Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра Комп'ютерних систем управління

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

" Розробка клієнт-серверної частини системи ідентифікації місць паркування паркувальних майданчиків."

Виконав: студент 2 курсу, групи
3АКІТР-23м спеціальності 174 –
«Автоматизація, комп'ютерно-
інтегровані технології та робототехніка»
Сергій ГУСАК
(ім'я та прізвище)

Керівник: к.т.н., доц. каф. АІІТ
Володимир КОЦЮБИНСЬКИЙ
(ім'я та прізвище)
«4» 12 2024 р.

Опонент: д.т.н., професор каф. КН
Ярослав ІВАНЧУК
(ім'я та прізвище)
«4» 12 2024 р.

Допущено до захисту
Зав. кафедри КСУ, д.т.н., проф.
В'ячеслав КОВТУН
(ім'я та прізвище)
«4» 12 2024 р.

Вінниця ВНТУ - 2024 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра Комп'ютерних систем управління
Рівень вищої освіти другий (магістерський)
Галузь знань – 17 – Електроніка, автоматизація та електронні комунікації
Спеціальність – 174 – Автоматизація, комп'ютерно-інтегровані технології та робототехніка
Освітньо - професійна програма – Інформаційні системи і Інтернет речей

ЗАТВЕРДЖУЮ
Зав.кафедри КСУ, д.т.н., проф.
_____В'ячеслав КОВТУН
"_14_" ___жовтня____2024 року

ЗАВДАННЯ
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ

студенту ____Гусаку Сергію Вікторовичу_____
(прізвище, ім'я, по батькові)

1. Тема роботи _Розробка клієнт-серверної частини системи ідентифікації місць паркування паркувальних майданчиків._____
керівник роботи _____к.т.н доц.___ Коцюбинський В.Ю._____
затверджені наказом ВНТУ від "17" вересня 2024 року №310
2. Термін подання студентом роботи "1" грудня_ 2024 року
3. Вихідні дані до роботи: _____
_____1. Офіційний сайт довідкової та технічної документації проєкту OpenStreetMap [Електронний ресурс] - https://wiki.openstreetmap.org/_____
_____2. Офіційний сайт довідкової та технічної документації проєкту PostGis_ [Електронний ресурс] - https://postgis.net/_____
_____3. Офіційний сайт системи паркування SmartParking [Електронний ресурс] - https://www.smartparking.com/uk_____
_____4.Офіційний сайт MapBox [Електронний ресурс] - https://www.mapbox.com/___
4. Зміст текстової частини: _Вступ, Аналіз проблем ідентифікації місць паркування; формування програмних вимог та побудова концепцій продукту; збирання даних та розробка бази даних системи; розробка веб-системи ідентифікації місць для паркування._____
5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень)___ зображення інтерфейсу програми; ER-моделі бази даних, UML діаграма класів, UML діаграма об'єктів, архітектура програмного забезпечення веб-системи; карта з місцями для паркування; карта з відрізками, де паркування заборонене;_____
_____

1. Консультанти розділів роботи

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

2. Дата видачі завдання "14" жовтня 2024 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Аналіз предметної області	02.09.2024	18.09.2024	Виконано
2	Огляд існуючих систем та рішень	19.09.2024	30.09.2024	Виконано
3	Постановка задачі та визначення основних функцій системи	01.10.2024	06.10.2024	Виконано
4	Проектування архітектури серве-	07.10.2024	13.10.2024	Виконано
5	Аналіз та вибір технологій розро-	14.10.2024	20.10.2024	Виконано
6	Розробка системи	21.10.2024	17.11.2024	Виконано
7	Тестування розробленої системи	18.11.2024	24.11.2024	Виконано
8	Оформлення пояснювальної запи-	25.11.2024	01.12.2024	Виконано
9	Захист МКР	12.12.2024	12.12.2024	

Студент _____ Сергій ГУСАК
(підпис) (ім'я і ПРІЗВИЩЕ)

Керівник роботи _____ Володимир КОЦЮБИНСЬКИЙ
(підпис) (ім'я і ПРІЗВИЩЕ)

АНОТАЦІЯ

УДК 004.9+ 303.732.4

Гусак С.В. Розробка клієнт-серверної частини системи ідентифікації місць паркування паркувальних майданчиків.

Магістерська кваліфікаційна робота зі спеціальності 174 – Автоматизація, комп'ютерно-інтегровані технології та робототехніка, освітньо-професійна програма - Інформаційні системи і Інтернет речей. Вінниця: ВНТУ, 2024. 121 с.

На укр. мові. Бібліогр.: 21 назв; рис.: 55; табл. 1.

Метою даної роботи є розробка серверної частини клієнт-серверної системи ідентифікації місць паркування та паркувальних майданчиків на основі даних відкритих джерел та OpenStreetMap. Досліджено можливість знаходження місць для паркування та їх відображення на карті у місті Вінниця з використанням відкритих джерел та даних OpenStreetMap на платформі Java. Запропоновано функціонал для візуалізації частин проїжджої частини, де категорично заборонено зупинку та паркування.

Ілюстративна частина складається з 11 графічних матеріалів із результатами отриманих даних.

Ключові слова: паркування, автомобіль, геостатистичний аналіз, моніторинг, дорога, геоінформаційна система.

ABSTRACT

Husak S.V. Development of the Client-Server Part of a Parking Spot and Parking Lot Identification System (comprehensive MQW).

Master's Qualification Work in Specialty 174 – Automation, Computer-Integrated Technologies, and Robotics, Educational and Professional Program – Information Systems and Internet of Things. Vinnytsia: VNTU, 2024. 121 pages.

In Ukrainian. References: 21 sources; illustrations: 55; tables: 1.

The purpose of this work is to develop the server-side component of a client-server system for the identification of parking spots and parking lots based on open-source data and OpenStreetMap. The study explores the possibility of locating parking spaces and displaying them on a map of Vinnytsia using open-source data and OpenStreetMap on the Java platform. Functionality for visualizing sections of roads where stopping and parking are strictly prohibited has been proposed.

The illustrative section comprises 11 graphical materials showcasing the results of the obtained data.

Keywords: parking, vehicle, geostatistical analysis, monitoring, road, geoinformation system.

ЗМІСТ

ВСТУП

Актуальність теми. Зростання кількості автомобілів і зростаючі потреби в паркуванні вимагають розширення наявних паркувальних місць та поліпшення їх інфраструктури. Однак у деяких районах міст можливості для збільшення кількості парковок обмежені, що призводить до перевантаження цих територій. Крім того, відсутність ефективного механізму контролю за надходженнями від паркування та притягненням порушників до відповідальності створює затори, ускладнює рух і призводить до корупційних схем. В результаті кошти, що мали б надходити до міського бюджету для покращення та розвитку паркувальних майданчиків, не використовуються належним чином. Це також піднімає питання неефективності існуючого регулювання, що негативно впливає на громаду, бізнес і органи місцевої влади.

Мета і завдання роботи. Метою даної роботи є розробка серверної частини клієнт-серверної системи ідентифікації місць паркування та паркувальних майданчиків на основі даних відкритих джерел та OpenStreetMap.

Об’єктом дослідження магістерської кваліфікаційної роботи є процес розробки серверної частини клієнт-серверної системи ідентифікації місць паркування та паркувальних майданчиків на основі даних відкритих джерел та OpenStreetMap.

Предметом дослідження магістерської кваліфікаційної роботи є методи автоматизації процесу знаходження місць для паркування.

Процес розробки інформаційної веб-системи ідентифікації місць для паркування передбачає вирішення таких задач:

– збирання та формалізація вхідних даних для створення веб-системи ідентифікації місць для паркування;

–формування програмних вимог та вибір оптимальних інформаціних технологій;

– вибір та реалізація архітектури системи ідентифікації місць паркуванн;

–створення структури та розробка бази даних системи;

– розробка інформаційної веб-системи ідентифікації місць для паркування;

– практичне використання інформаційної веб-системи ідентифікації місць для паркування.

Практичне цінність роботи полягає у розроблені алгоритмічного та програмного забезпечення інформаційної системи для ідентифікації, відображення та повідомлень користувачів про наявність місць паркування, а також місць, де паркування заборонене.

Новизна одержаних результатів полягає у подальшому розвитку спеціалізованого підходу, щодо розробки веб-системи ідентифікації місць для паркування, шляхом інтегрованого використання механізму залучення відкритих картографічних даних та їх подальшої обробки.

Апробація. Представлені в роботі результати апробовані під час участі у конференції ВНТУ: Всеукраїнській науково-практичній інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи» (2023) та «Науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації (2024)».

Публікації результатів Гусак Сергій Вікторович, Коцюбинський Володимир Юрійович LIII Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації (2024) 2024. URL http://surl.li/chggid

Гусак Сергій Вікторович, Крижановський Євгеній Миколайович «Молодь в науці: дослідження, проблеми, перспективи» (2023) 2023. URL http://surl.li/riinlb

1 АНАЛІЗ ПРОБЛЕМ ІДЕНТИФІКАЦІЇ МІСЦЬ ПАРКУВАННЯ

1.1 Суть технічної проблеми

Розумна парковка – це спеціально обладнане місце для паркування, яке використовує датчики та сучасні технології для полегшення пошуку вільних місць, забезпечення безпеки та автоматизації процесу паркування. Такі системи вирішують питання обмежених паркувальних місць і сприяють впровадженню інновацій в організацію паркувального простору. Парковки почали з'являтися майже одночасно з автомобілями, і з того часу кількість автомобілів невпинно зростає, що створює постійний попит на ефективні рішення для паркування.

Паркування є однією з головних проблем сучасних міст, і Вінниця не є винятком. Згідно з рішенням виконкому Вінницької міської ради, було затверджено зональну схему для встановлення ставок збору за паркування. Враховуючи транспортний трафік і ділову активність, місто поділили на п’ять зон. Ще кілька десятиліть тому автомобіль вважався розкішшю, однак зараз він є необхідним засобом пересування. Однак хаотичне паркування автомобілів стало серйозною проблемою для багатьох міст. Багато житлових будинків, збудованих у радянські часи, не передбачали достатньої кількості паркувальних місць. Архітектори не враховували можливість наявності кількох автомобілів у кожній родині. У сучасних житлових комплексах забудовники намагаються виконувати вимоги державних будівельних норм щодо кількості паркомісць, проте у старих будинках це залишається проблемою. Проте хаотичне паркування автомобілів стало значною проблемою для більшості міст. Більшість багатоповерхових будинків у місті було збудовано ще за радянських часів, коли проектанти зазвичай планували двори для зовсім інших цілей — встановлення стійок для вибивання килимів або, у кращому випадку, кількох гойдалок і пісочниць. Концепція "одна сім'я — один автомобіль", а тим більше "два автомобілі на сім'ю", не була серйозно врахована архітекторами того часу. Тому зараз забезпечити такі умови практично неможливо[1].

У нових житлових комплексах забудовники намагаються дотримуватися вимог державних будівельних норм щодо кількості паркомісць, хоча ці норми варіюються залежно від часу затвердження проекту. Однак, якщо зробити

розрахунок, то для сучасної кількості автомобілів автостоянка у дворі повинна займати вдвічі більше площі, ніж сам житловий будинок. У багатьох містах під час проведення капітальних ремонтів прибудинкових територій обов’язково плануються місця для паркування автомобілів. Проте навіть з урахуванням цього, місць для тимчасового паркування все одно недостатньо (рис. 1.1).

Рисунок 1.1 – Приклад паркувальної зони

Ця проблема особливо гостро проявляється в центрі великих міст, де кількість автомобілів, які потрібно припаркувати на час відсутності власників, постійно зростає. Лише за минулий рік в Україні було зареєстровано 32,8 тисячі вживаних легкових автомобілів, ввезених з-за кордону, що на 35% більше порівняно з серпнем 2019 року. Як результат, безлад у сфері паркування посилюється, і машини часто залишають у заборонених місцях. "Майстри парковки" нерідко паркуються на газонах, дитячих майданчиках, пішохідних переходах і тротуарах. Деякі водії, припарковуючи свої автомобілі, перекривають смугу руху, змушуючи інших порушувати правила дорожнього руху, перетинаючи

подвійну суцільну лінію. Це створює не тільки незручності для водіїв і пішоходів, але й провокує аварійні ситуації.

Пандемія ще більше загострила проблему паркування, оскільки через карантинні обмеження у громадському транспорті дозволено перевозити пасажирів тільки на сидячих місцях. Це змусило мешканців частіше користуватися власними автомобілями, що, відповідно, підвищило попит на паркувальні місця [2].

Майже два роки тому в Україні набули чинності нові правила паркування. Наприклад, тепер для накладення штрафу не потрібно очікувати власника авто — достатньо зафіксувати порушення на фото, а квитанцію надіслати поштою. Однак ці правила не почали повноцінно діяти, оскільки система збору інформації та фотофіксації ще не запрацювала на всі 100%. Тому для ефективного контролю інфраструктура збору й аналізу даних повинна використовувати не лише камери спостереження, а й інформацію від інспекторів та поліцейських.

У деяких містах України додатковим інструментом контролю за дотриманням правил паркування стане новостворений відділ інспекції з паркування, який планують заснувати при управлінні транспорту та зв’язку. Інспектори отримають повноваження штрафувати водіїв, які залишають свої транспортні засоби у заборонених місцях[3].

Однак, як зазначив заступник міського голови, створення інспекції не зможе повністю вирішити проблему хаотичного паркування, але це стане важливим кроком для впровадження комплексних заходів. Серед таких заходів планується будівництво нових паркінгів, включно з багаторівневими, а також створення платних парковок у центральній частині міста. Це допоможе зменшити кількість автомобілів на вулицях, що дасть можливість іншим водіям скористатися паркувальними місцями для короткотривалого паркування.

Було проведено електроне опитуванні щодо проблем паркування у їхньому місті. В опитувані взяли участ 2266 людей. Основні результати опитування показали, що проблеми з паркуванням відчувають багато водіїв. Більшість підтримує ініціативу створення інспекції з паркування.

Більше 73% респондентів підтримують ідею платного паркування в центрі міста, а 56% — біля ринків. Загалом було отримано майже 700 пропозицій від

містян. Після ретельного аналізу всі ці побажання будуть враховані при розробці стратегії розв'язання проблем паркування.

Міста завжди були ключовими економічними й адміністративними центрами, і нині вони відіграють важливу роль у глобальних процесах. Подібно до міст-держав у давнину, сучасні мегаполіси знову стають самодостатніми політичними та економічними гравцями на світовій арені, що свідчить про циклічний характер історичних процесів.

Найбільші міста світу за чисельністю населення вже перевищують, а іноді й у кілька разів, населення цілих країн, і стають центрами масштабних змін. Стрімка урбанізація та розширення міст кардинально змінюють підходи до їх планування й розвитку інфраструктури. Наприклад, за даними Міжнародної організації з міграції (МОМ), щотижня до міст мігрує близько 3 мільйонів людей, що еквівалентно появі нового Мадрида або Буенос-Айреса.

Розвиток будівельних технологій дав змогу здійснювати вертикальне розширення міст, що дозволяє ефективніше використовувати простір. Транспортна мережа стала набагато щільнішою, інформаційне навантаження зросло, а доступна житлова площа на одного мешканця зменшилася. Водночас рівень і характер потреб міських жителів значно змінився, ставши більш комплексним та різноманітним.

У результаті, міста й агломерації стикаються з новими викликами, на які потрібно реагувати. Жителі мають нові очікування, зокрема щодо послуг, безпеки й комфорту. Мегаполіси адаптуються, надаючи додаткові послуги, щоб полегшити життя своїм мешканцям, роблячи його зручнішим і безпечнішим.

Міста та агломерації стикаються з новими викликами, оскільки мешканці мають змінені очікування та потреби. Мегаполіси прагнуть надавати додаткові послуги своїм громадянам, щоб полегшити, забезпечити безпеку та підвищити комфорт їхнього життя.

Це поєднання факторів вимагає пошуку нових рішень і нових підходів у відносинах між містом і його населенням. Міста змушені змінюватися, переходячи від концепції безособової «території для виживання та задоволення базових потреб» до формування самоідентифікації як «живої істоти», де мешканці

визнаються суб’єктами, а не об’єктами урбаністичних процесів. Це, в свою чергу, веде до інтерактивної комунікації між містом і його мешканцями, які виступають як колективний розум. Іншими словами, відбувається перетворення на «розумне» місто, або Smart City.

Ця трансформація, з появою нових технологічних рішень, часто відбувається органічно, на локальному рівні, внаслідок випадкового поєднання фінансових можливостей міста або його окремих адміністративних одиниць з бажаннями та потребами різних зацікавлених сторін у конкретному адміністративна одиниця в певний момент часу [1].

Сила трансформації безперечно є значною, проте вона також несе в собі суттєві ризики, зокрема ризики дисгармонії та невідповідності між різними технологічними рішеннями на рівні міста. Це стосується безпекових загроз, проблем у логістичному управлінні міськими потоками, а також дисбалансу між комфортом мешканців і виконанням критично важливих міських сервісів. Для зменшення цих ризиків процес перетворення на «розумне» місто повинен здійснюватися відповідно до середньо- та довгострокового плану, узгодженого всіма зацікавленими сторонами, а реалізація повинна базуватися на єдиній технологічній платформі. Розгляньмо детальніше концепцію розумних міст та наявні технологічні рішення. «Розумне» місто – це таке місто, яке «слухає» та адаптується до потреб своїх мешканців. Концепція Smart City, що забезпечує мешканцям вищий рівень комфорту, безпеки та екологічності в порівнянні з сучасними мегаполісами, вже не є лише мрією фантастів. Це реальні ідеї, елементи яких втілюються в Нью-Йорку, Сінгапурі, Токіо, Берліні, Чикаго та Києві. Південна Корея розпочала будівництво першого розумного міста, спроектованого з урахуванням цих принципів.

Основний напрямок розвитку – «інтелектуальні» парктроніки. Ці датчики інтегруються в дорожнє покриття на паркувальних місцях і здійснюють моніторинг зайнятості або вільності місць над ними, передаючи інформацію до централізованої системи. Завдяки мережі таких датчиків формується карта паркувальних місць, статус якої доступний користувачам на вулицях через спеціалізовані екрани або мобільний додаток. Існує безліч можливостей для розширення функціоналу

інтелектуальних датчиків паркування. Однією з нових концепцій закритих паркінгів є навісні датчики, які також виконують функції охорони та спостереження за автомобілем. Ще одним напрямком у сфері розумного паркування є розробка та реалізація автоматизованих паркінгів (найчастіше багаторівневих), де дії водіїв зведені до мінімуму. Водій заходить у спеціальну зону або на платформу і залишає автомобіль. Далі платформа самостійно переміщує автомобіль у спеціально відведене або вільне місце та інформує водія про його номер. Щоб отримати автомобіль, водієві потрібно авторизуватися та ввести цей номер на спеціальному екрані або панелі управління, після чого платформа також...Міста завжди слугували платформами для реалізації нових можливостей. Останні оцінки світової економіки свідчать, що 80% світового ВВП формується в містах. Сучасні мегатренди швидкої урбанізації, на фоні глобальних змін клімату та виснаження ресурсів, набувають особливої актуальності для сучасних міст [4].

Сьогодні міста починають вирішувати нові категорії завдань, стаючи майданчиками для великих інновацій та експериментальними полями для впровадження новітніх технологій, відкриваючи нові можливості для тестування та реалізації винаходів.

Міста пропонують широкий спектр можливостей для постачальників і споживачів інтелектуальних технологій, які можуть сприяти оптимізації споживання ресурсів і покращенню якості послуг завдяки більш ефективному управлінню попитом і пропозицією.

«Розумне місто» – це місто, в якому для підвищення якості життя міської спільноти активно застосовуються сучасні інформаційні технології. Інформаційні технології «Розумного міста» інтегруються у виробничі процеси міських структур з метою покращення надання послуг та зменшення витрат на споживані ресурси.

У країнах Європейського Союзу системна концепція «Розумного міста» охоплює такі ключові напрямки: мобільність інфраструктури, управління навколишнім середовищем, управління міським населенням, міським господарством, а також розвиток і підвищення економічної ефективності. У більшості існуючих проектів «Розумне місто» ще не визначено повний перелік та деталі застосування інформаційних технологій. Вони здебільшого фокусуються на

питаннях енергоефективності, оптимізації міського транспортного трафіку та використанні геоінформаційних технологій. Це пояснюється необхідністю вирішення актуальних проблем функціонування міста як складної багатокомпонентної системи. Інформаційні технології повинні виступати в ролі інтеграторів між різними інфраструктурними рівнями та підсистемами, забезпечуючи надійний та ефективний обмін інформацією між ними. Подібні виклики стоять перед багатьма українськими містами, що прагнуть реалізувати перспективні проекти. Для ефективного вирішення питань формування IT-класів та впровадження сучасних інформаційних технологій у проектах «Розумне місто», які враховують як загальнонаціональні, так і місцеві інтереси, необхідно застосовувати комплексний системний підхід, який забезпечить раціональне використання людських, технологічних, природних, матеріальних і фінансових ресурсів.

Дослідження складних інформаційних технологічних комплексів, таких як «Розумне місто», стикається з низкою об'єктивних труднощів, зокрема через такі фактори:

1. Складність класів інформаційних технологій — Технології, що використовуються в сучасних містах, належать до класу складних систем. Вони мають велику кількість як кількісних, так і якісних параметрів, що ускладнює їх моделювання та оптимізацію.

2. Різноманітність та динамічність зв'язків між технологіями — Між різними інформаційними технологіями існують зв'язки різного типу, що відрізняються за природою і інтенсивністю. З часом ці зв'язки змінюються, що створює труднощі в управлінні та інтеграції таких технологій у єдину систему.

3. Нестабільність та хаотичність системи управління — Системи управління в сучасних містах часто мають слабку організаційну структуру, є нестійкими та хаотичними. Це обмежує їх ефективність і здатність до прогнозування. Явища та параметри, які характеризують ці системи, часто не підпорядковуються сталим ймовірнісним розподілам, що ускладнює їх аналіз і подальше управління.

Динамічність параметрів складних систем — Сучасне місто, як складна система, визначається параметрами, що динамічно змінюються. Це відображає рівень знань про таку систему, коли неможливо однозначно визначити її морфологічні та функціональні особливості. Технології, що використовуються для управління містом, повинні враховувати ці зміни та бути гнучкими для адаптації до нових умов.

Таким чином, стан комплексної адміністративно-господарської та інформаційно-технологічної системи «Розумного міста» визначається її функціональним наповненням, характеристиками внутрішніх процесів, а також процедурами взаємодії з зовнішнім середовищем. Взаємодія з державними, муніципальними та громадськими чинниками також є ключовим аспектом у створенні ефективної і стабільної системи управління містом.

Можна виділити базові напрямки використання інформаційних технологій в проектах «Розумне місто»:

– використання великих масивів даних та аналітичного їх опрацювання з метою підвищення якості інформаційних послуг;

– мобільне збирання даних та оперативне реагування на проблемні ситуації;

– віртуальна взаємодія між різнорідними установами, структурами та мешканцями міста;

– активне використання соціокомунікаційних технологій, що покращують взаємодію місцевих органів влади з громадянами.

Для підготовки та прийняття управлінських рішень у рамках проектів «Розумного міста», інформаційні технології повинні застосовувати інтелектуальні методи, які допоможуть вирішити широкий спектр завдань. Вони повинні бути засновані на гіпервеликих базах даних, що дозволяють обробляти величезні обсяги інформації для отримання точних і своєчасних результатів. Одним із ключових напрямків у розв’язанні проблем інтеграції різноманітних інформаційних технологій є використання програмно-алгоритмічних застосунків, які реалізують:

Будівництво онтологій — це процес створення структурованих моделей знань, які дозволяють організувати і класифікувати різноманітні дані, що

використовуються в рамках «Розумного міста». Онтології визначають концепти і зв’язки між ними, що дає змогу зручніше та ефективніше працювати з великими масивами інформації.

Інтерпретація методів пошуку інформації — застосування ефективних алгоритмів і методів для швидкого та точного знаходження необхідної інформації серед величезної кількості даних. Це включає використання алгоритмів машинного навчання для покращення точності пошуку.

Системи формування профільних рекомендацій — ці системи на основі зібраних даних можуть автоматично генерувати персоналізовані рекомендації для користувачів або управлінців, що дозволяє приймати більш обґрунтовані і ефективні рішення в умовах постійно змінюваного міського середовища [7].

Адаптивні процедури вибору моделей і критеріїв оцінки управлінських рішень — ці процедури дозволяють системі навчатися і адаптуватися до нових умов та змін у поведінці міського середовища. Це дає змогу приймати рішення, які оптимізують використання ресурсів, зменшують витрати та покращують якість життя мешканців.

Таке використання передових технологій дозволяє створювати повноцінний інтегрований простір даних та знань, що є основою для ефективного функціонування «Розумного міста». В результаті, це забезпечує підтримку управлінських рішень на всіх рівнях — від планування до реального часу — і сприяє сталому розвитку міських територій.

1.2 Огляд існуючих методів вирішення технічної проблеми

Використання локальних систем позиціонування для різних об'єктів стало актуальною темою в багатьох сферах, оскільки знання місцезнаходження в реальному часі є важливим у різних ситуаціях. Для точного визначення положення об'єкта необхідно, щоб інтервал між вимірюваннями був достатньо малим, аби об'єкт, рухаючись з певною швидкістю, не виходив за межі точності позиціонування. Наприклад, для забезпечення точності до 1 метра при моніторингу

людини, що рухається зі швидкістю 5,4 км/год (1,5 м/с), вимірювання мають проводитися приблизно кожні 1,3 секунди.

Муніципальні системи моніторингу трафіку можуть бути інтегровані в інфраструктуру «розумного міста» для відстеження заторів і ситуацій на дорогах. Хоча відеокамери для моніторингу руху вже активно використовуються в багатьох містах, системи з низьким споживанням енергії можуть забезпечити більш стабільний і щільний потік даних. Такі системи можуть включати сенсори, GPS на транспортних засобах, датчики якості повітря та акустичні датчики вздовж доріг. Ці дані мають величезне значення для муніципальної влади та громадян, дозволяючи краще контролювати трафік, направляти дорожніх офіцерів у проблемні зони та допомагати водіям у плануванні маршрутів[5].

Муніципальна система моніторингу енергоспоживання має значний потенціал для оптимізації використання енергоресурсів в межах міста та для досягнення цілей, визначених у рамках європейської політики енергоефективності.

Основні компоненти та функції такої системи:

Моніторинг енергоспоживання: Система здійснює збір та аналіз даних про енергоспоживання в реальному часі для різних сегментів міської інфраструктури, таких як вуличне освітлення, транспорт, світлофори, відеокамери спостереження та опалення/охолодження будівель. Це дозволяє визначити точні обсяги енергії, що споживаються кожним компонентом, а також виявити можливості для оптимізації.

Інтеграція з енергосистемою міста: Для ефективного моніторингу й управління енергоспоживанням важливо, щоб система була інтегрована з енергосистемою міста, що дозволяє зібрати інформацію про споживану електричну потужність в різних районах і для різних потреб. Це може включати зв'язок з розподільчими мережами для аналізу пікових навантажень та оптимізації енергетичних ресурсів.

Аналіз джерел енергоспоживання: Система дозволяє ідентифікувати основні джерела енергоспоживання, визначити пріоритети для їх оптимізації та зниження витрат. Наприклад, система може виявити, що певні частини міста використовують занадто багато енергії для освітлення, і на основі цього можна запровадити

енергоефективні рішення (використання LED-освітлення, датчиків руху для вуличного освітлення тощо).

Інтеграція з альтернативними джерелами енергії: Для більшої ефективності система може включати інтеграцію з місцевими енергетичними виробничими структурами, такими як фотоелектричні панелі. Це дозволить не тільки оптимізувати споживання, але й сприятиме використанню відновлювальних джерел енергії для покриття потреб міської інфраструктури.

Підвищення енергоефективності: Завдяки збору даних і їх аналізу, органи місцевої влади можуть краще керувати енергетичними ресурсами, коригувати політики щодо енергозбереження та впроваджувати заходи для підвищення енергоефективності в міських будівлях і транспорті.

Спільне використання даних з моніторингу якості повітря: Зазначено, що система може поєднувати моніторинг енергоспоживання з моніторингом якості повітря. Це допомагає зрозуміти взаємозв'язок між високим споживанням енергії та забрудненням навколишнього середовища. Такий підхід дозволяє планувати міське управління з урахуванням екологічних аспектів, сприяючи сталому розвитку.

Вигоди та перспективи:

Оптимізація витрат: Система дозволяє суттєво знизити енергетичні витрати на міську інфраструктуру через точний моніторинг і впровадження енергоефективних технологій.

Сталий розвиток: Використання альтернативних джерел енергії та зменшення енергоспоживання сприяє сталому розвитку та зменшенню екологічного впливу на навколишнє середовище.

Покращення якості життя: Поліпшення енергетичної ефективності не тільки економічно вигідне, а й підвищує якість життя мешканців міста завдяки стабільному постачанню енергії та покращенню екологічної ситуації.

Реалізація:

Для реалізації цієї системи необхідно розробити технологічну інфраструктуру, включаючи пристрої моніторингу енергоспоживання, розширену

мережу сенсорів для збору даних про споживану потужність та підключення до відповідних інформаційних систем міста для подальшого аналізу та управління.

Сервіс «Розумні парковки» включає дорожні датчики та інтелектуальні дисплеї, які допомагають водіям, надаючи інформацію про найкоротший шлях до найближчих паркувальних місць у потрібній локації. Переваги цієї послуги є різноманітними: вона скорочує час, витрачений на пошук паркування, зменшує викиди CO_2 від автомобілів, а також знижує затори і кількість дорожніх аварій.

Служба «інтелектуального паркування» може бути інтегрована в міську інфраструктуру, оскільки численні компанії в Європі пропонують рішення для її впровадження. Додатково, завдяки технологіям короткочасної комунікації, таким як радіочастотні ідентифікатори (RFID) або безконтактна комунікація (NFC), можливо реалізувати електронну систему контролю доступу для зареєстрованих мешканців або осіб з обмеженими можливостями. Це забезпечить кращий сервіс для громадян, які мають право на використання спеціально відведених місць, а також стане ефективним інструментом для швидкого виявлення порушень правил паркування[6].

«Розумне муніципальне освітлення є ключовим компонентом для реалізації директиви 20-20-20, яка спрямована на підвищення енергоефективності. Ця система забезпечує можливість регулювання яскравості вуличного освітлення в залежності від часу доби, погодних умов та наявності людей. Для ефективної роботи такого освітлення важливо інтегрувати вуличні ліхтарі в інфраструктуру «розумного міста».

Крім того, використання більшої кількості підключених точок забезпечить доступ до Wi-Fi по всьому місту. Система виявлення несправностей може бути впроваджена на базі контролерів вуличного освітлення, що забезпечить оперативну реакцію на виникаючі проблеми.

Контроль зазначених параметрів підвищує комфорт осіб, які перебувають у цих середовищах, що, в свою чергу, може позитивно вплинути на їхню продуктивність. Одночасно це дозволяє зменшити витрати на опалення та кондиціонування, що також є важливим аспектом енергоефективності [6].

Платформа «Citrix XenMobile» надає комплексні рішення для проектів у сфері «розумного міста», зокрема управління мобільними пристроями, застосунками та контентом. Вона забезпечує високий рівень безпеки для мобільних пристроїв та даних, а також підключення до безпечного мережевого шлюзу для доступу до муніципальних сервісів. Централізоване управління службами здійснюється через один застосунок, що полегшує мобільність у рамках «розумного міста».

Платформа містить власне сховище мобільних застосунків і пропонує пакет «Citrix MDX Toolkit», який спрощує розробку мобільних застосунків, надаючи готові функції для інтеграції з муніципальною системою. Це скорочує час на створення міських мобільних застосунків, дозволяючи розробникам фокусуватися на специфічних задачах.

Технології «MobileIron» також відіграють важливу роль у забезпеченні інтеграції мобільних платформ з муніципальними ресурсами. Вони забезпечують захист даних, що передаються на муніципальні пристрої, можливість відстеження мобільних пристроїв та зменшення ризиків втрати конфіденційної інформації. Безпека даних гарантується за допомогою інтелектуального шлюзу.

«MobileIron» також підтримує управління мобільністю не лише для власних ресурсів, але і через хмарні технології, що робить її універсальним інструментом для розробників муніципальних застосунків, які прагнуть інтегрувати свої рішення у сучасну інфраструктуру «розумного міста».

Платформа «MobileIron» надає пакет «Mobile AppConnectSDK», який створений для упаковки мобільних додатків у захищені контейнери.. Цей пакет забезпечує захист конфіденційної інформації, безпечну передачу даних і можливість оновлення застосунків, що робить його важливим інструментом для муніципальних служб, які прагнуть зберегти дані у безпеці[7].

Платформа «Symantec Mobility Suite» пропонує комплекс програмних рішень для бізнесу, включаючи управління мобільними пристроями, додатками та контентом. Вона забезпечує захист від кіберзагроз і дозволяє встановлювати політики безпеки та доступу до мобільних пристроїв і муніципальних даних. Крім

того, Symantec надає власний SDK для створення безпечних контейнерів мобільних додатків та їх упаковки, що підвищує рівень безпеки.

Платформа «SOTI» надає можливість віддаленого управління мобільними пристроями та забезпечує всебічну конфігурацію функціонування застосунків у поєднанні з налаштуванням правил використання пристроїв. Вона підтримує багатофакторну авторизацію для інформаційних систем «розумного міста». Платформа також пропонує потужний SDK у вигляді «Android for Work» для розробки мобільних застосунків. Проте, одним із недоліків SOTI є відсутність можливості упаковки застосунків у безпечну оболонку, що може обмежувати використання її рішень у критично важливих додатках.

Таким чином, усі ці платформи пропонують різні рішення для забезпечення безпеки, управління і розробки мобільних застосунків, що є важливими аспектами для функціонування «розумного міста».

Розробникам надається пакет для створення як нативних, так і гібридних мобільних застосунків. Платформа підтримує нативні застосунки для «Android» та «iOS», а для розробки гібридних мобільних застосунків використовується фреймворк «Apache Cordova», що дозволяє швидко створювати кросплатформенні рішення.

Платформа «AirWatch» забезпечує ефективне управління мобільними середовищами, зручний доступ до муніципальних застосунків та захист інформації. Вона пропонує різноманітні сценарії взаємодії з мобільними пристроями в рамках єдиного рішення, яке охоплює весь життєвий цикл програми — від етапу розробки до стадії використання. Крім того, платформа надає власний SDK для створення мобільних застосунків, що працюють на її основі. Готові функції, які постачаються разом із SDK, включають управління мобільними пристроями, застосунками та даними.

Таким чином, обидві платформи, «MaaS 360 FiberLink» та «AirWatch», пропонують розширені можливості для забезпечення безпеки, управління і розвитку мобільних рішень у муніципальному контексті, дозволяючи ефективно інтегрувати мобільні технології в інфраструктуру «розумного міста».

Платформа BlackBerry Enterprise Mobility Suite забезпечує комплексне управління мобільними пристроями, застосунками та контентом. Вона включає механізми авторизації користувачів та інструменти для розробки безпечних контейнерів для мобільних застосунків, що робить її ідеальним рішенням для муніципальних потреб у сфері мобільності.

Платформа Good має стандартні інструменти для управління муніципальною мобільністю, зокрема можливість відокремлення особистих даних і застосунків від муніципальних на мобільних пристроях. Це досягається за рахунок концепції «BYOD» (Bring Your Own Device) з використанням спеціальних контейнерів для обгортки мобільних застосунків, що дозволяє співробітникам використовувати власні пристрої для роботи, забезпечуючи при цьому безпеку даних.

Програмно-алгоритмічна платформа Enterprise Mobility Manager складається з кількох ключових модулів:

- Модуль управління мобільними пристроями: дозволяє конфігурувати мобільні пристрої, призначати права доступу до функцій та ролей для кожного користувача у системі «розумного міста», а також моніторити дії користувача.
- Модуль управління мобільними застосунками: забезпечує зберігання та публікацію застосунків для муніципальних пристроїв. Завантаження і установка програмного забезпечення можуть здійснюватися віддалено, що спрощує управління і оновлення програм.
- Модулі для мобільних платформ: надають доступ до функцій пристроїв, що дозволяє інтегрувати різні функції в муніципальні застосунки.
- Консоль EMM: надає користувачам веб-інтерфейс для роботи з платформою. Зареєстровані користувачі мають набір прав для налаштування конфігурацій залежно від призначеної їм ролі (адміністратор, звичайний користувач тощо).

Ці компоненти разом формують потужну інфраструктуру для управління мобільними технологіями в контексті «розумного міста», що сприяє підвищенню ефективності та безпеки муніципальних служб.

Впровадження RTLS (системи для визначення місцезнаходження в реальному часі) у системи розумного паркування обумовлене специфікою завдань та цілей [8].

Смарт Паркінг - це безкоштовний додаток, однак послуги паркування є платними. Додаток «Смарт Паркінг» можна завантажити на платформи Android та iOS. Він пропонує широкий спектр функцій, які допомагають водіям знаходити вільні місця для паркування, отримувати інформацію про умови та тарифи на автостоянках, а також здійснювати безконтактні платежі. (рис. 1.2).

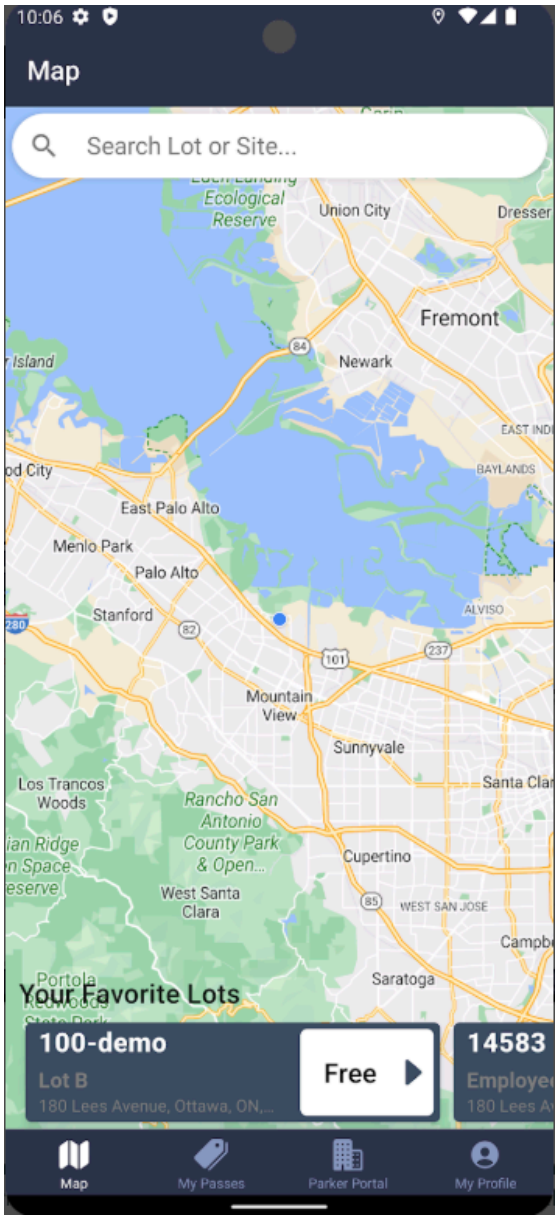


Рисунок 1.2 –Smart Parking

4Park Smart Parking App - доступний для платформ Android та iOS, що забезпечує інтеграцію з усіма системами Smart Parking. Додаток дозволяє виконувати різноманітні операції, такі як оплата та бронювання паркувальних місць, отримання спеціальних дозволів (для ділових поїздок, туристичних

автобусів, осіб з інвалідністю), паркування автомобіля, доступ до безкоштовних стоянок, віртуалізація карток та оплата за паркування. (рис. 1.3).

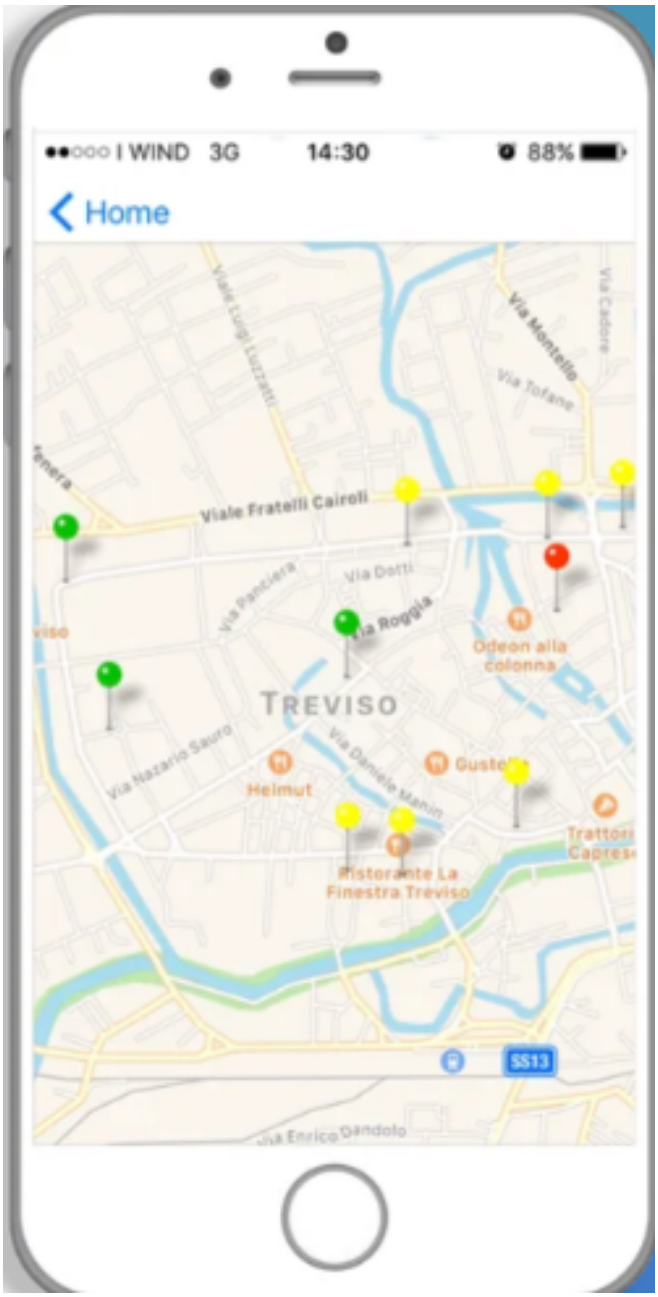


Рисунок 1.3 –Логотип 4Park Smart Parking App

GCC Smart Parking – це система, що включає в себе ряд функцій автоматизованого паркування, а також можливість живого чату для вирішення запитів, пов'язаних із платежами, історією бронювань та політикою скасування. Користувачі можуть здійснювати оплату за бронювання через свої мобільні пристрої, використовуючи інтернет-банкінг. У разі скасування бронювання, кошти

автоматично повертаються на банківський рахунок користувача. Додаток доступний для платформ Android та iOS. (рис. 1.4).

Рисунок 1.4 – Логотип GCC Smart Parking

Найбільша категорія, що складається з кількох підкатегорій, охоплює радіолокаційні технології. Ці технології, в свою чергу, поділяються на стандартні методи передачі даних (Wi-Fi, Bluetooth, ZigBee), які так чи інакше призначені для вимірювання відстаней, а також на ті, що, враховуючи фізичні властивості модуляції, є найбільш підходящими для визначення відстані (UWB, NFER та інші). [9].

Технологія UWB (Ultra-Wideband) забезпечує точне позиціонування завдяки високочастотним сигналам і використанню імпульсної радіотехніки. Однак є кілька фундаментальних обмежень та переваг, які визначають її застосування, особливо в порівнянні з іншими бездротовими технологіями.

Фундаментальні обмеження UWB:

Малопотужний сигнал:

Типова потужність передавача UWB становить 50 мкВт, що робить передачу на великі відстані складною. Найбільш потужний передавач може мати потужність до 10 мВт. Це обмеження зумовлено правилами частотних регуляторів, щоб запобігти завадам в інших системах.

Частотні обмеження:

Технологія працює в неліцензованих частотних діапазонах і зазвичай використовуються для застосувань у закритих приміщеннях, де її слабкий шумоподібний сигнал не створює перешкод для інших систем зв'язку [10].

Переваги UWB:

- Високий рівень завадостійкості:
- Завдяки розширеному спектру частот і низькій потужності, сигнал UWB стійкий до перешкод, що робить технологію надійною в складних умовах.
- Висока безпека:
- Завдяки коротким імпульсам і широкому спектру частот, UWB важко прослуховувати, що підвищує безпеку переданих даних.
- Висока точність позиціонування:
- Технологія забезпечує дуже точне визначення відстані завдяки використанню методів, таких як ToF (Time of Flight), ToA (Time of Arrival) та AoA (Angle of Arrival).

Недоліки UWB:

- Малий радіус дії:
- Радіус дії технології обмежений до 10 м, що обумовлено малопотужними сигналами.
- Складна інфраструктура:

- Для коректної роботи необхідна спеціалізована інфраструктура, яка включає безліч приймачів і передавачів для точного позиціонування.

Методи позиціонування за допомогою UWB:

AoA (Angle of Arrival):

Визначення кута прибуття сигналу на кілька приймачів для визначення положення передавача. Цей метод потребує наявності кількох антен.

ToF (Time of Flight):

Вимірювання часу, необхідного сигналу для проходження від передавача до приймача. Це дозволяє точно визначити відстань між пристроями.

ToA (Time of Arrival):

Визначення часу прибуття сигналу до приймача, що дозволяє обчислити відстань, враховуючи відому швидкість поширення сигналу.

TDoA (Time Difference of Arrival):

Визначення різниці часу, коли сигнал досягає кількох приймачів. Це дозволяє точно визначити координати об'єкта, використовуючи множину точок прийому.

Застосування UWB:

Інтер'єрне позиціонування: Завдяки своїй точності, технологія UWB активно використовується для точного позиціонування в приміщеннях, таких як склади, магазини чи навіть для стеження за активами.

Розумні системи безпеки та моніторинг: Відстеження місцезнаходження людей і об'єктів з високою точністю, що робить UWB ідеальним для таких застосувань, як доступ до приміщень або моніторинг здоров'я.

Wi-Fi для локалізації:

Wi-Fi, хоча й не призначений для високоточної локалізації, може бути використаний для оцінки місцеположення через метод RSSI (Received Signal Strength Indicator) або спеціалізовані модифікації, такі як TDoA. Wi-Fi має більший радіус дії, ніж UWB (десятки метрів), але його точність позиціонування зазвичай нижча, оскільки вона обмежується тільки інформацією, доступною з точки доступу.

Wi-Fi може використовуватись для попередньої оцінки місця розташування, а комбінація з іншими методами, як TDoA, дозволяє покращити точність

визначення позиції у випадках, коли традиційні методи не дають достатньо точних результатів.[8].

Переваги:

– широке застосування;

– низька вартість обладнання. Недоліки: для підвищення точності потрібне збільшення щільності базових станцій;

– перевантаження ефірного Wi-Fi;

– недостатня точність позиціонування для ряду завдань,

– навіть при використанні спеціальних подовжувачів Wi-Fi (в ідеальних умовах 3-5 метрів, реально 10-15 метрів).

Методи визначення позиції: RSSI на основі TDoA.

ZigBee — це технологія для створення бездротових персональних мереж, розроблена на основі стандарту IEEE 802.15.4, що забезпечує ефективну та енергоощадну комунікацію для малопотужних пристроїв. Вона особливо підходить для застосувань, де потрібна передача даних на низьких швидкостях з можливістю роботи від автономних джерел живлення, таких як батареї. Ось ключові аспекти цієї технології:

Ключові характеристики ZigBee:

Малопотужність і енергоефективність:

ZigBee оптимізований для пристроїв, які працюють від батарей і повинні мати тривалий час роботи. Це робить його ідеальним для застосувань у розумних будинках, промислових системах та інших IoT рішеннях, де важлива низька енергоспоживаність.

Швидкість передачі даних:

Швидкість передачі даних у мережі ZigBee становить 250 Кбіт/с, що є достатнім для більшості застосувань, де потрібно передавати не надто великі об'єми даних, наприклад, сенсорні показники або контрольні сигнали.

Середня пропускна здатність для корисних даних коливається від 5 до 40 кбіт/с, залежно від кількості повторних передач та завантаженості мережі.

Частотні діапазони:

ZigBee працює в кількох частотних діапазонах: 868 МГц (Європа), 915 МГц (Північна Америка) та 2,4 ГГц (міжнародний). Діапазон 2,4 ГГц є найбільш поширеним завдяки високій швидкості передачі і меншій чутливості до перешкод, хоча і може бути підданий інтерференції з іншими бездротовими технологіями (Wi-Fi, Bluetooth).

Зона покриття:

Відстань між вузлами в мережі ZigBee залежить від середовища: в приміщенні вона становить десятки метрів, а на відкритому повітрі може досягати сотень метрів. За допомогою ретрансляції даних можна значно збільшити зону покриття.

Самоорганізуюча мережа:

Одна з основних переваг ZigBee — автоматичне самоорганізування та самовідтворення мережі. Це дозволяє вузлам мережі автоматично з'єднуватися і відновлювати з'єднання в разі відмови, що підвищує надійність та стійкість системи.

Методи визначення позиції в ZigBee:

RSSI (Received Signal Strength Indicator):

Один із способів визначення позиції об'єкта в мережі ZigBee — це використання RSSI. За допомогою цього методу можна оцінити відстань між вузлами мережі на основі сили отриманого сигналу. Однак цей метод має обмеження через вплив на сигнал перешкод і відбиття.

TDoA (Time Difference of Arrival):

Інший метод — це TDoA, при якому вимірюється різниця в часі, за який сигнал доходить до кількох приймачів. Це дозволяє точно визначити місце розташування об'єкта, який генерує сигнал, використовуючи декілька точок прийому.

ToF (Time of Flight):

ToF вимірює час, необхідний сигналу для того, щоб пройти від передавача до приймача. Цей метод може бути дуже точним, однак він потребує точно налаштованих часових міток.

TWR (Two Way Ranging):

TWR — метод, який вимірює час, необхідний сигналу для подорожі в обидва напрямки (від передавача до приймача і назад). Це дозволяє визначити точну відстань між двома пристроями, зменшуючи вплив зовнішніх факторів на точність.

Переваги та застосування ZigBee:

Розумне паркування: Використовувати ZigBee для відстеження автомобілів та управління паркувальними місцями [12].

Домашня автоматизація: Застосовується для контролю та моніторингу різноманітних пристроїв, таких як термостати, датчики руху, віконні сенсори тощо.

Моніторинг стану здоров'я: У сфері охорони здоров'я можна використовувати ZigBee для відстеження біометричних даних, передачі інформації між медичними пристроями і моніторингом пацієнтів.

ZigBee є одним з важливих компонентів технологій IoT завдяки своїй низькій енергоспоживаності, масштабованості та надійності, що робить його ідеальним вибором для численних застосувань у розумних мережах.[9].

Переваги [7]:

– підтримує як прості топології мережі («точка-точка», «дерево» і «зірка»), так і mesh-топологію з ретрансляцією і маршрутизацією повідомлень;

– містить можливість вибору алгоритму маршрутизації залежно від вимог програми та стану мережі;

– простота розгортання, обслуговування та модернізації;

– здатність до самоорганізації та самовідтворення;

– низьке енергоспоживання.

Недоліки:

– низька швидкість передачі даних. Методи визначення позиції: RSSI, TDoA, ToF.

NanoLOC — це технологія від Nanotron Technologies, яка забезпечує високоточне визначення місця розташування об'єктів за допомогою радіочастотного зв'язку. Вона схожа на попередню версію NanoNET, але має покращену швидкість передачі даних і точність визначення місцезнаходження.

Ключові характеристики та переваги NanoLOC:

Швидкість передачі даних: Технологія дозволяє передавати дані зі швидкістю до 1 Мбіт за секунду на відстані кілька сотень метрів. Це робить її придатною для застосувань, де важлива передача даних у реальному часі, таких як системи моніторингу чи відстеження.

Визначення відстані між приймачами: NanoLOC має можливість вимірювати відстань між приймачами з похибкою 2 метри, що дозволяє точно визначати місце розташування об'єкта або трансивера в межах кількох метрів. Для більш точного визначення в тривимірній системі координат необхідно використовувати чотири або більше передавачів з відомими координатами, що дозволяє точно визначити місцезнаходження в просторі.

Переваги технології:

Неліцензійний діапазон частот: NanoLOC працює в неліцензійних частотних діапазонах з потужністю до 100 мВт, що знижує витрати на ліцензії та дозволяє більш гнучко використовувати технологію.

Зона обслуговування: Технологія дозволяє здійснювати локалізацію об'єктів не тільки в межах основної зони обслуговування, але й за її межами, з пониженням точності.

Стійкість до шуму: Завдяки автокореляційним властивостям сигналу, NanoLOC є стійкою до зовнішнього шуму, що робить її надійною для використання в умовах з високими рівнями електромагнітних завад.

Програмне забезпечення з відкритим кодом: Технологія забезпечує великий вибір програмного забезпечення з відкритим кодом, що дає змогу користувачам модифікувати та налаштовувати систему відповідно до своїх потреб.

Застосування NanoLOC:

Завдяки своїм характеристикам, NanoLOC може бути використана в багатьох сферах, таких як:

Моніторинг і відстеження об'єктів у реальному часі.

Рішення для розумного паркування (наприклад, для визначення наявності автомобіля на паркувальному місці).

Інтер'єрне позиціювання для відстеження місцезнаходження товарів у магазинах або складах.

Логістика для відстеження переміщення вантажів.

Таким чином, NanoLOC є перспективною технологією для створення систем точного позиціювання та локалізації, де важлива висока точність і надійність роботи в реальному часі.

Недоліки:

- обмеження на кількість пристроїв у сегменті;
- власна розробка.

Методи визначення положення (RSSI, TDoA, ToF, TWR):

1. RSSI (Received Signal Strength Indicator):
   - Метод оцінки відстані між пристроями на основі сили отриманого сигналу. Чим сильніший сигнал, тим ближче пристрій.
   - Використовуваний у Bluetooth і Wi-Fi для визначення позиції в реальному часі. Однак його точність може бути знижена через шум або перешкоди в середовищі.
2. TDoA (Time Difference of Arrival):
   - Визначення положення об'єкта на основі різниці в часі, коли сигнал від нього досягає кількох приймачів.
   - За допомогою кількох приймальних станцій можна визначити, де знаходиться об'єкт, використовуючи різниці у часі приходу сигналу.
3. ToF (Time of Flight):
   - Вимірювання часу, який проходить сигнал від передавача до приймача. Знаючи швидкість сигналу, можна розрахувати відстань до об'єкта.
   - Застосовується в системах, що використовують ультразвук, лазер, радіохвилі або інші типи хвиль для вимірювання відстаней.
4. TWR (Two-Way Ranging):
   - Метод, при якому два пристрої обмінюються сигналами для визначення відстані між ними. Одне з пристроїв відправляє сигнал, а інше його приймає і відправляє назад. Вимірюючи час, необхідний для обміну, можна точно оцінити відстань між ними [13].

Система розумного паркування (Axioma Group):

Проект Axioma Group з розумного паркування передбачає використання датчиків для моніторингу паркувальних місць у реальному часі. Вся система працює наступним чином:

1. Датчики на паркувальних місцях: У кожному паркувальному місці встановлюються датчики (електромагнітні або інфрачервоні), які визначають, чи зайняте місце транспортним засобом.

2. Передача даних на базові станції: Датчики через радіоканал передають інформацію на базові станції, які накопичують дані про стан паркувальних місць.

3. Передача даних на сервер: Базові станції передають інформацію на сервер управління, де вона обробляється і систематизується.

4. Виведення інформації: Програмне забезпечення дозволяє створювати аналітичні звіти, передавати дані до навігаційних систем або на LED-дисплеї, що знаходяться в місті для інформування водіїв про наявність вільних місць.

5. Адаптованість до кліматичних умов: Все обладнання сертифіковано за стандартами IP65, що означає високу ступінь захисту від води та пилу, що дозволяє використовувати його в різних кліматичних умовах.

6. Інтеграція з іншими системами: Завдяки отриманим даним, можна інтегрувати систему з різними навігаційними додатками, що дозволяє водіям отримувати інформацію про вільні місця для паркування в реальному часі.

Послідовність дій для розумного паркування:

1. Прибуття на паркувальне місце:

- Водій під'їжджає до паркувального місця, де встановлено датчик.
- Датчик реєструє наявність автомобіля і передає інформацію на базову станцію.

2. Моніторинг місця:

- Програмне забезпечення обробляє дані та відображає їх на сервері, який визначає зайнятість паркувального місця в реальному часі.

3. Інформування водія:

- Виведення інформації на LED-дисплеї або в навігаційні системи для сповіщення водіїв про доступні паркувальні місця.

4. Оплата та резервування:

- У разі необхідності, система може ввести механізм оплати за паркування або зарезервувати місце через мобільний додаток.

Система дистанційного керування паркуванням:

1. Отримання даних від камери: Система отримує зображення або відео з камери, яка фіксує номерний знак автомобіля.

2. Додавання даних до бази даних: Якщо клієнт вже зареєстрований, система додає або оновлює інформацію про його членство та дату резервування в базі даних.

3. Підйом шлагбаума: Шлагбаум на в'їзді автоматично піднімається після того, як система верифікує дані. Це дозволяє водієві заїхати на стоянку.

4. Датчик безпеки: Шлагбаум автоматично опускається після того, як автомобіль проходить через датчик безпеки, що підтверджує, що автомобіль проїхав.

1. Зупинка автомобіля на виїзді: Коли автомобіль під'їжджає до виїзду і зупиняється перед шлагбаумом, система перевіряє наявність автомобіля через спеціальний датчик.

2. Перевірка номерного знака: Камера знову перевіряє номерний знак автомобіля і порівнює його з базою даних для перевірки наявності оплати або дозволу на виїзд.

3. Підтвердження платежу: Якщо система підтверджує правильність платежу або дозволяє виїзд без оплати (наприклад, для членів програми лояльності), шлагбаум піднімається.

4. Підйом шлагбаума на виїзді: Після підтвердження інформації про оплату або членство, шлагбаум на виїзді піднімається, дозволяючи водієві покинути стоянку.

- Піднімаємо шлагбаум на виїзді, дозволяючи водієві залишити стоянку;
- Шлагбаум автоматично закривається, щойно автомобіль перетинає датчик безпеки.;

Mikkom AS101 ProPark

Принцип функціонування системи полягає у точному визначенні розташування вільних та зайнятих паркувальних місць, а також у підрахунку

кількості автомобілів, що в'їжджають і виїжджають. Світлова індикація для кожного паркувального місця та інформаційні табло надають водіям дані про вільні місця та оптимальний маршрут до них, відповідно до рисунка 2. Система забезпечує оперативний і безперервний моніторинг завантаженості, надаючи всю необхідну інформацію персоналу, а також контролює роботу світлофорів, дисплеїв і шлагбаумів. [14]. Система створена на базі охоронного комплексу AS101 Pro і має притаманні комплексу високу надійність, зручність і простоту експлуатації.

В рамках цього проекту створено низку нових пристроїв (ультразвукові датчики присутності транспортних засобів згідно з рисунком 3, дистанційні індикатори зайнятості місць, інформаційні табло, датчики входу/виїзду, суматори транспортних засобів, що проїжджають) та програмного забезпечення.

1.3 Висновки

У цьому розділі було здійснено всебічний аналіз предметної галузі, зокрема охарактеризовано концепцію пошуку місць для паркування. Було визначено, що для ефективного виконання цієї задачі необхідно працювати з географічними даними, що дозволяє точно визначати наявність вільних паркувальних місць у реальному часі.

Огляд відомих аналогів систем паркування показав, що більшість із них успішно справляються зі своїми функціями та пропонують можливість оплати паркування, що значно спрощує процес для водіїв.

Аналіз актуальності нової системи в порівнянні з існуючими аналогами показав, що існує потреба в інтеграції більш сучасних технологій, які можуть підвищити ефективність та зручність користування. Зокрема, пропозиція впровадження функцій на базі GPS та інших технологій позиціонування може значно покращити досвід користувачів, зменшити час пошуку вільних місць для паркування і оптимізувати рух транспорту в містах.

В цілому, виявлено, що розробка нової системи має великі перспективи та відповідає актуальним вимогам суспільства, що прагне до зручності і ефективності в умовах швидко зростаючого автомобільного трафіку.

2 ФОРМУВАННЯ ПРОГРАМНИХ ВИМОГ ТА ПОБУДОВА КОНЦЕПЦІЙ ПРОДУКТУ

2.1 Вибір оптимальних інформаційних технологій

Мова програмування — це спеціально розроблена штучна мова, призначена для передачі інструкцій комп'ютерам та іншим машинам. Вона дозволяє програмістам писати програми, які виконують певні алгоритми та керують поведінкою систем.

Більш формально, мова програмування — це система позначень, яка дозволяє описувати алгоритми та структури даних. Вона визначається набором лексичних, синтаксичних та семантичних правил, що вказують, як програма повинна виглядати і які дії мають бути виконані комп'ютером під її управлінням.

З моменту появи перших програмованих машин було розроблено понад 2500 мов програмування, і цей список продовжує зростати щороку. Деякі мови використовуються лише вузьким колом розробників, тоді як інші стають широко поширеними і вивчаються мільйонами програмістів по всьому світу. Професійні програмісти зазвичай володіють кількома мовами програмування і застосовують їх залежно від завдань, з якими працюють[10].

Мови програмування класифікують за кількома критеріями, що дозволяє розділити їх за рівнем абстракції, ділянкою застосування та підтримуваними парадигмами програмування.

Рівень абстракції:

- Високого рівня: Ці мови оперують сутностями, зрозумілими людині, такими як об'єкти, змінні та функції. Вони наближені до природної мови людини, а тому код таких мов легше писати та розуміти. Програми мовами високого рівня є коротшими, але при цьому можуть займати більше місця в пам'яті після компіляції, порівняно з мовами низького рівня.
- Низького рівня: Мови низького рівня працюють на рівні, близькому до апаратури, оперуючи байтами, адресами пам'яті та інструкціями процесора. Такі мови дозволяють більш точний контроль над ресурсами комп'ютера, проте вони складніші у використанні.

Ділянка застосування:

- Універсальні: Мови, що підходять для широкого спектра завдань і галузей програмування. Вони використовуються для створення різноманітних програмних продуктів, від вебдодатків до операційних систем.
- Спеціалізовані: Призначені для вирішення конкретних завдань в певних галузях. Хоча деякі з них можуть бути Тюрінг-повними (можуть виконувати будь-який алгоритм), вони обмежені певною сферою, як, наприклад, мова shell для командної оболонки[11].

Підтримувані парадигми програмування:

- Імперативні: Основою імперативних мов є змінні та послідовні команди, які змінюють їхні значення. Програми складаються з послідовностей команд, що змінюють стан програми шляхом присвоєння значень змінним.
- Об'єктно-орієнтовані: Зосереджені на концепції "об'єктів", які є екземплярами класів і взаємодіють через методи.
- Функційні: Ґрунтуються на обчисленнях функцій і не мають змінних як таких. Вони фокусуються на незмінних даних і чистих функціях.
- Логічні: Оперують на основі логічних висновків, де програма описується як набір правил, і виконання полягає у пошуку висновків з цих правил.

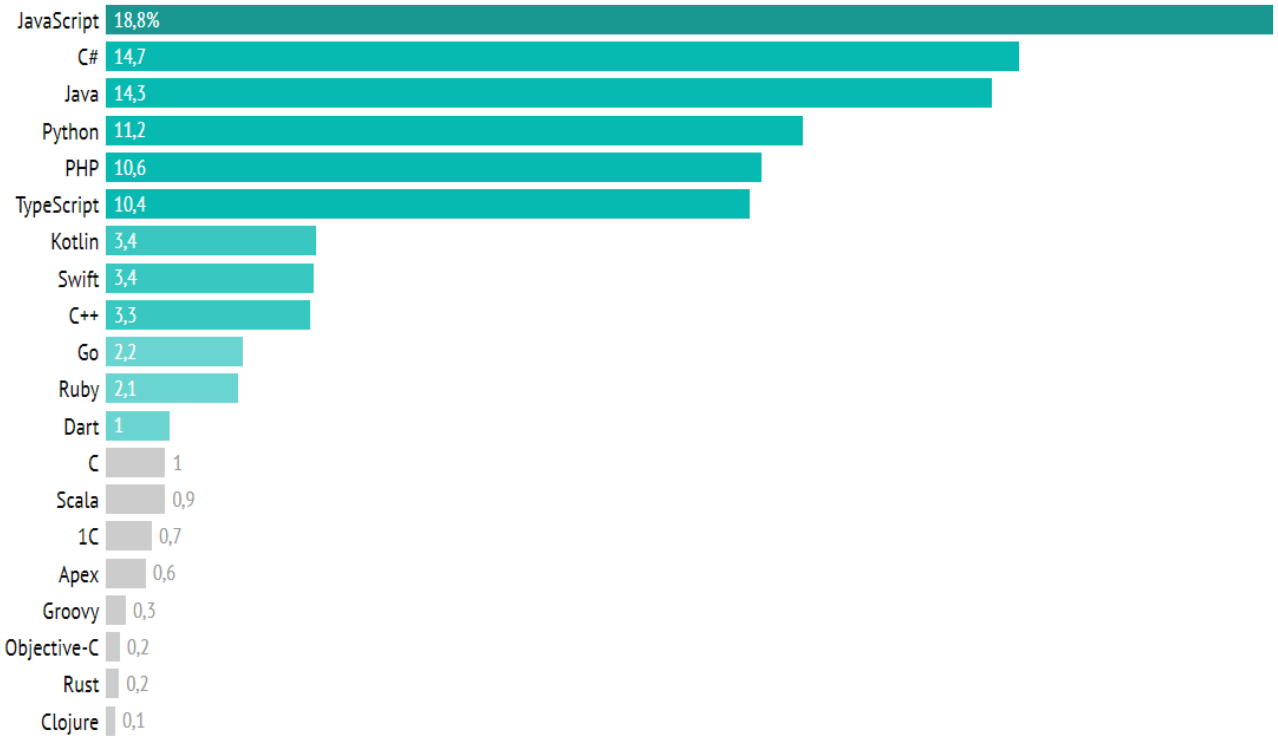


Рисунок 2.1 – Рейтинг мов програмування.

Об'єктно-орієнтоване програмування (ООП) — це парадигма програмування, яка фокусується на використанні об'єктів як основних будівельних блоків програми. Об'єкти є сутностями, які мають властивості (дані) та методи (функції), і взаємодіють між собою для виконання завдань. ООП має чотири ключові концепції:

1. Інкапсуляція: Захист даних об'єкта від зовнішнього втручання. Це означає, що внутрішній стан об'єкта може бути змінений лише через методи цього об'єкта, а зовнішні об'єкти не можуть безпосередньо змінювати його дані.

2. Успадкування: Механізм, що дозволяє створювати нові класи на основі існуючих, успадковуючи їх властивості та методи. Це сприяє повторному використанню коду та спрощує його підтримку.

3. Поліморфізм: Можливість використання одних і тих самих імен методів для різних типів об'єктів. Поліморфізм дозволяє викликати методи одного імені на різних об'єктах, які виконують їх відповідно до свого класу.

4. Абстракція: Приховування деталей реалізації та забезпечення доступу до сутності через простий інтерфейс. Абстракція дозволяє сконцентруватися на тому, що робить об'єкт, а не як він це робить.

ООП з’явилось у 1960-х роках як відповідь на зростаючу складність програмного забезпечення. Попри ранні спроби використання, ця парадигма набрала популярності в 1990-х роках, коли великі та складні системи вимагали ефективнішої організації та модульності коду.

Сьогодні багато мов підтримують ООП, включаючи C++, Java, C#, Python, Ruby, і Swift. Це дозволяє програмістам створювати коди, які легко модифікувати, підтримувати та повторно використовувати, що особливо важливо при розробці великих програмних проектів[12].

ООП стало інструментом, що допоміг подолати ці труднощі завдяки модульності програм, яка дозволяє організувати код навколо окремих сутностей — об'єктів. На відміну від процедурного підходу, де програма є набором інструкцій для виконання комп'ютером, ООП дозволяє програмістам моделювати реальний

світ, представляючи програми як сукупність об'єктів, кожен з яких має свої власні методи та властивості.

Об'єкти в ООП взаємодіють один з одним шляхом передачі повідомлень (виклику методів) і обробки даних. Це дозволяє створювати більш гнучкі, підтримувані та масштабовані системи, де кожен об'єкт відповідає за виконання певного завдання. Таким чином, ООП значно полегшує модифікацію та повторне використання коду.

Для створення клієнтської частини в сучасних веб-додатках були розглянуті різні фреймворки, такі як **React Native** та **AngularJS**, з метою вибору найбільш підходящого. Було обрано **AngularJS**, оскільки він базується на **JavaScript**, що дозволяє швидко розробляти додатки та забезпечує мультиплатформенність. Основними факторами вибору стали швидкість розробки та підтримка різних платформ[13].

JavaScript — це високорівнева мова програмування, яка дозволяє створювати як фронтенд, так і бекенд рішення для веб-додатків. Вона підтримує кілька парадигм, включаючи імперативний, функціональний та подієво-орієнтований підходи. Завдяки динамічній типізації JavaScript надає розробникам гнучкість у написанні коду, що інтерпретується в браузері без необхідності додаткових інструментів або компіляції.

JavaScript є важливою частиною веб-розробки разом з HTML і CSS, які відповідають за структуру та стиль. JavaScript, в свою чергу, надає інтерактивність і можливість створювати складні веб-застосунки з реакцією на дії користувача в реальному часі.

З розвитком платформи Node.js, JavaScript вийшов за межі браузера і тепер використовується для створення серверних застосунків. Node.js дозволяє виконувати JavaScript на сервері, що робить його потужним інструментом для розробки повноцінних програмних рішень, як-от Netflix та PayPal, які використовують JavaScript як для фронтенд, так і для бекенд частини.

AngularJS — це популярний фреймворк для розробки односторінкових веб-додатків, який базується на патерні MVC (Model-View-Controller). Використовуючи цей патерн, AngularJS дозволяє структурувати код, що робить

розробку більш логічною та організованою, а також полегшує тестування. (рис. 2.2).

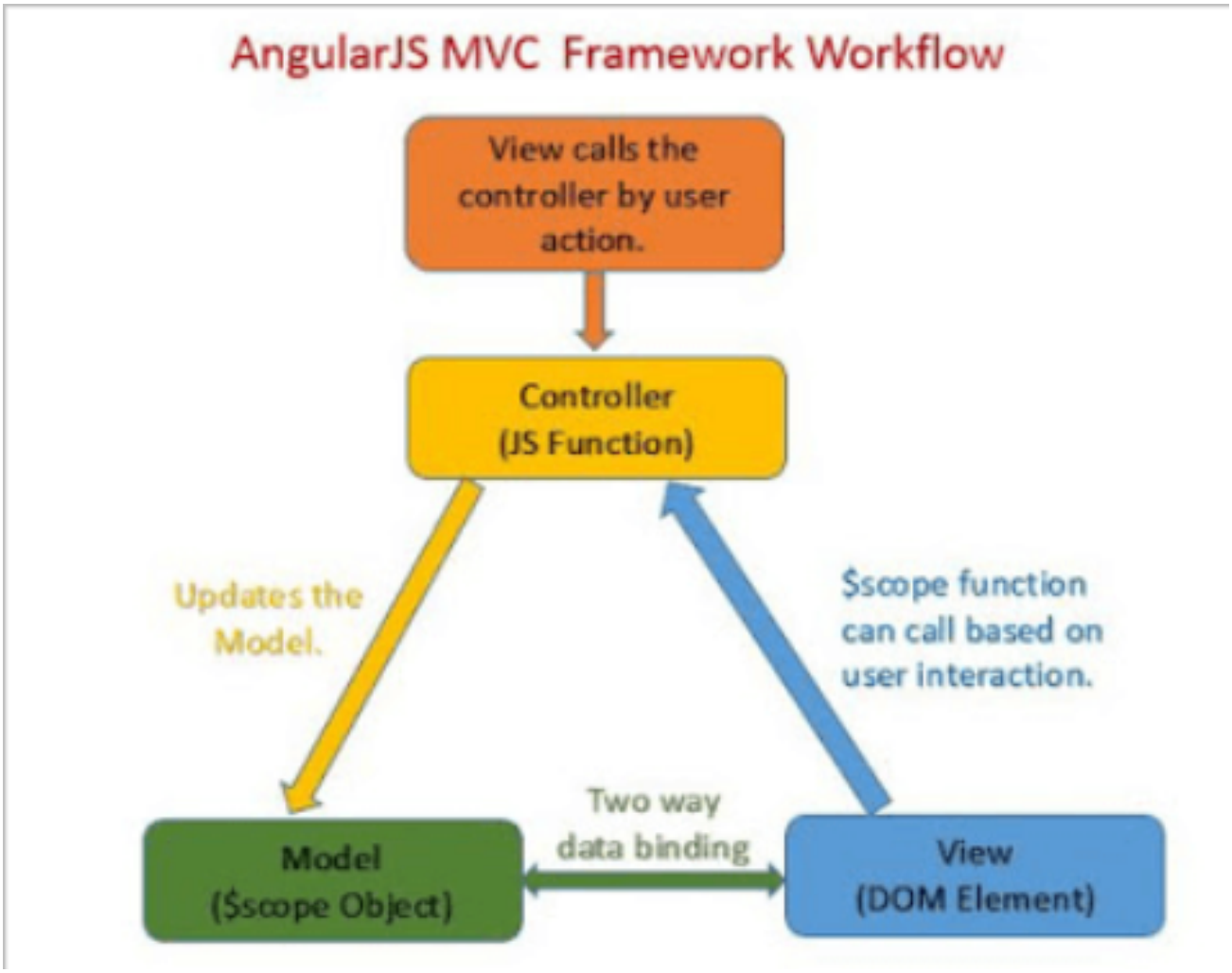


Рисунок 2.2 – Модель MVC для AngularJS

Модель MVC в AngularJS розподіляє додаток на три основні частини:

- Model (Модель): відповідає за зберігання та обробку даних.
- View (Подання): відображає дані, отримані з моделі.
- Controller (Контролер): обробляє взаємодію між моделлю та поданням, координуючи логіку[14].

Однією з переваг AngularJS є те, що для початкового освоєння азів не потрібно багато часу, і вже за кілька годин можна почати створювати прості додатки. Проте, для повного опанування всіх функцій та особливостей фреймворку може знадобитися більше часу, адже він має багатий набір інструментів та можливостей для створення складних і масштабованих застосунків [15].

Однією з головних переваг AngularJS є підтримка двостороннього зв'язування даних (*two-way data binding*), що означає, що будь-які зміни в моделі автоматично відображаються в поданні, і навпаки — зміни в інтерфейсі користувача негайно оновлюють модель. Це дозволяє зменшити кількість ручного коду для синхронізації даних між моделлю та поданням, полегшуючи та прискорюючи розробку[14].

Сильні сторони AngularJS включають:

- Підтримка Google та Microsoft: фреймворк активно підтримується технологічними гігантами, що забезпечує його постійний розвиток та стабільність.
- Двостороннє зв'язування даних: автоматичне оновлення інтерфейсу при зміні даних і навпаки, що спрощує процеси взаємодії з користувачем.
- Широке коло ком'юніті: активна спільнота розробників сприяє розвитку фреймворку, створюючи додаткові інструменти та надаючи підтримку.
- Чудова взаємодія з іншими бібліотеками: можливість інтеграції AngularJS з іншими JavaScript-бібліотеками та фреймворками.
- Відображення шаблонів як HTML-тегів: це дозволяє серверу не перевантажуватись, делегуючи більше роботи клієнту.
- Декларативне програмування: воно спрощує підтримку та читання коду завдяки чіткій структурі, де програміст описує *що* має бути зроблено, а не *як*.
- Розробка веб-додатків: AngularJS дозволяє створювати односторінкові додатки, які завантажуються, як звичайні веб-сторінки, але з додатковими можливостями та інтерактивністю.

Двостороннє зв'язування даних є важливою функцією, яка робить AngularJS привабливим для розробки інтерактивних додатків, дозволяючи негайно відображати зміни в інтерфейсі користувача без необхідності перезавантаження сторінки (рис. 2.3)

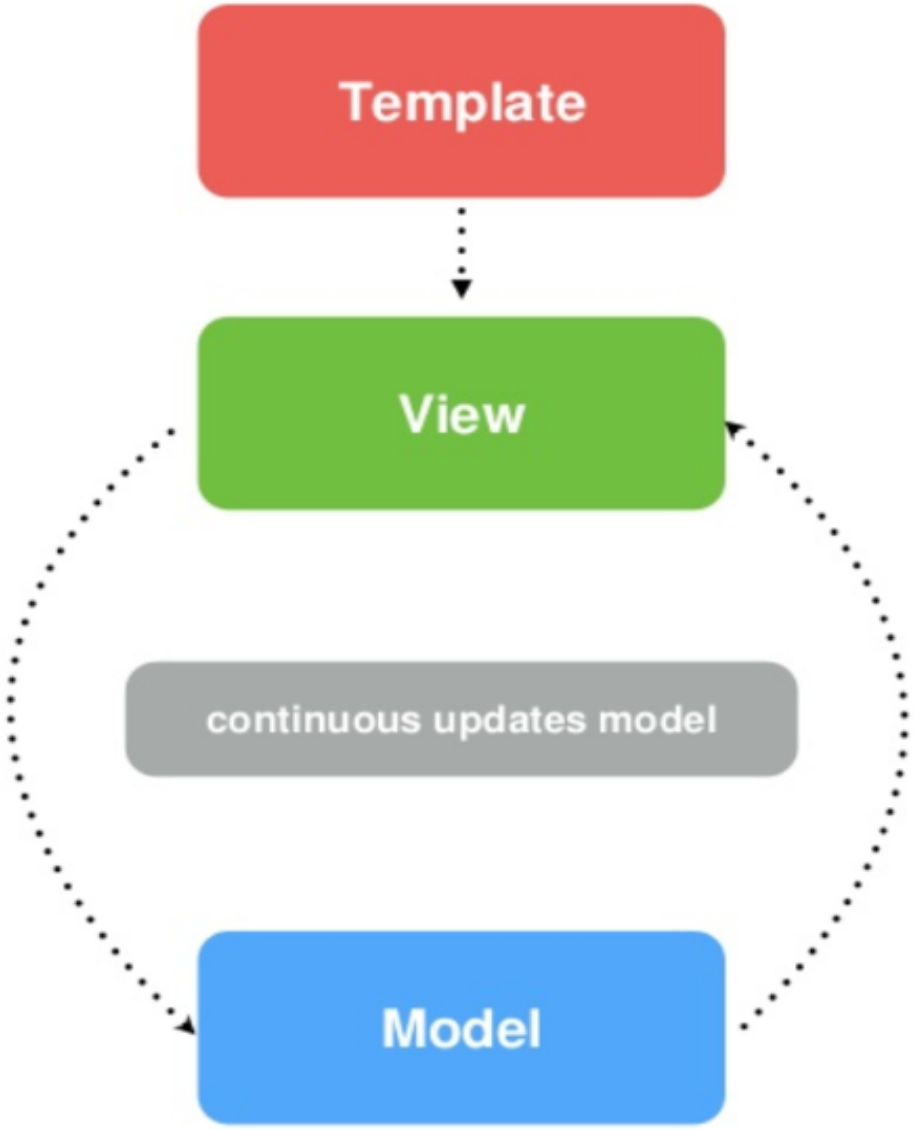


Рисунок 2.3 – Структура використання AngularJs

Two-way binding формується з допомогою компілятора та, безпосередньо, завдяки $compile. Тоді Angular.js генерує DOM-елементи, що використовуються в подальшому. $compile перетворює HTML-код у функцію, яка викликається для генерації контенту.

Архітектура Angular:

Module - структури, у яких директиви, сервіси, компоненти об’єднані якоюсь логікою;

Component - клас і з шаблоном, який відповідає за застосування логіки на сторінці. Частина цього коду може використовуватись у всьому додатку;

Template - частина HTML-коду із синтаксисом;

Service - клас typescript, який призначений для зберігання, отримання і обробки даних;

Router - відповідає за відображення контенту при переході між екранами;

Pipe - форматує дані в HTML-шаблоні. Pipe, які найчастіше використовуються - date (форматування дати), number (форматування числа), uppercase (переведення рядка у верхній регістр), slice (обрізання рядка), decimal (“диктує” формат числа);

Directives - зміна зовнішнього вигляду або поведінки DOM-елемента (Document Object Model).

Охарактеризуємо основні директиви Angular.js:

ng-app - забезпечує зміну поведінки, використовуючи спеціальні теги;

ng-bind - виконує прив’язку до властивості innerText HTML-елемента;

ng-init - відбувається ініціалізація змінних додатку;

ng-model - вказує на необхідність двостороннього зв’язування даних;

ng-class - виконує визначення наборів класу, що застосовуватимуться до елементу;

ng-controller - забезпечує приєднання контролерів до HTML DOM;

ng-repeat - для кожного елементу колекції створюється декілька екземплярів;

ng-form - директива створює об’єкт FormGroup і відповідає за його подальше прив’язування його до форми;

ng-if - видалення/додавання елемента;

ng-for - перебирає в певному шаблоні елементи масиву.

В якості хмарного провайдера було обрано Heroku, головна перевага цього провайдера полягає в тому, що він підтримує найпопулярніші мови програмування, а також надає безкоштовний обліковий запис і базу даних. Для датчика було обрано інфрачервоний датчик Sharp, який підключається до апаратної обчислювальної платформи Arduino, а для з’єднання датчика та сервера використовувався міні-модуль Ethernet. В якості бази даних було обрано PostgreSQL [16].

Під час вибору мов програмування я обрав Java для backend частини веб-сервісу та для отримання даних з OSM. Це обумовлено доступом Java до великої кількості бібліотек з відкритим кодом. Провідні програмісти розробили шаблони, які значно спрощують процес розробки. Підтримка таких гігантів, як Google і

Apache, свідчить про довговічність використання цієї мови в проєктах. Java відрізняється добре продуманим API, широким вибором інструментів і численними фреймворками[15].

Java підходить для створення різноманітних програм: від корпоративного ПЗ до новинних сайтів, настільних ігор і пошукових систем. Цією мовою користуються популярні ресурси, як-от Amazon.com, eBay.com, LinkedIn, AliExpress та інші. Крім того, Java дозволяє швидше розробляти програми, знижує витрати на реалізацію і забезпечує створення надійного застосунку, що працює на будь-яких операційних системах і пристроях.

Коротко про Spring Boot: Spring широко використовується для створення масштабованих застосунків. Для веб-застосунків він пропонує модуль Spring MVC, який допомагає створювати масштабовані веб-застосунки. Однак, одним з головних недоліків традиційних проєктів на Spring є складність та тривалість їх конфігурації, що може бути особливо складним для новачків. Це питання вирішує Spring Boot.

Spring Boot — це open-source фреймворк, який розробляє та підтримує компанія Pivotal. Він базується на Spring і включає в себе всі його можливості. Використовуючи Spring Boot, розробники можуть швидко розпочати роботу, не витрачаючи багато часу на налаштування конфігурацій для запуску веб-застосунку. (рис. 2.4)

Рисунок 2.4 – Структура Spring Boot

Отже, Spring Boot вирішує наступні завдання:

1. Спрощує процес початку роботи з фреймворком Spring.
2. Зменшує необхідність ручного створення конфігурацій застосунку.
3. Виконує автоконфігурацію на основі файлів property і JAR classpath.
4. Допомагає усувати конфлікти з залежностями для Maven та Gradle.
5. Надає вбудований сервер Tomcat.
6. Спрощує створення та підтримку REST endpoints.
7. Деплой застосунку відбувається дуже просто: файли WAR або JAR можуть бути легко розгорнуті на сервері Tomcat.
8. Підтримує мікросервісну архітектуру.

Архітектура Spring Boot застосунку складається з чотирьох основних рівнів (рис. 2.5, рис. 2.6):

- Presentation Layer – обробляє HTTP-запити, перетворює JSON у Java-об'єкти та автентифікує запити, передаючи їх до бізнес-рівня.
- Business Layer – відповідає за всю бізнес-логіку застосунку. Цей рівень складається з класів сервісів і використовує методи, надані рівнем доступу до даних. Він також виконує валідацію та авторизацію.
- Persistence Layer – містить усю логіку взаємодії з базою даних, конвертуючи Java-об'єкти в рядки бази даних та назад.
- Database Layer – на цьому рівні виконуються всі CRUD-операції.

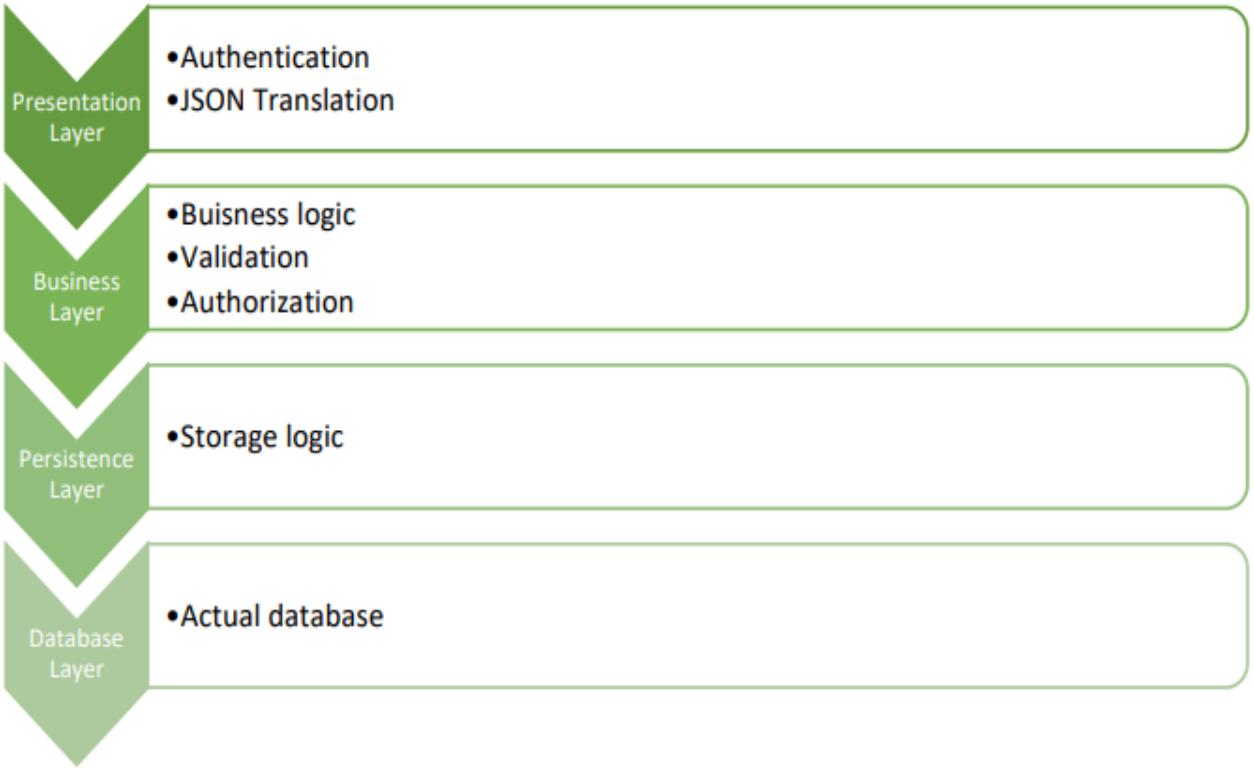


Рисунок 2.5 – Рівні архітектури Spring Boot

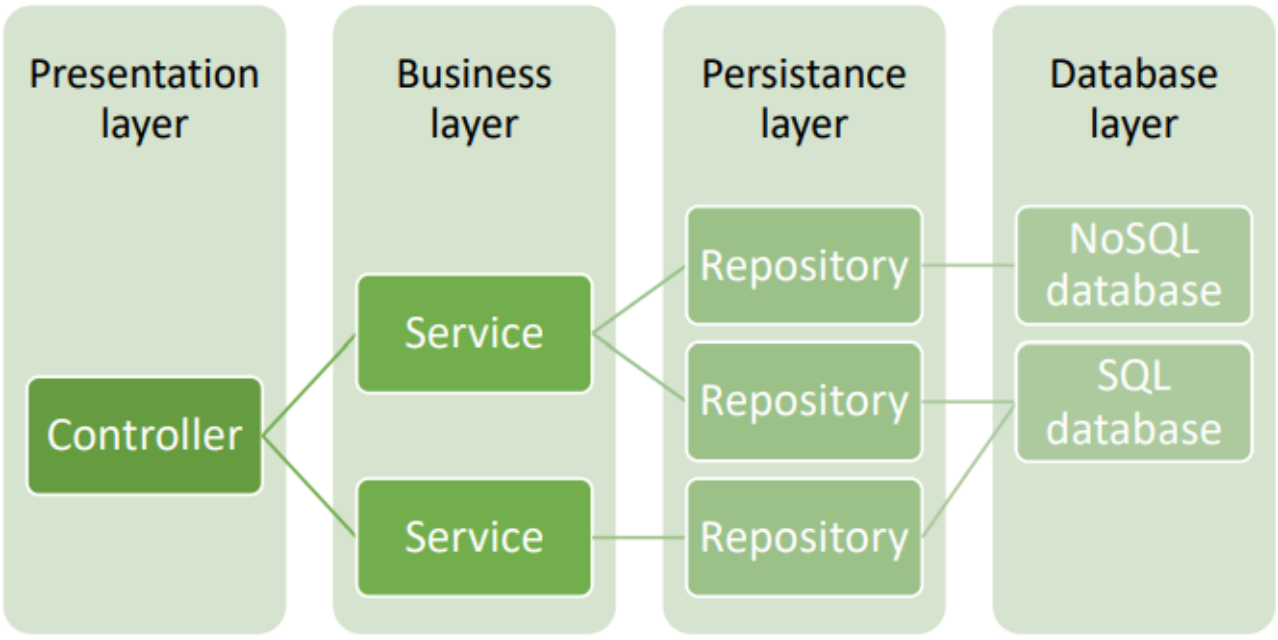


Рисунок 2.6 – Рівні архітектури Spring Boot застосунку

Spring Boot Flow (рис. 2.7):

Клієнт виконує HTTP запит: Клієнт ініціює запит до серверу (наприклад, через браузер чи мобільний додаток).

Запит надходить до контролера: Контролер отримує запит і визначає, який метод обробить цей запит, а також викликає необхідні сервіси для виконання логіки. Presentation Layer:

Authentication: Перевірка автентифікації користувача.

JSON Translation: Перетворення даних в формат JSON для обміну.

Обробка бізнес-логіки на рівні сервісів: На цьому етапі обробляється вся необхідна бізнес-логіка. Сервісний шар виконує операції над даними, використовує валідатори, авторизацію і взаємодіє з моделями даних через JPA.

Business Layer:

- Business Logic: Реалізація бізнес-правил.
- Validation: Перевірка вхідних даних.
- Authorization: Перевірка прав доступу.

Повернення відповіді клієнту: Після обробки запиту, сервер формує відповідь у вигляді View (наприклад, HTML-сторінки) або JSON даних (для API).

Persistence Layer:

- Storage Logic: Операції з даними, збереження та оновлення. Database Layer:
- Actual Database: Реальна база даних (SQL або NoSQL).
- Repository Layer: Взаємодія з базою даних через репозиторії для доступу до даних.

Опис рівнів:

- Presentation Layer: Відповідає за взаємодію з користувачем.
- Business Layer: Реалізує логіку програми.
- Persistence Layer: Забезпечує доступ до даних.
- Database Layer: Фізичне зберігання даних у базі даних.

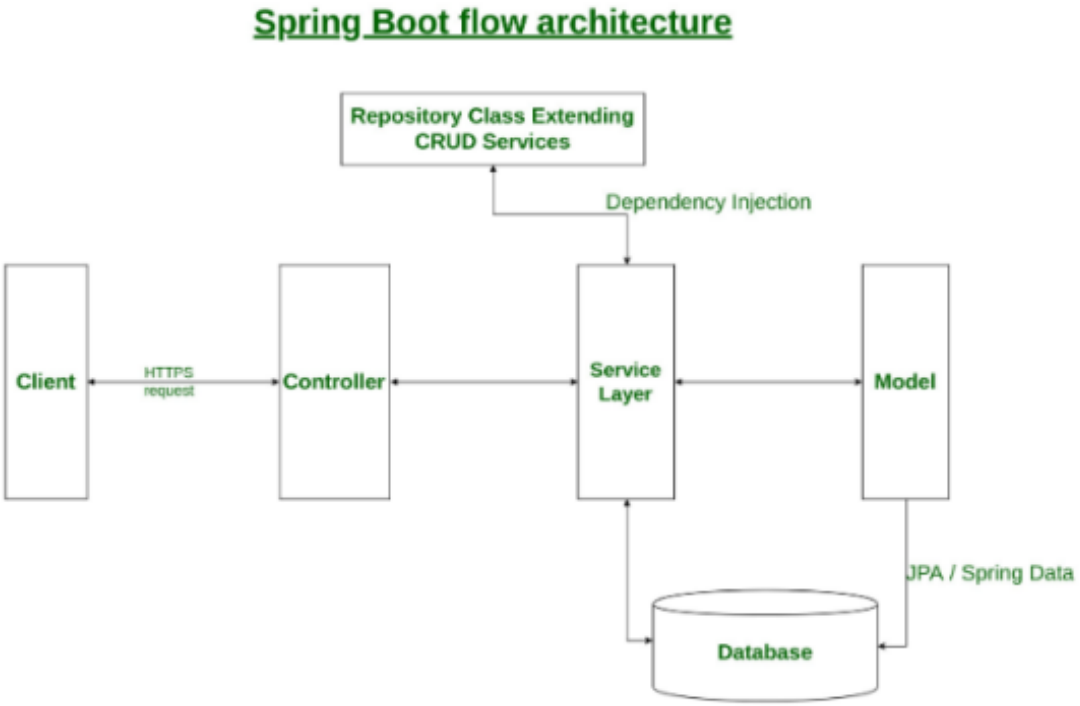


Рисунок 2.7 – Spring boot flow

Дані зберігаються та передаються між рівнями і клієнтом у формі DTO та моделей. Модель відображає формальні основні структури даних. DTO (об'єкт передачі даних) розроблений для переміщення інформації від сервера до клієнта і назад..

PostgreSQL — це об'єктно-реляційна система керування базами даних (СУБД), яка підтримує кілька типів індексів:

- B-дерево,
- хеш,
- R-дерево,
- GiST (Generalized Search Tree),
- GIN (Generalized Inverted Index).

При потребі користувач може створювати додаткові, найновіші типи індексів.

У цій роботі для швидкого пошуку найближчих точок, окрім звичайних індексів, використовувався індекс GiST [17].

GiST (англ. *Generalized Search Tree*) — це узагальнена індексна структура, що базується на ідеях R-дерева. Вона забезпечує стандартні методи:

- навігації по дереву,
- оновлення дерева (розбиття та видалення вузлів).

GiST є збалансованим деревом за висотою:

- Листові вузли містять пари *(key, rid)*, де:
- key — ключ,
- rid — вказівник на відповідний запис на сторінці даних.
- Внутрішні вузли містять пари *(p, ptr)*, де:
- p — предикат (ключ пошуку), що застосовується до всіх успадкованих вузлів,
- ptr — вказівник на інший вузол дерева.

Ця структура підтримує основні методи:

- SEARCH для пошуку,
- INSERT для додавання елементів,
- DELETE для видалення, а також інтерфейс для розробки спеціалізованих методів, що контролюють роботу базових функцій.

Метод SEARCH у структурі індексів контролюється функцією Consistent, яка повертає значення true, якщо вузол задовольняє заданий предикат.

Метод INSERT функціонує за допомогою трьох ключових функцій:

- penalty — оцінює складність вставки нового елемента у вузол.
- picksplit — виконує розділення вузла в разі його переповнення.
- union — забезпечує перебудову дерева за необхідності.

Метод DELETE призначений для пошуку листового вузла, що містить ключ, видалення пари (key, rid) і, за потреби, перебудови батьківських вузлів за допомогою функції union.

GiST (Generalized Search Tree) є загальним індексом, який використовується для повнотекстового пошуку в PostgreSQL. Він створює описовий підпис для кожного вектора, який включає всі маркери, присутні в документі. Принцип роботи нагадує розрядні індекси, але має певні відмінності.

Наприклад, нехай токен w1 пов'язаний із сигнатурою 001000 маркер w2 дорівнює 000010. Тоді документ, що містить лише маркери w1 і w2, приєднається до підпису 001010 (001000 | 000010). На відміну від бітових індексів, відображення

токенів у сигнатурі є неоднозначним, тобто маркер w3 із сигнатурою 001000 може існувати. Це дає змогу суттєво зменшити обсяг індексу, проте потребує додаткової перевірки на повну відповідність між маркерами запиту та документом. Слід також зазначити, що якщо маркер запиту має підпис, наприклад, 000001, то документ із підписом 001010 не містить його однозначно, незважаючи на неоднозначність відображення маркера.

Перевагою GiST є швидкість формування та обсяг індексу в порівнянні з GiN, що в три рази менше, що робить його більш придатним для даних, які постійно оновлюються. Натомість для статичних або рідко оновлюваних даних більш ефективним буде індекс GiN, оскільки він забезпечує в три рази швидший пошук..

2.2 Огляд можливостей ресурсу OpenStreetMap

OpenStreetMap (OSM) — це відкритий проект, метою якого є збір, збереження та розповсюдження загальнодоступних геопросторових даних, а також створення інструментів для їх обробки зусиллями спільноти волонтерів.

Готові до використання карти можна знайти на офіційному сайті OpenStreetMap та на численних інших ресурсах, які пропонують додаткові функції. За допомогою функції «Експорт» користувачі можуть отримати HTML-код для вставки на свій сайт з можливістю додавання маркерів. Існують також більш потужні інструменти, такі як Easymap, які дозволяють, наприклад, будувати маршрути.

Карти OSM доступні для завантаження у вихідному форматі та у форматах, сумісних з різними автомобільними та іншими навігаторами. Також розроблені спеціальні програми для мобільних пристроїв, таких як Java, iOS, Windows Mobile, Android та інших, а також для комп'ютерів.

Фонд вільного програмного забезпечення закликає всіх долучитися до проекту OpenStreetMap, щоб створити вільну альтернативу пропрієтарному Google Earth. Заміна останнього є одним з найвищих пріоритетів Фонду. OpenStreetMap

позиціонується як відповідь на комерційно обмежені геодані, пропонуючи альтернативу з відкритим кодом.

Карти OpenStreetMap активно використовуються багатьма організаціями та ресурсами, серед яких Організація Об'єднаних Націй, Вікіпедія, Microsoft Bing Maps, MapQuest, Вікімапія, Оксфордський університет, сайт президента США, французька газета *Libération*, американські рятувальники, а також такі проекти, як Monopoly City Streets.

Офіційно про підтримку використання карт OpenStreetMap у своїх продуктах заявляють навігаційні компанії Garmin, Автосупутник і PocketGIS.

За даними Alexa.com, більшість відвідувачів платформи походять з Німеччини та США, а лідерами за популярністю OpenStreetMap є Австрія та Німеччина [18].

OpenStreetMap (OSM) — це не просто карта, а повноцінна база геопросторових даних. Вона включає географічні координати точок, а також дані про об'єкти вищого рівня: лінії, що з'єднують точки, зв’язки, які охоплюють точки та лінії, і атрибути цих об'єктів. Завдяки цим даним можна створювати різноманітні сервіси з унікальними способами відображення та функціоналом.

Мапа, пошук об’єктів і сервіс прокладання маршрутів, представлені на головній сторінці OpenStreetMap, є лише одним із прикладів можливого використання цієї багатофункціональної бази даних. (рис. 2.8.).

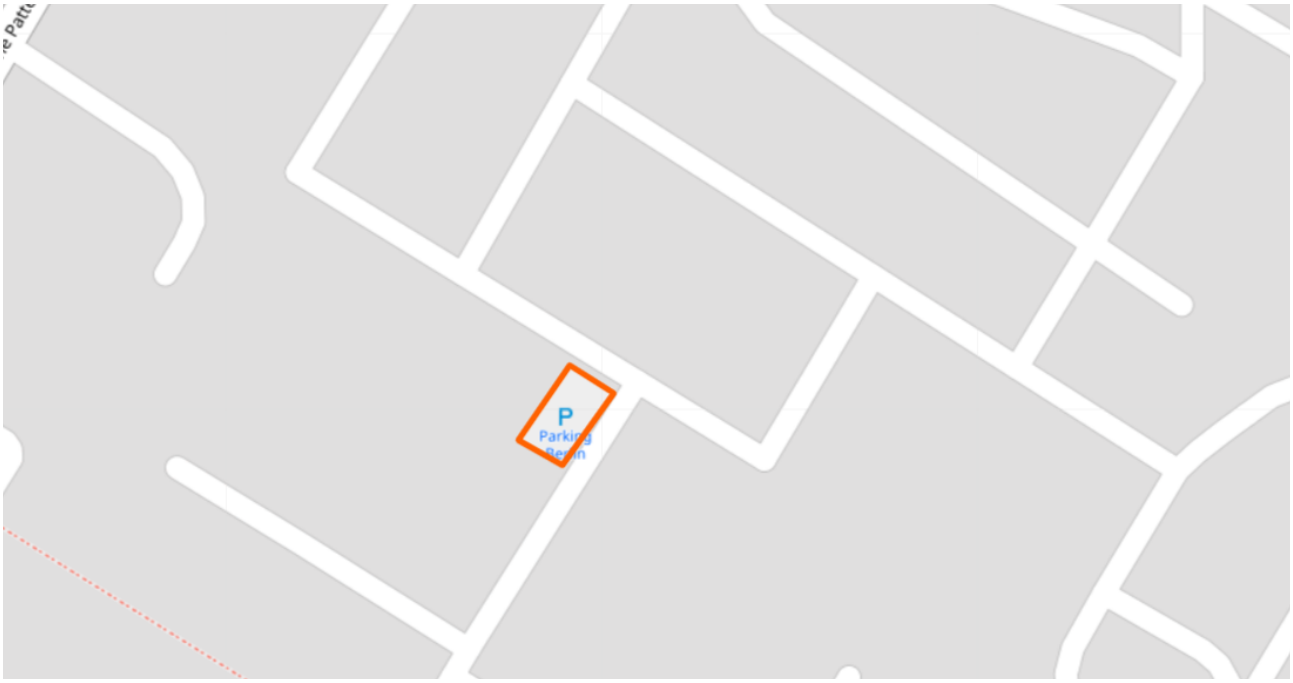


Рисунок 2.8. – Зображення паркувальної зони на карті OSM.

Мапи OpenStreetMap є двовимірними і не містять даних про висоти над рівнем моря чи ізолінії. Проте зростає популярність позначення висотних характеристик окремих об'єктів, а також розвиваються проекти для їх рендерингу.

Дані з карт, які охоплюють як всю Землю, так і окремі її ділянки, можуть бути завантажені у форматі OSM або конвертовані в інші формати для використання в геоінформаційних системах, для створення друкованих карт або для перетворення у формати, сумісні з GPS-навігаторами.

OpenStreetMap надає можливість стилізувати дані відповідно до потреб користувача. На головному веб-сайті OpenStreetMap доступні чотири різні стилі (шари) карт, які реалізуються за допомогою безкоштовної JavaScript-бібліотеки leaflet.js. Однак користувачі можуть не обмежуватися лише цією бібліотекою, а обрати інші вільні або патентовані аналоги [1].

2.3 Висновки

У даному розділі виконано формування програмних вимог для веб-сервісу та було побудована концепція кінцевого продукту. Було проведено відбір оптимальних інформаційних технологій для серверної частини було обрано мову

програмування Java та фреймворк для створення веб серверу із використанням патерну MVC що підрозуміває під собою умовний поділ додатку на три частини. Для користувацької частини додатку було використано мову програмування JavaScript та бібліотеку AngularJs яка використовується для створення одно сторінкових додатків за допомогою патерну MVC. OpenStreetMap розглянуто для отримання даних по місцезнаходженню місця для паркування та їх зображення на карті.

3 ВИБІР ОПТИМАЛЬНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

3.1 Збирання та адаптація даних

Для отримання даних будемо використовувати дані які нам пропонує OSM. Для їх отримання будемо використовувати Overpass API. Overpass API (раніше відомий як OSM Server Side Scripting або OSM3S до 2011 року) — це API лише для читання, який обслуговує спеціальні вибрані частини картографічних даних OSM. Він діє як база даних у мережі: клієнт надсилає запит до API та отримує назад набір даних, який відповідає запиту.

Процес створення застосунку для різних інформаційно-технологічних платформ зазвичай потребує написання унікального коду для кожної з них на відповідній мові програмування. Це завдання досить трудомістке. Застосування кросплатформних засобів розробки дозволяє суттєво зменшити час та зусилля, необхідні для розроблення додатків під різні платформи.

Однією з таких технологій є Apache Cordova, яка забезпечує підтримку багатьох платформ. Демонстрація створення прототипу фреймворку проводиться на прикладі платформи Android. Популярність гібридних застосунків зростає, оскільки їх розроблення можливе за участю фахівців з HTML, CSS та JavaScript. Окрім цього, розробники мають змогу повторно використовувати напрацювання, отримані під час створення веб-застосунків для муніципальних програмних комплексів та супутнього програмного забезпечення [17].

Для створення кросплатформних міських мобільних застосунків використовуються такі технології:

- Cordova/PhoneGap;
- React Native;
- Appcelerator Titanium;
- Xamarin.

Для отримання даних будемо використовувати дані які нам пропонує OSM. Для їх отримання будемо використовувати Overpass API. Overpass API (раніше відомий як OSM Server Side Scripting або OSM3S до 2011 року) — це API лише для читання, який обслуговує спеціальні вибрані частини картографічних даних

OSM. Він діє як база даних у мережі: клієнт надсилає запит до API та отримує назад набір даних, який відповідає запиту.

На відміну від основного API , який оптимізовано для редагування, Overpass API оптимізовано для споживачів даних, яким потрібно кілька елементів за один раз або до приблизно 10 мільйонів елементів за кілька хвилин, обидва вибрані за критеріями пошуку, як-от розташування, тип об'єктів, властивості тегів, близькість або їх комбінації. Він діє як сервер бази даних для різних служб [18].

Крім того, існує довідник Overpass QL/мовна довідка. Настійно рекомендуємо ознайомитися з різними функціями за допомогою Overpass Turbo, інтерактивного веб-інтерфейсу. Для застарілих програм також існує рівень сумісності, який забезпечує плавний перехід від XAPI (рис. 3.1).

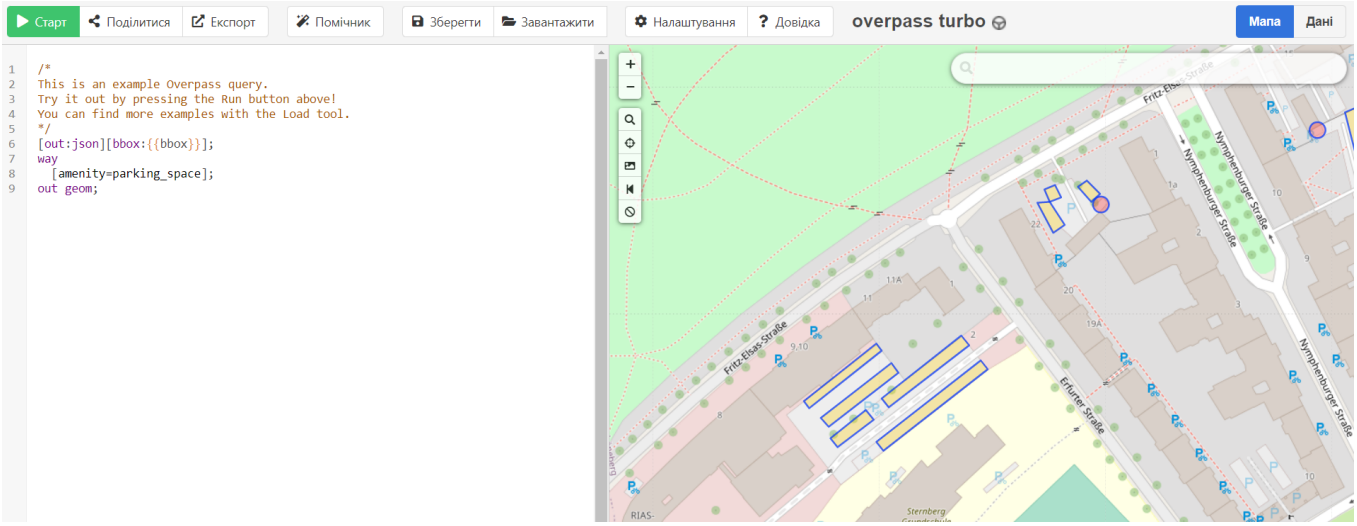


Рисунок 3.1 – Інтерфейс Overpass turbo

На даному малюнку зображено інтерфейс Overpass turbo на якому зображено приклад запиту для отримання парковок у заданому місці на карті та повернення даних у форматі json. Overpass API має можливість повертати дані в різних форматах, таких як json, xml, csv [19].

```
{
  "type": "way",
  "id": 834681322,
  "bounds": {
    "minlat": 52.4807606,
    "minlon": 13.3387922,
    "maxlat": 52.4810250,
    "maxlon": 13.3393207
  },
  "nodes": [
    3406961568,
    7791676788,
    7791650681,
    7791650682,
    7791676787,
    3406961568
  ],
  "geometry": [
    { "lat": 52.4810250, "lon": 13.3392765 },
    { "lat": 52.4809905, "lon": 13.3393207 },
    { "lat": 52.4807606, "lon": 13.3388365 },
    { "lat": 52.4807952, "lon": 13.3387922 },
    { "lat": 52.4808614, "lon": 13.3389317 },
    { "lat": 52.4810250, "lon": 13.3392765 }
  ],
  "tags": {
    "amenity": "parking_space",
    "capacity": "18"
  }
},
```


Рисунок 3.2 – Приклад даних у вигляді JSON.

На рисунку 3.2 зображено BoundingBox парковки, координати її меж та загальна кількість місць на парковці [10].

Оскільки за останні кілька років кількість автомобілів значно зросла (див. рис. 3.3), зростає і потреба в паркувальних місцях та їх пошуку. Якщо вважати, що середній водій щодня витрачає 20 хвилин на пошуки паркування, то це становить близько 120 годин на рік, які могли б бути використані на більш продуктивні справи [17].

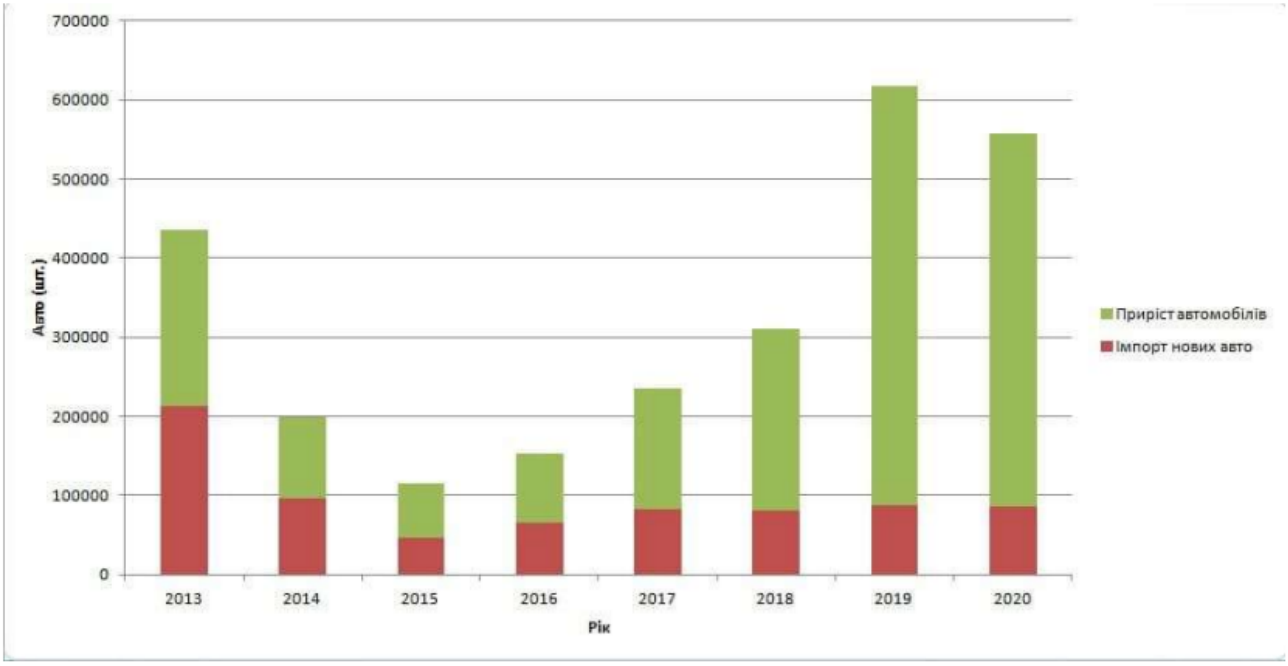


Рисунок 3.3 – Приріст кількості транспортних засобів в Україні

Як видно з рисунка 3.3, загальне збільшення кількості транспортних засобів складається з імпорту нових автомобілів (червона частина колонки) та вживаних автомобілів, ввезених з-за кордону (зелена частина колонки). Зокрема, у період з 2016 по 2020 рік кількість імпорту нових автомобілів залишалася майже стабільною, тоді як частка імпорту вживаних автомобілів поступово зростала. Це явище пояснюється законодавчими змінами щодо розмитнення автомобілів, ухваленими 25 листопада 2018 року, які спростили процедуру розмитнення вживаних транспортних засобів, ввезених з-за кордону[5].

Для створення прототипу системи розумного паркування першим кроком є дослідження функцій і вимог, необхідних для такого типу систем. Це дозволяє краще зрозуміти процес розробки всієї системи. Установлення камери у громадській зоні вимагає спеціального дозволу, оскільки виникають питання конфіденційності. Надсилання зображень на сервер або хмару через Інтернет також супроводжується проблемами безпеки, що слід враховувати під час проєктування. Щоб вирішити цю проблему безпеки, можна використовувати певний процес шифрування. Обмеження, пов’язані із системами інтелектуального паркування, можуть вплинути на їх зручність. Виконання завдань з обробки зображень без

передачі даних через Інтернет дозволяє уникнути питань, пов'язаних із конфіденційністю та безпекою. Тому ці аспекти слід враховувати при розробці інтелектуальних рішень для паркування. Для динамічної розумної системи паркування важливо автоматично визначати доступні місця. Це можна зробити за допомогою алгоритмів виявлення об’єктів, зокрема через технології глибокого навчання, які забезпечують простоту й ефективність.

Альтернативний підхід – це використання визначення ліній паркувальних місць, але цей метод має свої недоліки: лінії можуть стиратися, розмиватися чи бути прихованими, наприклад, снігом. Передача даних у системі забезпечує реєстрацію інформації на сервері, щоб користувачі могли отримати дані про доступність паркувальних місць і кількість вільних зон для паркування.

Перед подорожжю Європою важливо вивчити правила дорожнього руху в кожній країні, через яку проходить маршрут. Хоча основні принципи та знаки схожі на українські, у кожної країни є свої особливості, зокрема щодо штрафів і покарань. Штрафи в Європі значно вищі й можуть сягати кількасот євро, а спроби "домовитися" із поліцією караються суворіше.

Також важливо записати контактні телефони дорожньої поліції та страхової компанії у кожній країні, оскільки вони знадобляться у разі ДТП чи технічних проблем.

Хоча правила дорожнього руху в Україні складно назвати ідеальними, мало хто приходить в замішання. Але є країни, які дивують несподіваними, неймовірними і навіть абсурдними законами.

Ми вирішили розібратися, які правила дорожнього руху в інших країнах дивують іноземців і чим пояснюються незвичайні заборони і дозволи.

Вирушаючи в Європу в поїздку, ви їздитимете дуже багато, тому бажано встановити газ на авто в Сервіс Газ Vip. Газобалонне обладнання дозволить долати далекі відстані без необхідності дозаправитися. Також це суттєво заощадить гроші на паливі, а заощаджені гроші можна буде витратити на приємні дрібниці.

Часто водії, які приїхали до країн Європи, ризикують потрапити під штраф, а через незнання правил паркування в країнах Європи.

Основні правила паркування у Європі запам'ятати не складно, оскільки розмітка використовується однакова.

Жовта зигзагоподібна лінія – паркуватися заборонено.

Біла - можна безкоштовно залишити автомобіль з 18:00 до 8:00, а в решту годин доби доведеться сплатити стоянку автомобіля.

На дорожніх знаках є вся необхідна інформація - в які дні та час паркуватись можна.

У Європі поширені підземні стоянки, а наявність вільних місць повідомляє табло над в'їздом.

У центрі великих міст є багато безкоштовних парковок з обмеженим часом перебування. Вони, в основному, розташовані неподалік головних визначних пам'яток, а також великих торгових центрів.

Знайти паркування будь-якого типу у більшості великих міст вам допоможуть спеціалізовані сайти на кшталт car-parking.eu.

Ну і, звичайно ж, не забувайте, що у разі неправильного паркування водія очікує штраф і досить високий. В основному сума у вашій квитанції коливатиметься в межах від 40 до 80 євро [7]. Однак, якщо ви залишили авто на місці, призначеному для інвалідів, сума штрафу буде від 200 до 400 євро в залежності від країни, в якій ви знаходитесь.

Інспекція з паркування в Києві змінила свою стратегію, перейшовши від попереджувальних заходів до штрафування водіїв за порушення правил парковки. Це стало відомо через публікацію начальника інспекції з парковки Дмитра Рахматулліна на його сторінці в Facebook. Він повідомив, що інспектори досі не мають сертифікованих технічних засобів фотофіксації, тому штрафи накладаються шляхом розміщення повідомлень про штрафи на лобових стеклах автомобілів. Крім того, інспектори почали використовувати велосипеди сервісу прокату Nextbike для покращення результативності своєї роботи [19].

У Луцьку проблема з паркуванням залишається актуальною: 80 % порушень ПДР стосуються паркування в недозволених місцях. Водночас, незважаючи на активність місцевих активістів, ситуація з паркуванням не змінюється. У місті достатньо заборонених місць для паркування, але не вистачає облаштованих

парковок. Пішоходи часто скаржаться на водіїв, які паркуються неправильно, а водії, в свою чергу, нарікають на відсутність парковок, підкреслюючи бездіяльність влади. Влада наголошує, що бюджет обмежений і не дозволяє вирішити всі проблеми, зокрема й ці [8].

На відміну від основного API, який оптимізовано для редагування, Overpass API оптимізовано для споживачів даних, яким потрібно кілька елементів за один раз або до приблизно 10 мільйонів елементів за кілька хвилин, обидва вибрані за критеріями пошуку, як-от розташування, тип об’єктів, властивості тегів, близькість або їх комбінації. Він діє як сервер бази даних для різних служб.

Також в подальшому можна буде використовувати інші ресурси для збору даних про наявність місць для паркування.

Блок-схема системи показує загальну структуру, включаючи основні блоки, підсистеми та їх взаємозв'язки. Система поділяється на дві основні підсистеми:

- підсистема автомобіля;
- підсистема паркування.

Підсистема автомобіля складається з чотирьох груп:

- блок датчиків та інформаційних пристроїв (ІЧ);
- блок виконавчих пристроїв автомобіля;
- блок бездротових модулів автомобіля;
- пристрої обробки інформації (бортовий комп'ютер та інші блоки).

До першої групи входять камери, лідери, ультразвукові датчики, радари, датчики тиску в гальмівній системі та датчик кута повороту керма. Отримані дані передаються на подальшу обробку. Пристрої обробки інформації, зокрема бортовий комп'ютер і блоки управління виконавчими пристроями, обробляють дані від датчиків і камер. Бортовий комп'ютер передає потік даних у паркувальну підсистему через бездротові модулі, а також отримує команди управління від системи для передачі в блоки виконавчих механізмів.

Підсистема паркування також складається з чотирьох груп:

- блок парктроніків та інформаційних підсистем (ППБ, ПВС);
- блок виконавчих пристроїв для паркування;
- блок бездротових паркувальних модулів;

- пристрої обробки інформації (блоки керування, сервери).

Дані, отримані з автомобіля, через комунікаційні модулі (бездротові модулі, Ethernet) передаються на сервер даних.

При цьому на сервер сервісу передаються дані з датчиків паркування: датчиків руху, датчиків освітлення при необхідності, але в основному дані про зайняті місця і дані про локалізацію від таких виконавчих пристроїв, як блок маячків (міток). Особливістю цієї структури є те, що самі мітки не є елементами інфраструктури. Мітки не беруть участі у формуванні мережі, маршрутизації та інших подібних завданнях. Їх основне завдання — вимірювання відстані до мобільних об’єктів (автомобілів) і передача отриманих результатів на сервер через транспортну інфраструктуру. Це дозволяє значно спростити конструкцію міток, знижуючи їх енергоспоживання та вартість. Мітки взаємодіють з інфраструктурою через двонаправлений радіоінтерфейс, який визначає частотний діапазон, форму радіохвилі, методи модуляції та кодування, формати пакетів, а також команди та звіти для обміну між мітками та інфраструктурою.

1. Діаграму можна умовно розділити на складові елементи:

- Автомобіль, на бортовий комп’ютер якого встановлено додаток, через комунікаційні модулі створює запит на підключення до системи, передаючи ідентифікатор для авторизації користувача.
- Встановлюється канал зв’язку з системою, перевіряється клієнт у базі даних, надається дозвіл на резервування медіаресурсу.
- Запит на виділення медіаресурсу (IP-адреси), на який будуть надходити медіадані з автомобіля.
- Збір даних з датчиків і відеокамер автомобіля, генерація медіапотоку та відправка його на спеціальний ресурс.
- Збір даних з датчиків паркування та їх передача разом з даними автомобіля на сервер сервісу для розрахунку контрольних точок, маршруту та паркувального маневру.
- Обчислення на сервері додатків для генерації керуючих команд.
- Блок управління рухом автомобіля (БКР) передає команди на бортовий комп’ютер автомобіля.

- Команди обробляються і надсилаються на відповідні блоки керування виконавчими механізмами.

Для кращого розуміння функціонування системи розглянемо основні її етапи. У спрощеній реалізації виключено процес створення бази цифрових відбитків пальців і динамічного уникнення перешкод. Натомість модель середовища подається у вигляді карти, яку система отримує через інфраструктуру V2X або камери паркування. Це передбачає використання статичної карти паркування та точну самолокалізацію автомобіля.

Після підключення автомобіля до системи визначається його початкова позиція на карті, яка слугує стартовою точкою для планування маршруту. Планування здійснюється на основі статичної карти паркування та глобального плану маршруту, що складається з послідовності координат, які визначають шлях до пункту призначення. Система постійно перевіряє місцезнаходження транспортного засобу та оновлює маршрут до моменту досягнення кінцевої точки.

Після аналізу і перегляду функціональної схеми переходимо до етапу давайте розглянемо схему системи і послідовності дій наступним чином. Схема випадків використання, яка показана для розуміння того, як рухається автомобіль.

Після проведення аналізу та огляду функціональної діаграми, етапів роботи системи та діаграми послідовності, перейдемо до діаграми варіантів використання, яка ілюструє, як автомобіль рухається. Після виявлення перехідних позицій (цілей) з запланованого маршруту та маневру паркування на сервісному сервері, дані зберігаються або оновлюються на сервері даних. Після цього вони використовуються на сервері додатків для формування команд керування. Блок управління дорожнім рухом (БКР) передає їх в автомобіль через модулі зв'язку. Після цього здійснюється попередня обробка команд управління бортовим комп'ютером автомобіля і передача на відповідні блоки управління виконавчими механізмами. Наприклад, профіль швидкості, команди прискорення та уповільнення обробляються блоком електромотора (ЕМ) і блоком приводу гальм.

(BGP), щоб скинути швидкість при зміні напрямку або зупинці на кінцевій траєкторії.

3.2 Створення та заповнення бази даних

Система управління базами даних (СУБД) — це програмне забезпечення, яке забезпечує створення, зберігання, управління та доступ до баз даних. Вона об'єднує програмні та мовні засоби, що дозволяють ефективно працювати з даними.

Основними функціями СУБД є управління даними як у зовнішній пам'яті (на дисках), так і в оперативній пам'яті, використовуючи дисковий кеш. СУБД також забезпечує запис змін у журналі, виконання резервного копіювання та відновлення бази даних у разі збоїв. Вона підтримує мови баз даних, включаючи мову визначення даних та мову маніпулювання даними.

Сучасна СУБД зазвичай складається з таких компонентів: ядро, що відповідає за управління даними в пам'яті та журналізацію; процесор мови бази даних, який оптимізує запити та створює виконуваний внутрішній код; підсистема підтримки часу виконання, яка інтерпретує операції з даними та забезпечує інтерфейс для користувачів; а також сервісні програми, які додають додаткові функції для обслуговування інформаційної системи.

Для збереження даних будемо використовувати базу даних PostgreSQL.

PostgreSQL — це потужна система керування базами даних із відкритим кодом, яка широко використовується завдяки своїй стабільності, гнучкості та підтримці сучасних стандартів SQL. Розробка PostgreSQL почалася в 1986 році в Каліфорнійському університеті Берклі як еволюція проєкту Ingres. Під назвою POSTGRES система отримала кілька значних оновлень: у 1990 році була випущена друга версія з вдосконаленою системою маніпулювання таблицями, а в 1991 році з’явилася третя версія з підтримкою багатьох менеджерів збереження та покращеними механізмами запитів. У 1994 році проєкт отримав інтерпретатор SQL, оновлений сирцевий код і нову назву Postgres95.

У 1996 році систему перейменували на PostgreSQL, щоб підкреслити її фокус на підтримці SQL, і тоді ж вийшов перший офіційний реліз під новою назвою. З того часу PostgreSQL активно розвивається завдяки спільноті добровольців, що складається з провідних спеціалістів у галузі баз даних.

Система застосовується в різних галузях, таких як фінансовий аналіз, моніторинг потоків даних, відстеження астероїдів, медичні інформаційні системи та географічні інформаційні системи. Поточна версія PostgreSQL 14 підтримує всі основні операційні системи, включаючи Windows, Linux і macOS, і продовжує залишатися одним із найпопулярніших інструментів для роботи з базами даних. [13].

Для збереження гео-даних використовується бібліотека PostGIS.

PostGIS — це просторове розширення для об'єктно-реляційної бази даних PostgreSQL, яке додає підтримку географічних об'єктів і дозволяє виконувати просторові запити безпосередньо за допомогою SQL.

Основні переваги PostGIS:

- Підтримка географічних об'єктів: можливість зберігати та обробляти просторові дані (точки, лінії, полігони тощо).
- Інтеграція з SQL: виконання запитів про розташування безпосередньо в SQL (наприклад, обчислення відстаней, перевірка перетинів, обробка буферів).
- Розширена функціональність: надає функції, які рідко зустрічаються в інших системах, таких як Oracle Locator/Spatial та SQL Server.

Цей інструмент є ідеальним вибором для геоінформаційних систем (ГІС), аналізу просторових даних і розробки додатків, які потребують складних операцій із географічними даними [20].

Розглянемо ER-моделі бази даних яка відображає структуру бази даних що буде використовувати наш веб-сервіс(рис.3.4)

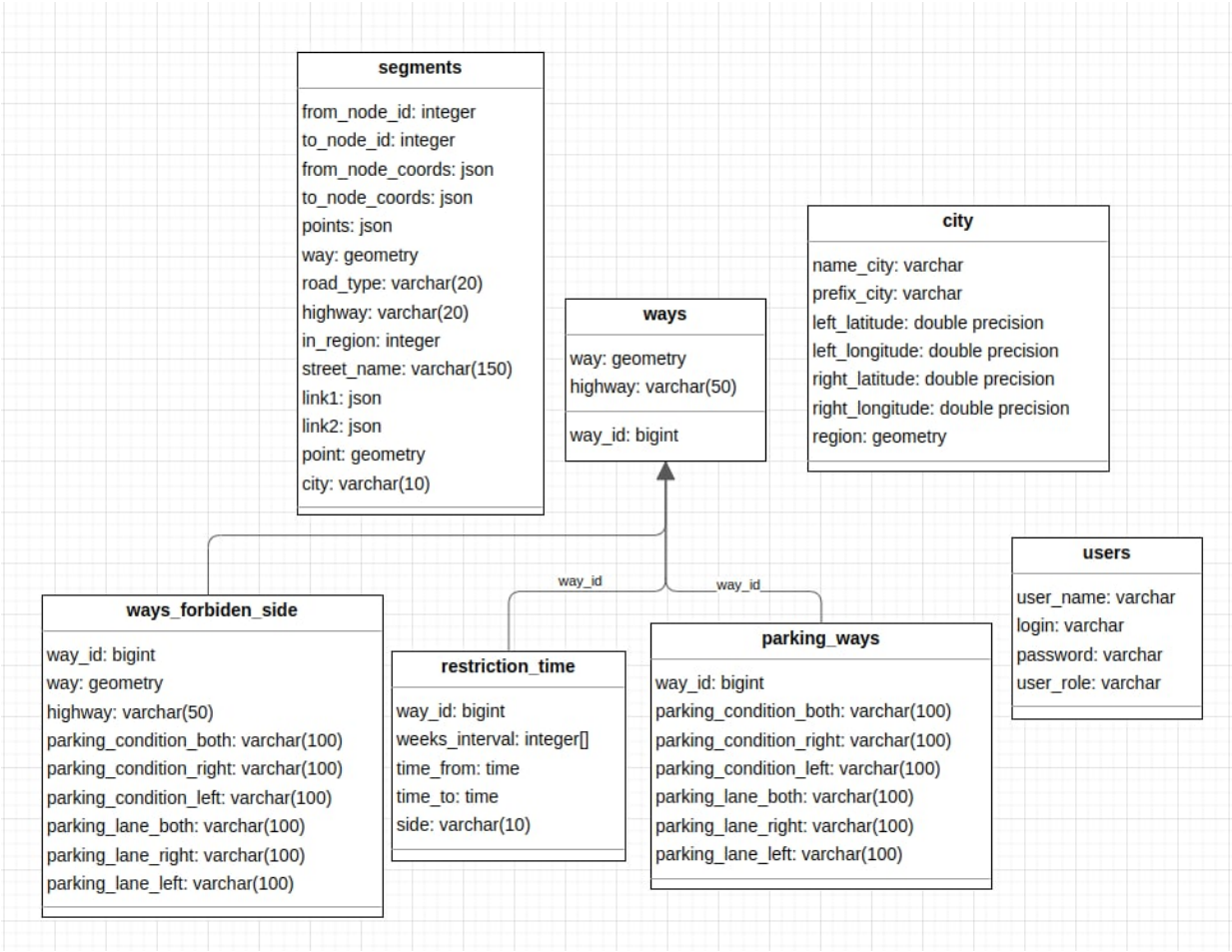


Рисунок 3.4 – ER-моделі бази даних.

3.4Висновки

У даному розділі було розглянуто можливі види отримання даних з OSM, розглянуто мову QL для написання запитів на Overpass Turbo для отримання та обробки даних. Розроблено базу даних веб-системи ідентифікації місць паркування.

4 РОЗРОБКА ВЕБ-СИСТЕМИ ІДЕНТИФІКАЦІЇ МІСЦЬ ДЛЯ ПАРКУВАННЯ ЗА ДАНИМИ СЕРВІСУ OPENSTREETMAP

4.1 Постановка задачі та вибір оптимальних підходів

Задля реалізації веб-сервісу ідентифікації місць для паркування, потрібно здійснити порівняльний аналіз існуючих технологій, для забезпечення дійсно швидко працюючого інструменту.

Для створення REST-контролера у Spring необхідно виконати наступні кроки:

Створення класу:

- Клас позначається анотацією @RestController, яка вказує, що цей клас є контролером для обробки REST-запитів.

Вказівка базового шляху:

- Анотація @RequestMapping використовується для визначення базового URL-шляху, який буде пов'язаний з контролером.

Методи-ендпоінти:

- В класі створюються методи для обробки запитів, кожен з яких позначається відповідною анотацією:
  - @GetMapping для отримання даних.
  - @PostMapping для створення нових записів.
  - @PutMapping для оновлення даних.
  - @DeleteMapping для видалення даних.

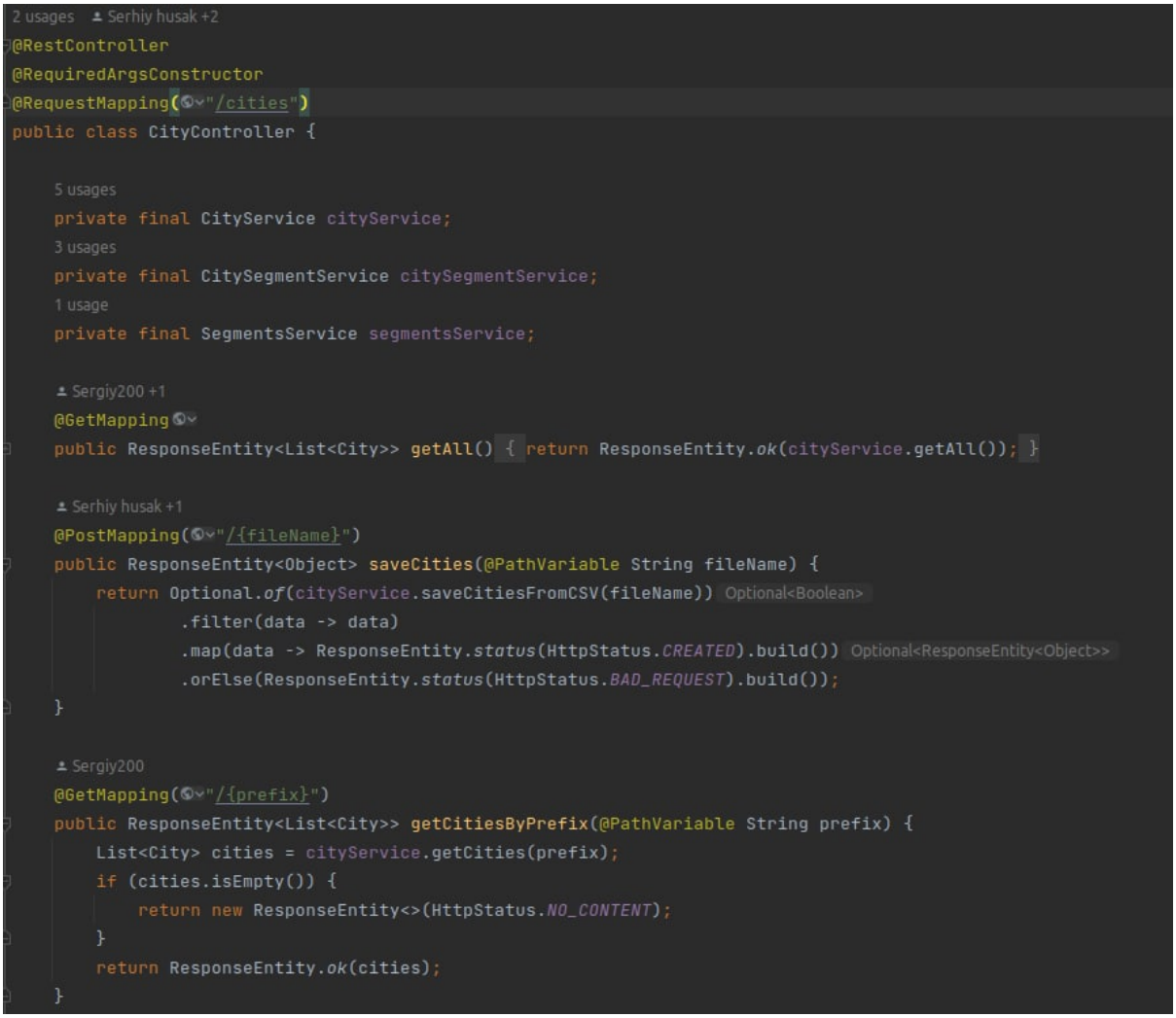
Обробка параметрів:

- Методи контролера можуть приймати параметри, передані в запитах, або об'єкти DTO, що містять передані дані.

Повернення результатів:

- Методи повертають об'єкти DTO, які відображаються у форматі JSON або XML (залежно від налаштувань).

Цей підхід дозволяє легко створювати REST API для взаємодії з клієнтами через HTTP-запити [20].



```
2 usages  Serhiy husak +2
@RestController
@RequiredArgsConstructor
@RequestMapping("/cities")
public class CityController {

    5 usages
    private final CityService cityService;
    3 usages
    private final CitySegmentService citySegmentService;
    1 usage
    private final SegmentsService segmentsService;

    Sergiy200 +1
    @GetMapping
    public ResponseEntity<List<City>> getAll() { return ResponseEntity.ok(cityService.getAll()); }

    Serhiy husak +1
    @PostMapping("/{fileName}")
    public ResponseEntity<Object> saveCities(@PathVariable String fileName) {
        return Optional.of(cityService.saveCitiesFromCSV(fileName)) Optional<Boolean>
                .filter(data -> data)
                .map(data -> ResponseEntity.status(HttpStatus.CREATED).build()) Optional<ResponseEntity<Object>>
                .orElse(ResponseEntity.status(HttpStatus.BAD_REQUEST).build());
    }

    Sergiy200
    @GetMapping("/{prefix}")
    public ResponseEntity<List<City>> getCitiesByPrefix(@PathVariable String prefix) {
        List<City> cities = cityService.getCities(prefix);
        if (cities.isEmpty()) {
            return new ResponseEntity<>(HttpStatus.NO_CONTENT);
        }
        return ResponseEntity.ok(cities);
    }
```

Рисунок 4.1 – Використання Rest контролерів для отримання даних

Amazon Simple Storage Service (Amazon S3) — це об'єктний сервіс зберігання, що забезпечує високу продуктивність, масштабованість, доступність та безпеку даних. Цей сервіс підходить для організацій будь-якого розміру та галузі, дозволяючи зберігати й захищати великі обсяги даних для різноманітних сценаріїв використання, таких як:

- Озера даних (Data Lakes).
- Хмарні та мобільні додатки.

Amazon S3 пропонує:

- Гнучкі класи зберігання для оптимізації витрат.
- Інтуїтивно зрозумілі інструменти адміністрування, що допомагають ефективно організовувати дані.
- Тонке налаштування доступу, яке відповідає бізнес-вимогам чи нормативним стандартам.

Це робить S3 ідеальним рішенням для зберігання даних у сучасних бізнес-середовищах. Та було додано логіку для отримання файлів із Amazon S3 (рис. 4.2)

```
@Slf4j
@Service
@RequiredArgsConstructor
public class S3ReaderServiceImpl implements S3ReaderService {
    1 usage
    private final AmazonS3 amazonS3;
    1 usage
    private final ReadCSVService readCSVService;

    5 usages  Serhiy husak
    public <T> List<T> getDataFromBucket(String bucketName, String fileName, Class<T> tClass, int lineToSkip) {
        try (Reader reader = getFileFromS3(bucketName, fileName)) {
            return readCSVService.csvReader(reader, tClass, lineToSkip);
        } catch (Exception e) {
            log.error("Error parsing file {}", fileName, e);
            throw new RuntimeException(e);
        }
    }

    1 usage  Serhiy husak
    private InputStreamReader getFileFromS3(String bucketName, String fileName) {
        S3Object s3Object = amazonS3.getObject(bucketName, fileName);
        return new InputStreamReader(s3Object.getObjectContent());
    }
}
```

Рисунок 4.2 – Зчитування файлів із Amazon S3

Отримавши файли із Amazon S3 потрібно отримати дані із CSV файлу та для цього було додано парсер який витягує дані із CSV у довільний тип даних. CSV — це текстовий формат, який може бути використаний для представлення табличних даних. Рядок таблиці може бути порівнянний з рядком тексту, який містить одне або кілька полів, розділених комами.

Формат CSV не є повністю стандартизованим. Хоча використання коми для поділу полів є очевидною ідеєю, такий підхід може мати проблеми, якщо вихідні табличні дані містять коми або переклади рядків. Можливим вирішенням проблеми ком і переносів рядків є укладання даних у лапки, однак вихідні дані можуть містити лапки. Крім цього терміном CSV можуть позначатися схожі формати, в яких роздільником є символ табуляції (TSV) або точка з комою. Багато програм, які працюють із форматом CSV, дозволяють вибирати символ роздільника та символ лапок (рис. 4.3).

```java
@Service
@RequiredArgsConstructor
public class ReadCSVServiceImpl implements ReadCSVService {

    1 usage  Serhiy husak
    @Override
    public <T> List<T> csvReader(Reader reader, Class<T> clazz, int lineToSkip) {
        return new CsvToBeanBuilder<T>(reader)
                .withType(clazz) CsvToBeanBuilder<T>
                .withSkipLines(lineToSkip)
                .build() CsvToBean<T>
                .parse();
    }
}
```

Рисунок 4.3 – Зчитування даних з CSV файлу

Також дані для обробки можна отримати із OpemStreetMap за допомогою Overpass API, що дуже зручно адже це відкриті дані (рис. 4.4).

```java
8 usages  Serhiy husak
@Override
public <T> List<T> getElements(BoundingBox boundingBox, String queryPattern, Class<T> tClass) {
    String query = createQuery(boundingBox, queryPattern);
    try {
        return mapper
                .readerForListOf(tClass)
                .readValue(getContent(query));
    } catch (Exception e) {
        log.error("Error parsing request with BoundingBox {}", boundingBox, e);
        throw new RuntimeException(e);
    }
}
```

Рисунок 4.4 – Отримання даних з Overpass API.

Для роботи з базами даних потрібно jOOQ об’єктно-орієнтовані запити, широко відомий як jOOQ, - це легке відображення бази даних бібліотека програмного забезпечення в Java що реалізує шаблон активного запису. Його мета - бути обома реляційний і об'єктно-орієнтований шляхом надання а мова домену для побудови запитів з класи, генеровані від а схема бази даних.

jOOQ стверджує, що SQL має бути на першому місці при будь-якій інтеграції баз даних. Таким чином, він не вводить нового текстового мова запитів, але

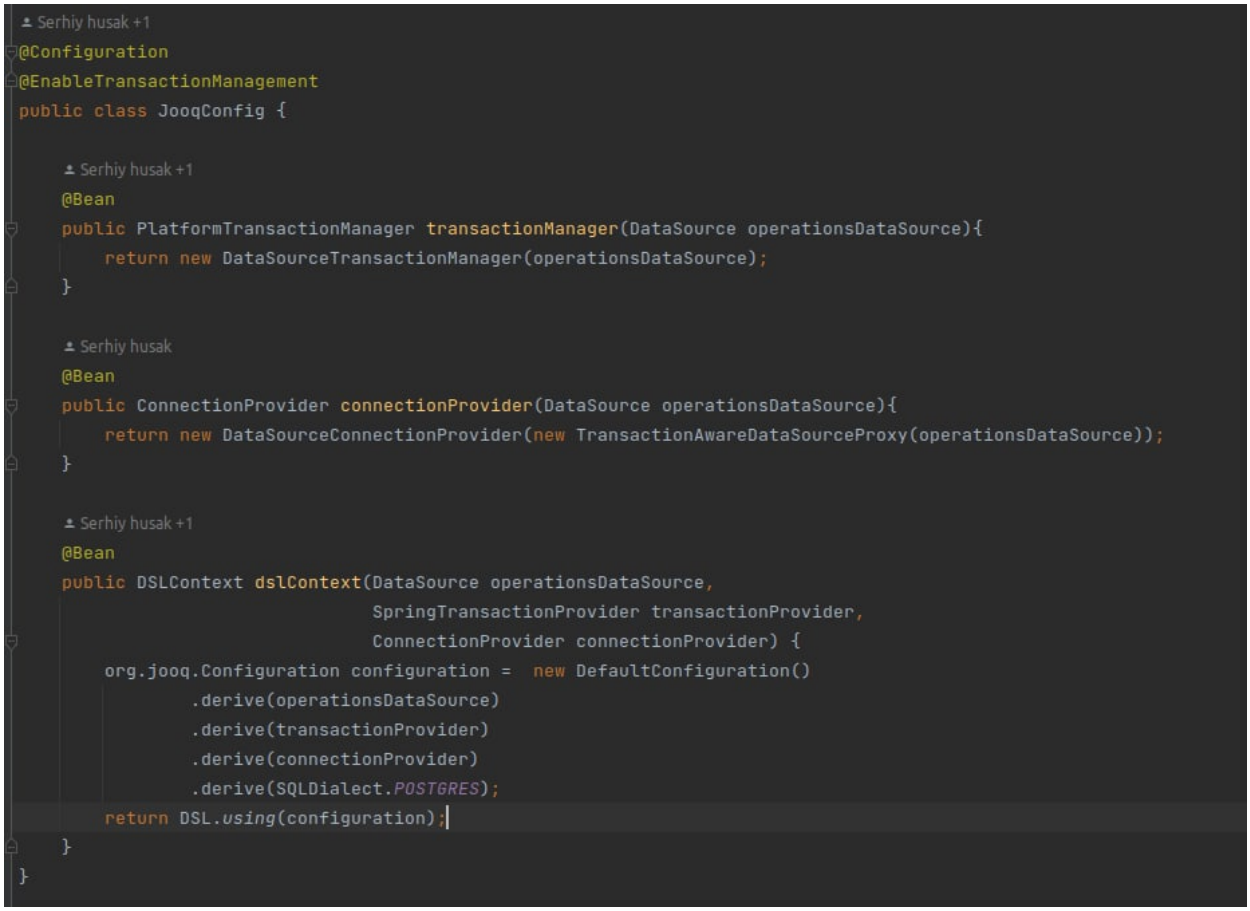
швидше дозволяє побудувати рівнину SQL з об’єктів jOOQ та коду, сформованого зі схеми бази даних. jOOQ використовує JDBC викликати базові запити SQL.

Хоча це забезпечує абстракція на додаток до JDBC, jOOQ не має такої ж функціональності та складності, як стандарт об’єктно-реляційне відображення бібліотеки, такі як EclipseLink або Зимовий сон.

Близькість jOOQ до SQL має переваги перед типовими об'єктно-реляційними бібліотеками зіставлення. SQL має багато функцій, які не можна використовувати в об'єктно-орієнтований парадигма програмування; цей набір відмінностей позначається як невідповідність об’єктно-реляційного імпедансу. Знаходячись близько до SQL, jOOQ допомагає запобігти синтаксичні помилки і проблеми зіставлення типів. Крім того, подбає про змінне зв’язування. Також у jOOQ можна створювати дуже складні запити, які включають псевдоніми, профспілки, вкладений виділяє та складні приєднання. jOOQ також підтримує специфічні для бази даних функції, такі як UDT, типи переліку, збережені процедури і рідні функції.

JOOQ — гарний вибір у програмі Java, де важливі SQL та конкретна реляційна база даних. Це альтернатива, коли JPA / Hibernate занадто багато абстрагує, JDBC занадто мало. Він показує, як сучасна предметно-орієнтована мова може значно підвищити продуктивність праці розробників, проваджуючи SQL у Java [21].

Jooq має як спростили простоту та безпеку експлуатації традиційних даних OPM, і зберігає гнучкість рідного SQL, який більше схожий на проміжний шар ORMS та JDBC. Для коду ферми, які люблять писати SQL, Jooq може повністю задовольнити свій контроль, може бути використаний для запису SQL з кодом Java (рис. 4.5).



```
Serhiy husak +1
@Configuration
@EnableTransactionManagement
public class JooqConfig {

    Serhiy husak +1
    @Bean
    public PlatformTransactionManager transactionManager(DataSource operationsDataSource){
        return new DataSourceTransactionManager(operationsDataSource);
    }

    Serhiy husak
    @Bean
    public ConnectionProvider connectionProvider(DataSource operationsDataSource){
        return new DataSourceConnectionProvider(new TransactionAwareDataSourceProxy(operationsDataSource));
    }

    Serhiy husak +1
    @Bean
    public DSLContext dslContext(DataSource operationsDataSource,
                                 SpringTransactionProvider transactionProvider,
                                 ConnectionProvider connectionProvider) {
        org.jooq.Configuration configuration =  new DefaultConfiguration()
                .derive(operationsDataSource)
                .derive(transactionProvider)
                .derive(connectionProvider)
                .derive(SQLDialect.POSTGRES);
        return DSL.using(configuration);
    }
}
```

Рисунок 4.5 – Конфігурація jooq для роботи з базою даних.

Окрім підключення до бази даних JOOQ створює власні файли які він використовую для написання запитів(рис 4.6). Конфігурація, яку ми створили під час цього повідомлення у блозі, забезпечує створення наступних класів:

Класи, створені в пакеті com.example.test.jooq.tables, містять метадані бази даних. jOOQ називає ці класи «глобальними» артефактами .

Класи у пакеті com.example.test.jooq.tables є класами таблиці, які описують структури таблиць бази даних. Можемо використовувати цей клас для запису запитів бази даних до даних, що зберігаються в таблиці бази даних todos .

Класи у пакеті com.example.test.jooq.tables.records класами записів, які містять інформацію про рядки таблиці. Запити до бази даних, які отримують дані з таблиці бази даних todos, повертають об'єкти Record (якщо ми вирішимо це зробити).

```
<plugin>
    <groupId>org.jooq</groupId>
    <artifactId>jooq-codegen-maven</artifactId>
    <version>3.15.3</version>
    <executions>
        <execution>
            <phase>generate-sources</phase>
            <goals>
                <goal>generate</goal>
            </goals>
            <configuration>
                <jdbc>
                    <driver>org.postgresql.Driver</driver>
                    <url>jdbc:postgresql://localhost:5432/OSMgis</url>
                    <!--suppress UnresolvedMavenProperty -->
                    <user>${DB_USERNAME}</user>
                    <!--suppress UnresolvedMavenProperty -->
                    <password>${DB_PASSWORD}</password>
                </jdbc>
                <generator>
                    <database>
                        <includes>city|users|segments|ways|parking_ways|restriction_time</includes>
                    </database>
                    <target>
                        <packageName>com.example.test.jooq</packageName>
                        <directory>src/main/java</directory>
                    </target>
                </generator>
            </configuration>
        </execution>
```

Рисунок 4.6- Конфігурація для генерування файлів.

Також для користувачів було додано авторизацію яка була виконана за допомогою Spring Security. Зазвичай включення будь-якого стартера в POM-файл нічого не дає: щоб щось запрограмувати, треба написати додатковий код. У випадку Spring Security генерує пароль:

При додаванні залежності spring-boot-starter-security до POM-файлу Spring Boot автоматично активує базову конфігурацію безпеки, яка включає кілька функцій за замовчуванням:

Автоматично створений користувач:

- Користувач з ім'ям user.
- Пароль генерується автоматично при кожному запуску програми і відображається у консолі.

Форма автентифікації:

- Веб-форма для входу (Form-based автентифікація) автоматично додається.

- Після введення ім’я та пароля виконується перевірка їхньої валідності.

Обмеження доступу:

- Усі URL-адреси програми недоступні, поки користувач не ввійде в систему за допомогою згенерованих облікових даних.

Функція виходу (logout):

- Автоматично створюється сторінка для виходу, доступна за URL /logout.

AuthenticationManager:

- Автентифікація обробляється компонентом AuthenticationManager, який Spring Security створює автоматично.
- Для зміни поведінки автентифікації необхідно перевизначити метод configure(AuthenticationManagerBuilder auth) у класі, що наслідує WebSecurityConfigurerAdapter. Через цей метод можна налаштувати AuthenticationManager відповідно до потреб.

Кастомізація AuthenticationManager:

- Ви можете уточнити, як AuthenticationManager отримує збережені дані користувача та порівнює їх із введеними. Це робиться через реалізацію специфічних налаштувань.

Таким чином, навіть базове додавання Spring Security забезпечує мінімальний рівень захисту для програми, який можна поступово адаптувати до вимог вашого проекту[].

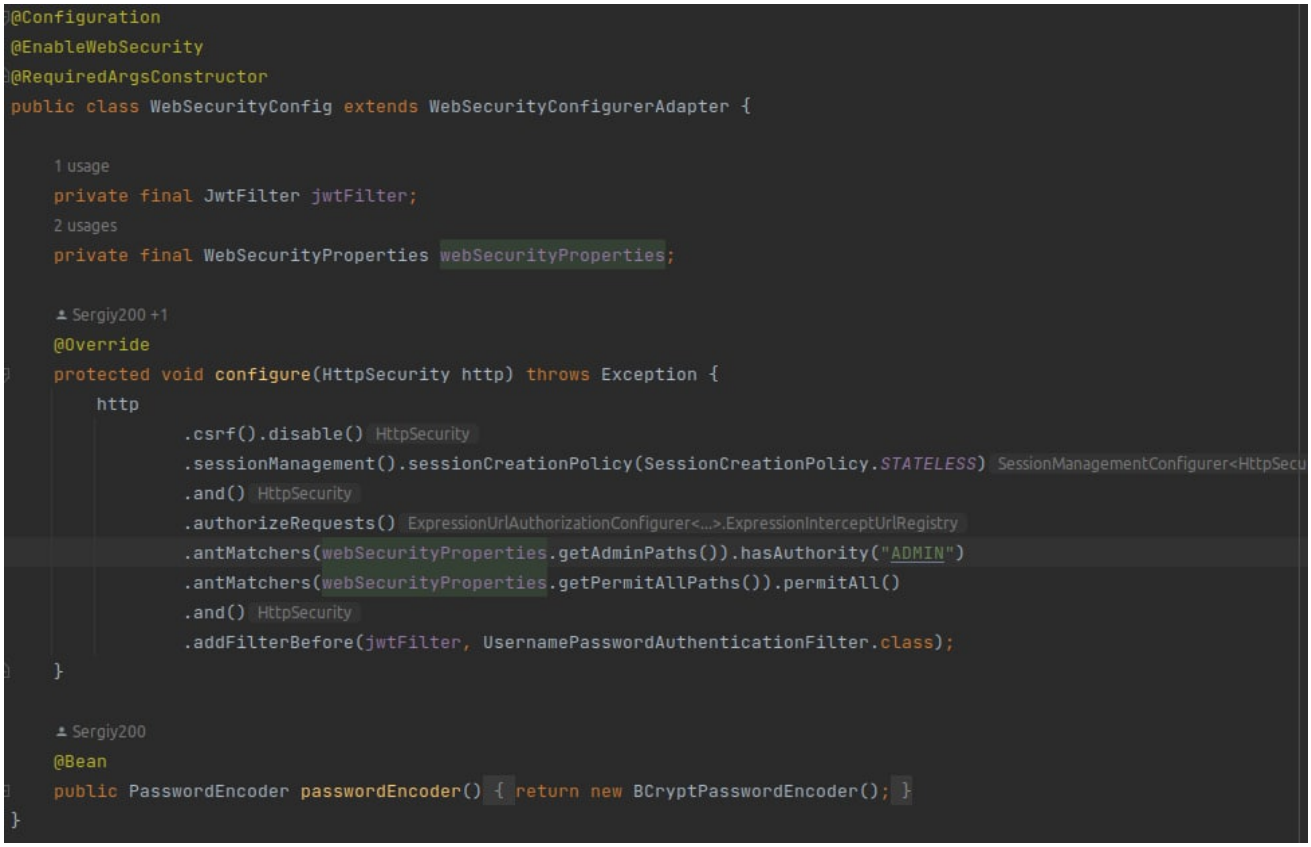


Рисунок 4.7 – Налаштування автентифікації для користувача.

Для зручнішої роботи з базою даних було вирішено також підключити Liquibase

Liquibase – це система управління міграціями бази даних. Це друга стаття про Liquibase, що на цей раз містить поради «бойового» використання системи. Для отримання базових відомостей підійде перша стаття-переклад «Управління міграціями БД з Liquibase»

Як і багато інструментів, які служать для полегшення життя розробників програмного забезпечення, Liquibase має «зворотний бік медалі», з яким доводиться рано чи пізно зіткнутися. Завжди можна легко сказати, які зміни схеми даних відбулися під час переходу з версії на версію програми. Підтримується акуратність у папках, що не дозволяє розслаблятися (рис.4.8).

```
<changeSet author="serhiy" id="users">
    <createTable tableName="users">
        <column autoIncrement="true" name="id" type="INTEGER">
            <constraints nullable="false" primaryKey="true" primaryKeyName="users_pk"/>
        </column>
        <column name="user_name" type="VARCHAR"/>
        <column name="login" type="VARCHAR"/>
        <column name="password" type="VARCHAR"/>
        <column name="user_role" type="VARCHAR"/>
    </createTable>
</changeSet>
<changeSet author="serhiy" id="city">
    <createTable tableName="city">
        <column autoIncrement="true" name="id" type="INTEGER">
            <constraints nullable="false" primaryKey="true" primaryKeyName="city_pk"/>
        </column>
        <column name="name_city" type="VARCHAR"/>
        <column name="prefix_city" type="VARCHAR"/>
        <column name="left_latitude" type="FLOAT8"/>
        <column name="left_longitude" type="FLOAT8"/>
        <column name="right_latitude" type="FLOAT8"/>
        <column name="right_longitude" type="FLOAT8"/>
        <column name="region" type="GEOMETRY"/>
    </createTable>
</changeSet>
<changeSet id="setUnique" author="serhiy">
    <addUniqueConstraint tableName="users" columnNames="login"/>
</changeSet>
```

Рисунок 4.8 – Приклад скрипта для Liquibase

Mapbox надає потужні механізми маршрутизації, точний час подорожі з урахуванням заторів та інтуїтивно зрозумілі покрокові вказівки, які допоможуть вам створити привабливу навігацію.

Mapbox.js є асинхронним – коли ви створюєте такий шар, як, шар не відразу знає, які плитки завантажити та інформацію про його присвоєння. Натомість він завантажує інформацію у виклик API.

Для більшості речей, які ви напишете, це не проблема, оскільки Mapbox.js добре справляється з цими оновленнями на льоту. Якщо ви пишете код, який повинен знати, коли шари та інші динамічно завантажуванні об’єкти готові, ви можете використовувати подію, щоб прослухати їх стан готовності (рис. 4.9).

```
function getWays($http, linesId) {
    $http.get('/segments/way', {
        params: {
            leftLatitude: firstPoint.lat,
            leftLongitude: firstPoint.lng,
            rightLatitude: secondPoint.lat,
            rightLongitude: secondPoint.lng
        }
    }).then(
        function successCallback(response) {
            let paint = {
                'line-width': 3,
                'line-color': '#25831c',
            };
            let lineLayer = new Layer(linesId, type: 'line', linesId, paint);
            drawLines(lineLayer, response.data);
        },
        function errorCallback(response) {
            console.log(response);
        });
}
```

Рисунок 4.9 – Приклад додавання нового леєра на карту

На сьогоднішній день є дуже велика кількість різноманітних технологій та фреймворків, проте вибір зупинився на Spring Boot.

Spring Boot — це відкритий кодовий фреймворк на основі Java, який використовується для створення мікросервісів. Використовується Pivotal Team для створення автономних і готових до виробництва Spring додатків. У цьому розділі ви дізнаєтеся про Spring Boot і основні ідеї.

Micro Service — це архітектура, яка дозволяє розробникам створювати та запускати сервіси самостійно. Кожна служба має свій власний процес, і ця полегшена модель підтримує бізнес-додатки.

Переваги:

- Легко розгортати.
- Просто масштабувати.
- Сумісний з контейнерами.
- Мінімальна комплектація.

Spring Boot надає хорошу платформу Java-розробникам для створення автономної версії Spring-програми високого рівня продуктивності, яку можна запустити за лічені хвилини. Ви можете почати роботу з невеликими конфігураціями, не налаштовуючи конфігурацію Spring повністю.

Spring Boot надає ефективну платформу для Java-розробників, що дозволяє створювати автономні програми Spring продуктивного рівня, які легко запускати. Завдяки мінімальним налаштуванням конфігурації, розробники можуть швидко розпочати роботу без необхідності глибокої інтеграції та налаштування Spring.

Переваги Spring Boot

- Спрощує розробку програм Spring.
- Підвищує продуктивність.
- Зменшує час розробки.

Основні цілі Spring Boot

- Усунення необхідності складної XML-конфігурації.
- Спрощення створення готових до використання додатків Spring.
- Зменшення часу розробки з можливістю швидкого запуску програми.
- Забезпечення простого старту для нових проектів.

Ключові функції та переваги Spring Boot

- Гнучке налаштування Java Beans, XML-конфігурацій і баз даних.
- Потужна пакетна обробка та управління REST-API кінцевими точками.
- Автоматичне налаштування без потреби ручної конфігурації.
- Використання анотацій для побудови додатків Spring.
- Спрощене управління залежностями.
- Вбудований контейнер сервлетів для швидкого запуску.

Ці можливості роблять Spring Boot ідеальним вибором для швидкої та ефективної розробки сучасних Java-додатків.

Spring Boot автоматично налаштовує програму відповідно до залежностей, доданих до проекту, завдяки анотації @EnableAutoConfiguration. Наприклад, якщо у вашому клас-паті присутня MySQL, але ви не налаштували з’єднання з базою даних, Spring Boot автоматично створить конфігурацію для бази даних у пам’яті.

Точка входу програми — це клас із анотацією @SpringBootApplication, що містить основний метод. Spring Boot також автоматично сканує всі компоненти в проекті за допомогою анотації @ComponentScan.

Для спрощення управління залежностями, особливо у великих проектах, Spring Boot пропонує набір стартових залежностей. Наприклад, для використання Spring і JPA для доступу до бази даних достатньо додати залежність spring-boot-starter-data-jpa.

Всі стартові пакети Spring Boot слідують єдиному шаблону іменування: spring-boot-starter-*, де * вказує на тип функціональності, що забезпечується залежністю.

Приклади:

Для кращого розуміння перегляньте наведені нижче початкові елементи Spring Boot

Залежність Spring Boot Starter Security використовується для Spring Security (рис. 4.10).

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

Рисунок 4.10 – Підключення залежності для Spring Security

Веб-залежність Spring Boot Starter використовується для написання REST-API (рис. 4.11).

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Рисунок 4.11 – Підключення залежності для написання REST-API

Залежність Spring Boot Starter Thyme Leaf використовується для створення веб-програми(рис. 4.12).

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

Рисунок 4.12 – Підключення залежності для створення веб-програми

Залежність Spring Boot Starter Test використовується для написання тестів(рис.4.13).

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-test</artifactId>
</dependency>
```

Рисунок 4.13 – Підключення залежності для написання тестів

Додаток Spring Boot.

Точкою входу Spring Boot Application є клас, що містить анотацію @SpringBootApplication. Цей клас повинен мати основний метод для запуску програми Spring Boot. Анотація @SpringBootApplication включає автоконфігурацію, сканування компонентів і конфігурацію завантаження Spring (рис. 4.14).

Рисунок 4.14 – Точка входу програми.

Якщо ви додали анотацію @SpringBootApplication до класу, вам не потрібно додавати анотації @EnableAutoConfiguration, @ComponentScan і @SpringBootConfiguration. Анотація @SpringBootApplication включає всі інші анотації.

Spring Boot Application зазвичай побудований на основі архітектури MVC. Цей шаблон проектування забезпечує розділення програми на три основні компоненти: Модель, Подання та Контролер. Такий підхід дозволяє незалежно змінювати кожен із компонентів, оскільки дані, інтерфейс користувача та керуюча логіка функціонують окремо один від одного.

Модель містить дані програми. Зазвичай включає в себе POJO-класи (Plain Old Java Objects) - прості старі Java-об'єкти або по-іншому - біни.

Подання або вигляд. Використовується для візуалізації та відображення даних програми за допомогою інтерфейсу користувача. Відповідає за те, як виглядатимуть дані моделі у браузері користувача.

Контролер потрібен для обробки запитів користувача і виклику серверних служб. Він структурує запит, створює відповідну модель для її подальшого відображення в браузері.

Як ми вже з вами розібралися, Spring MVC Framework логічно побудований на основі фронт-контролера DispatcherServlet, що обробляє всі HTTP-запити та відповіді. Щоб краще зрозуміти як він працює, подивимося, як усе влаштовано зсередини(рис. 4.15).

:

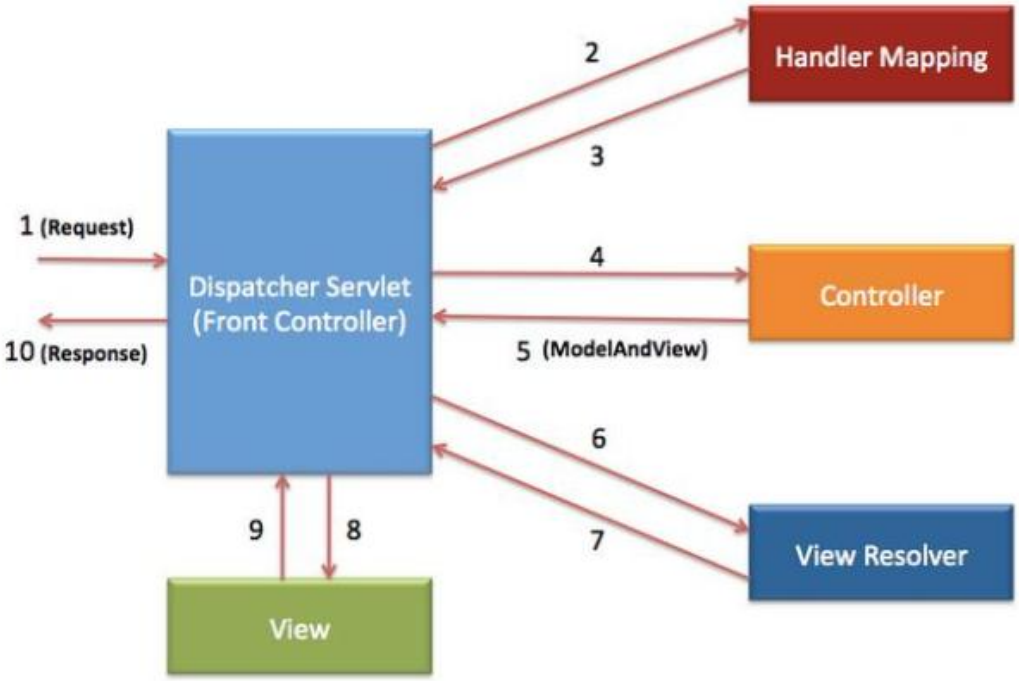


Рисунок 4.15 – Концепція роботи MVC

4.2 Архітектура програмного забезпечення веб-системи

Архітектура нашого сервісу базується на типовій моделі MVC. Наш рівень клієнта (подання) буде написаний на Javascript, Javascript, HTML. Користувач буде взаємодіяти з цим рівнем архітектури, коли він отримає доступ до функцій нашого додатку. Перш за все, потрібно створити чітку структуру папок, файлів, імпортів та експортів. На рисунку 4.16 зображено файлову структуру яку використовує веб-сервіс.

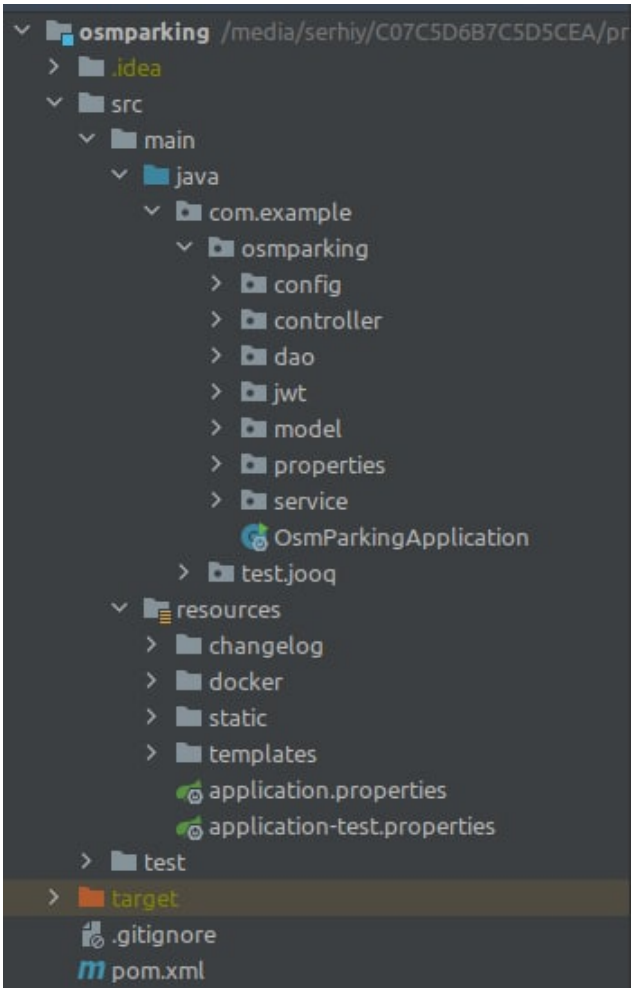


Рисунок 4.16 – Файлова структура клієнтської частини веб-сервісу

На рисунку 4.17 зображена UML Діаграма розгортання системи для клієнт-серверного додатку з ідентифікації паркувальних місць. Система організована за принципом розділення функціональних рівнів, що сприяє її масштабованості, гнучкості та підтримуваності. Нижче наведений опис кожного рівня:

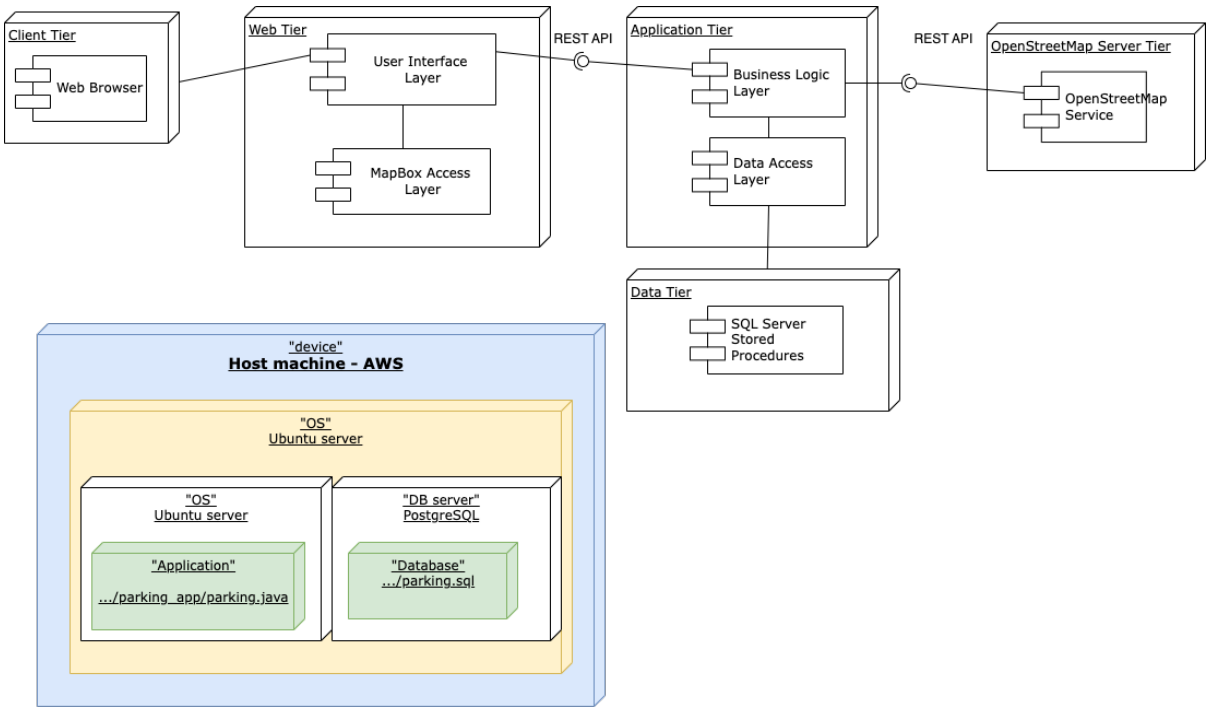


Рисунок 4.17 – UML Діаграма розгортання

Отже, після встановлення всіх залежностей веб-сервісу та налаштування середовища розробки, подальший прогрес над веб-сервісом не повинен вимагати занадто багато витрачених ресурсів.

Для реалізації системи ідентифікації місць паркування та управління даними про заборони зупинки, паркування і обмеження в часі було розроблено UML-діаграму класів. Вона відображає основні класи системи, їх атрибути та зв’язки, необхідні для організації взаємодії між компонентами системи(рис. 4.18-4.22).

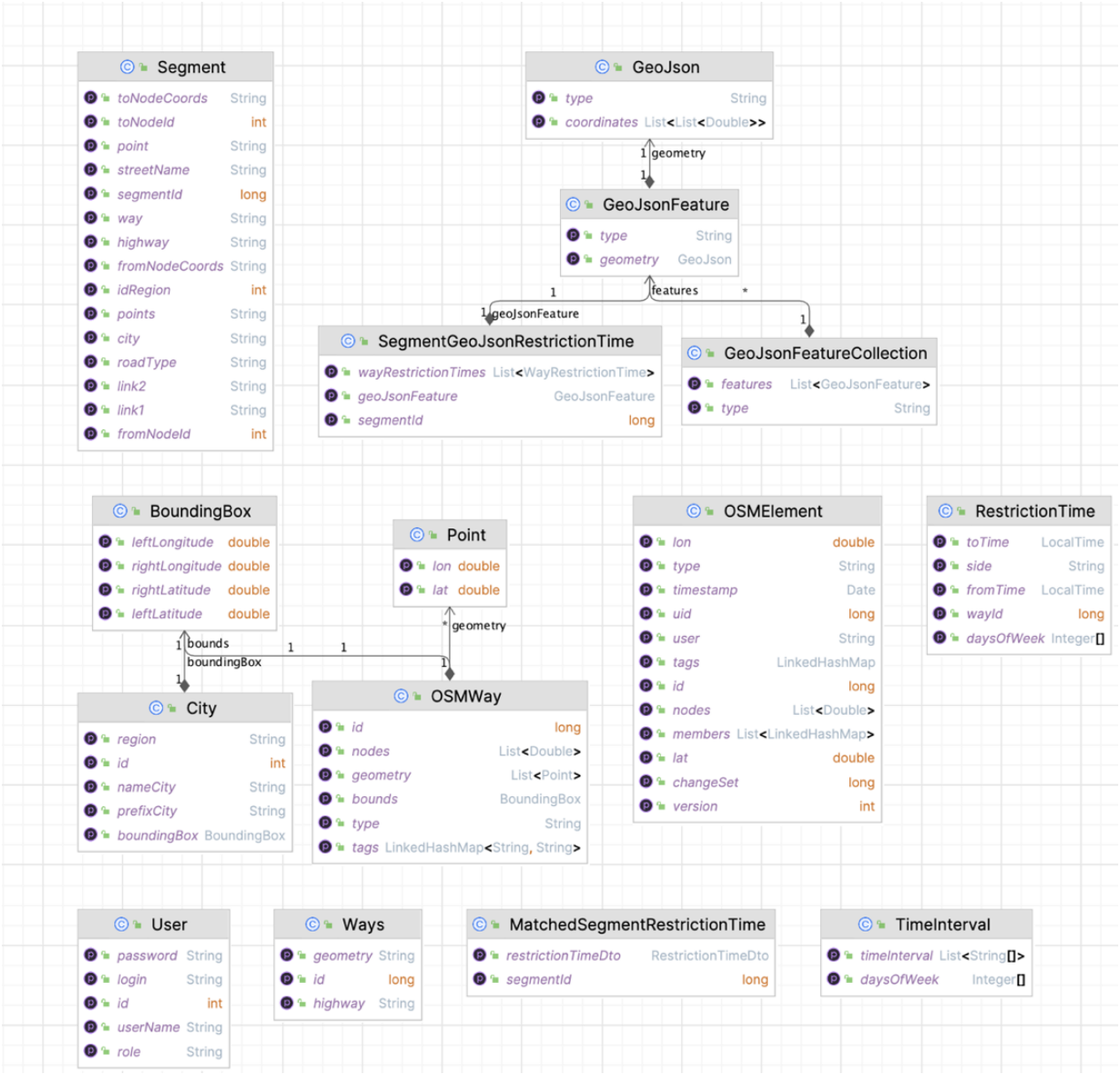


Рисунок 4.18 – UML діаграма класів об’єктів

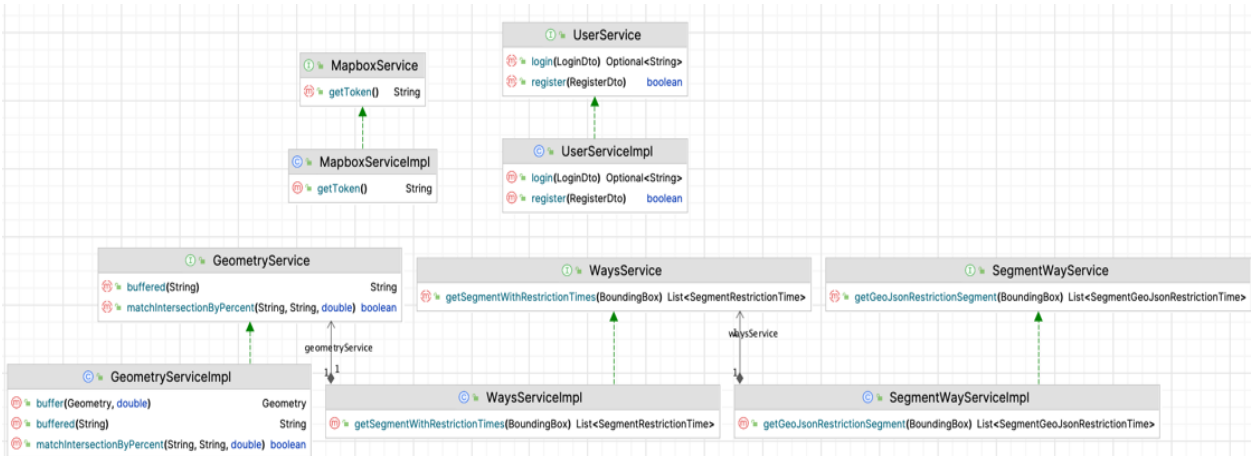


Рисунок 4.19 – UML діаграма класів частина 1

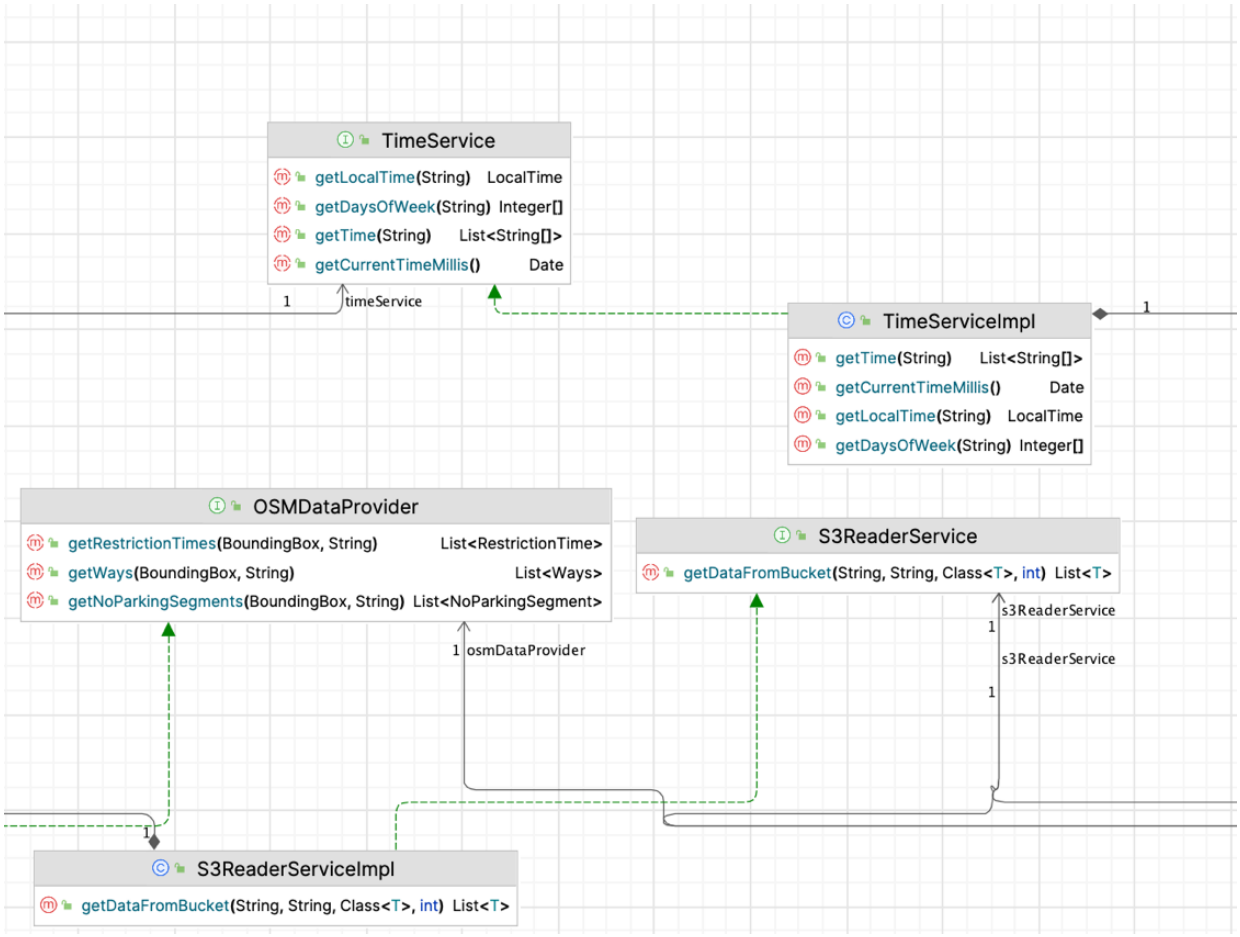


Рисунок 4.20 – UML діаграма класів частина 2

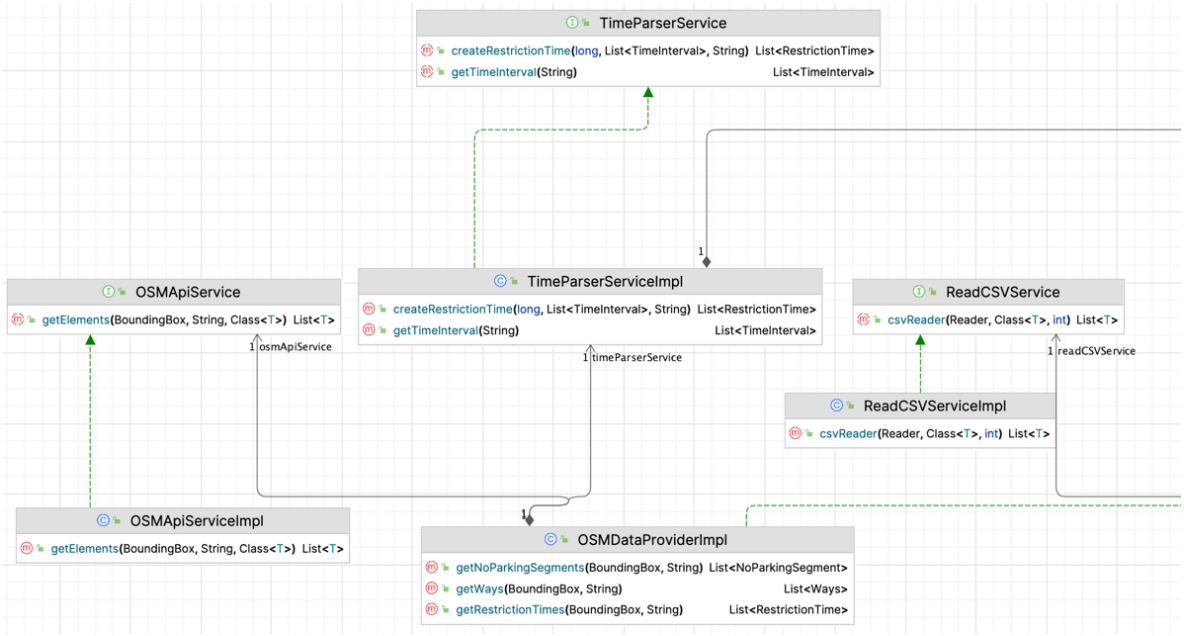


Рисунок 4.21 – UML діаграма класів частина 3

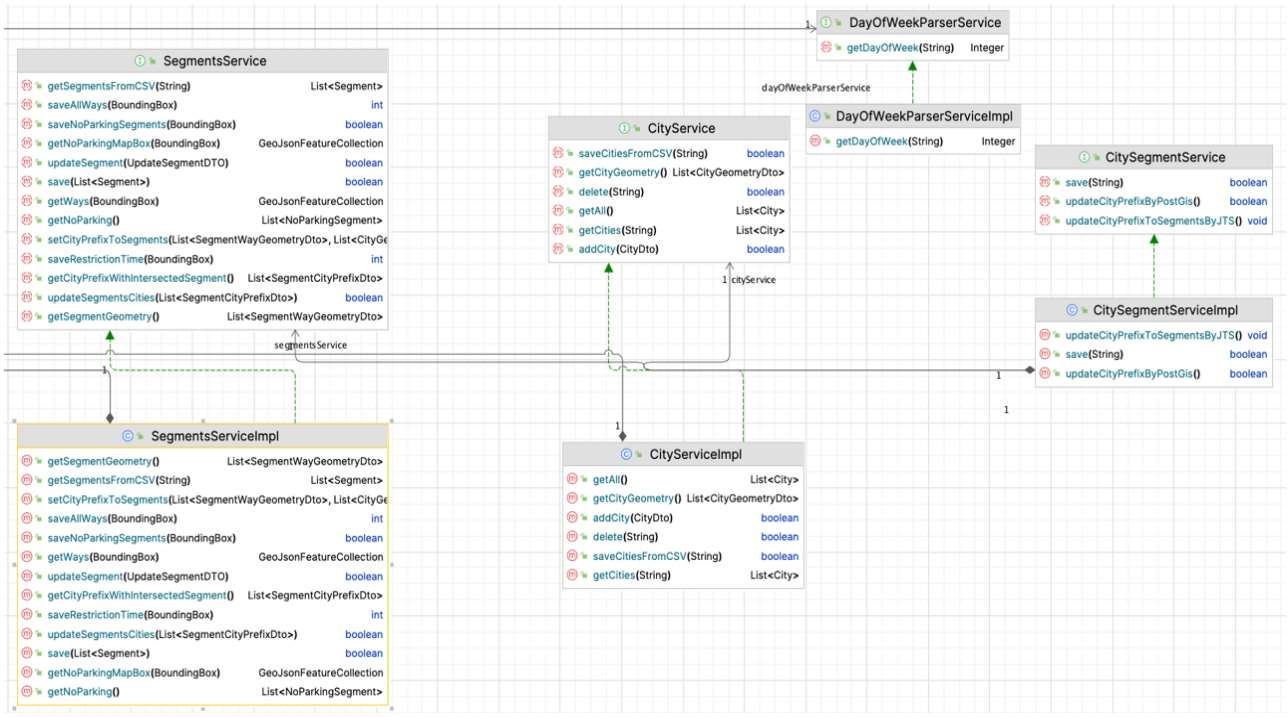


Рисунок 4.22 – UML діаграма класів частина 4

4.3 Програмна реалізація веб-системи

Ознайомлення та аналіз з даною предметною областю здійснювалось поетапно – від огляду набору даних до програмної реалізації.

На всіх етапах підготовки програмного модуля до реалізації було проведено аналіз з метою зменшення можливості виникнення помилок та незапланованих сценаріїв роботи в майбутньому.

Проаналізувавши виконану роботу, можна визначити задачі подальшого дослідження та вдосконалення, а саме:

1. Розроблення авторизацію у веб систему для надання певним користувачам доступ для оновлення карти (рис. 4.18)

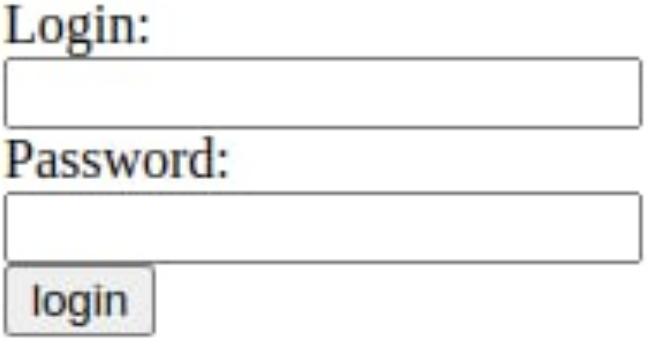


Рисунок 4.18 – Форма яка використовується для авторизації користувача

2. Підключено та відображено інтерактивну карту з бібліотеки MapBox, яку можна приховати якщо це необхідно (рис.4.19).

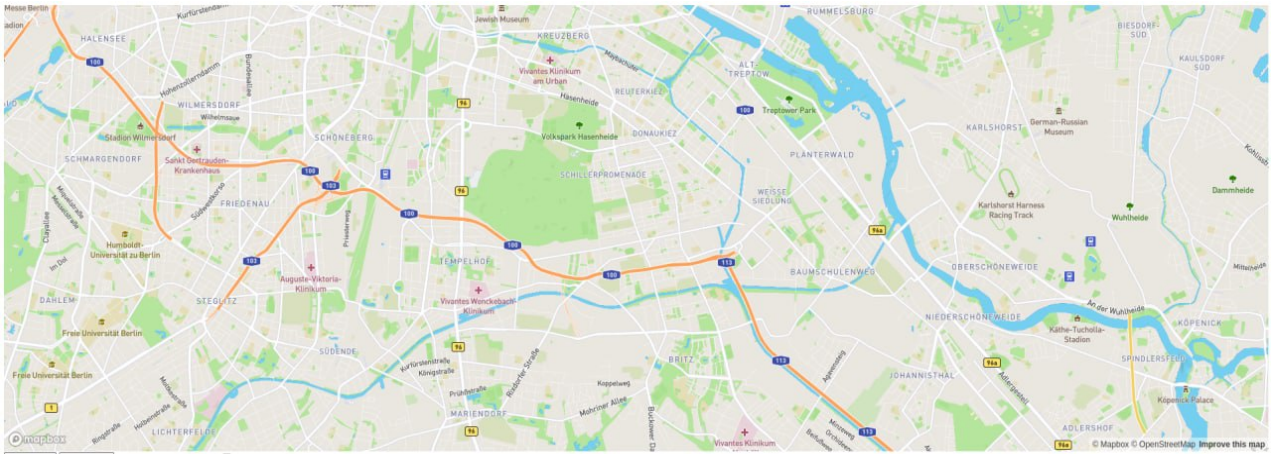


Рисунок 4.19 – Інтерактивна карта

3. Додано можливість вибору області для відображення даних на карті (рис. 4.20).

Рисунок 4.20 – Виділена область для відображення даних.

4. Відображення місць для паркування у вінницькій області(рис 4.21)

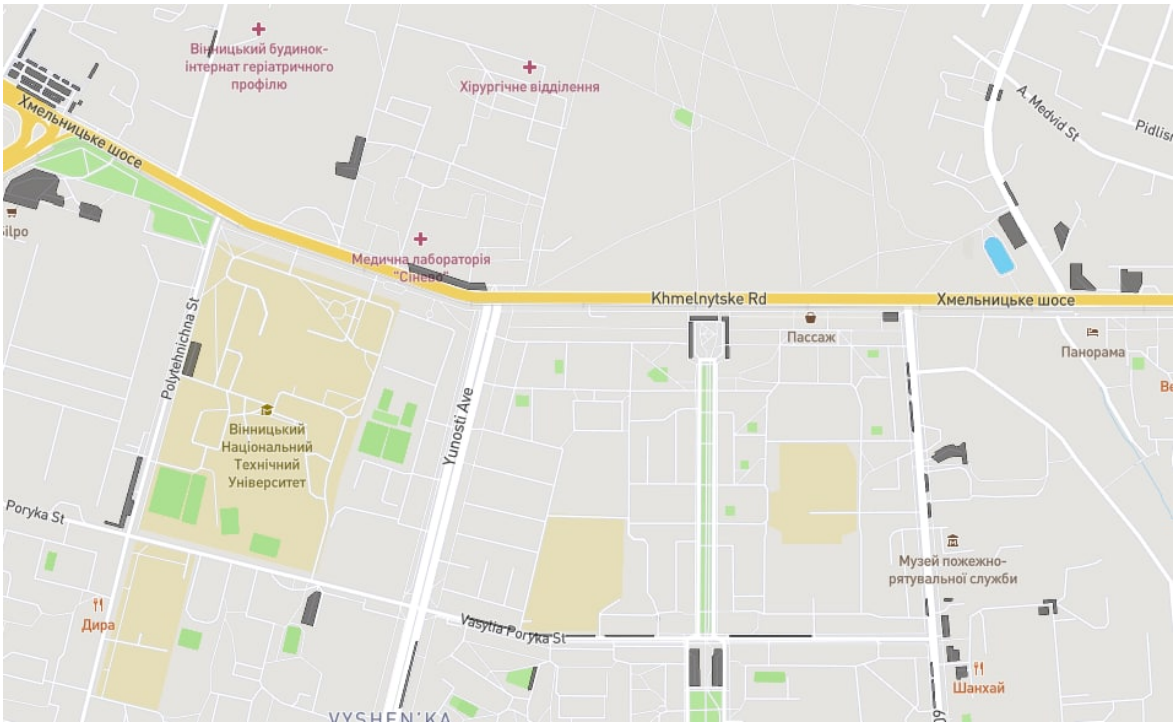


Рисунок 4.21 – Відображення місць для паркування у Вінниці.

5. Відображення всіх доріг на карті (рис. 4.21).

Рисунок 4.1 – Відображення всіх доріг у вибраній області

6. Відображення червоним кольором частин дороги де не можна паркуватись через забороняючий знак (рис. 4.22).

Рисунок 4.22 – Відображення доріг на карті із забороненим місцем паркування.

Усі запити із клієнтської частини обробляються за допомогою такого концепту як роутинг та концепту передачі даних REST API.

REST API – це прикладний програмний інтерфейс (API), який використовує HTTP-запити для отримання, вилучення, розміщення і видалення даних. Абревіатура REST в контексті API розшифровується як «передача стану уявлення» (Representational State Transfer).

Основна ідеї REST API полягає в поділі різних при зверненні до одного і того ж URL за допомогою HTTP методів, основні з яких:

– GET – використовується для отримання даних;

– POST – використовується для створення нового запису;

– PUT – використовується для оновлення вже існуючої записи

– PATCH – використовується для оновлення, але тільки тоді, коли змінюється ідентифікатор запису;

– DELETE – використовується для видалення запису [9].

На рисунках 4.23 та 4.24 зображено обробку запиту із клієнтської частини та його зворотня відповідь.

```java
@PostMapping("/city")
public ResponseEntity<Object> addCity(@RequestBody CityDto cityDto) {
    return Optional.of(cityService.addCity(cityDto)) Optional<Boolean>
            .filter(data -> data)
            .map(data -> ResponseEntity.status(HttpStatus.CREATED).build()) Optional<ResponseEntity<Object>>
            .orElse(ResponseEntity.status(HttpStatus.BAD_REQUEST).build());
}
```

Рисунок 4.23 – Фрагмент обробки POST-запиту веб-системи


```
"type": "FeatureCollection",
"features": [
    {
        "geometry": {
            "type": "LineString",
            "coordinates": [
                [
                    28.4260128,
                    49.2277116
                ],
                [
                    28.4266778,
                    49.2277435
                ],
                [
                    28.4267109,
                    49.2274339
                ],
                [
                    28.427681,
                    49.2274753
                ],
                [
                    28.427768,
                    49.2267228
                ],
                [
                    28.4271892,
                    49.226695
                ],
                [
                    28.4271495,
                    49.227048
                ],
```


Рисунок 4.24 – Відповідь обробки POST-запиту

Після того як, серверна частина була реалізована, щоб запустити у безперебійну роботу сервер достатньо прописати в командній стрічці npm run dev. Результат роботи сервера зображено на рисунку 4.25.

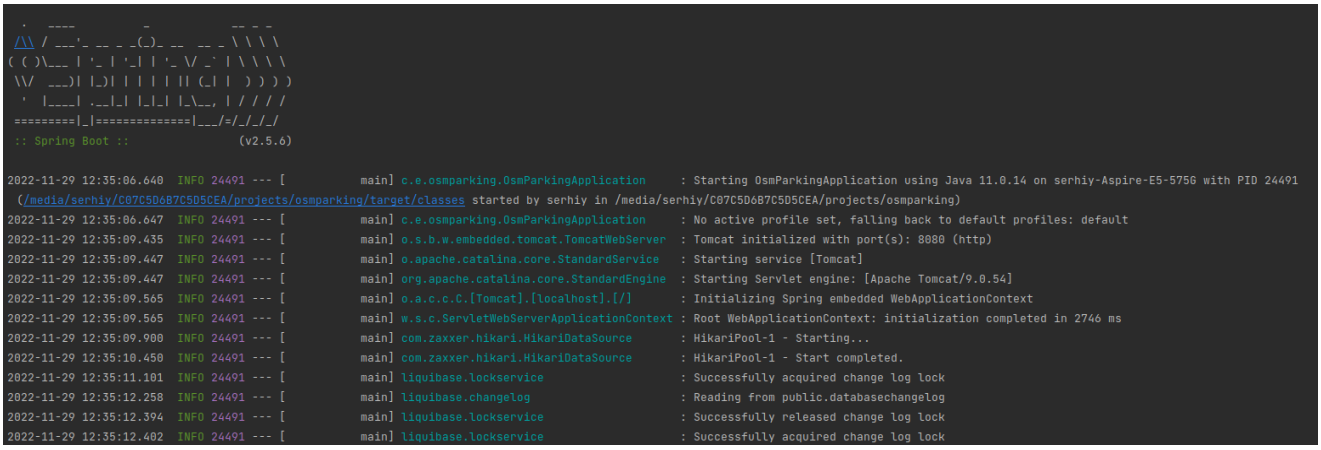


Рисунок 4.25 – Запуск сервера системи

Після того, як серверна частина була реалізована, саме час приступити до розробки інтерфейсу користувача. Як раніше було вже згадано в цьому, допоможе бібліотека javascript AngularJs.

Перш за все, щоб згенерувати базові залежності проекту потрібно скористатися інструментом AngularJs виклику даних з серверної частини

На рисунку 4.26 зображени приклад виклику даних з серверної частини.

```
function getParkings($http, polygonId) {
    $http.get('/parking', {
        params: {
            leftLatitude: firstPoint.lat,
            leftLongitude: firstPoint.lng,
            rightLatitude: secondPoint.lat,
            rightLongitude: secondPoint.lng
        }
    }).then(
        function successCallback(response) {
            console.log("succes");
            let paint = {
                'fill-color': '#0e0e0e',
                'fill-opacity': 0.5
            };
            let lineLayer = new Layer(polygonId, type: 'fill', polygonId, paint);
            drawLines(lineLayer, response.data);
        },
        function errorCallback(response) {
            console.log("error");
            console.log(response);
        });
}
```

Рисунок 4.26 – Виклик даних з серверної частини

AngularJs Router – одна з найпопулярніших платформ маршрутизації. Бібліотека розроблена з інтуїтивно зрозумілими компонентами, що дозволяє створити декларативну систему маршрутизації для вашого додатка. Це означає, що ви можете точно заявити, який із ваших компонентів має певний маршрут. За допомогою декларативної маршрутизації ви можете створювати інтуїтивно зрозумілі маршрути, зручні для читання, полегшуючи управління архітектурою вашого додатка.

На рисунку 4.27 відображено налаштування роутингу між компонентами інтерфейсу.

```
wayManagement.factory('loginInterceptor', ['$q', '$window', function ($q, $window) {
    return {
        responseError: function (response) {
            console.log('Failed with', response.status, 'status');
            if (response.status === 403 || response.status === 401) {
                $window.location.replace('#!login');
            }
            return $q.reject(response);
        }
    }
}])

wayManagement.config(function ($httpProvider) {
    $httpProvider.interceptors.push('loginInterceptor')
})

wayManagement.config(function ($routeProvider) {
    $routeProvider.when("/", {
        templateUrl: 'views/mapbox.html',
        controller: 'way-controller',
    })
    $routeProvider.when("/login", {
        templateUrl: 'views/login.html',
        controller: 'way-controller',
    })
})
```

Рисунок 4.27 – Налаштування роутингу веб-системи

4.5 Висновки

В даному розділі було проаналізовано передові технологій, які б задовольнили весь функціонал, який був описаний у завданні. Дані технології чітко поєднуються між собою, створюючи потужний та швидкий веб-додаток, який забезпечує користувача всіма необхідними ресурсами. Також, було розроблено архітектуру програмного забезпечення, де було описано всі найважливіші аспекти розробки веб-застосунку. Розроблено веб-систему ідентифікації місць для паркування.

ВИСНОВКИ

1) Проведено аналіз предметної галузі, охарактеризовано, що таке пошук місць для паркування, визначено, що для виконання даного завдання необхідно працювати із географічними даними. Здійснено огляд відомих аналогів і більшість з них працюють чудово та мають можливість оплати паркування. Проаналізовано, актуальність даної системи у порівняні з існуючими аналогами.

2) Виконано формування алгоретмічних та програмних вимог для веб-сервісу та було побудована концепція кінцевого продукту. Було проведено відбір оптимальних інформаційних технологій для серверної частини було обрано мову програмування Java та фреймворк для створення веб серверу із використанням патерну MVC що підрозуміває під собою умовний поділ додатку на три частини. Для користувацької частини додатку було використано мову програмування JavaScript та бібліотеку AngularJs яка використовується для створення одно сторінкових додатків за допомогою патерну MVC. OpenStreetMap розглянуто для отримання даних по місцезнаходженню місця для паркування та їх зображення на карті.

3) Розглянуто можливі види отримання даних з OSM, розглянуто мову QL для написання запитів на Overpass Turbo для отримання та обробки даних з джерела OpenStreetMap. Розроблено базу даних веб-системи ідентифікації місць паркування.

4) Проаналізовано передові технологій, які б задовольнили весь функціонал. Дані технології чітко поєднуються між собою, створюючи потужний та швидкий веб-додаток, який забезпечує користувача всіма необхідними ресурсами.

5) Було розроблено архітектуру програмного забезпечення та реалізовано всі найважливіші аспекти розробки веб-застосунку. Розроблено веб-систему ідентифікації місць для паркування, яка надає користувачеві можливість авторизуватись на сайті, відкрити карту, виділити на карті область у вигляді BoundingBox та обрати які дані відобразити у даній області, це можуть бути місця для паркування, частини дороги, де паркування заборонене. Також була реалізована можливість завантаження даних з OSM, а також даних з Amazon S3.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Крижановський Є. М., Гусак С.В. Інформаційна веб-система ідентифікації місць для паркування за даними сервісу OpenStreetMap. *Всеукраїнська науково-практична інтернет-конференція «Молодь в науці: дослідження, проблеми, перспективи» (Вінниця, 2022-2023 рр.)*. URL: https://conferences.vntu.edu.ua/index.php/mn/mn2023/paper/viewFile/16821/14039

2. Коцюбинський В.Ю., Гусак С.В. Розробка клієнт-серверної частини системи ідентифікації місць паркування паркувальних майданчиків. *Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації (2024)*. URL https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2024/paper/view/20818/17277

3. Штрафи за паркування. URL: https://vezha.ua/na-sajti-vinnytskoyi-miskrady-pochav-pratsyuvaty-servis-dlya-porushnykiv-pravyl-parkuvannya/

4. Mark Heckler, Spring Boot: Up and Running. 1st, , O'Reilly Media, USA, 2021 300 p.

5. Mobile-parking. URL: https://ktps.kyiv.ua/services/pay/mobile-parking

6. Роберт С. Мартін, Чиста архітектура, Фабула, Харків, 2019. 416 с.

7. Henrietta Dombrovska, Boris Novikov, Anna Bailliekova, PostgreSQL Query Optimization. 1st, Apress, NY, 2021. 319 p.

8. Web-орієнтована система моніторингу автостоянок. URL: http://dspace.wunu.edu.ua/bitstream/316497/18094/1/%D0%92%D1%96%D0%B2%D1%87%D0%B0%D1%80.pdf

9. Web-орієнтована розумна парковка. URL: http://elar.khmnu.edu.ua/jspui/bitstream/123456789/11951/1/%D0%94%D0%A0_%D0%9A%D0%BE%D0%B2%D0%B0%D0%BB%D0%B5%D0%BD%D0%BA%D0%BE%20%D0%92.%D0%92._%D0%9A%D0%862%D0%BC-20-1%20%281%29.pdf

10. Мова програмування. URL: https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F

11. Об’єктно-орієнтоване програмування. URL: https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%27%D1%94%D0%BA%D1%82%D0%BD%D0%BE%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F

12. Robert C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship, Upper Saddle River, NJ, 2019. 464 p.

13. Brett D. McLaughlin, Head First Object-Oriented Analysis and Design, O'Reilly Media, California, 2006. 236 c.

14. Josh Long Cloud Native Java: Designing Resilient Systems with Spring Boot, Spring Cloud, and Cloud Foundry 1st Edition, O'Reilly Media, California, 2020. 624 p.

15. Joshua Bloch, Effective Java 3rd Edition Addison-Wesley Professional, O'Reilly Media, California, 2020. 392 p.

16. Роберт С. Мартін, Чистий код, Фабула, Харків, 2019. 416 с.

17. Overpass API. URL: https://wiki.openstreetmap.org/wiki/Overpass_API

18. OpenStreetMap. URL: https://uk.wikipedia.org/wiki/OpenStreetMap

19. Написання запитів для отримання даних з OpenStreetMap. URL: https://overpass-turbo.eu/

20. Spring security. URL: https://sysout.uk/dobavlenie-spring-security-v-proekt-nastrojki-po-umolchaniyu/

21. Jooq spring. URL: https://coderlessons.com/articles/java/ispolzovanie-jooq-s-spring-generatsiia-koda

ДОДАТКИ

Додаток А
(обов'язковий)

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: «Розробка клієнт-серверної частини системи ідентифікації місць паркування паркувальних майданчиків.»
Тип роботи: магістерська кваліфікаційна робота
(кваліфікаційна робота, курсовий проект (робота), реферат, аналітичний огляд, інше (зазначити))
Підрозділ кафедра комп'ютерних систем управління
(кафедра, факультет (інститут), навчальна група)
Керівник Коцюбинський В.Ю., доцент кафедри АІІТ
(прізвище, ініціали, посада)

Показники звіту подібності

Turnitin	
Оригінальність	91%
Загальна схожість	9%

Аналіз звіту подібності (відмітити потрібне)

√ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Заявляю, що ознайомлений (-на) з повним звітом подібності, який був згенерований Системою щодо роботи (додається)

Автор _____ Сергій ГУСАК
(підпис) (прізвище, ініціали)

Опис прийнятого рішення

_____
_____
_____
_____

Особа, відповідальна за перевірку _____ Володимир ДУБОВОЙ
(підпис) (прізвище, ініціали)

Експерт _____ _____
(за потреби) (підпис) (прізвище, ініціали, посада)

Додаток Б
(обов'язковий)

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації

ЗАТВЕРДЖЕНО
Зав кафедри КСУ ВНТУ,
д.т.н., проф.
_____ В'ячеслав КОВТУН.
«14» 10 2024 р.

ТЕХНІЧНЕ ЗАВДАННЯ
на магістерську кваліфікаційну роботу
«Розробка клієнт-серверної частини системи ідентифікації місць паркування паркувальних майданчиків»
08-33.МКР.15.03.000 ТЗ

Студент групи ЗАКІТР-23м
_____ Сергій ГУСАК

Керівник к.т.н., доцент каф. АІІТ
_____ Володимир КОЦЮБИНСЬКИЙ

Вінниця 2024

1. Назва та галузь застосування

1.1 Назва – Розробка клієнт-серверної частини системи ідентифікації місць паркування паркувальних майданчиків.

1.2 Галузь застосування – транспорт та логістика

2. Підстава для проведення розробки.

Тема магістерської кваліфікаційної роботи затверджена наказом по ВНТУ від №310 від 17.09.2024р.

3. Мета та призначення розробки.

Метою магістерської кваліфікаційної роботи є зменшення часу для знаходження паркувальних мість що зменшить викиди CO2 в атмоферу.

4. Джерела розробки.

Магістерська кваліфікаційна робота виконується вперше. В ході проведення розробки повинні використовуватись такі документи:

- Системний аналіз та проектування ГІС. – Електронний навчальний посібник / Є. М. Крижановський, В.Б. Мокін, А.Р. Ящолт, Л.М. Скорина. – Вінниця : ВНТУ, 2015. – 127 с.

OpenStreetMap (OSM): відкриті географічні дані, які можна використовувати для визначення розташування вулиць, зон паркування, обмежень зупинки та паркування:

- Інструменти доступу до даних OSM: Overpass API, Mapbox API, Osmosis.

5. Вимоги до розробки.

5.1. Перелік головних функцій:

- збирання інформації з відкритих джерел
- відображення данних на карту;
- керування базою даних.

5.2. Основні технічні вимоги до розробки.

5.2.1. Вимоги до програмної платформи:

- Linux;
- Хмарне середовище Azure;
- IntelliJ IDEA.

5.2.2. Умови експлуатації системи:

- робота на мобільних та веб-додатках;
- можливість цілодобового функціонування системи;

- дані оновлюються і є актуальними.

6. Стадії та етапи розробки.

6.1 Пояснювальна записка:

1. Аналіз методів, принципів, підходів і засобів реалізації задачі автоматизації процесами в об’єкті управління відповідно до теми кваліфікаційної роботи. Постановка задач дослідження «03»09 2024р.
2. Удосконалення технології прийняття рішень при автоматизації об’єкту управління «_01_»_10_ 2024 р.
3. Визначення технічних характеристик системи. «_06_»_10_ 2024 р.
4. Розробка програмного забезпечення системи «_17_»_11_ 2024 р.

6.2 Графічні матеріали:

1. Розробка UML-діаграм системи «_24_»_10_ 2024 р.
2. Розробка моделі бази даних системи «_29_»_10_ 2024 р.
3. Тестування програмного забезпечення «_24_»_11_ 2024 р.

7. Порядок контролю і приймання.

7.1. Хід виконання роботи контролюється керівником роботи. Рубіжний контроль провести до «_20_»__11__ 2024 р.

7.2. Атестація МКР здійснюється на попередньому захисті. Попередній захист магістерської кваліфікаційної роботи провести до «1»12 2024 р.

7.3. Підсумкове рішення щодо оцінки якості виконання роботи приймається на засіданні ЕК. Захист магістерської кваліфікаційної роботи провести до

ДОДАТОК В

(вибірковий).

Лістинг програми

```
package com.example.osmparking;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class OsmParkingApplication {

    public static void main(String[] args) {
        SpringApplication.run(OsmParkingApplication.class, args);
    }
}
package com.example.osmparking.controller;

import com.example.osmparking.model.BoundingBox;
import com.example.osmparking.service.ParkingService;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequiredArgsConstructor
@RequestMapping("parking")
public class ParkingPlacesController {
    private final ParkingService;
```

```
    @PostMapping("osm")
    public ResponseEntity<Object> findParking(BoundingBox boundingBox) {
        return ResponseEntity.ok(parkingService.loadParking(boundingBox));
    }

    @GetMapping
    public ResponseEntity<Object> getParking(BoundingBox boundingBox) {
        return ResponseEntity.ok(parkingService.getParking(boundingBox));
    }
}
package com.example.osmparking.dao.impl;

import com.example.osmparking.dao.ParkingPlaceDao;
import com.example.osmparking.model.BoundingBox;
import com.example.osmparking.model.GeoJson;
import com.example.osmparking.model.ParkingPlace;
import com.fasterxml.jackson.databind.ObjectMapper;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.jooq.DSLContext;
import org.jooq.Query;
import org.jooq.Record1;
import org.springframework.stereotype.Repository;

import java.util.List;
import java.util.stream.Collectors;

import static com.example.osmparking.dao.fields.Fields.*;
import static com.example.test.jooq.public_.Tables.PARKING_PLACES;
import static com.example.test.jooq.public_.tables.Ways.WAYS;
```

```
import static java.text.MessageFormat.format;

@Slf4j
@Repository
@RequiredArgsConstructor
public class ParkingPlacesDaoImpl implements ParkingPlaceDao {

   private static final String PARKING_PLACE_GEO = "parking_place";
   private final DSLContext;
   private final ObjectMapper;

   @Override
   public void upsert(List<ParkingPlace> parkingPlaces) {
      List<Query> queries = buildQueries(parkingPlaces);
      dslContext.batch(queries).execute();
   }

   @Override
   public List<GeoJson> get(BoundingBox boundingBox) {
      return dslContext

.select(stGeoJson((PARKING_PLACES.PARKING)).as(PARKING_PLACE_GEO))
            .from(PARKING_PLACES)
            .where(stContains(getBboxField(boundingBox),
PARKING_PLACES.PARKING))
            .fetch(this::getGeoJson);
   }

   private GeoJson getGeoJson(Record1<String> geoJson) {
      try {
```

```
                return objectMapper.readValue(geoJson.get(PARKING_PLACE_GEO,
String.class), GeoJson.class);
            } catch (Exception e) {
                log.error("Error parsing result {}", geoJson);
                throw new RuntimeException(format("Error parsing result {0} {1}",
geoJson, e));
            }
        }

        private List<Query> buildQueries(List<ParkingPlace> parkingPlaces) {
            return parkingPlaces.stream()
                    .map(parkingPlace -> dslContext.insertInto(PARKING_PLACES,
PARKING_PLACES.ID, PARKING_PLACES.NAME,
PARKING_PLACES.PARKING)
                            .values(parkingPlace.getId(), parkingPlace.getName(),
parkingPlace.getGeometry())
                            .onConflict(PARKING_PLACES.ID)
                            .doUpdate()
                            .set(PARKING_PLACES.NAME, parkingPlace.getName())
                            .set(PARKING_PLACES.PARKING,
parkingPlace.getGeometry()))
                    .collect(Collectors.toUnmodifiableList());
        }
    }
    package com.example.osmparking.dao.fields;

    import com.example.osmparking.model.BoundingBox;
    import lombok.AccessLevel;
    import lombok.NoArgsConstructor;
    import org.jooq.Field;
    import org.jooq.QueryPart;
```

```
import static org.jooq.impl.DSL.field;

@NoArgsConstructor(access = AccessLevel.PRIVATE)
public class Fields {
   public static Field<?> stAsText(QueryPart... parts) {
      return field("st_asText({0})", parts);
   }

   public static Object setToGeometry(Object... bindings) {
      return field("{0}::geometry", bindings);
   }

   public static Field<Boolean> stIntersects(QueryPart... parts) {
      return field("ST_Intersects({0}, {1})", boolean.class, parts);
   }

   public static Field<Object> getBboxField(BoundingBox boundingBox) {
      return field("'BOX(" +
            boundingBox.getLeftLongitude() + " " + boundingBox.getLeftLatitude()
+ "," +
            boundingBox.getRightLongitude()  +  "  "  +
boundingBox.getRightLatitude() + ")'::box2d"
      );
   }

   public static Field<Boolean> stContains(QueryPart... parts) {
      return field("ST_Contains({0}, {1})", boolean.class, parts);
   }

   public static Field<String> stGeoJson(Object... geometry) {
```

```
        return field("ST_asGeoJson({0})", String.class, geometry);
    }

    public static Field<Object> getGeometrySRID(String geometry) {
        return field("st_geometryFromText({0},4326)", geometry);
    }

}
```

Додаток Г

(обов'язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

Розробка клієнт-серверної частини системи ідентифікації місць паркування паркувальних майданчиків

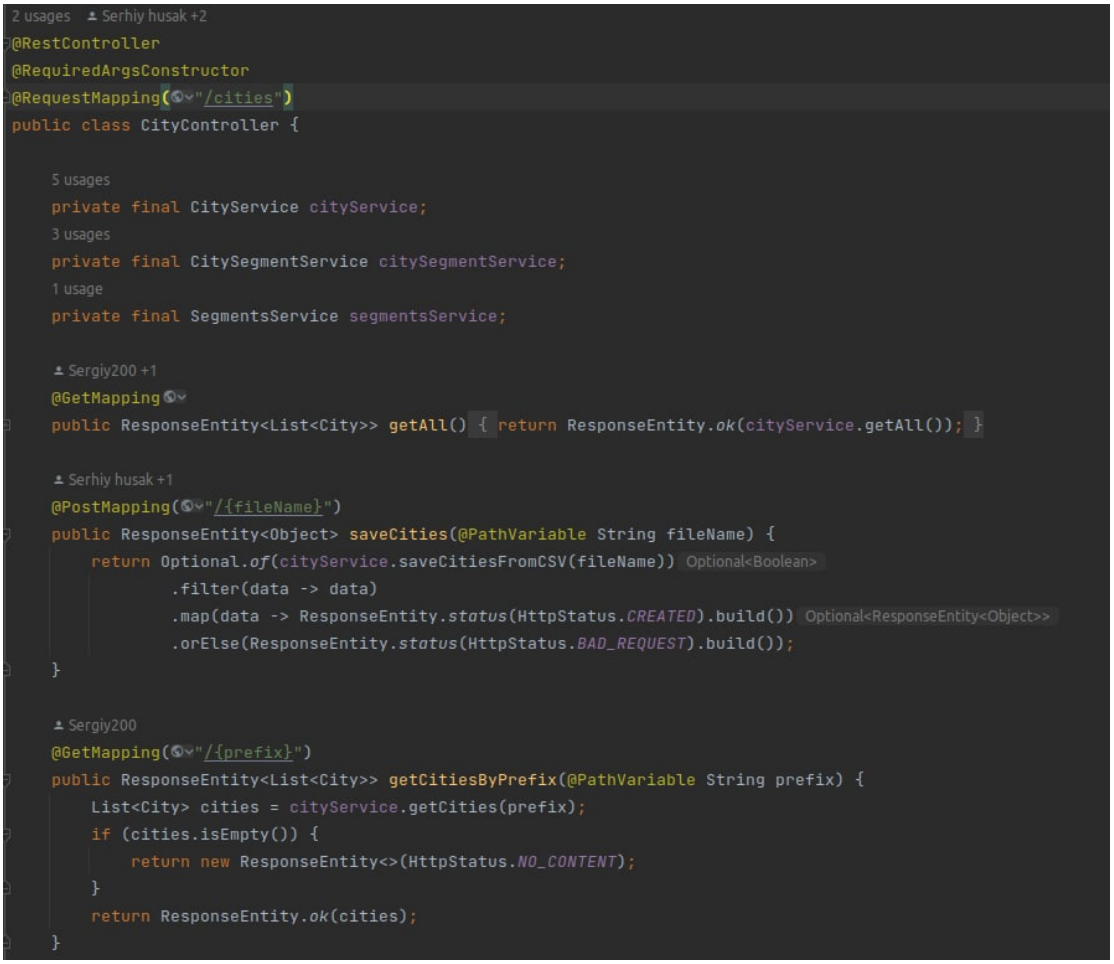
Студент групи. ЗАКІТР-23м

_____ Сергій ГУСАК

Керівник: д.т.н., доц. каф. АІІТ

_____ Володимир КОЦЮБИНСЬКИЙ

«___» _____ 2024 р.



```
2 usages  Serhiy husak +2
@RestController
@RequiredArgsConstructor
@RequestMapping("/cities")
public class CityController {

    5 usages
    private final CityService cityService;
    3 usages
    private final CitySegmentService citySegmentService;
    1 usage
    private final SegmentsService segmentsService;

    Sergiy200 +1
    @GetMapping
    public ResponseEntity<List<City>> getAll() { return ResponseEntity.ok(cityService.getAll()); }

    Serhiy husak +1
    @PostMapping("/{fileName}")
    public ResponseEntity<Object> saveCities(@PathVariable String fileName) {
        return Optional.of(cityService.saveCitiesFromCSV(fileName)) Optional<Boolean>
                .filter(data -> data)
                .map(data -> ResponseEntity.status(HttpStatus.CREATED).build()) Optional<ResponseEntity<Object>>
                .orElse(ResponseEntity.status(HttpStatus.BAD_REQUEST).build());
    }

    Sergiy200
    @GetMapping("/{prefix}")
    public ResponseEntity<List<City>> getCitiesByPrefix(@PathVariable String prefix) {
        List<City> cities = cityService.getCities(prefix);
        if (cities.isEmpty()) {
            return new ResponseEntity<>(HttpStatus.NO_CONTENT);
        }
        return ResponseEntity.ok(cities);
    }
```

Рисунок Г.1 – REST-API веб-сервісу

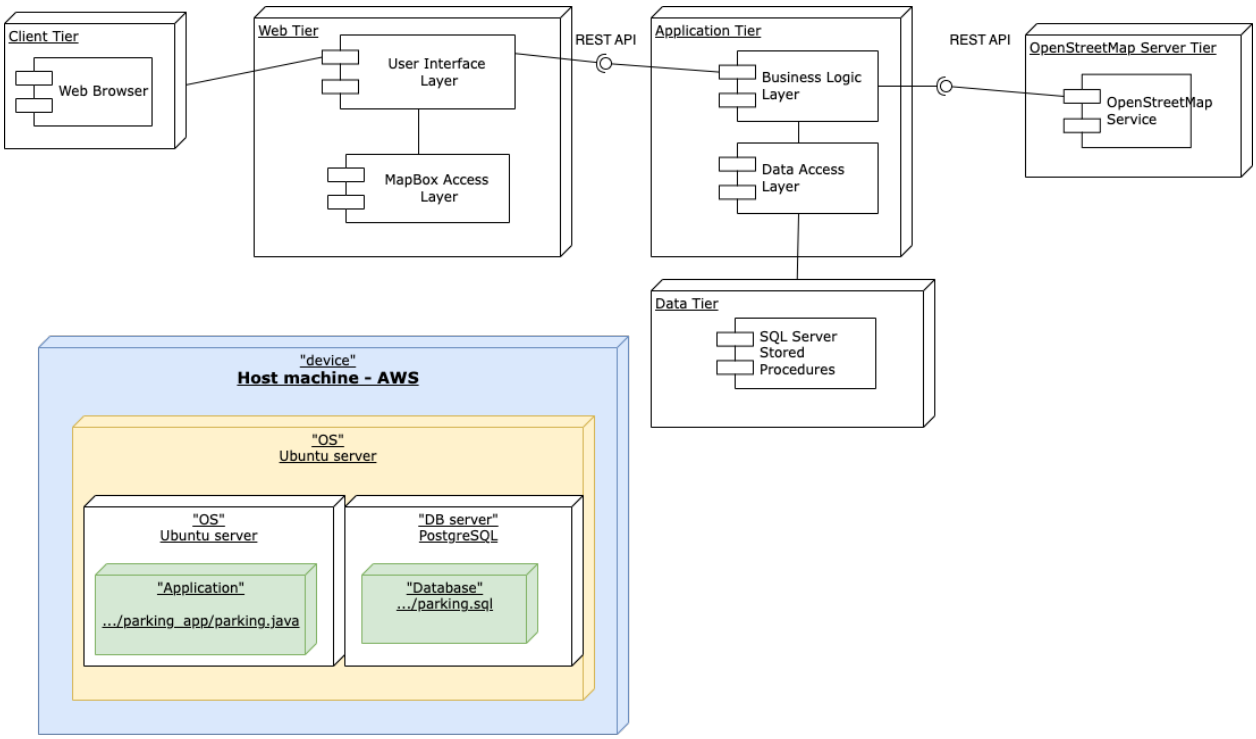


Рисунок Г.2 – UML діаграма розгортання

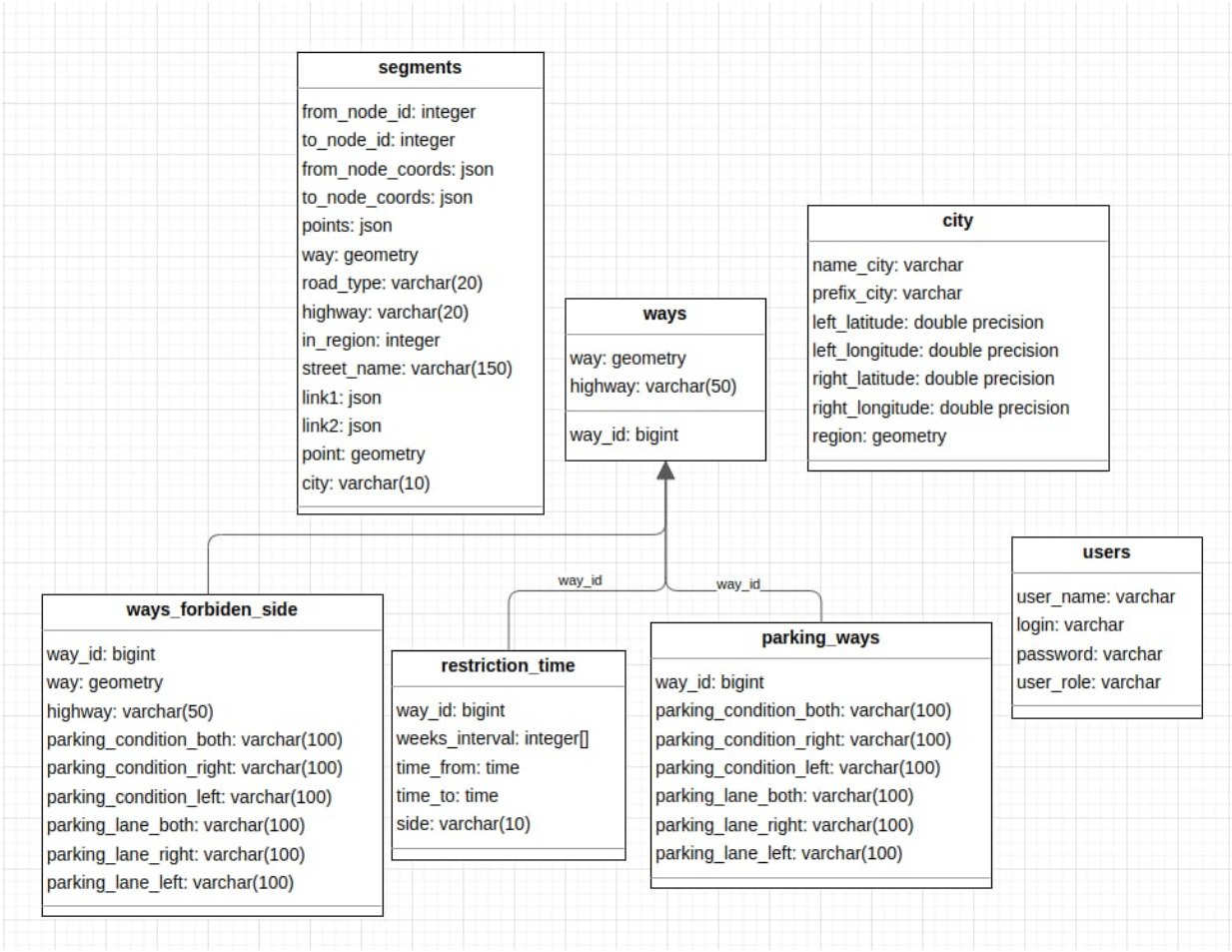


Рисунок Г.3 – ER діаграма бази даних

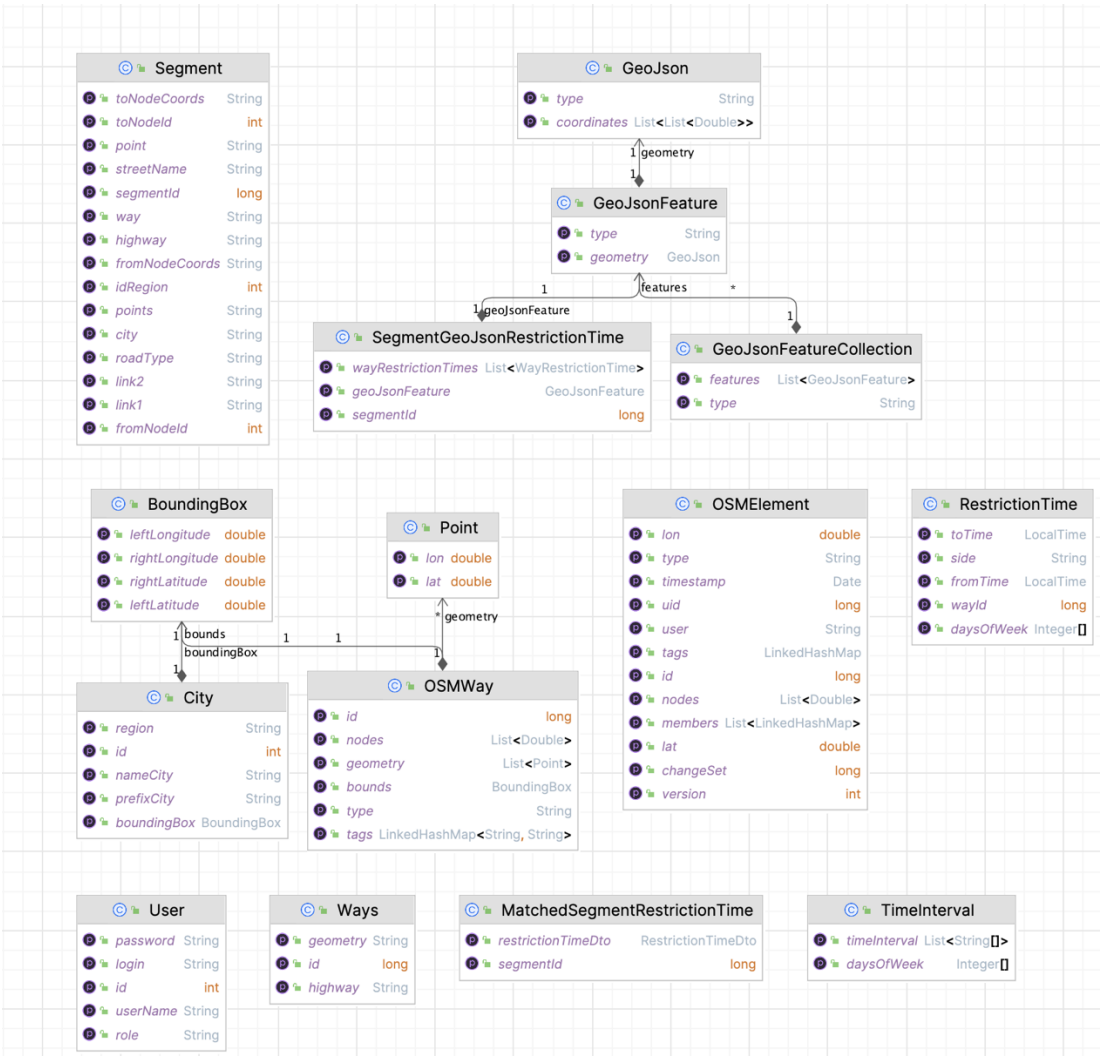


Рисунок Г.4 – UML діаграма основних класів системи

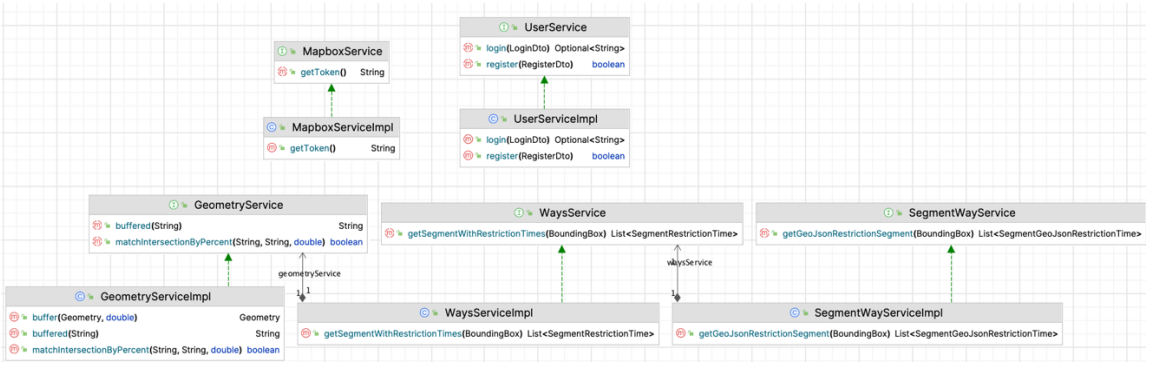


Рисунок Г.5 – UML діаграма класів сервісу опрацювання та аналізу даних паркування (Частина 1)

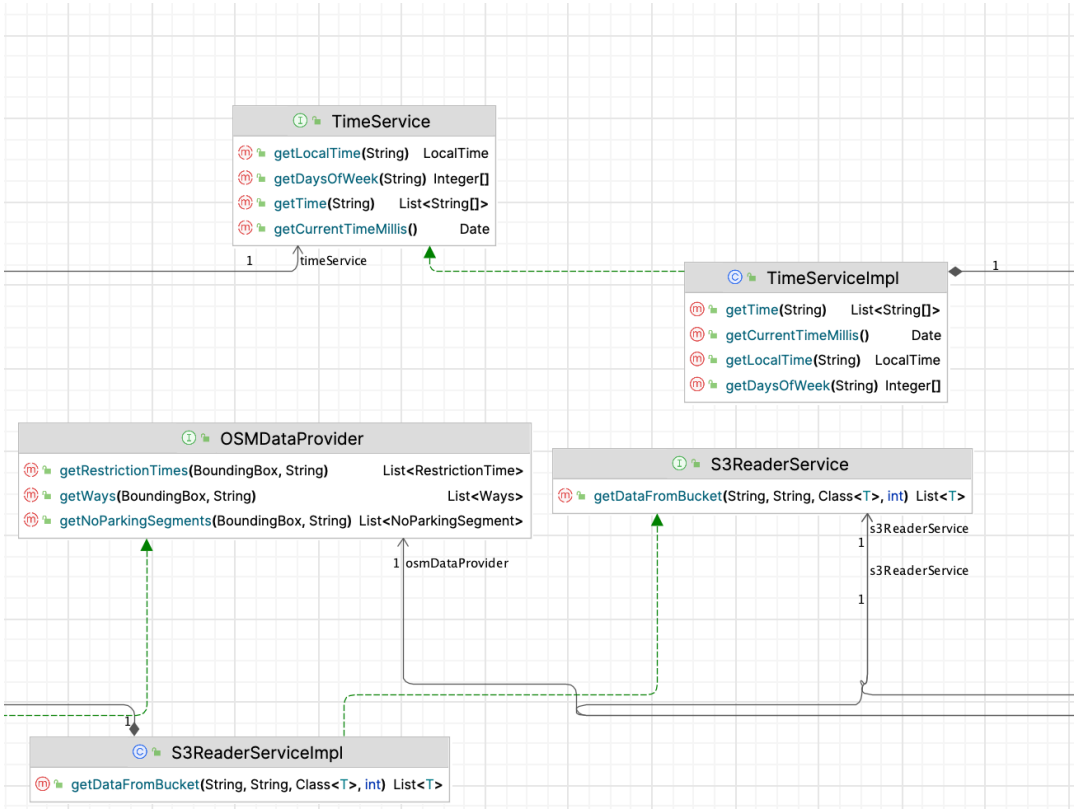


Рисунок Г.6 – UML діаграма класів сервісу опрацювання та аналізу даних паркування (Частина 2)

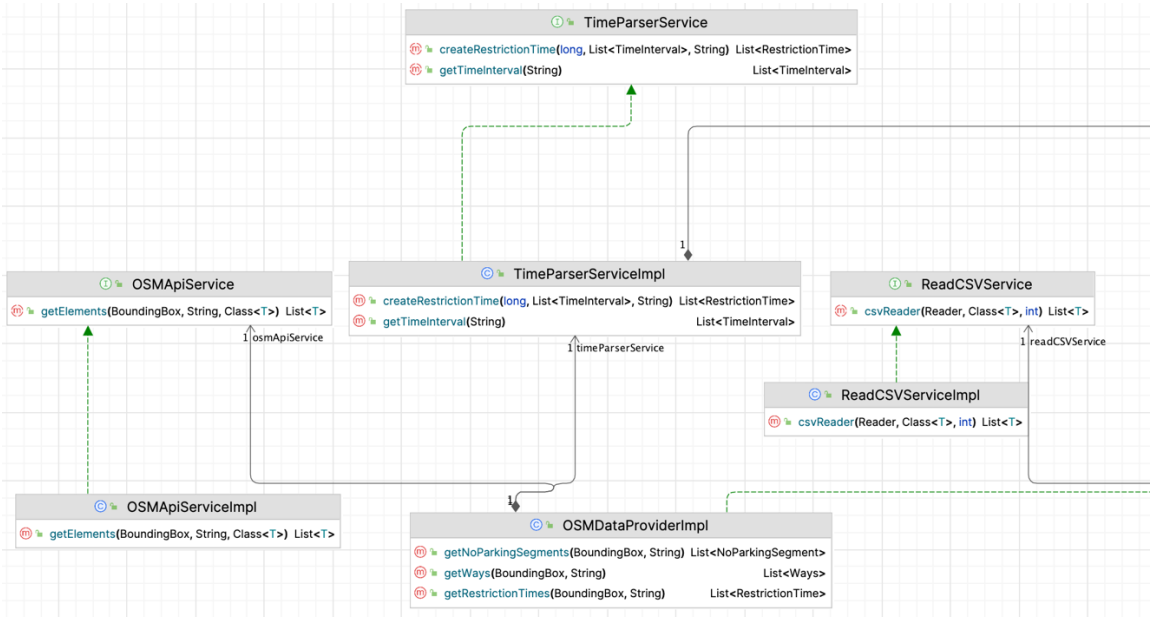


Рисунок Г.7 – UML діаграма класів сервісу опрацювання та аналізу даних паркування (Частина 3)

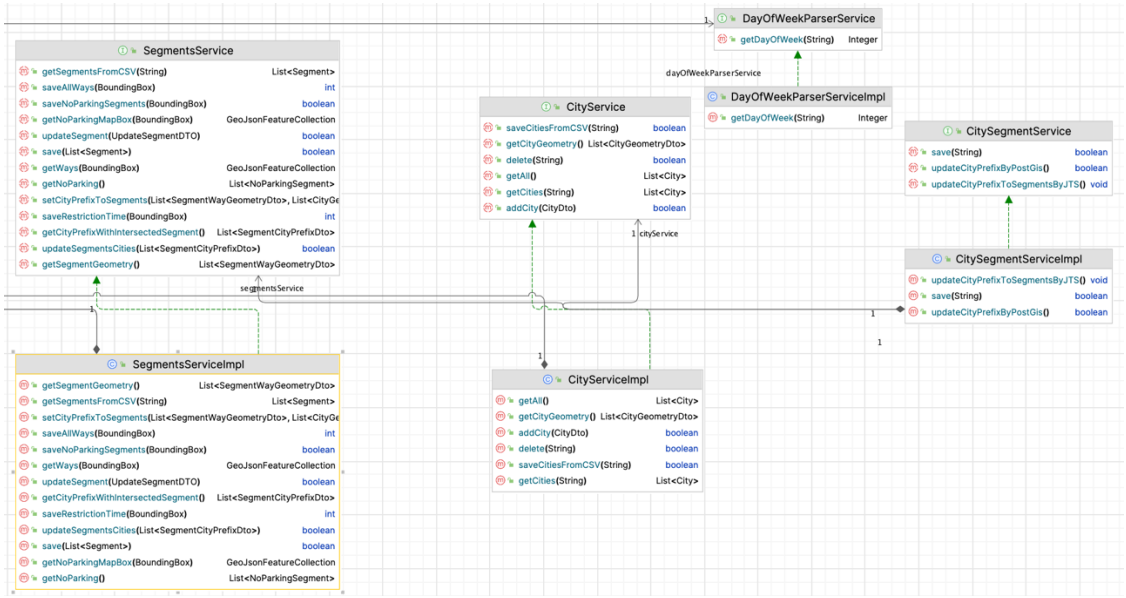


Рисунок Г.8 – UML діаграма класів сервісу опрацювання та аналізу даних паркування (Частина 4)

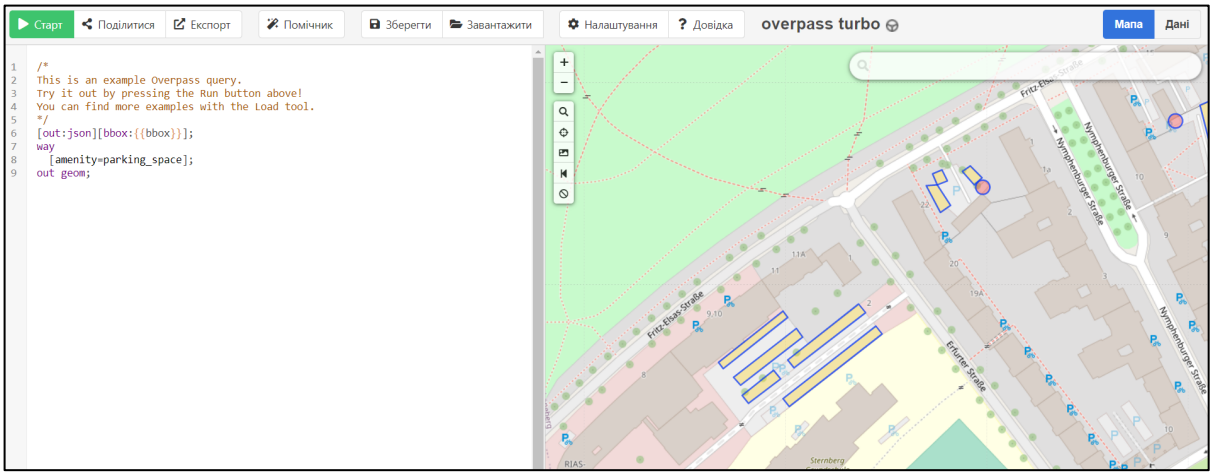


Рисунок Г.9 –Результати тестування GEOJson запиту для отримання даних від OpenStreetMap

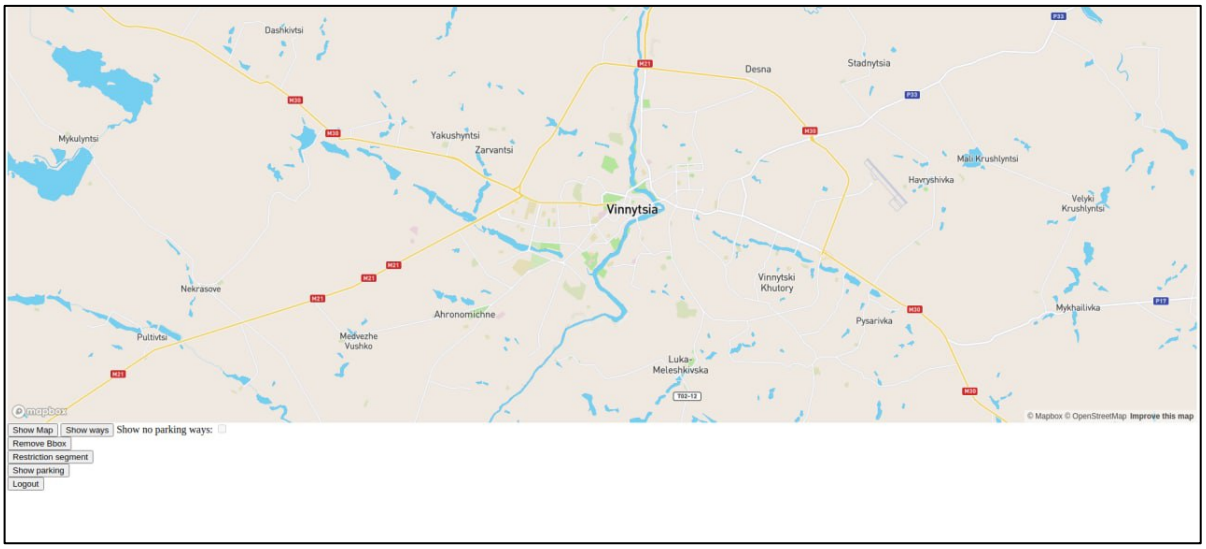


Рисунок Г.10 – Головна сторінка інтерфейсу користувача веб-системи

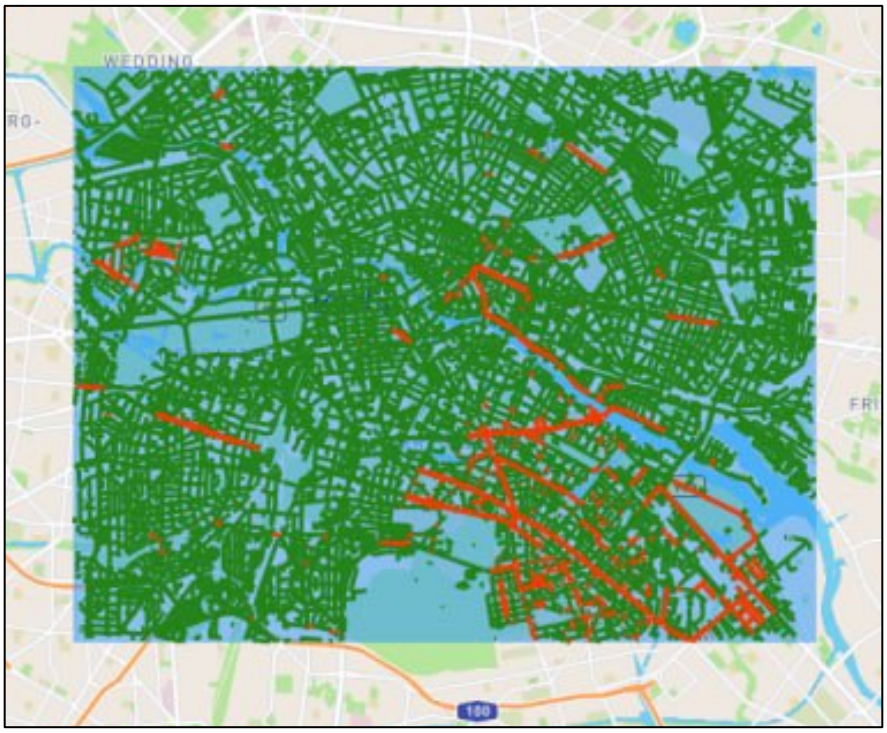


Рисунок Г.11 – Візуалізація заборонених місць для паркування на екрані карти міста

Рисунок Г.12 – Візуалізація наявних місць для паркування на екрані карти міста