Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет _____Комп'ютерних наук_____
(повна назва)

Кафедра _____Програмної інженерії_____
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

_____другий (магістерський)_____
(рівень вищої освіти)

Дослідження методів кластеризації даних про використання розумних паркінгів з метою прогнозування їх завантаження
(тема)

Виконав: студент 2 курсу, групи ІПЗм-17-1___
спеціальності 121- Інженерія програмного забезпечення
(код і повна назва спеціальності)
Освітньо-наукової програми
Інженерія програмного забезпечення_____
(повна назва освітньої програми)

_____Хомишин Д.О._____
(прізвище, ініціали)
Керівник _____доц. Лещинський В.О._____
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри, проф. _____ З.В.Дудар

2019 р.

Харківський національний університет радіоелектроніки

Факультет _Комп'ютерних наук_____

Кафедра_Програмної інженерії_____

Рівень вищої освіти_____другий (магістерський)_____

Спеціальність _121-Інженерія програмного забезпечення_____
(код і повна назва)

освітньо-наукова програма _Інженерія програмного забезпечення_____
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 20 ___ р.

ЗАВДАННЯ

НА АТЕСТАЦІЙНУ РОБОТУ

студентові _Хомишину Дмитру Олександровичу_____
(прізвище, ім'я, по батькові)

1. Тема роботи _Дослідження методів кластеризації даних про використання розумних паркінгів з метою прогнозування їх завантаження _____
затверджена наказом по університету від "___" _____20 ___ р № _____
2. Термін подання студентом роботи до екзаменаційної комісії
05 червня 2019 р.
3. Вихідні дані до роботи *розробити програмну систему для автоматизації велосипедних парковок та кластеризації даних за допомогою Visual Studio 2019, Visual Studio Code, мова програмування C# та JavaScript*

4. Перелік питань, що потрібно опрацювати в роботі *вступ, аналіз проблемної галузі, аналіз ринку, аналіз аналогів, постановка задачі, перелік вимог до програмної системи, методи кластеризації даних, опис прийнятих проектних рішень, опис розробленої програмної системи, тестування, висновки*

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) *вікова структура населення України, велоспаровка «Велиб» у Парижі, Use Case, діаграма послідовностей, діаграма розгортання, схема паттерну Model-View-ViewModel, схема роботи паттерну Dependency Injection, схема архітектури, головний керан програми, головний екран веб-сайту, логін вікно, вікно менеджменту, вікно введення додаткової інформації, віно реєстрації, вікно пошуку велопаркінгу, вікно роботи з комірками для велосипедів, вікно статусу велосипеду*

6 Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Снецчастина	доц. Лещинський В.О.		

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1.	Аналіз предметної галузі	19 квітня 2019р.	
2.	Огляд існуючих методів	27 квітня 2019р.	
3.	Методи кластеризації	10 травня 2019р.	
4.	Підготовка пояснювальної записки	25 травня 2019р.	
5.	Спецчастина	26 травня 2019р.	
6.	Підготовка презентації та доповіді	28 травня 2019р.	
7.	Попередній захист	30 травня 2019р.	
8.	Нормоконтроль, рецензування	02 червня 2019р.	
9.	Занесення диплома в електронний архів	03 червня 2019р.	
10.	Допуск до захисту у зав. кафедри	04 червня 2019р.	

Дата видачі завдання _____ _____2019 р.

Студент _____
(підпис)

Керівник роботи _____ __доц. Лещинський В.О._____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до атестаційної роботи: __ с., _ рис., __табл., _20_ джерел.

ПАРКІНГ, АВТОМАТИЗАЦІЯ, CONTINUOUS INTEGRATION, ПОСТАНОВКА ЗАДАЧІ, JAVASCRIPT, .NET, MVC, RESTAPI.

Об’єкт дослідження – методи кластеризації та візуалізації з навчанням без вчителя з метою виявлення залежностей і подальшого прогнозування на прикладі розумних паркінгів.

Метою роботи є реалізація системи для дослідження використання паркінгів, що надасть можливість оптимізації роботи.

Методи розробки базуються на технологіях .Net Framework, мови програмування C# та JavaScript, MVC, середовище Visual Studio 2019 та Visual Studio Code.

У результаті роботи здійснено програмну реалізацію програмного продукту, що автоматизує роботу паркінгу.

PARKING, AUTOMATION, CONTINUOUS INTEGRATION, TASKING, JAVASCRIPT, .NET, MVC, RESTAPI.

The object of the study is the methods of clustering and visualization with non-teacher training with the aim of detecting dependencies and further prediction on the example of intelligent parking.

The purpose of the work is to implement a system for studying the use of parking, which will enable the optimization of work.

Development methods are based on .Net Framework, C # programming language and JavaScript, MVC, Visual Studio 2019, and Visual Studio .NET.

As a result of work implemented software implementation of software product that automates the work of the parking.

ЗМІСТ

ВСТУП

Ми живемо у новому, двадцять першому, сторіччі. З кожним днем створюється все більше інновацій таких як: розумний будинок, розумні зупинки. Інформаційні технології проникають все глибше та глибше у наше повсякденне життя. Інформаційне суспільство відрізняється від суспільства, у якому домінують традиційна промисловість і сфера послуг тим, що інформація, знання, інформаційні послуги, і всі галузі, пов’язані з їхнім виробництвом (телекомунікаційна, комп’ютерна, телевізійна та ін.) ростуть більш швидкими темпами, а також є джерелом нових робочих місць і стають домінуючими в економічному розвитку [1].

Світ змінюється, тож ми змінюємося зі світом також. З моменту розпаду Радянського Союзу народилося нове покоління людей, так зване "покоління Z". Те, що минулі покоління називали "технологіями майбутнього", покоління Z вважає невід'ємною частиною повсякденного життя. Саме це, передусім, відрізняє їх від покоління Y, так як дитинство других минуло ще до "технологічного буму". Молоде покоління активно користується Інтернетом та смартфонами, але у той же час полюбляє активний відпочинок. Наприклад, під час туризму, молодь віддасть перевагу "Google maps" натомість паперовій карті, "Tripadviser" натомість путівника тощо. Цей приклад гарно ілюструє, що технології та життя молоді тісно перехрещуються кожен день. Також зараз активно розвивається велосипедний спорт. На минулорічному харківському велодні зібралася велика кількість велосипедистів та був встановлений рекорд під час виконання рухомої фігури у виді велосипеду, що складався більш ніж з 3000 чоловік. Міські управління виділяють гроші на побудову велодоріжок, біля популярних закладів міста з’явилися пракінги для велосипедів. Не став виключенням і Харківський національний університет радіелектроніки, що має власний велопаркінг для студентів на території навчального закладу.

Широкомасштабне розгортання Інтернет-речей (IoT) масово застосовують в розумних містах, і поряд з їх прийняттям існує одночасна потреба у вдосконалених функціях для обробки величезної кількості даних, створених сенсорними пристроями, і, що важливіше, зробити ці дані корисними для державних адміністрацій та громадян. Технологія IoT дозволяє контролювати широкий спектр фізичних об'єктів за допомогою недорогих і, потенційно, малопотужних сенсорних і трансмісійних технологій.

Таким чином, є можливість отримати і дослідити системи інтелектуального паркування, де дані про зайнятість паркінгу збираються з зовнішніх датчиків і надсилаються на сервери для подальшої обробки та використання у додатку. Моя мета полягає в тому, щоб зробити ці дані корисними для кінцевих користувачів, таких як паркувальники, і, власне, для велосипедистів. З цією метою я сконструюю і перевірю автоматизований алгоритм класифікації, який має дві цілі: виявлення викидів (виявлення датчиків з аномальними поведінковими моделями) і кластеризація: групувати сенсори, що демонструють подібні структури в окремі кластери.

1 АНАЛІЗ ПРОБЛЕМНОЇ ГАЛУЗІ

1.1 Аналіз ринку

Для початку необхідно визначити портрет типового користувача розробляемого проекту. Оскільки така людина має бути активним користувачем смартфону - це особа у віці від 16 до 40 років. Як видно з рисунку 1.1, така вікова група населення досі є найбільшою в Україні. Кожен з них може бути потенційним користувачем мого програмного продукту у разі популяризації велосипедної культури.

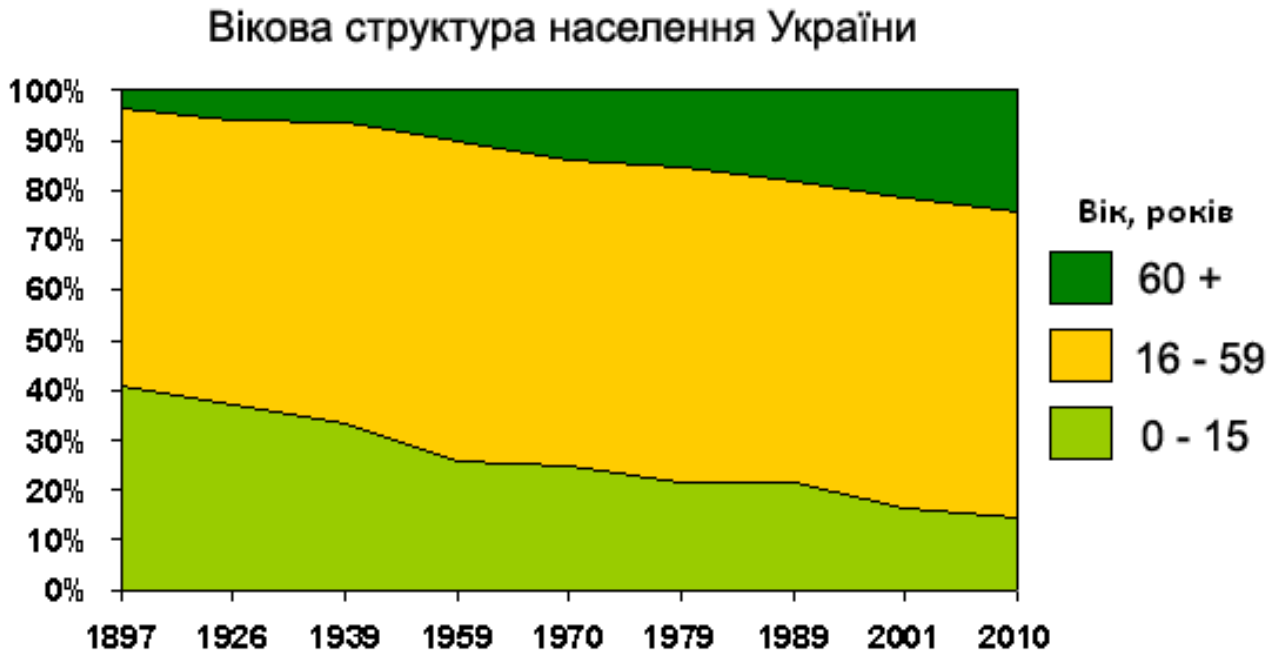

Рисунок 1.1 – Вікова структура населення України

Але, оскільки, кількість велосипедистів у цій групі не сто відсоткова, маємо дещо нижчий показник можливих користувачів. Слід зазначити, що влада поступово спонукає користуватися українців сама велотранспортом. Наприклад, відбувається поступове створення велосипедних доріжок. В разі, якщо кожен велосипедний паркінг в Україні матиме необхідне обладнання, матимемо достатньо великий показник користувачів.

Наступним кроком аналізу є аналіз світових показників велосипедних користувачів. Кількість власників велосипедів в Індії та Китаї скорочується, тоді як в Європі попит на велосипеди, навпаки, збільшується на тлі проведених кампаній з популяризації цього виду транспорту. В даний час у світі існує близько 1600000000 велосипедів, при цьому 500 млн з них припадають на Китай, 250 млн - на Європу, і 150 млн - на США [3].

Оскільки Україна рухається у політичному напрямку до Європейського Союзу, слід надати більше інформації про європейський ринок. У Нідерландах 27% всіх поїздок і 25% поїздок на роботу відбувається на велосипеді. Середня відстань подолана на велосипеді на людину в день складає 2,5 км. Голландія і велосипед йдуть пліч о пліч, як хліб і масло. Амстердам є одним з найбільш екологічно чистих міст у світі. Він має 400 км велосипедних доріжок, і майже 40% поїздок відбувається в Амстердамі на велосипеді. У Данії 18% поїздок відбувається на велосипеді. Середня відстань подолана на велосипеді на людину в день складає 1,6 км. Велоспорт, як правило, сприймають як більш здоровий, дешевший, екологічно дружній і часто навіть більш швидкий спосіб подорожі містами, ніж машиною або громадському транспорті. У Копенгаген 37% всіх громадян, пересуваються на велосипеді на щодня [4].

Таким чином, можна зробити висновок, що ринок достатньо розвинений та є потреба у розробці мого програмного продукту.

1.2 Аналіз аналогів

Прямими аналогами мого проекту є розумні велосипедні паркінги, що мають апаратні системи захисту та програмне забезпечення для користувачів. На даний момент таких паркінгів для велосипедів не існує або принаймні не має інформації в Інтернеті. Таким чином можна зробити висновок, що прямих аналогів не існує. Натомість існує багато непрямих аналогів. Найяскравіші з них

слід навести нижче для того, щоб підкреслити необхідність у програмному продукті.

Насамперед слід розповісти про можливість оренди велосипеду у багатьох містах світу, що використовують розумні паркінги. Наприклад, розповімо про таку систему у Парижі, що має назву "Велиб" [5]. Вона має широке покриття велостанцій (рис. 1.2) по всьому місту, на кожній з яких є можливість орендувати велосипед безкоштовно на пів години, якщо за цей період велосипед не буде повернений, надалі з вашої банківської картки буде зніматися платня рівна вартості проїзду у метрополітені. Дуже зручно, що є можливість взяти велосипед на одній станції, а залишити на іншій, таким чином можна пересуватися містом безкоштовно беручи новий велосипед кожні пів години. Ця система також має програмне забезпечення для смартфонів під управлінням iOS, Android та Windows phone. За допомогою застосування, користувачі мають можливість відслідковувати місцезнаходження паркінгів, кількість наявних велосипедів, кількість вільних місць, отримувати поточну інформацію про свій рахунок тощо.

Цей велопаркінг гарний приклад роботи влади на благо мешканців та гостей міста. Але у вас не вийде залишити на такому велопркінгу свій власний велотранспорт, тож таким користувачам доведеться шукати альтернативи. У такому випадку велосипедисти можуть отримати в кращому випадку закриту охороняєму станцію велопаркінгу з відео спостереженням, що необхідно буде оплачувати щомісяця, та у найпоганішому випадку - це буде загородження біля місця призначення, що можна використовувати для того, щоб закрити велосипед на замок.

Нажаль, поки що в Україні мають переваги велопаркінги другого типу. Але, навіть паркінги першого типу зручні лише для тих, хто живе поруч з ними або працює, тоді з’являється сенс залишати велосипед там та сплачувати за користування. Тож необхідно знайти щось середнє між першим варіантом паркінгів та другим. Це має бути відкритий паркінг, якою зможе користуватися кожен, сплачуючи гроші лише за час збереження велосипеда або взагалі безкоштовно. На даний момент таких варіантів дуже багато, але вони не мають

жодної інновації у собі. Це може буті серія труб або спіраль, у яку зручно завести велосипед та застібнути замком.

Рисунок 1.2 – Велопаркінг "Веліб" у Парижі

Таким чином маємо можливість зробити висновок, що на даний момент ринок не заповнив увесь попит. Існують розумні паркінги з можливістю використовувати тільки надані велосипеди, але, якщо людина бажає використовувати власний транспортний засіб, вона не зможе використовувати усі переваги розумного паркінгу.

1.3 Постановка задачі

Після детального аналізу предметної області та існуючих програмних продуктів ринку аналогів було зроблено багато важливих висновків щодо створення даного програмного продукту.

По-перше, на ринку не існує прямого аналогу. Здебільшого аналоги мають велику різницю з розроблюваним програмним продуктом. Тож моя програмна система матиме здебільшого унікальний функціонал. По-друге, цей інструмент спростить життя велосипедистам, дозволить вирішити ще одну повсякденну проблему та, можливо, стимулює використання велосипедів для щоденних поїдок в Україні.

Перевагою цього сервісу має бути стимуляція зменшення крадіжок велосипедів та більш зручне керування паркінгами.

Тож маємо дві мети у даній атестаційній роботі:

– створити додаток для керування мережею розумних паркінгів по місту. Враховуючи усі недоліки та переваги існуючих програмних продуктів, розглянутих мною, та проведений аналіз ринку, з’ясовується, що основною проблемою є те, що аналогічні сервіси пропонують розумні паркінги лише для власних велосипедів. Тому дані програмні продукти є дуже локальними рішеннями. Як виявив аналіз ринку, такий сервіс може бути цікавий як активним людям, що використовують щоденно велосипед як основний транспорт для пересування містом, так і людям, що користуються велотранспортом де інколи, але мали сумніви щодо необхідності залишати велосипед без нагляду на довгий час через можливість крадіжки.

– проаналізувати статистику даних паркування, отримуючи відповідні імітаційні моделі для паркування. Потім розглянути простий алгоритм класифікації на основі емпіричної комплементарної функції розподілу часу зайнятості і показати його обмеження. Розробити більш складний алгоритм, що використовує методи бездискретного навчання (самоорганізовуючі карти). Вони використовують генератор трасування і порівнюються з іншими схемами кластеризації, а саме максимізацією очікування, кластеризацією k-means і DBSCAN.

Оскільки сплачувати за паркінг, слідкувати за статусом велосипеда та кількістю вільних місць необхідно перебуваючи на вулиці, важливо створити додатки для мобільних пристроїв.

На етапі розробки необхідно реалізувати:

– сервер для обробки та зберігання даних від користувачів;

– для швидкого відгуку серверу слід використовувати просте nosql сховище даних;

– мобільні додатки, що будуть працювати на найпопулярніших мобільних операційних системах;

– дві ролі: адміністратор та користувач, перший має права для створення та редагування паркінгів;

– максимально спростити інтерфейс користувача, не перенасичувати елементами управління;

– оскільки цей проект надасть можливість лише керувати паркінгами, нема необхідності створювати функціонал зв’язку з паркінгами, але необхідно створити інтерфейс для майбутнього підключення.

2 АНАЛІЗ МЕТОДІВ КЛАСТЕРІЗАЦІЇ

2.1 Огляд літератури

Використання автоматизованих приладів для моніторингу на вуличних паркінгах стало популярним у кількох містах світу. У існуючих середовищах невеликі пристрої для зчитування зазвичай розміщуються в кожному місці для паркування. Це необхідно для моніторингу великих міських територій. Гарними прикладами серед багатьох інших є Лос-Анджелес, Сан-Франциско і Барселона. Перший шар цих складних систем складається з датчиків паркування на вулиці: дрібними бездротовими пристроями, що використовуються для моніторингу наявності транспортних засобів. Кожен датчик періодично прокидається, щоб перевірити стан зайнятості призначеної стоянки. Отримані дані надсилаються до серверу для подальшої обробки, управління віддаленим паркінгом та візуалізацією.

Основна мета цих систем полягає в підвищенні ефективності роботи громадської паркінгу, що досягається за рахунок збору дрібнозернистої, постійної і точної інформації про зайнятість стоянки. Зібрані дані аналізуються та надаються відділу управління паркінгами міста через відповідні інформаційні панелі. Крім того, наявність інформації про паркування в режимі реального часу також дає змогу надавати нові послуги, забезпечуючи покращений досвід роботи з міськими користувачами. Як приклад, системи паркування та інформації (PGI) допомагають водіям більш ефективно знаходити місця для паркування, тим самим вирішуючи проблему довгого пошуку вільного місця. Резервування паркомісця на вулиці є іншим прикладом застосування.

Підкреслю, що, незважаючи на стрімке зростання кількості реальних установок, до теперішнього часу з'явилося лише декілька наукових праць щодо інтелектуального аналізу даних та обробки інформації для розумного паркінгу. Використання аналітики великих даних обговорюється в доповіді «Cloud based big data analytics for smart future cities» [6], а соціальні дослідження у містах у

«Urban and social sensing for sustainable mobility in smart cities» [7], де також досліджуються відкриті проблеми, пов'язані з необхідною технологією збору, зберігання, аналізу та візуалізації великих обсягів даних про паркування. Архітектурна основа для збору та аналізу даних обговорюється в «Architecture for parking management in smart cities» [8].

Автори «Integrating pervasive computing, InfoStations and swarm intelligence to design intelligent context-aware parking-space location mechanism» [9] висунули систему паркування транспортних засобів на основі роя, що використовує розуміння контексту та бездротовий зв'язок. У цій системі паркінги обладнані інструментами на базі Інтернету та бездротовою сенсорною інфраструктурою. Інформація про паркінг збирається та візуалізується через відповідні інформаційні панелі, що передаються в Інтернеті, і повідомляється транспортним засобам, які шукають місця для паркування. Маршрут до найближчих доступних автостоянок обчислюється за допомогою алгоритмів оптимізації рою частинок і надсилається водіям. Нещодавній документ «A real-time parking prediction system for smart cities» [10] використовує дані від датчиків паркування на вулиці з міста Сантандер в Іспанії, щоб розробити та підтвердити рамки для прогнозування доступності паркінгу (за площею або за оцінкою майбутнього стану конкретних місць для паркування).

2.2 Огляд існуючих алгоритмів

Розглядаються наступні алгоритми:

– k-means;

– просторова кластеризація додатків на основі щільності (DBSCAN), який є де-факто стандартним алгоритмом кластеризації без нагляду;

– схема кластеризації, заснована на максимізації очікувань (EM), що взято з недавнього документа про кластеризацію для даних про інтелектуальну парковку і адаптований до наших конкретних параметрів.

В рамках даної роботи розробимо оригінальну техніку кластеризації, використовуючи самоорганізовучі карти (SOM), які є самокерованими нейронними мережами, що здатні вивчати прототипи в багатовимірних векторних просторах. Це тісно пов'язано з пошуком областей, по одному на прототип, і вирішенням багатовимірної проблеми класифікації. Наш підхід на основі кластеризації SOM представлений у двох варіантах: схема, яка вимагає заздалегідь відомого числа класів даних (кластерів) і алгоритму, який автоматично знаходить кількість класів з аналізу векторів ознак.

Всі алгоритми кластеризації регулюються під час виконання, використовуючи синтетичні дані, які нагадують реальні. Таким чином, ми отримуємо набір даних, який використовується для перевірки правильності класифікаторів. В цілому, після тестування, виявилося, що k-means, DBSCAN та EM-кластеризація мають проблеми з класифікацією і не можуть відокремити відхилення від інших кластерів, тоді як SOM-схема працює задовільно, досягаючи найвищої класифікації при тестуванні на синтезованих подіях, і надійно виявляє всі викиди, які застосовуються до реальних даних.

2.3 Високорівневі техніки класифікації

У цьому розділі опишемо деякі вдосконалені методи класифікації, які призведуть до покращення продуктивності. З цією метою визначаємо поняття викидів і вводимо відповідний набір ознак, які використовуються наступними алгоритмами для ефективної класифікації сенсорів.

Викиди зазвичай визначаються як точки даних, які глобально мають найменший ступінь подібності до набору даних, до якого вони належать, і для

нашої класифікаційної задачі точка даних відповідає тимчасовим рядам, що генеруються певним датчиком, який представляє поведінку пов'язаного з ним простору для паркування. Крім того, зауважимо, що оцінка того, чи є певний датчик викидом, сильно залежить від визначення, яке використовується для подібності. Якщо, наприклад, ми б порівнювали послідовності паркінгів тільки на основі середньої тривалості їх паркування, то дві послідовності з однаковою тривалістю середньої події і різною частотою подій будуть розглядатися як подібні. Звичайно, це прийнятно до тих пір, поки нашій програмі не потрібно відстежувати частоту подій. Наприклад, тривалість події має значення лише для того, щоб оцінити, чи закінчився час паркування, і в цьому випадку власник автомобіля повинен накласти штраф.

Отже, визначення особливостей, що представляють інтерес, має вирішальне значення для правильного встановлення викидів, і нам також необхідно визначити метрику подібності над цими ознаками.

У машинному навчанні та розпізнаванні образів, ознака може бути визначена як індивідуальна вимірювана властивість явища, що спостерігається. Інформаційні, дискримінаційні та незалежні ознаки є ключовими для розробки ефективних алгоритмів класифікації. Для їх визначення ми розглянемо для кожного датчика паркування *i* наступні статистичні заходи:

– Зайнятість датчика (SO): враховує кількість часу, протягом якого зайнято $s_i(t) = 1$, тобто відповідне місце для паркування;

– частота подій (EF): визначає кількість паркінгів за одиницю часу;

– тривалість паркування (PD): вимірює тривалість паркування;

– тривалість вільного часу (VD): вимірює тривалість часу без користувачів.

Розрахуємо середню погодинну тенденцію для кожної з вищезазначених заходів, враховуючи два класи: (cl1) будні та (cl2) вихідні дні, і усереднення точок даних, що відповідають тій самій годині для всіх днів одного класу. Для кожної статистичної міри це призводить до 24 середніх значень, де значення, пов'язане з годиною $t \in \{1, \ldots, 24\}$ для класів cl1 / cl2, отримується усередненням за годину t протягом всіх днів одного класу. Таким чином, функції годинного датчика були

отримані для кожної години дня t ∈ {1, ⋯, 24} для класів cl1 та cl2, і їх ми відповідно позначаємо через $m_{SO}^{1}(t)$ та $m_{SO}^{2}(t)$. Аналогічні функції були отримані для інших заходів, тобто, $m_{EF}^{*}(t)$, $m_{PD}^{*}(t)$, $m_{VD}^{*}(t)$, де * або один (cl1) або два (cl2). Зауважимо, що всі функції $m_a^b(t)$ нормалізувалися через $m_a^b(t) \leftarrow (m_a^b(t) - m_{min})$ / $(m_{max} - m_{min})$, де (m_{max} та m_{min} відповідно представляють максимальний та мінімальний елементи у наборі для вимірювання a ∈ {SO, EF, PD, VD} і клас b ∈ {1, 2}. Також були оцінені інші нормалізації, але ця призвела до найкращих результатів.

Це призводить до усього 2 × 24 = 48 середніх значень для кожного показника (два класи і 24 год на клас), що становить 4 × 48 = 192 значень для представлення чотирьох статистичних заходів, які ми використовуємо для характеристики поведінки датчика при парковці. Зауважимо, що ми навмисно вирішили окремо обробляти дані з робочих та вихідних днів. Це пояснюється тим, що дані парковки з цих двох класів демонструють дуже різну поведінку, і врахування цього факту підвищує точність алгоритму (хоча й за рахунок більшої кількості елементів).

Функція ознаки *f*(*t*) обчислюється як показано на формулі 1.

$$f(t) = \begin{cases} w_1 m_{\mathrm{SO}}^{1}(t) + w_2 m_{\mathrm{PD}}^{1}(t) & t \in \{1, \cdots, 24\} \\ w_3 m_{\mathrm{EF}}^{1}(t') + w_4 m_{\mathrm{VD}}^{1}(t') & t \in \{25, \cdots, 48\} \\ w_1 m_{\mathrm{SO}}^{2}(t') + w_2 m_{\mathrm{PD}}^{2}(t') & t \in \{49, \cdots, 72\} \\ w_3 m_{\mathrm{EF}}^{2}(t') + w_4 m_{\mathrm{VD}}^{2}(t') & t \in \{73, \cdots, 96\} \end{cases} \quad (1)$$

де $t' = ((t-1) \bmod 24) + 1$.

Таким чином, вісім функцій (чотири на клас) обчислюються для кожного датчика в середовищі паркування, і їх зважена сума обчислюється з використанням відповідних ваг w_1, w_2, w_3, w_4 з $\sum_{k=1}^{4} w_k = 1$ і $w_k \in [0, 1]$. Таким чином, ми отримуємо функцію єдиної ознаки *f (t)* (див. Рівняння (2)), що складається з 96 середніх значень: перші 48 значень є репрезентативними

середніми часовими показниками класу cl1, тоді як другі 48 значень представляють погодинну заходи з класу cl2. Ваги визначають відносну важливість кожної статистичної міри; їх правильний розподіл має вирішальне значення для отримання хорошої класифікації.

Приклад роботи функції нашого набору даних для паркування показаний на рисунку 2.1.

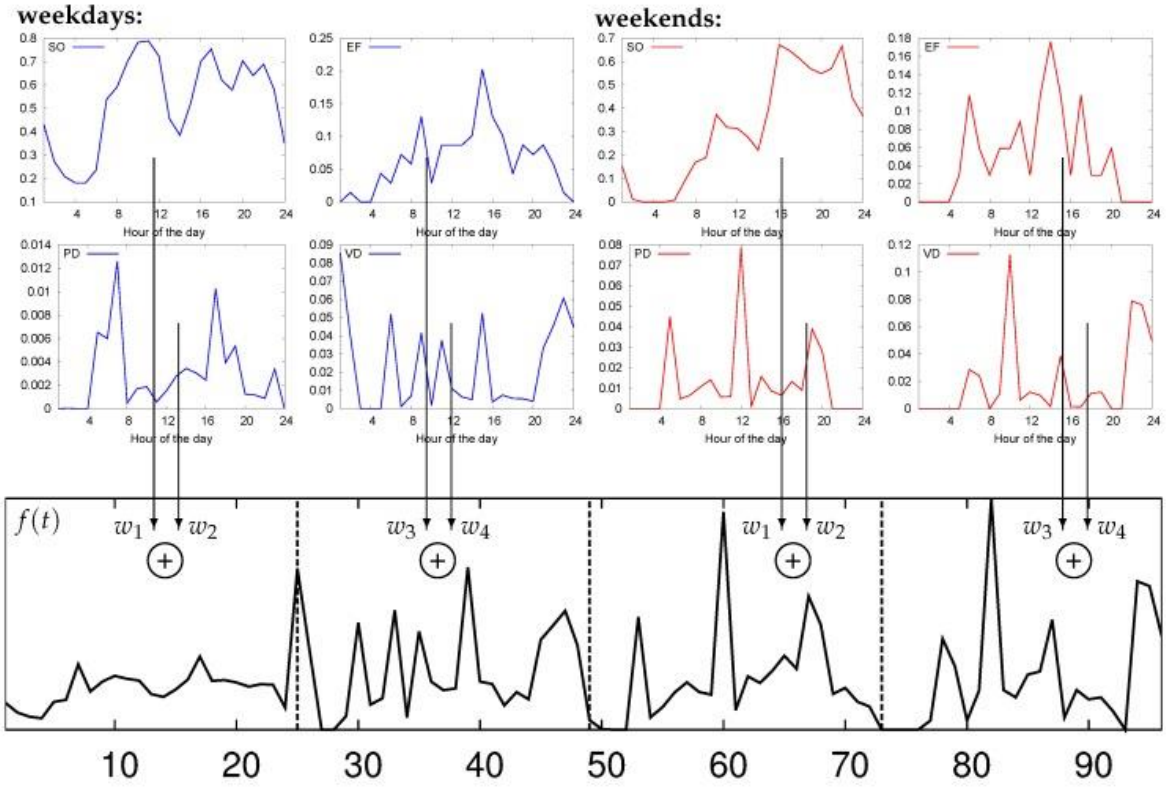

Рисунок 2.1 – Приклад отриманих даних з паркінгу

Зауважимо, що $f(t)$ є специфічною для датчика, тобто одна така функція обчислюється для кожного датчика паркування. Крім того, для кожного датчика паркування i, ця функція визначає вектор характеристик $\boldsymbol{x}_i = [x_{i1}, x_{i2}, \cdots, x_{iK}]^T \in \mathbb{R}^K$, де K = 96 елементів, які використовуються для навчання алгоритмів кластеризації наступних розділів. Зокрема, для датчика i задаємо $x_{it} \leftarrow f(t)$, $t = 1, \cdots, K$, де $m_a^b(t)$ в $f(t)$ обчислюється з мірків цього датчика.

2.4 Класифікація на основі самоорганізуючих карт

Самоорганізуючі карти (SOM) – це алгоритм кластеризації, який використовує можливості навчання без керівника для автоматичного виявлення кластерів і одночасного виявлення відхилень.

SOM є нейро-обчислювальним алгоритмом, який перетворює високовимірні дані в одно- або двовимірний простір через нелінійний, конкурентний і без нагляду процес навчання. SOM відрізняється від інших штучних нейронних мереж, оскільки використовує функцію сусідства для збереження топологічних властивостей вхідного простору. Вона вивчається за допомогою вхідних прикладів, а вхідний простір відображається в двовимірну решітку нейронів, зберігаючи властивість, що подібні вхідній структурі відображені сусідніми нейронами на карті.

Розглянемо одно- та двовимірні карти, які відповідно складаються з послідовностей M × 1 нейронів і решітки ℓ = M × M нейронів, з M> 1. Ці карти вибірково налаштовані на вхідні структури через безкваліфікований (також називається конкурентним) процес навчання. Коли навчання прогресує, ваги нейронів мають тенденцію ставати впорядкованими по відношенню один до одного таким чином, що над решіткою створюється значна система координат для різних функцій введення. Іншими словами, SOM створює топографічну карту вхідного простору даних, де просторові місця розташування або координати нейронів в решітці відповідають певній області або власній статистичній функції вхідних даних. Примітно, що це досягається без необхідності будь-яких попередніх знань щодо розподілу вхідних даних.

За допомогою $\mathcal{X} \subset \mathbb{R}K$ вказуємо множину ознак (вхід), а $xi \in \mathcal{X}$ - вхідний вектор ознак, пов'язаний з сенсором $i = 1, \ldots, N$, де $xi = [xi1, xi2, \cdots, xiK]T$. З N ми маємо на увазі число датчиків, і $|\mathcal{X}| = N$. Нехай $\mathscr{L}$ - решітка. Кожен нейрон з'єднаний з кожним компонентом вхідного вектора, як показано на рисунку 2.2.

Зв'язки між вхідним вектором і нейронами зважені, так що *j*-й нейрон пов'язаний з синаптичним ваговим вектором wj ∈ ℝK, де *wj* = [*wi*1,*wi*2,⋯,*wiK*]*T*.

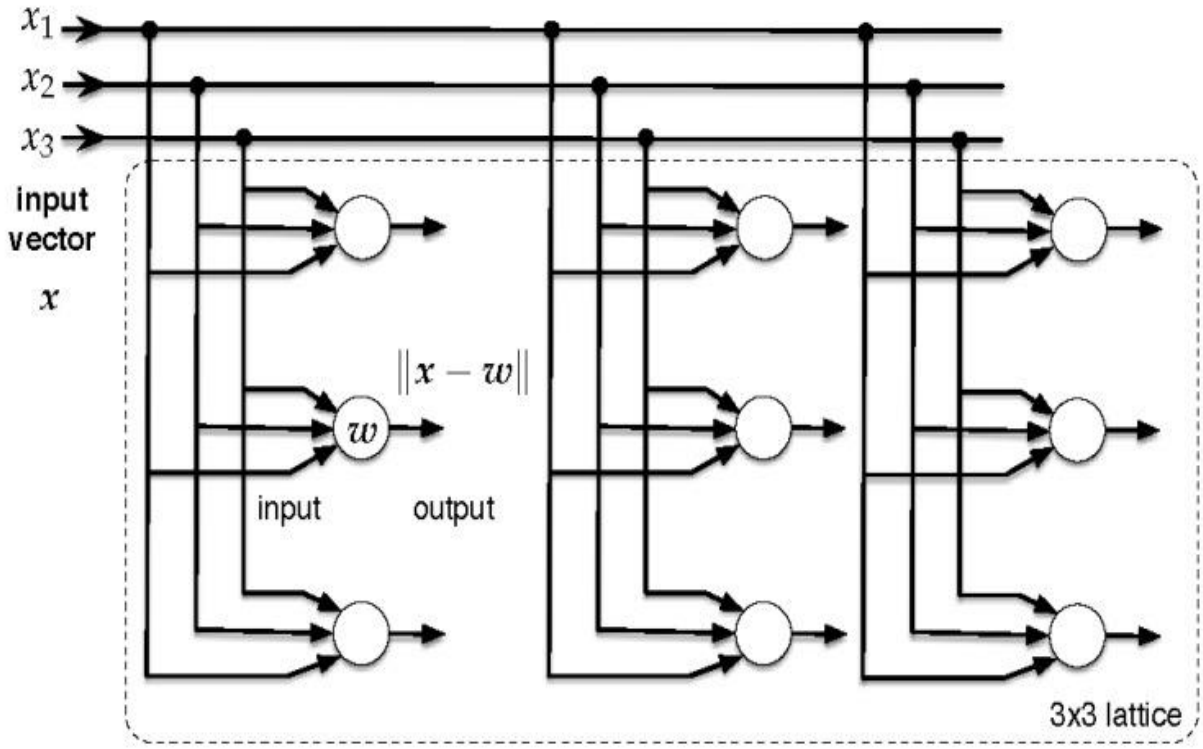

Рисунок 2.2 – Схема процесу навчання SOM

Алгоритм кластеризації на основі SOM має 5 пунктів.

1. Ініціалізація: встановимо початковий крок часу до $n = 0$ і оберемо малі випадкові значення для початкових векторів синаптичної ваги $wj(0)$, $j \in \mathcal{L}$.

2. Вибірка: встановимо $n \leftarrow n + 1$ і відберемо вхідний паттерн xn, тобто вектор ознак з *n*-го датчика паркування.

3. Узгодження подібності: знайдемо виграючий нейрон, індекс якого $j^*(n)$ на етапі часу *n*, використовуючи критерій мінімальної відстані (2)

$$j^*(\mathrm{n}) = \arg min_j \| x_n - w_j(n) \|, j \in \mathcal{L}, \quad (2)$$

де з $\| a - b \|$ маємо на увазі евклідову відстань між векторами a і b

4. Оновлення: налаштуємо вектори синаптичної ваги всіх нейронів за допомогою рівняння оновлення (3).

$$w_j(n+1) = w_j(n) + \eta(n)h_{ji}(n)(x_n - w_j(n)), \quad (3)$$

де $\eta(n)$ – параметр швидкості навчання при ітерації n

$h_{ji}(n)$ – функція околиці, центрована на $j^*(n)$ при ітерації n

5. Сусідство: відрегулюємо розмір можливого сусідства ($h_{ji}(n)$), швидкість навчання ($\eta(n)$) і повернемося до кроку 2, якщо $n < n_{\text{iter}}$; інакше закінчимо.

Етапи алгоритму зображені на рисунку 2.3.

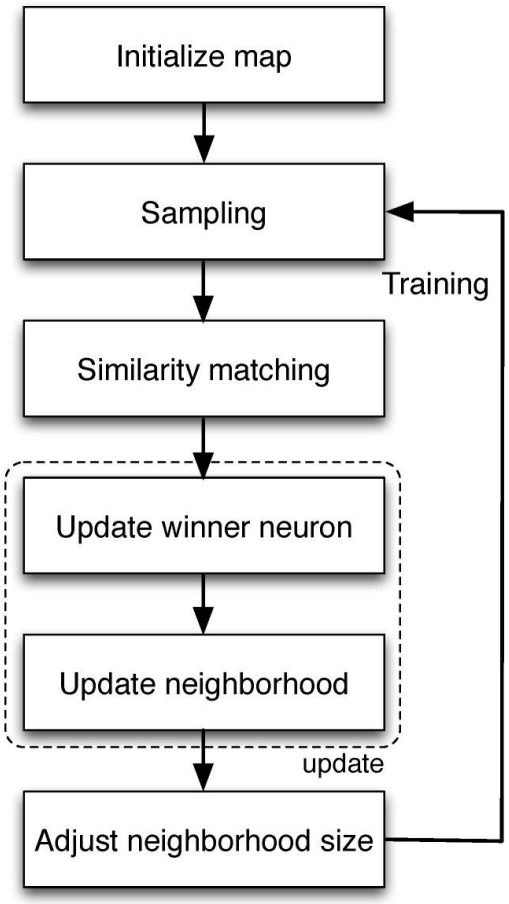

Рисунок 2.3 – Алгоритм SOM в загальном вигляді

Підсумуємо роботу алгоритму кластеризації, шляхом визначення його основних кроків - навчання і кластеризації.

Крок 1: Навчання. Процес навчання відбувається за допомогою показу вхідних даних $xi \in \mathcal{X}$ в SOM. Кожного разу, коли на карту вводиться новий зразок, нейрони конкурують між собою для активації, в результаті чого обирається тільки один виграючий нейрон. Щоб визначити виграючий нейрон, вхідний вектор x_i порівнюється з векторами синаптичної ваги w_j всіх нейронів. Домінує тільки нейрон, чий синаптично-ваговий вектор максимально відповідає поточному вхідному вектору за даною мірою відстані (яку ми обираємо рівною евклідової відстані, яка зазвичай використовується). Отже, ваги виграючого нейрона і вузлів решітки в межах його околиці регулюються так, щоб вони нагадували вхідний вектор.

Хорошим і загальним вибором для $h_{ji}(n)$ є функція Гауса, чий проміжок в момент часу $n = 0$ вибирається для того, щоб охопити всі нейрони в решітці, а потім зменшити при підготовці карти. Причина цього полягає в тому, що широке початкове топологічне сусідство, тобто грубий просторовий дозвіл у процесі навчання, спочатку індукує грубий глобальний порядок у векторних значеннях синаптичної маси. Отже, під час тренування звуження покращує просторовий дозвіл карти, не руйнуючи глобальний порядок. Швидкість навчання також зменшується коли вибирається достатньо велика кількість ітерацій, так що всі синаптичні ваги стабілізуються. Більш того, якщо кількість доступних елементів входу менше ніж у n_{iter}, нові приклади можуть бути переповнені з вхідних даних до збіжності.

Крок 2: Кластеризація. Після завершення навчання карта адаптується для ефективного і компактного відображення простору ознак. Кластеризація безпосередньо працює за допомогою тренованої карти. Вектори $\boldsymbol{x}_i \in \mathcal{X}$ з $i = 1, \ldots, N$ подаються як вхід до карти. Для кожного вектора xi використовуємо рівняння 3, щоб оцінити виграючий нейрон на карті для цього вектора. З j^* вказуємо індекс виграючого нейрона. Кластери будуються шляхом групування векторів ознак, що повертають той самий індекс. Потім негайно усвідомлюють, що максимальне число кластерів, повернених SOM, дорівнює кількості нейронів на карті, тобто один кластер на нейрон. В деяких випадках невелика кількість

нейронів ніколи не може бути активована, тобто бути обрана як найкращий придатний нейрон для певного вхідного перемикача (вектору). Як наслідок, число кластерів може бути меншим, ніж $M2$. При такому підході кількість кластерів дещо фіксується заздалегідь, оскільки вона суворо залежить від кількості нейронів на карті.

Кластеризація на основі автоматичного SOM: Наведений вище підхід кластеризації SOM вимагає заздалегідь знати кількість кластерів у наборі даних (тобто кількість нейронів у карті SOM). Надалі опрацюємо алгоритм кластеризації без нагляду, який не повинен заздалегідь знати кількість класів даних. Замість кластеризації даних через агломераційний підхід, приймаємо дивізійний підхід, тобто починаємо з великого кластера, що містить весь набір даних, і ітеративно розбиваємо цей початковий кластер на прогресивно менші. Карта SOM з двома нейронами використовується як нелінійний класифікатор для розбиття кластерів на дві підмножини. SOM має вищу дискримінантну силу, ніж лінійно-дискримінантні функції. Крім того, використовуємо процедуру локальної глобальної адаптивної кластеризації, в основі якої лежить самоподібність, знайдена в глобальних і локальних характеристиках багатьох наборів даних реального світу. Зокрема, оцінимо міру кореляції серед особливостей усього набору даних (глобальної метрики) і показників менших кластерів, отриманих на певному етапі алгоритму (локальні метрики). Таким чином, глобальні та локальні показники порівнюються, щоб визначити, коли поточні кластери повинні бути ще більш розділені. Перед описом алгоритму введемо такі поняття:

– точка даних: Вхідний набір даних складається з N точок даних, де «точка даних» і - вектор стовпців властивостей $xi \in \mathcal{X}$, пов'язаний з датчиком паркування $i = 1, \dots, N$. Ці вектори зручно представлені через повну характеристику матриці $X = [x_1, \cdots, x_N]$. Під X_p мається на увазі підматриця X, отримана шляхом збору p стовпців (векторів ознак), не обов'язково першого p. Загальний кластер $\mathcal{C}$, що містить p елементів, однозначно ідентифікується за допомогою набору p датчиків і відповідної матриці X_p;

– кластерна когезивність: розглянемо кластер $\mathcal{C}$ з p елементами, а X_p - відповідну матрицю характеристик. Використаємо функцію розсіювання як міру його когезивності, тобто для визначення відстані між елементами кластера та його середнього значення (центроїд). Центроїд $X_p = [x_1, \cdots, x_p]$ обчислюється як: μp=(∑pj=1xj)/p. Дисперсія членів кластера навколо μ_p оцінюється через стандартне відхилення зразка (5).

$$\boldsymbol{\sigma}(X_p) = \sqrt{\frac{1}{p-1}\sum_{j=1}^{p} ||x_j - \mu_p||^2} \qquad (5)$$

де ‖ x ‖ є евклідова норма вектора *x*.

Аналогічно, дисперсію повної матриці характеристик X позначаємо σ(X), і визначимо подальший поріг $\sigma_{\text{th}} = \gamma\sigma(X)$ для відповідного γ ∈ [0, 1];

– глобальні та місцеві метрики кластеризації: у тестах експериментували з різними показниками, і найкращі результати були отримані шляхом відстеження кореляції між функціями, як ми зараз деталізуємо. Продовжимо обчислювати два статистичні показники: першу метрику, яка називається глобальною, що отримаємо для всієї матриці ознак *X*; локальну метрику обчислюють для менших кластерів (матриця X_p)).

3 ПЕРЕЛІК ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

Дана програмна система відноситься так званих проектів Інтернету речей, оскільки має за мету автоматизацію велосипедних паркінгів.

Інтернет речей (англ. Internet of Things, IoT) — це мережа, що складається з взаємозв'язаних фізичних об'єктів (речей) або пристроїв, які мають вбудовані датчики, а також програмне забезпечення, що дозволяє здійснювати передачу і обмін даними між фізичним світом і комп'ютерними системами, за допомогою використання стандартних протоколів зв'язку. Крім датчиків, мережа може мати виконавчі пристрої, вбудовані в фізичні об'єкти і пов'язані між собою через дротові і бездротові мережі. Ці взаємопов'язані об'єкти (речі) мають можливість зчитування та приведення в дію, функцію програмування та ідентифікації, а також дозволяють виключити необхідність участі людини, за рахунок використання інтелектуальних інтерфейсів [10].

Основною концепцією IoT є можливість підключення всіляких об'єктів (речей), які людина може використовувати в повсякденному житті, наприклад, холодильник, кондиціонер, автомобіль, велосипед і навіть кросівки. Всі ці об'єкти (речі) повинні бути оснащені вбудованими датчиками або сенсорами, які мають можливість обробляти інформацію, що надходить з навколишнього середовища, обмінюватися нею і виконувати різні дії в залежності від отриманої інформації. Прикладом впровадження такої концепції є система «розумний будинок» або «розумна ферма». Ця система аналізує дані навколишнього середовища і в залежності від показників регулює температуру в приміщенні. У зимовий період регулюються інтенсивність опалення, а в разі спекотної погоди будинок має механізми відкривання і закривання вікон, завдяки чому провітрюється будинок, і все це відбувається без втручання людини.

Для об'єднання повсякденних речей у мережу потрібні декілька технологій:

– для ідентифікації кожного об'єкту потрібна проста, компактна технологія. Тільки при наявності системи унікальної ідентифікації можна збирати та

накопичувати інформацію про певний предмет. Такий функціонал можна забезпечити за допомогою чіпів RFID (Radio-Frequency IDentification). Вони здатні без власного джерела струму передавати інформацію приладам зчитування. Кожен чіп має індивідуальний номер. Як альтернатива для даної технології для ідентифікації об'єктів можуть використовуватись QR-коди. Для визначення точного місця знаходження речі підійде технологія GPS, яка ефективно використовується вже сьогодні у смартфонах та навігаторах;

– для відслідковування змін у стані елементу чи оточуючого середовища об'єкти повинні оснащуватися сенсорами;

– для обробки та накопичення даних з сенсорів повинен використовуватися вбудований комп'ютер (наприклад Raspberry Pi, Intel Edison);

– для обміну інформацією між пристроями можуть бути використані технології бездротових мереж (Wi-Fi, Bluetooth, ZigBee, 6LoWPAN).

Інтеграція з Internet має на увазі, що пристрої будуть використовувати IP-адреси в якості унікального ідентифікатора. Проте, через обмежені адресні простори в IPv4 (що дозволяє використовувати 4,3 мільярда унікальних адрес), необхідно використовувати IPv6, який забезпечує унікальними адресами мережевого рівня не менше 300 млн пристроїв на одного жителя Землі. Об'єктами в IoT можуть бути не тільки пристрої із сенсорними можливостями, але також пристрої, які виконують дії (наприклад, лампочки або замки, якими керують через Інтернет).

Для бездротової передачі даних особливо важливу роль в побудові інтернету речей грають такі якості, як ефективність, відмовостійкість, адаптивність, можливість самоорганізації. Основний інтерес в цій якості представляє стандарт IEEE 802.15.4, що управляє доступом для організації енергоефективних персональних мереж, і є основою для таких протоколів, як ZigBee, WiFi, Bluetoot, 6LoWPAN.

ZigBee – це комунікаційна технологія, заснована на протоколі IEEE 802.15.4 для реалізації низькошвидкісних бездротових приватних мереж. ZigBee володіє такими характеристиками, як низьке енергоспоживання, низька швидкість

передачі даних, низька вартість і висока пропускна здатність. В даний час ZigBee використовується в основному при передачі інформації серед різного електронного обладнання, які знаходяться в межах короткої відстані і швидкості передачі даних не дуже висока. Це, в основному периферійні пристрої (миша, клавіатура) і побутова електроніка (TV, DVD), а також промислові управління (монітори, датчики і засоби автоматизації).

Серед провідних технологій важливу роль у проникненні інтернету речей грають рішення PLC - технології побудови мереж передачі даних по лініях електропередач, так як у багатьох додатках присутній доступ до електромереж (наприклад, торгові автомати, банкомати, інтелектуальні лічильники, контролери освітлення спочатку підключені до мережі електропостачання). 6LoWPAN, який реалізує шар IPv6 як над IEEE 802.15.4, так і над PLC, будучи відкритим протоколом, стандартізуемих IETF, відзначається як особливо важливий для розвитку інтернету речей.

Важливою вимогою до проекту є забезпечення безпеки інформації. Інтернет речей може викликати величезні зміни у повсякденному житті, надавши звичайним користувачам абсолютно новий рівень комфорту. Але якщо елементи такої системи не будуть належним чином захищені від несанкціонованого втручання, за допомогою надійного криптографічного алгоритму, замість користі вони принесуть шкоду, надавши кіберзлочинцям лазівку для підриву інформаційної безпеки. Оскільки речі із вбудованими комп'ютерами зберігають дуже багато інформації про свого власника, зокрема можуть знати його точне місцезнаходження, доступ до такої інформації може допомогти зловмисникам вчинити злочин.

4 ОПИС ПРИЙНЯТИХ ПРОЕКТНИХ РІШЕНЬ

4.1 UML – моделювання

Моделювання в широкому сенсі – це особливий пізнавальний процес, метод теоретичного та практичного опосередкованого пізнання, коли суб'єкт замість безпосереднього об'єкта пізнання вибирає чи створює схожий із ним допоміжний об'єкт-замісник (модель), досліджує його, а здобуту інформацію переносить на реальний предмет вивчення. Моделювання - це процес створення та дослідження моделі, а модель - засіб, форма наукового пізнання.

Під моделлю розуміється об'єкт будь-якої природи (мислено уявлена або матеріально реалізована система), котрий, відображаючи чи відтворюючи в певному сенсі об'єкт дослідження, здатний заміщати його так, що вивчення моделі дає нову інформацію про об’єкт. Для найбільш повного уявлення про сервіс та його принцип дії, були побудовані наочні UML діаграми, котрі дають повний опис програмного продукту. UML – мова графічного опису для об’єктного моделювання в області розробки програмного забезпечення. UML являє собою мову широкого профілю, це відкритий стандарт, котрий використовує графічні позначення для створення абстрактної моделі системи, котру називають UML-моделлю.

UML може бути застосовано на всіх етапах життєвого циклу аналізу бізнес-систем і розробки прикладних програм. Різні види діаграм які підтримуються UML, і найбагатший набір можливостей представлення певних аспектів системи робить UML універсальним засобом опису як програмних, так і ділових систем. Діаграми дають можливість представити систему у такому вигляді, щоб її можна було легко перевести в програмний код.

Use-cases діаграма (діаграма варіантів використання) – це опис поведінки системи, як вона відповідає на зовнішні запити. Іншими словами, сценарій використання описує, «хто» і «що» може зробити з розглянутою системою.

Основні важливі функції проекту для користувача зображені на use-cases діаграмі зображеної на рисунку 4.1. З моделі видно, що існує три головні функції:

– пошук вільного паркінгу;

– сплатити за паркінг;

– отримати статус велосипеду.

Для кожної з функцій є включення та розширення, що уточнюють головні варіанті використання.

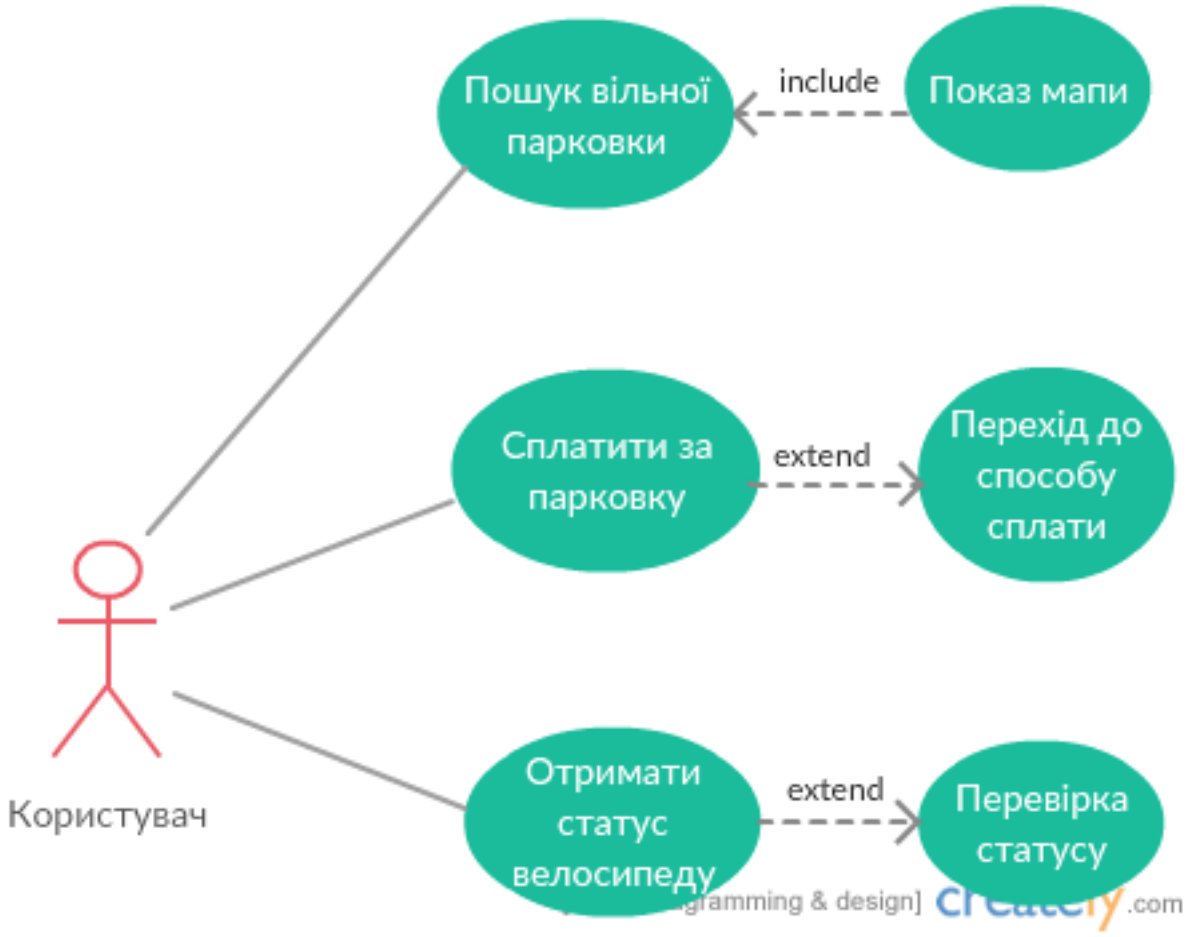

Рисунок 4.1 – Use Case

Діаграма послідовності відображає взаємодії об'єктів впорядкованих за часом. Зокрема, така діаграми відображає задіяні об'єкти та послідовність відправлених повідомлень між ними. Нижче на рисунку 3.2 приведена діаграма послідовностей, що відображає взаємодію об`єктів під час процесу створення нової публікації в системі.

На діаграмі послідовностей можна побачити що відбувається в системі поза очима користувача. Якщо користувач хоче залишити свій велосипед, він має бути зареєстрованим користувачем та відправити запит на замикання велосипеду. Після цього сервер обробить інформацію так замкне велосипед. Те саме відбувається при відмиканні.

Послідовність замикання має такий вигляд:

– користувач знаходить свій паркінг та обирає місце;

– відбувається сплата, якщо паркінг платний;

– запит на сервер для замикання;

– сервер відправляє сигнал на паркінг для замикання;

– паркінг замикається та починає відправляти статус.

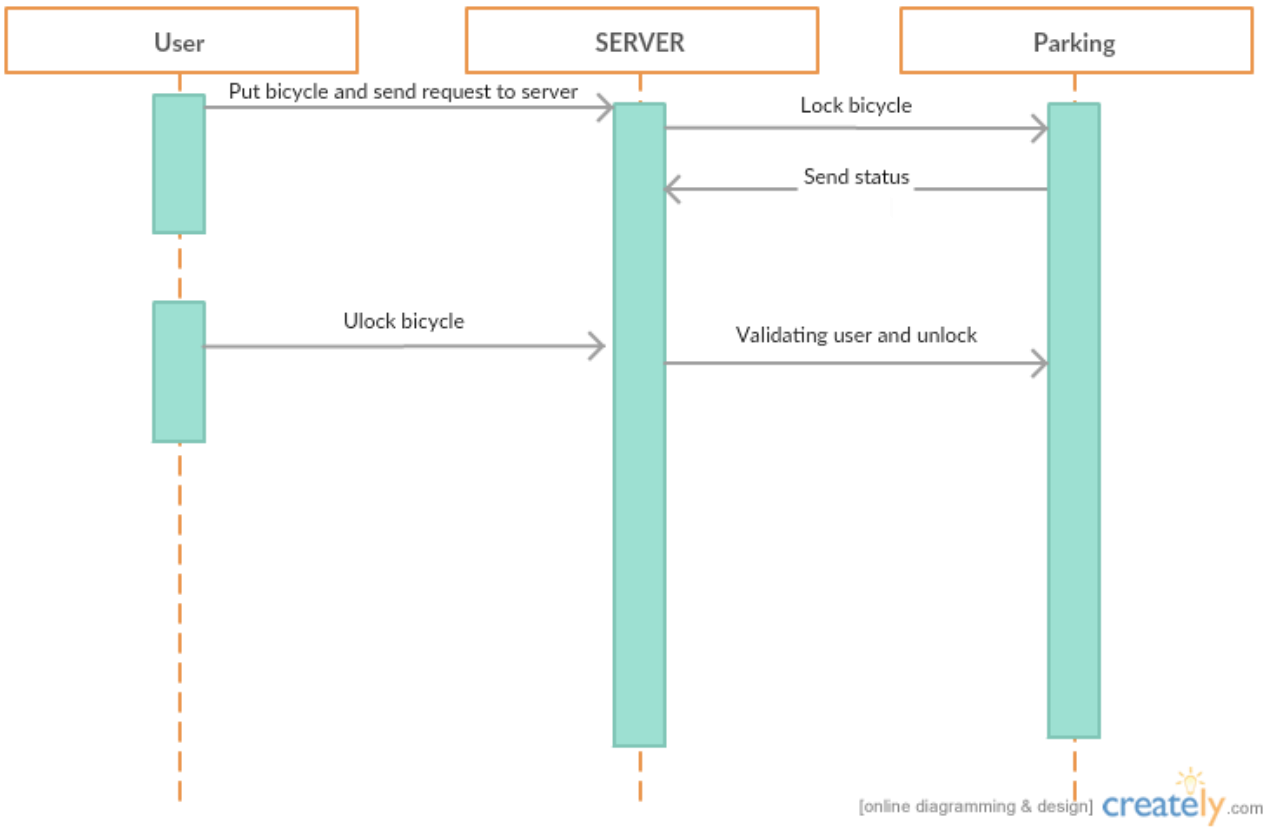

Рисунок 4.2 – Діаграма послідовностей

Послідовність відмикання:

– користувач відправляє запит на сервер для відмикання;

– сервер перевіряє користувача та відмикає вілосипед в разі успішної валідації.

Для зображення процесу розгортання програмної системи необхідно побудувати діаграму розгортання.

На рисунку 4.3 зображено діаграму розгортання.

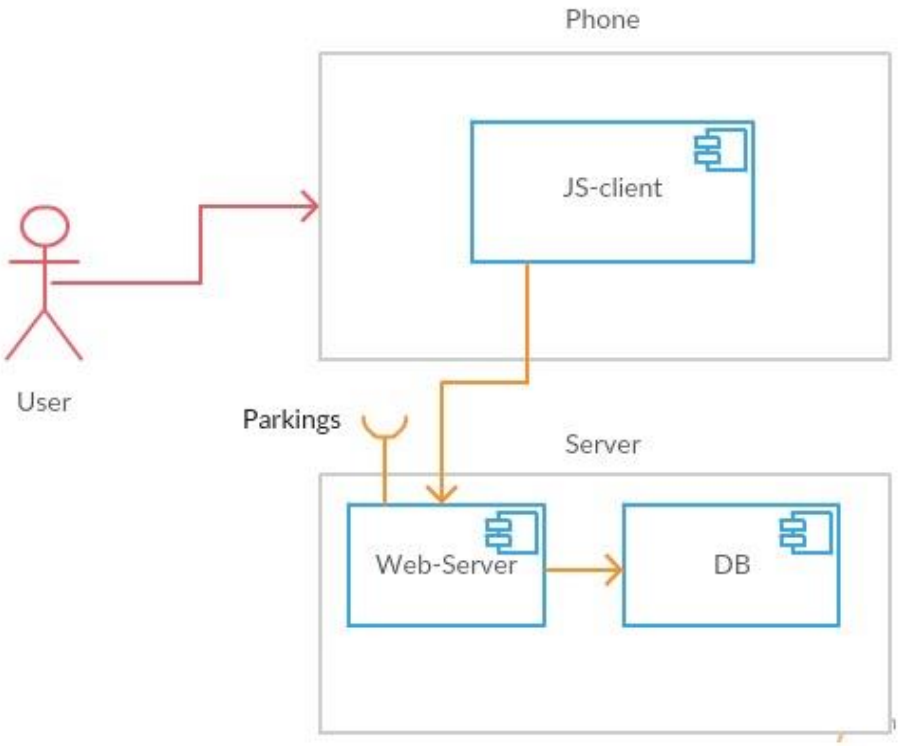

Рисунок 4.3 – Діаграма розгортання

Діаграма розгортання (англ. deployment diagram) – діаграма в UML, на якій відображаються обчислювальні вузли під час роботи програми, компоненти, та об'єкти, що виконуються на цих вузлах. Компоненти відповідають представленню робочих екземплярів одиниць коду. Компоненти, що не мають представлення під час роботи програми на таких діаграмах не відображаються; натомість, їх можна відобразити на діаграмах компонент. Діаграма розгортання відображає робочі екземпляри компонент, а діаграма компонент, натомість, відображає зв'язки між типами компонент.

Як видно з діаграми, для розгортання необхідно мати сервер де буде знаходитись уся бизнес-логіка додатку та база даних. Сервер має з’єднання з паркінгами. Маємо доступ з бізнес-логіки до БД.

4.2 Архітектура програмної системи

Для реалізації серверної частини було вирішено використовувати DDD (дизайн на основі предметної області) архітектурний стиль. Проектування на основі предметної області (Domain Driven Design, DDD) – це об'єктно-орієнтований підхід до проектування ПО, заснований на предметної області, її елементах, поведінці і відносинах між ними. Метою є створення програмних систем, реалізація яких лежить в основі предметної області, шляхом визначення моделі предметної області, вираженої мовою фахівців у цій галузі. Модель предметної області може розглядатися як каркас, на підставі якого будуть реалізовуватися рішення [11].

Для застосування DDD необхідно чітко розуміти предметну область, яку передбачається моделювати, або мати здібності для оволодіння такими знаннями. При створенні моделі предметної області група розробки нерідко працює у співпраці з фахівцями в даній області. Архітектори, розробники і фахівці в даній області володіють різною підготовкою і в багатьох ситуаціях використовуватимуть різні мови для опису своїх цілей, бажань і вимог. Проте в рамках DDD вся група домовляється використовувати тільки одну мову, орієнтований на предметну область і виключає всі технічні жаргонізми.

В якості ядра ПО виступає модель предметної області, яка є прямою проекцією загальної мови; з її допомогою шляхом аналізу мови група швидко знаходить прогалини в ПЗ. Створення спільної мови це не просто вправа з отримання відомостей від фахівців і їх застосування. Досить часто в групах виникають проблеми з обміном інформацією не тільки через нерозуміння мови

предметної області, але також і через невизначеність мови самого по собі. Процес DDD має на меті не тільки реалізацію використовуваної мови, але також поліпшення та уточнення мови предметної області. Це, в свою чергу, позитивно відбивається на створюваному ПЗ, оскільки модель є прямою проекцією мови предметної області.

Основними перевагами стилю DDD є:

– обмін інформацією. Всі учасники групи розробки можуть використовувати модель предметної області та описувані нею суті для передачі відомостей і вимог предметної області за допомогою спільної мови предметної області, не вдаючись до технічного жаргону;

– розширюваність. Модель предметної області часто є модульною і гнучкою, що спрощує оновлення і розширення при зміні умов і вимог;

– зручність тестування. Об'єкти моделі предметної області характеризуються слабкою зв'язаністю, що полегшує їх тестування.

Основні визначення DDD-архітектури:

– домен: предметна область, середовище, галузь. Предметна область, яку програміст використовує при створенні програмного забезпечення;

– модель: система абстракцій, яка описує окремі аспекти предметної області;

– загальна мова: мова побудована навколо моделі предметної області. Використовується як програмістами при написанні програмного забезпечення, так і іншими членами команди (експертами обраної галузі);

– контекст: середовище, в якому предмет або дія означує своє значення.

Для забезпечення розширюваності та масштабування додатку використаємо паттерни розробки. Їх буде два:

– Model-View-ViewModel;

– Dependency Injection.

На основі цих паттернів буде будуватися основна структура та архітектура додатку. Ці паттерни дозволять забезпечити:

– кожен об’єкт виконує лише один обов’язок;

– програмні сутності відкриті для розширення, але закриті для змін;

– об’єкти в програмі можуть бути заміщені їх нащадками без змін коду;

– багато спеціалізованих інтерфейсів

– модулі вищих рівнів не залежать від модулів нижніх рівнів.

Model-View-ViewModel (MVVM) дозволяє створювати додаток зі слабкою зв’язністю завдяки розподіленню його на шари. Таким чином, code behind у класах представлення не буде містити в собі логіки, не буде повторення коду та дизайн додатку не буде залежати від логіки. Схема паттерну MVVM представлена на рисунку 3.4.

Шаблон MVVM ділиться на три частини:

– модель являє собою фундаментальні дані, необхідні для роботи програми;

– представлення – це графічний інтерфейс, тобто вікно, кнопки і т. п. Представлення є підписником на подію зміни значень властивостей або команд, що надаються моделлю уявлення;

– модель представлення є, з одного боку, абстракцією представлення, а з іншого, надає обгортку даних з моделі, які підлягають скріпленню. Тобто, вона містить модель, яка перетворена до представлення, а також містить у собі команди, якими може користуватися представлення, щоб впливати на модель.

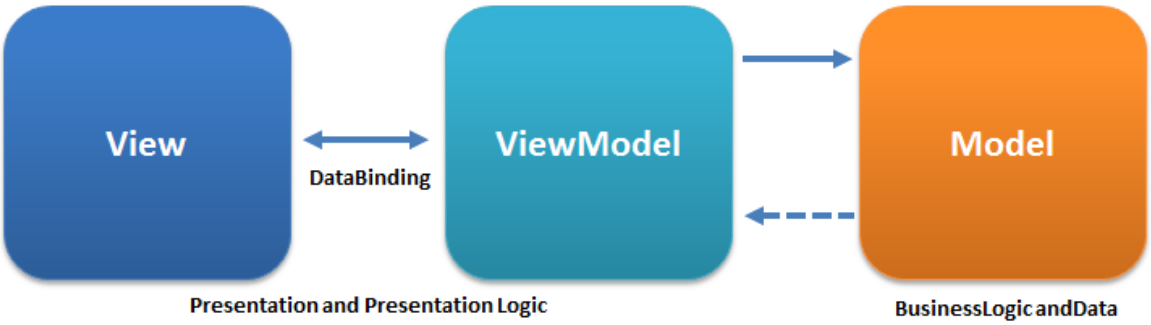

Рисунок 4.4 – Схема паттерну Model-View-ViewModel

Впровадження залежності (англ. Dependency injection, DI) - процес надання зовнішньої залежності програмному компоненту. Є специфічною формою

«інверсії управління» (англ. Inversion of control, IoC), де зміна порядку зв'язку здійснюється шляхом отримання необхідної залежності (рис. 4.5).

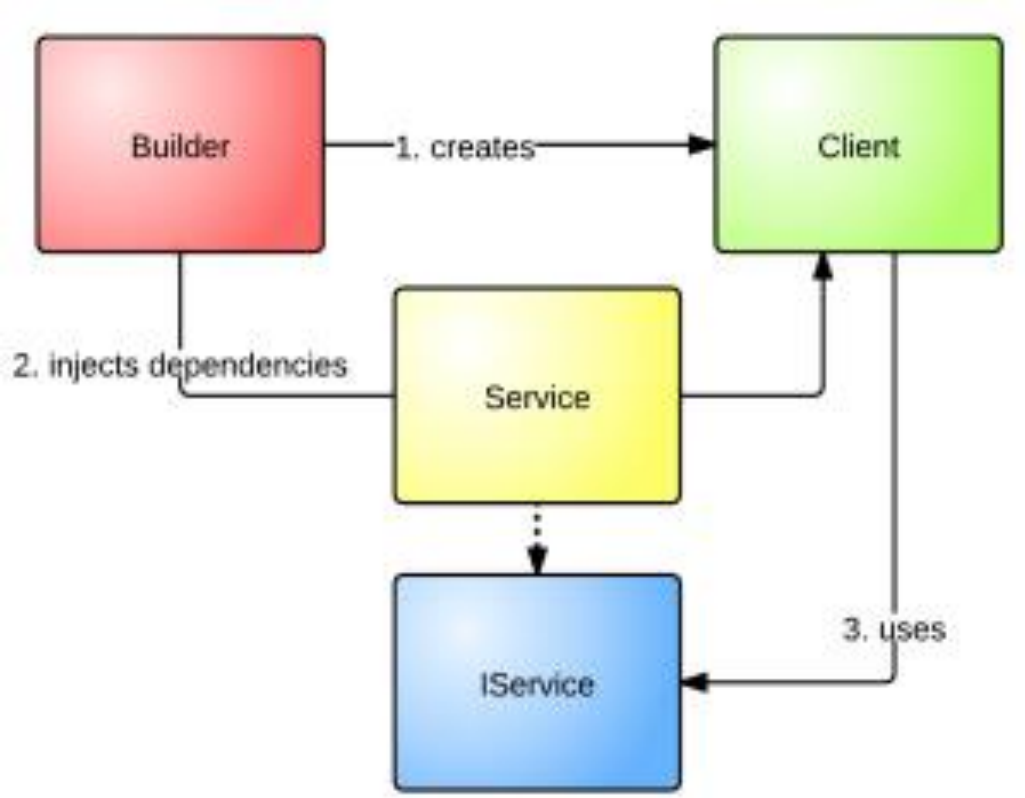

Рисунок 4.5 – Схема роботи паттерну Dependency Injection

Умовно, якщо об'єкту потрібно отримати доступ до певного сервісу, об'єкт бере на себе відповідальність за доступ до цього сервісу: він або отримує пряме посилання на місцезнаходження сервісу, або звертається до відомого «сервіс-локатору» і запитує посилання на реалізацію певного типу сервісу. Використовуючи ж впровадження залежності, об'єкт просто надає властивість, яка в змозі зберігати посилання на потрібний тип сервісу; і коли об'єкт створюється, посилання на реалізацію потрібного типу сервісу автоматично вставляється в цю властивість (поле), використовуючи засоби середовища.

Завдяки цьому, ми зможемо позбутися залежності шару з логікою від якогось конкретного репозиторію, чи якоїсь конкретної соціальної мережі. Це досягається тим, що усі сервіси спілкуються з логікою за допомогою інтерфейсів.

Впровадження залежності більш гнучко, бо стає легше створювати альтернативні реалізації даного типу сервісу, а потім вказувати, яка саме реалізація повинна бути використана в, наприклад, конфігураційному файлі, без змін в об'єктах, які цей сервіс використовують. Це особливо корисно в юніт-тестуванні, бо вставити реалізацію «заглушки» сервісу в тестований об'єкт дуже просто.

З іншого боку, зайве використання впровадження залежностей може зробити програми більш складними і важкими в супроводі: так як для розуміння поведінки програми програмістові необхідно дивитися не тільки у вихідний код, а ще й у конфігурацію, а конфігурація, як правило, невидима для IDE, які підтримують аналіз посилань і рефакторинг, якщо явно не зазначена підтримка фреймворків з впровадженнями залежностей.

Таким чином, для розроблюваної програми було використано не тільки стиль DDD, а й багатошаровий та багаторівневий стилі.

На рисунку 4.6 зображено кінцеву архітектуру додатку.

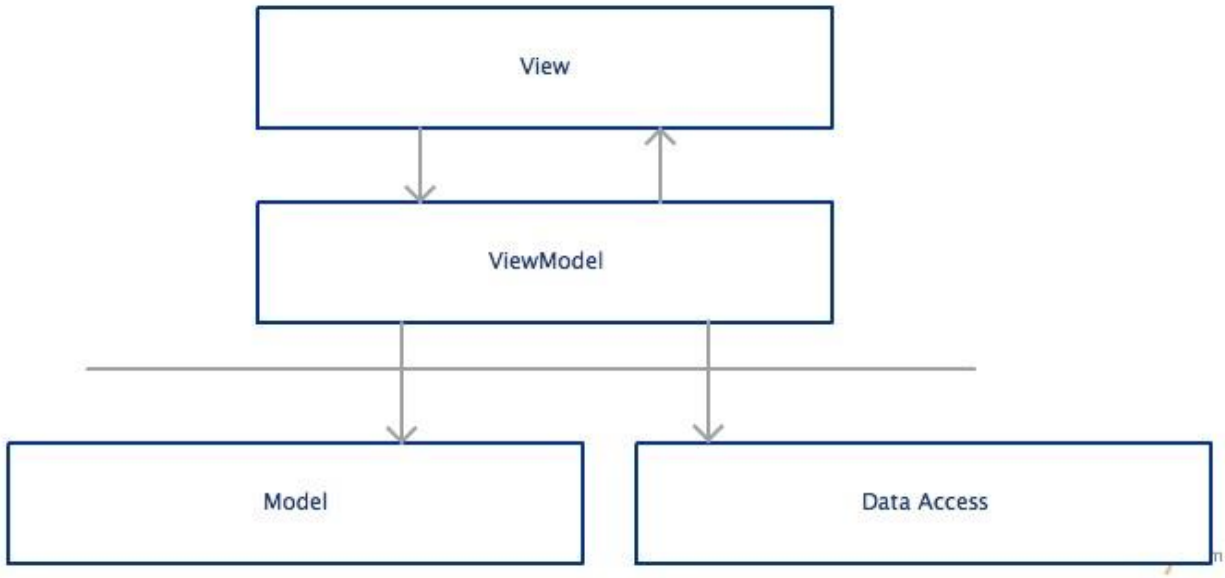

Рисунок 4.6 – Схема архітектури

На цій схемі можна побачити, що додаток має два рівні. Перший рівень містить в собі Model та Date Access. Model мітить елементи предметної області.

Date Access дозволяє отримати, зберегти та змінити налаштування додатку. У шарі Date Access реалізований паттерн DI. Другий рівень – це рівень представлення. У нашому випадку це javascript клієнт. Він, в свою чергу розбитий на два шари: View та ViewModel. View – це шар, що містить в собі розмітку HTML, в якій відбувається виклик методів та отримання інформація для роботи програми. ViewModel використовується для забезпечення передачі інформації між View та рівнем сервісів. В результаті маємо гнучку, легко масштабовану архітектуру створюваного додатку, що дозволить з легкість додати будь-які нові соціальні мережі для використання та будь-які клієнти, оскільки усі сервіси відокремленні у сторонній рівень.

Таким чином маємо архітектурне рішення, що включає в себе серверну частину, де буде відбуватися обробка і зберігання даних, та клієнти, що будуть відображати контент. Для реалізації такої архітектури необхідно використовувати RESTFull з’єднання між клієнтом та сервером. Клієнт можна реалізовувати за допомогою мови програмування javascrip, оскільки вона є кросплатформенною та дасть можливість створювати мобільні додатки для різних операційних систем. Сервер має бути створений на мові програмування C#, оскільки ця мова має багато переваг для створення серверного функціоналу, таких як: легкий доступ до даних, швидкість роботи, легкість інтеграції та розгортання на хмарних інструментах.

4.3 Інструменти розробки

Основні інструменти розробки включають в себе:

– MS Visual Studio 2019;

– MS Visual Studio Code;

– Azure;

– Visual Studio Team Fundation.

MS Visual Studio 2019 дозволить створювати серверну частину. Вона дає можливість зручно писати C#-код. Використовуємо останню версію продукту, оскільки в ній вже вбудовані механізми ASP.NET 5. Це забезпечить найновітніший функціонал на сьогоднішній день.

MS Visual Studio Code ма широкі можливості щодо редагування javascript коду та html розмітки, оскільки увесь синтаксис підсвічується. Також з нею зручно використовувати Git, отже таким чином відпадає необхідність у використанні консольного вікна для використання систему управління версіями Git.

Azure буде використовуватися як майданчик для розсортування серверу. Azure – це хмарне застосування, що гарно працює з .Net додатками. Таким чином, ми матимемо можливість заздалегідь багато можливостей для масштабування та розширення.

Visual Studio Team Fundation (VSTF) – майданчик для зберігання різних версій коду та його інтеграції на майданчик розгортання. VSTF має вбудований функціонал з побудови проекту, запуску тестів та його релізу.

Серверна частина побудована за допомогою нової технології від компанії Microsoft – ASP.NET 5, оскільки ASP.NET 5 привносить значні зміни в платформу ASP.NET. ASP.NET 5 – це позбавлений всього надлишкового .NET стек для побудови сучасних веб-додатків. Цей фреймворк був побудований з нуля, таким чином він є оптимізованою платформою для розробки додатків, які будуть розгорнуті в хмарі або працювати на власних серверах. Для підтримки гнучкості при побудові рішень дана платформа складається з модульних компонентів з мінімальними накладними витратами.

ASP.NET 5 включає в себе наступні можливості:

– нова гнучка і кросплатформенна середа виконання;

– новий модульний конвеєр для HTTP-запитів;

– конфігурація готова до використання в хмарі;

– уніфікована програмна модель, яка поєднує в MVC, Web API і Web Pages;

– можливість побачити зміни без повторного побудови проекту;

– використання декількох версій .NET Framework пліч-о-пліч;

– можливість self-hosting або хостингу на IIS;

– нові інструменти в Visual Studio 2019;

– відкритий вихідний код в GitHub.

Найголовнішою перевагою ASP.NET 5 є те, що коли ви створюєте новий проект, цей проект структурується для легкого розгортання в хмарі. Visual Studio 2019 надає нову систему конфігурації середовища, яка замінює файл Web.config. Нова система дозволяє запитувати іменовані значення з різних джерел (наприклад, JSON, XML або змінні середовища). Ви вказуєте значення для кожного середовища, і після розгортання ваше додаток просто читає коректні значення. Також включені інструменти для діагностики і трасування, які спрощують виявлення проблем додатків в хмарі.

Також слід зазначити, що ASP.NET 5 дозволяє розміщувати свій додаток на IIS або в режимі self hosting. Коли ви використовуєте Core CLR, ви можете розгорнути додаток з усіма залежностями зібраними в пакет розгортання. Таким чином, додаток і його залежності повністю автономні і більше не залежать від установки .NET в системі. Додаток може працювати на будь-якому типі пристрою або хостингової платформи. Ця нова можливість дає багато свободи. Налаштування хостингу більш не визначають, який фреймворк використовувати для розробки і навпаки. Наразі, для створення даного програмного продукту, я використовував операційну систему OS X.

Таким чином, зваживши усі недоліки та переваги платформи ASP.NET 5, можемо з впевненістю зробити висновок, що використання даного інструменту є найкращим варіантом на сьогоднішній день для розробки API.

Була поставлена задача створювати шар доступу до даних на основі nosql бази даних. Для цього напишемо простий модуль, що буде працювати за рахунок документів, що під час запуску будуть завантажуватися у оперативну пам’ять хостингу. Така модель використання дозволить збільшити швидкість відгуку додатку в багато разів, оскільки читання з оперативної пам’яті відбувається набагато швидше ніж з бази даних, що знаходиться на твердотільному носії. Для

збереження даних з додатку відбувається запис їх у документи у вигляді json-об’єктів, тож в разі потреби, такі документи можна буде прочитати навіть за допомогою javascript.

В серверній частині реалізований шаблон проектування Model-View-Controller. Завдяки цьому шаблону увесь додаток поділено на шари, кожен шар зв’язується з шаром нижчого рівня за допомогою впровадження залежностей. Шари нижнього рівня нічого не знають про шари верхнього рівня. Також активно використовується розбиття на модулі. Таким чином, додаток можливо з легкістю інтегрувати в інші системи, змінити, масштабувати, додати новий функціонал.

Для створення API було використано Web-api технологію. API (інтерфейс програмування додатків, інтерфейс прикладного програмування) - набір готових класів, процедур, функцій, структур і констант, що надаються додатком (бібліотекою, сервісом) або операційною системою для використання у зовнішніх програмних продуктах. Платформа Web-api ASP.NET дозволяє з легкістю створювати служби HTTP для широкого діапазону клієнтів, включаючи браузери і мобільні пристрої. Web-api ASP.NET ідеально підходить для розробки додатків RESTful на платформі .NET Framework. Під час створення api були використані та враховані усі важливі аспекти, що покращують розуміння API сторонніми розробниками без необхідності довгих роз’яснень. Слід зазначити, що виклик кінцевих точок відбувається в асинхронному режимі, який дозволить працювати додатку без затримок у багатопоточному режимі.

На етапі постановки задачі було визначено, що для створення клієнтських додатків слід використовувати мову програмування javascript. Для запуску javascript-додатків на мобільних пристроях використовується платформа NativScript.

Native script (NS) – це бібліотека, що дозволяє робити Кросплатформені додатки, використовуючи XML, CSS, JavaScript. Native script вирішує ту ж задачу, що і вже всім відомий phonegap (створення крос-платформних додатків), але підходи у них різні. Phonegap використовує движок браузера, щоб відобразити ваш UI (фактично ви отримуєте веб-сторінку), Native script використовує

нативний рендеринг, використовує елементи нативного UI. Наступна важлива відмінність: щоб отримати доступ до камери, gps і так далі в phonegap необхідно встановлювати плагіни, в той час як NS дає доступ з коробки. Варто підкреслити, що додатки можна писати для Android 4.2 і вище, і для iOS 7.1 і вище.

4.4 Безперервна інтеграція і безперервне розгортання

Для того, щоб мати можливість зручно тестувати та випускати продукт у користування, необхідно продумати систему безперервної (неперервної) інтеграції і безперервного (неперервного) розгортання.

Неперервна інтеграція (англ. Continuous Integration) – це практика розробки програмного забезпечення, яка полягає у виконанні частих автоматизованих складань проекту для якнайшвидшого виявлення та вирішення інтеграційних проблем. У звичайному проекті, де над різними частинами системи розробники працюють незалежно, стадія інтеграції є завершальною. Вона може непередбачувано затримати закінчення робіт. Перехід до неперервної (постійної) інтеграції дозволяє знизити трудомісткість інтеграції і зробити її передбачуванішою за рахунок найбільш раннього виявлення та усунення помилок і суперечностей.

Вимоги до проекту:

– сирцеві коди і все, що необхідно для побудови та тестування проекту, зберігається в репозиторії системи керування версіями;

– операції копіювання з репозиторію, складання і тестування всього проекту автоматизовані і легко викликаються із зовнішньої програми.

Усе це дозволяє робити Visual Studio Team Fundation. За допомогою вбудованих інструментів зі збереження версій коду та build-системи, маємо можливість автоматично створювати нові версії програмної системи, готові до розгортання на сервері, під час кожного додавання нового кластеру коду.

Безперервне розгортання являє собою програмний інженерний підхід, в якому команди виробляють програмне забезпечення в коротких циклах, гарантуючи, що програмне забезпечення може бути випущений в будь-який час. Він спрямований на створення, тестування і випускання програмного забезпечення швидше і частіше. Такий підхід дозволяє знизити витрати, час і ризик розгортання, що дозволяє створювати більше додаткових оновлень для додатків в виробництві. Прямолінійний і повторюваний процес розгортання має важливе значення для безперервної доставки програмного продукт користувачам.

Таким чином, після створення нового build, завдяки інструменту Release, маємо можливість налаштувати безперервне розгортання. Реліз програмного продукту відбувається в два етапи: створення навколишнього середовища для тестування та створення навколишнього середовища для релізної версії. Навколишнє середовище для тестування створюється автоматично для кожного build, а версію для готового продукту будуємо тільки після тестування.

Отже для даного програмного продукту маємо набір скриптів для швидкого створення нових версій додатку, подальшого тестування та відправки користувачам. Це дозволяє прискорити процес розробки та тестування програмного продукту у декілька разів, оскільки немає необхідності в ручному розгортанні.

4.5 Інтерфейс користувача

Для кращого розуміння того, як має виглядати кінцевий програмний продукт, необхідно побудувати концепцію інтерфейсу користувача. Він має містити основні елементи керування на різних сторінках. Такий підхід до проектування програмної системи дозволить заздалегідь створити якісний інтерфейс користувача з мінімальними затратами часу. Такі концепти мають назву макети.

Оскільки наразі ринок порівну поділяється на android-пристрої та на iOS-пристрої, існує декілька підходів для створення інтерфейсу мобільних додатків з урахуванням різних операційних систем та принципів їх робочих елементів:

– інтерфейс додатку для OS Android та iOS відрізняється та побудований на основі стандартних елементів керування операційної системи;

– інтерфейс додатків для обох пристроїв має однаковий дизайн, що не являється стандартом для жодної операційної системи;

– стандартні елементи керування перекочовують з однієї операційної системи на іншу.

Усі принципи мають свої переваги та надолі. Перший, коли інтерфейси на різних операційних системах відрізняються, має за недолік важкість розробки, оскільки одну й ту саму роботу доведеться виконувати двічі, користувачам важче буде пристосовуватися до керування на пристроях, в яких операційна система відрізняється від їхньої, але на своїй - легше. Другий, коли інтерфейси однакові, найлегший для розробників, має за перевагу легкість пристосування під час переходу від пристрою до пристрою користувачами, оскільки їм буде завжди зрозуміло що й до чого. Третій варіант активно використовує компанія гугл під час створення своїх клієнтів для iOS, і таке прийнятне лише для них, оскільки вони мають свою власну концепцію створення інтерфейсу, що активно використовується в операційній системі Android.

Проаналізувавши кожен з варіантів створення інтерфейсу користувача, приходимо до висновку, що для розроблюваного додатку прийнятний другий варіант, оскільки вартість такого варіанту дуже низька, а доки в системі не буде необхідної кількості користувачів, немає сенсу створювати щось коштовне.

Основний робочий екран повинен мати шапку, де користувачі зможуть викликати меню та вийти зі свого аккаунту. Меню має складатися лише з іконок, що будуть виїжджати злива. Таким чином, додатком буде зручно користуватися та він не буде перенавантажений різними контролерами.

Наступним блоком є мапа, де відобразимо маркери з вже існуючими паркінгами. Мапа має відкриватись на весь екран та закриватись лише під час

клику на один з маркерів. Завдяки цій мапі користувачі зможуть знайти паркінг поблизу якнайшвидше.

Після кліку на один з маркерів, повинна з’явитися додаткова інформація про паркінг. Тут користувач зможе зробити маніпуляцій зі сплати за місце чи перевірки наявності вільних місць.

Інші екрани матимуть схожу структуру, тож немає потреби розписувати їх детально, для розуміння концепції має вистачити наявного опису. Але слід зазначити, що мають бути наступні екрани:

– екран пошуку паркінгу;

– екран редагування персональної інформації;

– екран авторизації;

– екран реєстрації;

– екран адміністрування.

На рисунку 4.7 зображено головний екран програми. Сюди користувачі потраплятимуть відразу після авторизації.

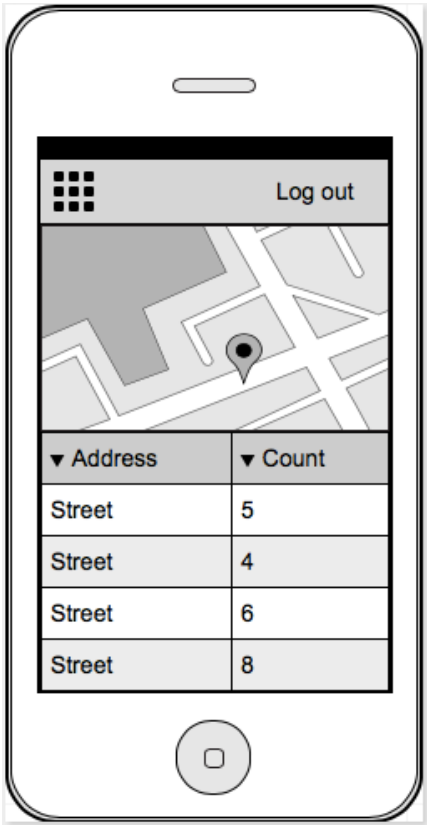

Рисунок 4.7 – Головний екран програми

Веб-сайт планується як додатковий інструмент для роботи з сервісом. Він не буде використовуватися як основний оскільки керувати паркінгами найчастіше необхідно за відсутності стаціонарного комп’ютера. Але в разі, якщо користувач схоче перевірити статус свого велосипеду зі стаціонарного комп’ютера, він матиме таку можливість.

Отже, веб-сайт має повторювати весь функціонал, що має мобільний додаток. В такому разі краще створювати його в єдиному стилі з додатком, але вже з урахуванням, що буде використовуватися великий екран.

Таким чином можемо побудувати та відобразити усі побажання на макеті. На рисунку 4.8 зображено макет екрану веб-сайту відкритого у браузері. Як можна побачити, він має схожий інтерфейс з мобільними додатками.

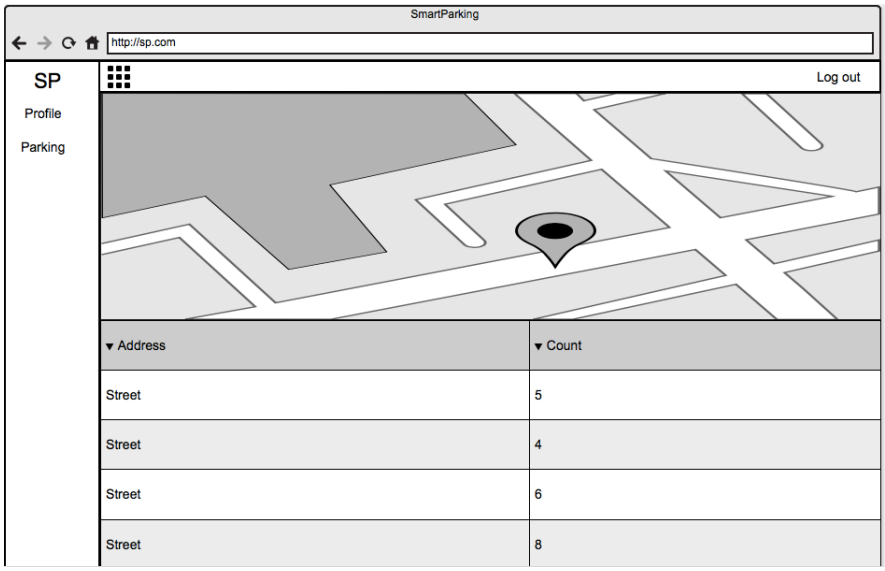

Рисунок 4.8 – Головний екран веб-сайту

Макет веб-сайту, так само як и макет мобільного додатку, має верхній бар, мапу та інформацію під мапою після кліку на маркер. Завдяки великому екрану, маємо більше можливостей для розміщення інформації більш ергономічно, тож бокове меню замість іконок матиме назви розділів. Як и в мобільному додатку, меню можливо буде відкрити і закрити.

5 ОПИС РОЗРОБЛЕННОЇ ПРОГРАМНОЇ СИСТЕМИ

Під час розробки програмного продукту було створено веб-сайт, що може працювати у веб-браузері та на мобільних пристроях.

Для логіну існує дві ролі: адміністратора та користувача. Якщо зайти на сайт як адміністратор, отримаємо сторінку для керування паркінгами (рис. 5.1). На даному екрані зображено мапу з списком паркінгів.

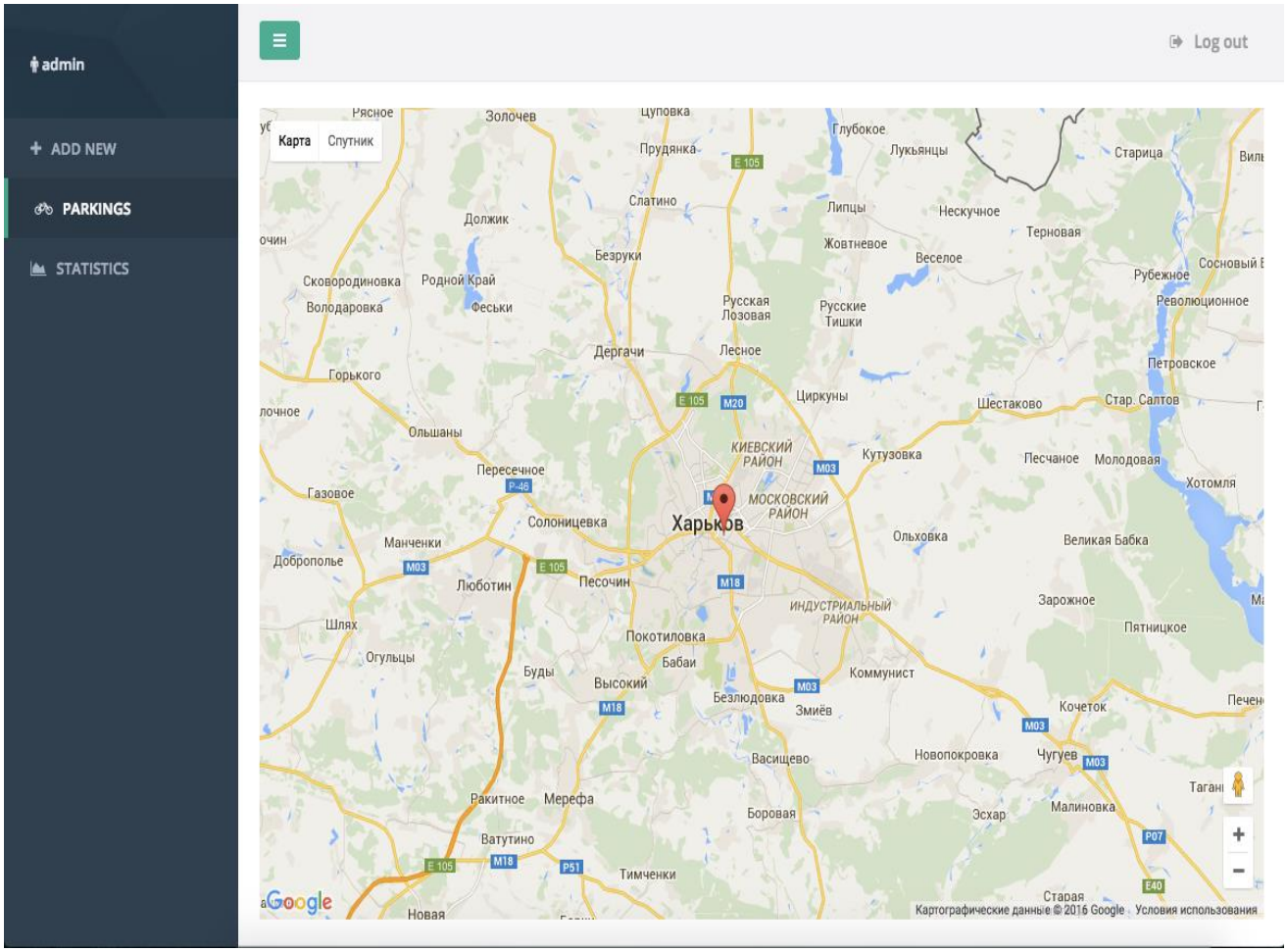

Рисунок 5.1 – Вікно менеджменту

У верхньому лівому кутку бачимо, що маємо користувача саме з роллю адміністратора. Є можливість створити новий паркінг або подивитися статистику використання паркінгів.

На рисунку 5.2 зображено логін вікно для веб-сайту. На даній сторінці існує можливість авторизуватися як користувач чи адміністратор або перейти до вікна створення нового користувача.

Рисунок 5.2 – Логін вікно

На рисунку 5.3 зображено вікно реєстрації з непройденою валідацією.

Рисунок 5.3 – Вікно реєстрації

Така валідація існує для усіх форм у програмній системі, таким чином, користувачі не матимуть можливість додати невірні дані.

Зареєструватися можливо тільки як "користувач".

Після реєстрації або логіна усі користувачі потрапляють на сторінку пошуку паркінгу на мапі. На рисунку 5.4 зображено усі додані на даний момент паркінги. Оскільки користувачі будуть здебільшого користуватися застосунком з мобільних пристроїв, додаємо скріншоти з мобільного вигляду.

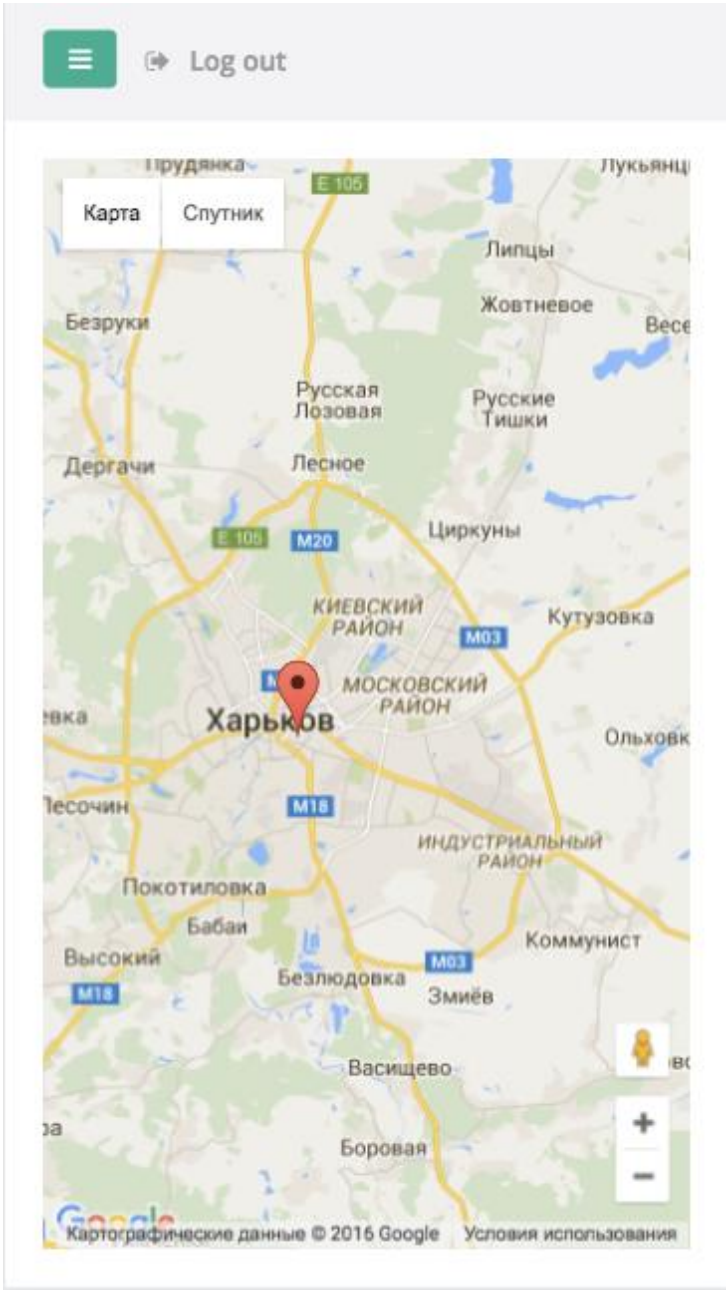

Рисунок 5.4 – Вікно пошуку валопаркінгу

Після того як користувач знайшов свою велосипедний паркінг, він має натиснути на маркер, якщо бажає скористатися паркінгом. На рисунку 5.5

зображено вікно, на якому є можливість замкнути свій велосипед. Якщо комірка вже зайнята кимсь іншим, вона буде підсвічуватися червоним кольором, відімкнути її матиме можливість лише користувач, який її замкнув.

Рисунок 5.5 – Вікно роботи к комірками для велосипедів

Для замикання комірки користувач має натиснути на кнопку "Замкнути". Якщо паркінг має платню, користувач потрапить на вікно сплати, інакше комірка замкнеться. Після успішної сплати та замикання комірки, користувач має

можливість відслідковувати статус свого велосипеду. Для цього необхідно перейти до вікна статус, що зображене на рисунку 5.6.

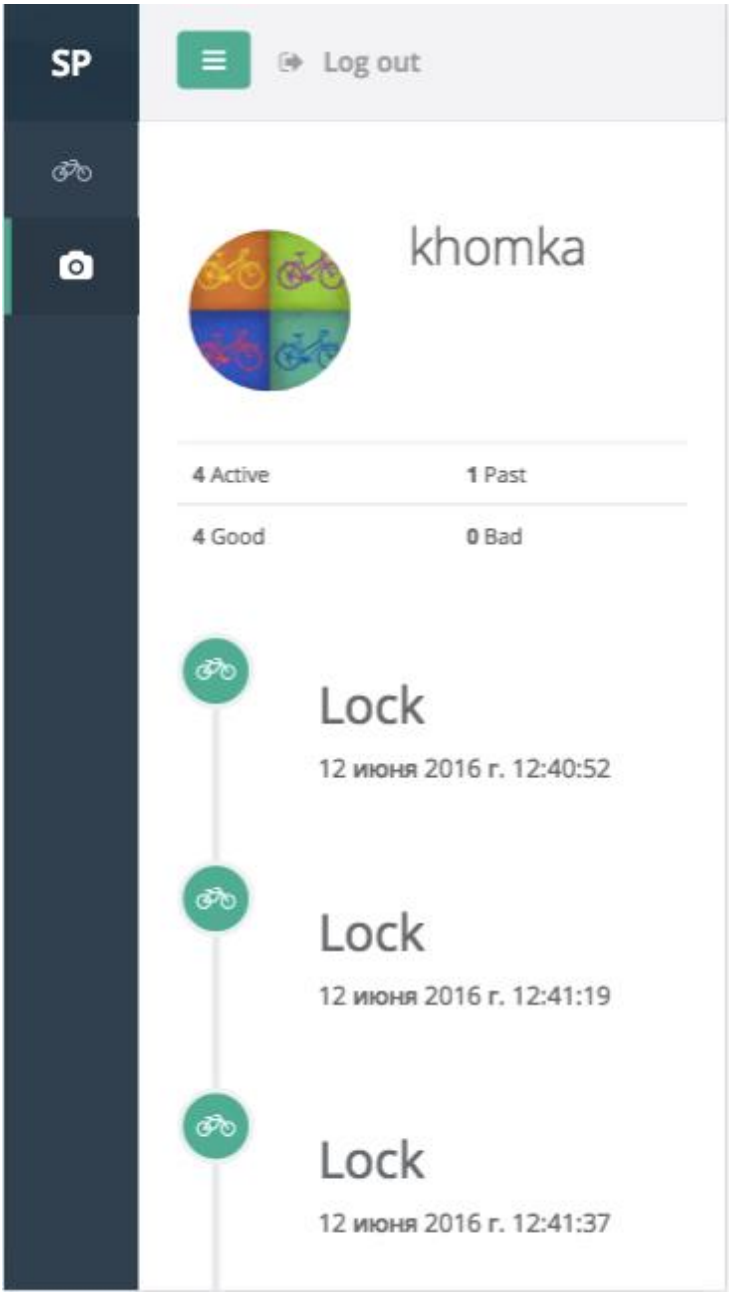

Рисунок 5.6 – Вікно статусу велосипеду

На даному вікні у верхній частині користувач може побачити своє ім’я, кількість активних замкнутих велосипедів, кількість велосипедів, що були відімкнуті та інформацію про статус активних комірок. Якщо щось трапиться з велосипедом, статус "Bad" збільшиться на один пункт. Далі йде інформація у вигляді ленти про велосипеди та статуси.

6 ТЕСТУВАННЯ

6.1 Опис методу тестування

Тестовий випадок (Test Case)[16] – це сукупність кроків, конкретних умов і параметрів, необхідних для перевірки реалізації всієї системи загалом або її окремої частини. Основним методом організації тестування є використання тест-кейсів.

Під тест-кейсом розуміється структура виду:

– Action > Expected Result > Test Result.

Кожен тест кейс повинен мати 3 частини:

– preConditions – список дій, які призводять систему до стану придатного для проведення основної перевірки. Це список умов, виконання яких свідчить про те, що система знаходиться в придатному для проведення основного тесту стані;

– test Case Description – список дій, що переводять систему з одного стану в інший для отримання результату, на підставі якого можна зробити висновки про відповідність реалізації поставленим вимогам;

– postConditions – список дій, що переводять систему в первинний стан (до початку проведення тесту) – initial state.

6.2 Результати тестування системи

Головне вікно програми не перевантажено зайвою інформацією, немає незрозумілого інтерфейсу. Після запуску користувач бачить інтуїтивно зрозумілий інтерфейс, яким легко користуватися.

Результати тестування працездатності програми представлено у вигляді таблиці (див. табл. 6.1). Загалом було проведено перевірку основних дій – додавання нового аккаунту користувача для усіх доступних соціальних мереж, відправка повідомлення у мережі.

Таблиця.6.1 – Тестування програми

Назва:	Створення нового паркінгу	
Функція:	Створення нового паркінгу	
Дія	Очікуваний результат	Результат тесту: – пройдено – провалено – заблоковано
Проходимо авторизацію як адміністратор	Відображається адміністративна панель	пройдено
Обираємо "Додати паркінг"	Відображається порожня мапа	пройдено
Клікаємо на мапі на місце з паркінгом	Відкривається вікно введення та збереження даних	пройдено
Зберегти	Відкривається мапа з паркінгами	пройдено
Назва:	Активація паркінгу	
Функція:	Активація паркінгу	
Дія	Очікуваний результат	Результат тесту: – пройдено – провалено – заблоковано
Проходимо авторизацію як користувач	Відкривається мапа	пройдено
Обираєм паркінг	З’являється інформація про паркінг	пройдено
Активуємо	Отримуємо статус велосипеда	пройдено

З результатів тестування програми видно що функціонал програми працює вірно, збоїв не виявлено. Інтерфейс програми зручний у використанні, труднощів з його використанням не виникає.

ВИСНОВКИ

В ході виконання атестаційної роботи було виконано:

– підготовка для створення програмного продукту;

– проведений аналіз ринку та конкурентів;

– обґрунтування обраної платформи для першочергової реалізації додатку;

– побудовані вимоги та поставлено завдання до майбутнього застосунку;

– робота з проектування програмної системи;

– створено UML-діаграми, що допоможуть реалізовувати програмний продукт з узгодженням усіх вимог;

– розроблено програмний продукт;

– проведено тестування програмного продукту.

Отже, було виявлено, що кількість потенційних користувачів залежить лише від кількості велосипедних паркінгів. Для великої кількості користувачів необхідно будувати та обладнувати програмним забезпеченням більше паркінгів. Проведений аналіз конкурентів показав, що наразі не існує прямого конкурента чи аналога. Тому можна вважати проект унікальним. Також, для полегшення етапу розробки було створено необхідні UML-діаграми, що дозволить розробити якісну програмну систему та пришвидшить процес розробки і тестування. Для кращого розуміння роботи додатку була розроблена архітектура серверної частини та макети для клієнтських додатків.

Розробка завершена повністю. У майбутньому планується провести інтеграцію з апаратною частиною велосипедних паркінгів, що в свою чергу дозволить користуватися моєю програмною системою, розробленою в рамках даної атестаційної роботи.

Лістинг коду програми представлено у додатку Б.

Розроблено презентацію (див. додаток В).

Усі матеріали з атестаційної роботи наведено на диску (див. додаток Г).

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Єжова Л. Інформаційний маркетинг: [Текст] / Єжова Л. — К. : КНЕУ, 2002. — 560с.

2. Стартап [Електронний ресурс]/ Вікіпедія – Україна – Режим доступу: https://uk.wikipedia.org/wiki/Стартап - Загол. з екрану.

3. Сколько в мире велопарквок [Електронний ресурс]/ UNIAN – Україна – Режим доступу: http://www.unian.net/sport/28548-skolko-v-mire-velosipedov.html. - 01.03.2016 р. - Загол. з екрану.

4. Топ 10 стран по количеству велосипедов [Електронний ресурс]/ thenextweb – Україна – Режим доступу: http://xn--b1adef0ban2h.com.ua/popularly/524-top-10-stran-s-naibolshim-kolichestvom-velosipedov-na-dushu-naseleniya. - 29.04.2016 р. - Загол. з екрану.

5. Велиб [Електронний ресурс]/ unian – Франція – Режим доступу: http://www.velib.paris. - 30.04.2016 р. - Загол. з екрану.

6. Anastasi G., Antonelli M., Bechini A., Brienza S., de Andrea E., de Guglielmo D., Ducange P., Lazzerini B., Marcelloni F., Segatori A. Urban and social sensing for sustainable mobility in smart cities; Proceedings of the 2013 Sustainable Internet and ICT for Sustainability (SustainIT); Palermo, Italy. 30.10.2013.

7. Barone R.E., Giuffrè T., Siniscalchi S.M., Morgano M.A., Tesoriere G. Architecture for parking management in smart cities. - IET Intell. Transp. Syst. 11.11.2015 p. -325c.

8. Gupta A., Sharma V., Ruparam N.K., Jain S., Alhammad A., Ripon M.A.K. Integrating pervasive computing, InfoStations and swarm intelligence to design intelligent context-aware parking-space location mechanism; Proceedings of the International Conference on Advances in Computing, Communications and Informatics (ICACCI); Delhi, India. 24.10.2014.

9. He W., Yan G., Xu L.D. Developing vehicular data cloud services in the IoT environment - IEEE Trans. Ind. Inform., 14.10.2017 p. -1405c.

10. Vlahogiannia E.I., Kepaptsogloua K., Tsetsosa V., Karlaftisa M.G. A real-time parking prediction system for smart cities. J. Intell. Transp. Syst. Technol. Plan. Oper. 14.10.2017 p. -1405c.

11. Інтернет речей [Елекронний ресурс]/ Вікіпедія – Україна – Режим доступу: https://uk.wikipedia.org/wiki/Інтернет_речей. - 12. 04.2016 р. – Загол. з екрану.

12. Буч, Г., Язык UML. Руководство пользователя [Текст] / Г. Буч, – М.: ДМК Пресс, 2-е издание, 2006. – 248 с.

13. Microsoft Patterns & Practices Team: Руководство MICROSOFT по проектированию архитектуры приложений (2 издание), Radmond, USA: Microsoft; 2010. – 210 с.

14. Байдачный, С. Windows 8 для C# разработчиков [Текст] / С. Байдачный, С. Лутай. – Х.: «Самиздат», 2011. – 278 с.

15. Пугачев, С. Разработка приложений для Windows 8 на языке C# [Текст] / С.В. Пугачев, А.М. Шериев, К.А. Кичинский. – СПб.: БХВ-Петербург, 2013. – 312 с.

16. Савин Р., Тестирование Дот Ком [Текст] / Р. Савин,– М.: Дело, 2007. – 312 с.

17. Buss, A, Strauss N. Online Communities Handbook: Building Your Business and Brand on the Web. Berkeley, CA: New Riders; 2009.

18. Microsoft.com [Електронний ресурс] / Операційна система Windows 8 – режим доступу: www/URL – 12. 04.2016 р. – Загол. з екрану.

19. Unian – Україна [Електронний ресурс]: «Российские социальные сети теряют популярность в Украине». – Режим доступу: http://www.unian.net/science/945548-rossiyskie-sotsialnyie-seti-teryayut-populyarnost-v-ukraine.html. – 12. 04.2016 р. – Загол. з екрану.

20. Фаулер, М., Скотт К., UML Основи [Текст] / М. Фаулер, К. Скот, – Пер. с англ. – СПб: Символ-Плюс, 2002. – 192с.