Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра Автоматизації та інтелектуальних інформаційних технологій

Пояснювальна записка

до магістерської кваліфікаційної роботи

на тему: «Розробка клієнт-серверної системи моніторингу паркувальних майданчиків(Серверна частина)»

Виконав: студент групи 1ІСТ-21м
Спеціальність: 126 – Інформаційні системи та технології
Копиця В. О
Керівник: к.т.н., доц. каф. АІІТ
Коцюбинський В. Ю.
Рецензент к.т.н., доц. каф. КСУ
Биков М.М.

Вінниця ВНТУ – 2022 рік

АНОТАЦІЯ

В результаті даної роботи було розроблено серверну частину системи моніторингу паркувальних майданчиків. В даній роботі розглянуто та проаналізовано клієнт-серверні системи моніторингу паркувальних майданчиків. Спроектовано архітектуру основних модулів системи моніторингу, а саме: модуль для збору інформації паркувальних майданчиків, перевірки зібраних даних, створення пакетів даних про паркувальні майданчики та модуль, що відповідає за інтеграцію створених пакетів з інформаційною панеллю.

На основі отриманих результатів спроектовано архітектуру серверної частини системи моніторингу паркувальних майданчиків.

Розроблено базовий функціонал серверної частини системи моніторингу паркувальних майданчиків.

ANNOTATION

As a result of this work, the server part of the parking lot monitoring system was developed. In this work, client-server systems for monitoring parking lots are considered and analyzed. The architecture of the main modules of the monitoring system was designed, namely: a module for collecting parking lot information, checking the collected data, creating parking lot data packages, and a module responsible for integrating the created packages with the information panel.

Based on the obtained results, the architecture of the server part of the parking lot monitoring system was designed.

The basic functionality of the server part of the parking lot monitoring system has been developed.

ВІДГУК

наукового керівника
на магістерську кваліфікаційну роботу
студента групи 1ІСТ-21м Копиці В. О.
на тему: «Розробка клієнт-серверної системи моніторингу паркувальних майданчиків(Серверна частина)»

Представлена магістерська кваліфікаційна робота присвячена розробці серверної частини системи моніторингу паркувальних майданчиків. Враховуючи дифіцит подібних клієнт-серверних систем та постійне зростання попиту на інструменти для моніторингу паркувальних майданчиків, виникає гостра необхідність у зручних, багатофункціональних статистичних та аналітичних інструментах, що максимально відповідатимуть потребам клієнтів. З огляду на це, магістерська робота характеризується актуальністю та своєчасністю.

В якості новизни роботи слід відзначити запропонований автором спосіб збору статистичних даних по паркувальних майданчиках на основі збору сесій водіїв та автоматичну перевірку зібраних даних, що, на відміну від існуючих аналогів, дає можливість автоматизувати процес перевірки статистичних даних та у разі виникнення невідповідностей відправляти їх на модерацію. Використання технології Spring Boot для реалізації серверної частини системи дозволяє зменшити час та вартість розробки системи в цілому.

Позитивними рисами дипломної роботи є системність та послідовність викладення матеріалу, а також застосування прогресивного досвіду в моніторингу паркувальних майданчиків та його практичне застосування.

Цінність роботи полягає у розробці алгоритмічного та програмного забезпечення що дозволяє аналізувати та оцінювати якість створених пакетів інвентаризації.

Студентом було проведено аналіз та порівняння можливих методів розв’язання поставленої задачі та обрано оптимальний варіант. Під час виконання магістерської кваліфікаційної роботи студент Копиця В. О. проявив себе грамотним, кваліфікованим спеціалістом здатним приймати самостійно складні технічні рішення.

Вважаю, що магістерська робота Копиці В. О. в загальному відповідає вимогам магістерської кваліфікаційної роботи, а її автор, Копиця Вадим Олександрович, заслуговує оцінку «відмінно» та присвоєння кваліфікації: ступінь вищої освіти магістр, спеціальність 126 – «Інформаційні системи та технології», освітня програма «Інтелектуальні інформаційні технології».

к.т.н., доц. каф. АІІТ Коцюбинський В. Ю.

РЕЦЕНЗІЯ

на магістерську кваліфікаційну роботу

студента групи 1ІСТ-21м Копиці В. О.

на тему: «Розробка клієнт-серверної системи моніторингу паркувальних майданчиків(Серверна частина)»

Магістерська кваліфікаційна робота, яку подано на рецензію, виконана у повному обсязі у встановлений термін. Робота складається з таких частин: аналіз та порівняння існуючих клієнт-серверних систем моніторингу паркувальних майданчиків, проектування архітектури системи, розробка програмного забезпечення, економічна частина, висновки, додатки.

Актуальність розробки викликана необхідністю вирішення проблем, пов’язаних з автоматизованим збором статистики по паркувальних майданчиках, перевіркою зібраних даних та інтеграцією, створених на базі статистики, даних інвентаризації з інформаційною панеллю, а також дослідженням процесів, пов’язаних з ними.

В межах роботи реалізовано серверну частину систему моніторингу паркувальних майданчиків, яка базується на основі статистичних даних по паркувальних майданчиках міст. Запропоновано та обгрунтовано можливі технології реалізації. Для розробки серверної частини автором обрана мова програмування Java та фреймворк Spring. Для тестування – Mockito.

Розроблена система розширює функціональні можливості роботи інших систем, та поліпшує їх роботу.

Автор провів складне дослідження не тільки теоретичного матеріалу, а також дослідив потреби реальних підприємств, які спеціалізуються у наданні подібних сервісів. Результати роботи були прийняті до впровадження на НВП ТОВ «Спільна Справа».

Зауваження до роботи: принцип роботи модулю інтеграції з інформаційною панеллю можна було б описати більш детально. Дане зауваження не є критичним, робота написана на професійному рівні та її актуальність не викликає сумніву.

Магістерська кваліфікаційна робота виконана у відповідності із завданням та з дотриманням всіх вимог. Робота заслуговує оцінки «відмінно», а її автор – присвоєння кваліфікації: ступінь вищої освіти магістр, спеціальність 126 – «Інформаційні системи та технології», освітня програма «Інтелектуальні інформаційні технології».

Рецензент: к.т.н., доц. кафедри КСУ Биков М. М.

ЗМІСТ

ВСТУП

Актуальність. Зі зростанням населення земної кулі та збільшенням кількості транспортних засобів в містах, проблема з моніторингом паркувальних майданчиків стає все більш критичною. Паркувальні оператори витрачають час, людський ресурс і кошти для організації моніторингу ситуації в режимі реального часу з паркувальними майданчиками та збором статистики по таким ключовим параметрами як, розташування, кількості місць для паркування, тип паркувального місця та дорожні знаки, які дозволяють або забороняють паркування на конкретному майданчику.

Саме тому є необхідність в системах, які будуть не тільки обробляти отриману статистику, а також проводити перевірку отриманих даних в автоматичному режимі, у разі виявлення недостовірних параметрів відправляти на перевірку до модератора, створювати на основі отриманої статистики дані у відповідному форматі, перевірки та каналу для автоматизованої доставки створених даних до інформаційної панелі.

Поширення таких клієнт-серверних систем моніторингу паркувальних майданчиків в наступні роки буде активно розвиватися, завдяки розвитку інформаційних технологій та реальній необхідності в моніторингу. З кожним роком все більше користувачів надають перевагу цифровим технологіям і готові використовувати їх у повсякденному житті.

Система передбачає надійний захист всіх персональних даних користувачів. За допомогою даних систем місцева влада буде мати доступ до інформаційної панелі моніторингу за паркувальними майданчиками усього міста, може прогнозувати та аналізувати інформацію з метою їх контролю та покращення надання послуг. Дані системи також доступні для інтеграції з іншими компаніями, наприклад з великими паркувальними операторами, з метою удосконалення процесу надання послуг користувачам.

Метою даної роботи є покращення процесу надання послуг у моделі взаємодії B2C (Bussiness-to-Consumer), B2B (Bussiness-to-Bussiness), B2G (Bussiness-to-Government) за рахунок автоматизації процесу збору та перевірки зібраної статистики по паркувальних майданчиках та інтеграції створених пакетів інвентаризації з інформаційною панеллю для моніторингу даних за основними параметрами з метою контролю та покращення надання послуг.

Для того, щоб досягнути вищевказаної мети потрібно розв’язати низку таких *задач*:

- розглянути основні принципи функціонування систем моніторингу паркувальних майданчиків;
- проаналізувати технології та підходи для створення серверної частини системи моніторингу;
- дослідити технології та підходи для створення клієнтської частини системи моніторингу;
- проаналізувати механізми та технології взаємодії серверної частини із базою даних PostgreSQL;
- на основі розглянутих підходів та технологій здійснити реалізацію серверної частини системи моніторингу паркувальних майданчиків.

Об’єктом дослідження є процес взаємодії користувачів із сервісами системи моніторингу паркувальних майданчиків.

Предметом дослідження є процес реалізації серверної частини системи моніторингу паркувальних майданчиків.

Методи дослідження. У роботі використано методи ідентифікації, збору і обробки даних про користувачів та їх взаємодію із представниками держави та бізнесу, а також аналіз роботи консультантів для підвищення якості процесу надання послуг.

Наукова новизна отриманих результатів полягає у тому, що дана система на основі удосконалених моделей взаємодії B2B, B2G та B2C для представників бізнесу, державних організацій та звичайних користувачів, на відміну від існуючих розробок, надає можливість виконувати моніторинг паркувальних майданчиків за допомогою інформаційної панелі.

Практична цінність. Проведені дослідження дають можливість розробити автоматизовану клієнт-серверну систему моніторингу паркувальних майданчиків для надання користувачам безпечної взаємодії із сервісами та послугами представникам держави та бізнесу, збору статистики та веденню правок по зверненню, аналізу отриманих даних та параметрів.

Апробація результатів та публікації. Результати роботи були розглянуті на L науково-технічній конференції Факультету інтелектуальних інформаційних технологій та автоматизації 2022 [1].

Результати роботи були впровадженні на підприємстві НВП ТОВ “Спільна Справа”, що підтверджується відповідними актами впровадження.

1 АНАЛІЗ ТА ПОРІВНЯННЯ ІСНУЮЧИХ КЛІЄНТ-СЕРВЕРНИХ СИСТЕМ МОНІТОРИНГУ ПАРКУВАЛЬНИХ МАЙДАНЧИКІВ

1.1 Аналіз проблеми та постановка задачі

За останні роки, у великих по населенню та площею містах, настає проблема з дефіцитом паркувальних майданчиків та зростанням кількості порушень правил дорожнього руху. Одним із рішень цієї проблеми є збільшення паркувальних місць, реорганізація дорожнього простору та його оптимізація. Тому правильні кроки та рішення є одним із найбільш впливових напрямків міської політики, який безпосередньо впливає на якість життя в місті в цілому.

В деяких великих містах це є великою проблемою, так як немає можливості розширювати паркувальні місця, але є можливість оптимізувати дорожній простір, а також повністю автоматизувати всі процеси пов`язані з паркуванням, аналізом та збором статистики по кожному паркувальному майданчику та їх моніторингу в цілому.

Загалом клієнт-серверні системи моніторингу паркувальних майданчиків в сучасному світі користуються великим попитом через:

- малу кількість подібних рішень, які дозволяють проводити збір статистики по паркувальним майданчикам;
- дефіцит паркувальних майданчиків в містах в цілому;
- необхідність систем для моніторингу паркувальних майданчиків та отримання інформації по розташуванні, типу та кількості місць для паркування;
- необхідність в аналізу даних за ключовими параметрами.

Саме клієнт-серверні частини систем моніторингу паркувальних майданчиків надають муніципалітетам та бізнесу можливість моніторингу даних по паркувальних майданчиках за допомогою інформаційної панелі, що дозволяє заощадити час та кошти.

Сучасні клієнт-серверні системи моніторингу паркувальних майданчиків, мають забезпечувати оцінку завантаження процесора, використання пам'яті роботи жорстких дисків системи, завантаження мережевих інтерфейсів, тестування критичних місць.

Також такі системи повинні відновлювати роботу систему у разі не критичних помилок без залучення системного адміністратора.

Основною задачею в межах даної роботи є розробка серверної частини системи моніторингу паркувальних майданчиків.

Базовими вимоги до системи є:

- відслідковування основних параметрів паркувальних майданчиків за допомогою інформаційної панелі у вигляді мапи;
- опрацьовування та порівняння параметрів на основі отриманих еталонних значень, які встановлюються для кожного параметру;
- обробка та збереження статистики по паркувальних майданчиках;
- автоматична перевірка зібраної статистики;
- створення пакету даних у відповідному форматі типу GEOJSON;
- оновлення пакету у разі внесення правок від муніципалітету або бізнесу у автоматичному режимі.

Не менш важливим є базовий набір параметрів, які зберігаються підчас процесу збору статистики паркувальних майданчиків, до таких параметрів відносяться:

- розташування;

- кількість місць для паркування;
- тип паркувального місця;
- дорожні знаки, які дозволяють або забороняють паркування на конкретному майданчику.

Основною метою являється розробка серверної частини інформаційної панелі за допомогою якої представники муніципалітету або бізнесу можуть проводити моніторинг за паркувальними майданчиками та відслідковувати їх за параметрами що дозволяє заощадити час та кошти.

1.2 Аналіз подібних систем

Для забезпечення моніторингу паркувальних майданчиків існує багато новітніх клієнт-серверних систем з великою кількістю можливостей.

Parking Dashboard. Розробка компанії Cleverciti, одна з найпопулярніших систем в західній Європі для моніторингу паркувальних майданчиків.

Переваги:

- велике покриття паркувальних операторів;
- великий набір інструментів для моніторингу;
- великий набір параметрів для налаштування;
- можливість використання у сумісності з різними операційними системами;

Недоліки:

- покриття операторів тільки центральної та західної частин Європи;
- співпрацюють тільки з великими паркувальними операторами;
- недоступність для звичайного користувача.

Parkopedia. Ще одна цікава клієнт-серверна система, яка направлена на зручність користування та низькому порозі входу для нових операторів.

Переваги:

- зручність використання;
- можливість швидкого освоєння основного функціоналу;
- можливість моніторингу основних параметрів системи через користувацький інтерфейс;

Недоліки:

- обмежений набір інструментів для моніторингу систем;
- обмежена безкоштовна версія.

Розглянемо переваги та недоліки системи, яка розробляється в межах даної роботи.

Переваги:

- можливість моніторингу у режимі реального часу;
- простота інтеграції з новими операторами;
- простий та зрозумілий клієнтський інтерфейс;
- можливість моніторингу у режимі віддаленого доступу;
- можливість отримання повідомлень про системні збої;

Недоліки:

- обмежений набір параметрів моніторингу;
- необхідність проведення додаткових налаштувань збоку паркувального оператора;
- необхідність базових знань щодо використання таких систем.

Як висновок можемо відзначити, що клієнт-серверна частина системи моніторингу паркувальних майданчиків, яка розробляється у ході даної роботи,

акцентується на досягненні максимальної швидкодії, легкості у використанні та можливості оптимізації використання зовнішніх ресурсів.

Також доволі вагомою перевагою є інтегрована система перевірки зібраної статистики, яка допомагає проводити перевірку даних у автоматичному режимі, у разі невідповідності параметрів відправка на перевірку до модератора.

1.3 Аналіз роботи систем моніторингу паркувальних майданчиків

Аналізуючи існуючі системи, та стрімкий розвиток IT технологій, можемо розглянути наступні тенденції, щодо подальшої перспективи використання системи моніторингу паркувальних майданчиків[1].

Найважливіші вимоги, які мають бути реалізовані найближчим часом:

- збір статистики;
- надання статистики по основних параметрах;
- розширення функціоналу (включаючи масштабування системи та синхронізації з хмарними технологіями);
- підвищення ефективності методів оптимізації, задля забезпечення мінімального використання ресурсів;
- підвищення ефективності перевірки статистики.

Завдяки таким удосконаленням системи та переваги перед конкурентами ця клієнт-серверна система моніторингу паркувальних майданчиків має великі комерційні перспективи.

1.4 Аналіз вибору архітектурного патерну та вимог до розробки сучасних клієнт-серверних систем моніторингу паркувальних майданчиків

Клієнт – серверна система є домінуючою концепцію інформаційної мережі в якій основна частина її ресурсів зосереджена в серверах, обслуговуючих своїх клієнтів [2]. Така архітектура визначає такі типи компонентів:

- набір серверів, які надають інформацію або інші послуги програмам, які звертаються до них;
- набір клієнтів, які використовують сервіси, що надаються серверами;
- мережа, яка забезпечує взаємодію між клієнтами та серверами.

Сервер являє собою програмний комплекс, який знаходиться на віддаленому комп'ютері, та забезпечує організацію, структуризацію, збереження та обробки певного набору даних, яка залежить від рішень на стороні клієнта.

Забезпечення зв'язку, кодування інформації в клієнт - серверних реалізується за допомогою новітніх інтернет протоколів та різних методів кодування.

Для авторизації користувача в системі зазвичай використовується дворівнева аутентифікація OAuth2, з подальшим наданням JWT токену зі сторони серверу [3].

Клієнтом являється будь-яка програма чи апаратний компонент, який використовує ресурси сервера і надає клієнту певний функціонал, зазвичай за допомогою зручних інтерфейсів користувача.

Основною задачею клієнта є збір інформації та передача на сервер у вигляді запиту, або відображення реакції серверу на певні дії користувача у зрозумілому для нього вигляді.

Дана система розробляється за допомогою трьохрівневої клієнт – серверної архітектури (модель тонкого клієнта).

Така архітектура ділиться на два типи за функціональними особливостями [4]:

- модель тонкого клієнта, в рамках якої вся логіка застосунку та управління даними зосереджена на сервері. Клієнтська програма забезпечує тільки функції рівня представлення;
- модель товстого клієнта, в якій сервер тільки керує даними, а обробка інформації та інтерфейс користувача зосереджені на стороні клієнта. Товстими клієнтами часто також називають пристрої з обмеженою потужністю: кишенькові комп'ютери, мобільні телефони та ін.

Шаблоном, або патерном проектування клієнт – серверної системи являється архітектурне рішення, за допомогою якого відбувається структуризація даних та виділення окремих компонентів бізнес логіки на рівні ядра системи. Вони допомагають зробити зрозумілішим код програми, розподілити функції в межах свого функціоналу та реалізувати як найменший зв'язок між компонентами програми [5].

Для розробки клієнт-серверних систем має місце використання таких архітектурних рішень, на основі яких система стає якомога гнучкішою.

Існує ряд сучасних патернів такі, як: MVC, MVP, MVVM та MVI [6]. Дана система планується розроблюватись за допомогою архітектури MVC, яка дозволяє чітко відділити бізнес – логіку, моделі обробки, та вид відображення [7, 8].

1.5 Висновки до розділу

В даному розділі розглянуто та проаналізовано подібні клієнт-серверні системи моніторингу паркувальних майданчиків, сформульовано основні вимоги, щодо роботи системи.

Проаналізовані існуючі методи моніторингу, патерни програмування та основні вимоги, щодо розробки клієнт-серверних додатків, їх переваги та недоліки.

2 ПРОЕКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ

2.1 Аналіз інформаційних систем

Інформаційна система – це сукупність взаємопов’язаних компонентів, які визначають різні сторони інформаційної діяльності об’єкта.

Основні напрямки використання:

- засоби фіксації і збору інформації;
- засоби передачі відповідних даних та повідомлень;
- засоби збереження інформації;
- засоби аналізу, обробки і представлення інформації;
- будь-яка інформаційна система має вирішувати поставлені задачі:
- задача оцінки ситуації;
- розрахункова задача, або задача моделювання;
- задача прийняття рішення при різних поведінках системи.

Інформаційна система програми, яка розробляється у рамках даної роботи, відповідає даним вимогам.

Клієнт-серверна частина системи моніторингу паркувальних майданчиків, відповідає за забезпечення усіх заходів щодо безпеки, збір статистики по заданим параметрам, перевірка зібраної статистики в автоматичному режимі, створення пакетів даних по паркувальних майданчиках в форматі GEOJSON та їх перевірку, автоматичну доставка пакетів до інформаційної панелі.

Аналіз інформаційної системи наведений на рисунку 2.1.

Рисунок 2.1 – Інформаційна схема клієнт-серверної системи моніторингу паркувальних майданчиків

2.2 Розробка архітектури системи

Клієнт-серверна система моніторингу паркувальних майданчиків буде складатись з таких модулів:

- модуль отримання статистики у вигляді сесій водіїв;
- модуль обробки та перевірки отриманої статистики;
- модуль для створення пакетів інвентаризації;
- модуль, який відповідає за перевірку створеного пакету;
- модуль для автоматичної доставки пакетів до інформаційної панелі.

Модуль отримання статистики у вигляді сесій водіїв відповідає за збір встановлених параметрів, до цих параметрів зазвичай належать: розташування, кількість місць для паркування, тип паркувального місця, дорожні знаки, які дозволяють або забороняють паркування на конкретному майданчику.

Модуль обробки та перевірки отриманої статистики відповідає за автоматичну перевірку отриманої статистики в результаті сесії користувачів, у разі невідповідності параметрів система відправляє сесію на перевірку модератору, який має змогу перевірити зібрану статистику.

Модуль створення пакетів інвентаризації відповідає за створення пакетів у таких форматах, як: GEOJSON та CSV, які підтримуються такими мапами, як: Google Maps, Mapbox, OpenStreetMaps. Даний модуль є невід'ємною складовою серверної частини, в якому сконцентрована бізнес – логіка системи для ініціалізації створення пакетів.

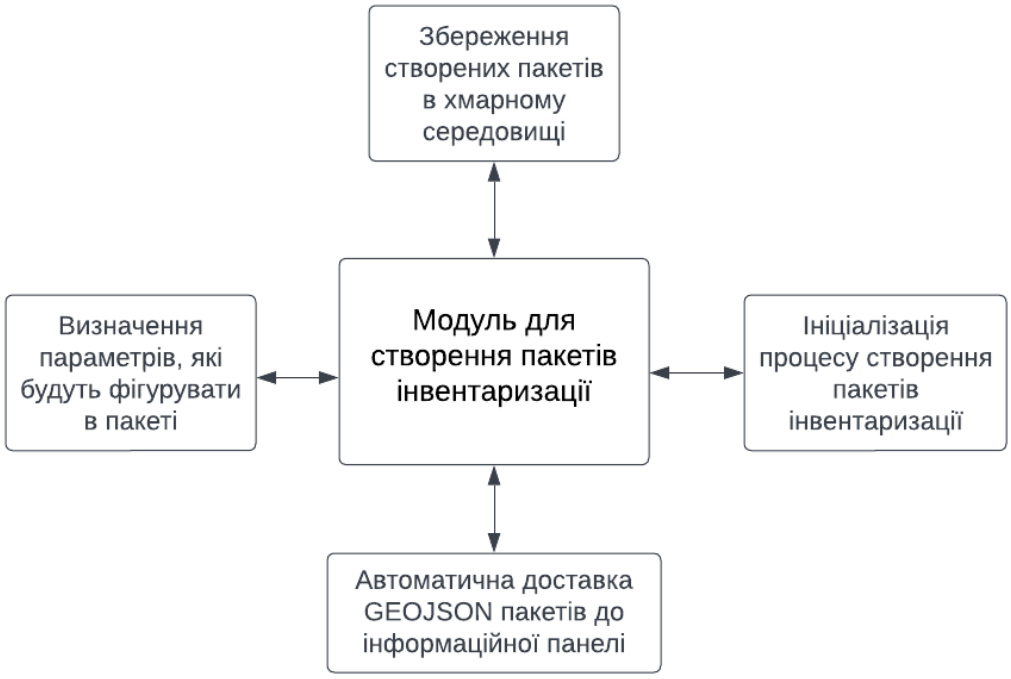


Рисунок 2.2 – Функціональна схема модулю для створення пакетів інвентаризації

Модуль автоматичної відправки пакетів інвентаризації відповідає за відправку та інтеграцію створених пакетів з інформаційною панеллю.

У підсумку варто зазначити, що кожен із перелічених модулів має свої підсистеми, які виконують певний об’єм функцій.

Для прикладу розглянемо функціональну схему модулю для створення пакетів інвентаризації. Функціональна схема модулю для створення пакетів інвентаризації зображена на Рисунку 2.2.

Як зазначено на рисунку, даний модуль буде виконувати доволі великий обсяг роботи для забезпечення стабільності та коректності роботи системи.

UML діаграму архітектури системи наведено у додатку В.

2.3 Проектування модулю збору статистики про паркувальні майданчики

Давайте розглянемо способи збору статистики про паркувальні майданчики за допомогою інформаційних технологій.

Перший спосіб полягає в тому, що необхідно буде розробити мобільний додаток за допомогою якого, водії можуть збирати статистику по паркувальних майданчиках міста. Для цього необхідно запустити додаток у якому є навігація по маршруту який необхідно покрити, далі прикріпити свій мобільний пристрій на панель автомобіля. Додаток під час подорожі буде робити фото маршруту і відправляти на сервер. Водій, якщо бачить паркувальний майданчик, клікає за допомогою спеціальної гарнітури. Кількість кліків дорівнює кількості місці для парковки, також записуються географічні координати, всі дані відправляються на сервер, який зберігає всі дані, які йдуть від водія у вигляді окремих сесій[11].

Другий спосіб полягає у використанні Google Maps, а саме Street View режиму, він надає можливість переглядати мапу міст в 3D і збирати необхідну інформацію. Для цього нам потрібен мапер, який буде працювати з мапою і збирати

необхідну статистику. Він зібрані дані будуть оброблятись системою і зберігатись у базі даних.

Третій спосіб полягає в інтеграції статистики з сторонніх ресурсів, логіка системи зосереджена на обробці даних у відповідному форматі та збереження їх у базі даних.

За допомогою модулю збору статистики можна зібрати всі необхідні дані для створення пакету інвентаризації та інтегрувати ці дані в інформаційну панель.

Workflow діаграма модулю збору статистики про паркувальні майданчики наведена у додатку Г.

2.4 Проектування модулю перевірки зібраної інформації

Після того як статистика була зібрана за допомогою модулю збору статистики про паркувальні оператори, необхідно провести перевірку даних. Перевірку зібраних даних можна провести в автоматичному та у ручному режимах. Якщо перевірка у автоматичному режимі показала, що дані мають невідповідності або не властиві дані, зібрана статистика відправляється на перевірку до команди модераторів, які перевіряють дані.

Workflow діаграма модулю перевірки статистики про паркувальні майданчики наведена у додатку Д.

2.5 Вибір технологій розробки серверної частини

Під час планування розробки системи моніторингу паркувальних майданчиків, для реалізації серверної частини системи, було обрано мову програмування Java.

Однією із особливостей створення Java – додатків, цікавих з точки зору реалізації корпоративних рішень, слід зазначити методи підвищення продуктивності роботи розробників та підтримки колективної роботи.

Для прикладу легка реалізація підтримки різних етапів життєвого циклу програм, як попередніх (написання коду додатків), так і наступних (супровід, впровадження та тестування), повторне використання моделей та коду, підтримка засобів та стандартів для створення розподілених додатків і їх інтеграції (включаючи підтримку стандарту J2EE та Web-сервісів), а також можливість створення мобільних рішень, у тому числі підтримка стандарту J2ME. [20].

Сучасні інструменти для розробки Java – додатків, у більшості випадків, володіють різними засобами для підвищення продуктивності роботи програмістів, наприклад:

- редактори для створення інтерфейсів користувача;
- підтримка створення додатків з інтеграцією баз даних (у випадку Java – за допомогою драйверу, що надає доступ до даних у БД – JDBC, Java DataBase Connectivity);
- набір готових рішень для створення REST – сервісів;
- підтримка сучасних архітектурних патернів;
- простота інтеграції з зовнішніми ресурсами;
- набір фреймворків для швидкої і продуктивної розробки;
- легка інтеграція з сучасними середовищами розробки;

У більшості випадків, сучасні засоби створення Java – додатків забезпечують кросплатформову розробку.

Це означає, що розроблений додаток, може бути запущений на різних операційних системах, які не пов’язані між собою. Зазвичай в їх число входять Linux, Windows, Solaris, а іноді й інші операційні системи.

Також слід зазначити, що Java, є доволі перспективною мовою програмування, завдяки регулярним оновленням від Oracle та легкою інтеграцією з відомими сервісами Google, Amazon і тд.

Java має великий набір готових бібліотек та плагінів, які роблять розробку швидкою та цікавою.

2.5.1 Вибір архітектури мережевих протоколів

Під час вибору підходу до архітектури мережевих протоколів, під час розробки розподіленої системи моніторингу інтернет сервісів та під час проектування системи прийняття рішень, був вибраний один із найпростіших та прогресуючих архітектурних стилів – REST.

Як аналог для порівняння можна розглянути набір стандартів та протоколів SOAP.

В основі REST закладено принципи функціонування всесвітньої павутини і, зокрема, можливості HTTP. [33]

REST представляє собою передачу або зміну стану через представлення.

Ключове поняття в REST - це ресурс. Ресурс має стан, і ми можемо його отримувати або змінювати за допомогою представлень. [34]

Додаток, який проектується буде відповідати за безліч таких ресурсів (безліч API для доступу до них). Кожна одиниця інформації однозначно визначається глобальним ідентифікатором, таким як URL.

Отримання даних за допомогою REST – сервісів не потребує додаткових «обгорток» в XML, так як це робить SOAP, а просто отримуємо данні у зручному для нас форматі, перейшовши за URL – посиланням.

Якщо розглядати REST архітектуру, яка використовується у даній системі, то запити до контролера приходять у вигляді REST – запитів, які бувають таких видів:

- GET. Запит на отримання даних з відповідного ресурсу;
- POST. Запит на запис даних у ресурс;
- PUT. Запит на створення або оновлення даних ресурсу;
- PATCH. Запит на оновлення даних ресурсу;
- DELETE. Запит на видалення даних з ресурсу;
- UPDATE. Запит на оновлення даних у ресурсі;
- За допомогою цієї архітектури, ми отримуємо такі переваги:
- отримання різних уявлень для різних типів пристроїв, при цьому користуючись одними і тими ж даними;
- чітка структура коду програми;
- явний розподіл окремих компонентів програми за їх функціональним призначенням.

SOAP - це ціле сімейство протоколів і стандартів, звідки безпосередньо випливає, що це більш великоваговий і складний варіант з точки зору машинної обробки.

Основними перевагами REST є простота та швидкість, а SOAP – безпека та структурованість.

Хотілось би зауважити, що REST і SOAP не являються конкурентами, адже використовуються у різних цільових напрямах і навряд чи знайдеться задача, для якої буде складно сказати, який підхід раціональніше використовувати.

Для розробки системи моніторингу інтернет сервісів використовується REST архітектура, для забезпечення швидкої та простої взаємодії клієнта з сервером.

Ця архітектура дуже проста в плані використання та легко інтегрується в Java за допомогою сучасних фрейм ворків, таких як: Spring Framework, Dropwizard і т.д.

Саме тому даний підхід до архітектури мережевих протоколів буде використовуватись під час розробки системи прийняття рішень, щодо підбору віртуальної машини.

2.5.2 Вибір архітектурного шаблону для проектування та розробки програмного забезпечення

Для структуризації ресурсів, під час проектування розробки серверної частини, було вирішено використовувати відомий патерн програмування MVC.

Для реалізації поставленої задачі необхідно розібрати основні компоненти цього патерну.

Патерн модель-вигляд-контролер (MVC) – це архітектурний шаблон, який використовується під час проектування та розробки програмного забезпечення.

Цей шаблон розділяє програму на три основних компоненти, які відповідають за свій функціональний обов’язок.

Такими компонентами є:

- модель;
- відображення;
- контролер;

MVC застосовується для відокремлення окремих функціональних частин, з метою «гнучкого» дизайну програми та забезпеченням найслабкіших зв’язків між програмними компонентами.

Розглянемо детальніше основні функції кожного із компонентів.

Модель. Цей структурний компонент використовується для організації обробки даних і забезпечення доступу до них (бізнес – логіка, робота з базами даних).

Відображення. Цей структурний компонент використовується для представлення вже готових даних користувачеві у зручному вигляді.

Контролер. Цей структурний компонент використовується як прошарок між відображенням та моделлю, та забезпечує доступ до даних з боку інтерфейса користувача, за допомогою користувацьких запитів. Його основна функція – викликати і координувати дію необхідних ресурсів та об'єктів, потрібних для виконання дій, що задаються користувачем.

Завдяки перевагам MVC, які використовувались під час розробки розподіленої системи моніторингу інтернет сервісів, ми маємо змогу розширити існуючу систему, завдяки гнучкості рішень.

Саме тому під час проектування системи прийняття рішень, щодо підбору віртуальної машини є сенс продовжувати тенденцію попередньої розробки та використовувати архітектурний шаблон MVC з можливістю подальшого розширення системи.

2.5.3 Вибір програмного каркасу для імплементації серверної частини

Під час розробки розподіленої системи моніторингу інтернет сервісів (а саме серверної частини), у якості програмного каркасу використовувався Spring Framework. [35]

Під час проектування розробки системи прийняття рішень, щодо підбору віртуальної було вирішено використовувати той ж самий фреймворк, задля забезпечення легкої інтеграції та взаємодії з попередньо - розробленою системою.

Spring Framework включає в себе:

- контейнери, які забезпечують автоматизовану, централізовану конфігурацію та з’єднання об’єктів. Контейнери складаються з наборів вільно з’єднаних компонентів, складної системи (POJO);
- абстрактні рівні для управління транзакціями, які дозволяють змінювати менеджерів транзакцій та розподіляти їх, не маючи справи з низькорівневими проблемами;

- стратегії JTA та драйвера доступу до бази даних JDBC DataSource. Абстрактний рівень JDBC зменшує об’єм написання коду та організовує ієрархічну обробку виключень;
- легку інтеграцію з Hibernate, Toplink, JDO і тд;
- інтегровану парадигма програмування AOP, яка дозволяє виокремити перехресну функціональність;
- гнучкий MVC фреймворк, який побудований на основній функціональності Spring. Цей фреймворк високо конфігурований через інтерфейси і забезпечує численні технології, такі як Velocity , JSP, iText, Tiles та POI. Проміжний рівень Spring може бути легко об’єднаний з веб рівнем, написаним за допомогою будь - якого іншого MVC веб фреймворку, наприклад, WebWork, Struts чи Tapestry;
- легка інтеграція з зовнішніми ресурсами, такими як: Google API, Amazon API і тд.;

Цей фреймворк ідеально підходить для вирішення поставленої задачі, включаючи інтеграції MVC патерна, та можливості створення REST – контролерів для обміну інформації з клієнтом.

Також Spring Framework має зручний інтерфейс для роботи з зовнішніми ресурсами та великий набір анатоцій, які забезпечують більш абстрактний рівень програмного коду.

2.6 Вибір технологій розробки клієнтської частини

Клієнтський інтерфейс є невід’ємною складовою під час проектування системи прийняття рішень, щодо підбору віртуальної машини, так як саме він забезпечує взаємодію системи з кінцевим користувачем.

За допомогою клієнтського інтерфейсу, користувач матиме змогу відслідковувати зміну параметрів віддаленого сервера (розроблено у модулі розподіленої системи моніторингу інтернет сервісів), а також надати усі необхідні вхідні дані, для забезпечення коректної роботи системи прийняття рішень, щодо підбору віртуальної машини.

Для реалізації клієнтського інтерфейсу було вибрано популярну мову розмітки HTML 5, таблиці стилів CSS, Bootstrap.

Для обміну інформацією з сервером, необхідно реалізувати скриптовий шар, між клієнтом та сервером. Для даної імплементації було вибрано скриптову мову програмування JavaScript та популярний фреймворк Angular для забезпечення простої інтеграції даних у клієнтський інтерфейс [36][37].

2.6.1 Вибір інструменту для упорядкування Web – контенту

HTML – це інструмент для упорядкування Web – контенту.

Він призначений для спрощення Web – проектування і Web – розробки за рахунок мови розмітки, що забезпечує стандартизований і інтуїтивно зрозумілий інтерфейс.

HTML 5 надає розробнику можливості для секціонування і структуризації Web – сторінок, а також дозволяє створювати відокремлені компоненти, які не тільки забезпечують логічну організацію Web – сайту, але і надають йому можливості синдикації. [38][39]

Мова HTML 5 реалізує підхід до проектування Web – сайтів, заснований на відображенні інформації, оскільки він втілює саму суть відображення інформації – поділ і маркування інформації для спрощення її використання і розуміння. Саме в цьому полягає величезна семантична та естетична цінність HTML 5. [40]

HTML 5 надає дизайнерам і розробникам всіх рівнів можливості для надання в публічний доступ буквально будь-якого контенту – від простих текстів до мультимедійно насичених інтерактивних матеріалів.

HTML5 надає ефективні інструменти для управління даними, для малювання, для відтворення відео- та аудіоконтенту. Він полегшує розробку крос – браузерних Web – додатків, а також додатків для мобільних пристроїв.

HTML 5 відноситься до числа технологій, які стимулюють розвиток мобільних сервісів на основі хмарних обчислень. Крім того, HTML 5 сприяє підвищенню гнучкості - завдяки можливості створення вражаючих і інтерактивних Web – сайтів. І, нарешті, HTML5 пропонує нові теги і вдосконалення, в числі яких такі:

- елегантна структура;
- органи управління формами;
- API – інтерфейси.

Нові теги HTML 5 мають інтуїтивно – зрозумілі назви, які розкривають призначення і характер використання цих елементів. [41]

2.6.2 Вибір технології взаємодії клієнтської частини та сервера

Для забезпечення отримання інформації від серверу, компонування та обрахунку клієнтських даних, було вирішено використовувати JavaScript.

JavaScript є одним з найпростіших, універсальних та ефективних мов, які використовуються для розширення функціональності веб-сайтів.

JavaScript Development Services легко допомагає у візуальних ефектах на екрані, обробляє та обчислює дані на веб-сторінках. Також ця мова програмування допомагає розширювати функціональні можливості веб-сайтів, що використовують сторонні скрипти серед кількох інших зручних функцій.

Розглянемо основні переваги JavaScript:

- легкий у вивченні та використанні;
- є відносно швидкий для кінцевого користувача;
- має розширений функціонал та набір плагінів для роботи з безпосередньо веб – сторінками;
- не потребує компіляції;
- легкий для дебагу та тестування;
- не залежить від платформи, на якій використовується;
- має набір функцій для відслідковування подій зі сторони клієнтського інтерфейсу;
- має можливості процедурного програмування;
- високо стійкий до помилок програміста при написанні коду;
- Ядро мови JavaScript складається з певної кількості звичайних можливостей, які дозволяють робити наступне:
- зберігати дані всередині змінних;
- запускати код відповідно до певних подій, що відбуваються на веб – сторінці;
- виконувати обрахунки.

Ще більш цікавою є функціональність, створена поверх основної мови JavaScript. Так звані інтерфейси прикладного програмування (API) які надають вам додаткові можливості для використання в коді JavaScript.

API – це готові набори блоків коду, які дозволяють розробнику реалізовувати програми, які в іншому випадку було б важко або неможливо реалізувати.

API – інтерфейси браузера вбудовані в веб – браузер і можуть відображати дані з навколишнього комп'ютерного оточення або робити корисні складні речі, наприклад:

- API – інтерфейс DOM (Document Object Model) дозволяє маніпулювати HTML і CSS, створювати, видаляти і змінювати HTML, динамічно застосовувати нові стилі до сторінки;
- API геолокації витягує географічну інформацію. Так Google Maps може знайти ваше місце розташування і нанести його на карту;
- API Canvas і WebGL дозволяють створювати анімовані 2D і 3D-графіки;
- Аудіо та відео API, такі як HTMLMediaElement і WebRTC, дозволяють робити дійсно цікаві речі з мультимедійними файлами (програвання аудіо і відео прямо на веб-сторінці).

Код JavaScript виконується JavaScript – движком браузера, після того як код HTML і CSS був оброблений і сформований в веб-сторінку. Це гарантує, що структура і стиль сторінки вже сформовані до моменту запуску JavaScript.

Кожна вкладка браузера являє собою окрему коробку для запуску коду (технічною мовою, ці коробки називаються "середовищами виконання") – це означає, що в більшості випадків код на кожній вкладці запускається повністю окремо, а код однієї вкладки не може на пряму впливати на код інший вкладки або на іншому веб-сайті. Це хороша міра безпеки.

JavaScript є мовою, що інтерпретується – код запускається зверху вниз і результат запуску негайно повертається. Вам не потрібно перетворювати код в іншу форму, перед запуском в браузері.

З іншого боку, компільовані мови перетворюються (компілюються) в іншу форму, перш ніж вони будуть запущені комп'ютером. Наприклад, C / C ++ компілюються в мову асемблера, який потім запускається комп'ютером.

За допомогою JavaScript можна писати гнучкий та динамічний код. Це відноситься до можливості поновлення відображення веб – сторінки, щоб показувати різний контент в різних обставинах, генеруючи новий його в міру необхідності, наприклад створює нову HTML таблицю, вставляючи в неї дані

отримані з сервера, а потім відображає таблицю на веб-сторінці, яку бачить користувач.

Отже як висновок – це зручна та проста мова програмування, яку доволі легко інтегрувати у проект та за допомогою її можливостей, набору фреймворків та плагінів, розробляти гнучкий та динамічний клієнтський інтерфейс. [42]

2.6.3 Вибір технологій для розробки одно-сторінкових додатків

Для швидкої та гнучкої інтеграції даних, отриманих з серверу, на веб – сторінку клієнта, під час розробки клієнтської частини планується використання зручного та багатофункціонального фреймворку для JavaScript – Angular JS.

AngularJS це фреймворк з відкритим програмним кодом, який розробляє Google. Він призначений для розробки односторінкових застосунків, що складаються з одної HTML сторінки з CSS і JavaScript. Його мета — розширення браузерних застосунків на основі шаблону модель – вид – контролер (MVC), а також спрощення їх тестування та розробки. [43]

Angular дозволяє з «коробки» створювати великі і відносно складні бізнес – логіки додатка.

Основні переваги:

- підтримка Google, Microsoft;
- інструменти розробника (CLI);
- єдина структура проекту;
- TypeScript з «коробки»;
- реактивний програмування з RxJS;
- єдиний фреймворк з Dependency Injection з «коробки»;
- шаблони, засновані на розширенні HTML;
- кросбраузерності Shadow DOM з «коробки» (або його емуляція);

- кросбраузерна підтримка HTTP, WebSockets, Service Workers;
- не потрібно нічого додатково налаштовувати. Більше ніяких обгорток.

Для розробки продуктивних і швидко працюючих додатків будь – якого рівня складності, можна використовувати наступні концепції:

Form Builder. Використовується для розробки воістину складних форм, для роботи з цією концепцією, слід знати реактивні форми, точніше принципово забути про декларативні форми. Ось один з хороших прикладів (реактивна форма + валідація); [44]

Change Detection. Так як Angular за замовчуванням використовує двостороннє зв'язування моделі даних, то при роботі з великим об'ємом таких даних ваші програми будуть працювати повільніше, тому в деяких випадках варто подбати про правильну стратегію виявлення змін.

Routing. Одне з основних явищ у розробці веб – додатків. Тут важливо розуміти, що маршрутизація так само, як і компоненти, має свій життєвий цикл, розуміючи це, Angular надає великі можливості при написанні веб - додатків. Ще варто відзначити: якщо на якійсь із маршрутів ви вішаєте модуль, а не компонент, який відповідає за відображення сторінки по цьому маршруту, то модуль буде довантажувати компоненти на сторінці на вимогу.

Annotations. Декоратори є специфікацією EcmaScript і коли браузери почнуть підтримувати їх, вони будуть нативно виконуватися в browser runtime. Декоратори вельми корисні і забезпечують досить високу читабельність коду.

Observables. Незабаром Observables будуть специфікацією EcmaScript і все це буде нативно підтримуватися в браузерах. З точки зору теорії, розкрити поняття Observer (спостерігач) - це поведінковий шаблон проектування, також відомий як «залежний» (Dependents). Він створює механізм у класі, який дозволяє отримувати примірник об'єкта цього класу, оповіщення від інших об'єктів про зміну їх стану, тим самим спостерігаючи за ними.

Shadow DOM. Засіб для створення окремого DOM – дерева всередині елемента, яке не видно зовні без застосування спеціальних методів, є специфікацією W3C.

Це зручний спосіб створення ізольованих веб – компонентів. Технічно, якби браузери вже сьогодні підтримували багато концепцій, які використовує Angular, нам не потрібні були б транспайлери і інші системи збирання інформації. Все, що ми б писали на Angular, працювало б нативно.

Цей фреймворк є дійсно дуже легкий у використанні, та надає більше прямий та безпосередній доступ до контенту HTML сторінки.

2.6.4 Вибір інструментів оформлення клієнтської частини

Для створення графічного інтерфейсу, разом у парі з HTML в розробці системи прийняття рішень, щодо підбору віртуальної машини, будуть використовуватись каскадні таблиці стилів CSS, та фреймворк Bootstrap для використання готових компонентів. [45]

CSS (Cascading Style Sheets) – це загально прийнятий стандарт, який задає форму представлення даних у браузерах. Мова розмітки HTML подає браузеру структуру HTML – документу, а таблиці стилів задають стилізацію цього документу.

Стиль представляє собою набір команд, які будуть застосовані до елементів HTML – документу і будуть визначати стиль його відображення. До стилю входять такі дизайнерські елементи: фон, шрифт, кольори посилань, текст, поля і тд.

Таблиця стилів – це сукупність стилів, які можна застосувати до гіпертекстових документів.

Документ, як правило, являє собою текстовий файл, структурований з використанням найпопулярнішої мови розмітки - HTML, або інших, таких як SVG або XML.

Представити документ користувачеві – означає перетворити його у форму, яка може бути використана вашою аудиторією. Браузери, такі як Firefox, Chrome або Edge, призначені для візуального подання документів, наприклад, на екран комп'ютера, проектора чи принтера.

CSS можна використовувати для базового стилю тексту документа - наприклад, для зміни кольору та розміру заголовків та посилань. Він може бути використаний для створення макета - наприклад, перетворення однієї колонки тексту в макет з основною областю вмісту та бічною панеллю для відповідної інформації. Його навіть можна використовувати для таких ефектів, як анімація.

CSS відкриває нам нові, раніше невідомі і недоступні межі. З їх допомогою ми отримуємо можливість більш вишукано оформити свій веб – сайт, надати йому нових рис і «симпатичне обличчя».

Синтаксис CSS досить простий у вивченні, тому освоєння каскадних стилів є справою досить легкою.

Використання таблиць стилів можна розглянути на простому прикладі.

Візьмемо типову таблицю HTML і встановимо атрибут border = "1". Після інтерпретації даного коду браузером ми отримаємо самий звичайний результат – таблицю з простою рамкою.

Ось тут і приходять на допомогу CSS стилі, за допомогою яких, можна оформити будь – яку таблицю на свій смак. Це можна реалізувати або за допомогою готових стилів, використовуючи фреймворки, або за допомогою власно – розробленого стилю. Одним із таких фреймворків – є популярний Bootstrap.

Bootstrap – це фреймворк, який допоможе швидко і легко створювати статичні веб-сайти і веб-додатки.

Цей фреймворк отримав дуже широке поширення завдяки масі факторів, головним з яких було використання класів для швидкої побудови адаптивної сітки. Також Bootstrap надає в своєму складі набір плагінів jQuery, що дозволяє простим

додаванням коду вставити такі інтерактивні елементи, як карусель (або слайдер), вспливаючи підказки, модальні вікна та ін.

Основною перевагою використання Bootstrap є Less – це динамічні стилі, які істотно розширюють можливості CSS. З їх допомогою розробник може створювати змінні, колонки, керувати кольорами і т.д. Для інтеграції Less елементів достатньо вставити код в сторінку і він працюватиме. Завдяки цим компонентам, можна налаштувати стиль сторінки на свій смак.

2.7 Висновки до розділу

У даному розділі було проведено аналіз інформаційних систем та було встановлено, що клієнт-серверна частина системи моніторингу паркувальних майданчиків, відповідає за забезпечення усіх заходів, щодо безпеки, збір статистики по заданим параметрам, перевірку зібраної статистики в автоматичному режимі, створення пакетів даних по паркувальних майданчиках в форматі GEOJSON та їх перевірку, автоматичну доставку пакетів до інформаційної панелі.

Далі було проведено проектування модулів системи моніторингу а саме: модулю збору статистичних даних та модулю, який відповідаю за перевірку зібраних даних.

Беручи до уваги розглянуті технічні засоби реалізації серверної та клієнтської частини програми, вибрано найоптимальніші рішення, для забезпечення кросплатформної інтеграції, швидкодії та простоти реалізації програми. Дані висновки побудовані на базі порівнянь та переваг існуючих реалізацій.

Розглянуті технічні засоби використовувалися під час розробки модулів, та будуть використовуватись під час розробки системи моніторингу паркувальних майданчиків.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Розробка базових компонентів функціоналу програми

Для розробки серверної частини буде використовуватись мова програмування Java та Spring Framework.

Для розробки клієнтської частини – JavaScript (Angular), графічний інтерфейс – HTML, CSS (Bootstrap). Детальніше про технології, які будуть використовуватись під час розробки програми описано у розділі 2.

Під час проектування архітектури основних модулів програми, було вирішено дотримуватись вимог патерну MVC.

Основні вимоги, щодо взаємодії компонентів патерну MVC, та невід'ємні частини реалізації, наведені на Рисунку 3.1.

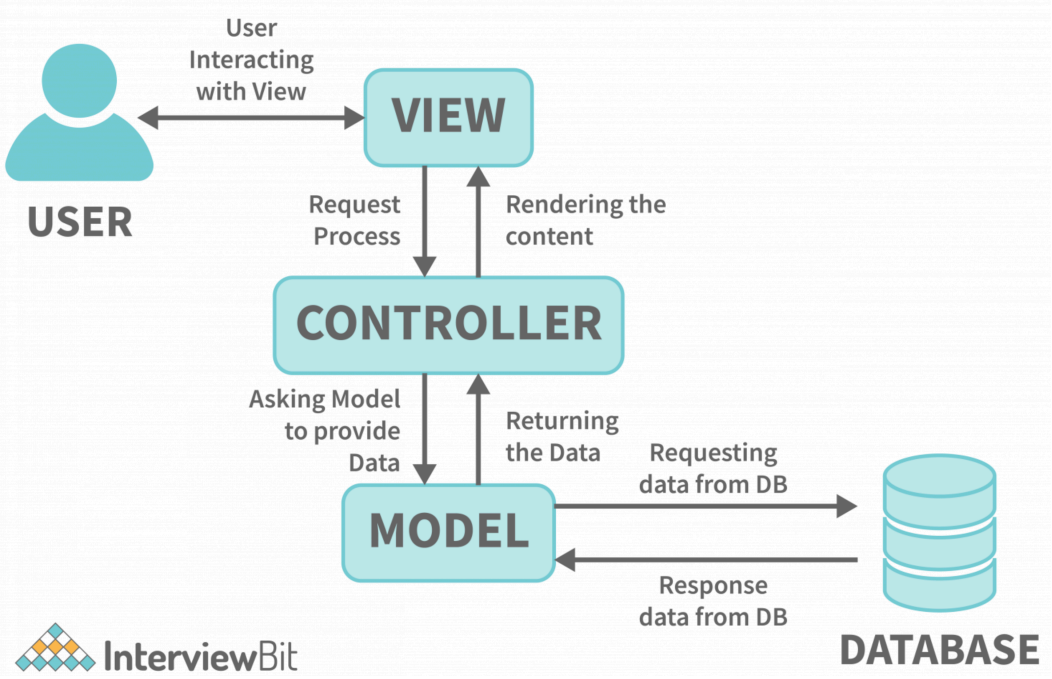


Рисунок 3.1 – Структурні компоненти патерну MVC, та зв'язки взаємодії між ними

Архітектура програма буде складатися з набору класів взаємодіючих один з одним.

Усі класи серверної частини поділяються на такі типи:

- сервіс – класи, містять у собі набір функцій, які реалізують бізнес логіку програми;
- модель – класи, представляють собою об’єкт, який створюється для збереження інформації у межах заданого циклу програми, та не містять функцій для роботи з логікою;
- клієнт – класи, містять у собі набір функцій для отримання результатів з зовнішніх ресурсів по заданому URL;
- контролер – класи, представляють собою набір кінцевих точок, для доступу до основних джерел інформації зі сторони клієнта; [46]

```
private List<EPInventory> generateParko(String cityPrefix, Set<String> types) {
    Integer countryId = cityDao.getCountryId(cityPrefix);
    List<ClusterPermits> clusterPermits = clusterRepository.getAll(countryId);
    return inventorySubLinkService.getAll(cityPrefix).parallelStream() Stream<SubLink>
            .map(subLink -> createParkoInventory(subLink, clusterPermits, types, countryId)) Stream<EPInventory>
            .filter(inventory -> containsAnyIgnoreCase(inventory.getType(), types.toArray(new String[0])))
            .toList();
}

private EPInventory createParkoInventory(SubLink subLink, List<ClusterPermits> clusters, Set<String> types, Integer countryId) {
    ClusterPermits cluster = findSubLinkCluster(subLink, clusters);
    return EPInventory.builder() EPInventoryBuilder<capture of ?, capture of ?>
            .id(subLink.id()) capture of ?
            .segmentId(subLink.segmentId())
            .parkingSpots(subLink.spots())
            .geometry(subLink.way())
            .streetName(subLink.streetName())
            .type(getSubLinkType(cluster, types))
            .sign(cluster.vectorizedSignId())
            .signUrl(buildSignUrl(SIGNS_BUCKET, countryId.toString(), cluster.vectorizedSignId()))
            .areaId(subLink.areaId())
            .build();
}
```

Рисунок 3.2 – Функції отримання та створення пакетів інвентаризації

Розглянемо декілька прикладів функцій, які будуть реалізовувати основний функціонал програми.

Функції отримання та створення пакетів інвентаризації наведені на Рисунку 3.2.

Для початку нам необхідно визначити місто для якого ми будемо створювати пакет (метод getCountryId(String cityPrefix)). Після того як місто було визначено, потрібно отримати колекцію статистичних даних з бази даних по місту за допомогою методу getAll(int countryId). Далі потрібно згрупувати дані, які ми отримали з бази даних в об'єкт EPInventory, який має в особі набір всіх необхідних параметрів, таких як: ідентифікатор сегмента, кількість місці для паркування, геометрія паркувального місця, назву вулиці, тип паркувального майданчика, назву файлу знаку, посилання на знак та ідентифікатор паркувальної ареї. На рисунку 3.3 наведено діаграму класів і загальні ієрархію класів Inventory.

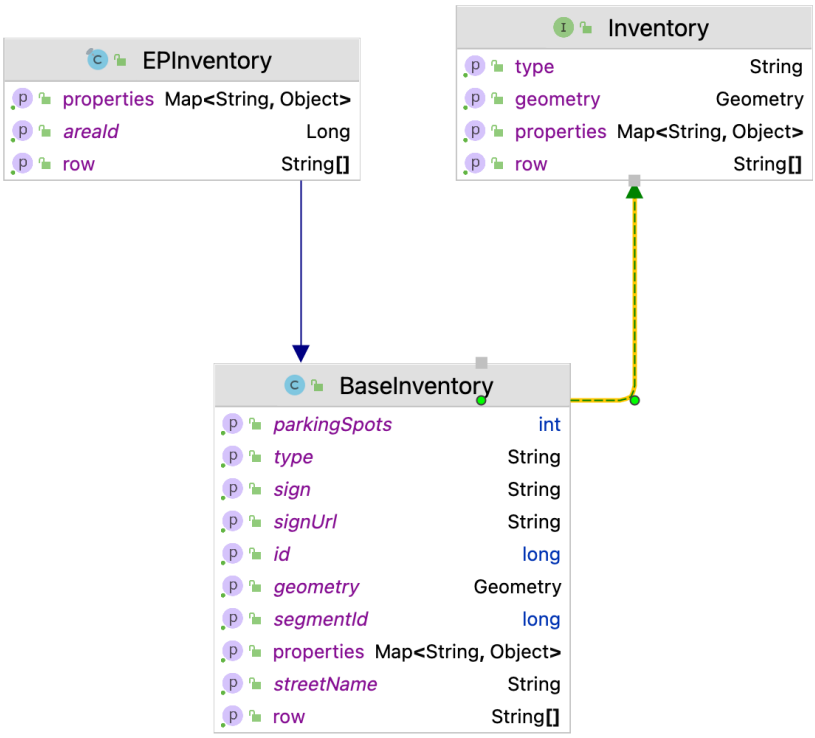


Рисунок 3.3 – Діаграма класів Inventory

Для зв'язки серверної частини з клієнтською використовуються Rest – контролери, у даному випадку це класи, які помічені анотацією @RestController, яка входить у пакет Spring Framework.

Приклад Rest – контролеру для запуску процесу генерації пакетів інвентаризації наведений на Рисунку 3.4.

```
@PostMapping("generate/{cityPrefix}")
public ResponseEntity<Long> startInventoryDataGenerationProcess(@PathVariable String cityPrefix,
        @RequestParam ParkingDataSource parkingDataSource, @RequestBody Set<String> types) {
    long processId = processService.runProcessAsync(
            () -> inventoryService.startGenerationProcess(cityPrefix, parkingDataSource, types),
            ProcessType.INVENTORY_GENERATION
    );
    return ResponseEntity.status(HttpStatus.ACCEPTED).body(processId);
}
```

Рисунок 3.4 — Контролер для запуску процесу генерації пакетів інвентаризації

Усі ресурси системи будуть надаватись лише авторизованим користувачам. Авторизація відбуватиметься за допомогою налаштування конфігурацій Spring Security.

Spring Security являє собою це фреймворк, який забезпечує аутентифікацію та авторизацію користувача до закритих API в межах системи. Даний фреймворк зручний у використанні завдяки ініціалізації багатьох базових класів через автоконфігурацію Spring. Не менш важливим є гнучкий вибір налаштувань, які може здійснити розробник наприклад додати власний фільтр який буде перевіряти валідність JWT токену чи модифікувати доступ до деяких ресурсів за допомогою функції permitAll(). Ще одною цікавою особливість є наявність готових інтерфейсів, для перевірки даних доступу користувачів, без використання кастомних реалізаці. Також Spring Security надає автоматично згенеровану сторінку авторизації, яку у підсумку можна переробити під свій дизайн.

Структурна UML діаграма патерну MVC наведена у додатку Е. UML діаграма класів, розробленого модулю створення пакетів інвентаризації, наведена у додатку Є.

3.2 Перевірка створених пакетів інвентаризації

Пакети інвентаризації, це готовий до інтеграції з інформаційною панелю продукт, але перед відправкою пакету потрібно провести візуальну перевірку пакету.

Далі пакет зберігається в системі, і доступний для перевірки. Для цього необхідно ввести місто пакети для, якого ми хочемо перевірити і вибрати створений пакет в списку доступних пакетів. На рисунку 3.5 наведено приклад панелі для пошуку створеного пакету.

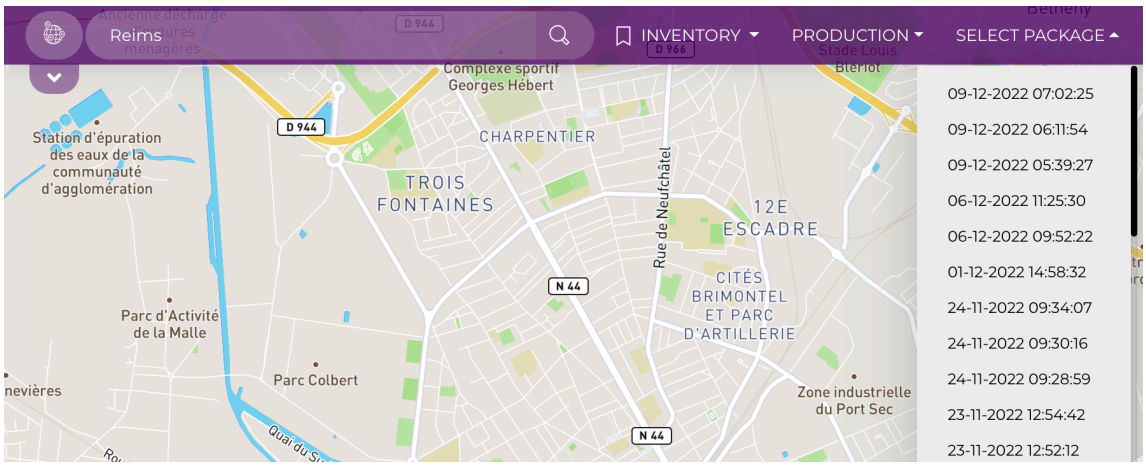


Рисунок 3.5 – Панель для пошуку створеного пакету інвентаризації

Після вибору пакету, потрібно натиснути на кнопку Load для того, щоб показати пакет на мапі. На рисунку 3.6 наведено мапу з даними інвентаризації.

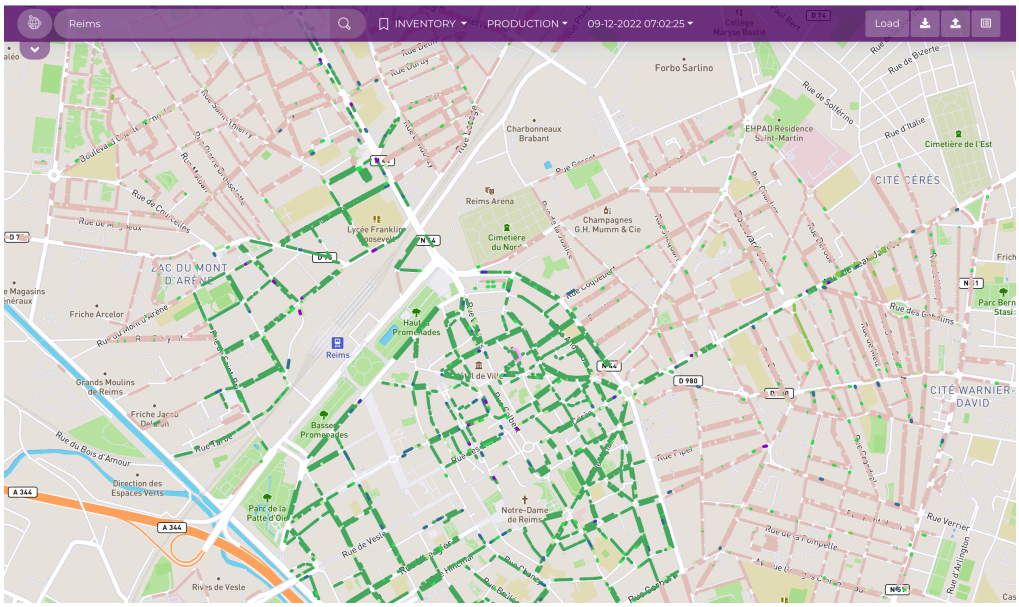


Рисунок 3.6 – Мапа з даними інвентаризації.

Для того щоб перевірити дані інвентаризації по конкретною майданчику потрібно знайти його на мапі і клікнути на нього, далі буде виведена основна статистика по ньому. На рисунку 3.7 зображено панель з статистикою паркувального майданчика.

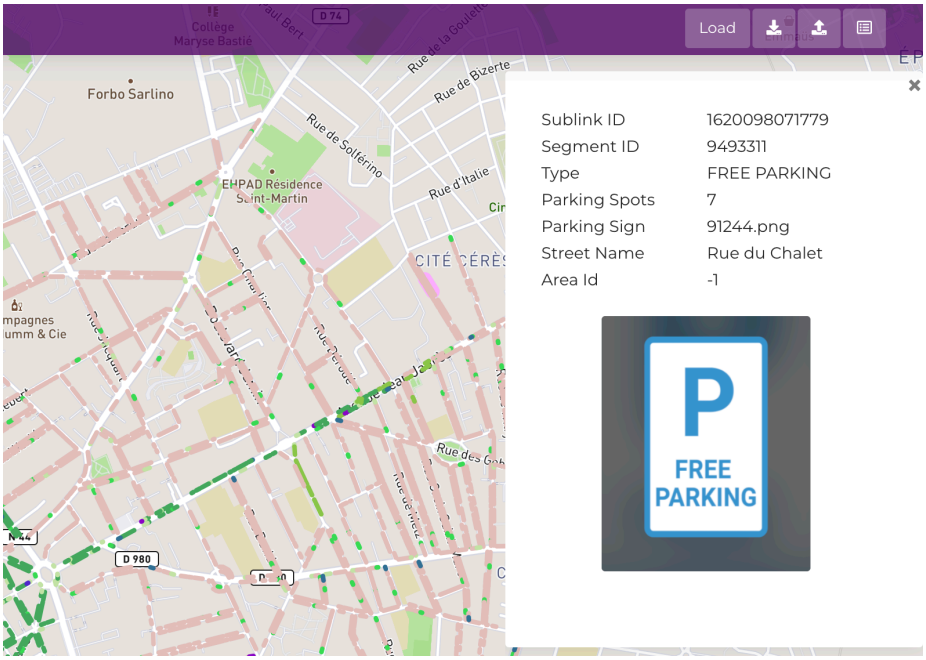


Рисунок 3.7 – Панель з статистикою паркувального майданчика.

Дані параметри є базовими. Додаткові параметри обумовлюються при інтеграції сервісів до зовнішніх ресурсів.

3.3 Тестування програми

Тестування системи, яка проектується в межах даної роботи, буде виконуватись за допомогою unit тестів за допомогою фреймворка Mockito.

Використання Mockito обумовлюється такими перевагами:

- створення «заглушок» для класів і інтерфейсів;
- перевірка виклика методу і значень переданих методу в якості параметрів;
- використання концепції «часткової заглушки», при якій заглушка створюється на клас з визначенням поведінки, необхідної для деяких методів класу;
- підключення до реального класу «шпигуна» Spy для контролю виклику методів.

Також цей фреймворк має набір методів визначення поведінки функції в залежності від тих чи інших параметрів.

За допомогою Mockito можна робити підмін реальних значень, та викликати шаблонні методи, замість реальних. Для цього у бібліотеці цього фреймворку є функція when() та verify().

З їх допомогою можливо не тільки робити підміну значень та функцій, а й обраховувати час виконання методу, підраховувати скільки разів даний метод був викликаний і при яких умовах.

Приклад тесту з використанням фреймворку Mockito наведений на Рисунку 3.8.

```
@Test
void testStartGenerationProcess() {
    List<File> packages = List.of(new File( pathname: tmpDir + "test.json"), new File( pathname: tmpDir + "test.zip"), new File(
    long expectedFolder = 999999L;
    List<? extends Inventory> inventories = List.of(EASY_PARK_INVENTORY);
    when(timeService.currentTimeMillis()).thenReturn(expectedFolder);
    doReturn(inventories).when(inventoryGenerationService).generate(CITY_PREFIX, PARKO_DATA_SOURCE, INVENTORY_TYPES);
    when(inventoryCreationService.createPackages(CITY_PREFIX, PARKO_DATA_SOURCE, inventories)).thenReturn(packages);
    inventoryService.startGenerationProcess(CITY_PREFIX, PARKO_DATA_SOURCE, INVENTORY_TYPES);
    verify(inventoryStorageService).uploadPackages(CITY_PREFIX, PARKO_DATA_SOURCE, String.valueOf(expectedFolder), packages);
    verify(inventoryRepository).save(CITY_PREFIX, String.valueOf(expectedFolder), PARKO_DATA_SOURCE, INVENTORY_TYPES);
}
```

Рисунок 3.8 – Приклад unit тесту з використанням Mockito

Кожен тест помічається анотацією @Test, яка вказує unit на те, що даний метод є тестовий. Кожен тестовий метод можна запускати окремо, не запускаючи увесь проект. Усі тести запускаються за допомогою класу – стартера MockitoJUnitRunner, який уже входить у пакет Spring.

Для прикладу, запустим усі тести класу InventoryServiceImpl, який був розроблений у межах модулю створення пакетів інвентаризації, результат виконання тестів наведено на Рисунку 3.9.

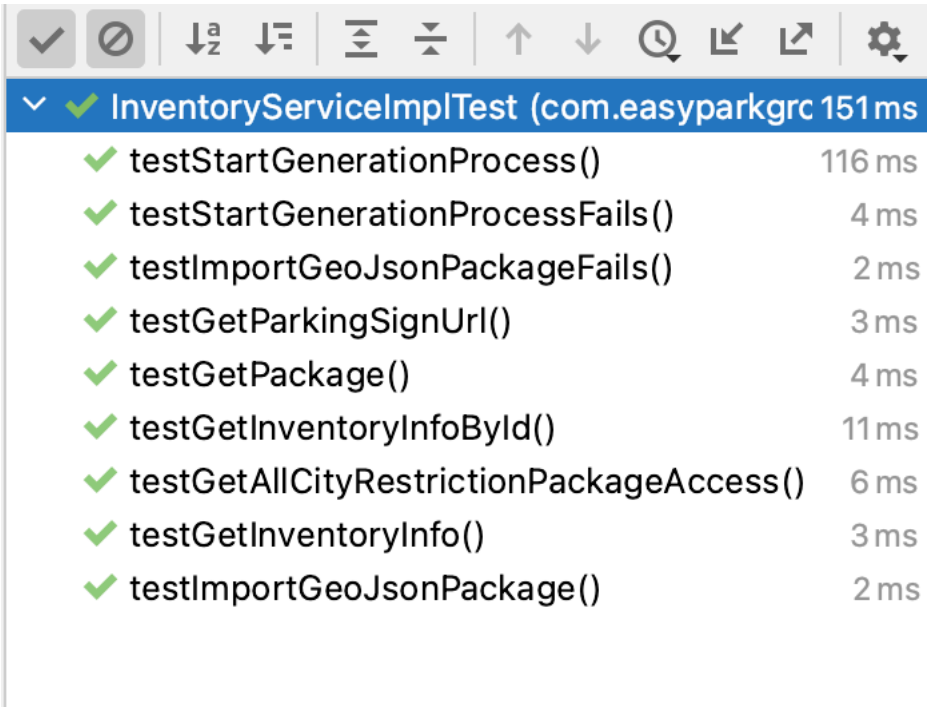


Рисунок 3.9 – Результат виконання тестів класу InventoryServiceImpl

Після покриття тестами певної частини коду, ми можемо дивитись на відсоток покритих методів.

Наприклад прокриємо усі методи класу InventoryServiceImpl тестами, результат покриття зображено на Рисунку 3.10.

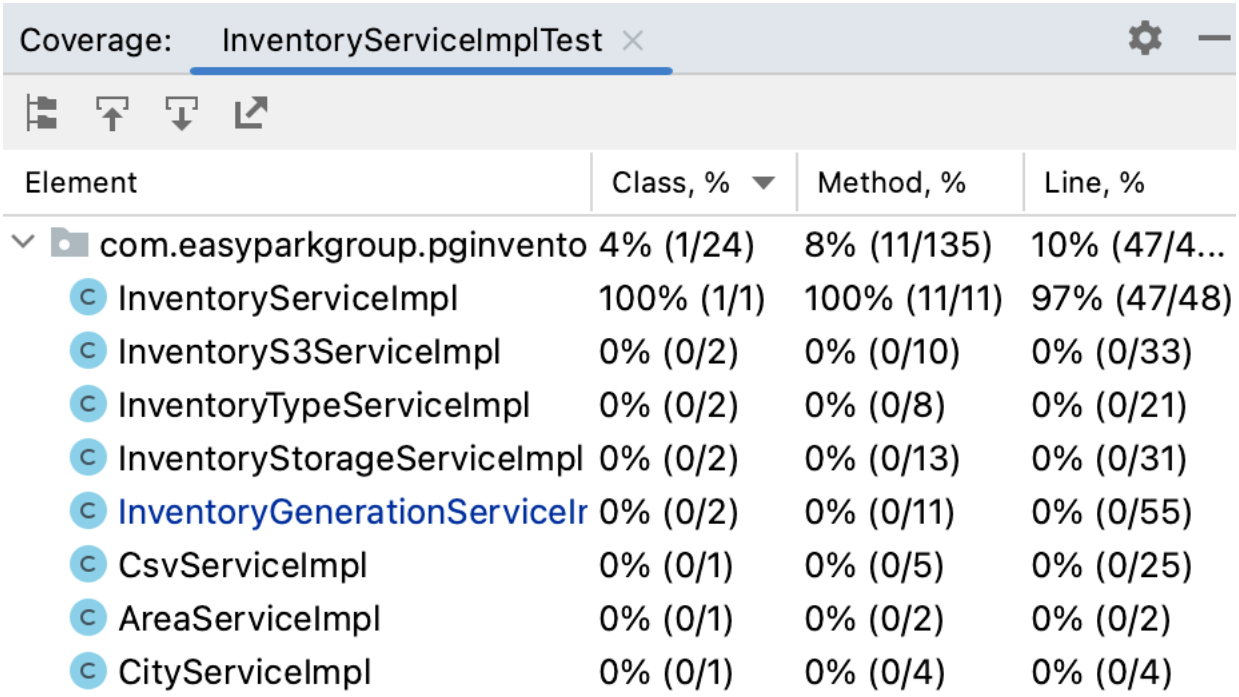


Рисунок 3.10 – Приклад покриття тестами усіх методів класу

Як бачимо на рисунку, клас покритий тестами на 100%. За допомогою цього Covarage плагіна, ми можемо просто відслідковувати класи та методи, які повністю, або частково не покриті тестами. Це доволі зручно при великому обсязі класів. Загально прийнятий мінімальний відсоток покриття коду тестами складає біля 80%.

3.4 Висновки до розділу

У даному розділі розглянуто основні етапи розробки базових компонентів клієнт-серверної частини системи моніторингу паркувальних майданчиків.

Розроблено UML діаграму класів модулю створення пакетів інвентаризації, доведена доцільність використання архітектурного патерну MVC.

Розроблено базові компоненти серверної частини програми за допомогою мови програмування JAVA, та фреймворку Spring Boot. Також уся бізнес логіка, яка відповідає за створення пакетів інвентаризації покрита unit тестами.

Лістинг програми розробленого модулю моніторингу інтернет сервісів наведено у додатку З.

4 ЕКОНОМІЧНА ЧАСТИНА

Науково-технічна розробка має право на існування та впровадження, якщо вона відповідає вимогам часу, як в напрямку науково-технічного прогресу та і в плані економіки. Тому для науково-дослідної роботи необхідно оцінювати економічну ефективність результатів виконаної роботи.

4.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення комерційного і технологічного аудиту дослідження за темою «Розробка клієнт-серверної системи моніторингу паркувальних майданчиків (Серверна частина)» є оцінювання науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням 5-ти бальної системи оцінювання за 12-ма критеріями [52].

Таблиця 4.1 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт (ПІБ, посада)		
	1	2	3
	Бали:		
1. Технічна здійсненність концепції	5	5	5
2. Ринкові переваги (наявність аналогів)	1	1	1
3. Ринкові переваги (ціна продукту)	2	3	2
4. Ринкові переваги (технічні властивості)	1	2	2
5. Ринкові переваги (експлуатаційні витрати)	2	2	2
6. Ринкові перспективи (розмір ринку)	3	3	3
7. Ринкові перспективи (конкуренція)	3	2	3
8. Практична здійсненність (наявність фахівців)	5	5	5
9. Практична здійсненність (наявність фінансів)	3	4	3

10. Практична здійсненність (необхідність нових матеріалів)	5	5	5
11. Практична здійсненність (термін реалізації)	4	4	4
12. Практична здійсненність (розробка документів)	4	4	4
Сума балів	38	40	39
Середньоарифметична сума балів $СБ_с$	39,0		

За результатами розрахунків, наведених в таблиці 4.1, зробимо висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При цьому використаємо рекомендації, наведені в підручнику[52].

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Розробка клієнт-серверної системи моніторингу паркувальних майданчиків (Серверна частина)» становить 39,0 бала, що, відповідно до [52], свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вище середнього).

4.2 Розрахунок узагальненого коефіцієнта якості розробки

Узагальнений коефіцієнт якості ($B_н$) для нового технічного рішення розрахуємо за формулою [Кавецький практикум 2016]:

$$B_н = \sum_{i=1}^{k} \alpha_i \cdot \beta_i , \qquad (4.1)$$

де k – кількість найбільш важливих технічних показників, які впливають на якість нового технічного рішення;

α_i – коефіцієнт, який враховує питому вагу *і-го* технічного показника в загальній якості розробки. Коефіцієнт α_i визначається експертним шляхом і при цьому має виконуватись умова $\sum_{i=1}^{k} \alpha_i = 1$;

β_i – відносне значення *і-го* технічного показника якості нової розробки.

Результати порівняння зведемо до таблиці 4.2.

Таблиця 4.2 – Порівняння основних параметрів розробки та аналога.

Показники (параметри)	Одиниця вимірю-вання	Аналог	Проектований продукт	Відношення параметрів нової розробки до аналога	Питома вага показника
Об'єм памяті slim дистрибутивів для запуску серверу (на сервері)	МБ	430	120	3,58	0,2
Об'єм серверу	МБ	120	20	6	0,25
Кількість таблиць бази даних	шт	210	43	4,88	0,2
Об'єм бази даних	ГБ	1000	320	0,32	0,1
Кількість мікросервісів	шт	50	15	3,33	0,25

Узагальнений коефіцієнт якості ($B_н$) для нового технічного рішення складе:

$$B_н = \sum_{i=1}^{k} \alpha_i \cdot \beta_i = 3,58 \cdot 0,2 + 6 \cdot 0,25 + 4,88 \cdot 0,2 + 0,32 \cdot 0,1 + 3,33 \cdot 0,25 = 4,06.$$

Отже за технічними параметрами, згідно узагальненого коефіцієнту якості розробки, науково-технічна розробка переважає існуючі аналоги приблизно в 4,06 рази.

4.3 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи на тему «Розробка клієнт-серверної системи моніторингу паркувальних майданчиків (Серверна частина)», під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуємо за відповідними статтями.

4.3.1 Витрати на оплату праці

Основна заробітна плата дослідників

Витрати на основну заробітну плату дослідників ($З_о$) розраховуємо у відповідності до посадових окладів працівників, за формулою [52]:

$$З_о = \sum_{i=1}^{k} \frac{M_{ni} \cdot t_i}{T_p}, \quad (4.2)$$

де k – кількість посад дослідників залучених до процесу досліджень;

M_{ni} – місячний посадовий оклад конкретного дослідника, грн;

t_i – число днів роботи конкретного дослідника, дн.;

T_p – середнє число робочих днів в місяці, T_p=24 дні.

$$З_о = 16700{,}00 \cdot 24 / 24 = 16700{,}00 \; грн.$$

Проведені розрахунки зведемо до таблиці.

Таблиця 4.3 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Керівник НДР	16700,00	695,83	24	16700,00
Аналітик з розробки інформаційних систем	15200,00	633,33	18	11400,00
Інженер-програміст 1-ї категрії	15500,00	645,83	24	15500,00
Консультант (менеджер-адміністратор)	15000,00	625,00	5	3125,00
Технік 1-ї категорії	7450,00	310,42	20	6208,33
Всього				52933,33

Основна заробітна плата робітників

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт НДР на тему «Розробка клієнт-серверної системи моніторингу паркувальних майданчиків (Серверна частина)» розраховуємо за формулою:

$$З_p = \sum_{i=1}^{n} C_i \cdot t_i, \quad (4.3)$$

де C_i – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою:

$$C_i = \frac{М_М \cdot K_i \cdot K_c}{T_p \cdot t_{зм}}, \quad (4.4)$$

де $М_М$ – розмір мінімальної місячної заробітної плати, приймемо $Мм$=6700,00 грн;

K_i – коефіцієнт міжкваліфікаційного співвідношення (табл. Б.2, додаток Б) [52];

K_c – мінімальний коефіцієнт співвідношень місячних тарифних ставок;

T_p – середнє число робочих днів в місяці, приблизно T_p = 24 дн;

$t_{зм}$ – тривалість зміни, год.

$C_1 = 6700{,}00 \cdot 1{,}10 \cdot 1{,}65 / (24 \cdot 8) = 63{,}34$ *грн.*
$З_{р1} = 63{,}34 \cdot 6{,}00 = 380{,}02$ *грн.*

Таблиця 4.4 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника грн
Установка обчислювального обладнання для проведення досліджень	6,00	2	1,10	63,34	380,02
Підготовка робочого місця розробника програмного забезпечення	8,00	4	1,50	86,37	690,94
Інсталяція програмного забезпечення розробки інформаційної системи	5,67	5	1,70	97,88	555,00

Введення дослідних баз даних інформаційного ресурсу	15,20	3	1,35	77,73	1181,50
Ведення кодів модульних блоків інформаційної системи	10,00	3	1,35	77,73	777,30
Компіляція програмних модулів інформаційної системи моніторингу паркувальних майданчиків	6,30	5	1,70	97,88	616,66
Налагодження блоків інформаційної системи	4,30	6	2,00	115,16	495,17
Тестування серверної частини інформаційної системи	12,00	3	1,35	77,73	932,77
Всього					5629,36

Додаткова заробітна плата дослідників та робітників

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (4.5)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати. Приймемо 11%.

$З_{дод}$ = (52933,33 + 5629,36) · 11 / 100% = 6441,90 грн.

4.3.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%} \quad (4.6)$$

де $Н_{зп}$ – норма нарахування на заробітну плату. Приймаємо 22%.

Зн = (52933,33 + 5629,36 + 6441,90) · 22 / 100% = 14301,01 грн.

4.3.3 Сировина та матеріали

Витрати на матеріали (*М*), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою:

$$M = \sum_{j=1}^{n} H_j \cdot Ц_j \cdot К_j - \sum_{j=1}^{n} B_j \cdot Ц_{вj}, \quad (4.7)$$

де H_j – норма витрат матеріалу *j*-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу *j*-го найменування, грн/кг;

$К_j$ – коефіцієнт транспортних витрат, ($К_j$ = *1,1 ... 1,15*);

B_j – маса відходів *j*-го найменування, кг;

$Ц_{вj}$ – вартість відходів *j*-го найменування, грн/кг.

$М_1 = 3,0 \cdot 273,00 \cdot 1,1 - 0 \cdot 0 = 900,90$ *грн.*

Проведені розрахунки зведемо до таблиці.

Таблиця 4.5 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за 1 кг, грн	Норма витрат, кг	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн
Багатофункціональний білий офісний папір FAXE-500 A4	273,00	3,0	0	0	900,90
Папір для записів FAXE 70 A5-250	148,00	4,0	0	0	651,20
Органайзер офісний OFFICE 100	180,00	4,0	0	0	792,00
Набір офісний DATUM X-2	190,00	3,0	0	0	627,00
Картридж для принтера HP-2100	1120,00	1,0	0	0	1232,00
Диск оптичний OPTIMA CD	22,50	2,0	0	0	49,50
Flesh-пам'ять GOODRAM 64 C10A	632,00	1,0	0	0	695,20
Всього					4947,80

4.3.4 Розрахунок витрат на комплектуючі

Витрати на комплектуючі *($К_в$)*, які використовують при проведенні НДР на тему «Розробка клієнт-серверної системи моніторингу паркувальних майданчиків (Серверна частина)» відсутні.

4.3.5 Спецустаткування для наукових (експериментальних) робіт

Балансову вартість спецустаткування розраховуємо за формулою:

$$B_{спец} = \sum_{i=1}^{k} Ц_i \cdot C_{пр.i} \cdot K_i \ , \qquad (4.8)$$

де $Ц_i$ – ціна придбання одиниці спецустаткування даного виду, марки, грн;

$C_{пр.i}$ –кількість одиниць устаткування відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує доставку, монтаж, налагодження устаткування тощо, ($K_i = 1,10...1,12$);

k – кількість найменувань устаткування.

$B_{спец} = 42560,00 \cdot 1 \cdot 1,12 = 47667,20$ *грн.*

Отримані результати зведемо до таблиці:

Таблиця 4.6 – Витрати на придбання спецустаткування по кожному виду

Найменування устаткування	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Сервер - Компютер Expert PC Balance (I91F8H1S115E429)	1	42560,00	47667,20
Всього			47667,20

4.3.6 Програмне забезпечення для наукових (експериментальних) робіт

Балансову вартість програмного забезпечення розраховуємо за формулою:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.i} \cdot K_i \ , \qquad (4.9)$$

де $Ц_{iпрг}$ – ціна придбання одиниці програмного засобу даного виду, грн;

$C_{прг.і}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

$K_і$ – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, ($K_і$ = *1,10…1,12);*

k – кількість найменувань програмних засобів.

$B_{прг}$ = *6250,00 · 1 · 1,11 = 6937,50 грн.*

Отримані результати зведемо до таблиці:

Таблиця 4.7 – Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Емулятор серверу для моделювання поведінки інформаційної системи	1	6250,00	6937,50
Всього			6937,50

4.3.7 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12}, \quad (4.10)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$A_{обл}$ = *(26800,00 · 1) / (3 · 12) = 744,44 грн.*

Проведені розрахунки зведемо до таблиці.

Таблиця 4.8 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Персональний комп'ютер розробника ПЗ	26800,00	3	1	744,44
Персональний комп'ютер розробника інформаційних систем	23120,00	3	1	642,22
Робоче місце інженера-програміста	8500,00	5	1	141,67
Робоче місце аналітика	6800,00	5	1	113,33
Пристрої передачі даних	7000,00	4	1	145,83
Оргтехніка	9200,00	5	1	153,33
Приміщення лабораторії розробки ІС	780300,00	20	1	3251,25
ОС Windows 11	8450,00	2	1	1363,75
Прикладний пакет Microsoft Office 2019	7890,00	2	1	1765,00
Всього				8320,83

4.3.8 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію (B_e) розраховуємо за формулою:

$$B_e = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot Ц_e \cdot K_{вni}}{\eta_i}, \quad (4.11)$$

де W_{yi} – встановлена потужність обладнання на визначеному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_е$ – вартість 1 кВт-години електроенергії, грн; приймемо $Ц_е = 6,15$ грн;

$К_{впі}$ – коефіцієнт, що враховує використання потужності, $К_{впі} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

$В_е = 0,45 \cdot 190,0 \cdot 6,15 \cdot 0,95 / 0,97 = 525,83$ грн.

Проведені розрахунки зведемо до таблиці.

Таблиця 4.9 – Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Персональний комп'ютер розробника ПЗ	0,45	190,0	525,83
Персональний комп'ютер розробника інформаційних систем	0,32	160,0	314,88
Робоче місце інженера-програміста	0,10	160,0	98,40
Робоче місце аналітика	0,10	40,0	24,60
Пристрої передачі даних	0,06	160,0	59,04
Оргтехніка	0,36	5,0	11,07
Всього			1033,82

4.3.9 Службові відрядження

Витрати за статтею «Службові відрядження» розраховуємо як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$В_{св} = (З_о + З_р) \cdot \frac{Н_{св}}{100\%}, \qquad (4.12)$$

де $Н_{св}$ – норма нарахування за статтею «Службові відрядження», приймемо $Н_{св} = 20\%$.

$В_{св} = (52933,33 + 5629,36) \cdot 20 / 100\% = 11712,54$ грн.

4.3.10 Витрати на роботи, які виконують сторонні підприємства, установи і організації

Витрати розраховуємо як 30…45% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{сп} = (З_о + З_р) \cdot \frac{Н_{сп}}{100\%}, \quad (4.13)$$

де $Н_{сп}$ – норма нарахування за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації», приймемо $Н_{сп}$= 40%.

$B_{сп}$ = (52933,33 + 5629,36) · 40 / 100% = 23425,08 грн.

4.3.11 Інші витрати

Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$І_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (4.14)$$

де *$Н_{ів}$* – норма нарахування за статтею «Інші витрати», приймемо $Н_{ів}$ = 50%.

$І_в$ = (52933,33 + 5629,36) · 50 / 100% = 29281,34 грн.

4.3.12 Накладні (загальновиробничі) витрати

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (4.15)$$

де *$Н_{нзв}$* – норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $Н_{нзв}$ = 125%.

$B_{нзв}$ = (52933,33 + 5629,36) · 125 / 100% = 73203,36 грн.

Витрати на проведення науково-дослідної роботи на тему «Розробка клієнт-серверної системи моніторингу паркувальних майданчиків (Серверна частина)» розраховуємо як суму всіх попередніх статей витрат за формулою:

$$B_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + B_{спец} + B_{прг} + A_{обл} + B_е + B_{св} + B_{сп} + I_в + B_{нзв}. \quad (4.16)$$

$B_{заг}$= 52933,33 +5629,36 +6441,90 +14301,00866 +4947,80 +0,00 +47667,20 + 6937,50 + 8320,83 +1033,82 +11712,54 +23425,08 +29281,34 +73203,36 = 285835,06 грн.

Загальні витрати *ЗВ* на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою:

$$ЗВ = \frac{B_{заг}}{\eta}, \quad (4.17)$$

де η - коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи, приймемо η=0,95.

ЗВ = 285835,06 / 0,95 = 300879,01 грн.

4.4 Розрахунок економічної ефективності науково-технічної розробки при її можливій комерціалізації потенційним інвестором

Результати дослідження проведені за темою «Розробка клієнт-серверної системи моніторингу паркувальних майданчиків (Серверна частина)» передбачають комерціалізацію протягом 4-х років реалізації на ринку.

В цьому випадку основу майбутнього економічного ефекту будуть формувати:

ΔN – збільшення кількості споживачів яким надається відповідна інформаційна послуга у періоди часу, що аналізуються;

Показник	1-й рік	2-й рік	3-й рік	4-й рік
Збільшення кількості споживачів, осіб	5000	15000	20000	10000

N – кількість споживачів яким надавалась відповідна інформаційна послуга у році до впровадження результатів нової науково-технічної розробки, приймемо 10000 осіб;

$Ц_б$ – вартість послуги у році до впровадження інформаційної системи, приймемо 250,00 грн;

$\pm\Delta Ц_о$ – зміна вартості послуги від впровадження результатів, приймемо 49,13 грн.

Можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із 4-х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховуємо за формулою [52]:

$$\Delta П_i = (\pm\Delta Ц_о \cdot N + Ц_о \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \quad (4.18)$$

де λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2022 році ставка податку на додану вартість складає 20%, а коефіцієнт $\lambda = 0,8333$;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту). Приймемо ρ =40%;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2022 році $\vartheta = 18\%$;

Збільшення чистого прибутку 1-го року:

$\Delta П_1$ =(49,13·10000,00+299,13·5000)·0,83·0,4·(1-0,18/100%)=540906,85 грн.

Збільшення чистого прибутку 2-го року:

$\Delta П_2$ =(49,13·10000,00+299,13·20000)·0,83·0,4·(1-0,18/100%)=1762413,70 грн.

Збільшення чистого прибутку 3-го року:

$\Delta П_3$ =(49,13·10000,00+299,13·40000)·0,83·0,4·(1-0,18/100%)=3391089,50 грн.

Збільшення чистого прибутку 4-го року:

$\Delta П_4$ =(49,13·10000,00+299,13·50000)·0,83·0,4·(1-0,18/100%)=4205427,40 грн.

Приведена вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \quad (4.19)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ =0,22;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$ПП$ =540906,85/(1+0,22)^1+1762413,70/(1+0,22)^2+3391089,50/(1+0,22)^3+

+4205427,40/(1+0,22)^4=443366,27+1184099,50+1867496,34+1898326,09=

=5393288,21 грн.

Величина початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки:

$$PV = k_{інв} \cdot ЗВ, \quad (4.20)$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо $k_{інв}$=2;

$ЗВ$ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 300879,01 грн.

$PV = k_{інв} \cdot ЗВ$= 2 · 300879,01 = 601758,02 грн.

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV \tag{4.21}$$

де $ПП$ – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 5393288,21 грн;

PV – теперішня вартість початкових інвестицій, 601758,02 грн.

$E_{абс} = ПП - PV$ = 5393288,21 - 601758,02 = 4791530,19 грн.

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1, \tag{4.22}$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, 4791530,19 грн;

PV – теперішня вартість початкових інвестицій, 601758,02 грн;

$T_ж$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, 4 роки.

$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1$ = (1+4791530,19/601758,02)$^{1/4}$= 0,73.

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$:

$$\tau_{мін} = d + f, \quad (4.23)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2022 році в Україні d =0,1;

f – показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,25.

$\tau_{мін}$= 0,1+0,25 = 0,35 < 0,73 свідчить про те, що внутрішня економічна дохідність інвестицій $E_в$, вища мінімальної внутрішньої дохідності. Тобто інвестувати в науково-дослідну роботу за темою «Розробка клієнт-серверної системи моніторингу паркувальних майданчиків (Серверна частина)» доцільно.

Період окупності інвестицій $T_{ок}$ які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_в}, \quad (4.24)$$

де $E_в$ – внутрішня економічна дохідність вкладених інвестицій.

$T_{ок}$= 1 / 0,73 = 1,37 р.

$T_{ок}$ < 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

4.5 Висновки до розділу

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Розробка клієнт-серверної системи моніторингу паркувальних майданчиків (Серверна частина)» становить 39,0 бала, що, свідчить про комерційну важливість

проведення даних досліджень (рівень комерційного потенціалу розробки вище середнього).

При оцінюванні за технічними параметрами, згідно узагальненого коефіцієнту якості розробки, науково-технічна розробка переважає існуючі аналоги приблизно в 4,06 рази.

Також термін окупності становить 1,37 р., що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Отже можна зробити висновок про доцільність проведення науково-дослідної роботи за темою «Розробка клієнт-серверної системи моніторингу паркувальних майданчиків (Серверна частина)».

ВИСНОВКИ

В результаті виконаної роботи було розроблено серверну частину системи моніторингу паркувальних майданчиків, доведено актуальність роботи, проведено аналіз підходів розробки клієнт - серверних систем та архітектурних патернів. Сформульовано основні вимоги до даної системи. Також проаналізовано різні методи збору статистики паркувальних майданчиків, з'ясовано їх переваги та недоліки.

Виконано порівняння сучасних клієнт-серверних систем моніторингу паркувальних майданчиків, з системою, яка досліджується у даній роботі. Обґрунтовано вибір архітектурного патерна та технології розробки.

На основі отриманих результатів аналізу технічних рішень, розроблено базові модулі додатку:

- модуль отримання статистики у вигляді сесій водіїв;
- модуль обробки та перевірки отриманої статистики;
- модуль для створення пакетів інвентаризації;
- модуль, який відповідає за перевірку створеного пакету;
- модуль для автоматичної доставки пакетів до інформаційної панелі.

Протестовано розроблені модулі за допомогою unit тестів та фреймворку Mockito. Розроблена архітектура системи, структурні та інформаційні схеми, а також UML діаграма класів.

В результаті виконаних робіт було досягнуто основну мету, а саме: реалізацію системи для отримання статистики, обробки та перевірки отриманої статистики, створення пакетів інвентаризації, автоматичної доставки пакетів до інформаційної панелі що дозволяє підвищити ефективності організації дорожнього руху муніципалітетам за рахунок інформаційної панелі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Копиця В. О., Коцюбинський В. Ю., Розробка клієнт-серверної частини системи моніторингу паркувальних майданчиків – Копиця В. О. – LI Науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації (2022), Вінницького національного технічного університету – 2022. – 2 с.

2. Копиця В. О., Коцюбинський В. Ю., Розробка клієнт-серверної частини системи моніторингу оплати системи парковки – Копиця В. О. – L Науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації (2021), Вінницького національного технічного університету – 2021. – 2 с.

3. Поняття клієнт-серверних систем [Електронний ресурс]. – Режим доступу: http://bourabai.kz/dbt/client1.html/ - Назва з екрану

4. Клиент-серверная архитектура приложения [Електронний ресурс]. – Режим доступу: http://fkn.ktu10.com/?q=node/9330 - Назва з екрану.

5. Kyle M. Learning Material Design / Kyle M. – New York: Pack Publishing, 2015. – 186 p. – ISBN - 978-1785289811.

6. Serving Web Content with Spring MVC [Електронний ресурс]. – Режим доступу: https://spring.io/guides/gs/serving-web-content/ - Назва з екрану.

7. TutorialsTeatcher. MVC Architecture [Електронний ресурс]. – Режим доступу: https://www.tutorialsteacher.com/mvc/mvc-architecture - Назва з екрану.

8. Сравнение систем мониторинга Zabbix vs Nagios [Електронний ресурс]. – Режим доступу: https://www.netping.ru/Blog/sravnenie-sistem-monitoringa-zabbix-vs-nagios - Назва з екрану.

9. Barth W. Nagios, 2nd Edition: System and Network Monitoring / Barth W. – London: No Starch Press, 2015. – 719 p. – ISBN - 978-1593271794.

10. Olups R. Zabbix Network Monitoring / Olups R. – Ltd: Packt Publishing, 2016. – 754 p. – ISBN - 978-1782161295.

11. Анализ юзабилити сайта, изучение внешнего вида и эффективности работы сайта (usability) [Електронний ресурс]. – Режим доступу: http://www.m-mix.com.ua/marketing/analiz-usability.html – Назва з екрану.

12. Веб зсередини [Електронний ресурс]. – Режим доступу: http://www.zhu.edu.ua/mk_school/mod/page/view.php – Назва з екрану.

13. Веб-програмування. HTML5 [Електронний ресурс]. – Режим доступу: http://webstudio2u.net/ua/programming/489-html5-css3.html. – Назва з екрану.

14. Новые теги HTML5.0 [Електронний ресурс]. – Режим доступу: http://www.wisdomweb.ru/HTML5d/ – Назва з екрану.

15. Goldstein Alexis HTML5 & CSS3 ForTheRealWorld / Alexis Goldstein, Louis Lazaris, Estelle Weyl. – NY.: Science, 2015. – 350 p. – ISBN 978-0-9874674-9-2.

16. Статті / CSS [Електронний ресурс]. – Режим доступу: http://www.webostudio.com/ua/stats/CSS/ – Назва з екрану.

17. Настройки CSS, стилизация HTML-элементов с использованием расширенных классов, и передовая система разметки [Електронний ресурс]. – Режим доступу: http://bootstrap-3.ru/css.php – Назва з екрану.

18. Роббинс Дж. HTML5. Карманный справочник / Роббинс Дж. – М.: ВИЛЬЯМС, 2015. – 192с. – ISBN: 978-5-8459-1937-3.

19. Основні компоненти мови UML [Електронний ресурс]. – Режим доступу: http://um.co.ua/9/9-2/9-29941.html – Назва з екрану.

20. Spring Security [Електронний ресурс]. – Режим доступу: http://projects.spring.io/spring-security/ - Назва з екрану.

21. Refactoring Book / М. Фаулер, К. Бек, Д. Брант, У. Апдайк. – М.: Вильямс. – 2017. – 63 - 78 с. – ISBN 5-93286-045-6

22. Pro Angular. Second Edition / Adam Freeman – A.:Apress, 2017. – 788 с. – ISBN 978-1-4842-2307-9;

23. AngularJS Tutorial: A Comprehensive 10000 World Guide [Електронний ресурс]. – Режим доступу: https://www.airpair.com/angularjs – Назва з екрану.

24. AngularJS – MVC архітектура [Електронний ресурс]. – Режим доступу: http://thewebland.net/development/javascript/angularjs/angularjs-mvc/ – Назва з екрану.

25. Внедрение компонентого подхода в вебе: обзор веб – компонентов [Електронний ресурс]. – Режим доступу: https://habrahabr.ru/company/microsoft/blog/264791/ – Назва з екрану.

26. Компонентный подход к созданию приложения с помощью AngularJS 1.5 & 2.0 [Електронний ресурс]. – Режим доступу: http://bogdanovblog.ru/komponentnyj-podhod-k-sozdaniyu-prilozheniya-s-pomoshhyu-angularjs-1-5- 2-0/ – Назва з екрану.

27. Каскадные таблицы стилей [Електронний ресурс]. – Режим доступу: http://www.4stud.info/web-programming/css.html – Назва з екрану.

28. Software Testing Fundamentals [Електронний ресурс]. – Режим доступу: http://softwaretestingfundamentals.com/defect/ – Назва з екрану.

29. Unit tests with Mockito – Tutorial [Електронний ресурс]. – Режим доступу: http://www.vogella.com/tutorials/Mockito/article.html – Назва з екрану.

30. Документація Java по фреймворку Mockito 2017 [Електронний ресурс]. – Режим доступу: http://static.javadoc.io/org.mockito/mockitocore/1.10.19/org/mockito – Назва з екрану.

31. Spring Framework Overview [Електронний ресурс]. – Режим доступу: http://www.tutorialspoint.com/spring/spring_overview.htm - Назва з екрану.

32. Балансировка нагрузки: основные алгоритмы и методы [Електронний ресурс]. – Режим доступу: https://habr.com/ru/company/selectel/blog/250201/ – Назва з екрану.

33. Round-Robin Game Scheduling [Електронний ресурс]. – Режим доступу: http://gilmerbaseball.com/library/roundrobin.asp – Назва з екрану.

34. Amazon EC2 T3 Instances [Електронний ресурс]. – https://aws.amazon.com/ec2/instance-types/t3/ – Назва з екрану.

35. Brown N. Amazon Elastic Compute Cloud (EC2): Guide for Beginners. Easy to Understand / Nicholas Brown, – South Carolina: CreateSpace Independent Publishing Platform, 2017. – 74 c. – ISBN – 978-1542885621.

36. Broyles P. Amazon Web Services. The Ultimate Guide for Beginners, Intermediates and Expert / Phillip Broyles, - KDP Print US: Amazon Digital Services LLC, 2020. – 122 c. – ISBN – 979-8601352512.

37. Blokdyk G. Load Balancer: A Complete Guide / Gerardus Blokdyk, – South Carolina: CreateSpace Independent Publishing Platform, 2018. – 110 c. – ISBN – 978-1718616219.

38. Kopparapu C. Load Balancing Servers, Firewalls and Caches / Chandra Kopparapu, - Hoboken: John Wiley & Sons, 2002. – 224 c. – ISBN – 978-0471421283.

39. Virtual Instance. Future Trends and Research Directions [Електронний ресурс]. – https://www.sciencedirect.com/topics/computer-science/virtual-instance – Назва з екрану.

40. Пасічник В.В. Глобальні інформаційні системи та технології: моделі ефективного аналізу, опрацювання та захисту даних. Монографія / В.В. Пасічник, П. І. Жежнич, Р. Б. Кравець, А. М. Пелещишин, Д. О. Тарасов – Львів: Видавництво Львівської політехніки, 2006. – 348 с. ISBN: 966-553-578-1.

41. What is client-server architecture / Що таке клієнт-серверна архітектура [Електронний ресурс]. – Режим доступу: http://apachebooster.com/kb/what-is-client-server-architecture-and-what-are-its-types/ - Назва з екрану.

42. RESTful API Design [Електронний ресурс] – Режим доступу: https://hackernoon.com/restful-api-design-step-by-step-guide-2f2c9f9fcdbf – Назва з екрану.

43. Robert M. Clean Architecture: Guide to Software Structure and Design / Robert M. – Washington: Pearson 2017. – 432 p. ISBN 978-0134494166.

44. Clean Code Blog [Електронний ресурс] – Режим доступу: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html - Назва з екрану.

45. Use Case Diagrams [Електронний ресурс] – Режим доступу: https://www.smartdraw.com/use-case-diagram/ - Назва з екрану.

46. Regina O. PostgreSQL: Up and Running / Regina O. - Sebastopol: O'Reilly Media 2012. – 168 p. – ISBN 978-1449326333.

47. Robert M. Clean Code: A Handbook of Agile Software Craftsmanship / Robert M. – Washington: Pearson 2008. – 464 p. – ISBN 978-0132350884.

48. Ajit K. Sencha MVC Architecture / Ajit K. – Birmingham: Packt Publishing 2012. -126 p. – ISBN 978-1849518888.

49. Mark M. REST API Design Rulebook / Mark M. – Sebastopol: O'Reilly Media, 2011. -116 p. – ISBN 978-1449310509.

50. Методичні рекомендації з комерціалізації розробок, створених в результаті науково-технічної діяльності – К.: Наказ Державного комітету України з питань науки, інновацій та інформатики (Лист № 1/06-4-97 від 13.09.2010 р.).

51. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. – Вінниця : ВНТУ, 2021. – 42 с.

52. Кавецький В. В. Економічне обґрунтування інноваційних рішень: практикум / В. В. Кавецький, В. О. Козловський, І. В. Причепа – Вінниця : ВНТУ, 2016. – 113 с.

ДОДАТКИ

Додаток А (обов’язковий). Технічне завдання

ЗАТВЕРДЖЕНО
Зав. кафедри АІІТ
_____ Бісікало О. В.
«__»_____ 2022 р.

ТЕХНІЧНЕ ЗАВДАННЯ
на магістерську кваліфікаційну роботу
«Розробка клієнт-серверної системи моніторингу паркувальних майданчиків(Серверна частина)»
08-02.МКР.000.00.017 ТЗ

Керівник роботи:
к.т.н., доц. каф. АІІТ, Коцюбинський В.Ю.
“___” _____ 2022 р.

Виконав: ст. гр. 1ІСТ-21м
Копиця В.О.
“___” _____ 2022 р.

Вінниця 2022

1. Назва та галузь застосування

«Розробка клієнт-серверної системи моніторингу паркувальних майданчиків(Серверна частина)» - призначена для застосування в галузі автоматизованого моніторингу паркувальних майданчиків з метою покращення якості моніторингу, автоматизації процесів збору статистики та зменшення впливу людського фактору.

2. Підстави для розробки

Розробку системи здійснювати на підставі наказу по університету № 216 від 25.09.2022 та завдання до магістерської кваліфікаційної роботи, складеного та затвердженого кафедрою «Автоматизації та інтелектуальних інформаційних технологій».

3. Мета та призначення розробки

Метою даної роботи є покращення процесу надання послуг у моделі взаємодії B2C (Bussiness-to-Consumer), B2B (Bussiness-to-Bussiness), B2G (Bussiness-to-Government) за рахунок надання інформаційної панелі для моніторингу даних за основними параметрами з метою контролю та покращення надання послуг.

4. Джерела розробки

1. Копиця В. О., Розробка клієнт-серверної частини системи моніторингу паркувальних майданчиків – Копиця В. О. – LI Науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації (2022), Вінницького національного технічного університету – 2022. – 2 с.

2. Пасічник В.В. Глобальні інформаційні системи та технології: моделі ефективного аналізу, опрацювання та захисту даних. Монографія / В.В. Пасічник,

П. І. Жежнич, Р. Б. Кравець, А. М. Пелещишин, Д. О. Тарасов – Львів: Видавництво Львівської політехніки, 2006. – 348 с. ISBN: 966-553-578-1.

3. Поняття клієнт-серверних систем [Електронний ресурс]. – Режим доступу: http://bourabai.kz/dbt/client1.html/ - Назва з екрану

4. Ranade J. Client/Server Architecture / Ranade J. – Boston: Mcgraw-Hill, 1992. - 452 p. – ISBN - 978-0070050761.

5. Brown N. Amazon Elastic Compute Cloud (EC2): Guide for Beginners. Easy to Understand / Nicholas Brown, – South Carolina: CreateSpace Independent Publishing Platform, 2017. – 74 c. – ISBN – 978-1542885621.

5. Показники призначення

Клієнт-серверна система моніторингу паркувальних майданчиків має забезпечувати коректний збір статистичних даних по основним параметрам паркувальних майданчиків, опрацювати та перевіряти ці дані, а також давати можливість генерувати пакети інвентаризації для інтеграції з інформаційною панелю .

Вхідні дані: статистичні дані паркувальних майданчиків (розташування, кількість місці для паркування, тип паркувального майданчику, дорожній знак CPU).

Вихідні дані: пакет інвентаризації у форматі GEOJSON.

6. Економічні показники

- прогнозовані витрати на розробку – не більше 300 тис. грн;
- термін окупності витрат для інвестора – не більше 3 років.

7. Стадії розробки

1. Розділ 1 «Аналіз та порівняння існуючих клієнт-серверних систем моніторингу паркувальних майданчиків» має бути виконаний до 29.10.2022.

2. Розділ 2 «Проектування архітектури системи» має бути виконаний до 03.11.2022.

3. Розділ 3 «Розробка програмного забезпечення» має бути виконаний до 15.11.2020.

4. Економічний розділ має бути виконаний до 30.11.2022.

7. Порядок контролю та приймання

1. Рубіжний контроль провести до 15.11.2022

2. Попередній захист магістерської кваліфікаційної роботи провести до 01.12.2020.

3. Захист магістерської кваліфікаційної роботи провести в період з 07.12.2021 до 06.01.2022.

Додаток Б (обов'язковий). Графічна частина

в. о. Зав. кафедри АІІТ _____ д-р техн. наук, професор каф. АІІТ
Бісікало О. В.
(підпис) (науковий ступінь, вчене звання, ініціали та прізвище)

Науковий керівник _____ канд. техн. наук, доцент каф. АІІТ
Коцюбинський В. Ю.
(підпис) (науковий ступінь, вчене звання, ініціали та прізвище)

Тех. контроль _____ канд. техн. наук, доцент каф. АІІТ
Коцюбинський В. Ю.
(підпис) (науковий ступінь, вчене звання, ініціали та прізвище)

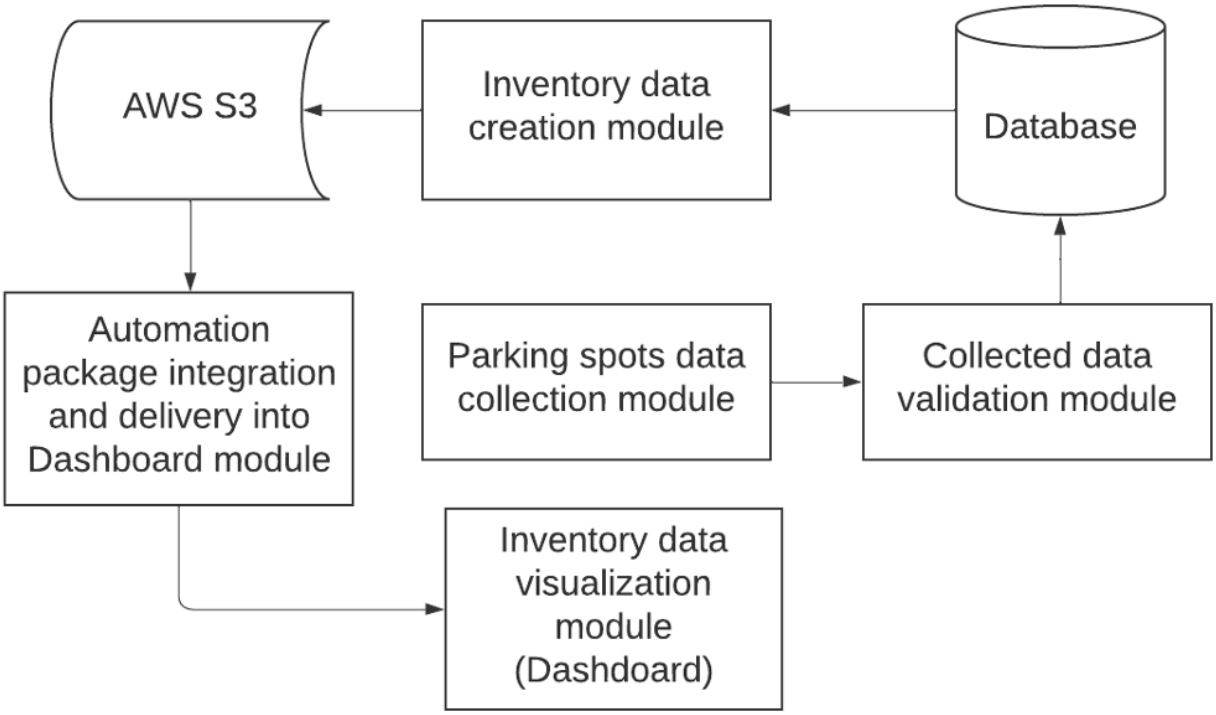
Нормоконтроль _____ канд. техн. наук, доцент каф. АІІТ
Маслій Р. В.
(підпис) (науковий ступінь, вчене звання, ініціали та прізвище)

Рецензент _____ канд. техн. наук, доцент каф. КСУ
Биков М. М.
(підпис) (науковий ступінь, вчене звання, ініціали та прізвище)

Студент гр. 1ІСТ-21м _____ Копиця В. О.
(підпис) (ініціали та прізвище)

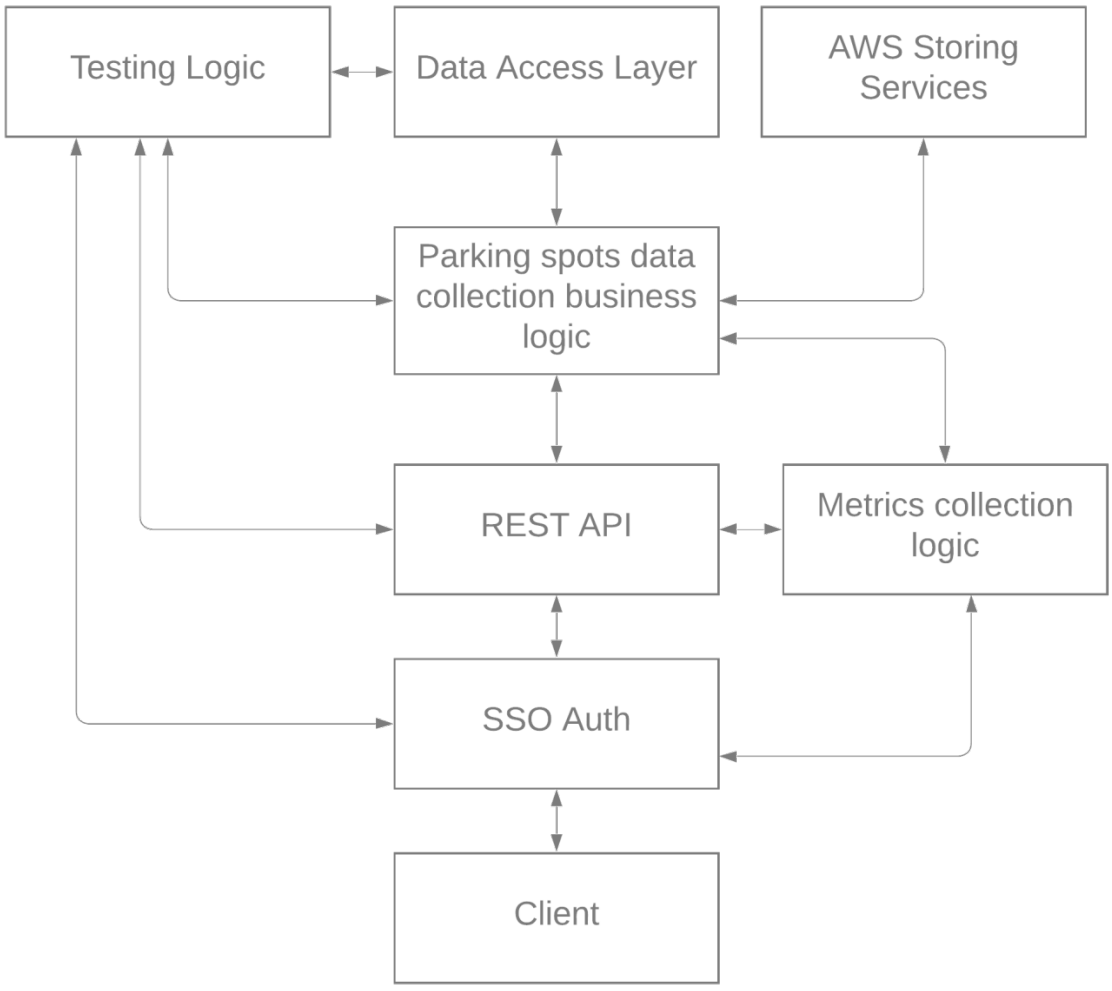
Додаток В (обов’язковий). UML діаграма архітектури системи

UML діаграма архітектури системи моніторингу паркувальних операторів



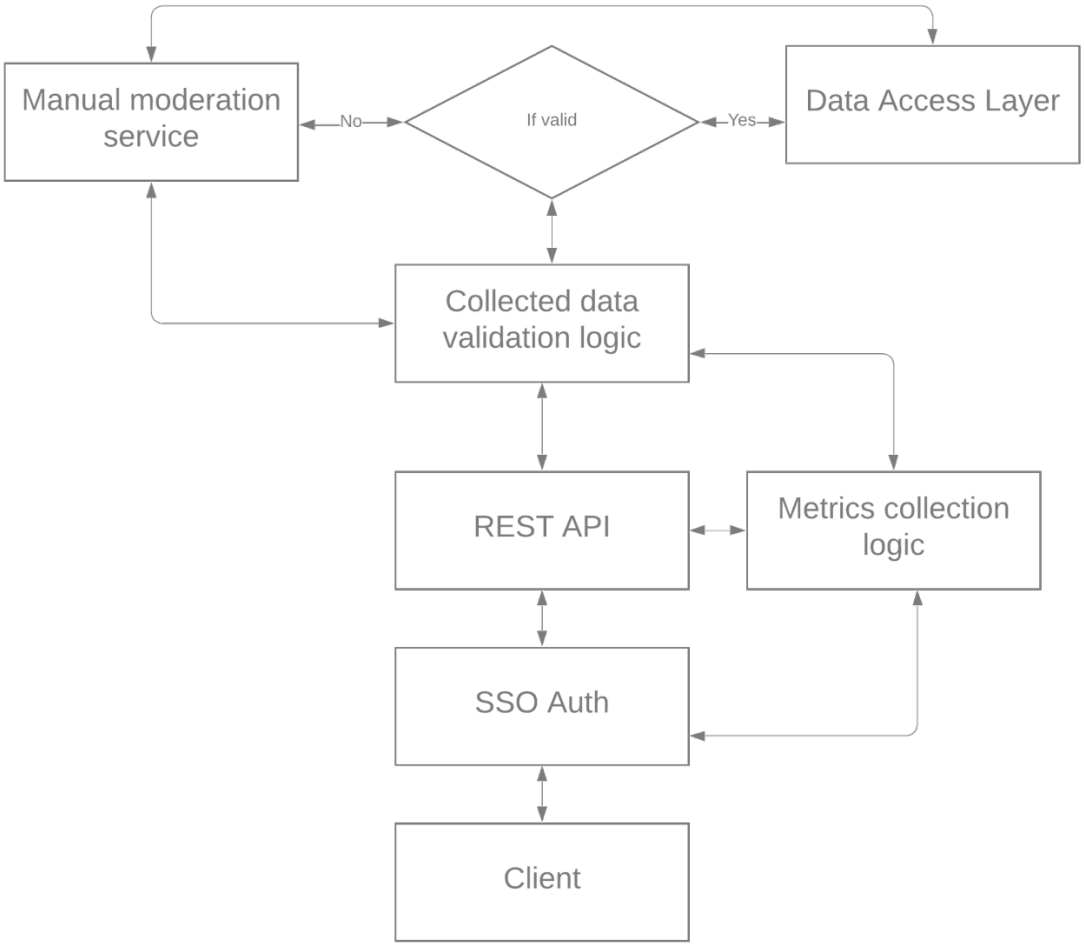
Додаток Г (обов’язковий). Workflow діаграма модулю збору статистики

Workflow діаграма модулю збору статистики про паркувальні майданчики



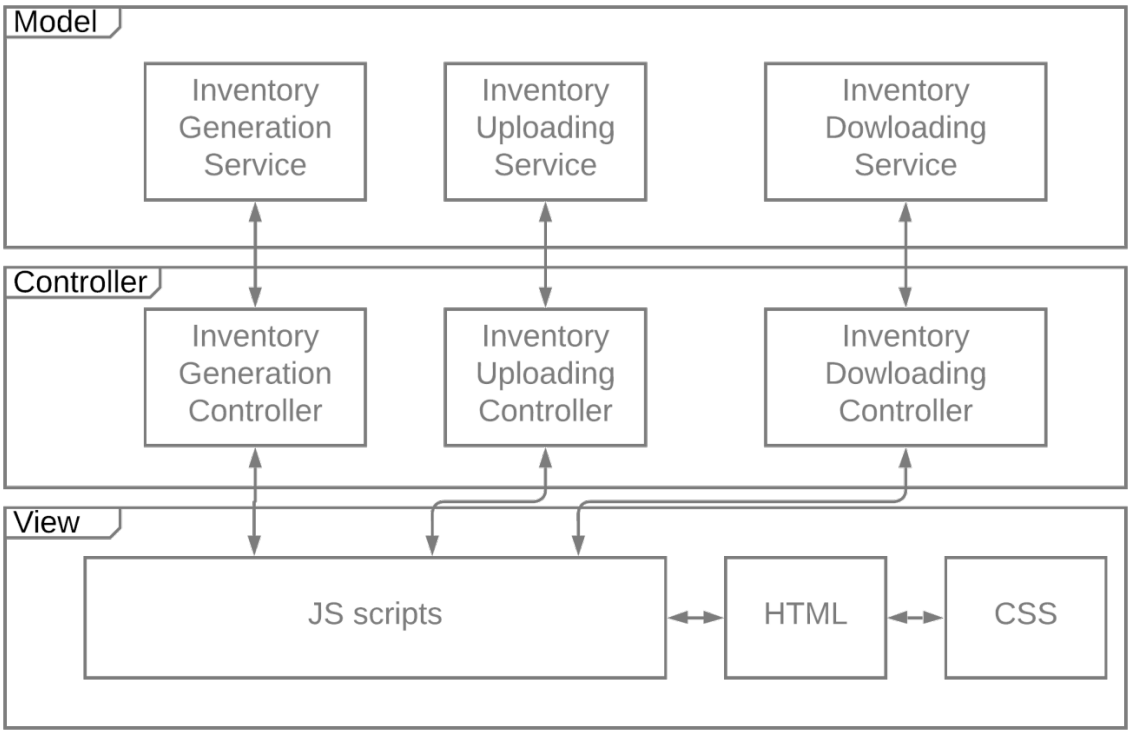
Додаток Д (обов’язковий). Workflow діаграма модулю перевірки статистики

Workflow діаграма модулю для перевірки статистики про паркувальні майданчики



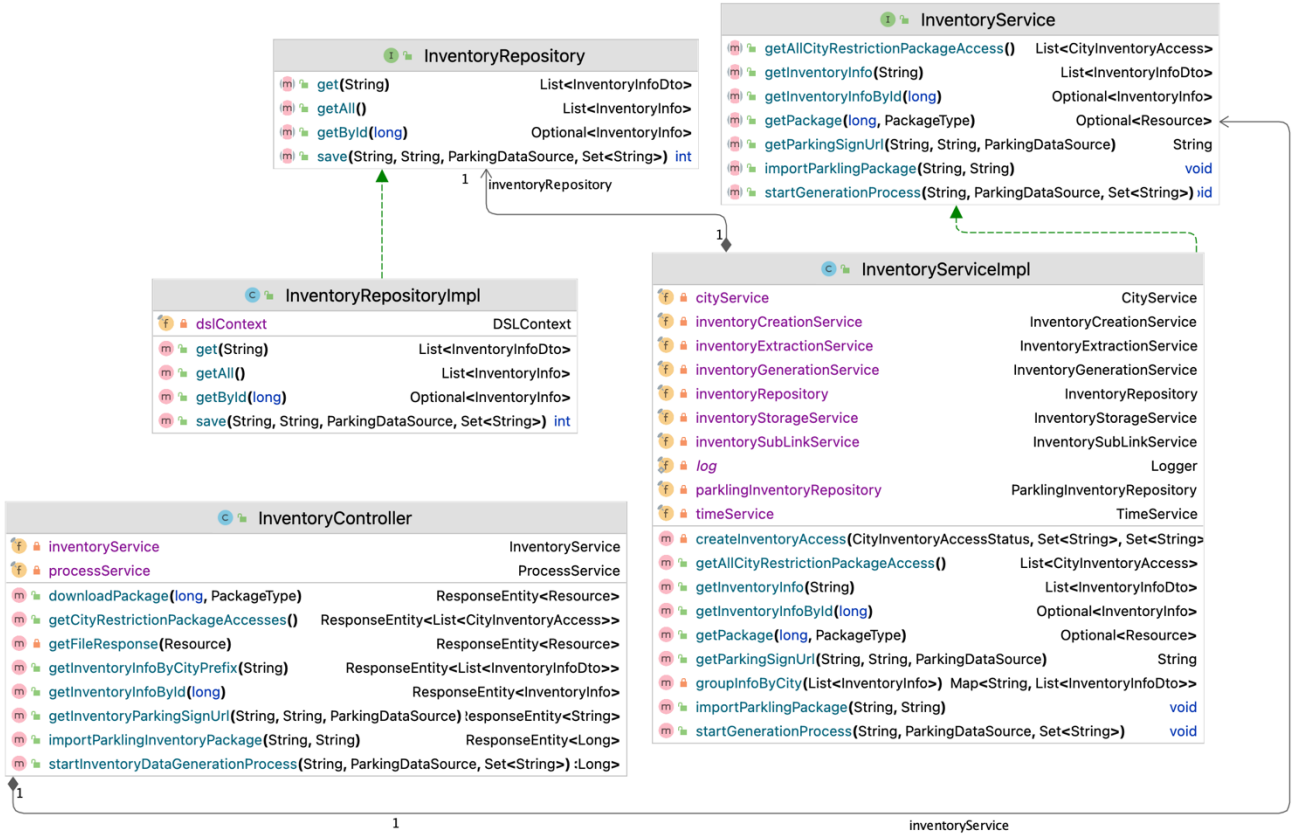
Додаток Е (обов’язковий). Структурна UML діаграма патерну MVC

Структурна UML діаграма патерну MVC модулю створення пакетів інвентаризації



Додаток Є (обов'язковий). UML діаграма класів модулю створення пакетів

UML діаграма класів модулю створення пакетів інвентаризації



Додаток Ж (обов'язковий). Акт впровадження

Акт впровадження

Науково-виробниче підприємство
"СПІЛЬНА СПРАВА" Товариство з обмеженою відповідальністю
Україна, 21000, м. Вінниця, вул. 1 Травня, 36/3, тел. +380 (67) 974 73 77
код ЄДРПОУ 31041670, код платників податків 310416702282, свідоцтво № 0183329
IBAN : UA 41 322313 0000026004000003226 в філії АТ «Укрексімбанк» в м. Вінниці

Затверджую

Директор

НВП «СПІЛЬНА СПРАВА», ТОВ

_____

Чикалова Анастасія Олександрівна

«_____» _____ 20__ р.

АКТ
впровадження результатів магістерської роботи
Копиці Вадима Олександровича «Розробка клієнт-серверної системи моніторингу паркувальних майданчиків(Серверна частина)»

Комісія у складі головного спеціаліста, керівника відділу обробки даних Я. Мовчана та керівника відділу розробки інформаційних систем О. Антонюка склали цей акт про те, що у Науково-виробничому підприємстві "Спільна Справа", ТОВ впроваджуються результати, які отримані магістрантом Копицею В. О. при виконанні магістерської кваліфікаційної роботи мають практичну цінність. Важливим процесом функціонування системи є збір статистики по паркувальним майданчикам, проведення перевірки зібраної статистики у автоматичному режимі та моніторинг за допомогою інформаційної панелі. Клієнт-серверна система для моніторингу паркувальних майданчиків знаходяться в стадії активного розвитку. Таким чином, актуальність роботи В. О. Копиці не викликає сумнівів, а низка методик та підходів та результати їх застосування можуть бути застосованими у практичній діяльності.

Науково-виробничому підприємству "Спільна Справа", ТОВ Копицею В. О. передано програмне забезпечення, яке дозволяє проводити моніторинг паркувальних майданчиків.

Члени комісії:

Головний спеціаліст. Я. Мовчан

Керівник відділу О. Антонюк

Додаток З (обов'язковий). Лістинг програми

Лістинг програми розробленого модулю створення пакетів інвентаризації

```
import com.pginventory.model.*;
import com.pginventory.model.process.ProcessType;
import com.pginventory.service.InventoryService;
import com.pginventory.service.ProcessService;
import lombok.RequiredArgsConstructor;
import org.springframework.core.io.Resource;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Set;

import static java.nio.file.Files.probeContentType;

@RestController
@RequiredArgsConstructor
@RequestMapping("inventory/data")
public class InventoryController {

    private final InventoryService inventoryService;
    private final ProcessService processService;

    @PostMapping("generate/{cityPrefix}")
    public ResponseEntity<Long> startInventoryDataGenerationProcess(@PathVariable
String cityPrefix,
           @RequestParam ParkingDataSource parkingDataSource, @RequestBody
Set<String> types) {
        long processId = processService.runProcessAsync(
                () -> inventoryService.startGenerationProcess(cityPrefix,
parkingDataSource, types),
                ProcessType.INVENTORY_GENERATION
        );
        return ResponseEntity.status(HttpStatus.ACCEPTED).body(processId);
    }

    @GetMapping("download/{id}")
    public ResponseEntity<Resource> downloadPackage(@PathVariable long id,
@RequestParam PackageType packageType) {
        return inventoryService.getPackage(id, packageType)
                .map(this::getFileResponse)
                .orElse(ResponseEntity.notFound().build());
    }

    @GetMapping("info/{id}")
    public ResponseEntity<InventoryInfo> getInventoryInfoById(@PathVariable long
id) {
        return ResponseEntity.of(inventoryService.getInventoryInfoById(id));
    }
}
```

Продовження додатку З

```
import com.pginventory.datalayer.repository.InventoryRepository;
import com.pginventory.datalayer.ParklingInventoryRepository;
import com.pginventory.exception.InventoryDataImportException;
import com.pginventory.exception.InventoryGenerationException;

import java.io.File;

import static java.util.stream.Collectors.*;

@Slf4j
@Service
@RequiredArgsConstructor
public class InventoryServiceImpl implements InventoryService {

    private final InventoryRepository inventoryRepository;
    private final ParklingInventoryRepository parklingInventoryRepository;
    private final TimeService timeService;
    private final InventoryGenerationService inventoryGenerationService;

    @Override
    @Transactional(isolation = Isolation.READ_COMMITTED, rollbackFor =
Exception.class)
    public void startGenerationProcess(String cityPrefix, ParkingDataSource
parkingDataSource, Set<String> types) {
        try {
            List<? extends Inventory> inventories =
inventoryGenerationService.generate(cityPrefix, parkingDataSource, types);
            if (!inventories.isEmpty()) {
                String packageFolder =
String.valueOf(timeService.currentTimeMillis());
                List<File> files =
inventoryCreationService.createPackages(cityPrefix, parkingDataSource,
inventories);
                inventoryStorageService.uploadPackages(cityPrefix,
parkingDataSource, packageFolder, files);
                inventoryRepository.save(cityPrefix, packageFolder,
parkingDataSource, types);
            } else {
                throw new InventoryGenerationException("Inventory list empty!");
            }
        } catch (Exception e) {
            String msg = String.format("Inventory generation process was failed for
city: [%s], source: [%s]. Reason: %s", cityPrefix, parkingDataSource.name(),
e.getMessage());
            throw new InventoryGenerationException(msg, e);
        }
    }

    @Override
    public Optional<Resource> getPackage(long id, PackageType packageType) {
        return inventoryRepository.getById(id)
                .map(packageInfo ->
inventoryStorageService.downloadPackage(packageInfo, packageType));
    }
```

Продовження додатку З

```
import com.pginventory.datalayer.repository.ParklingInventoryRepository;
import com.pginventory.model.ParklingEntryDto;
import com.pginventory.model.ParklingInventoryEntry;
import com.pginventory.model.ParklingSubLink;
import com.pginventory.service.GeometryService;
import com.pginventory.utils.PostGisFunctions;
import lombok.RequiredArgsConstructor;
import org.jooq.Record;
import org.jooq.*;
import org.springframework.stereotype.Repository;

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

import static com.pginventory.constaints.InventoryConstraints.PNG;
import static com.pginventory.datalayer.schema.jooq.inventory.tables.PARKLING_DATA;
import static com.pginventory.utils.PostGisFunctions.*;
import static com.pginventory.utils.StringUtils.toUpperCase;
import static java.util.Optional.ofNullable;
import static java.util.Set.of;
import static org.apache.commons.lang3.ArrayUtils.EMPTY_STRING_ARRAY;
import static org.apache.commons.lang3.StringUtils.EMPTY;
import static org.jooq.impl.DSL.*;

@Repository
@RequiredArgsConstructor
public class InventoryRepositoryImpl implements InventoryRepository {

    private final DSLContext dslContext;
    private final GeometryService geometryService;

    @Override
    public int importData(List<InventoryEntry> inventoryData) {
        return dslContext
                .batch(prepareBatchInsert(i))
                .execute().length;
    }

    @Override
    public int deleteAll(String cityPrefix) {
        return dslContext
                .deleteFrom(DATA)
                .where(PARKLING_DATA.CITY_PREFIX.eq(cityPrefix))
                .execute();
    }
```

Продовження додатку З

```
    @Override
    public List<ParklingEntryDto> getAll(String cityPrefix) {
        return dslContext
                .select(PARKLING_DATA.ID, PARKLING_DATA.RAW_LENGTHS,
PARKLING_DATA.PERMISSIONS,
stAsText(getGeometryOffsetCurve()).as(PARKLING_DATA.GEOMETRY),
                        nullif(PARKLING_DATA.STREET_NAME,
EMPTY).as(PARKLING_DATA.STREET_NAME), PARKLING_DATA.IMAGE_KEY,
PARKLING_DATA.SEGMENT)
                .from(PARKLING_DATA)
                .where(PARKLING_DATA.CITY_PREFIX.eq(cityPrefix))
                .fetch(r -> new ParklingEntryDto(
                        r.get(PARKLING_DATA.ID),
                        r.get(PARKLING_DATA.SEGMENT),
                        r.get(PARKLING_DATA.RAW_LENGTHS),
                        toUpperCase(of(r.get(PARKLING_DATA.PERMISSIONS))),

geometryService.readGeometryAsLineString(r.get(PARKLING_DATA.GEOMETRY,
String.class)),
                        ofNullable(r.get(PARKLING_DATA.STREET_NAME)),
                        r.get(PARKLING_DATA.IMAGE_KEY)
                ));
    }

    private Field<Object> getGeometryOffsetCurve() {
        return case_()
                .when(PARKLING_DATA.STREET_SIDE.eq("right"),
stOffsetCurve(PARKLING_DATA.GEOMETRY, "-0.00002"))
                .when(PARKLING_DATA.STREET_SIDE.eq("left"),
stOffsetCurve(PARKLING_DATA.GEOMETRY, "0.00002"));
    }

    @Override
    public Set<String> getTypes(String cityPrefix) {
        return dslContext
                .selectDistinct(PARKLING_DATA.PERMISSIONS)
                .from(PARKLING_DATA)
                .where(PARKLING_DATA.CITY_PREFIX.eq(cityPrefix))
                .fetch(r -> r.get(PARKLING_DATA.PERMISSIONS))
                .stream()
                .flatMap(Arrays::stream)
                .map(String::toUpperCase)
                .collect(Collectors.toSet());
    }
```

Продовження додатку З

```
@Override
public Set<String> getCityWithData() {
    return dslContext
            .selectDistinct(PARKLING_DATA.CITY_PREFIX)
            .from(PARKLING_DATA)
            .fetchSet(PARKLING_DATA.CITY_PREFIX);
}

@Override
public List<ParklingSubLink> getParklingSubLinks(String cityPrefix) {
    return dslContext
            .select(PARKLING_DATA.ID, PARKLING_DATA.SEGMENT,
PARKLING_DATA.CITY_PREFIX,
stAsText(getGeometryOffsetCurve()).as(PARKLING_DATA.GEOMETRY),
                    PARKLING_DATA.RAW_LENGTHS, concat(PARKLING_DATA.IMAGE_KEY,
PNG).as(PARKLING_DATA.IMAGE_KEY), PARKLING_DATA.STREET_NAME,
PARKLING_DATA.PERMISSIONS)
            .from(PARKLING_DATA)
            .where(PARKLING_DATA.CITY_PREFIX.eq(cityPrefix))
            .fetch(this::fetchParklingSubLink);
}

@Override
public boolean isDataAvailable(String cityPrefix) {
    return dslContext
            .fetchExists(PARKLING_DATA, PARKLING_DATA.CITY_PREFIX.eq(cityPrefix));
}

private ParklingSubLink fetchParklingSubLink(Record8<Long, Long, String, String,
Integer, String, String, String[]> record) {
    return new ParklingSubLink(
            record.get(PARKLING_DATA.ID, Long.class),
            record.get(PARKLING_DATA.SEGMENT, Long.class),
            record.get(PARKLING_DATA.CITY_PREFIX, String.class),

geometryService.readGeometryAsLineString(record.get(PARKLING_DATA.GEOMETRY,
String.class)),
            record.get(PARKLING_DATA.RAW_LENGTHS, Integer.class),
            record.get(PARKLING_DATA.IMAGE_KEY, String.class),
            record.get(PARKLING_DATA.STREET_NAME, String.class),
            Set.of(record.get(PARKLING_DATA.PERMISSIONS, String[].class))
    );
}
```

Продовження додатку З

```
private List<InsertSetMoreStep<Record>>
prepareBatchInsert(List<ParklingInventoryEntry> parklingInventoryData) {
    return parklingInventoryData.parallelStream()
            .map(entry -> insertInto(PARKLING_DATA)
                    .set(PARKLING_DATA.SEGMENT, entry.segment())
                    .set(PARKLING_DATA.STREET_SIDE, entry.streetSide())
                    .set(PARKLING_DATA.DIST1, entry.dist1())
                    .set(PARKLING_DATA.DIST2, entry.dist2())
                    .set(PARKLING_DATA.RAW_LENGTHS, entry.rawLengths())
                    .set(PARKLING_DATA.PARK_ANGLE, entry.parkAngle())
                    .set(PARKLING_DATA.ORIENTATION, entry.orientation())
                    .set(PARKLING_DATA.IS_CONSTRUCTION, entry.isConstruction())
                    .set(PARKLING_DATA.PARKING_SIGN, entry.parkingSign())
                    .set(PARKLING_DATA.PAYMENT_PERIOD, entry.paymentPeriod())
                    .set(PARKLING_DATA.RESTRICTION_PERIOD,
entry.restrictionPeriod())
                    .set(PARKLING_DATA.RESTRICTION_SIGN, entry.restrictionSign())
                    .set(PARKLING_DATA.PERMISSIONS,
entry.permission().toArray(EMPTY_STRING_ARRAY))
                    .set(PARKLING_DATA.PERMISSION_PERIOD,
entry.permissionPeriod().toArray(EMPTY_STRING_ARRAY))
                    .set(PARKLING_DATA.TIME_LIMITED, entry.timeLimited())
                    .set(PARKLING_DATA.TIME_LIMITED_PERIOD,
entry.timeLimitedPeriod())
                    .set(PARKLING_DATA.GEOMETRY, stGeomFromText(entry.geometry()))
                    .set(PARKLING_DATA.CITY_PREFIX, entry.cityPrefix())
                    .set(PARKLING_DATA.TIME_INFO, entry.timeInfo())
                    .set(PARKLING_DATA.IMAGE_KEY, entry.imageKey().orElse(null))
                    .set(PARKLING_DATA.SIGN_GEOMETRY,
entry.signGeometry().map(PostGisFunctions::stGeomFromText).orElse(null))
                    .set(PARKLING_DATA.STREET_NAME, entry.streetName()))
            .toList();
}
```

Додаток І (обов'язковий). Протокол перевірки навчальної роботи

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: *Комплексна магістерська кваліфікаційна робота «Розробка клієнт-серверної системи моніторингу паркувальних майданчиків(Серверна частина)».*
Частина 2. Технічне проектування
Тип роботи: кваліфікаційна робота
(кваліфікаційна робота, курсовий проект (робота), реферат, аналітичний огляд, інше – зазначити)
Підрозділ: кафедра АІІТ, ФКСА, 1ІСТ-21м
(кафедра, факультет, навчальна група)
Науковий керівник: Коцюбинський В.Ю., доц. каф. АІІТ
(прізвище, ініціали, посада)

Показники звіту подібності

Plagiat.pl (StrikePlagiarism)		*Unicheck*	
КП1	-	Оригінальність	95%
КП2	-		
Тривога/Білі знаки	/	Схожість	5%

Аналіз звіту подібності (відмітити потрібне)
Х Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

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

Автор _____ Копиця В.О.
(підпис) (прізвище, ініціали)

Опис прийнятого рішення:
Допустити до захисту

Особа, відповідальна за перевірку _____ Маслій Р.В.
(підпис) (прізвище, ініціали)