().set('Authorization', `Bearer
${this.cookie.get('token')}`);
    return this.http.put(`${environment.apiUrl}${url}`, body, {headers});
  }

  delete(url) {

```

```

    const headers = new HttpHeaders().set('Authorization', `Bearer
    ${this.cookie.get('token')}`);
    return this.http.delete(`${environment.apiUrl}${url}`, {headers});
  }

  patch(url, body){
    const headers = new HttpHeaders().set('Authorization', `Bearer
    ${this.cookie.get('token')}`);
    return this.http.patch(`${environment.apiUrl}${url}`, body, {headers});
  }
}
export class SupplierService {
  constructor(private requestService: RequestService){}

  getInfo(){
    return this.requestService.get('supplier', true);
  }

  updateInfo(data){
    const body = {
      name: data.name.value,
      vatNumber: data.vatNumber.value,
      address: {
        country: data.country.value,
        city: data.city.value,
        street: data.street.value,
        streetNumber: data.streetNumber.value
      }
    };
    return this.requestService.put('supplier', body);
  }

  updateParking(data){
    const body = {
      price: data.price.value,
      penalty: data.penalty.value,
      address: {
        country: data.country.value,
        city: data.city.value,
        street: data.street.value,
        streetNumber: data.streetNumber.value
      },
      places: []
    };
    data.parkings.forEach(place => {
      body.places.push(place.number);
    });
    if(data.reference.length > 0){
      return this.requestService.put('parkings/' + data.reference, body);
    }else {
      return this.requestService.post('parkings', body);
    }
  }

  getAllParkings(){
    return this.requestService.get('parkings', true);
  }

  getParking(reference){
    return this.requestService.get('parkings/' + reference, true);
  }

  deleteParking(reference){
    return this.requestService.delete('parkings/' + reference);
  }
}

```



```

    deletePlace(parkingReference, placeReference){
        return this.requestService.delete('parkings/' + parkingReference +
'/places/' + placeReference);
    }

    getInfoPrices(reference){
        return this.requestService.get('parkings/' + reference + "/prices", true);
    }
}

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 + " дня ";
        }
        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() + 'p ' + this.convertTime(date.getHours()) + ':' +
this.convertTime(date.getMinutes());
    }
    return '';
}
}

export class AccountComponent implements OnInit {
    editableInfo: boolean = false;
    infoForm: FormGroup;
    documentForm: FormGroup;
    isSubmit = false;
    typePage: string = 'account';
    typeClient: string = 'supplier';
    countries;
    documentTypes;
    progress = 0;

    info = {
        name: {value: '', error: false},
        vatNumber: {value: '', error: false},
        email: {value: '', error: false},
        password: {value: '', error: false},
        country: {value: '', error: false},
        city: {value: '', error: false},
        street: {value: '', error: false},
        streetNumber: {value: '', error: false},
        documents: []
    };
};

    componentRef: any;
    @ViewChild('viewContainerRefAccount', {read: ViewContainerRef, static: false})
    entry: ViewContainerRef;

    constructor(
        private formBuilder: FormBuilder,
        private router: Router,
        private cookie: CookieService,
        private supplierService: SupplierService,
        private countryService: CountryService,
        private resolver: ComponentFactoryResolver,
        private title: Title,
        private documentService: DocumentService,
        private http: HttpClient
    ) {
    }

    ngOnInit() {
        if (!this.cookie.check('token')) {
            window.location.href = environment.siteUrl
        }
        this.title.setTitle("Персональна інформація");
    }

```

```

    this.getInfo();
    this.getCountry();
    this.infoForm = this.formBuilder.group({
      name: ['', [Validators.required, Validators.minLength(4)]],
      vat_number: ['', [Validators.required, Validators.minLength(8),
Validators.maxLength(8)]]
    });
    this.documentForm = new FormGroup({
      type: new FormControl('', [Validators.required, Validators.minLength(3)]),
      file: new FormControl('', [Validators.required]),
      fileSource: new FormControl('', [Validators.required])
    });
  }

  get f() {
    return this.infoForm.controls;
  }

  getInfo() {
    const promise = new Promise((resolve, reject) => {
      this.supplierService.getInfo().toPromise()
        .then((res: any) => {
          console.log(res);
          this.info.name.value = res.name;
          this.info.vatNumber.value = res.vatNumber;
          this.info.country.value = res.address.country;
          this.info.city.value = res.address.city;
          this.info.street.value = res.address.street;
          this.info.streetNumber.value = res.address.streetNumber;
          this.info.documents = res.documents;
          this.getDocumentTypes();
          resolve();
        },
        err => {
          // Error
          reject(err);
        }
      );
    });
  }

  getDocumentTypes() {
    const promise = new Promise((resolve, reject) => {
      this.documentService.getSupplierDocumentTypes().toPromise()
        .then((res: any) => {
          console.log(res);
          this.setDocumentType(res);

          resolve();
        },
        err => {
          // Error
          reject(err);
        }
      );
    });
  }

  setDocumentType(documentTypes) {
    documentTypes.forEach(type => {
      type.show = true;
      this.info.documents.forEach(document => {
        if(document.documentType.code == type.code) {
          type.show = false;
        }
      })
    })
  }

```

```

    })
  });
  this.documentTypes = documentTypes;
}

setStatusTypeDocument(code, status){
  this.documentTypes.forEach(type => {
    if(type.code == code){
      type.show = status
    }
  });
}

getCountry() {
  const promise = new Promise((resolve, reject) => {
    this.countryService.getCountries().toPromise()
      .then((res: any) => {
        this.countries = res;
        resolve();
      },
      err => {
        // Error
        reject(err);
      }
    );
  });
}

selectFile(event): void {
  if (event.target.files.length > 0) {
    const file = event.target.files[0];
    this.documentForm.patchValue({
      fileSource: file
    });
  }
}

addDocument(): void {
  const formData = new FormData();
  console.log(this.documentForm.get('fileSource').value);
  if(!this.documentForm.get('fileSource').value){
    return;
  }
  let filesize = this.documentForm.get('fileSource').value.size;

  if (filesize < environment.maxFileSize) {

    let documentType = this.documentForm.get('type').value;
    if (documentType.length == 0) {
      return;
      documentType = this.documentTypes[0].code;
    }

    formData.append('supplierDocument',
this.documentForm.get('fileSource').value);
    formData.append('documentTypeCode', documentType);
    $('.progress').show();

    this.progress = 0;

    this.documentService.addSupplierDocument(formData).subscribe(
      event => {
        if (event.type === HttpEventType.UploadProgress) {
          this.progress = Math.round(100 * event.loaded / event.total);
        } else if (event instanceof HttpResponse) {

```

```

        let document = event.body;
        this.info.documents.push({
            reference: document['reference'],
            documentType: {name: document['documentType'].name},
            name: document['name'],
            status: document['status']
        });

        this.documentForm.reset();
        $('.progress').hide();
        this.setNotification('success', 'uploadFile', '105');
        this.setStatusTypeDocument(documentType, false);
    }
},
err => {
    this.setNotification('error', 'uploadFile', err['error'][0].code);
});
} else {
    this.setNotification('error', 'uploadFile', '100');
}
}

deleteDocument(reference, index) {
    const promise = new Promise((resolve, reject) => {
        this.documentService.deleteSupplierDocument(reference).toPromise()
            .then((res: any) => {
                this.setNotification('success', 'deleteSupplierDocument', '106');
                this.info.documents.splice(index, 1);
                this.documentTypes[index].show = true;
                resolve();
            },
            err => {
                // Error
                if (err['error'].length) {
                    this.setNotification('error', 'deleteSupplierDocument',
err['error'][0].code);
                }
                reject(err);
            }
        ).catch();
    });
}

updateInfo(): void {
    this.isSubmit = true;
    if (this.validation()) {
        this.startLoader();
        const promise = new Promise((resolve, reject) => {
            this.supplierService.updateInfo(this.info).toPromise()
                .then((res: any) => {
                    this.setNotification('success', 'updateAccountSupp', 104);
                    this.endLoader();
                    resolve();
                },
                err => {
                    if (err['error'].length) {
                        this.setNotification('error', 'updateAccountSupp',
err['error'][0].code);
                    }
                    this.endLoader();
                    reject(err);
                }
            ).catch();
        });
    }
}

```

```

    }

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

    validation(): boolean {
        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.name.value.length < 4) {
            this.info.name.error = true;
            error = true;
        } else {
            this.info.name.error = false;
        }

        if (this.info.vatNumber.value.length != 10 &&
this.info.vatNumber.value.length != 8) {
            this.info.vatNumber.error = true;
            error = true;
        } else {
            this.info.vatNumber.error = false;
        }

        if (this.info.city.value.length < 2) {
            this.info.city.error = true;
            error = true;
        } else {
            this.info.city.error = false;
        }

        if (this.info.street.value.length < 2) {
            this.info.street.error = true;
            error = true;
        } else {
            this.info.street.error = false;
        }

        if (this.info.streetNumber.value.length < 1) {
            this.info.streetNumber.error = true;
            error = true;
        } else {
            this.info.streetNumber.error = false;
        }
        return !error;
    }

    startLoader() {
        $('#saveInfo').hide();
        $('.block-loader').show();
    }

    endLoader() {
        $('.block-loader').hide();
        $('#saveInfo').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);
    }
}
export class MyParkingComponent implements OnInit {
    componentRef: any;
    @ViewChild('viewContainer', {read: ViewContainerRef, static: false}) entry:
ViewContainerRef;
    typePage: string = 'myParking';
    typeClient: string = 'supplier';

    documentForm: FormGroup;
    countries;
    arrayCountries = new Map<string, string>();
    uploader: FileUploader;
    hasBaseDropZoneOver: boolean;
    hasAnotherDropZoneOver: boolean;
    response: string;
    parkingPlace: string = '';
    allParking: [];
    documentTypes;
    progress = 0;
    limitDocuments = 1;
    disableSave: Boolean = true;

    parkingInfoBeforeChanges = {
        price: '',
        penalty: '',
        country: '',
        city: '',
        street: '',
        streetNumber: ''
    };

    parkingInfo = {
        reference: '',
        price: {value: '', error: false},
        penalty: {value: '', error: false},
        country: {value: '', error: false},
        city: {value: '', error: false},
        street: {value: '', error: false},
        streetNumber: {value: '', error: false},
        status: '',
        documents: [],
        parkings: []
    };

    constructor(
        private router: Router,
        private cookie: CookieService,
        private countryService: CountryService,
        private supplierService: SupplierService,
        private resolver: ComponentFactoryResolver,
        private title: Title,
        private documentService: DocumentService,
        private http: HttpClient
    ) {
        if (!this.cookie.check('token')) {
            window.location.href = environment.siteUrl
        }
    }
}

```

```

    this.title.setTitle("Мои парковки");
    this.uploader = new FileUploader({
      url: '',
      disableMultipart: true, // 'DisableMultipart' must be 'true' for
formatDataFunction to be called.
      formatDataFunctionIsAsync: true,
      formatDataFunction: async (item) => {
        return new Promise((resolve, reject) => {
          resolve({
            name: item._file.name,
            length: item._file.size,
            contentType: item._file.type,
            date: new Date()
          });
        });
      }
    });
    this.hasBaseDropZoneOver = false;
    this.hasAnotherDropZoneOver = false;

    this.response = '';

    this.uploader.response.subscribe(res => this.response = res);
  }

  ngOnInit() {

    this.getCountry();
    this.getAllParkings();
    this.getDocumentTypes();

    this.documentForm = new FormGroup({
      type: new FormControl('', [Validators.required, Validators.minLength(3)]),
      file: new FormControl('', [Validators.required]),
      fileSource: new FormControl('', [Validators.required])
    });
  }

  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);
        }
      );
    });
  }

  getAllParkings() {
    const promise = new Promise((resolve, reject) => {
      this.supplierService.getAllParkings().toPromise()
        .then((res: any) => {
          $('.loader').hide();
          this.allParking = res;
          resolve();
        },
        err => {
          // Error
          reject(err);
        }
      );
    });
  }

```



```

    },
    err => {
      // Error
      reject(err);
    }
  );
});
}

getDocumentTypes() {
  const promise = new Promise((resolve, reject) => {
    this.documentService.getParkingDocumentTypes().toPromise()
      .then((res: any) => {
        this.documentTypes = res;
        resolve();
      },
      err => {
        // Error
        reject(err);
      }
    );
  });
}

getParking(reference) {
  const promise = new Promise((resolve, reject) => {
    this.supplierService.getParking(reference).toPromise()
      .then((res: any) => {

        this.parkingInfo.reference = res.reference;
        this.parkingInfo.price.value = this.parkingInfoBeforeChanges.price =
res.price;
        this.parkingInfo.penalty.value =
this.parkingInfoBeforeChanges.penalty = res.penalty;
        this.parkingInfo.country.value =
this.parkingInfoBeforeChanges.country = res.address.country;
        this.parkingInfo.city.value = this.parkingInfoBeforeChanges.city =
res.address.city;
        this.parkingInfo.street.value = this.parkingInfoBeforeChanges.street
= res.address.street;
        this.parkingInfo.streetNumber.value =
this.parkingInfoBeforeChanges.streetNumber = res.address.streetNumber;
        this.parkingInfo.documents = res.documents;
        res.places.sort(function (place1, place2) {
          const A = place1.number;
          const B = place2.number;
          let comparison = 0;

          if (A > B) {
            comparison = 1;
          } else if (A < B) {
            comparison = -1;
          }
          return comparison;
        });
        this.parkingInfo.parkings = res.places;
        this.parkingInfo.status = res.status;
        resolve();
      },
      err => {
        // Error
        reject(err);
      }
    );
  });
}

```

```

    }

    checkChangesForm() {
        this.disableSave = true;
        if (
            this.parkingInfoBeforeChanges.price != this.parkingInfo.price.value ||
            this.parkingInfoBeforeChanges.penalty != this.parkingInfo.penalty.value ||
            this.parkingInfoBeforeChanges.streetNumber !=
this.parkingInfo.streetNumber.value ||
            this.parkingInfoBeforeChanges.street != this.parkingInfo.street.value ||
            this.parkingInfoBeforeChanges.city != this.parkingInfo.city.value ||
            this.parkingInfoBeforeChanges.country != this.parkingInfo.country.value
        ) {
            this.disableSave = false;
        }
    }

    addPlace(): void {
        if (this.parkingPlace.length > 0) {
            let find = false;
            this.parkingInfo.parkings.forEach(place => {
                if (place.number === this.parkingPlace) {
                    find = true;
                }
            });

            if (!find) {
                this.parkingInfo.parkings.push({
                    reference: this.parkingInfo.parkings.length,
                    number: this.parkingPlace,
                    status: 'Вільне'
                });
                this.parkingPlace = '';
            }
        }
        this.disableSave = false;
    }

    selectFile(event): void {
        if (event.target.files.length > 0) {
            const file = event.target.files[0];
            this.documentForm.patchValue({
                fileSource: file
            });
        }
    }

    addDocument(reference): void {
        const formData = new FormData();
        let filesize = this.documentForm.get('fileSource').value.size;

        if (filesize < environment.maxFileSize) {

            let documentType = this.documentForm.get('type').value;
            if (documentType.length == 0) {
                documentType = this.documentTypes[0].code;
            }

            formData.append('parkingDocument',
this.documentForm.get('fileSource').value);
            formData.append('documentTypeCode', documentType);
            $('.progress').show();

            this.progress = 0;
        }
    }

```

```

    this.documentService.addParkingDocument(formData, reference).subscribe(
      event => {
        if (event.type === HttpEventType.UploadProgress) {
          this.progress = Math.round(100 * event.loaded / event.total);
        } else if (event instanceof HttpResponse) {

          let document = event.body;
          this.parkingInfo.documents.push({
            reference: document['reference'],
            documentType: {name: document['documentType'].name},
            name: document['name'],
            status: document['status']
          });
          this.documentForm.reset();
          $('.progress').hide();
          this.setNotification('success', 'uploadFile', '105');
        }
      },
      err => {
        this.setNotification('error', 'uploadFile', err['error'][0].code);
      });
  } else {
    this.setNotification('error', 'uploadFile', '100');
  }
}

deleteDocument(parkingReference, documentReference, index) {
  const promise = new Promise((resolve, reject) => {
    this.documentService.deleteParkingDocument(parkingReference,
documentReference).toPromise()
      .then((res: any) => {
        this.setNotification('success', 'deleteSupplierDocument', '106');
        this.parkingInfo.documents.splice(index, 1);
        resolve();
      },
      err => {
        // Error
        if (err['error'].length) {
          this.setNotification('error', 'deleteSupplierDocument',
err['error'][0].code);
        }
        reject(err);
      })
      .catch();
  });
}

updateParkingInfo() {
  if (this.validation()) {
    const promise = new Promise((resolve, reject) => {
      this.supplierService.updateParking(this.parkingInfo).toPromise()
        .then((res: any) => {
          if (this.parkingInfo.reference.length) {
            this.setNotification('success', 'updateParking', '101');
          } else {
            this.setNotification('success', 'createParking', '100');
          }
          //
          $('.close-modal').click();
          this.getAllParkings();
          resolve();
        },
        err => {
          // Error
          if (err['error'].length) {

```

```

        if (this.parkingInfo.reference.length) {
            this.setNotification('error', 'updateParking',
err['error'][0].code);
        } else {
            this.setNotification('error', 'createParking',
err['error'][0].code);
        }
    }
    reject(err);
}
).catch();
});
}

deleteParking(reference) {
    const promise = new Promise((resolve, reject) => {
        this.supplierService.deleteParking(reference).toPromise()
        .then((res: any) => {
            this.setNotification('success', 'deleteParking', '102');
            this.getAllParkings();
            resolve();
        },
        err => {
            // Error
            if (err['error'].length) {
                this.setNotification('error', 'deleteParking',
err['error'][0].code);
            }
            reject(err);
        }
        ).catch();
    });
}

deletePlace(parkingReference, placeReference) {

    if (!isNumber(placeReference)) {
        const promise = new Promise((resolve, reject) => {
            this.supplierService.deletePlace(parkingReference,
placeReference).toPromise()
            .then((res: any) => {
                this.setNotification('success', 'deleteParkingPlace', '103');
                this.deletePlaceFront(placeReference);
                resolve();
            },
            err => {
                // Error
                reject(err);
            }
            );
        });
    } else {
        this.deletePlaceFront(placeReference);
    }
}

deletePlaceFront(placeRef) {
    this.parkingInfo.parkings.forEach((place, index) => {
        if (place.reference === placeRef) {
            this.parkingInfo.parkings.splice(index, 1);
            if (isNumber(placeRef)) {
                this.setNotification('success', 'deleteParkingPlace', '103');
                return true;
            }
        }
    });
}

```

```

    }
  });
}

validation(): boolean {
  let error = false;

  if (this.parkingInfo.country.value.length == 0) {
    this.parkingInfo.country.error = true;
    error = true;
  } else {
    this.parkingInfo.country.error = false;
  }

  if (this.parkingInfo.city.value.length < 2) {
    this.parkingInfo.city.error = true;
    error = true;
  } else {
    this.parkingInfo.city.error = false;
  }

  if (this.parkingInfo.street.value.length < 2) {
    this.parkingInfo.street.error = true;
    error = true;
  } else {
    this.parkingInfo.street.error = false;
  }

  if (this.parkingInfo.streetNumber.value.length < 1) {
    this.parkingInfo.streetNumber.error = true;
    error = true;
  } else {
    this.parkingInfo.streetNumber.error = false;
  }

  if (this.parkingInfo.price.value.length === 0) {
    this.parkingInfo.price.error = true;
    error = true;
  } else {
    this.parkingInfo.price.error = false;
  }

  if (this.parkingInfo.penalty.value.length === 0) {
    this.parkingInfo.penalty.error = true;
    error = true;
  } else {
    this.parkingInfo.penalty.error = false;
  }

  return !error;
}

clearParkingForm() {
  this.parkingInfo = {
    reference: '',
    price: {value: '', error: false},
    penalty: {value: '', error: false},
    country: {value: '', error: false},
    city: {value: '', error: false},
    street: {value: '', error: false},
    streetNumber: {value: '', error: false},
    status: '',
    documents: [],
    parkings: []
  };
}

```

```

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);
}

}

export class OrderComponent implements OnInit {
    typePage: string = 'orders';
    typeClient: string = 'supplier';
    arrayCountries = new Map<string, string>();
    countries;
    currentDate: Date = new Date(Date.now());
    pageSizes = [5, 15, 25, 50];
    filters = {
        reference: '',
        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[] = [];

    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.getAllParkings();
    }

    getCountry() {
        const promise = new Promise((resolve, reject) => {

```

```

        this.countryService.getCountries().toPromise()
        .then((res: any) => {
            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();
}

getAllParkings() {
    const promise = new Promise((resolve, reject) => {
        this.supplierService.getAllParkings().toPromise()
        .then((res: any) => {
            this.allParking = res;
            if (!res[0].length) {
                this.filters.reference = res[0].reference;
                this.getOrders();
            }
            resolve();
        },
        err => {
            // Error
            reject(err);
        }
    );
});
}

getOrders() {
    const promise = new Promise((resolve, reject) => {
        this.orderService.getOrders(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) {
        orders.orders.forEach(order => {
            order.orderItems.forEach(orderItem => {
                orderItem.duration =
this.timeService.calculateTimeDuration(orderItem.from, orderItem.to);

```

```

        if(orderItem.depart != null){
            orderItem.realDuration =
this.timeService.calculateTimeDuration(orderItem.from, orderItem.depart);
            orderItem.depart = this.timeService.dateToString(orderItem.depart);
        }else{
            orderItem.depart = 'Не виїхав';
            orderItem.realDuration = '';
        }

        orderItem.from = this.timeService.dateToString(orderItem.from);
        orderItem.to = this.timeService.dateToString(orderItem.to);
    });
});
}
this.orders = orders;
}

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({
                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();
    }
}

export class PriceManagingComponent implements OnInit {
    typePage: string = 'priceManaging';
    typeClient: string = 'supplier';
    arrayCountries = new Map<string, string>();
    countries;
    allParking: [];
    initializedParking = new Map<string, boolean>();
    constructor(
        private countryService: CountryService,
        private supplierService: SupplierService,
        private cookie: CookieService,
        private title: Title
    ) {
        this.title.setTitle("Керування цінами");
    }

    ngOnInit() {
        if(!this.cookie.check('token')){
            window.location.href = environment.siteUrl
        }
        this.getCountry();
        this.getAllParkings();
    }

    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);
                }
            );
        });
    }

    getAllParkings(){
        const promise = new Promise((resolve, reject) => {
            this.supplierService.getAllParkings().toPromise()
                .then((res: any) => {
                    this.allParking = res;
                    console.log(this.allParking);
                    $(".loader").hide();
                    resolve();
                },
                err => {
                    // Error
                    reject(err);
                }
            );
        });
    }
}

```

```

    }

    getPricesParking(reference, number){

        if(!this.inizializedParking.has(number) ||
!this.inizializedParking.get(number)) {
            this.showParkingLoader(number);
            const promise = new Promise((resolve, reject) => {
                this.supplierService.getInfoPrices(reference).toPromise()
                    .then((res: any) => {
                        console.log(res);
                        this.initChar(res.prices, number);
                        resolve();
                    },
                    err => {
                        // Error
                        reject(err);
                    }
                );
            });
            this.inizializedParking.set(number, true);
        }else{
            this.inizializedParking.set(number, false);
        }
    }

    initChar(prices, number){
        this.hideParkingLoader(number);
        let dataPoints = [];
        let y = 0,value, startDate, dateObj ;
        prices.forEach((price) => {
            value = price.price;
            dateObj = new Date(price.startDate);
            startDate = dateObj.getFullYear()+'-' + (dateObj.getMonth()+1) + '-'
'+dateObj.getDate();
            dataPoints.push({y: value, label:startDate});
        });

        let chart = new CanvasJS.Chart("chartContainer"+number, {
            width:876,
            height:300,
            zoomEnabled: true,
            animationEnabled: true,
            exportEnabled: true,
            title: {
                text: ""
            },
            subtitles:[{
                text: ""
            }],
            data: [
                {
                    type: "line",
                    dataPoints: dataPoints
                }
            ]
        });
        chart.render();

    };

    showParkingLoader(number){
        $('#item'+number).find('.loader-graph').show();
        $('#chartContainer'+number).hide();
    }
}

```

```
hideParkingLoader(number) {  
    $("#item"+number).find('.loader-graph').hide();  
    $('#chartContainer'+number).show();  
}  
}
```