НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Кафедра автоматизованих систем обробки інформації і управління

УДК: 656.13

До захисту допущено:
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (вл.ім'я, прізвище)
"___"_____2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інформаційні управляючі системи та технології»
спеціальності 122 «Комп'ютерні науки та інформаційні технології»

на тему: « *Інформаційна система підтримки процесу автомобільних паркувань» (комплексна тема)*
Загальна частина

Виконали: студент *IV* курсу, групи *ІС-63*
Іщенко Владислав Сергійович _____
(прізвище, ім'я, по батькові) (підпис)
студент *IV* курсу, групи *ІС-63*
Левчук Володимир Іванович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник *доц., к.т.н. Сперкач Майя Олегівна* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Консультант з графічної документації *ст.викл. Проскура Світлана Леонідівна* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Рецензент *доц., к.т.н., доц. Писаренко Андрій Володимирович* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Засвідчуємо, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студенти _____ _____
(підпис) (підпис)

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) *інформатики та обчислювальної техніки*
(повна назва)

Кафедра *автоматизованих систем обробки інформації і управління*
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки та інформаційні технології»

Освітньо-професійна програма «Інформаційні управляючі системи та технології»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (вл.ім’я, прізвище)
“___”_____2020 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЄКТ СТУДЕНТАМ

Іщенку Владиславу Сергійовичу
(прізвище, ім’я, по батькові)

Левчуку Володимиру Івановичу
(прізвище, ім’я, по батькові)

1. Тема проєкту «Інформаційна система підтримки процесу автомобільних паркувань»
керівник проєкту *Сперкач Майя Олегівна, к.т.н., доцент*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “07” травня 2020 р. №1081-с
2. Термін подання студентом проєкту “*01*”*червня* 2020 року
3. Вихідні дані до проєкту
Технічне завдання
4. Зміст пояснювальної записки
1. Загальні положення: основні визначення та терміни, опис предметного середовища, огляд існуючих аналогів, постановка задачі
2.Інформаційне забезпечення: вхідні дані, вихідні дані, опис структури бази даних
3.Програмне та технічне забезпечення: засоби розробки, вимоги до технічного забезпечення, архітектура програмного забезечення
4.Технологічний розділ: керівництво користувача, випробування програмного продукту

5. Перелік графічного матеріалу

Схема структурна діяльності

Схема структурна варіантів використання

Схема бази даних

Схема структурна пакетів

Схема структурна розгортання

Схема структурна компонентів

Креслення вигляду екранних форм

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання *«1» грудня 2019 року*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	*Вивчення рекомендованої літератури*	*15.12.2019*	
2.	*Аналіз існуючих методів розв'язання задачі*	*30.12.2019*	
3.	*Постановка та формалізація задачі*	*10.01.2020*	
4.	*Розробка інформаційного забезпечення*	*20.01.2020*	
5.	*Алгоритмізація задачі*	*10.02.2020*	
6.	*Обґрунтування використовуваних технічних засобів*	*15.02.2020*	
7.	*Розробка програмного забезпечення*	*07.04.2020*	
8.	*Налагодження програми*	*20.04.2020*	
9.	*Виконання графічних документів*	*30.04.2020*	
10.	*Оформлення пояснювальної записки*	*13.05.2020*	
11.	*Подання ДП на попередній захист*	*15.05.2020*	
12.	*Подання ДП на основний захист*	*01.06.2020*	
13.	*Подання ДП рецензенту*	*03.06.2020*	

Студент _____ Владислав ІЩЕНКО
(підпис)

Студент _____ Володимир ЛЕВЧУК
(підпис)

Керівник проєкту _____ Майя СПЕРКАЧ
(підпис)

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	ДП 6310.00.000 ПЗ	Пояснювальна записка. Загальна частина	69	
3	А4	ДП 6310.00.001 ПЗ	Пояснювальна записка. Індивідуальна частина №1	98	
4	А4	ДП 6313.00.002 ПЗ	Пояснювальна записка. Індивідуальна частина №2	87	
5	А4	ДП 6310.01.000 ТЗ	Технічне завдання	11	
			Графічний матеріал:		
			Загальна частина		
6	А4	ДП 6310.02.000 ССД	Схема структурна діяльності	4	
7	А3	ДП 6310.03.000 ССВ	Схема структурна варіантів використання	1	
8	А3	ДП 6310.04.000 СБД	Схема бази даних	1	
9	А3	ДП 6310.05.000 ССП	Схема структурна пакетів	1	
10	А4	ДП 6310.06.000 ССР	Схема структурна розгортання	1	
11	А4	ДП 6310.07.000 ССК	Схема структурна компонентів	1	
12	А3	ДП 6310.08.000 КЕ	Креслення вигляду екранних форм	1	
			Індивідуальна частина №1		
13	А3	ДП 6310.09.000 ССП	Схема структурна послідовності	1	
14	А3	ДП 6310.10.000 ССС	Схема структурна станів системи	1	
15	А3	ДП 6310.11.000 КЕ	Креслення вигляду екранних форм	1	
			Індивідуальна частина №2		
16	А3	ДП 6313.12.000 ССП	Схема структурна послідовності	1	
17	А3	ДП 6313.13.000 ССС	Схема структурна станів системи	1	
18	А3	ДП 6313.14.000 КЕ	Креслення вигляду екранних форм	1	

Зм.	Арк.	ПІБ	Підп.	Дата	ДП 6310 00.000.00		
Розробн.		Іщенко В.С.			Відомість дипломного проєкту	Літ.	Лист 1 / Листів 1
Розробн.		Левчук В.І.					
Керівн.		Сперкач М.О.				КПІ ім. Ігоря Сікорського Каф. АСОІУ Гр. ІС-63	
Н/контр.		Проскура С.Л.					
В.о.зав каф.		Павлов О.А.					

Пояснювальна записка
до дипломного проєкту

на тему: «Інформаційна система підтримки процесу автомобільних паркувань» (комплексна тема)

Київ – 2020 року

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка дипломного проєкту складається з чотирьох розділів, містить 8 рисунків, 7 таблиць, 1 додаток, 7 джерел.

Дипломний проєкт присвячений розробці інформаційної системи, метою якої є покращення процесу автомобільних паркувань – використання онлайн оплати, пошук найближчих парковок та ведення паркобізнесу через власний кабінет.

У розділі «Загальні положення» було описано предметне середовище, процеси діяльності. Було визначено варіанти використання та функціональні вимоги до системи. Визначено основні аналоги даної системи та проведено порівняльний аналіз. Також, було визначено призначення, цілі і задачі розробки.

У розділі «Інформаційне забезпечення» було визначено перелік вхідних та вихідних даних. Також, було описано структуру бази даних.

У розділі «Програмне та технічне забезпечення» наведено перелік технічних засобів, які використовувались під час розробки, з описом кожного. Було висунуто вимоги до технічного забезпечення де буде розгортатися та запускатися програмний продукт. Також, описана архітектура система і наведено перелік діаграм для її графічного відображення.

Технологічний розділ містить керівництво користувача для реєстрації та авторизації користувачів. Також, в цьому розділі описано процес проведення тестування системи.

БРОНЮВАННЯ, ПАРКОМІСЦЕ, ПОШУК ВІЛЬНИХ ПАРКОМІСЦЬ, ІНФОРМАЦІЙНА СИСТЕМА, БАЗА ДАНИХ

ABSTRACT

Structure and scope of work. Diploma project consists of four sections, contains 9 drawings, 7 tables, 1 application, 7 sources.

The diploma project is devoted to the development of an information system for improvment of the process of car parking – using of online payment, a search of nearest parking and running the parking business via personal cabinet.

In the section “General provision”, subject area and activity processes were described. Use cases and functional requirements of the system were defined. The main analogs of this system were found and a comparison was made. The purpose, goals, and tasks of development were also determined.

In the section “Information provision”, input and output data were defined. Also, structure of the database was described in scope of this section.

In the section “Software and hardware”, a list of main tools that were using during the development is given with a description of each. Requirements have been defined for the hardware, where information system will be deployed and launched. The architecture of the system is described and a list of diagrams for it’s graphical display is given.

The technology section contains the user’s manual for processes of registration and authorization. Also, this section describes the process of testing of the information system.

BOOKING, PARKING PLACE, SEARCH OF AVAILABLE PARKING PLACES, INFORMATION SYSTEM, DATABASE

ЗМІСТ

ВСТУП

У міру зростання кількості населення у світі, збільшується кількість транспортних засобів, адже кожен хоче бути незалежним від громадського транспорту. Власний автомобіль відкриває великі можливості для переміщення по місту і не тільки. Так як автомобілі весь час не можуть знаходитися в русі, а тільки тоді коли потрібно кудись доїхати, то вони близько 80% свого часу, десь мають стояти.

З кожним роком кількість автомобілів збільшується і проблема паркування стає все більш гострою. У великих містах вона просто критична, водії залишають власні автомобілі на тротуарах, на крайніх смугах дорожнього руху, а все через брак вільних паркомісць, чи складності їх швидкого пошуку.

Вирішенням даної проблеми є система онлайн паркувань. Ця система дозволить власникам парковок здавати в оренду нові майданчики і слідкувати за статистикою їхнього використання. Водії матимуть можливість знаходити паркомісця і здійснювати бронювання на бажаний час.

Тема даної роботи є актуальною, адже кількість автомобілів росте дуже швидко і паркомісць вистачає не завжди. Нові парковки не вигідно відкривати через складність і затратність їх ведення. Даний дипломний проєкт покликаний спростити процес ведення паркобізнесу для власників парковок і дати водіям зручний доступ до паркомісць за допомогою розроблюваної системи.

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Опис предметного середовища

Кількість автомобілів у світі постійно збільшується, а місця для їх паркування не вистачає. За даними Міжнародної асоціації автовиробників, кількість автомобілів у 2015 році складала 1.2 млрд. і 95% з них – це легкові автомобілі [1]. За прогнозами «**Navigant Research**», в найближчому майбутньому річні продажі легкових автомобілів можуть вирости до 126,9 млн. штук. При таких показниках вже до 2035 року світовий автопарк досягне 2 млрд.

Країни світи вирішують дану проблему різними способами. Візьмемо для прикладу Японію, де населення становить 126 млн. чоловік, а кількість авто на 1000 людей, за даними Вікіпедії, становить 591 одиниць [2]. Вищезгадана країна досить розвинута і будує високотехнологічні парковки. На рисунку 1.1 наведено приклад парковки в Японії.

Рисунок 1.1 – Приклад парковки в Японії

Парковка [3] – споруда, будівля (частина будинку, споруди) або спеціальний відкритий майданчик, призначений для зберігання тимчасового чи постійного транспортних засобів.

Проблема парковок дуже поширена в сучасному світі, особливо дана проблема помітна у великих містах, де концентрація автомобілів вища ніж будь-де. Якщо взяти для прикладу житлові будинки, то забудовник не завжди будує паркінги з достатньою к-тю місць. Про будинки, які побудовані до 90-х років не варто і говорити, адже вони часто не мають паркінгів взагалі.

Кожне місто України, зокрема Київ, має великі проблеми з парковками, а саме загальної кількості паркомісць просто недостатньо. Окрім того, що кількість паркомісць недостатня, існує проблема ведення паркобізнесу. Для того, щоб ефективно здавати в оренду паркомісця, необхідно мати онлайн систему для бронювань. Розробка та подальша реклама такої системи досить дорога. Рішенням цієї проблеми є розробка єдиного маркетплейсу парковок, адже даний тип системи буде працювати не з одним власником парковки. Будь-хто може додати власні парковки і отримувати дохід від їх оренди. Звісно, така система має передбачувати перевірку права на ведення даного типу бізнесу аби уникнути його неправомірного ведення і як наслідок ухилянь від податків.

Маркетплейс парковок об’єднає велику кількість парковок в одному місці, що буде дуже зручно для водіїв. Це дасть великий розвиток для мікро- і малого бізнесу, адже для запуску роботи системи, необхідно буде мати власний кабінет із відповідними доданими даними і встановленими на парковках шлагбауми, які можуть зчитувати QR коди. Цей тип шлагбаумів необхідний для фіксації в’їзду та виїзду з парковки.

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

1.1.1 Опис процесу діяльності

В даній роботі розглядається автоматизація наступних процесів:

- бронювання паркомісць;
- здача паркомісць в оренду.

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

Для здачі паркомісць в оренду процес більш складний і дорогий. Власник парковок має встановити шлагбаум при в'їзді на парковку і найняти працівника, який буде контролювати в'їзд. Відповідно даному працівнику необхідно платити, заробітну плату. Також, необхідно встановити систему, яка буде видавати талони, матиме можливість приймати готівку або інші способи оплати. Така система є досить дорогою.

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

Коли водій приїде до парковки, йому необхідно буде зайти у власний кабінет, згенерувати QR-код і зчитати його на шлагбаумі перед в'їздом. Система отримає запит на в'їзд і дозволить водію припаркуватися.

Власник парковок має авторизуватися у власному кабінеті, якщо незареєстрований, то необхідно заповнити реєстраційну форму для

комерційного акаунта. 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.

Схеми структурні діяльності до автоматизації і після, наведені в графічних матеріалах.

1.1.2 Опис функціональної моделі

Перед початком проектування діаграми використання потрібно визначити перелік акторів, що дозволить нам 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.
- **Клієнт.** Зареєстрований користувач веб-сервісу, котрий використовує його для бронювання паркомісць заздалегідь.
- **Власник парковок.** Зареєстрований користувач, котрий має на меті здачу в оренду власних паркомісць за допомогою даної системи.

Тепер визначимо дії, які можуть виконувати актори системи та відобразимо їх за допомогою схеми структурної варіантів використання.

Схема структурна варіантів використання наведена в частині графічного матеріалу.

1.2 Огляд наявних аналогів

Під час проведення пошуку аналогів були виявлені наступні системи зі схожим функціоналом: **KyivSmartCity, Barking, Parking UA** та **Privat24.**

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

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

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.

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	KyivSmartCity	Barking	ParkingUA	Privat24
Онлайн оплата	Так	Так	Так	Так	Так
Вибір парковки на карті	Ні	Ні	Так	Так	Так
Доступ до парковки по QR-коду	Так	Ні	Так	Ні	Ні
Можливість здачі в оренду власних паркомісць	Так	Ні	Ні	Ні	Ні
Пошук найближчих парковок в певному радіусі	Так	Ні	Ні	Ні	Ні

1.3 Постановка задачі

1.3.1 Призначення розробки

Інформаційна система призначена для підтримки процесу автомобільних паркувань.

1.3.2 Цілі та задачі розробки

Цілями розробки даної системи є:

- спрощення ведення паркобізнесу в умовах діджиталізації за рахунок створення загального майданчику для паркувань з великим асортиментом парковок;
- полегшення процесу пошуку та бронювання паркомісць за рахунок доступу до паркувальних майданчиків в різних локаціях та можливості онлайн бронювання паркомісць на них.

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

Висновок до розділу

У цьому розділі було обгрунтовано доцільність розробки системи підртимки автомобільних паркувань. Було описано процеси діяльності до та після автоматизації, які були відображені в структурних схемах діяльності. Були висунуті функціональні вимоги до системи, які наведені в структурній діаграмі варіантів використання.

Було сформульовано декілька цілей розробки даної системи: спрощення ведення паркобізнесу та полегшення процесу пошуку та бронювання паркомісць. Також, було побудовано перелік задач, розв'язання яких сприятиме досягнення поставлених цілей.

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

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Вхідні дані

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

- клієнтів;
- власників парковок;
- адміністраторів.

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

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

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

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

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

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

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

- номер;
- термін дії;
- CVV2 код.

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

- назву компанії чи фізичної-особи підприємця;

– адресу реєстрації;
– індивідуальний податковий номер;
– електронну пошту;
– корпоративний номер телефону.

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

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

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

– адреса розташування;
– тариф на паркування;
– розмір штрафу;
– копію технічного паспорту парковки;
– перелік номерів паркомісць.

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

2.2 Вихідні дані

Вихідними даними роботи системи є бронювання та штрафи. Бронювання представлені наступним переліком параметрів:

– адреса парковки;
– статус;
– номер паркомісця;
– час в’їзду на парковку;
– час виїзду з парковки;
– фактичний час виїзду з парковки;

– загальна сума за перебування на парковці.

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

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

2.3 Опис структури бази даних

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

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

Наведемо перелік сутностей системи в таблиці 2.1.

Таблиця 2.1 – Перелік сутностей

№	Назва таблиці	Сутність
1	Address	Адреса
2	Country	Країна
3	Credit_card	Кредитна картка клієнта
4	Customer	Клієнт
5	Document	Метадані документа
6	Document_type	Тип документа
7	Order_item	Бронювання паркомісця
8	Orders	Бронювання
9	Parking	Парковка

Продовження таблиці 2.1

№	Назва таблиці	Сутність
10	Penalty	Штраф
11	Place	Паркомісце
12	Price_history	Історія тарифів
13	Role	Роль
14	Secret_key	Секретний ключ
15	Supplier	Власник парковок
16	Users	Користувач

У таблиці 2.2 наведено детальний опис таблиць бази даних.

Таблиця 2.2 – Детальний опис таблиць бази даних

Назва таблиці	Назва стовпця	Тип даних	Опис поля
Address	id	bigint	Первинний ключ
	country_iso3	varchar(3)	Код країни в форматі ISO3
	city	varchar(255)	Місто
	street	varchar(255)	Вулиця
	street_number	varchar(255)	Номер будинку
	latitude	double precision	Широта
	longitude	double precision	Довжина
Country	iso3	varchar(3)	Код країни в форматі ISO3
	name	varchar(255)	Назва країни
Credit_card	id	bigint	Первинний ключ
	reference	varchar(255)	Унікальне посилання

Продовження таблиці 2.2

Назва таблиці	Назва стовпця	Тип даних	Опис поля
Credit_card	card_number	varchar(255)	Номер картки
	expiration_date	date	Терім закінчення дії
	cvv	varchar(4)	Захисний код
	status	varchar(50)	Статус
	owner	varchar(255)	Ім'я власника
	customer_id	bigint	Посилання на клієнта
	secret_key_id	bigint	Посилання на секретний ключ, за допомогою якого зашифровано номер картки
Customer	id	bigint	Первинний ключ
	first_name	varchar(255)	Ім'я клієнта
	last_name	varchar(255)	Прізвище клієнта
	phone_number	varchar(255)	Номер телефону
	user_id	bigint	Посилання на користувача
Document	id	bigint	Первинний ключ
	name	varchar(255)	Назва
	size	integer	Розмір
	path	varchar(255)	Шлях до документу
	status	varchar(255)	Статус
	reference	varchar(255)	Унікальне посилання

Продовження таблиці 2.2

Назва таблиці	Назва стовпця	Тип даних	Опис поля
	document_type_code	varchar(50)	Посилання на тип документа
Document_type	code	varchar(50)	Код документа
	name	varchar(255)	Повна назва типу документа
	supplier	boolean	Тип документа належить власнику парковок
	parking	boolean	Тип документа належить парковці
Order_item	id	bigint	Первинний ключ
	reference	varchar(255)	Унікальне посилання
	place_number	varchar(50)	Номер місця
	datetime_from	timestamp	Запланований час заїзду на парковку
	datetime_to	timestamp	Запланований час виїзду з парковки
	datetime_depart	timestamp	Фактичний час виїзду з парковки
	order_id	bigint	Посилання на бронювання
Orders	id	bigint	Первинний ключ

Продовження таблиці 2.2

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

Продовження таблиці 2.2

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

Продовження таблиці 2.2

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

Продовження таблиці 2.2

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

Висновок до розділу

В даному розділі було визначено декілька джерел вхідної інформації. Для кожного з них було визначено перелік вхідних та вихідних даних.

Також, було описано структуру бази даних. Вона налічує 26 таблиць, з них 19 забезпечують роботу бізнес логіки системи, а 7 – зберігання метаданих про роботу системи. Наведено перелік сутностей, котрі використовуються в системі, з вказанням таблиць, до яких вони відносяться. Також, побудовано таблицю, в котрій наведено перелік таблиць бази даних, які забезпечують робот бізнес логіки системи, з детальним описом всіх параметрів.

3 ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Засоби розробки

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

Таблиця 3.1 – Технічні засоби використанні під час розробки

Тип	Назва	Версія
Мови програмування	Java/TypeScript	8/3.5.3
Фреймворки	Spring/Angular	5/8
ORM	Hibernate	5.3
IDE	IntelliJ Idea	2020.1
RDBMS	PostgreSQL	12
Засіб проектування	PowerDesigner	16.5
Засіб керування базами даних	DataGrid	2019.3.2
Система контроля версій	Git	2.16.1
Репозиторій	GitHub	–
Хостинг	Heroku	–

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

TypeScript [4] мова програмування, яка була розроблена компанією Microsoft. Ця мова програмування є зворотньо сумісною з JavaScript. Використовується для розробки веб-застосунків і розширює можливості мови програмування JavaScript.

Фреймворк Spring [5] розроблений для мови програмування Java з метою спрощення та прискорення процесу написання веб-застосунків. Цей фреймворк підтримує інверсію управління, що значно полегшує написання програмного коду. Spring має велику кількість модулів для абсолютно різних призначень.

Angular – це фреймворк з відкритим вихідним кодом, який був розроблений на мові програмування TypeScript і підтримується компанією Google. Даний фреймворк спрощує процес побудови клієнтської частини веб застосунків.

Hibernate – даний фреймворк використовується для того, щоб відображати реляційні структури на об’єкти мови програмування Java. Він є однією з найпопулярніших реалізацій специфікації JPA. Метою даного фреймворку є звільнення розробника від ручного зв’язування таблиць бази даних та результатів виконання запитів. Він надає зручний інтерфейс для побудови SQL запитів за допомогою власної специфікації Hibernate Query Language (HQL). Перевагою цієї специфікаціє є побудова запитів в об’єктно орієнтованому стилі, що є більш зрозумілим для розробників.

Для зберігання даних використовувалась об’єктно-реляційна система керування базами даних (далі СКБД) PostgreSQL [6]. Вихідний код цієї СКБД є відкритим і за рахунок цього вона має широкий функціонал і велику кількість оптимізації, що дозволяє їй складати конкуренцію таким комерційним СКБД як Oracle, MsSQL і т.д [7]. Вона є абсолютно безкоштовною, що також надає велику перевагу.

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

Git [8] – система контроля версій, що була розроблена на базі ядра операційної системи Linux. Він дозволяє відстежувати всі зміни, що відбувалися в проекті. За допомогою даного програмного забезпечення вирішується проблема роботи в команді, адже Git дозволяє працювати окремо, а в необхідний момент часу об’єднувати всю роботу.

GitHub – віддалене сховище git-репозиторіїв. Воно дозволяє керувати доступом до проектів, обмінюватися вже виконаною роботою, відстежувати помилки в програмному забезпечення. Головною перевагою хостингу є зручний веб-інтерфейс з великою кількістю доступного функціоналу.

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

3.2 Вимоги до технічного забезпечення

3.2.1 Вимоги до серверу клієнтської частини застосунку

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

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

3.2.2 Вимоги до серверу API частини застосунку

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

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

3.2.3 Вимоги до серверу бази даних

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

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

3.2.4 Вимоги до клієнта

Клієнт повинен мати встановлений один з веб-браузерів:

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

3.3 Архітектура програмного забезпечення

Програмний продукт було спроектовано на основі архітектурного шаблону Model-View-Controller [9] (далі MVC). Цей шаблон має на меті розділення застосунку на три окремих частини: дані, користувацький інтерфейс та бізнес логіка. Він широко використовується для проектування веб-застосунків. На рисунку 3.1 наведена схема взаємодії компонентів в шаблоні MVC.

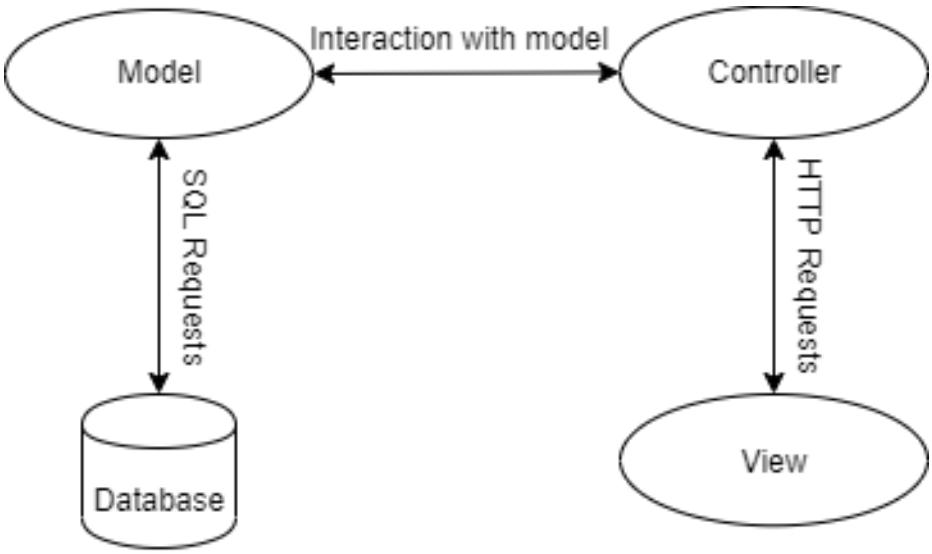


Рисунок 3.1 – Схема роботи шаблону MVC

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

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

3.3.1 Діаграма пакетів

У частині дипломного проєкту «Графічний матеріал» представлена діаграма пакетів, яка описує структуру серверної частини застосунку. Ця діаграма показує взаємодію різних пакетів проекту, що дає розуміння про його структуру. Можна помітити, що багато пакетів мають однакові імена, це свідчить про те, що при розробці програмного продукту дотримувалася конвенція іменування.

3.3.2 Діаграма розгортання

У частині дипломного проєкту «Графічний матеріал» представлена діаграма розгортання, котра 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 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.

3.3.3 Діаграма компонентів

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

3.3.4 Специфікація функцій

В таблиці 3.2 наведено перелік методів серверної частини програмного продукту з вказанням класів до яких вони належать та детальним описом.

Таблиця 3.2 – Специфікація методів серверної частини програми

Клас	Сигнатура	Опис
Application	main(String[] args)	Вхідна точка в програму при запускі на сервері
LoginController	loginUser(@RequestBody User credentials)	Автентифікація користувача
UserCredentialsAreCorrectValidator	isValid(User user, ConstraintValidatorContext constraintValidatorContext)	Перевірка існування вказаного користувача та правильності введеного пароля

Продовження таблиці 3.2

Клас	Сигнатура	Опис
UserExistValidator	isValid(String email, ConstraintValidatorContext constraintValidatorContext)	Перевірка існування користувача з вказаною електронною поштою
LoginResponseFactory	buildResponse(@NotNull UserType userType, @NotBlank String username)	Побудова відповіді на запит автентифікації з генеруванням JWT токену сесії
RegistrationController	registerCustomer(@RequestBody CustomerRequest customerRequest)	Реєстрація клієнта
	registerSupplier(@RequestBody SupplierRequest supplierRequest)	Реєстрація власника парковок
PhoneNumberFormat Validator	isValid(String number, ConstraintValidatorContext constraintValidatorContext)	Перевірка формату номера телефону
UniqueEmailValidator	isValid(String email, ConstraintValidatorContext constraintValidatorContext)	Перевірка унікальності електронної пошти
ValidEmailValidator	isValid(String email, ConstraintValidatorContext constraintValidatorContext)	Перевірка формату електронної пошти
RegistrationService	createCustomerAccount(CustomerRequest customerRequest)	Створення акаунта клієнта
	createSupplierAccount(SupplierRequest supplierRequest)	Створення акаунта власника парковок
CheckUserAspect	checkCustomer()	Перевірка прав клієнта
	checkSupplier()	Перевірка прав власника парковок

Продовження таблиці 3.2

Клас	Сигнатура	Опис
	checkAdmin()	Перевірка прав адміністратора
JwtConfigurer	configure(HttpSecurity builder)	Конфігурація фільтру безпеки
JwtTokenProvider	generateToken(String username)	Генерація JWT токену сесії
	resolveToken(HttpServletRequest request)	Перевірка токену в запиті
	validateToken(String token)	Перевірка коректності токену
UserDetailServiceImpl	loadUserByUsername(String email)	Завантаження користувача по електронній пошті
AuthenticationFilter	doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)	Фільтрація запитів, які не потребують наявності JWT токену
UserContext	getCurrentUser()	Отримання та кешування поточного користувача
	getCurrentUsername()	Отримання електронної пошти поточного користувача
UserRepository	findWithRolesByEmail(@Param("email") String email)	Знаходження користувача за електронною поштою

В таблиці 3.3 наведено перелік функцій клієнтської частини програмного продукту з вказанням класів до яких вони належать та детальним описом.

Таблиця 3.3 – Специфікація функцій клієнтської частини програми

Клас	Сигнатура	Опис
HomeComponent	initHomeComponent()	Ініціалізація головної сторінки

Продовження таблиці 3.3

Клас	Сигнатура	Опис
HeaderComponent	ngOnInit()	Ініціалізація хедеру сайту
	accountOpen()	Відкриття форми авторизація
LoginComponent	ngOnInit()	Ініціалізація форми авторизації
	setNotification(code)	Виведення повідомлення про авторизацію
	login()	Запит на авторизацію
	validation()	Валідація форми авторизації
	hideModals()	Закриття форми авторизації
	startLoader()	Запуск анімації прелоадера кнопки авторизації
	endLoader()	Закінчення анімації
NotificationComponent	ngOnInit()	Ініціалізація даними повідомлення про авторизацію
RegisterComponent	getCountry()	Отримання списку країн
	changeType(type)	Зміна типу користувача
	register()	Реєстрація користувача
	createCustomer()	Створення кабінету клієнта
	createSupplier()	Створення кабінету власника парковок
	hideModals()	Закриття форми реєстрації
	validateClient()	Валідація форми реєстрації
	setNotification(code)	Виведення повідомлень про реєстрацію

Продовження таблиці 3.3

Клас	Сигнатура	Опис
CountryService	getCountries()	Отримання списку країн
NotificationService	getTitle(action)	Отримання заголовку повідомлення
	getMessage(code, type)	Отримання повідомлення
UserService	login(email, password)	Авторизація на сайті
	createCustomer(data)	Створення кабінету клієнта
	createSupplier(data)	Створення кабінету власника парковок
RequestService	post(url, body)	Запит POST
	get(url, isheaders = false, query = null)	Запит GET

Висновок до розділу

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

Була описана архітектура застосунку і наведено її схематичне зображення. Побудовано діаграми пакетів, компонентів та розгортання. Висунуто вимоги до серверів, на яких буде розгортатися застосунок.

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

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

4 ТЕХНОЛОГІЧНИЙ РОЗДІЛ

4.1 Керівництво користувача

Авторизація користувача відбувається однаково для всіх типів користувачів системи. Для того, щоб відкрити вікно авторизації потрібно натиснути на кнопку «Авторизуватися», котра знаходиться на головній сторінці, як показано на рисунку 4.1.

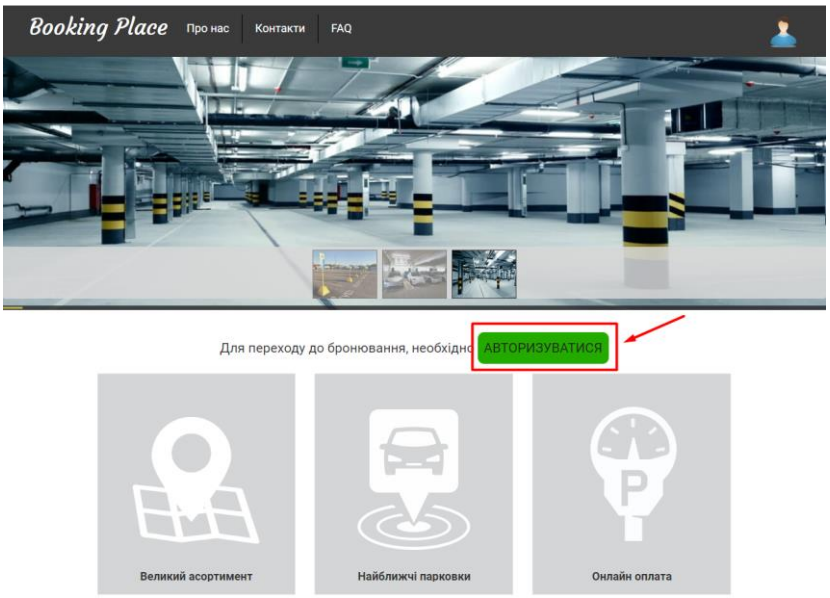


Рисунок 4.1 – Головна сторінка системи

Після того, як вікно авторизації буде відкрито, потрібно ввести електронну пошту та пароль, які були вказані при реєстрації і натиснути на кнопку «Авторизуватися». На рисунку 4.2 зображено вікно авторизації.

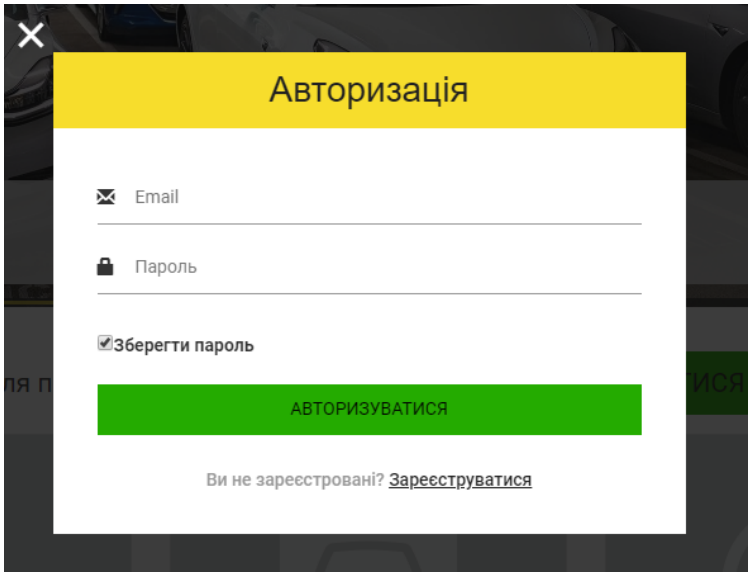


Рисунок 4.2 – Вікно авторизації

4.1.1 Власник парковок

Перед початком використання системи, власник парковок повинен створити комерційний акаунт. Для того, щоб перейти до заповнення форми реєстрації, необхідно відкрити вікно авторизації, як показано на рисунках 4.1-4.2, 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 4.3 зображено вікно реєстрації для власників парковок.

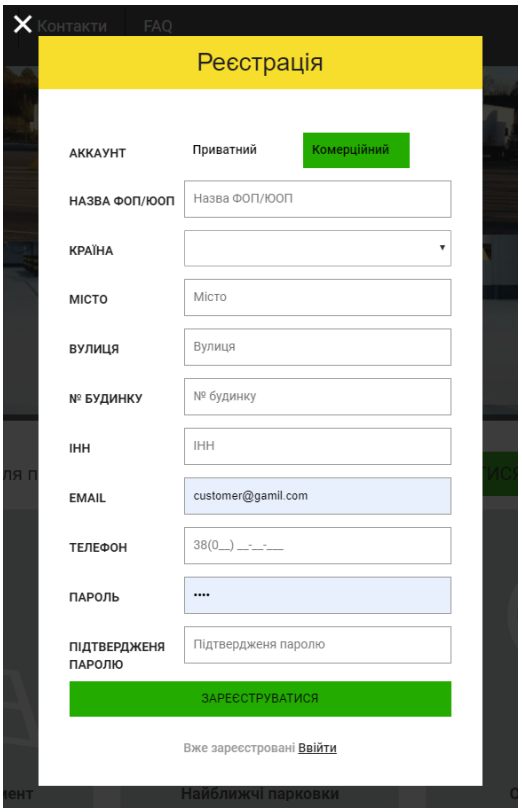


Рисунок 4.3 – Вікно реєстрації для власника парковок

Після успішної реєстрації система автоматично виконає авторизацію і відкриється власний кабінет власника парковок, як показано на рисунку 4.4.

Рисунок 4.4 – Початкова сторінка кабінету власника парковок

4.1.2 Клієнт

Для того, щоб почати роботу з системою, клієнт має пройти реєстрацію. Спочатку потрібно відкрити вікно авторизації і натиснути на посилання «Зареєструватися». У відкритому вікні необхідно обрати тип акаунта «Приватний». На рисунку 4.5 зображено вікно реєстрації для клієнтів.

Рисунок 4.5 – Вікно реєстрації для клієнтів

Після проходження процесу реєстрації, система відкриє головну сторінку клієнта, яка показана на рисунку 4.6.

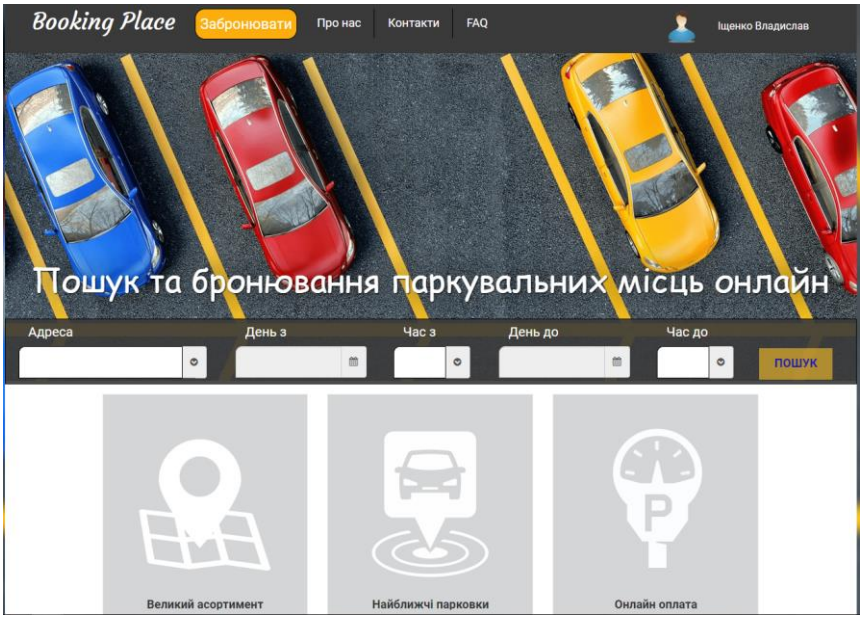


Рисунок 4.6 – Головна сторінка клієнта

4.2 Випробування програмного продукту

4.2.1 Мета випробувань

Випробування проводяться з метою перевірки функцій розробленої інформаційної системи підтримки процесу автомобільних паркувань відповідно до вимог технічного завдання.

4.2.2 Загальні положення

Випробування проводились на основі наступних документів:

- ГОСТ 34.603−92. Інформаційна технологія. Види випробувань автоматизованих систем;
- ГОСТ РД 50–34.698–90. Автоматизовані системи вимог до змісту документів.

4.2.3 Результати випробувань

В таблицях 4.1 – 4.4 наведені тести випробування застосунку.

Таблиця 4.1 – Тест випробування реєстрації клієнта в системі

Назва:	Тест реєстрації клієнта	
Функція/UseCase:	Реєстрація	
Дія: Клієнт реєструється в системі	**Очікуваний результат:** Створення акаунта клієнта і перехід на головну сторінку для авторизованого клієнта	**Результат тесту:** • пройдений • провалений • заблокований
Передумова:		
Перейти до системи за посиланням: https://booking-place.herokuapp.com/	Відкрита головна сторінка системи	Пройдений
Натиснути на кнопку «Авторизуватися»	Форма авторизації відкрита	Пройдений
Натиснути на посилання «Зареєструватися»	Форма реєстрації клієнта відкрита	Пройдений
Кроки тесту:		
Заповнити форму реєстрації коректними даними	Введена інформація відображається у відповідних полях	Пройдений
Натиснути на кнопку «Зареєструватися»	Відкрита головна сторінка для авторизованого клієнта	Пройдений
Післяумова:		
Навести курсор мишки на своє ім’я біля значка користувача	Розкривний список з посиланнями відкритий	Пройдений
Натиснути на кнопку «Вийти»	Відкрита головна сторінка системи	Пройдений

Таблиця 4.2 – Тест випробування реєстрації власника парковок в системі

Назва:	Тест реєстрації власника парковок	
Функція/UseCase:	Реєстрація	
Дія: Власник парковок реєструється в системі	**Очікуваний результат:** Створення акаунта власника парковки і перехід у власний кабінет	**Результат тесту:** • пройдений • провалений • заблокований
Передумова:		
Перейти до системи за посиланням: https://booking-place.herokuapp.com/	Відкрита головна сторінка системи	Пройдений
Натиснути на кнопку «Авторизуватися»	Форма авторизації відкрита	Пройдений
Натиснути на посилання «Зареєструватися»	Форма реєстрації відкрита	Пройдений
Натиснути на вкладку «Комерційний»	Форма реєстрації власника парковок відкрита	Пройдений
Кроки тесту:		
Заповнити форму реєстрації коректними даними	Введена інформація відображається у відповідних полях	Пройдений
Натиснути на кнопку «Зареєструватися»	Відкрита головна сторінка для авторизованого власника парковок	Пройдений

Продовження таблиці 5.2

Післяумова:		
Навести курсор мишки на своє ім’я біля значка користувача	Розкривний список з посиланнями відкритий	Пройдений
Натиснути на кнопку «Вийти»	Відкрита головна сторінка системи	Пройдений

Таблиця 5.3 – Тест випробування авторизації клієнта в системі

Назва:	Тест авторизації клієнта	
Функція/UseCase:	Авторизація	
Дія: Клієнт авторизується в системі	**Очікуваний результат:** Авторизація в системі і перехід на головну сторінку клієнта	**Результат тесту:** • пройдений • провалений • заблокований
Передумова:		
Перейти до системи за посиланням: https://booking-place.herokuapp.com/	Відкрита головна сторінка системи	Пройдений
Натиснути на кнопку «Авторизуватися»	Форма авторизації відкрита	Пройдений
Кроки тесту:		
Ввести логін та пароль, що вказувались при реєстрації	Введена інформація відображається у відповідних полях	Пройдений
Натиснути на кнопку «Авторизуватися»	Відкрита головна сторінка для авторизованого клієнта	Пройдений

Продовження таблиці 5.3

Післяумова:		
Навести курсор мишки на своє ім’я біля значка користувача	Розкривний список з посиланнями відкритий	Пройдений
Натиснути на кнопку «Вийти»	Відкрита головна сторінка системи	Пройдений

Таблиця 5.4 – Тест випробування авторизації власника парковок в системі

Назва:	Тест авторизації власника парковок	
Функція/UseCase:	Реєстрація	
Дія: Власник парковок авторизується на сайті	**Очікуваний результат:** Авторизація на сайті і перехід у власний кабінет	**Результат тесту:** • пройдений • провалений • заблокований
Передумова:		
Перейти до системи за посиланням: https://booking-place.herokuapp.com/	Відкрита головна сторінка системи	Пройдений
Натиснути на кнопку «Авторизуватися»	Форма авторизації відкрита	Пройдений
Кроки тесту:		
Ввести логін та пароль, що вказувались при реєстрації	Введена інформація відображається у відповідних полях	Пройдений

Продовження таблиці 5.4

Натиснути на кнопку «Авторизуватися»	Відкрита головна сторінка для авторизованого власника парковок	Пройдений
Післяумова:		
Навести курсор мишки на своє ім’я біля значка користувача	Розкривний список з посиланнями відкритий	Пройдений
Натиснути на кнопку «Вийти»	Відкрита головна сторінка системи	Пройдений

Висновок до розділу

В цьому розділі було розглянуто частину користувацького інтерфейсу. Побудовано керівництво користувача з проходження процесу реєстрації та авторизації для клієнта та власника парковок.

Підрозділ «Випробування програмного продукту» містить мету випробування та перелік документів, згідно яких проводилось тестування. Наведено перелік сценаріїв, по яким виконувалась перевірка коректності роботи реєстрації та авторизації. За результатами тестування можна сказати, що всі перевірки були успішно пройдені.

ЗАГАЛЬНІ ВИСНОВКИ

В рамках даного дипломного проєкту було розроблено систему підтримки процесу автомобільних паркувань, яка вирішує проблему ведення паркобізнесу власниками парковок та проблему пошуку та бронювання паркомісць клієнтами. Власники парковок можуть керувати парковками та наявними паркомісцями за допомогою власного кабінету, а для клієнтів відкривається можливість зручного пошуку вільних паркомісць та їх бронювання з онлайн оплатою.

У розділі «Загальні положення» було наведено опис предметного середовища. Було розглянуто процесу, які підлягають автоматизації і відповідно до опису були побудовані схеми структурні діяльності. Визначено перелік функціоналу, який необхідний для власників парковок та клієнтів і відповідного до нього поставлено задачі. Також, було визначено основні аналоги системи та проведено порівняльний аналіз, за допомогою якого визначено основні недоліки інших систем.

У розділі «Інформаційне забезпечення» визначено вхідні та вихідні дані для кожного типу користувача. Також, була спроектована структура бази даних, яка забезпечить функціонування системи. Наведено опис таблиць бази даних і перелік сутностей програмного продукту.

У розділі «Програмне та технічне забезпечення» було визначено технічні засоби, які необхідні для розробки та керування системою. Наведено опис архітектури програмного продукту з вказанням її переваг. Було висунуто вимоги до технічного забезпечення, на якому буде виконуватись розгортання системи та її подальше використання.

В рамках технологічного розділу було побудовано керівництво користувача для проходження процесу реєстрації та авторизації кожного з користувачів. Проведено тестування системи, в результаті якого було знайдено і усунуто деякі недоліки.

В загальні частині дипломного проєкту було спроектовано та розгорнуто базу даних, яка буде використовуватись системою. Також, була спроектована

архітектура застосунку і побудовано функціональну модель, на базі якої буде виконуватись розробка. Також, в рамках загальної частини було сконфігуровано проєкт і розроблено функціонал для реєстрації та авторизації в системі.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Цифра дня: скільки автомобілів на планеті? [Електронний ресурс] // Режим доступу: http://mmr.net.ua/autoworld/news/94527
2. Список країн за кількістю автомобілів на 1000 осіб [Електронний ресурс] // Режим доступу: /https://uk.wikipedia.org/wiki/Список_країн_за_кількістю_автомобілів_на_1000_осіб
3. Парковка [Електронний ресурс] // Режим доступу: https://slovotvir.org.ua/words/parkovka
4. Spring Framework Overview [Електронний ресурс] // Режим доступу: https://docs.spring.io/spring/docs/current/spring-framework-reference/overview.html
5. About PostgreSQL [Електронний ресурс] // Режим доступу: https://www.postgresql.org/about/
6. TypeScript [Електронний ресурс] // Режим доступу: https://uk.wikipedia.org/wiki/TypeScript
7. Oracle vs PostgreSQL [Електронний ресурс] // Режим доступу: https://www.educba.com/oracle-vs-postgresql/
8. Git [Електронний ресурс] // Режим доступу: https://sebweo.com/scho-take-git-ta-github-kerivnitstvo-dlya-pochatkivtsiv/
9. MVC Architecture [Електронний ресурс] // Режим доступу: https://developer.chrome.com/apps/app_frameworks

ДОДАТОК А

Тексти програмного коду

Інформаційна система підтримки процесу автомобільних паркувань
(Найменування програми (документа))

DVD-R
(Вид носія даних)

22 арк, 151 Кб
(Обсяг програми (документа), арк., Кб)

Київ – 2020 року

Вихідні коди серверної частини застосунку:

```
@SpringBootApplication
public class Application implements ApplicationRunner {

   public static void main(String[] args) {
      SpringApplication.run(Application.class, args);
   }

   @Override
   public void run(ApplicationArguments args) throws Exception {

   }
}


@Component
@Scope(value = WebApplicationContext.SCOPE_REQUEST, proxyMode =
ScopedProxyMode.TARGET_CLASS)
public class UserContext {

   private UserPrincipal currentUser;

   private UserPrincipal getCurrentUser() {
      if (currentUser == null) {
         currentUser = (UserPrincipal)
SecurityContextHolder.getContext().getAuthentication().getPrincipal();
      }

      return currentUser;
   }

   public String getCurrentUsername() {
      return getCurrentUser().getUsername();
   }
}


public interface UserRepository extends JpaRepository<User, Long>,
QuerydslPredicateExecutor<User> {

   @Query("SELECT u FROM User u LEFT JOIN FETCH u.roles WHERE u.email = :email")
   User findWithRolesByEmail(@Param("email") String email);

   User findByEmail(String email);
}


public interface RoleRepository extends JpaRepository<Role, Long> {

   Set<Role> findDistinctByAdminIsTrue();

   Set<Role> findDistinctBySupplierIsTrue();

   Set<Role> findDistinctByCustomerIsTrue();
}


@Component
public class UserProvider {
```

```
    @Autowired
    private UserRepository userRepository;

    @Cacheable(cacheManager = "cacheManager", cacheNames = "users")
    public User getUser(String email) {
        return userRepository.findWithRolesByEmail(email);
    }
}


@Aspect
@Component
public class CheckUserAspect {

    private static final Logger LOGGER = LoggerFactory.getLogger(CheckUserAspect.class);
    private static final String ACCESS_DENIED_MESSAGE = "User %s tried to access resource
which is not allowed.";

    @Autowired
    private CustomerRepository customerRepository;

    @Autowired
    private SupplierRepository supplierRepository;

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private UserContext userContext;

    @Before("@annotation(com.kpi.parking.validation.CheckCustomer)")
    public void checkCustomer() {
        checkUser(customerRepository::existsByUserEmail);
    }

    @Before("@annotation(com.kpi.parking.validation.CheckSupplier)")
    public void checkSupplier() {
        checkUser(supplierRepository::existsByUserEmail);
    }

    @Before("@annotation(com.kpi.parking.validation.CheckAdmin)")
    public void checkAdmin() {

        Function<String, Boolean> checkAdmin = email -> {
          User user = userRepository.findByEmail(email);

          return user.getType() == UserType.ADMIN;
        };

        checkUser(checkAdmin);
    }

    private void checkUser(Function<String, Boolean> checkFunction) {

        String username = userContext.getCurrentUsername();

        if (checkFunction.apply(username)) {
            return;
        }
```

```
        String message = String.format(ACCESS_DENIED_MESSAGE, username);

        LOGGER.error(message);

        throw new AccessDeniedException(message);
    }
}


public class AuthenticationFilter extends OncePerRequestFilter {

    private static final Collection<String> EXCLUDED_URL_PATTERNS = Arrays
            .asList("/login", "/registration/*", "/countries");

    private JwtTokenProvider jwtTokenProvider;
    private AntPathMatcher pathMatcher;

    public AuthenticationFilter(JwtTokenProvider jwtTokenProvider) {
        this.jwtTokenProvider = jwtTokenProvider;
        this.pathMatcher = new AntPathMatcher();
    }

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response,
FilterChain filterChain) throws ServletException, IOException {
        String token = jwtTokenProvider.resolveToken(request);

        if (StringUtils.isBlank(token) || !jwtTokenProvider.validateToken(token)) {
            response.sendError(HttpServletResponse.SC_UNAUTHORIZED);
            return;
        }

        filterChain.doFilter(request, response);
    }

    @Override
    protected boolean shouldNotFilter(HttpServletRequest request) throws ServletException {
        return EXCLUDED_URL_PATTERNS.stream().anyMatch(p -> pathMatcher.match(p,
request.getServletPath()));
    }
}


public class JwtConfigurer extends SecurityConfigurerAdapter<DefaultSecurityFilterChain,
HttpSecurity> {

    private JwtTokenProvider jwtTokenProvider;

    public JwtConfigurer(JwtTokenProvider jwtTokenProvider) {
        this.jwtTokenProvider = jwtTokenProvider;
    }

    @Override
    public void configure(HttpSecurity builder) throws Exception {
        JwtTokenFilter jwtTokenFilter = new JwtTokenFilter(jwtTokenProvider);
        builder.addFilterBefore(jwtTokenFilter, UsernamePasswordAuthenticationFilter.class);
    }
}
```

```
public class JwtTokenFilter extends GenericFilterBean {

    private JwtTokenProvider jwtTokenProvider;

    public JwtTokenFilter(JwtTokenProvider jwtTokenProvider) {
        this.jwtTokenProvider = jwtTokenProvider;
    }

    @Override
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws
IOException, ServletException {

        String token = jwtTokenProvider.resolveToken((HttpServletRequest) request);

        if (StringUtils.isNotBlank(token) && jwtTokenProvider.validateToken(token)) {
            Authentication authentication = jwtTokenProvider.getAuthentication(token);
            SecurityContextHolder.getContext().setAuthentication(authentication);
        }

        chain.doFilter(request, response);
    }
}


@Component
public class JwtTokenProvider {

    private static final String AUTHORIZATION_HEADER_NAME = "Authorization";
    private static final String BEARER = "Bearer ";

    @Value("${security.jwt.token.secret:defaultsecret}")
    private String secret;

    @Value("${security.jwt.token.validity.period:3600000}")
    private Long validityPeriod;

    @Autowired
    private UserDetailServiceImpl userDetailService;

    @PostConstruct
    protected void init() {
        secret = Base64.getEncoder().encodeToString(secret.getBytes());
    }

    public String generateToken(String username) {

        Claims claims = Jwts.claims().setSubject(username);
        Date now = new Date();
        Date validity = new Date(now.getTime() + validityPeriod);

        return Jwts.builder()
                .setClaims(claims)
                .setIssuedAt(now)
                .setExpiration(validity)
                .signWith(SignatureAlgorithm.HS256, secret)
                .compact();
    }
```

```
    private String getUsername(String token) {
        return Jwts.parser().setSigningKey(secret).parseClaimsJws(token).getBody().getSubject();
    }

    Authentication getAuthentication(String token) {
        UserDetails userDetails = userDetailService.loadUserByUsername(getUsername(token));
        return new UsernamePasswordAuthenticationToken(userDetails, "",
userDetails.getAuthorities());
    }

    public String resolveToken(HttpServletRequest request) {

        String token = request.getHeader(AUTHORIZATION_HEADER_NAME);

        if (StringUtils.isNotBlank(token) && token.startsWith(BEARER)) {
            return token.substring(7);
        }

        return null;
    }

    public boolean validateToken(String token) {
        try {
            Jws<Claims> claimsJws = Jwts.parser().setSigningKey(secret).parseClaimsJws(token);
            return claimsJws.getBody().getExpiration().after(new Date());
        } catch (JwtException e) {
            throw new JwtException("Expired or invalid JWT token");
        }
    }
}


@Configuration
public class AuthenticationConfiguration {

    @Bean
    public FilterRegistrationBean<AuthenticationFilter> authenticationFilter(@Autowired
JwtTokenProvider jwtTokenProvider) {

        FilterRegistrationBean<AuthenticationFilter> registrationBean = new
FilterRegistrationBean<>();
        registrationBean.setFilter(new AuthenticationFilter(jwtTokenProvider));
        registrationBean.addUrlPatterns("*");
        registrationBean.setName("authenticationFilter");
        registrationBean.setOrder(1);

        return registrationBean;
    }
}


@Service
public class UserDetailServiceImpl implements UserDetailsService {

    @Autowired
    private UserRepository userRepository;

    @Override
    public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {
```

```
        User user = userRepository.findWithRolesByEmail(email);

        if (user == null) {
            throw new UsernameNotFoundException("User not found.");
        }

        return new UserPrincipal(user);
    }
}


@Service
@Transactional
public class RegistrationService {

    @Autowired
    private PasswordEncoder passwordEncoder;

    @Autowired
    private CustomerRepository customerRepository;

    @Autowired
    private SupplierRepository supplierRepository;

    @Autowired
    private CustomerMapper customerMapper;

    @Autowired
    private SupplierMapper supplierMapper;

    @Autowired
    private AddressService addressService;

    @Autowired
    private RoleRepository roleRepository;

    @Autowired
    private LoginResponseFactory loginResponseFactory;

    public LoginResponse createCustomerAccount(CustomerRequest customerRequest) {

        Customer customer = customerMapper.map(customerRequest);
        User user = customer.getUser();

        user.setPassword(passwordEncoder.encode(customerRequest.getPassword()));
        user.setRoles(roleRepository.findDistinctByCustomerIsTrue());
        user.setType(UserType.CUSTOMER);

        customerRepository.save(customer);

        return loginResponseFactory.buildResponse(user.getType(), user.getEmail());
    }

    public LoginResponse createSupplierAccount(SupplierRequest supplierRequest) {

        Supplier supplier = supplierMapper.map(supplierRequest);
        User user = supplier.getUser();

        user.setPassword(passwordEncoder.encode(supplierRequest.getPassword()));
        user.setRoles(roleRepository.findDistinctBySupplierIsTrue());
```

```
        user.setType(UserType.SUPPLIER);

        addressService.enrichWithCoordinates(supplier.getAddress());

        supplierRepository.save(supplier);

        return loginResponseFactory.buildResponse(user.getType(), user.getEmail());
    }
}


@GroupSequence({
        Mandatory.class,
        ValidEmail.class,
        UniqueEmail.class,
        PhoneNumberFormat.class,
        KnownCountry.class,
        Default.class
})
public interface RegistrationSequence {
}


@Target(ElementType.PARAMETER)
@Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy = PhoneNumberFormatValidator.class)
public @interface PhoneNumberFormat {

    String message() default "{EPS005_number_format}";

    Class<?>[] groups() default {};

    Class<? extends Payload>[] payload() default {};
}


public class PhoneNumberFormatValidator implements StatelessValidator<PhoneNumberFormat,
String> {

    private Pattern pattern = Pattern.compile("\\+?\\d{12}");

    @Override
    public boolean isValid(String number, ConstraintValidatorContext constraintValidatorContext) {
        return pattern.matcher(number).matches();
    }
}


@Target(ElementType.PARAMETER)
@Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy = UniqueEmailValidator.class)
public @interface UniqueEmail {

    String message() default "{EPS004_unique_email}";

    Class<?>[] groups() default {};

    Class<? extends Payload>[] payload() default {};
}
```

```
public class UniqueEmailValidator implements StatelessValidator<UniqueEmail, String> {

    @Autowired
    private UserRepository userRepository;

    @Override
    public boolean isValid(String email, ConstraintValidatorContext constraintValidatorContext) {
        return !userRepository.exists(QUser.user.email.eq(email));
    }
}


@Target(ElementType.PARAMETER)
@Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy = ValidEmailValidator.class)
public @interface ValidEmail {

    String message() default "{EPS003_valid_email}";

    Class<?>[] groups() default {};

    Class<? extends Payload>[] payload() default {};
}


public class ValidEmailValidator implements StatelessValidator<ValidEmail, String> {

    private EmailValidator emailValidator = EmailValidator.getInstance();

    private Pattern pattern = Pattern.compile("^[\\w!#$%&'*+/=?`{|}~^-
]+(?:\\.[\\w!#$%&'*+/=?`{|}~^-]+)*@(?:[a-zA-Z0-9-]+\\.)+[a-zA-Z]{2,6}$");

    @Override
    public boolean isValid(String email, ConstraintValidatorContext constraintValidatorContext) {
        return pattern.matcher(email).matches();
    }
}


@RestController
@Validated(RegistrationSequence.class)
public class RegistrationController implements RegistrationApi {

    @Autowired
    private RegistrationService registrationService;

    @Override
    public ResponseEntity<LoginResponse> registerCustomer(@RequestBody CustomerRequest
customerRequest) {
        return ResponseEntity.ok(registrationService.createCustomerAccount(customerRequest));
    }

    @Override
    public ResponseEntity<LoginResponse> registerSupplier(@RequestBody SupplierRequest
supplierRequest) {
        return ResponseEntity.ok(registrationService.createSupplierAccount(supplierRequest));
    }
}
```

```
@Component
public class LoginResponseFactory {

  @Autowired
  private JwtTokenProvider jwtTokenProvider;

  @Autowired
  private SupplierRepository supplierRepository;

  @Autowired
  private CustomerRepository customerRepository;

  public LoginResponse buildResponse(@NotNull UserType userType, @NotBlank String username)
{
    return new LoginResponse()
        .userType(userType.name())
        .name(buildName(userType, username))
        .token(jwtTokenProvider.generateToken(username));
  }

  private String buildName(UserType userType, String username) {
    switch (userType) {
      case CUSTOMER:
        Customer customer = customerRepository.findByUserEmail(username);
        return String.format("%s %s", customer.getFirstName(), customer.getLastName());
      case SUPPLIER:
        Supplier supplier = supplierRepository.findByUserEmail(username);
        return supplier.getName();
      case ADMIN:
        return "Admin";
      default:
        return "Unknown";
    }
  }
}


@GroupSequence({
    Mandatory.class,
    UserCredentialsAreCorrect.class,
    Default.class
})
public interface LoginSequence {
}


@Target(ElementType.PARAMETER)
@Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy = UserCredentialsAreCorrectValidator.class)
public @interface UserCredentialsAreCorrect {

  String message() default "{EPS007_user_credential_are_not_correct}";

  Class<?>[] groups() default {};

  Class<? extends Payload>[] payload() default {};
}
```

```
public class UserCredentialsAreCorrectValidator implements
StatelessValidator<UserCredentialsAreCorrect, User> {

   @Autowired
   private UserProvider userProvider;

   @Autowired
   private PasswordEncoder passwordEncoder;

   @Override
   public boolean isValid(User user, ConstraintValidatorContext constraintValidatorContext) {

      com.kpi.parking.users.domain.User persistedUser = userProvider.getUser(user.getEmail());

      return persistedUser != null && passwordEncoder.matches(user.getPassword(),
persistedUser.getPassword());
   }
}


@Target(ElementType.PARAMETER)
@Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy = UserExistValidator.class)
public @interface UserExist {

   String message() default "{EPS006_user_does_not_exist}";

   Class<?>[] groups() default {};

   Class<? extends Payload>[] payload() default {};
}


public class UserExistValidator implements StatelessValidator<UserExist, String> {

   @Autowired
   private UserProvider userProvider;

   @Override
   public boolean isValid(String email, ConstraintValidatorContext constraintValidatorContext) {
      return userProvider.getUser(email) != null;
   }
}


@RestController
@Validated(LoginSequence.class)
public class LoginController implements LoginApi {

   @Autowired
   private UserProvider userProvider;

   @Autowired
   private LoginResponseFactory loginResponseFactory;

   @Override
   public ResponseEntity<LoginResponse> loginUser(@RequestBody User credentials) {
      String username = credentials.getEmail();
      com.kpi.parking.users.domain.User user = userProvider.getUser(username);
```

```
    return ResponseEntity.ok(loginResponseFactory.buildResponse(user.getType(), username));
  }
}
```

Вихідні коди клієнтської частини застосунку:

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


@Injectable({
  providedIn: 'root'
})
export class CountryService {
  constructor(private request: RequestService) {
  }

  getCountries() {
    return this.request.get('countries');
  }
}

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


@Injectable({
  providedIn: 'root'
})
export class NotificationService {
  actions = new Map([
    ['login', 'Авторизація'],
    ['register', 'Реєстрація'],
    ['updateAccountSupp', 'Оновлення персональної інформаці'],
    ['createParking', 'Створення парковки'],
    ['updateParking', 'Оновлення інформації парковки'],
    ['deleteParking', 'Видалення парковки'],
    ['deleteParkingPlace', 'Видалення паркомісця'],
    ['uploadFile', 'Завантаження документу'],
    ['deleteSupplierDocument', 'Видалення документу'],
    ['updateAccountCust', 'Оновлення персональної інформаці'],
    ['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 забронювано. 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);
    }
}

import {Injectable} from '@angular/core';
import {HttpClient, HttpRequest} from '@angular/common/http';
import {HttpHeaders} from "@angular/common/http";

import {environment} from "../../environments/environment";
import {CookieService} from "ngx-cookie-service";

@Injectable({
  providedIn: 'root'
})
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});
  }

  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});
  }
}

import { Injectable } from '@angular/core';
import { RequestService } from "./request.service";
import { CookieService } from "ngx-cookie-service";
import { Router } from "@angular/router";
```

```
import {environment} from "../../environments/environment";


@Injectable({
  providedIn: 'root'
})
export class UserService {
  constructor(
    private requestService: RequestService,
    private cookie:CookieService,
    private router: Router) {}

  login(email, password){
    const body = {
      email: email,
      password: password
    };
    return this.requestService.post('login', body);
  }

  createCustomer(data){
    const body = {
      firstName: data.firstName.value,
      lastName: data.lastName.value,
      email: data.email.value,
      phoneNumber: data.phoneNumber.value,
      password: data.password.value
    };
    return this.requestService.post('registration/customer', body);
  }

  createSupplier(data){
    const body = {
      name: data.name.value,
      address: {
        country: data.country.value,
        city: data.city.value,
        street: data.street.value,
        streetNumber: data.streetNumber.value
      },
      vatNumber: data.vatNumber.value,
      email: data.email.value,
      phoneNumber: data.phoneNumber.value,
      password: data.password.value
    };
    return this.requestService.post('registration/supplier', body);
  }

  isLogged():boolean{
    return this.cookie.check('token');
  }
  getUserType():string{
    let type = '';
    if(this.cookie.check('userType')){
      type = atob(this.cookie.get('userType'));
    }
    return type;
  }

  logout(){
    this.cookie.delete('token');
    this.cookie.delete('userType');

    window.location.href = environment.siteUrl
  }

}


import { Component, OnInit } from '@angular/core';
import {Title } from "@angular/platform-browser";
```

```
import {CookieService} from "ngx-cookie-service";

@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.less']
})
export class HomeComponent implements OnInit {
  isLogged: Boolean = false;
  isLoggedCustomer: Boolean = false;
  constructor(
    private title:Title,
    private cookie: CookieService
  ) {
    this.title.setTitle("Booking Place");
  }

  ngOnInit() {
    this.initHomeComponent();

  }

  initHomeComponent(){
    if(this.cookie.check('token') && this.cookie.check('userType') &&
atob(this.cookie.get('userType')) == 'CUSTOMER'){
      this.isLoggedCustomer = true;
    }else{
      this.isLoggedCustomer = false;
    }

    this.isLogged = this.cookie.check('token');
  }

}

import { Component, OnInit } from '@angular/core';
import { Router } from "@angular/router";
import { CookieService } from "ngx-cookie-service";
import * as $ from 'jquery';
import {environment} from "../../../environments/environment";
import {UserService} from "../../_services/user.service";

@Component({
  selector: 'app-header',
  templateUrl: './header.component.html',
  styleUrls: ['./header.component.less']
})
export class HeaderComponent implements OnInit {
  renderLogin:boolean = true;
  isLogged: boolean = false;
  isCustomerLogged:boolean = false;
  userName: string = '';
  link = {
    home:environment.siteUrl
  };
  constructor(
    private router: Router,
    private cookie: CookieService,
    private userService: UserService
  ) { }

  ngOnInit() {

    this.isLogged = this.cookie.check('token');

    if(this.isLogged && this.cookie.check('userType') &&
atob(this.cookie.get('userType')) == 'CUSTOMER'){
      this.isCustomerLogged = true;
    }else{
      this.isCustomerLogged = false;
    }
```

```
    if(this.isLogged && this.cookie.check('userName')){
      this.userName = this.cookie.get('userName');
    }


  }
  accountOpen(): void{
    if(this.cookie.check('token')){
      if(atob(this.cookie.get('userType')) === 'CUSTOMER'){
        this.router.navigate(['cus-account'])
      }else {
        this.router.navigate(['account']);
      }
    }else{
      $('#login').click();
    }
  }

  logout(){
    this.userService.logout();
  }

}


import {Component, OnInit} from '@angular/core';
import {ComponentRef, ComponentFactoryResolver, ViewContainerRef, ViewChild} from
"@angular/core";
import {FormBuilder, FormGroup, Validators} from '@angular/forms';
import {UserService} from "../../_services/user.service";
import {Router} from '@angular/router';
import * as $ from 'jquery';
import 'magnific-popup';
import {CookieService} from "ngx-cookie-service";
import {NotificationComponent} from "../notification/notification.component";
import {environment} from "../../../environments/environment";

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.less']
})
export class LoginComponent implements OnInit {
  componentRef: any;
  @ViewChild('viewContainerRef', {read: ViewContainerRef, static: false}) entry:
ViewContainerRef;
  loginForm: FormGroup;
  email: string;
  isSubmit = false;
  miLengthError = false;

  constructor(
    private formBuilder: FormBuilder,
    private userService: UserService,
    private cookiesService: CookieService,
    private router: Router,
    private resolver: ComponentFactoryResolver
  ) {
  }

  ngOnInit() {
    let saveData = '', email = '', password = '';
    if (localStorage.getItem('saveData') && localStorage.getItem('saveData') != '0') {
      saveData = localStorage.getItem('saveData');
      email = localStorage.getItem('email');
      password = btoa(localStorage.getItem('password'));
    }
    this.loginForm = this.formBuilder.group({
      email: [email, [Validators.required, Validators.email]],
      password: [password, [Validators.required, Validators.minLength(4)]],
```

```
      saveData: [saveData]
    });
    $(document).ready(function () {
      $('.popup-with-zoom-anim').magnificPopup({
        type: 'inline',
        fixedContentPos: false,
        fixedBgPos: true,
        overflowY: 'auto',
        closeBtnInside: true,
        preloader: false,
        midClick: true,
        removalDelay: 300,
        mainClass: 'my-mfp-zoom-in'
      });
    });
  }

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

  // convenience getter for easy access to form fields
  get f() {
    return this.loginForm.controls;
  }

  login(): void {
    this.isSubmit = true;
    if (!this.validation()) {
      return;
    }
    this.startLoader();
    const promise = new Promise((resolve, reject) => {
      this.userService.login(this.loginForm.value.email,
this.loginForm.value.password).toPromise()
        .then((res: any) => {
            this.cookiesService.set('token', res.token, 2 / 24);
            this.cookiesService.set('userType', btoa(res.userType), 2 / 24);
            this.cookiesService.set('userName', res.name, 2 / 24);
            if (this.loginForm.value.saveData) {
              localStorage.setItem('email', this.loginForm.value.email);
              localStorage.setItem('password', atob(this.loginForm.value.password));
              localStorage.setItem('saveData', '1');
            } else {
              localStorage.setItem('saveData', '0');
            }
            this.hideModals();
            if (res.userType === 'CUSTOMER') {
              window.location.href = environment.siteUrl
            } else {
              window.location.href = `${environment.siteUrl}account`;
            }
            this.endLoader();
            resolve();
          },
          err => {
            // Error
            if (err['error'].length) {
              this.setNotification(err['error'][0].code);
            }
            this.endLoader();
          }
        ).catch();
```

```
    });
  }

  validation(): boolean {
    let password = this.loginForm.value.password;
    if (password.length < 4) {
      this.miLengthError = true;
    } else {
      this.miLengthError = false;
    }
    return !this.loginForm.invalid;
  }

  resetError(): void {
    this.isSubmit = false;
  }

  hideModals() {
    $('.mfp-close').click();
  }

  startLoader() {
    $('.block-loader').show();
    $('#login').prop("disabled", true)
  }

  endLoader() {
    $('.block-loader').hide();
    $('#login').prop("disabled", false);
  }
}

import {Component, OnInit} from '@angular/core';
import {isNumber} from "util";
import {UserService} from "../../_services/user.service";
import {CookieService} from "ngx-cookie-service";
import {Router} from "@angular/router";
import {CountryService} from "../../_services/country.service";


import * as $ from 'jquery';
import 'magnific-popup';

import {ComponentRef, ComponentFactoryResolver, ViewContainerRef, ViewChild} from
"@angular/core";
import {NotificationComponent} from "../notification/notification.component";


@Component({
  selector: 'app-register',
  templateUrl: './register.component.html',
  styleUrls: ['./register.component.less'],
  providers: [UserService]
})
export class RegisterComponent implements OnInit {
  type: string = 'user';
  isCustomer: boolean = true;
  isSupplier: boolean = false;
  isSubmitedForm: boolean = false;
  countries;
  user = {
    name: {value: '', error: false},
    vatNumber: {value: '', error: false},
    country: {value: '', error: false},
    city: {value: '', error: false},
    street: {value: '', error: false},
    streetNumber: {value: '', error: false},
    firstName: {value: '', error: false},
    lastName: {value: '', error: false},

    email: {value: '', error: false},
```

```
    phoneNumber: {value: '', error: false},
    password: {value: '', error: false},
    secondPassword: {value: '', error: false}
  };

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

  constructor(
    private userService: UserService,
    private cookies: CookieService,
    private router: Router,
    private country: CountryService,
    private resolver: ComponentFactoryResolver
  ) {
  }

  ngOnInit() {
    $(document).ready(function () {
      $('.popup-with-zoom-anim').magnificPopup({
        type: 'inline',
        fixedContentPos: false,
        fixedBgPos: true,
        overflowY: 'auto',
        closeBtnInside: true,
        preloader: false,
        midClick: true,
        removalDelay: 300,
        mainClass: 'my-mfp-zoom-in'
      });
    });
    this.getCountry();
  }

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

  changeType(type): void {
    if (type != this.type) {
      if (type == 'customer') {
        this.type = 'customer';
      } else {
        this.type = 'user';
      }
      this.type = type;
      this.isCustomer = !this.isCustomer;
      this.isSupplier = !this.isSupplier;
    }
  }

  register(): void {
    this.isSubmitedForm = true;
    if (this.validateClient()) {
      this.startLoader();
      if (this.isSupplier) {
        this.createSupplier();
      } else {
        this.createCustomer();
```

```
      }
    }
  }

  createCustomer(): void {
    const promise = new Promise((resolve, reject) => {
      this.userService.createCustomer(this.user).toPromise()
        .then((res: any) => {
            this.cookies.set('token', res.token, 2 / 24);
            this.cookies.set('userType', btoa(res.userType), 2 / 24);
            this.hideModals();
            this.router.navigate(['customer/account']);

            resolve();
            this.endLoader();
          },
          err => {
            // Error
            if (err['error'].length) {
              this.setNotification(err['error'][0].code);
            }
            reject(err);
            this.endLoader();
          }
        ).catch();
    });
  }

  createSupplier(): void {
    const promise = new Promise((resolve, reject) => {
      this.userService.createSupplier(this.user).toPromise()
        .then((res: any) => {
            this.cookies.set('token', res.token);
            this.cookies.set('userType', btoa(res.userType));
            this.hideModals();
            this.router.navigate(['account']);
            resolve();
            this.endLoader();
          },
          err => {
            // Error
            if (err['error'].length) {
              this.setNotification(err['error'][0].code);
            }
            reject(err);
            this.endLoader();
          }
        ).catch();
    });
  }

  hideModals() {
    $('.mfp-close').click();
  }

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

      if (isNumber(this.user.vatNumber.value) && this.user.vatNumber.value.length <
10) {
        this.user.vatNumber.error = true;
```

```
        error = true;
      } else {
        this.user.vatNumber.error = false;
      }

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

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

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

    if (this.isCustomer) {
      if (this.user.firstName.value.length < 2) {
        this.user.firstName.error = true;
        error = true;
      } else {
        this.user.firstName.error = false;
      }

      if (this.user.lastName.value.length < 2) {
        this.user.lastName.error = true;
        error = true;
      } else {
        this.user.lastName.error = false;
      }
    }
    if (this.user.phoneNumber.value.length < 10) {
      this.user.phoneNumber.error = true;
      error = true;
    } else {
      this.user.phoneNumber.error = false;
    }
    if (this.user.password.value.length < 4) {
      this.user.password.error = true;
      error = true;
    } else {
      this.user.password.error = false;
    }
    if (this.user.secondPassword.value != this.user.password.value) {
      this.user.secondPassword.error = true;
      error = true;
    } else {
      this.user.secondPassword.error = false;
    }
    if (!regexp.test(this.user.email.value)) {
      this.user.email.error = true;
      error = true;
    } else {
      this.user.email.error = false;
    }
    return !error;
  }

  setNotification(code) {
    this.entry.clear();
```

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

    startLoader() {
      $('.block-loader').show();
      $('#register').prop("disabled", true)
    }

    endLoader() {
      $('.block-loader').hide();
      $('#register').prop("disabled", false);
    }
  }
```

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”
Кафедра автоматизованих систем обробки інформації і управління

УЗГОДЖЕНО	**ЗАТВЕРДЖУЮ**
Керівник проєкту	**В.о. завідувача кафедри**
_____ *Майя СПЕРКАЧ*	_____ *Олександр ПАВЛОВ*
(підпис) (вл. ім’я, прізвище)	(підпис) (вл. ім’я, прізвище)
“13” квітня 2020 р.	“14” квітня 2020 р.

«Інформаційна система підтримки процесу автомобільних паркувань»

ТЕХНІЧНЕ ЗАВДАННЯ

Шифр *ДП 6310.01.000 ТЗ*

На 11 сторінках

Київ – 2020 року

ЗМІСТ

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Повне найменування системи та її умовне позначення

Повна назва системи: Інформаційна система підтримки процесу автомобільних паркувань.

Коротке найменування системи: «Booking Place».

1.2 Найменування організації-замовника та організації-учасника робіт

Замовником є кафедра автоматизованих систем обробки інформації та управління Національного технічного університету України "Київський політехнічний інститут ім. Ігоря Сікорського" (далі за текстом — Замовник). Адреса замовника: м. Київ, п. Перемоги 37, 18 корпус ФІОТ.

Розробники сервісу — студенти групи ІС-63 кафедри автоматизованих систем обробки інформації та управління Національного технічного університету України "Київський політехнічний інститут ім. Ігоря Сікорського" Іщенко Владислав Сергійович та Левчук Володимир Іванович.

1.3 Перелік документів, на підставі яких створюється система

При розробці системи і створення проектно-експлуатаційної документації Виконавець повинен керуватися вимогами наступних нормативних документів:

- ДСТУ 19.201-78. Технічне завдання. Вимоги до змісту і оформлення;
- ДСТУ 34.601-90. Комплекс стандартів на автоматизовані системи. Автоматизовані системи. Стадії створення;
- ДСТУ 34.201-89. Інформаційні технології. Комплекс стандартів на автоматизовані системи. Види, комплексність і позначення документів при створенні автоматизованих систем.

1.4 Планові терміни початку і закінчення роботи зі створення системи

Плановий термін початку роботи над створенням системи підтримки процесу автомобільних паркувань – 1 грудня 2019 року.

Плановий термін по закінченню роботи над створенням системи підтримки процесу автомобільних паркувань – 22 травня 2020 року.

2 ПРИЗНАЧЕННЯ І ЦІЛІ СТВОРЕННЯ СИСТЕМИ

2.1 Призначення системи

Інформаційна система призначена для підтримки процесу автомобільних паркувань.

2.2 Цілі створення системи

Цілями розробки даної системи є:

- спрощення ведення паркобізнесу в умовах діджиталізації за рахунок створення загального майданчику для паркувань з великим асортиментом парковок;
- полегшення процесу пошуку та бронювання паркомісць за рахунок доступу до паркувальних майданчиків в різних локаціях та можливості онлайн бронювання паркомісць на них.

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

.

3 ХАРАКТЕРИСТИКА ОБ'ЄКТА АВТОМАТИЗАЦІЇ

Для того, щоб користуватися системою, користувач повинен бути зареєстрованим та авторизованим. У залежності від типу користувача, необхідно створити різні типи акаунтів. Для клієнтів необхідно створити приватний акаунт, а для власників парковок – комерційний.

Об'єктами автоматизації є здача паркомісць в оренду на детермінований проміжок часу та бронювання паркомісць на парковці на вказаний проміжок часу.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

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

Система має виконувати наступні функції для всіх типів користувачів:

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

Для типу користувача «Власник парковок» система повинна виконувати наступні функції:

1) завантаження та видалення документів, що підтверджують право власника парковок на ведення підприємницької діяльності;
2) додавання нових парковок та паркомісць на них;
3) редагування інформації про парковку;
4) видалення парковок та паркомісць, за умови того, що вони не використовуються;
5) зміни тарифу на паркування та суми штрафу за несвоєчасне звільнення парковки;
6) завантаження та видалення документів, що підтверджують реєстрацію парковки;
7) перегляд списку бронювань на певній парковці за вказаний проміжок часу;

8) перегляд динаміки зміни тарифів на паркування по кожній парковці.

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

1) ведення власними банківськими картками;
2) перегляд історії замовлень за вказаний період;
3) генерація QR-коду для кожного замовлення, котрий використовується при в'їзді на парковку;
4) перегляд історії штрафів по статусам;
5) оплата штрафів;
6) пошук вільних місць на обраній парковці в заданий проміжок часу;
7) пошук найближчих парковок з вільними місцями;
8) бронювання паркомісць.

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

4.2 Вимоги до надійності

Система повинна функціонувати безвідмовно незважаючи на наявність ймовірних дефектів, які можуть проявлятися під час експлуатації. Виправлення таких дефектів повинно виконуватися на етапах розробки, бета-тестування та в процесі введення в дію і в межах промислової експлуатації.

Відмова програмного забезпечення сервісу не повинна призводити до руйнувань даних в інформаційних сховищах.

Будь-які аварійні ситуації повинні бути негайно задокументовані і надіслані розробникам задля усунення причин збою в роботі системи.

4.3 Вимоги до складу і параметрів технічних засобів

Коректне функціонування даного програмного продукту на стороні користувача може бути гарантоване тільки за наступних умов:

- наявність комп’ютера з наступними технічними характеристиками:
  1) процесор з тактовою частотою не менше 1 ГГц;
  2) об’єм оперативної пам’яті не менше 1 ГБ;
  3) підключення до мережі інтернет;
  4) інші частини комп’ютеру можуть мати будь-які параметри, так як не мають значного впливу.
- комп'ютерна периферія, до складу якої входить:
  1) монітор;
  2) мишка;
  3) клавіатура;
- встановлений один з веб-браузерів:
  1) Google Chrome 23+
  2) IE10+/Edge
  3) Firefox 21+
  4) Safari 6+
  5) Opera 15+

5 СТАДІЇ І ЕТАПИ РОЗРОБКИ

Основні етапи виконання робіт з розробки системи підтримки процесу автомобільних паркувань:

№ з/п	Назва етапу роботи	Термін виконання етапу	Результат виконання
1.	Підготовка технічного завдання на розробку програмного продукту	01.12.20119	
2.	Розробка сценарію роботи	13.12.2019	
3.	Технічне проектування – функціональність, модулі, задачі, цілі тощо	24.12.2019	
4.	Узгодження з керівником інтерфейсу користувача	10.01.2020	
5.	Розробка інформаційного забезпечення	25.01.2020	
6.	Розробка програмного забезпечення	07.04.2020	
7.	Налагодження програми	20.04.2020	
8.	Тестування програми	12.05.2020	
9.	Здача готового програмного продукту замовнику	22.05.2020	

6 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ СИСТЕМИ

6.1 Види випробувань

Задля перевірки правильності роботи програмного продукту буде проведено функціональне тестування. В ході тестування буде виконано перевірку всіх функціональних характеристик веб-застосунку. Також, система буде перевірена на відмовостійкість шляхом виконання некоректних дій користувачем. Окремими тестами буде перевірена безпека системи в цілому.

UNICHECK

Власник документу:
Попенко Володимир Дмитрович

Дата перевірки:
04.06.2020 18:31:39 EEST

Дата звіту:
04.06.2020 20:35:48 EEST

ID перевірки:
1003783676

Тип перевірки:
Doc vs Internet + Library

ID користувача:
77149

Назва документу: Ishenko_Levchuk_bachelor_Common

ID файлу: 1003798167 **Кількість сторінок:** 41 **Кількість слів:** 6073 **Кількість символів:** 45831 **Розмір файлу:** 2.19 MB

7.64% Схожість

Найбільша схожість: 3.59% з джерело бібліотеки. **ID файлу:** 1003798190

2.47% Схожість з Інтернет джерелами	13	Page 43
7.16% Текстові збіги по Бібліотеці акаунту	78	Page 43

0% Цитат

Не знайдено жодних цитат

0% Вилучень

Вилучений текст відсутній

Підміна символів

Не знайдено замінених символів

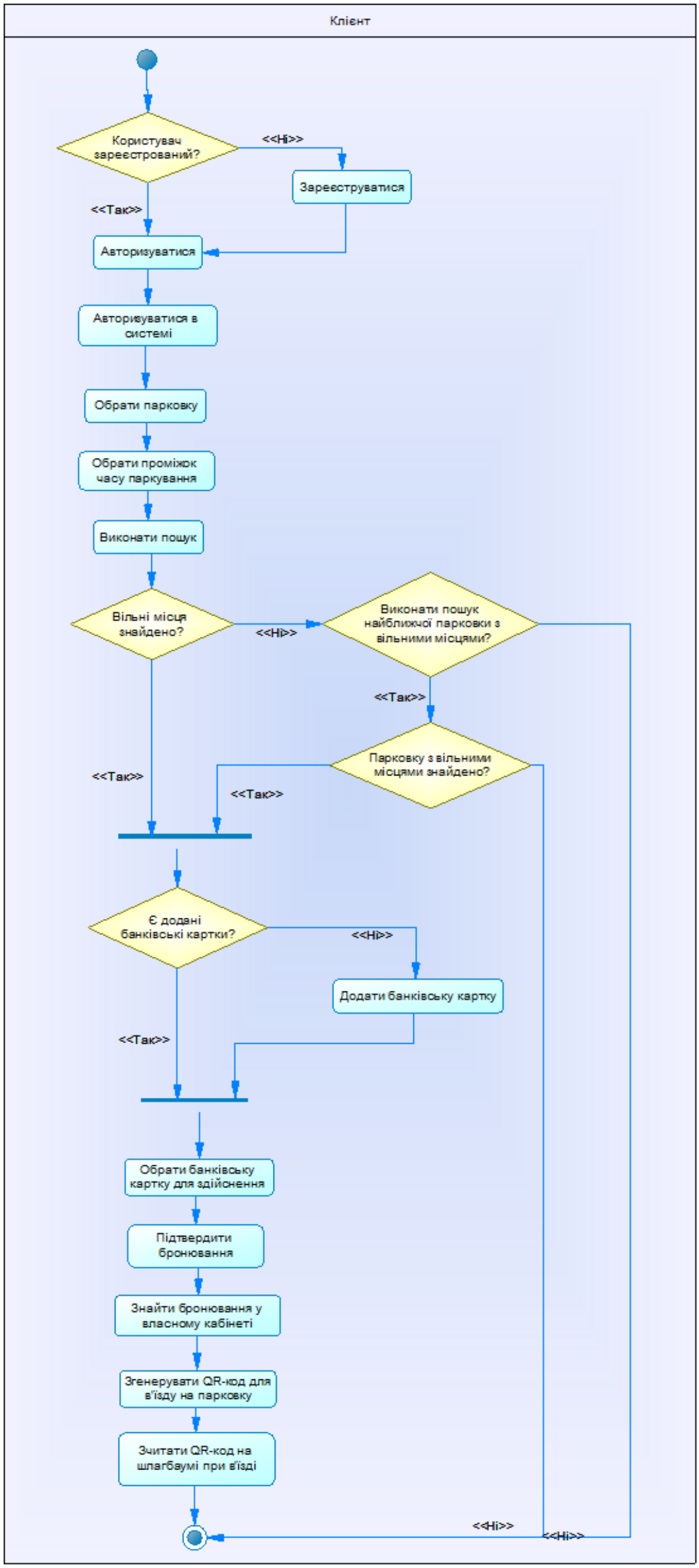
Графічний матеріал
до дипломного проєкту

на тему: «Інформаційна система підтримки процесу автомобільних паркувань» (комплексна тема)

Київ – 2020 року

					ДП 6310.02.000 ССД		
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна діяльності	Літера	Маса / Масштаб
Розробив		Іщенко В.С.					
		Левчук В.І.				Аркуш 1	Аркушів 4
Перевірив		Сперкач М.О.					
Т. кон.					Інформаційна система підтримки процесу автомобільних паркувань	КПІ ім. Ігоря Сікорського Каф. АСОІУ Гр. ІС-63	
Н. кон.		Проскура С.Л.					
Затвердив		Сперкач М.О.					

Власник парковок
Встановити шлагбаум
Найняти працівника для контролю
Встановити автомат для видачі талонів і проведення оплати



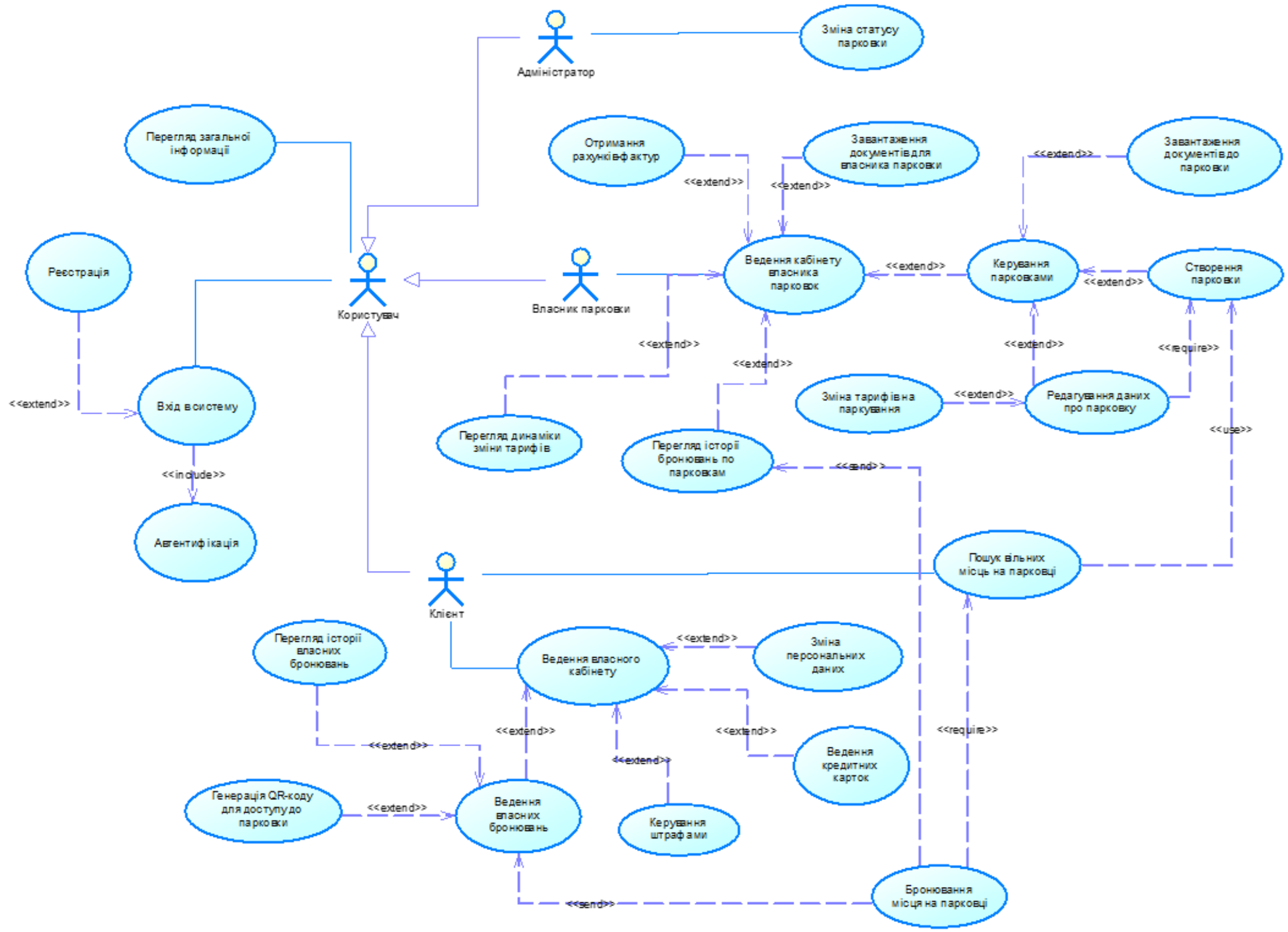
					ДП 6310.02.000 ССД
Зм.	Арк.	№ докум.	Підп.	Дата	
Розробив		Іщенко В.С.			Схема структурна діяльності
		Левчук В.І.			
Перевірив		Сперкач М.О.			
Т. кон.					Інформаційна система підтримки процесу автомобільних паркувань
Н. кон.		Проскура С.Л.			
Затвердив		Сперкач М.О.			

Лит.	Маса	Масштаб
Аркуш 3	Аркушів 4	

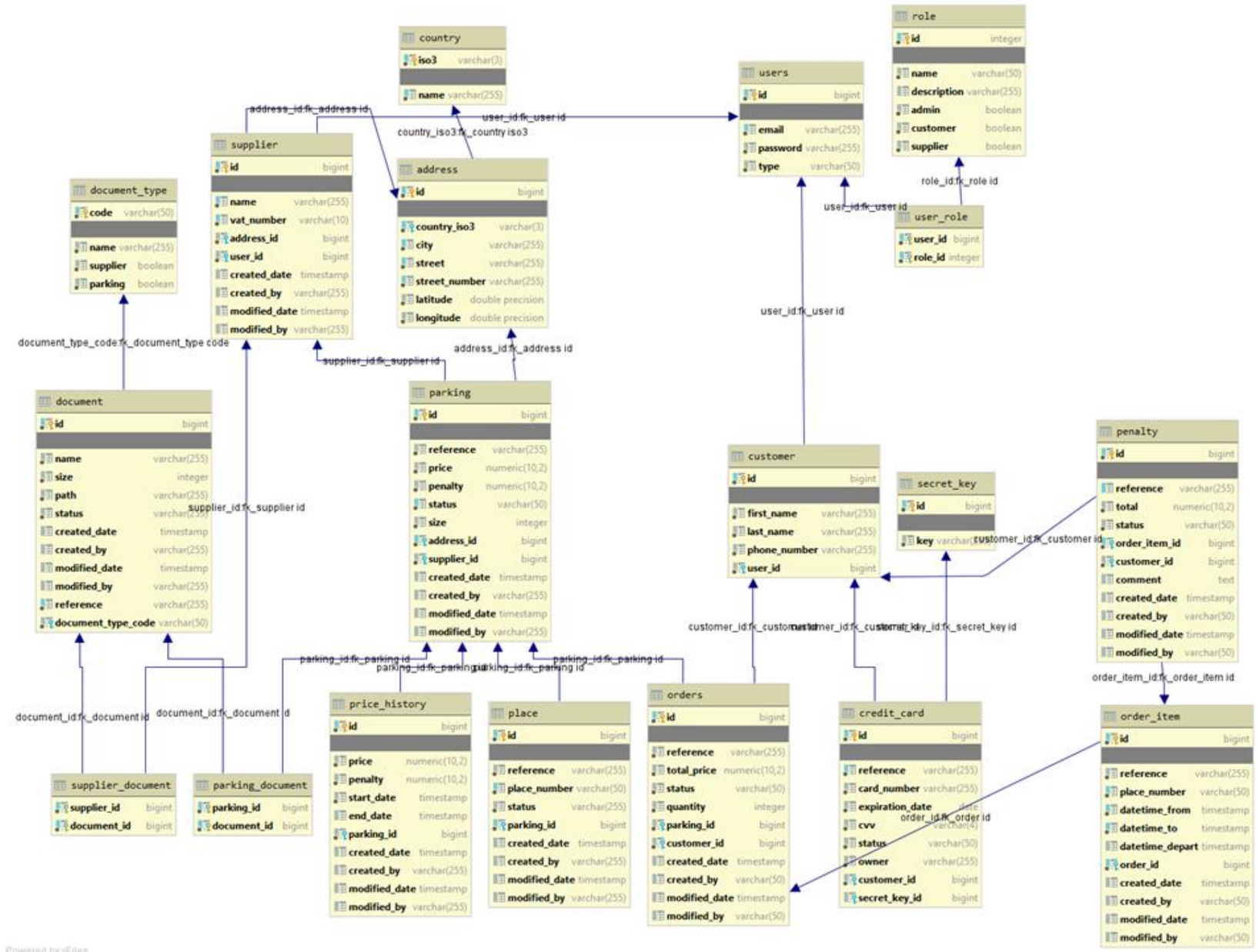
КПІ ім. Ігоря Сікорського
Каф. АСОІУ
Гр. ІС-63

					ДП 6310.02.000 ССД		
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна діяльності	Літера	Маса / Масштаб
Розробив		Іщенко В.С.					
		Левчук В.І.				Аркуш 4	Аркушів 4
Перевірив		Сперкач М.О.					
Т. кон.					Інформаційна система підтримки процесу автомобільних паркувань	КПІ ім. Ігоря Сікорського Каф. АСОІУ Гр. ІС-63	
Н. кон.		Проскура С.Л.					
Затвердив		Сперкач М.О.					

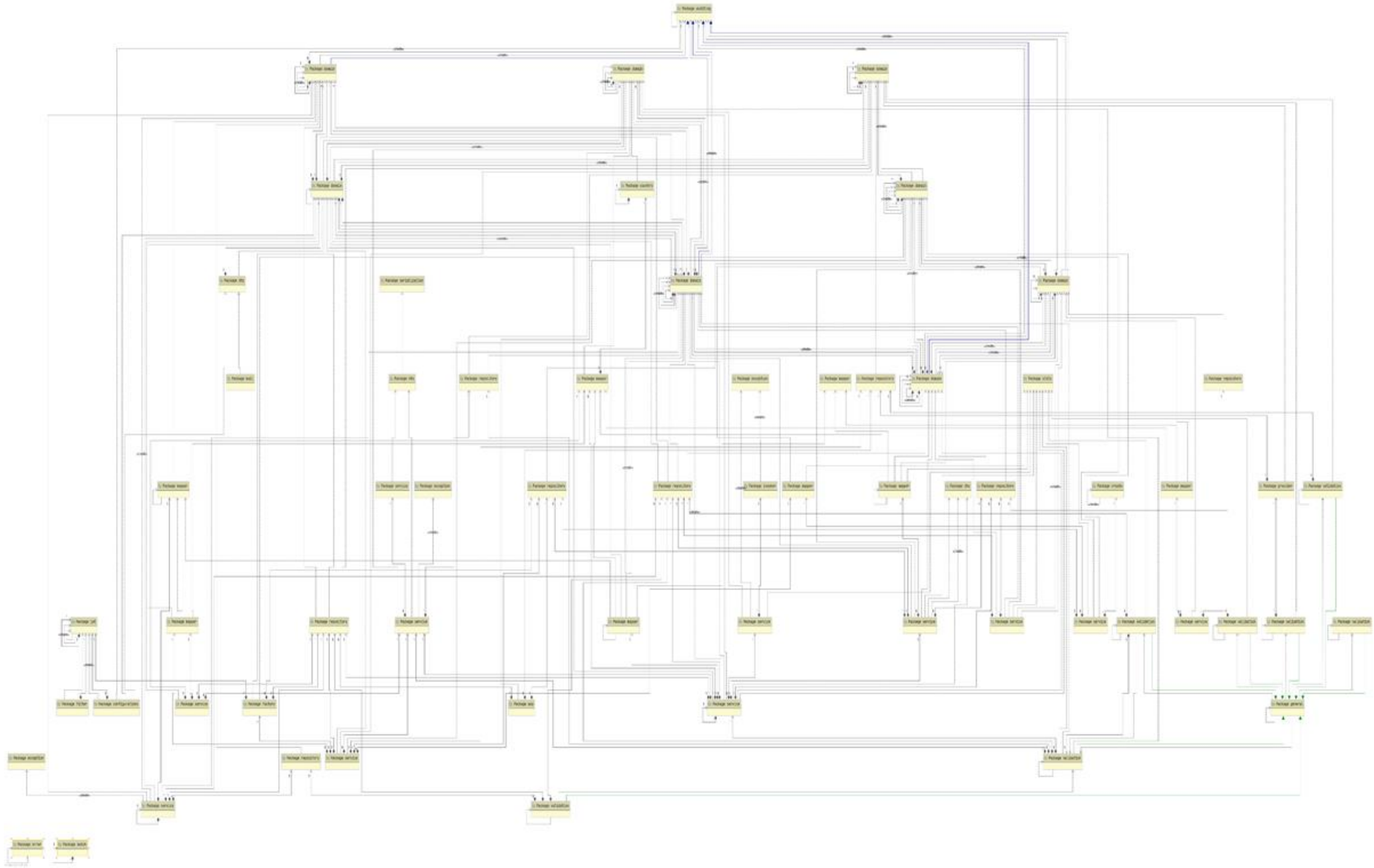
ДП 6310.03.000 ССВ



					ДП 6310.03.000 ССВ	
Зм.	Арк.	№ документа	Підпис	Дата		
Розробив		Іщенко В.С.			Схема структурна варіантів використання	Літера / Маса / Масштаб
		Левчук В.І.				Аркуш 1 / Аркушів 1
Перевірив		Сперкач М.О.				
Т. кон.					Інформаційна система підтримки процесу автомобільних паркувань	КПІ ім. Ігоря Сікорського Каф. АСОІУ Гр. ІС-63
Н. кон.		Проскура С.Л.				
Затвердив		Сперкач М.О.				

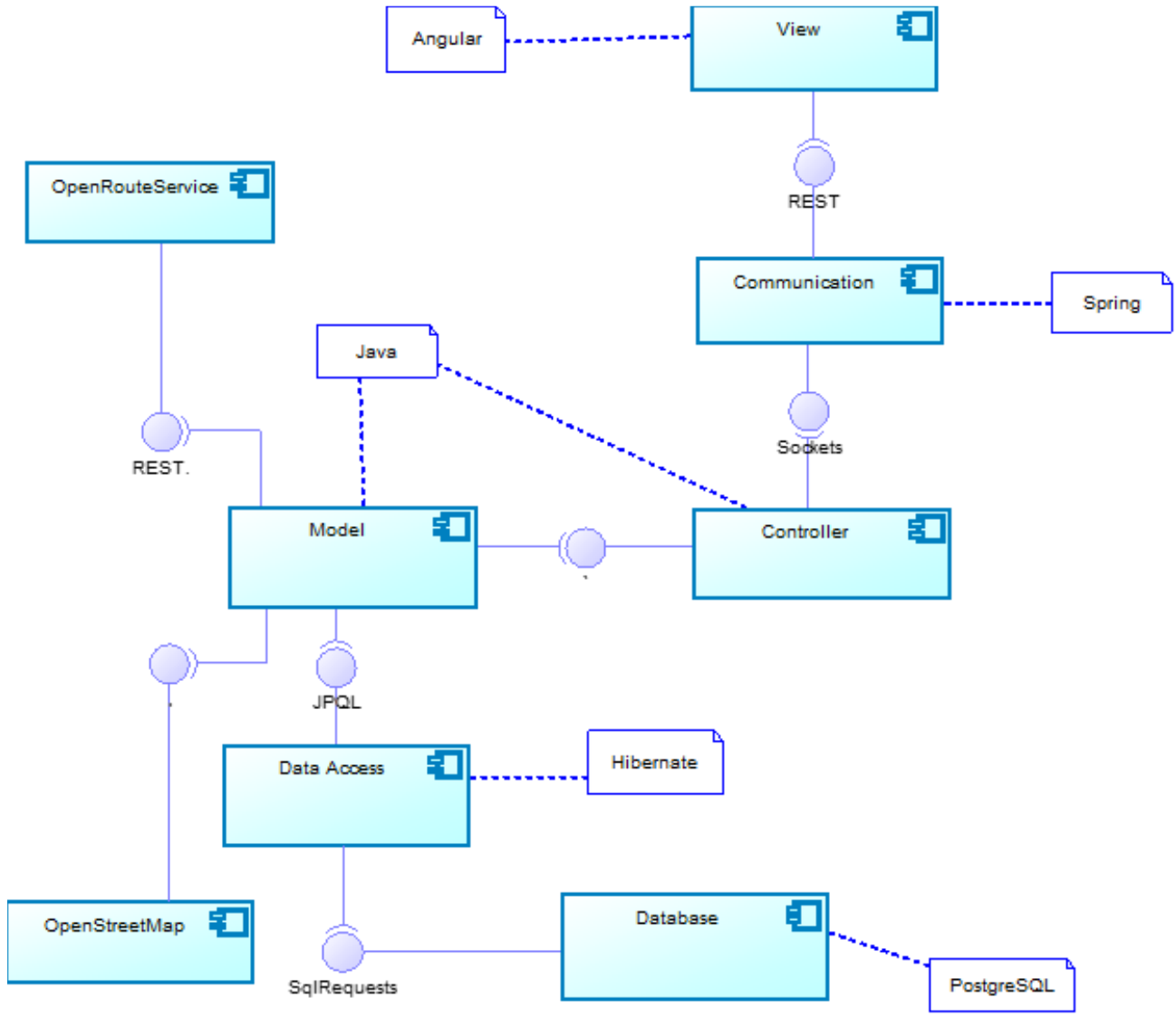


					ДП 6310.04.000 СБД			
Зм.	Арк.	№ документа	Підпис	Дата	*Схема бази даних*	Літера	Маса	Масштаб
Розробив		Іщенко В.С.						
		Левчук В.І.				Аркуш 1	Аркушів 1	
Перевірив		Сперкач М.О.						
Т. кон.					*Інформаційна система підтримки процесу автомобільних паркувань*	*КПІ ім. Ігоря Сікорського Каф. АСОІУ Гр. ІС-63*		
Н. кон.		Проскура С.Л.						
Затвердив		Сперкач М.О.						

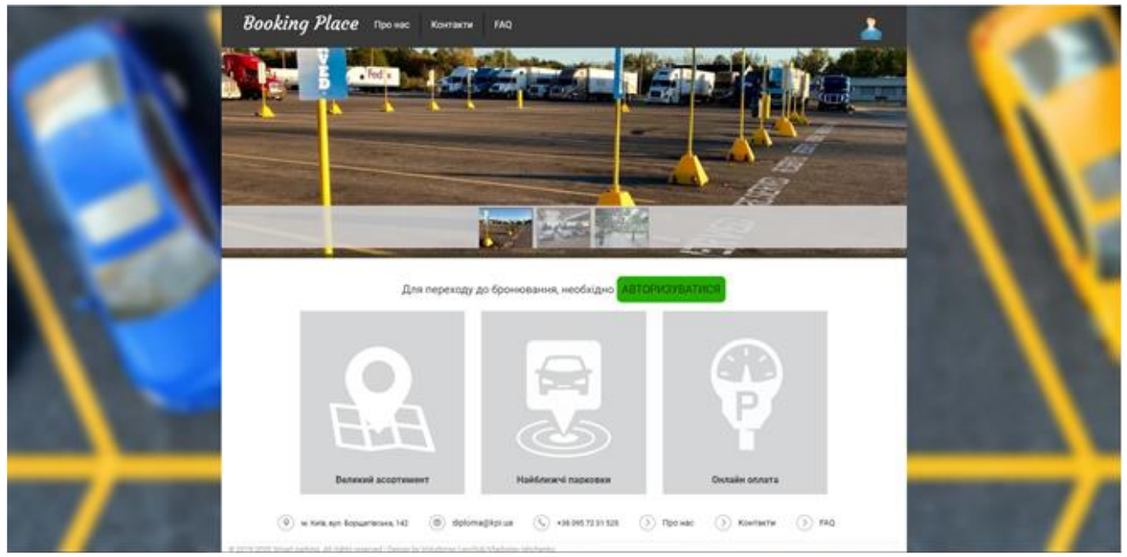


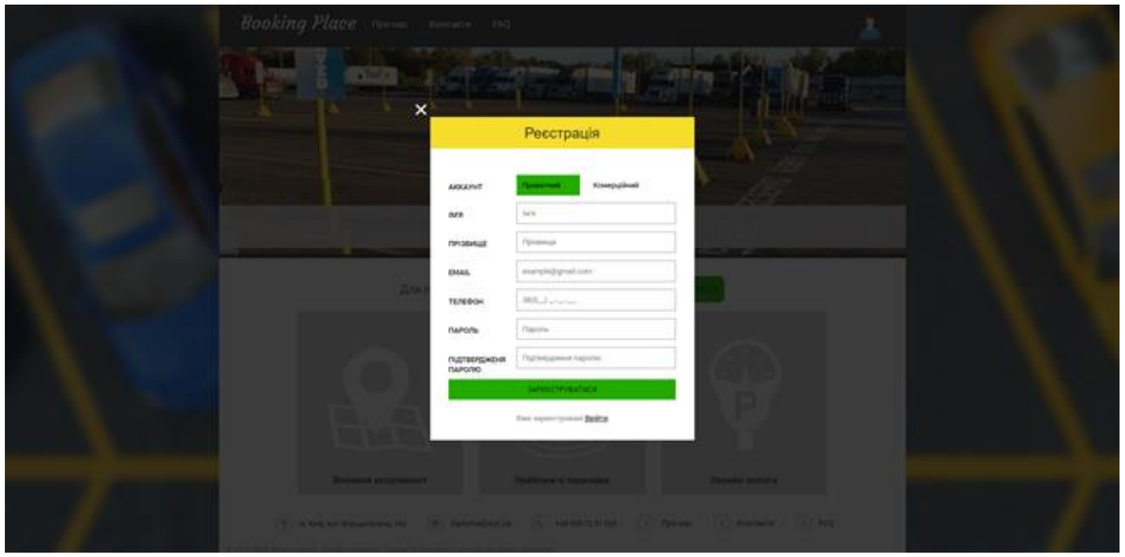
					ДП 6310.05.000 ССП		
Зм.	Арк.	№ документа	Підпис	Дата	*Схема структурна пакетів*	Літера	Маса / Масштаб
Розробив		Іщенко В.С.					
		Левчук В.І.				Аркуш 1	Аркушів 1
Перевірив		Сперкач М.О.					
Т. кон.					*Інформаційна система підтримки процесу автомобільних паркувань*	*КПІ ім. Ігоря Сікорського Каф. АСОІУ Гр. ІС-63*	
Н. кон.		Проскура С.Л.					
Затвердив		Сперкач М.О.					

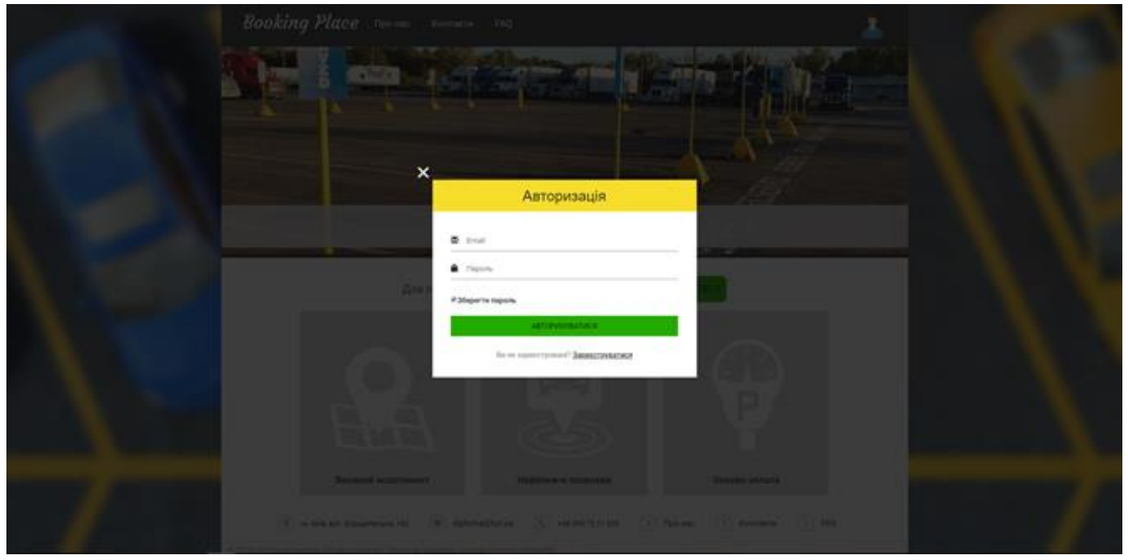
					ДП 6310.06.000 ССР		
Зм.	Арк.	№ документа	Підпис	Дата	*Схема структурна розгортання*	Літера	Маса / Масштаб
Розробив		Іщенко В.С.					
		Левчук В.І.				Аркуш 1	Аркушів 1
Перевірив		Сперкач М.О.					
Т. кон.					*Інформаційна система підтримки процесу автомобільних паркувань*	*КПІ ім. Ігоря Сікорського Каф. АСОІУ Гр. ІС-63*	
Н. кон.		Проскура С.Л.					
Затвердив		Сперкач М.О.					

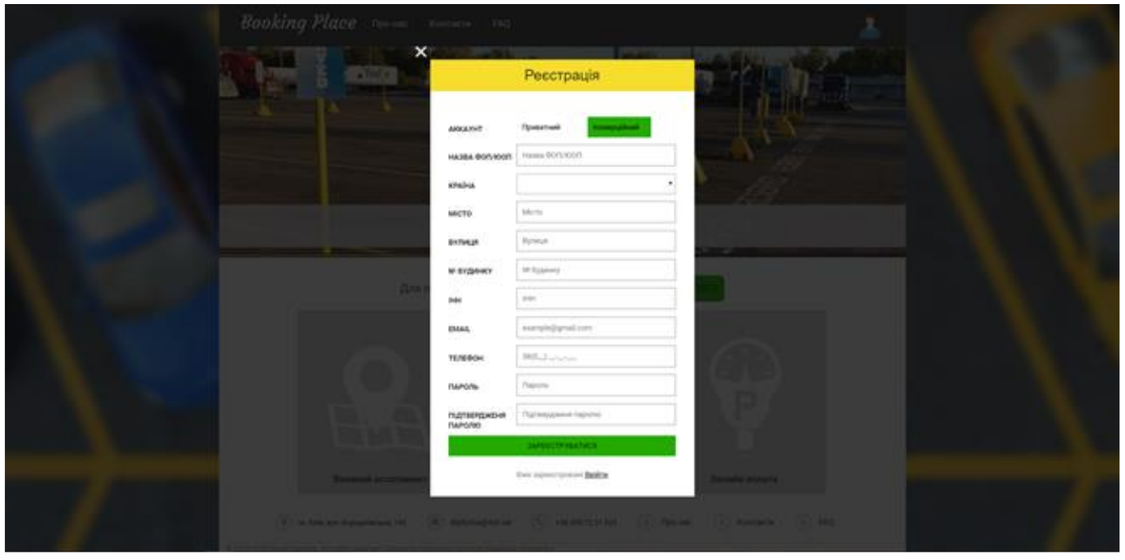


					ДП 6310.07.000 ССК		
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна компонентів	Літера	Маса / Масштаб
Розробив		Іщенко В.С.					
		Левчук В.І.				Аркуш 1	Аркушів 1
Перевірив		Сперкач М.О.					
Т. кон.					Інформаційна система підтримки процесу автомобільних паркувань	КПІ ім. Ігоря Сікорського Каф. АСОІУ Гр. ІС-63	
Н. кон.		Проскура С.Л.					
Затвердив		Сперкач М.О.					









					ДП 6310.08.000 КЕ		
						Літера	Маса / Масштаб
Зм.	Арк.	№ документа	Підпис	Дата	Креслення вигляду екранних форм		
Розробив		Іщенко В.С.				Аркуш 1	Аркушів 1
		Левчук В.І.					
Перевірив		Сперкач М.О.					
Т. кон.					Інформаційна система підтримки процесу автомобільних паркувань	КПІ ім. Ігоря Сікорського Каф. АСОІУ Гр. ІС-63	
Н. кон.		Проскура С.Л.					
Затвердив		Сперкач М.О.					

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
Кафедра автоматизованих систем обробки інформації і управління

УДК: 656.13

До захисту допущено:
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (вл.ім'я, прізвище)
"___"_____2020 р.

Дипломний проєкт
на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інформаційні управляючі системи та технології»
спеціальності 122 «Комп'ютерні науки та інформаційні технології»

на тему: *«Інформаційна система підтримки процесу автомобільних паркувань» (комплексна тема)*
Підсистема з керування кабінетом власника майданчиків автомобільних паркувань
Індивідуальна частина №2

Виконав: студент *IV* курсу, групи *ІС-63*
Левчук Володимир Іванович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник *доц., к.т.н. Сперкач Майя Олегівна* _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з графічної документації *ст.викл. Проскура Світлана Леонідівна* _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент *доц., к.т.н., доц. Писаренко Андрій Володимирович* _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) *інформатики та обчислювальної техніки*
(повна назва)

Кафедра *автоматизованих систем обробки інформації і управління*
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки (програма професійного спрямування) *6.050101*

«Комп’ютерні науки» («Інформаційні управляючі системи та технології»)

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр Павлов*
(підпис) (вл.ім’я, прізвище)
“___”_____2020 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЄКТ СТУДЕНТУ

Левчуку Володимиру Івановичу
(прізвище, ім’я, по батькові)

1. Тема проєкту *«Інформаційна система підтримки процесу автомобільних паркувань» (комплексна тема)*
Підсистема з керування кабінетом власника майданчиків автомобільних паркувань
Індивідуальна частина №2

керівник проєкту *Сперкач Майя Олегівна, к.т.н., доцент*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від “07” травня 2020 р. №1081-с

2. Термін подання студентом проєкту “*01*”*червня* 2020 року

3. Вихідні дані до проєкту

Технічне завдання

4. Зміст пояснювальної записки

1. Загальні положення: основні визначення та терміни, опис предметного середовища, огляд існуючих аналогів, постановка задачі

2. Інформаційне забезпечення: вхідні дані, вихідні дані, опис структури бази даних

3. Технологічний розділ: керівництво користувача, випробування програмного продукту

5. Перелік графічного матеріалу

Схема структурна послідовності

Схема структурна станів

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання *«1» грудня 2019 року*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	*Вивчення рекомендованої літератури*	*15.12.2019*	
2.	*Аналіз існуючих методів розв'язання задачі*	*30.12.2019*	
3.	*Постановка та формалізація задачі*	*10.01.2020*	
4.	*Розробка інформаційного забезпечення*	*20.01.2020*	
5.	*Алгоритмізація задачі*	*10.02.2020*	
6.	*Обґрунтування використовуваних технічних засобів*	*15.02.2020*	
7.	*Розробка програмного забезпечення*	*07.04.2020*	
8.	*Налагодження програми*	*20.04.2020*	
9.	*Виконання графічних документів*	*30.04.2020*	
10.	*Оформлення пояснювальної записки*	*13.05.2020*	
11.	*Подання ДП на попередній захист*	*15.05.2020*	
12.	*Подання ДП на основний захист*	*01.06.2020*	
13.	*Подання ДП рецензенту*	*02.06.2020*	

Студент _____ Володимир ЛЕВЧУК
(підпис)

Керівник проєкту _____ Майя СПЕРКАЧ
(підпис)

Пояснювальна записка
до дипломного проєкту

на тему: «Інформаційна система підтримки процесу автомобільних паркувань» (комплексна тема)

Підсистема керування кабінетом власника майданчиків автомобільних паркувань

Індивідуальна частина №2

Київ – 2020 року

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка дипломного проєкту складається з трьох розділів, містить 31 рисунок, 10 таблиць, 1 додаток, 3 джерела.

Індивідуальна частина №2 присвячена розробці підсистеми керування кабінетом власника майданчиків автомобільних паркувань.

У розділі «Математичне забезпечення» було сформульовано змістовну постановку задачі, визначено чітку математичну постановку задачі. Також обґрунтовано метод розв’язання задачі і детально описано її метод розв’язання.

У розділі «Програмне та технічне забезпечення» було побудовано діаграми послідовності та станів, наведено специфікації функцій для серверної і клієнтської частини підсистеми.

У технологічному розділі було розписано керівництво користувача. Було проведено випробування для перевірки відповідності функцій підсистеми вимогам технічного завдання.

ШИФРУВАННЯ, AES, GDPR, СИСТЕМА, ПАРКОВКА, БРОНЮВАННЯ

ABSTRACT

Structure and scope of work. The explanatory note to the diploma consists of three sections, contains 31 drawings, 10 tables, 1 application, 3 sources.

Individual part №2 is dedicated to the development of the subsystem for managing the personal account of the parking's owner.

The "Software" section gives a meaningful statement and defines a clear mathematical statement of the task. The task solvation method has been determined and described in detail.

The "Software and hardware" section shows the sequence and state diagrams, provides the function specifications for the server and client parts of the subsystem.

The technology section describes the user manual. There was conducted a test to verify compliance of the subsystem functions with the requirements of the technical specifications.

ENCRYPTION, AES, GDPR, SYSTE, PARKING, BOOKING

ЗМІСТ

1 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

1.1 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»[1] власники системи відповідають перед законом про цілісність і конфіденційність інформації. Для цього існує стандарт GDPR (General Data Protection Regulation)[2] – 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.

1.2 Математична постановка задачі

1.2.1 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.

1.3 Обґрунтування методу розв’язання

Для розв’язку задачі 1.2.1 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 Опис методу розв'язання

1.4.1 Задача шифрування даних за допомогою алгоритму AES

Advanced Encryption Standard (AES) [2] 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 1.1 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	3.31×1056 років

Рисунок 1.1 – 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,…,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 $w_0=k_0k_1k_2k_3$, $w_1= k_4k_5k_6k_7$ і т.д. Розбиття ключа показано на рисунку 1.2.

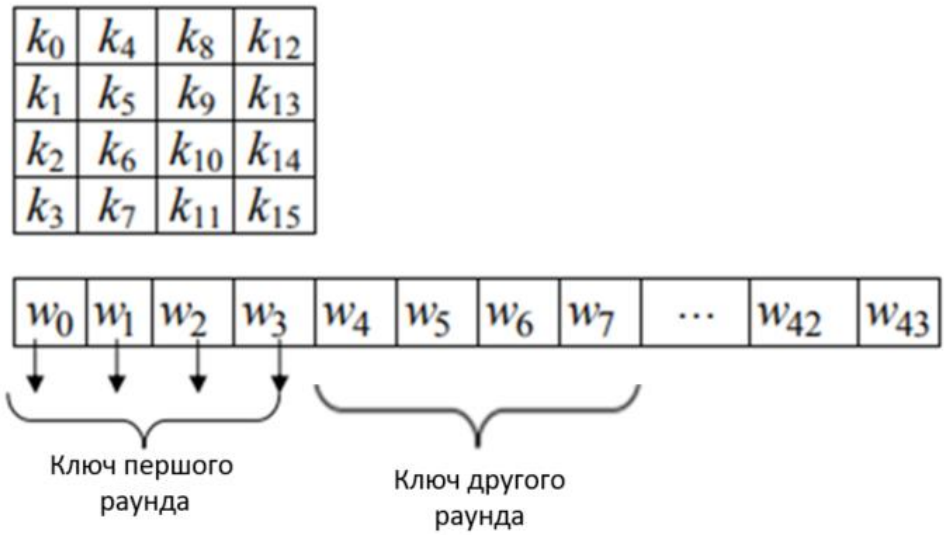


Рисунок 1.2 – Розбиття ключа

Кількість раундів роботи алгоритму залежить від довжини ключа. Для ключа довжиною 128 біт, кількість раундів становить 10. На рисунку 1.3 зображено схему перетворення даних в 10 раундів.

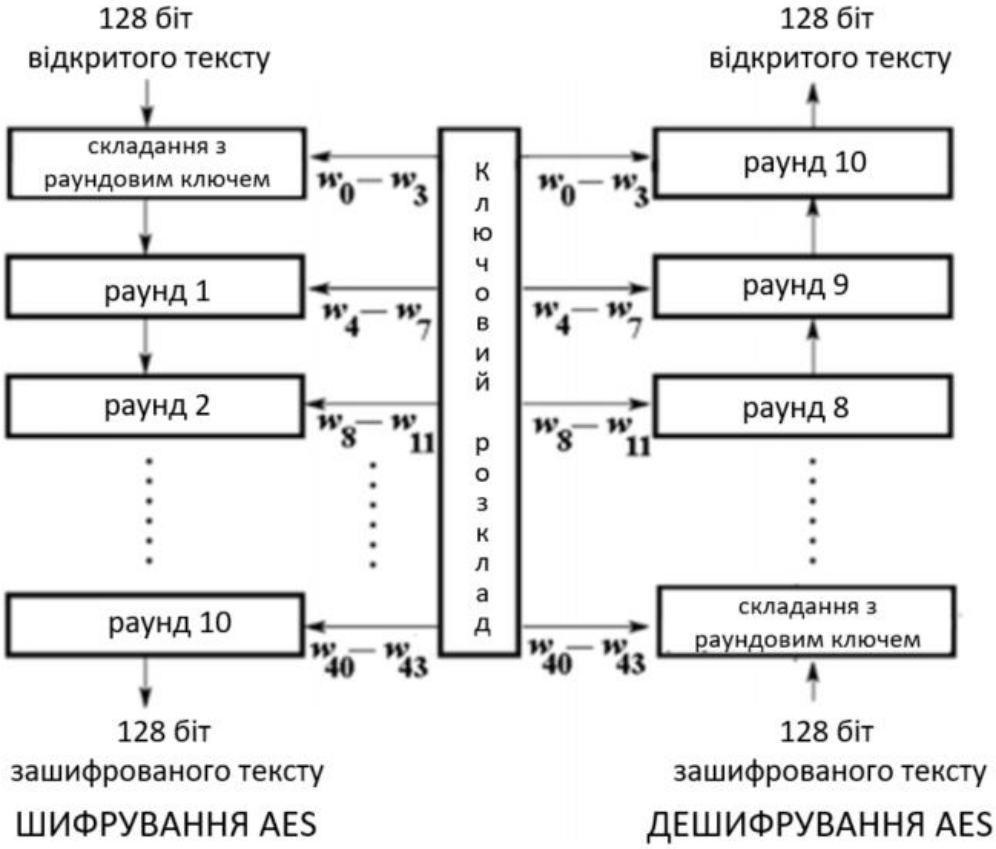


Рисунок 1.3 – Схема перетворення даних алгоритму AES

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 1.4.

Рисунок 1.4 – Процедура *SubBytes*

– *ShiftRows* – побайтовий здвиг рядків матриці *State* на різну кількість байт, де *State* – матриця, зо описує дані на вході та виході раунду. На рисунку 1.5 показано дане перетворення.

Рисунок 1.5 – Процедура *ShiftRows*

– *MixColumns* – перемішування байт в стовпцях. На рисунку 1.6 показано схематичне зображення цього перетворення.

Рисунок 1.6 – Процедура *MixColumns*

– *AddRoundKey* – складання з раундовим ключем (операція XOR). Графічне представлення даного перетворення показано на рисунку 1.7.

Рисунок 1.7 – 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.

Висновок 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 підсистеми. 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.

2 ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Засоби розробки

Засоби розробки наведені у відповідному розділі загальної частини даного проєкту.

2.2 Архітектура програмного забезпечення

2.2.1 Діаграма послідовності

У частині дипломного проєкту «Графічний матеріал» представлена діаграма послідовності, яка описує послідовність дій системи за часом. На якій вертикальні лінії, це об’єкти системи, які взаємодіють між собою і зв'язок показаний у формі повідомлень між ними. На діаграмі представлено повний процес створенння нової парковки від самої реєстрації власника парковок. 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.

2.2.2 Діаграма станів

У частині дипломного проєкту «Графічний матеріал» представлена діаграма станів, яка показує зміну станів об’єктів системи в часі. Дана діаграма поянює яким чином працює дана підсистема. На ній показані ключові стани системи, які відбуваються для створенння нової парковки, починаючи від входу на сайт і реєстрації власника парковок. Після успішної реєстрації і, за потреби, редагування персональної інформації, відобраються стани додавання парковки – це саме додавання парковки і додавання документу до парковки.

2.2.3 Специфікація функцій

У таблиці 1.1 наведено перелік методів які відносяться до серверної частини системи з вказанням до яких класів вони належать і їх детальним описом.

Таблиця 2.1 – Специфікація методів серверної частини системи

Клас	Сигнатура	Опис
SupplierManagement Controller	getSupplier()	Отримання інформації про власника парковок
	updateSupplierInfo(SupplierBase supplier)	Оновлення інформації про власника парковок
SupplierRepositoryI mpl	existWithDocument(Str ing documentReference, String email)	Перевірка на існування документа у власника парковок
SupplierService	getSupplierWithAllInfo(String username)	Отримання детальної інформації про власника парковок
	updateSupplierInfo(SupplierB ase supplierBase)	Оновлення інформації про власника парковок
Supplier	equals(Object obj)	Порівняння двох об’єктів
	hashCode()	Генерація Hash коду
ParkingController	getSupplierParkings()	Отримання парковок власника парковок
	getParkingByReference(String reference)	Отримання парковки
	createParking(ParkingRequest parking)	Створення парковки
	updateParking(String reference, ParkingRequest parking)	Оновлення інформації про парковку
	deleteParking(String reference)	Видалення парковки
	deleteParkingPlace(String placeReference)	Видалення паркомісць на парковці
	getParkingsPriceHistory(Strin g reference)	Отримання історичних даних цін парковок

Продовження таблиці 2.1

Клас	Сигнатура	Опис
	getParkingOrders(String reference, Integer pageNumber, Integer pageSize, LocalDate fromDate, LocalDate toDate)	Отримання замовлень для парковки
ParkingProvider	getParkingByReference(String reference)	Отримання парковки
ParkingService	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

Продовження таблиці 2.1

Клас	Сигнатура	Опис
DocumentController	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
FileManager	saveFile(String path, MultipartFile multipartFile)	Збереження файлу
DocumentService	getParkingDocumentTypes()	Отримання типів документів для парковок
	getSupplierDocumentTypes()	Отримання типів документів для власника парковок
	getDocumentTypesByPredicate(Predicate where)	Отримання документів за певною умовою

Продовження таблиці 2.1

Клас	Сигнатура	Опис
	uploadParkingDocument(String parkingReference, MultipartFile parkingDocument, String documentTypeCode)	Завантаження документу для парковки
	uploadSupplierDocument(MultipartFile supplierDocument, String documentTypeCode)	Завантаження документу для власника парковок
	uploadDocument(Document document, MultipartFile multipartFile)	Завантаження документу
	createDocument(String uniqueId, MultipartFile document, String documentTypeCode)	Створення документу
	deleteDocumentByReference(String documentReference)	Видалення документу
AddressProvider	getAddress(com.kpi.parking.model.Address address)	Отримання адреси
AddressRepositoryImpl	isAddressLinkedToActiveParking(String country, String city, String street, String streetNumber)	Перевірка існування адреси для парковки

В таблиці 2.2 наведено перелік методів які відносяться до клієнтської частити системи з вказанням до яких класів вони належать і детальним їх описом.

Таблиця 2.2 – Специфікація методів клієнтської частини системи

Клас	Сигнатура	Опис
CountryService	getCountries()	Отримання списку країн
SupplierService	getInfo()	Отримання інформації про власника парковок

Продовження таблиці 2.2

Клас	Сигнатура	Опис
	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
DocumentService	getSupplierDocumentTypes()	Отримання типів документів для підтвердження акаунта власника парковок
	getParkingDocumentTypes()	Отримання типів документів для підтвердження парковки
	addParkingDocument(formData, reference)	Додавання до парковки документу
	addSupplierDocument(formData)	Додавання документа до акаунта власника парковок

Продовження таблиці 2.2

Клас	Сигнатура	Опис
	deleteSupplierDocument(reference)	Видалення документа для підтвердження акаунта власника парковок
	deleteParkingDocument(parkingReference, documentReference)	Видалення документа для парковки
OrderService	getOrders(filters)	Отримання списку бронювань
RequestService	post(url, body)	Запит POST
	postFileUpload(url, body)	Запит POST для надсилання файлу
	get(url, isheaders = false, query = null)	Запит GET
	put(url, body)	Запит PUT
	delete(url)	Запит DELETE
TimeService	calculateTimeDuration(from, to)	Обчислення різниці часу
	dateToString(date)	Форматування дати
AccountComponent	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

Продовження таблиці 2.2

Клас	Сигнатура	Опис
	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
MyParkingComponent	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

Продовження таблиці 2.2

Клас	Сигнатура	Опис
	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
OrderComponent	getCountry()	Отримання списку країн
	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

Продовження таблиці 2.2

Клас	Сигнатура	Опис
	hideLoader()	Зупинка анімації

Висновок до розділу

В даному розділі було наведено діаграми послідовності та станів з коротким описом. На двох діаграмах показаний процес створення нової парковки починаючи від реєстрації власника парковок в системі. Також розписано специфікації функцій серверної та клієнтської частин підсистеми керування кабінетом власника майданчиків автомобільних паркувань. Для кожно

3 ТЕХНОЛОГІЧНИЙ РОЗДІЛ

3.1 Керівництво користувача

Так як реєстрація і авторизація були описані в загальній частині, вважатимемо, що власник парковок зареєстрований в системі.

Після успішної авторизації, власник парковок має доступ до власного акаунта наступним сторінками, які показані на рисунку 3.1. Також він переходити на інші сторінки, через меню, яке випадає при наведенні на його назву компанії, як це показано на рисунку 3.2.

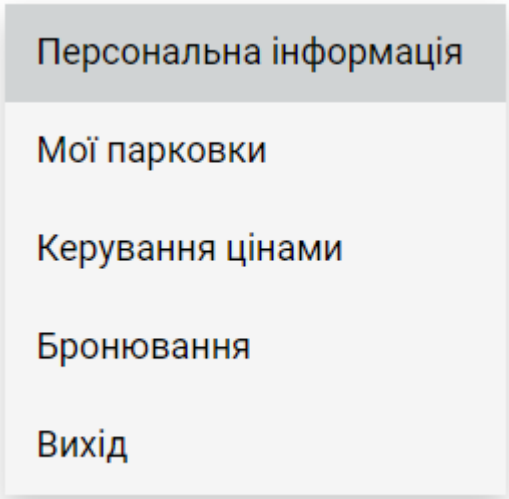


Рисунок 3.1 – Меню кабінету власника парковок

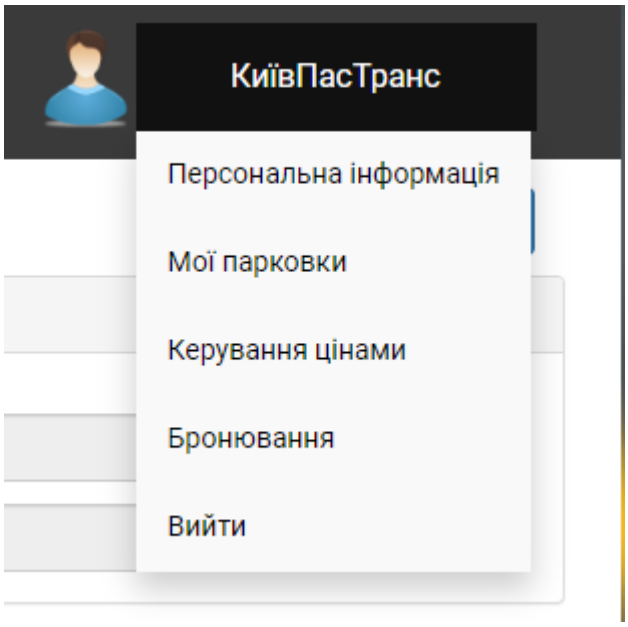


Рисунок 3.2 – Випадне меню кабінету власника парковок

3.1.1 Сторінка «Персональна інформація»

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

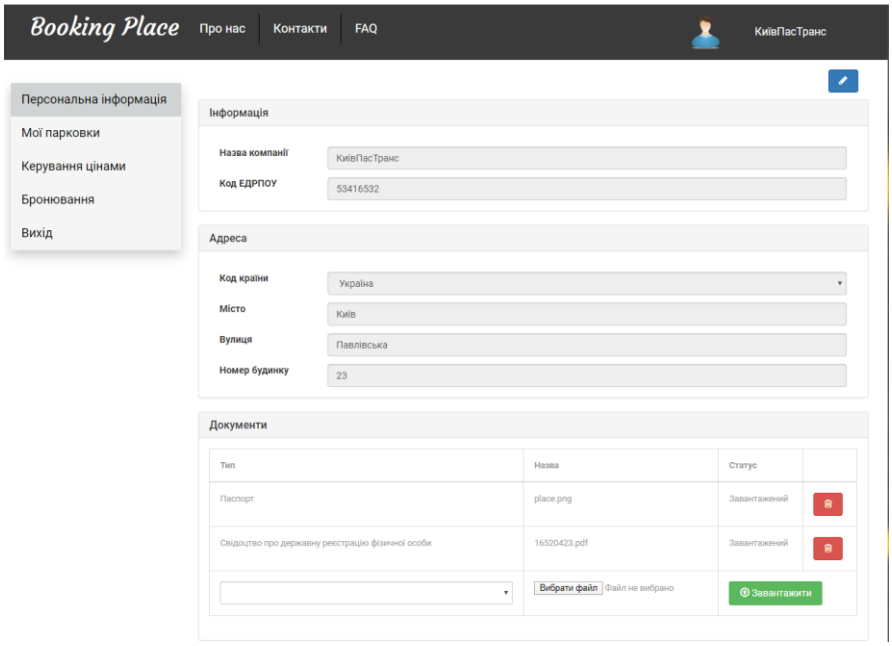


Рисунок 3.3 – Сторінка персональної інформації власника парковок

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

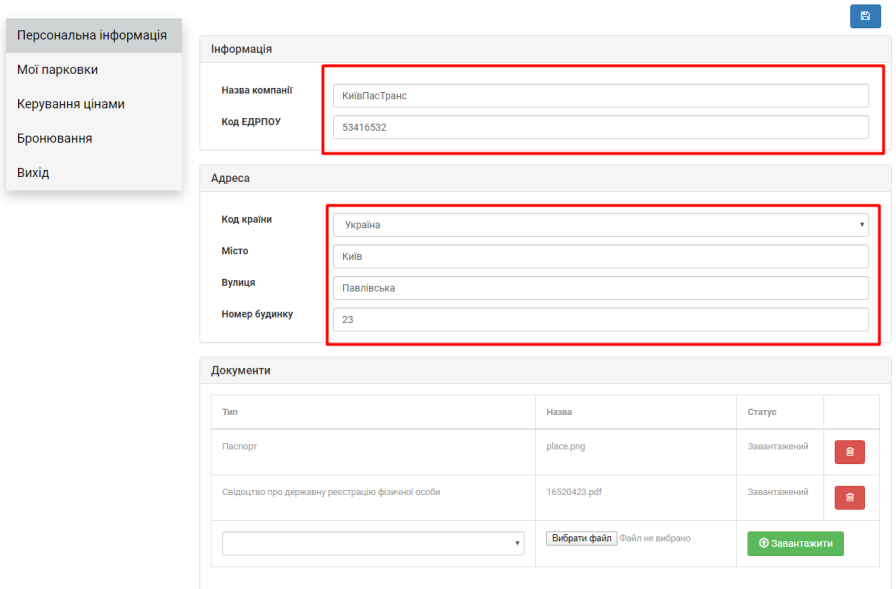


Рисунок 3.4 – Редагування персональної інформації

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.

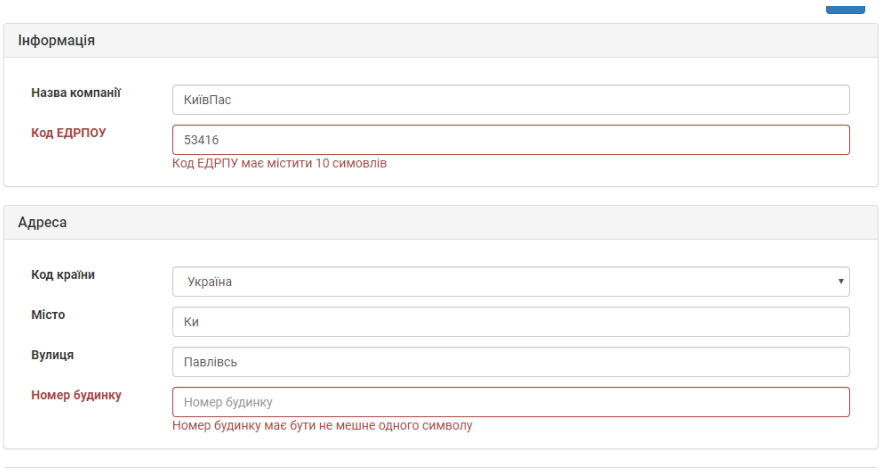


Рисунок 3.5 – Помилки під час неправильно заповнених даних

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

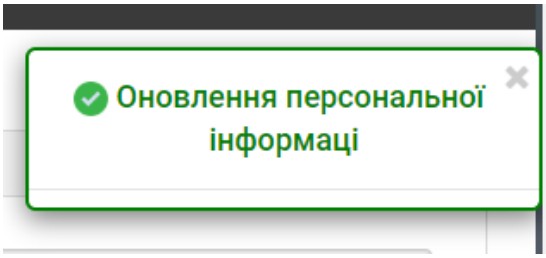


Рисунок 3.6 – Повідомлення про успішне редагування персональної інформації

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

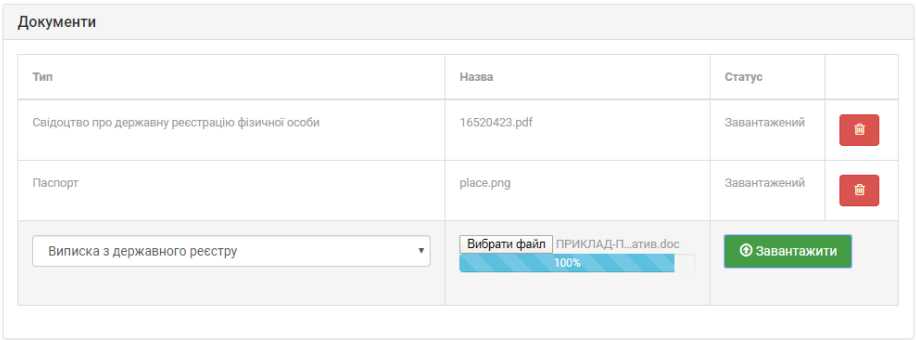


Рисунок 3.7 – Завантаження документів до кабінету власника парковок

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.

Рисунок 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.

Рисунок 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.

3.1.2 Сторінка «Мої парковки»

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

Рисунок 3.10 – Список парковок власника парковок

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

Рисунок 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

Рисунок 3.12 – Виведення помилок для некоректно заповнених полів

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

Рисунок 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.

Рисунок 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

Рисунок 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.

Рисунок 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.

Рисунок 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

3.1.3 Сторінка «Керування цінами»

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

Рисунок 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.

Рисунок 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.

Рисунок 3.21 – Додаткові опції для графіку

3.1.4 Сторінка «Бронювання»

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.

Рисунок 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.

Рисунок 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, яка показана на рисунку 3.23.

Рисунок 3.24 – Детальна інформація про бронювання

3.2 Випробування програмного продукту

3.2.1 Мета випробувань

Метою випробування є перевірка коректності роботи системи відповідно до вимого технічного завдання.

3.2.2 Загальні положення

Випробування проводились на основі наступних документів:

- ГОСТ 34.603–92. Інформаційна технологія. Види випробувань автоматизованих систем;
- ГОСТ РД 50–34.698–90. Автоматизовані системи вимог до змісту документів.

3.2.3 Результати випробувань

В таблицях 3.1 – 3.11 наведені тести випробування застосунку.

Таблиця 3.1 – Тест випробування редагування персональної інформації власника парковок

Назва:	Тест редагування персональної інформації власника парковок	
Функція/UseCase:	Ведення кабінету власника парковок	
Дія: Власник парковок веде власний кабінет	**Очікуваний результат:** Збережена, введена інформація, власником парковок	**Результат тесту:** • пройдений • провалений • заблокований
Передумова:		
Перейти до системи за посиланням: https://booking-place.herokuapp.com/	Відкрита головна сторінка системи	Пройдений
Авторизуватися в системі	Відкрита сторінка «Персональна інформація»	Пройдений
Кроки тесту:		
Натиснути на кнопку редагування інформації	Всі поля форми доступні для редагування	Пройдений
Натиснути на кнопку збереження інформації	Інформація збережена, поля заблоковані для редагування, з’явилося повідомлення про збереження даних	Пройдений
Післяумова:		
Навести курсор мишки на своє ім’я біля значка користувача	Розкривний список з посиланнями відкритий	Пройдений
Натиснути на кнопку «Вийти»	Відкрита головна сторінка системи	Пройдений

Таблиця 3.2 – Тест випробування завантаження документів для власника парковок

Назва:	Тест завантаження документів для власника парковок	
Функція/UseCase:	Завантаження документів для власника парковок	
Дія: Власник парковок завантажує документи про персональну інформацію	**Очікуваний результат:** Документи завантажені	**Результат тесту:** • пройдений • провалений • заблокований
Передумова:		
Перейти до системи за посиланням: https://booking-place.herokuapp.com/	Відкрита головна сторінка системи	Пройдений
Авторизуватися в системі	Відкрита сторінка «Персональна інформація»	Пройдений
Кроки тесту:		
Вибрати тип документу	Тип документу вибраний	Пройдений
Вибрати документ	Файл вибрано	Пройдений
Натиснути на кнопку «Завантажити»	Файл завантажено, з’явилося повідомлення про успішне завантаження документу	Пройдений
Виконати дії для всіх необхідних документів	Всі документи завантажені	Пройдений
Післяумова:		
Навести курсор мишки на своє ім’я біля значка користувача	Розкривний список з посиланнями відкритий	Пройдений
Натиснути на кнопку «Вийти»	Відкрита головна сторінка системи	Пройдений

Таблиця 3.3 – Тест випробування створення парковки

Назва:	Тест створення парковки	
Функція/UseCase:	Створення парковки	
Дія: Власник парковок створює нову парковку	**Очікуваний результат:** Створена нова парковка	**Результат тесту:** • пройдений • провалений • заблокований
Передумова:		
Перейти до системи за посиланням: https://booking-place.herokuapp.com/	Відкрита головна сторінка системи	Пройдений
Авторизуватися в системі	Відкрита сторінка «Персональна інформація»	Пройдений
Перейти на сторінку «Мої парковки»	Відкрита сторінка зі списком парковок	Пройдений
Кроки тесту:		
Натиснути на кнопку «+»(додавання парковки)	Відкрита порожня форма для додавання парковки	Пройдений
Заповнити всі доступні поля	Всі поля заповнені коректно	Пройдений
Натиснути на кнопку «Зберегти»	Інформація збережена, додана нова парковка, з’явилося повідомлення про створення нової парковки	Пройдений

Продовження таблиці 3.3

Післяумова:		
Навести курсор мишки на своє ім’я біля значка користувача	Розкривний список з посиланнями відкритий	Пройдений
Натиснути на кнопку «Вийти»	Відкрита головна сторінка системи	Пройдений

Таблиця 3.4 – Тест випробування редагування даних про парковку

Назва:	Тест редагування даних про парковку	
Функція/UseCase:	Редагування даних про парковку	
Дія: Власник парковок редагує інформацію про парковку	**Очікуваний результат:** Відредагована інформація про парковку	**Результат тесту:** • пройдений • провалений • заблокований
Передумова:		
Перейти до системи за посиланням: https://booking-place.herokuapp.com/	Відкрита головна сторінка системи	Пройдений
Авторизуватися в системі	Відкрита сторінка «Персональна інформація»	Пройдений
Перейти на сторінку «Мої парковки»	Відкрита сторінка зі списком парковок	Пройдений
Кроки тесту:		
Натиснути на кнопку редагування відповідної парковки	Відкрита форма з інформацією про парковку	Пройдений

Продовження таблиці 3.4

Відредагувати необхідну інформацію	Інформація змінена у формі на коректну	Пройдений
Натиснути на кнопку «Зберегти»	Форма закрита, інформація відредагована, з'явилося повідомлення про оновлення інформації	Пройдений
Післяумова:		
Навести курсор мишки на своє ім'я біля значка користувача	Розкривний список з посиланнями відкритий	Пройдений
Натиснути на кнопку «Вийти»	Відкрита головна сторінка системи	Пройдений

Таблиця 3.5 – Тест випробування завантаження документів до парковки.

Назва:	Тест завантаження документів до парковки	
Функція/UseCase:	Завантаження документів до парковки	
Дія: Власник парковок завантажує документи до парковки	**Очікуваний результат:** Збережені завантажені документи до парковки	**Результат тесту:** • пройдений • провалений • заблокований
Передумова:		
Перейти до системи за посиланням: https://booking-place.herokuapp.com/	Відкрита головна сторінка системи	Пройдений
Авторизуватися в системі	Відкрита сторінка «Персональна інформація»	Пройдений

Продовження таблиці 3.5

Перейти на сторінку «Мої парковки»	Відкрита сторінка зі списком парковок	Пройдений
Кроки тесту:		
Натиснути на кнопку редагування інформації	Всі поля форми доступні для редагування	Пройдений
Вибрати тип документу	Тип документу вибраний	Пройдений
Вибрати документ	Файл вибрано	Пройдений
Натиснути на кнопку «Завантажити»	Файл завантажено, з’явилося повідомлення про успішне завантаження документу	Пройдений
Післяумова:		
Закрити форму парковки	Форма закрита	Пройдений
Навести курсор мишки на своє ім’я біля значка користувача	Розкривний список з посиланнями відкритий	Пройдений
Натиснути на кнопку «Вийти»	Відкрита головна сторінка системи	Пройдений

Таблиця 3.6 – Тест випробування зміни тарифів на паркування

Назва:	Тест зміна тарифів на паркування	
Функція/UseCase:	Зміна тарифів на паркування	
Дія: Власник парковок змінює тариф на паркування на парковці	**Очікуваний результат:** Змінений тариф паркування на парковці	**Результат тесту:** • пройдений • провалений • заблокований
Передумова:		

Продовження таблиці 3.6

Перейти до системи за посиланням: https://booking-place.herokuapp.com/	Відкрита головна сторінка системи	Пройдений
Авторизуватися в системі	Відкрита сторінка «Персональна інформація»	Пройдений
Перейти на сторінку «Мої парковки»	Відкрита сторінка зі списком парковок	Пройдений
Кроки тесту:		
Натиснути на кнопку редагування відповідної парковки	Відкрита форма парковки з її інформацією	Пройдений
Змінити тариф паркування	Тариф паркування змінено	Пройдений
Натиснути на кнопку «Зберегти»	Форма закрита, тариф парковки відредаговано, з’явилося повідомлення про оновлення інформації	Пройдений
Післяумова:		
Навести курсор мишки на своє ім’я біля значка користувача	Розкривний список з посиланнями відкритий	Пройдений
Натиснути на кнопку «Вийти»	Відкрита головна сторінка системи	Пройдений

Таблиця 3.7 – Тест випробування перегляду динаміки зміни тарифів.

Назва:	Тест перегляд динаміки зміни тарифів	
Функція/UseCase:	Перегляд динаміки зміни тарифів	
Дія: Власник парковок переглядає динаміку змін тарифів	**Очікуваний результат:** Виведена динаміка зміни тарифів парковки	**Результат тесту:** • пройдений • провалений • заблокований
Передумова:		
Перейти до системи за посиланням: https://booking-place.herokuapp.com/	Відкрита головна сторінка системи	Пройдений
Авторизуватися в системі	Відкрита сторінка «Персональна інформація»	Пройдений
Перейти на сторінку «Керування цінами»	Відкрита сторінка зі списком парковок	Пройдений
Кроки тесту:		
Натиснути на вибрану парковку	Відкритий блок з побудованим графіком динаміки змін тарифів парковки	Пройдений
Післяумова:		
Навести курсор мишки на своє ім’я біля значка користувача	Розкривний список з посиланнями відкритий	Пройдений
Натиснути на кнопку «Вийти»	Відкрита головна сторінка системи	Пройдений

Таблиця 3.8 – Тест випробування перегляду історії бронювань по парковках

Назва:	Тест перегляд історії бронювань по парковках	
Функція/UseCase:	Перегляд історії бронювань по парковках	
Дія: Власник парковок переглядає бронювання по парковках	**Очікуваний результат:** Виведені бронювання по парковках	**Результат тесту:** • пройдений • провалений • заблокований
Передумова:		
Перейти до системи за посиланням: https://booking-place.herokuapp.com/	Відкрита головна сторінка системи	Пройдений
Авторизуватися в системі	Відкрита сторінка «Персональна інформація»	Пройдений
Перейти на сторінку «Бронювання»	Відкрита сторінка бронювань	Пройдений
Кроки тесту:		
Вибрати парковку і вказати період	Парковка вибрана і вказаний період пошуку	Пройдений
Натиснути на кнопку «Пошук»	Виведені бронювання для вибраної парковки за вказаний період	Пройдений
Післяумова:		
Навести курсор мишки на своє ім’я біля значка користувача	Розкривний список з посиланнями відкритий	Пройдений
Натиснути на кнопку «Вийти»	Відкрита головна сторінка системи	Пройдений

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.

Було наведено опис тестів та порядок їх виконання для перевірки відповідності роботи підсистеми функціональним вимогам, представленим у технічному завданні. Так як всі тести пройшли успішне тестування, можна робити висновок, що підсистема відповідає поставленим завданням.

ЗАГАЛЬНІ ВИСНОВКИ

У рамках індивідуальної частини №2 даного диплому була розроблена підсистема 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.

У розділі математично забезпечення 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.

У розділі «Програмне та технічне забезпечення» наведено діаграми послідовності та станів з їх коротким описом. Також було наведено специфікацію методів серверної і клієнтської частини підсистеми з керування кабінетом власника майданчиків автомобільних паркувань.

У технологічному розділі наведено керівництво користувача з описом всіх важливих функціональних елементів підсистеми. Було зроблено низку тестів та їх порядок виконання для перевірки коректності роботи підсистеми. За результатами всіх тестів, можна робити висновок, що розроблений функціонал відповідає функціональним вимогам і він в повній мірі реалізований для актора «власник парковок».

На даному етапі розробки, підсистема готова до бета-тестування реальними власниками парковок. Але необхідно ще доопрацювати низку важливих аспектів. Одним з яким є платіжна система. Гроші за бронювання паркомісць власники парковок мають отримувати на власні рахунки.

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 ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. ЗАКОН УКРАЇНИ Про захист персональних даних [Електронний ресурс] // Режим доступу: https://zakon.rada.gov.ua/laws/show/2297-17
2. GDPR [Електронний ресурс] // Режим доступу: https://dictionary.cambridge.org/ru/словарь/английский/gdpr
3. Advanced Encryption Standard [Електронний ресурс] // Режим доступу: https://ru.wikipedia.org/wiki/Advanced_Encryption_Standard

ДОДАТОК А

Тексти програмного коду

Підсистема з керування кабінетом власника майданчиків автомобільних паркувань

(Найменування програми (документа))

DVD-R

(Вид носія даних)

37 арк, 236 Кб

(Обсяг програми (документа), арк., Кб)

Київ – 2020 року

Вихідні коди серверної частини підсистеми:

```
@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)).ge
t();
        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 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.getCurrentUsername());

        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 findWithAllInformationByUserEmail(@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.findWithAllInformationByUserEmail(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(St
andardCharsets.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;
}
```

Вихідні коди клієнтської частини підсистеми:

```
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 інформаці'],
    ['createParking', 'Створення парковки'],
    ['updateParking', 'Оновлення інформації парковки'],
    ['deleteParking', 'Видалення парковки'],
    ['deleteParkingPlace', 'Видалення паркомісця'],
    ['uploadFile', 'Завантаження документу'],
    ['deleteSupplierDocument', 'Видалення документу'],
    ['updateAccountCust', 'Оновлення персональної інформаці'],
    ['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',
    'Грудня',
  ];

  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() + 'р ' + 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("Мої 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);
          }
        );
    });
  }

  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: [];
    inizializedParking = 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();
    }

  }
```

UNICHECK

Власник документу:
Попенко Володимир Дмитрович

Дата перевірки:
04.06.2020 18:32:07 EEST

Дата звіту:
04.06.2020 20:33:07 EEST

ID перевірки:
1003783677

Тип перевірки:
Doc vs Internet + Library

ID користувача:
77149

Назва документу: Levchuk_bachelor_Individual

ID файлу: 1003798190 **Кількість сторінок:** 42 **Кількість слів:** 4729 **Кількість символів:** 36980 **Розмір файлу:** 1.39 MB

15.5% Схожість

Найбільша схожість: 9.58% з джерело бібліотеки. **ID файлу:** 1003798210

5.82% Схожість з Інтернет джерелами	106	Page 44
15.5% Текстові збіги по Бібліотеці акаунту	165	Page 44

1.02% Цитат

Цитати	2	Page 45

Вилучення переліку посилань вимкнено

0% Вилучень

Вилучений текст відсутній

Підміна символів

Не знайдено замінених символів

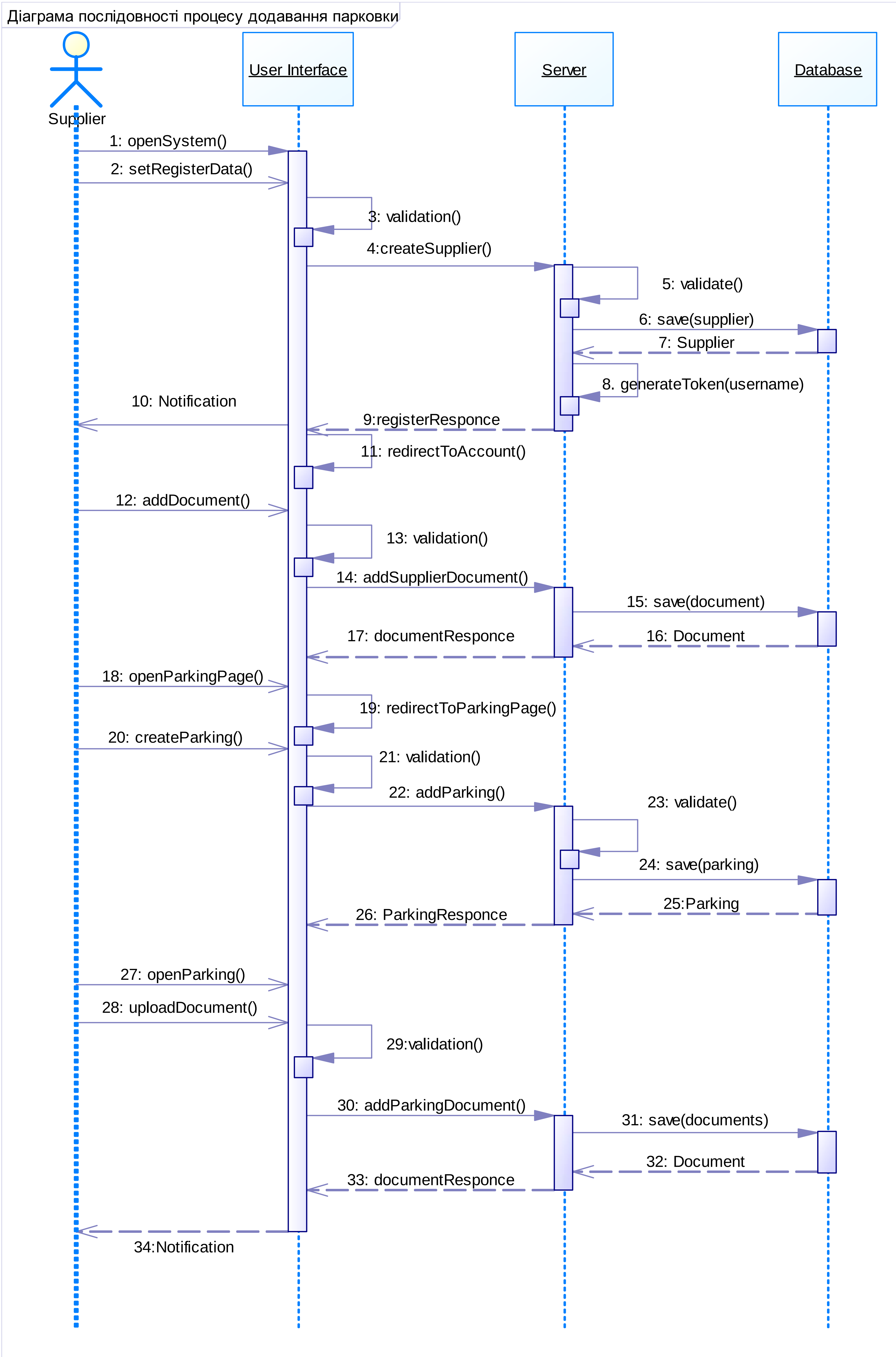
Графічний матеріал
до дипломного проєкту

на тему: «Інформаційна система підтримки процесу автомобільних паркувань» (комплексна тема)

Підсистема керування кабінетом власника майданчиків автомобільних паркувань

Індивідуальна частина №2

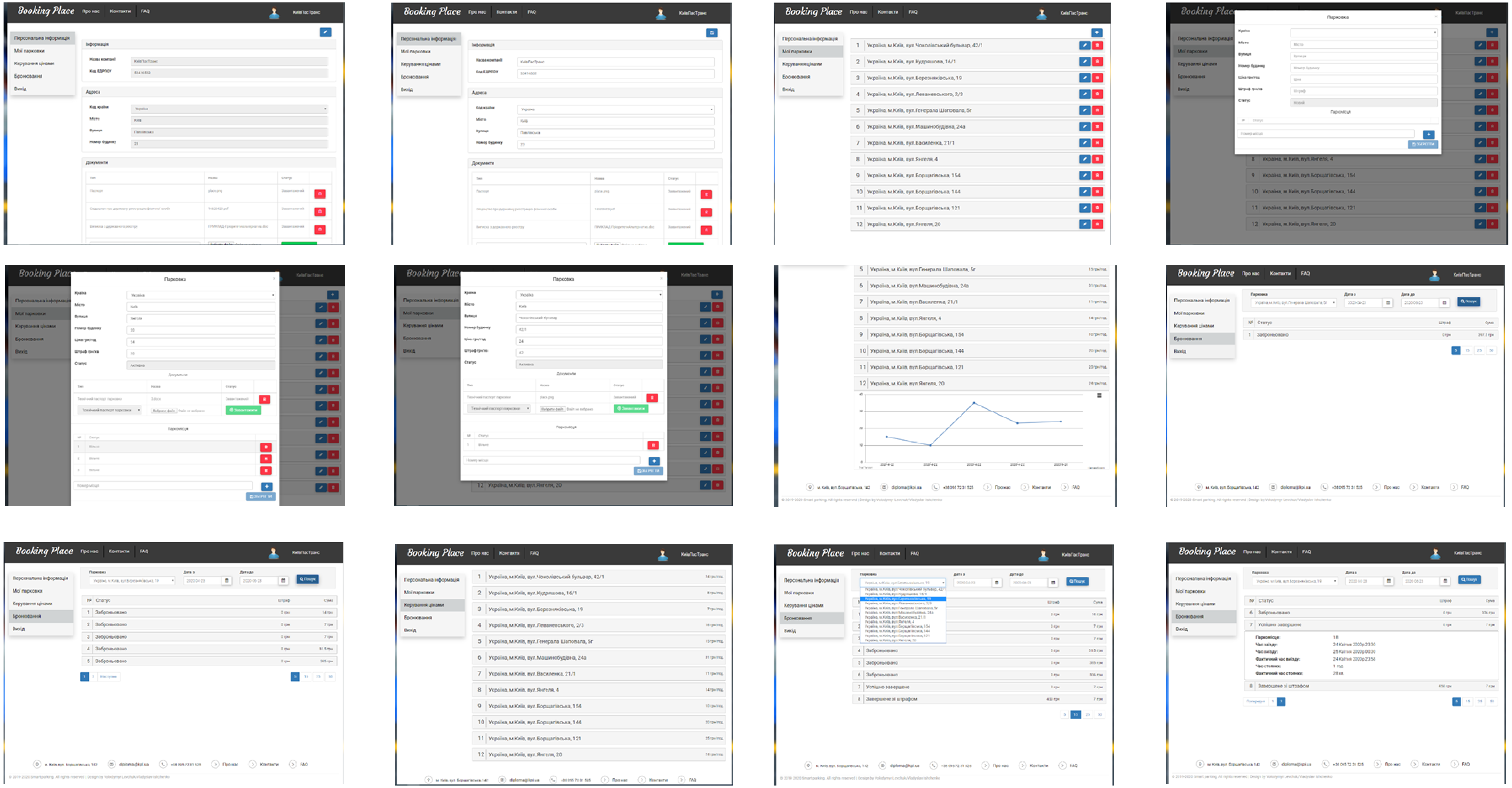
Київ – 2020 року



Зм.	Арк.	№ докум.	Підп.	Дата	ДП 6313.12.000 ССП			
Розробив		Левчук В.І.			Схема структурна послідовності	Лит.	Маса	Масштаб
Перевірив		Сперкач М.О.				Аркуш 1		Аркушів 1
Т. кон.					Інформаційна система підтримки процесу автомобільних паркувань	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-63		
Н. кон.		Проскура С.Л.						
Затвердив		Сперкач М.О.						

					ДП 6313.13.000 ССС		
						Літера	Маса / Масштаб
Зм.	Арк.	№ документа	Підпис	Дата	*Схема структурна станів системи*		
Розробив		Левчук В.І.				Аркуш 1	Аркушів 1
Перевірив		Сперкач М.О.					
Т. кон.					*Інформаційна система підтримки процесу автомобільних паркувань*	*КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-63*	
Н. кон.		Проскура С.Л.					
Затвердив		Сперкач М.О.					

ДП 6313.14.000 КЕ



Зм.	Арк.	№ документа	Підпис	Дата
Розробив		Левчук В.І.		
Перевірив		Сперкач М.О.		
Т. кон.				
Н. кон.		Проскура С.Л.		
Затвердив		Сперкач М.О.		

ДП 6313.14.000 КЕ

Креслення вигляду екранних форм

Літера	Маса	Масштаб
Аркуш 1		Аркушів 1

Інформаційна система підтримки процесу автомобільних паркувань

КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-63