МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ І ПРИКЛАДНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДИКИЙ ДМИТРО СЕРГІЙОВИЧ

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій,
д-р. техн. наук, професор
_____Наталія ВЕСЕЛОВСЬКА
«____»_____ 2025 р.

ВЕБПОРТАЛ УПРАВЛІННЯ ЛОГІСТИКОЮ МАГАЗИНУ КОМПЛЕКТУЮЧИХ ДЛЯ СМАРТ-СИСТЕМ

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (магістерська) робота

Науковий керівник:
Надія ПОТАПОВА,
доцент кафедри інформаційних технологій,
к. е. н., доцент

_____
(підпис)

Оцінка: _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____
(підпис)

Вінниця – 2025

АНОТАЦІЯ

Дикий Д. С. Вебпортал управління логістикою магазину комплектуючих для смарт-систем. Спеціальність 122 «Комп'ютерні науки». Освітня програма Комп'ютерна обробка даних (Data Science). Донецький національний університет імені Василя Стуса, Вінниця, 2025.

У даній кваліфікаційній (магістерській) роботі досліджено та створено вебпортал логістичного забезпечення діяльності магазину компонентів для смарт-систем.

Структура роботи складається із вступу, трьох розділів, висновків і додатків. У вступі визначаються завдання роботи та висвітлена актуальність теми. У першому розділі досліджено теоретичні основи розробки вебпорталу управління логістикою магазину комплектуючих для смарт-систем. У другому розділі методи та моделі розробки архітектури вебпоратлу. Третій розділ присвячено викладенню результати практичної розробки вебпорталу, складових взаємодії користувача з програмним забезпеченням, а також висвітлено складові розгортання програмного продукту.

Розроблено архітектурну модель системи на основі клієнт-серверного підходу та патерна Model-View-Controller (MVC). Для реалізації серверної частини застосовано мову програмування Java з фреймворком Spring, Hibernate для ORM, Gradle для керування залежностями та MySQL як основну СУБД. Клієнтська частина реалізована із використанням HTML, CSS та JavaScript. Забезпечено кешування даних (Caffeine Cache) та автоматизоване розгортання системи за допомогою AWS, Docker та Jenkins.

У процесі роботи сформовано UML- та Use-Case-діаграми, моделі баз даних, визначено ключові сутності та їх взаємозв'язки.

Ключові слова: автоматизований веб-портал, смарт-системи, логістичне забезпечення, облік та контроль товарів, автоматизація бізнес-процесів.

81 с., 32 рис., 2 дод., 47 джерел.

ABSTRACT

Dykyi D. S. Web Portal for Logistics Management of a Parts Store for Smart-Systems. Specialization 122 “Computer Science.” Educational program Computer Data Processing (Data Science). Vasyl’ Stus Donetsk National University, Vinnytsia, 2025.

This master's (qualification) thesis explores and develops a web portal for the logistics management of a smart-systems components store.

The structure of the work consists of an introduction, three sections, conclusions and appendices. The introduction defines the tasks of the work and highlights the relevance of the topic. The first section examines the theoretical foundations of the development of a web portal for managing the logistics of a component store for smart systems. The second section examines the methods and models for developing the web portal architecture. The third section is devoted to presenting the results of the practical development of the web portal, the components of user interaction with the software, and also highlights the components of the software product deployment.

An architectural model of the system was developed based on the client-server approach and the Model-View-Controller (MVC) pattern. The server-side implementation uses the Java programming language with the Spring framework, Hibernate for ORM, Gradle for dependency management, and MySQL as the main DBMS. The client-side is implemented using HTML, CSS, and JavaScript. Data caching (Caffeine Cache) and automated system deployment using AWS, Docker, and Jenkins were also implemented.

During the work, UML and Use-Case diagrams were created, database models were designed, and key entities and their relationships were defined.

Keywords: automated web portal, smart systems, logistics management, goods accounting and control, business process automation.

81 p., 32 fig., bibliography: 47 items.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

UML – Unified Modeling Language.

IDE – Integrated Development Environment.

CSS – Cascading Style Sheets.

HTML – HyperText Markup Language.

JOOQ – Java Object-Oriented Queries.

API – Application Programming Interface.

REST – REpresentational State Transfer.

ORM – Object Relational Mapping.

RAD – Rapid Application Development

JVM – Java Virtual Machine

IOT – Internet of Things

MVC – Model-View-Controller

AWS – Amazon Web Services

SQL – Structured Quety Language

JVM – Java Virtual Machine

IoC – Inversion Of Control

DI – Dependency Injection

JPA – Java Persistence API

AI – Artificial Intelligence

DOM – Document Object-Oriented

ПЗ – Програмне забезпечення.

БД – База даних.

ПК – Персональний комп’ютер.

СУБД – Система управління базами даних.

ООП – Об’єктно-орієнтоване програмування.

ВСТУП

Протягом історії розвитку технологій людство прагнуло створювати засоби, здатні підвищувати ефективність праці, автоматизувати обчислення та полегшувати виконання завдань. Від механічних пристроїв і автоматів до сучасних комп'ютерних та мережевих технологій кожен етап прогресу наближав суспільство до створення систем, здатних діяти автономно, аналізувати дані та приймати рішення без прямої участі людини. Розвиток обчислювальних платформ, алгоритмів штучного інтелекту та технологій Інтернету речей (IoT) сформував основу для нового покоління смарт-систем – інтегрованих комплексів, які навчаються на даних, взаємодіють між собою та адаптуються до змін у реальному часі.

Смарт-системи поєднують апаратні засоби, програмне забезпечення, алгоритми машинного навчання та інтелектуальні інтерфейси. Вони застосовуються в різних сферах – енергетиці, транспорті, медицині, побутових системах та «розумних містах». Впровадження таких технологій дозволяє підвищити ефективність управління, оптимізувати ресурси та створювати комфортне й безпечне середовище для користувачів.

У сучасних умовах користувачі віддають перевагу простим, стабільним і надійним системам. Смарт-системи відповідають цим вимогам лише за умови використання якісних комплектуючих, які безпосередньо впливають на їхню функціональність. Тому надзвичайно важливо налагодити ефективну логістику постачання компонентів [1] із суворим контролем усіх етапів.

Актуальність роботи полягає у тому, що традиційні методи організації логістики часто не забезпечують необхідного рівня оперативності та прозорості, що призводить до витрат і зниження конкурентоспроможності. Використання веб-технологій дозволяє централізовано керувати закупівлями, складським обліком, моніторингом залишків і взаємодією з постачальниками у режимі реального часу. Створення вебпорталу для автоматизації та оптимізації логістики компонентів смарт-систем підвищує ефективність управління

товарними потоками і скорочує час обробки замовлень.

Метою роботи є створення вебпорталу управління логістикою магазину комплектуючих для смарт-систем, що представляє собою веборієнтовану інформаційну систему для поліпшення процесів забезпечення логістичної мережі магазину.

Завданнями даної роботи було визначено:

1. Дослідити існуючі підходи до управління логістичними механізмами доставки, складування, обліку та розподілу компонентів смарт-систем.

2. Оцінити методи до керування бізнес-процесами у реальних системах і з'ясувати вимоги до інформаційного забезпечення даної предметної області разом із покроковою інструкцією принципів функціонування вебпорталу.

3. Розробити загальну архітектуру системи з урахуванням поєднання усіх компонентів та структуру бази даних для відповідних сутностей згідно із характерними особливостями та предметної області.

4. Забезпечити розробку інтерфейсу, який поєднує інтуїтивне керування, зрозумілу організацію елементів та виконання функцій.

5. Систематизувати та узагальнити результати дослідження.

Об'єкт дослідження є вебпортал та механізми управління в логістичних інформаційних системах, що спеціалізується на продажі компонентів смарт-систем.

Предметом дослідження є методи та підходи до інформатизації процесів управління логістичним забезпеченням постачання компонентів смарт-систем за допомогою вебтехнологій.

Наукова новизна магістерської роботи полягає у:

- Розробці унікальних архітектурних рішень вебпорталу управління логістикою магазину комплектуючих для смарт-систем, які забезпечують повний цикл продажів від замовлення до отриманого товару покупцем.

- Розробці функціоналу, який зменшує використання ручної праці при управлінні постачаннями, обліком товарів та взаємодією з постачальниками, що

підвищує оперативність і точність виконання бізнес-процесів.

- Удосконаленні підходів до використання вебтехнологій в управлінні логістикою товаропровідної мережі шляхом запровадження автоматизації бізнес-процесів через вебпортал.

У процесі дослідження були використані **методи**: аналізу та синтезу, порівнянь аналогів, методи об’єктно-орієнтованого програмування, тестового контролю.

Структура магістерської роботи складається зі вступу, трьох розділів, висновків та списку використаних джерел. У вступі визначаються завдання роботи та висвітлена актуальність теми. У першому розділі досліджено теоретичні основи розробки вебпорталу управління логістикою магазину комплектуючих для смарт-систем. У другому розділі методи та моделі розробки архітектури вебпорталу. Третій розділ присвячено викладенню результати практичної розробки вебпорталу, складових взаємодії користувача з програмним забезпеченням, а також висвітлено складові розгортання програмного продукту.

Практична цінність даної роботи є в тому, що створений вебпортал реалізує механізми інформатизації функціональних дій з компонентами смарт-систем та забезпечує зменшення ручної праці при управлінні процесами постачання, обліку товарів та взаємодії з постачальниками чим сприяє підвищенню оперативності та точності виконання бізнес-процесів.

Результати дослідження були апробовані в доповіді «Вебпортал управління логістикою магазину комплектуючих смарт-систем» на IV Міжнародній науково-практичній конференції «Прикладні аспекти сучасних міждисциплінарних досліджень» (м. Вінниця, 5 листопада 2025 року).

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ ВЕБПОРТАЛУ УПРАВЛІННЯ ЛОГІСТИКОЮ МАГАЗИНУ КОМПЛЕКТУЮЧИХ ДЛЯ СМАРТ-СИСТЕМ

1.1. Смарт-системи та специфіка їх застосування

Смарт-системи [5], [6] є результатом розвитку кібернетики, автоматизації та інформаційних технологій, поєднуючи в собі інтелектуальні алгоритми, сенсори та програмно-апаратні засоби для прийняття рішень у реальному часі. Поняття «смарт-система» набуло поширення наприкінці XX століття, зокрема після появи мікропроцесорних технологій, систем автоматичного керування та розвитку IOT [7], [2]. Основна ідея таких систем полягає у здатності пристроїв самостійно аналізувати дані, адаптуватися до змін середовища та виконувати дії без безпосереднього втручання людини.

Специфіка смарт-систем полягає у поєднанні трьох ключових компонентів: сенсорного рівня (збір даних із навколишнього середовища), обчислювального рівня (обробка та аналіз інформації) та виконавчого рівня (реалізація управлінських дій). Вони використовують методи штучного інтелекту, машинного навчання та аналітики великих даних для підвищення ефективності прийняття рішень. Завдяки цьому смарт-системи здатні оптимізувати процеси, передбачати відмови обладнання, контролювати споживання ресурсів і забезпечувати високий рівень автоматизації.

Історично перші прототипи інтелектуальних систем з’явилися у 1970–1980-х роках у промисловості – це були автоматизовані виробничі лінії, системи моніторингу та контролю технологічних процесів. Згодом розвиток цифрових технологій і мікроелектроніки дозволив розширити сферу їх застосування. Сьогодні смарт-системи активно інтегруються у побутову сферу (розумні будинки, побутова техніка), транспорт (інтелектуальні системи навігації), енергетику (розумні мережі), охорону здоров’я (системи моніторингу стану

пацієнтів) та промисловість.

Інтеграція смарт-систем у складні інформаційно-технічні комплекси передбачає взаємодію між різнорівневими підсистемами через стандартизовані протоколи обміну даними. Такі системи можуть бути як автономними [8], так і частиною ширших мережевих структур, що дозволяє забезпечити узгоджене функціонування різних компонентів – від сенсорних пристроїв до аналітичних платформ. У контексті складних систем смарт-технології сприяють підвищенню надійності, адаптивності та стійкості системного управління, роблячи можливим ефективне функціонування у динамічному та невизначеному середовищі.

Під час розроблення інтелектуальних інформаційних систем, що функціонують із великими обсягами даних, одним із ключових етапів є чітке визначення об’єкта, предмета та мети дослідження. Ці складові формують концептуальне розуміння архітектури майбутнього програмного продукту, окреслюють його функціональні можливості й визначають технічні та організаційні межі реалізації.

У межах даної роботи об’єктом дослідження виступає веб-орієнтована платформа [9] для управління логістичними процесами інтернет-магазину комплектуючих до смарт-систем. Призначення системи полягає у забезпеченні користувачів інструментами для оформлення замовлень, створення описів посилок, відстеження їх переміщення, перегляду історії взаємодій та отримання консультаційної підтримки.

До основних вимог, що висуваються до розроблюваної системи, належать зручний користувацький інтерфейс, стабільність роботи за умов підвищеного навантаження, дотримання принципів інформаційної безпеки та захисту персональних даних. Водночас важливо забезпечити точний облік товарних позицій і здатність системи адаптуватися до змін попиту, що сприятиме оптимізації логістичних витрат і підвищенню ефективності управління запасами.

Метою роботи є створення автоматизованої клієнт-серверної системи для підтримки логістичних процесів інтернет-магазину та вдосконалення внутрішніх бізнес-процесів, пов’язаних із обігом комплектуючих для смарт-систем.

Розроблена платформа орієнтована на обслуговування кількох типів користувачів [10]: адміністратора, представника служби підтримки, зареєстрованого клієнта та гостьового користувача. Для кожної категорії визначено окремий набір функцій, при цьому функціональні можливості реалізуються ієрархічно – від базових до розширених. На рисунку 1.1 подано узагальнену схему розподілу функцій між ролями користувачів системи.

Рисунок 1.1 – Обсяг функціональних повноважень користувацьких ролей у системі

У межах розробленої системи для кожної категорії користувачів визначено окремий набір функціональних можливостей, що забезпечує послідовну взаємодію всіх рівнів управління:

1. Анонімний користувач. Роль призначена для відвідувачів, які не мають облікового запису. Такі користувачі можуть створити новий профіль у системі, переглядати загальні відомості про окремі замовлення (наприклад, номер, пункт відправлення чи пункт призначення) та надсилати запити до менеджера електронною поштою.

2. Зареєстрований користувач. Після авторизації користувач отримує доступ до розширеного функціоналу: може керувати вмістом кошика (додавати, редагувати або вилучати товари), переглядати історію попередніх замовлень, оновлювати особисті дані, застосовувати фільтри для пошуку продукції, створювати нові замовлення, відстежувати їхній стан або скасовувати за потреби. Крім того, користувач має можливість оформлення додаткових послуг і зв’язку з менеджером через електронну пошту.

3. Представник служби підтримки. Ця роль відповідає за супровід клієнтів і роботу з товарними позиціями. До її обов’язків належать підготовка змін у каталозі продукції з подальшим погодженням адміністратором, перегляд та коригування статусу замовлень, а також комунікація з клієнтами з метою оперативного вирішення звернень.

4. Адміністратор. Він здійснює управління обліковими записами користувачів, зокрема їх активацію чи блокування. Також він має повноваження затверджувати або самостійно виконувати операції з редагування каталогу товарів, а крім того – аналізувати статистичні дані щодо обсягу замовлень і динаміки попиту.

1.2. Програмні рішення для магазинів смарт-систем

Сьогодні ринок програмних рішень для магазинів, що займаються реалізацією комплектуючих до смарт-систем, представлений широким спектром платформ, орієнтованих на автоматизацію логістичних процесів. Кожне з таких рішень має власну архітектуру та реалізує унікальний підхід до побудови функціональних модулів і взаємодії з користувачем.

Серед найвідоміших прикладів можна виокремити системи Deagor WMS, Logiwa WMS та SyncSpider, які поєднують у собі веб-платформи для управління поставками, обліком товарів та клієнтським сервісом. Ці продукти відзначаються стабільністю роботи, масштабованістю та високим рівнем технічної підтримки, що робить їх ефективними інструментами для цифровізації логістичних операцій.

Платформа Deagor WMS [11] – це хмарне рішення для управління складськими та логістичними процесами в електронній комерції, орієнтоване на забезпечення повного контролю над ланцюгом постачання від оформлення замовлення до доставки кінцевому споживачеві. Вона інтегрує функції управління запасами, відбору товарів, пакування, вибору кур'єра та обробки повернень у єдину автоматизовану систему. Зовнішній вигляд головної сторінки веб-застосунку Deagor WMS подано на рисунку 1.2.

Рисунок 1.2 – Інтерфейс сторінки вебплатформи «Deagor WMS»

Веб-платформа Deagor WMS як програмна система реалізує такі функціональні можливості:

1. Управління запасами та складом:

● Інтеграція з різними e-commerce платформами для автоматичного надходження замовлень та товарів.

● Прогнозування попиту за допомогою алгоритмів штучного інтелекту для підтримки оптимального рівня запасів.

● Організація зберігання товарів для підвищення ефективності роботи складу.

2. Обробка замовлень:

● Централізоване управління замовленнями та автоматичне оновлення статусів.

● Обробка повернень із можливістю налаштування правил для прискорення процесу.

● Управління персоналізованими цінами та даними про клієнтів.

3. Підбір, пакування і доставка:

● Організація пакування за замовленнями чи коробками, з автоматизацією процесів для зменшення помилок.

● Відстеження посилок у реальному часі та оновлення статусу доставки.

● Автоматична генерація товарно-транспортних накладних для кур’єрських служб.

4. Аналітика та звітність:

● Генерація звітів для моніторингу продуктивності складу та ефективності виконання замовлень.

● Настроювані аналітичні панелі для підтримки управлінських рішень.

Сильні сторони веб-платформи Deagor WMS:

● Легкий у використанні інтерфейс, що спрощує навчання і щоденне використання системи.

● Автоматичне створення накладних, управління поверненнями і

відстеження відправлень знижують ймовірність помилок і підвищують ефективність.

● Використання штучного інтелекту для прогнозування попиту допомагає підтримувати оптимальний рівень запасів і уникати дефіциту або надлишку товарів.

Слабкі сторони веб-платформи Deagor WMS:

● Висока вартість впровадження може бути бар'єром для малого бізнесу.

● Потреба в інтеграції з існуючими ERP та іншими системами може вимагати додаткових ресурсів і часу.

● Номенклатура представленої продукції є відносно обмеженою, що може зменшувати привабливість платформи для широкого кола користувачів і обмежує можливості залучення нових клієнтів.

Logiwa WMS [12] – хмарна система управління складом (Warehouse Management System), розроблена для комплексного контролю над товарними потоками та логістичними операціями. Платформа забезпечує централізоване управління запасами, автоматизацію складських процесів, інтеграцію з платформами електронної комерції та аналітичні інструменти для оцінки ефективності роботи складу.

Особливістю Logiwa WMS є використання алгоритмів оптимізації розташування товарів та інтелектуальних правил обробки замовлень, що дозволяє підвищити швидкість обробки замовлень, зменшити ризик помилок та покращити обслуговування клієнтів. Система також підтримує мобільні пристрої та сканери штрих-кодів, що робить роботу співробітників більш гнучкою та продуктивною. Зовнішній вигляд сторінки веб-платформи Logiwa WMS подано на рисунку 1.3.

Веб-платформа Logiwa WMS як програмна система реалізує такі функціональні можливості:

1. Управління запасами та складом:

● Підтримує автоматичне отримання, зберігання, підбір, пакування та

відправку товарів, зменшуючи людські помилки та підвищуючи швидкість виконання замовлень.

- Використовує алгоритми для оптимального розміщення товарів, враховуючи швидкість обігу та попит, що підвищує ефективність зберігання та обробки замовлень.

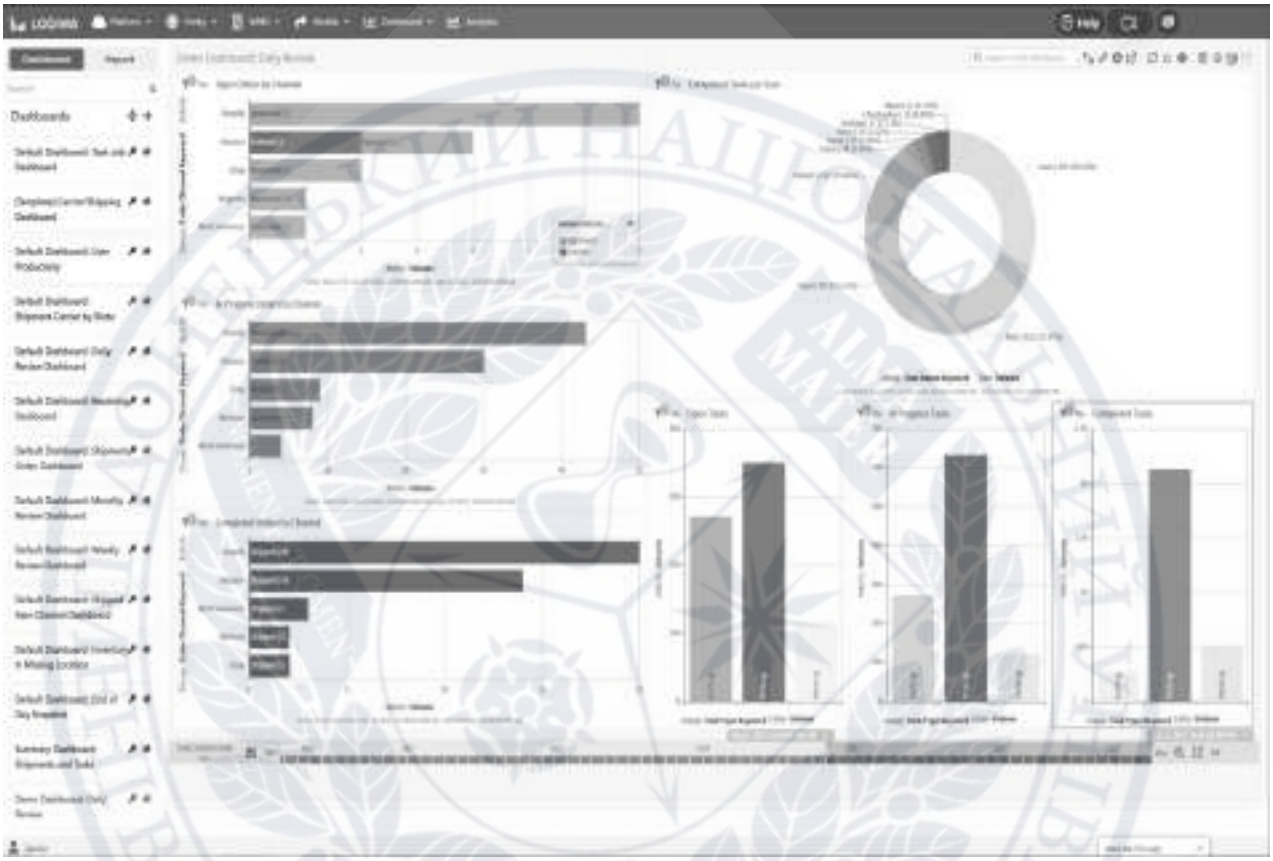

Рисунок 1.3 – Інтерфейс сторінки вебплатформи «Logiwa WMS»

Мобільний доступ та аналітика:

- Працівники можуть використовувати мобільні пристрої для реєстрації операцій у реальному часі, що забезпечує точність даних та оперативність.
- Платформа надає потужні інструменти для моніторингу ефективності, прогнозування попиту та оптимізації операцій.
- Прогнозування попиту та планування запасів на основі історичних даних.
- Можливість відстеження пересування товарів платформи між

складами.

3. Інтеграція з платформами електронної комерції:

● Сумісність з Shopify, Magento, WooCommerce та іншими дозволяє автоматично синхронізувати замовлення та оновлювати рівні запасів.

● Підтримка обробки різних типів замовлень (B2C, B2B, e-commerce).

Сильні сторони веб-платформи Logiwa WMS:

● Підходить для малого та середнього бізнесу, а також для великих підприємств, що займаються багатоканальним виконанням замовлень.

● Можливість налаштування автоматичних правил для підбору, пакування та відправки товарів підвищує ефективність та зменшує помилки.

● Можливість управління кількома складами з єдиного інтерфейсу забезпечує централізоване управління запасами та замовленнями.

● Мобільні інструменти та підтримка сканерів прискорюють комплектацію і приймання товарів.

Слабкі сторони вебплатформи Logiwa WMS:

● Обмежена кількість стандартних функцій: деякі функції, такі як брендування або налаштування електронної пошти, можуть вимагати додаткових витрат.

● Потреба в інтеграції з існуючими ERP та іншими системами може вимагати додаткових ресурсів і часу.

● Обмежена локалізація деяких функцій, відповідно, не всі мови та валютні стандарти можуть бути підтримані одразу.

SyncSpider B2B Portal [13] – це веборієнтована платформа для автоматизації інтеграцій та обміну даними між різними бізнес-системами та сервісами. Вона призначена для оптимізації роботи компаній, які використовують одночасно декілька інструментів, таких як e-commerce платформи, CRM, ERP, бухгалтерські системи та маркетингові сервіси. Головною метою даної платформи є забезпечення безперервного, безпомилкового та централізованого обміну інформацією між системами без

потреби у ручному введенні даних. Зовнішній вигляд сторінки веб-застосунку інтернет-магазину SyncSpider подано на рисунку 1.4, що ілюструє основні елементи інтерфейсу та функціональні блоки платформи.

Рисунок 1.4 – Інтерфейс сторінки вебплатформи «SyncSpider»

Вебплатформа SyncSpider як програмна система реалізує такі функціональні можливості:

1. Автоматизація бізнес-процесів:

- Створення правил автоматичного оновлення даних та планування регулярної синхронізації без ручного втручання.
- Можливість налаштування умов для обробки замовлень, оновлення цін, зміни статусів та обліку запасів.

2. Підтримка складської логістики:

- Наявність відстеження запасів товарів на декількох складах у режимі реального часу.
- Можливість інтеграції з WMS та ERP для централізованого обліку товарів.

3. Моніторинг та звітність:

● Сповіщення про помилки чи збої під час передачі даних, а також статистика про успішні та невдалі інтеграції.

● Відстеження всіх процесів синхронізації платформи у реальному часі.

Сильні сторони веб-платформи SyncSpider:

● Широка сумісність з платформами та сервісами дозволяє інтегрувати різноманітні системи без додаткового кодування.

● Моніторинг та повідомлення про помилки дозволяє швидко реагувати на неполадки та уникати збоїв у бізнес-процесах.

● Підтримка багатьох акаунтів і мульти-магазинів зручно для підприємств з кількома каналами продажу.

Слабкі сторони веб-платформи SyncSpider:

● Вартість підписки може бути високою для малого бізнесу з обмеженим бюджетом.

● Обмеженість кастомізації деяких готових шаблонів через яку необхідно створювати власні інтеграції для специфічних бізнес-процесів.

ВИСНОВКИ ДО РОЗДІЛУ 1

Реалізація вебплатформи логістичного забезпечення діяльності магазину компонентів смарт-систем потребує глибокого розуміння об’єкта, предмета та мети дослідження, адже саме ці елементи формують теоретичне підґрунтя та практичну структуру майбутнього рішення.

Було детально розглянуто специфіку смарт-систем, історію їх виникнення, основні принципи функціонування, а також роль у сучасному технологічному середовищі. Зокрема, було встановлено, що смарт-системи виникли як результат розвитку кібернетики, автоматизації та IOT, а їх основне призначення полягає у підвищенні ефективності управління, автоматизації процесів та адаптації систем до змін середовища. Ключовими характеристиками проєктованої системи управління логістикою є її гнучкість, адаптивність до змін у попиті, а також високий рівень безпеки та конфіденційності даних клієнтів. Система має забезпечувати користувачам зручний доступ до процесу придбання комплектуючих для смарт-систем, оптимізуючи при цьому логістичні процеси та управління запасами.

Однією із головних частиною проєктування є чітке визначення ролей користувачів системи: «Зареєстрований користувач», «Анонімний користувач», «Представник служби підтримки» та «Адміністратор», кожен із яких має власний набір функцій і прав доступу.

У процесі дослідження було здійснено порівняльний аналіз існуючих рішень: Deagor WMS, Logiwa WMS та SyncSpider B2B Portal. В результаті було визначено їх основний функціонал, сильні та слабкі сторони, що дало змогу обґрунтувати власну концепцію розробки. На основі узагальнення отриманих даних сформовано підхід до створення системи, яка поєднує найбільш ефективні функціональні можливості аналогів з урахуванням сучасних вимог до безпеки, продуктивності та зручності користування.

РОЗДІЛ 2
МЕТОДИ ТА МОДЕЛІ РЕАЛІЗАЦІЇ ВЕБПОРТАЛУ

2.1. Концептуальна модель архітектури вебпорталу

У контексті розробки вебпорталу одним із ключових архітектурних рішень є клієнт-серверна архітектура [14]. Вона передбачає розподіл системи на дві основні частини: клієнтську та серверну.

Клієнтська частина відповідає за взаємодію з користувачем, забезпечуючи інтерфейс для перегляду товарів, формування замовлень, пошуку та фільтрації продукції.

Серверна частина обробляє запити клієнтів, керує базою даних, забезпечує логіку бізнес-процесів та гарантує безпеку та цілісність даних.

Основною перевагою клієнт-серверної архітектури є чітке розділення обов’язків між компонентами системи. Клієнтська частина [15] залучає користувача до взаємодії з вебзастосунком, водночас серверна частина забезпечує централізоване зберігання даних та виконання складних обчислень.

Такий підхід дозволяє забезпечити масштабованість, оскільки сервер може обслуговувати одночасно великий обсяг клієнтських запитів, а також спрощує підтримку та оновлення системи, адже зміни у серверній логіці не впливають на роботу клієнтів.

На рисунку 2.1 наведено схематичне зображення клієнт-серверної архітектури, що демонструє принципи взаємодії між клієнтськими пристроями та сервером. Клієнти відправляють запити на сервер, який обробляє їх, звертається до бази даних та повертає відповідь, що відображається користувачу через інтерфейс.

У межах розроблення вебпорталу, що включає вебзастосунок, одним із ключових етапів є вибір архітектурного підходу, який забезпечуватиме стабільність, масштабованість та ефективність роботи системи. Серед можливих варіантів найчастіше розглядаються два підходи – монолітна архітектура [16] та архітектура мікросервісів [17].

Монолітна архітектура характеризується об’єднанням усіх складових системи в одну програмну структуру. Такий підхід спрощує процеси розгортання та адміністрування, оскільки всі компоненти функціонують у єдиному середовищі. Однак із зростанням обсягів даних і кількості користувачів система може втрачати гнучкість, а внесення змін в окремі модулі стає складнішим.

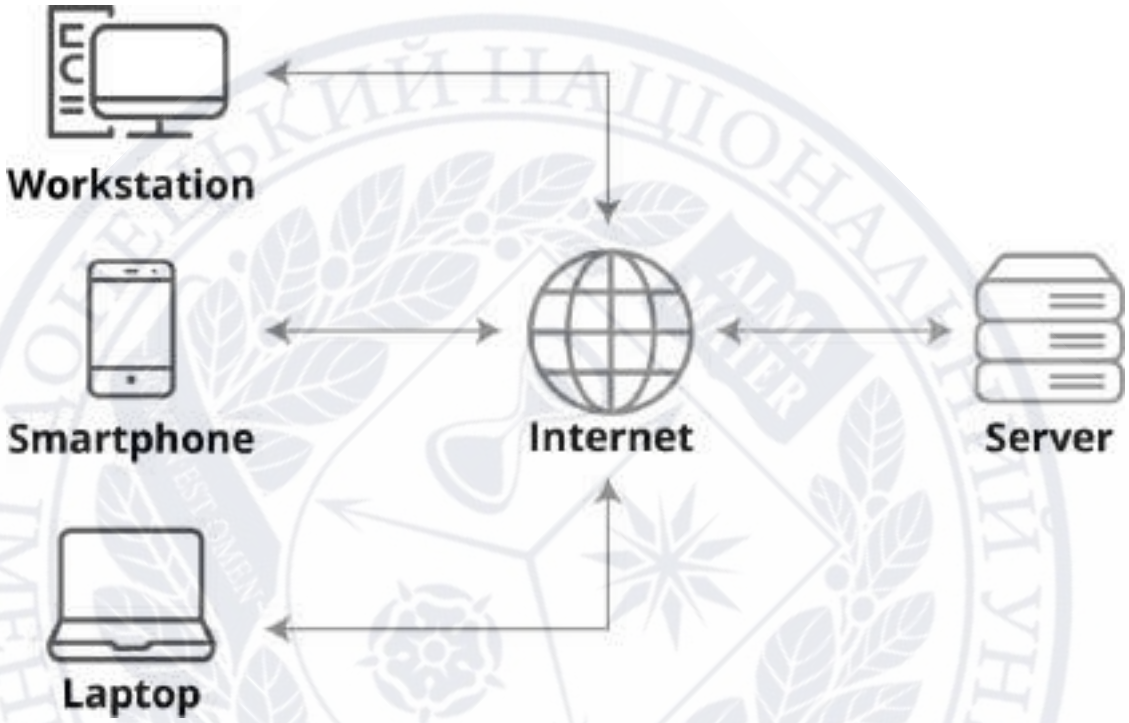

Рисунок 2.1 – Концептуальна даіграма клієнт-серверної архітектури

На противагу цьому, мікросервісна архітектура базується на поділі платформи на незалежні сервіси, кожен із яких виконує чітко визначене завдання. Це забезпечує можливість автономного оновлення компонентів, спрощує інтеграцію нових функцій і підвищує загальну адаптивність системи до змін. Такий підхід також сприяє кращій стійкості платформи до відмов окремих модулів.

На рис. 2.2 подано порівняльну схему монолітної та мікросервісної архітектур, що демонструє їхні структурні особливості та логіку взаємодії між елементами.

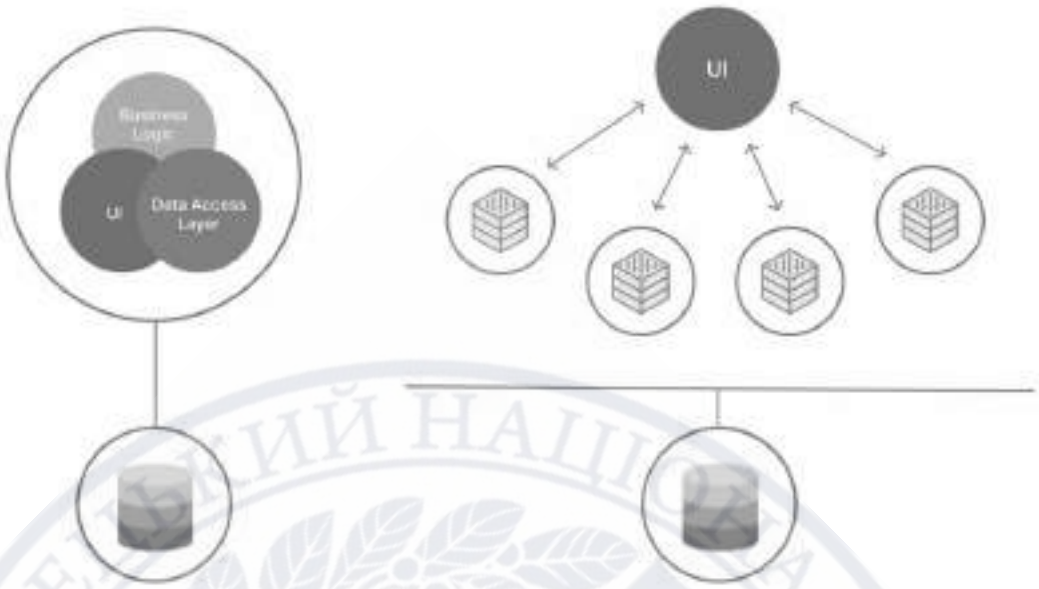

Рисунок 2.2 – Концептуальна даіграма монолітної та мікросервісної архітектур

Вибір конкретного архітектурного підходу для розроблюваного веб-застосунку здійснюється з урахуванням очікуваного навантаження, потреб у масштабованості та довгострокової підтримки системи. Застосування мікросервісної архітектури може бути особливо доцільним у випадку планування подальшого розширення функціоналу або інтеграції з іншими підсистемами, такими як системи управління складом, логістичні модулі та сервіси обробки замовлень. Саме тому даний підхід буде обраний як основний архітектурний план для побудови та підтримки досить комплексного підходу до проєктування системи.

2.2 Інструменти клієнтської частини вебпорталу

Клієнтська частина вебпорталу є ключовим компонентом інформаційної системи, оскільки саме через неї відбувається безпосередня взаємодія користувача із системою. Ефективність, зручність та інтуїтивна зрозумілість інтерфейсу безпосередньо впливають на досвід користування користувачів та

продуктивність виконання ними необхідних операцій, таких як перегляд товарів, формування замовлень та отримання інформації про стан обробки замовлень.

У межах розробки клієнтської частини вебпорталу потрібно забезпечувати логічне організування контенту, формування елементів інтерфейсу, таких як заголовки, таблиці, форми, кнопки та навігаційні блоки. Використання HTML [18] дуже пасуватиме до використання, як основний інструмент відображення інформації. HTML – базова мова розмітки, що використовується для створення структури вебсторінок і вебзастосунків. Переваги даної мови розмітки:

1. Простота освоєння. Мова HTML відзначається логічною та зрозумілою структурою, завдяки чому навіть новачки у веброзробці можуть швидко опанувати основи створення сторінок без глибоких технічних знань.
2. Повна сумісність із браузерами. HTML підтримується всіма сучасними вебоглядачами, що гарантує коректне відображення вебсайтів на різних пристроях та операційних системах.
3. Базова основа для вебтехнологій. HTML виступає каркасом для поєднання з іншими технологіями – такими як CSS для оформлення та JavaScript для інтерактивності, що дозволяє створювати адаптивні та динамічні інтерфейси.

Недоліки, які можна було б виділити у HTML:

1. Обмежена функціональність. HTML сам по собі не забезпечує динамічної поведінки вебсторінок. Для інтерактивності та обробки подій необхідно використовувати додаткові технології, зокрема JavaScript.

2. Відсутність логіки та умовних операторів. HTML призначений виключно для розмітки та структуризації контенту і не підтримує виконання умовних операцій або програмної логіки.

З огляду на роль HTML як фундаменту для структури вебсторінок, наступним критично важливим компонентом клієнтської частини є CSS [19], який забезпечує визуальне оформлення та стилізацію елементів інтерфейсу. CSS – це мова опису зовнішнього вигляду та форматування веб-документів, що

дозволяє визначати стилі для елементів HTML, включаючи кольори, шрифти, розміри, відступи, позиціонування та інші візуальні характеристики. CSS забезпечує відокремлення структури сторінки від її презентації, що підвищує зрозумілість коду, спрощує підтримку та дозволяє застосовувати однакові стилі до багатьох елементів чи сторінок одночасно. Переваги даної мови опису оформлення вебдокументів:

1. Розділення структури та стилю. Використання CSS забезпечує відокремлення логічного змісту сторінки від її візуального оформлення, що значно спрощує підтримку, модифікацію та масштабування вебресурсу.

2. Гнучкість та адаптивність. Завдяки засобам CSS можливо створювати динамічні макети, які автоматично підлаштовуються під розміри екрана користувача – від великих моніторів до смартфонів.

3. Узгоджений вигляд інтерфейсу. CSS дає змогу легко забезпечити єдиний дизайн для всіх елементів сайту, що підвищує візуальну цілісність і зручність сприйняття.

Єдиним суттєвим недоліком, який можна було б виділити у CSS – непередбачувана сумісність із різними браузерами. Деякі властивості CSS можуть відрізнятися у реалізації різними браузерами, що вимагає тестування та застосування додаткових префіксів для забезпечення коректного відображення.

Після визначення структури вебсторінки за допомогою HTML та її візуального оформлення та стилізації через CSS, наступним ключовим компонентом клієнтської частини є JavaScript. JavaScript [20] – мова програмування, що забезпечує динамічну поведінку та інтерактивність веб-платформи та суттєво розширює можливості взаємодії користувача із системою. Він дозволяє реалізовувати обробку подій, валідацію форм, асинхронне завантаження даних, динамічне оновлення вмісту сторінок без необхідності їх перезавантаження, а також управління складними елементами інтерфейсу, такими як модальні вікна, випадаючі списки та інтерактивні таблиці. Переваги даної мови програмування:

1. Кросплатформна універсальність. JavaScript є мовою, що підтримується всіма популярними веббраузерами, тому програми, створені з його використанням, можуть стабільно функціонувати на будь-яких операційних системах і пристроях без додаткових налаштувань.

2. Інтерактивність та робота з об’єктною моделлю документа DOM [21]. За допомогою JavaScript можливо динамічно змінювати вміст вебсторінки, реагувати на дії користувача та реалізовувати інтерактивні елементи інтерфейсу. Це надає вебзастосункам гнучкість і підвищує зручність взаємодії.

3. Підтримка різних парадигм програмування. Мова дозволяє використовувати як об’єктно-орієнтований, так і функціональний підходи, що робить її придатною для створення масштабованих і добре структурованих застосунків.

Недоліки, які можна було б виділити у JS:

1. Вразливість до безпеки. Код JavaScript виконується на стороні клієнта, що робить його потенційно вразливим до атак типу XSS [22] (Cross-Site Scripting) або маніпуляцій користувачем.

2. Залежність від браузера. Хоча JavaScript підтримується усіма сучасними браузерами, існують незначні відмінності у реалізації окремих функцій, що може призводити до некоректного відображення або поведінки вебсторінки без додаткової оптимізації.

2.3 Програмні та технічні ресурси для побудови серверної частини вебпорталу

Мова програмування є фундаментальним засобом формалізації алгоритмів та реалізації програмних систем, що забезпечує можливість опису логіки їхньої роботи у вигляді логічних інструкцій. Для реалізації веб-платформи обрано мову програмування Java, яка зарекомендувала себе як надійний, масштабований та багатофункціональний інструмент для розробки серверних застосунків. Java [23] – це мова ООП [4] загального призначення,

створена компанією Sun Microsystems (нині належить Oracle) у 1995 році. Переваги даної мови програмування:

1. Кросплатформна портативність. Однією з ключових особливостей Java є принцип «Write Once, Run Anywhere», який реалізується завдяки роботі через віртуальну машину JVM [24]. Це дозволяє запускати один і той самий застосунок на будь-якій операційній системі без необхідності змін у програмному коді.

2. Об’єктно-орієнтований підхід. Мова Java повністю побудована на засадах ООП, що сприяє структурованій організації програм, повторному використанню коду, легшому тестуванню та масштабуванню великих проєктів.

3. Безпечність і стабільність. Завдяки суворій типізації, автоматичному збиранню сміття (Garbage Collector [25]) і системі обробки винятків Java забезпечує високу надійність виконання програм та зменшує кількість критичних помилок під час роботи.

4. Підтримка багатопоточності. Java має вбудовані засоби для створення багатопотокових застосунків, що дозволяє ефективно використовувати ресурси процесора та забезпечує паралельне виконання завдань у реальному часі.

5. Розвинена екосистема та бібліотеки. Мова має обширну стандартну бібліотеку, а також тисячі сторонніх фреймворків і модулів, які охоплюють роботу з мережею, базами даних, UI, безпекою та аналітикою. Це суттєво скорочує час розробки та підвищує якість кінцевих продуктів.

Єдиним суттєвим недоліком, який можна було б виділити у Java – високе споживання ресурсів. Java-програми працюють через JVM, що збільшує витрати пам’яті та процесорного часу.

У якості системи збирання та керування залежностями використано Gradle. Gradle [26] – це система автоматизації збирання проєктів, яка використовується для компіляції, тестування, пакування та розгортання програмного забезпечення. Переваги даного інструменту для збирання проєктів:

1. Декларативна модель конфігурації. Gradle використовує гнучкий механізм опису структури проєкту, його залежностей і завдань через декларативний синтаксис. Це дозволяє розробникам чітко визначати логіку збірки, підвищуючи прозорість та керованість процесу.

2. Розширюваність і масштабованість. Завдяки підтримці мов Groovy та Kotlin, Gradle може бути налаштований під конкретні потреби проєкту. Такий підхід забезпечує можливість реалізації як базових, так і складних сценаріїв автоматизації для проєктів будь-якого розміру.

3. Сумісність з екосистемою розробки. Gradle тісно інтегрується з більшістю популярних середовищ розробки (IntelliJ IDEA, Eclipse, Android Studio) та систем контролю версій (Git, SVN). Це забезпечує зручність використання в існуючих робочих процесах і підвищує ефективність командної роботи.

Єдиним суттєвим недоліком, який можна було б виділити у Gradle – час, потрібний на збирання дуже об’ємних проєктів. Для великих проектів перші збірки Gradle можуть бути відносно повільними, оскільки система завантажує всі залежності, плагіни та виконує ініціалізацію.

Основою для побудови архітектури системи обрано Spring Framework [27], що забезпечує зручну інверсію керування, модульність і високу гнучкість при створенні веб- та корпоративних застосунків. Spring Framework – це потужний фреймворк для розробки застосунків на Java, який надає комплексну інфраструктуру для створення гнучких, масштабованих і легко підтримуваних програм. Переваги даного фреймворку:

1. Принцип інверсії керування та впровадження залежностей. Spring реалізує концепції Inversion of Control (IoC) і Dependency Injection (DI) [28], що забезпечує слабку зв’язність між компонентами застосунку. Такий підхід підвищує модульність системи, спрощує тестування та покращує підтримуваність коду.

2. Гнучка модульна структура. Архітектура Spring побудована за модульним принципом, що дозволяє підключати лише необхідні компоненти, наприклад, Spring Boot, Spring Security, Spring Data чи Spring MVC. Це оптимізує ресурсне навантаження й спрощує конфігурацію застосунку.

3. Сумісність з різними технологіями. Фреймворк підтримує інтеграцію з провідними технологіями – від ORM-рішень (JPA, Hibernate) до інструментів побудови мікросервісів та REST API. Це робить його універсальним середовищем для створення корпоративних і веб-систем.

4. Активна екосистема та підтримка спільноти. Spring має масштабну спільноту користувачів і розробників, багату базу знань, приклади реалізацій і відкриту документацію. Це забезпечує стабільність розвитку, постійне оновлення функцій і наявність готових рішень для типових задач.

Недоліки, які можна було б виділити у Spring framework:

1. Складність конфігурації у великих проєктах. Попри спрощення, які надає Spring Boot, налаштування великого корпоративного застосунку може бути трудомістким. Підтримка численних конфігураційних файлів, профілів та залежностей іноді потребує додаткових зусиль.

2. Витрати ресурсів. Spring Framework використовує велику кількість об’єктів та складну структуру, що може призводити до збільшеного споживання пам’яті та тривалішого часу запуску застосунку.

У сучасних вебзастосунках дуже важлива швидкодія передачі інформації для позитивного користувацького досвіду. Саме тому важливо використовувати кешування даних у системах такого типу. Caffeine Cache [29] – це високопродуктивна бібліотека кешування для Java, призначена для зберігання тимчасових даних у пам’яті з метою прискорення доступу до часто використовуваної інформації. Переваги даного інструменту кешування:

1. Висока продуктивність. Caffeine є однією з найшвидших бібліотек кешування для Java, оскільки використовує ефективні алгоритми, зокрема

Window TinyLFU (W-TinyLFU). Це забезпечує оптимальний баланс між швидкістю доступу до даних і ефективним використанням пам’яті.

2. Гнучкі політики видалення. Бібліотека підтримує динамічне керування даними в кеші – автоматично видаляє найменш використовувані або застарілі об’єкти, що допомагає підтримувати актуальність даних і зменшує навантаження на систему.

3. Мінімальні залежності. Бібліотека є легковаговою, не потребує складних зовнішніх залежностей і може бути легко додана до будь-якого Java-проєкту.

Єдиним суттєвим недоліком, який можна було б виділити у Caffeine cache – локальність кешу. Caffeine є локальним механізмом кешування, тобто зберігає дані лише в межах одного екземпляра застосунку. Це означає, що у розподілених або кластерних системах кеш не синхронізується між різними вузлами без додаткових рішень.

Для ефективної розробки програмного забезпечення та управління складними проектами критично важливо використовувати зручне і функціональне IDE. IntelliJ IDEA [30] – високопродуктивне інтегроване середовище розробки для мови програмування Java та інших сумісних мов, розроблене компанією JetBrains. Переваги даного середовища розробки:

1. Розумне автодоповнення коду. IntelliJ IDEA забезпечує інтелектуальне автодоповнення, яке не лише пропонує змінні, а й аналізує контекст коду.

2. Потужна навігація та рефакторинг. IDE дозволяє швидко переміщатися між класами, методами, файлами та залежностями, а також має широкий набір інструментів для рефакторингу – безпечної зміни структури коду без порушення його працездатності.

3. Інтеграція з популярними технологіями. IntelliJ IDEA підтримує велику кількість фреймворків і мов програмування – зокрема Java, Spring, Hibernate, JavaScript, SQL тощо. Вона також інтегрується з Gradle та базами даних, що робить її універсальним інструментом для повного циклу розробки.

Недоліки, які можна було б виділити у Intellij Idea:

1. Високе споживання ресурсів. IntelliJ IDEA є досить вимогливою до апаратного забезпечення. Вона споживає значний обсяг оперативної пам’яті та процесорного часу, особливо під час індексації великих проєктів або роботи з великою кількістю плагінів.

2. Платна версія для повного функціоналу. Хоча існує безкоштовна версія Community Edition, багато розширених функцій (зокрема підтримка Spring, Java EE, баз даних, вебтехнологій) доступні лише у комерційній версії Ultimate Edition.

2.4 Засоби реалізації доступу до бази даних

При розробці систем БД є ключовим елементом сучасних інформаційних технологій, що забезпечує структурування даних, швидкий доступ до інформації та її ефективне зберігання. Вся інформація, необхідна для персоналізації акаунтів користувачів, персональні дані клієнтів, тощо зберігаються безпосередньо у базі даних. Сфера застосування є досить обширною, починаючи від веброзробки та впровадження мобільних додатків, бізнес-аналітикою, закінчуючи управління підприємницькими процесами, електронну комерцію та системи підтримки прийняття рішень.

У наданні переваги використання між SQL та NoSQL базами даних є специфічні вимоги та завдання для конкретного проєкту. NoSQL-бази даних доцільно застосовувати у випадках, коли структура даних є нестандартною, динамічною та потребує продуктивності високого рівня. Натомість SQL-бази даних, як правило, є оптимальним рішенням для систем із чітко визначеними структурованими даними, що передбачають виконання комплексних запитів та встановлення зв’язків між конкретними таблицями.

У межах розробки даного вебпорталу було обрано MySQL, одного з найпоширеніших представників SQL-баз даних, що забезпечує стабільність, надійність та ефективне керування структурованими даними. MySQL [31] –

реляційна система управління базами даних (СУБД), що використовує мову структурованих запитів SQL (Structured Query Language) для створення, модифікації та обробки даних. Переваги даної СУБД:

1. Підтримка різних платформ і інтеграцій. MySQL сумісний з більшістю операційних систем і легко підключається до різних мов програмування та фреймворків, таких як Java, PHP, Python або Spring, що робить його гнучким рішенням для будь-яких проєктів.

2. Продуктивність при високих навантаженнях. СУБД здатна швидко обробляти великі обсяги інформації та одночасно обслуговувати численних користувачів, забезпечуючи стабільну роботу навіть у масштабних системах.

3. Захист даних і контроль доступу. MySQL пропонує комплекс механізмів безпеки, включаючи автентифікацію користувачів, управління правами доступу та шифрування даних, що гарантує надійний захист інформації та її цілісність.

Єдиним суттєвим недоліком, який можна було б виділити у MySQL – обмежені можливості масштабування горизонтально. MySQL підтримує реплікацію та шардінг, горизонтальне масштабування у порівнянні з NoSQL-базами даних може бути складнішим і потребує конфігурації.

У межах розробки вебпорталу логістичного забезпечення діяльності магазину компонентів смарт-систем передбачено інтеграцію інструменту Hibernate для взаємодією з базою даних. Hibernate [32] – ORM, яке забезпечить ефективну взаємодію між об’єктно-орієнтованою моделлю застосунку та реляційною базою даних MySQL, спрощуючи процес збереження, оновлення та отримання даних. Переваги даної ORM:

1. Об’єктно-реляційне відображення (ORM). Hibernate забезпечує автоматичне перетворення об’єктів Java на таблиці бази даних і навпаки, дозволяючи розробникам працювати з даними на рівні об’єктів, а не через складні SQL-запити.

2. Транзакційність та безпека даних. Фреймворк підтримує коректне виконання транзакцій, забезпечує цілісність інформації та захищає дані від втрат або пошкоджень під час системних збоїв.

3. Оптимізація запитів і кешування. Hibernate надає механізми кешування та оптимізації запитів до бази даних.

Суттєвих недолоків у даній ORM не виявлено, але наявне зниження продуктивності при складних запитах.

Під час внесення змін у базу даних протягом життєвого циклу проєкту забезпечення цілісності та узгодженості даних є досить важливим фактором. Саме тому, використання інструменту міграції баз даних є необхідним. Liquibase [33] – інструмент міграції баз даних, який забезпечує відтворюваність і контроль версій бази даних. Liquibase є системою управління змінами у схемі бази даних, яка дозволяє автоматизувати процес оновлення структури БД, синхронізуючи її стан із актуальною версією застосунку. Переваги даного інструменту міграції БД:

1. Гнучке описання змін у базі даних. Liquibase дозволяє створювати міграції у форматах XML, YAML, JSON або SQL, що забезпечує зручність для різних стилів розробки та інтеграції у процеси CI/CD.

2. Сумісність із різними СУБД. Інструмент працює з численними реляційними базами даних, включаючи MySQL, PostgreSQL, Oracle, SQL Server та інші, що гарантує портативність.

Єдиним суттєвим недоліком, який можна було б виділити у Liquibase – висока вимогливість до дисципліни розробників. Ефективне використання Liquibase потребує чіткої організації процесу роботи з файлами змін (changelog). Недотримання узгоджених правил або невчасне оновлення changelog-файлів може призвести до конфліктів і помилок при міграцій.

ВИСНОВКИ ДО РОЗДІЛУ 2

У цьому розділі проведено детальний аналіз архітектури та інструментів, необхідних для створення системи управління логістикою магазину компонентів смарт-систем. Визначено ключові структурні елементи системи та взаємозв’язки між ними, що дозволило обґрунтувати доцільність застосування клієнт-серверної архітектури як базового підходу. Така модель сприяє ефективному розподілу навантаження, підвищує масштабованість і полегшує адміністрування даних.

Особливу увагу приділено вибору між монолітною та мікросервісною архітектурою. Для даної платформи обрано мікросервісний підхід, що забезпечує автономність модулів, полегшує масштабування та інтеграцію з іншими підсистемами, а також підвищує стійкість роботи системи під час зміни функціональних вимог.

Для розробки клієнтської частини рекомендовано використання HTML, CSS та JavaScript. HTML формує базову структуру сторінок, CSS відповідає за оформлення та адаптивність інтерфейсу, а JavaScript додає інтерактивність, забезпечуючи користувачам динамічну взаємодію з платформою. Такий підхід дозволяє створити зрозумілий і зручний інтерфейс.

Серверна частина платформи буде реалізована на Java з використанням Spring Framework та Gradle для керування залежностями та збірки проекту. Це рішення забезпечує масштабованість, модульність та стабільну інтеграцію з іншими компонентами системи. Використання Caffeine Cache дозволить прискорити доступ до даних і знизити навантаження на базу.

У якості СУБД обрана MySQL, яка підтримує високу продуктивність та надійність. Для зручності роботи з даними використовується Hibernate ORM та JOOQ, що спрощує взаємодію між об’єктною моделлю та реляційною базою. Управління версіями та змінами структури бази здійснюватиметься за допомогою Liquibase.

РОЗДІЛ 3
ПРОЄКТУВАННЯ СТРУКТУРИ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБПОРТАЛУ

3.1. Моделювання структурних компонентів вебпорталу

Формування моделі системи є ключовим етапом проєктування, оскільки вона дозволяє чітко уявити внутрішню структуру платформи, взаємозв'язки між її компонентами та визначити основні потоки інформації. Такий підхід сприяє ефективному плануванню архітектури, підвищує узгодженість системи та забезпечує її гнучкість і масштабованість для майбутніх модифікацій.

Концептуальна модель [34] розроблюваної платформи для логістичного забезпечення магазину компонентів смарт-систем наочно демонструє ключові мікросервіси, їхні функції та взаємодію між собою, що створює цілісне уявлення про роботу вебпорталу (рисунок 3.1).

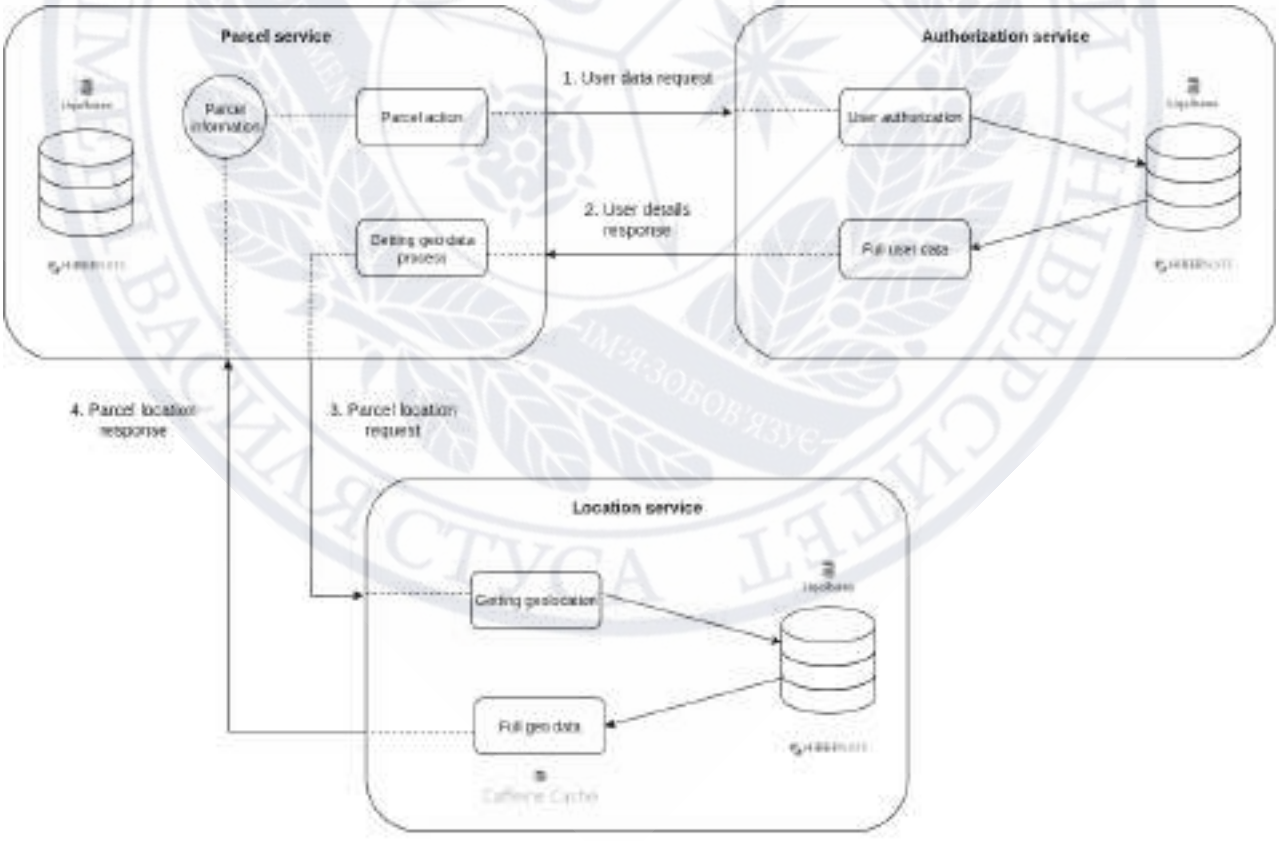

Рисунок 3.1 – Концептуальна модель структури вебпорталу

Мікросервіси у вебпорталі поділені за функціональними напрямами:

● Сервіс авторизації – відповідає за реєстрацію користувачів, автентифікацію та управління сесіями.

● Сервіс розташування – забезпечує збереження та оновлення інформації про місцезнаходження посилок, а також використовує механізм кешування на основі Caffeine Cache для швидкого доступу до даних.

● Сервіс дій із посилками – обробляє всі операції, пов’язані з посилками, включно з отриманням, переглядом історії переміщень, скасуванням чи повторним відправленням.

Для керування змінами у базах даних сервісів авторизації та розташування застосовується Liquibase, що спрощує процес впровадження міграцій та підтримує цілісність структури БД.

Для аутентифікації користувачів використовується технологія JWT [35], яка дозволяє видати токен доступу на обмежений проміжок часу (наприклад, 2–3 години), що значно зменшує кількість запитів до сервісу авторизації. Такий підхід забезпечує безпечний та ефективний контроль доступу користувачів.

Діаграми варіантів використання (Use-Case-діаграми [36]) виступають потужним інструментом комунікації між замовниками, розробниками та іншими зацікавленими сторонами проєкту. Вони забезпечують наочне відображення сценаріїв взаємодії користувачів із системою, визначаючи основні функціональні можливості та межі її застосування. Завдяки такому підходу підвищується рівень розуміння вимог до програмного забезпечення, що, у свою чергу, сприяє зниженню ризиків на етапах розроблення та впровадження системи. Use-Case-діаграми є фундаментальним елементом процесу аналізу та проєктування, оскільки вони дозволяють структурувати вимоги, визначити ролі користувачів і взаємозв’язки між ними та компонентами системи. На рисунку 3.2 подано діаграму логістичного забезпечення діяльності магазину компонентів смарт-систем.

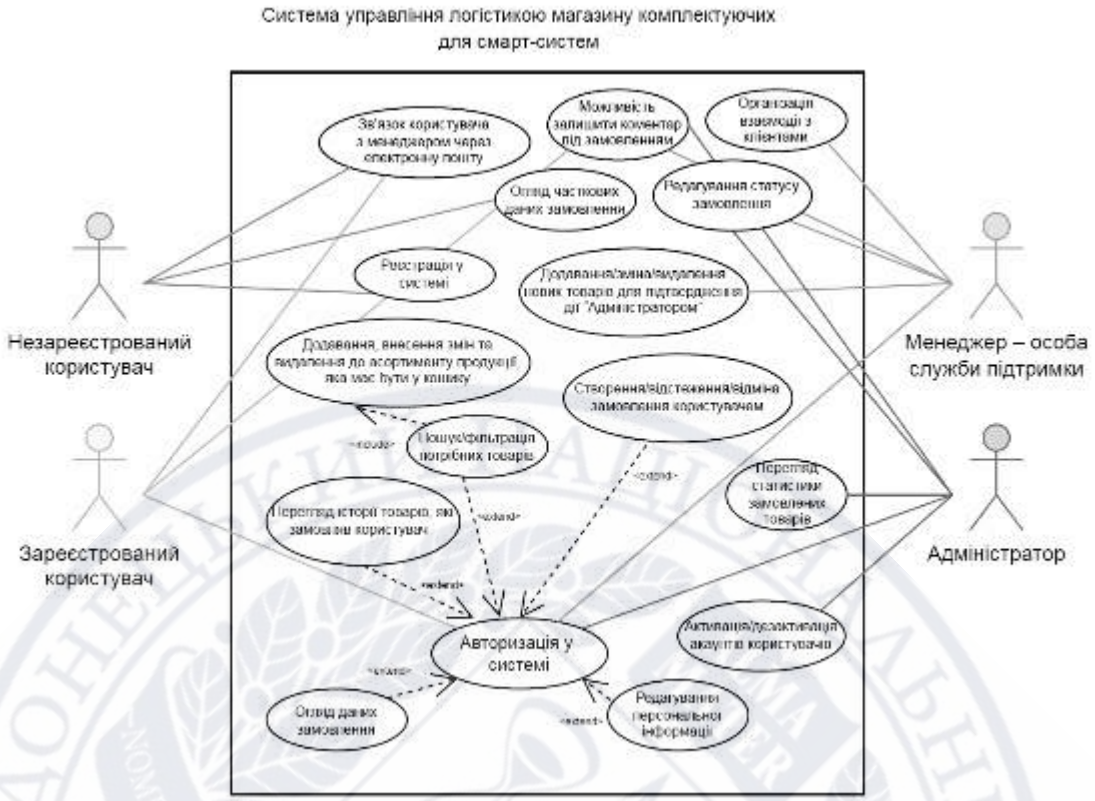

Рисунок 3.2 – Діаграма прецедентів, що відображає логіку взаємодії користувачів із вебпорталом

Діаграми варіантів використання (Use-Case-діаграми) виступають ефективним інструментом комунікації між замовниками, розробниками та іншими зацікавленими сторонами проєкту. Вони забезпечують наочне відображення сценаріїв взаємодії користувачів із системою, визначаючи основні функціональні можливості.

Роль «Анонімний користувач» характеризується обмеженим рівнем доступу до функціональних можливостей системи, оскільки такий користувач не проходить процедуру автентифікації та не ідентифікується системою як зареєстрований користувач. Функціонал цієї ролі включає базові можливості взаємодії із системою, зокрема: зв’язок із службою підтримки за допомогою електронної пошти, реєстрація в системі для отримання розширених прав доступу, а також перегляд обмеженої кількості даних щодо замовлення без можливості внесення змін чи здійснення транзакцій.

Роль «Анонімнмий користувач» має мінімальний набір можливостей, оскільки система не розпізнає його як автентифікованого. До базового функціоналу цієї ролі належать: можливість надіслати запит на контакт із менеджером через електронну пошту, створити новий обліковий запис у системі та переглядати обмежену інформацію щодо поточного стану замовлення.

Після реєстрації користувач переходить до ролі «Зареєстрований клієнт», що передбачає ширший спектр дій. Такий користувач може входити до системи, переглядати історію власних покупок, виконувати пошук і фільтрування товарів, а також управляти кошиком – додавати нові позиції, редагувати або видаляти їх. Крім того, передбачено можливість створювати, скасовувати чи відстежувати замовлення, а також оновлювати персональні дані профілю та переглядати детальну інформацію щодо кожного оформленого замовлення.

Роль «Менеджер служби підтримки» (представник технічного персоналу) отримує всі можливості, доступні звичайному зареєстрованому користувачу, але не має опції зв’язку зі службою підтримки через електронну пошту. Натомість менеджеру доступні додаткові адміністративні функції: редагування асортименту товарів, зміна статусів замовлень, координація запитів клієнтів та взаємодія з ними через внутрішню систему обміну повідомленнями.

Найвищий рівень доступу має роль «Адміністратор». Вона включає всі можливості менеджера, проте без дій, пов’язаних із підтвердженням змін товарів чи безпосереднім управлінням клієнтськими запитами. Адміністратор має право переглядати статистичні дані щодо динаміки продажів, аналізувати активність користувачів, а також активувати або деактивувати облікові записи інших учасників системи.

Моделі алгоритмічних процесів [37] слугують фундаментальним інструментом для формалізації, аналізу та оптимізації діяльності системи. Вони відображають логічну послідовність етапів, методів і взаємозв’язків, які забезпечують виконання основних функціональних завдань. Завдяки побудові таких моделей можливо ідентифікувати вузькі місця, дублювання операцій та

потенційні точки оптимізації. Візуалізація робочих процесів у вигляді діаграм сприяє підвищенню прозорості бізнес-логіки системи, забезпечує узгоджене розуміння структури процесів між усіма учасниками розробки та експлуатації, а також створює основу для подальшої автоматизації та вдосконалення управління системними компонентами.

Також на основі моделей можна здійснювати тестування різних сценаріїв поведінки системи, що дозволяє виявити можливі помилки або неефективні рішення ще на етапі проектування. Впровадження таких моделей у процес розроблення забезпечує підвищення якості кінцевого продукту, спрощує підтримку та оновлення системи, сприяє масштабованості архітектури та допомагає у визначенні критеріїв успішного завершення кожного етапу процесу.

На рисунку 3.3 наведено схему, що демонструє основні етапи проходження користувачем процедур автентифікації та реєстрації у системі. Візуалізовано логіку перевірки введених даних, підтвердження автентичності користувача та надання доступу до функціональних можливостей платформи. Процес включає перевірку правильності заповнених полів, унікальності облікових даних і надання повідомлень у разі виявлення помилок, що сприяє покращенню взаємодії користувача із системою та забезпечує цілісність інформації.

Користувач, який ініціює процес реєстрації в системі, повинен заповнити форму, вказавши необхідні персональні дані для створення власного облікового запису. Після введення інформації система здійснює валідацію отриманих даних з метою перевірки їх коректності та унікальності. У разі виявлення помилок або невідповідностей, користувач отримує автоматичне повідомлення про некоректність введених даних із рекомендаціями щодо їх виправлення, що підвищує зручність взаємодії та забезпечує цілісність інформації в базі даних.

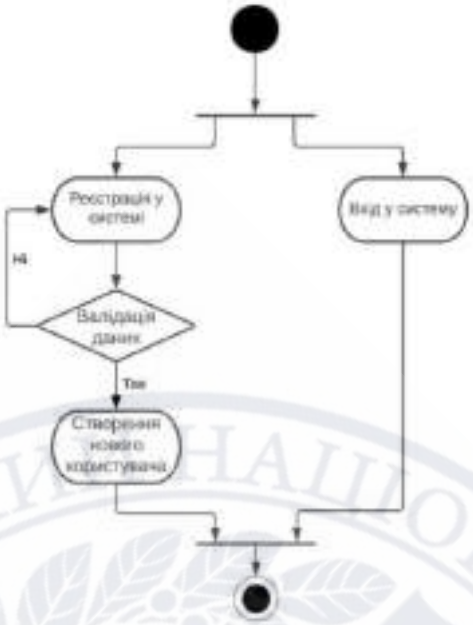

Рисунок 3.3 – Графічна модель процесів доступу користувачів до вебпорталу

На рисунку 3.4 представлено діаграму, яка відображає послідовність дій користувача під час здійснення пошуку певного товару в системі. Дана діаграма демонструє логіку взаємодії між користувачем та інтерфейсом веб-платформи – від моменту введення пошукового запиту до отримання релевантних результатів. Такий підхід дає змогу чітко простежити механізм обробки запиту, включно з етапами фільтрації, сортування та відображення знайдених позицій, що сприяє підвищенню ефективності навігації й покращує загальний користувацький досвід у межах системи.

Після проходження процедури реєстрації або авторизації в системі користувач отримує доступ до функціоналу пошуку необхідних товарів. Процес пошуку реалізовано таким чином, щоб забезпечити максимальну зручність і гнучкість у взаємодії з інтерфейсом. Користувач може здійснювати пошук за ключовими словами, певними характеристиками або параметрами товару, а також комбінованим способом, що дозволяє отримувати найбільш релевантні результати. Такий підхід сприяє підвищенню ефективності взаємодії

користувача із системою, зменшує час на пошук потрібного продукту та забезпечує інтуїтивно зрозумілу логіку використання функціоналу.

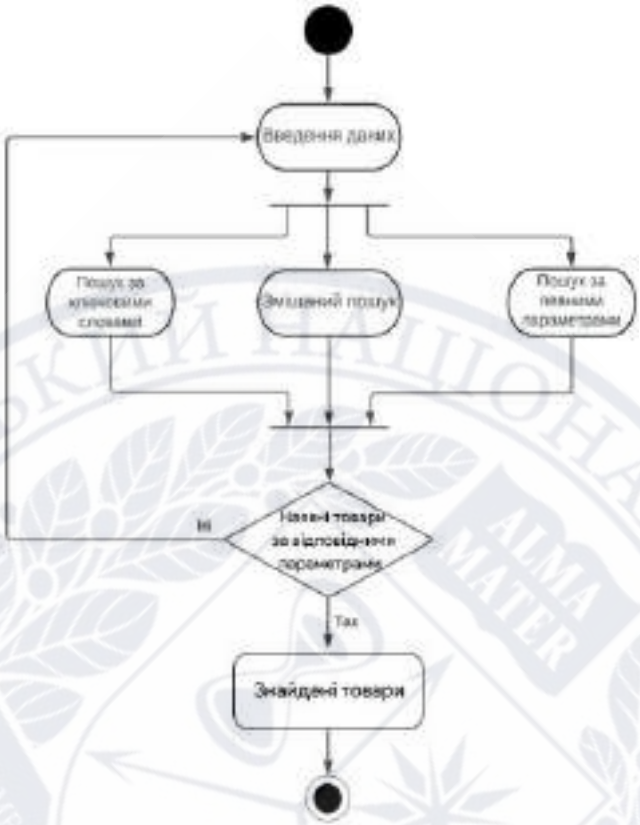

Рисунок 3.4 – Графічна модель процесів пошуку товарів

На рисунку 3.5 наявна ілюстрація діаграми виду діяльності, що відображає послідовність дій користувача під час процесу придбання товару у системі. Діаграма демонструє ключові етапи, починаючи від вибору товару та його додавання до кошика, продовжуючи оформленням замовлення, вибором способу оплати та завершуючи підтвердженням покупки.

Ідетифікуючи потрібний товар, користувач отримує можливість придбати його. На сторінці кожного товару передбачено відображення його наявності, що дозволяє оперативно оцінити можливість покупки. Користувач може додати обраний товар до кошика разом із іншими позиціями або оформити замовлення лише на конкретну одиницю цього товару. Такий підхід забезпечує гнучкість у процесі формування замовлення та підвищує зручність взаємодії користувача з веб-платформою, сприяючи оптимізації користувацького досвіду та ефективності процесу покупки.

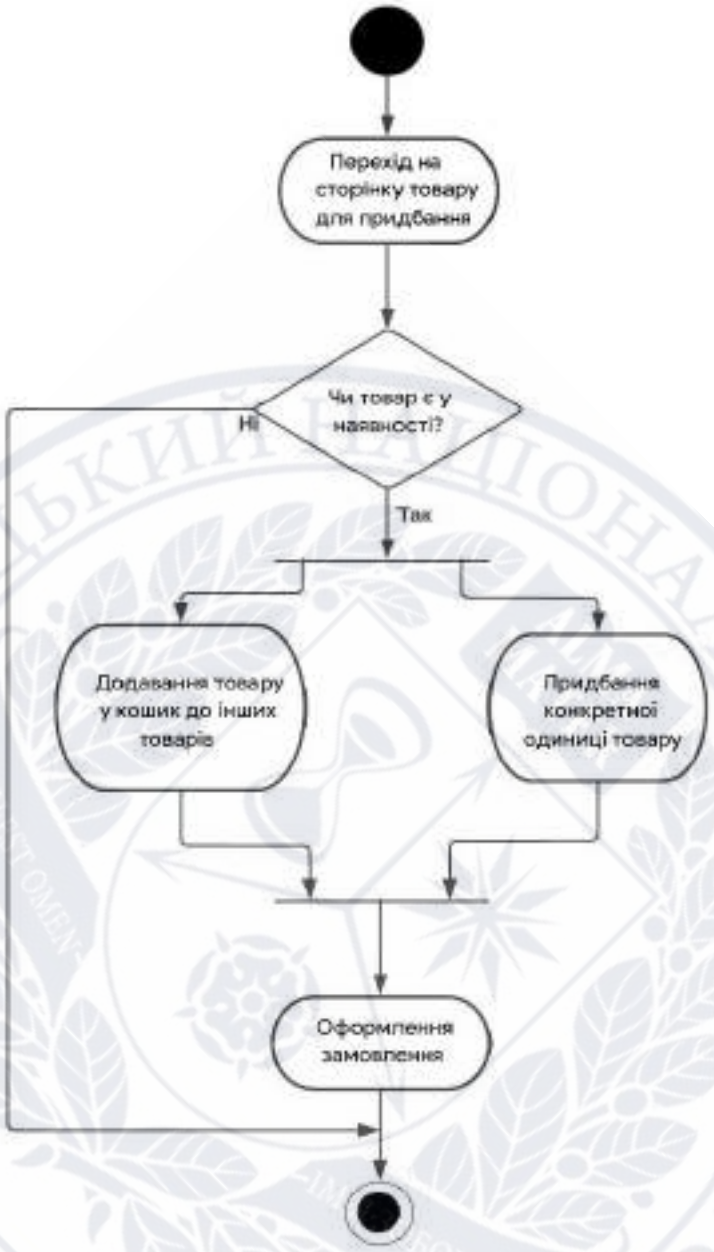

Рисунок 3.5 – Графічна модель процесів придбання товару

На рисунку 3.6 ілюструє діаграму виду діяльності, що описує послідовність операцій та процесів, пов’язаних із доставкою товару. Діаграма відображає ключові етапи від підтвердження замовлення до фактичного отримання продукції користувачем, включаючи обробку логістичних даних, маршрутизацію доставки та контроль за станом посилки. Такий візуальний опис сприяє кращому розумінню логістичного процесу та дозволяє оптимізувати взаємодію між компонентами системи.

Рисунок 3.6 – Графічна модель процесів надсилання товару

Оформлюючи замовлення, користувач зобов’язаний надати персональні дані, необхідні для доставки, включаючи точну адресу отримання. У разі невідповідності введеної інформації встановленим критеріям валідності система відобразить відповідне повідомлення про помилку. Після коректного введення даних посилка отримує статус «готова до відправлення». Користувач також має можливість обрати додаткові послуги для своєї посилки, наприклад страхування або спеціальну обробку. Після завершення цього етапу посилка переходить до стадії відправлення та доставки кінцевому отримувачу.

3.2. Розробка бази даних

Планування та організація інформаційного сховища даних [38] є невід’ємним компонентом побудови комплексної інформаційної системи, що забезпечує структуроване зберігання, систематизацію та раціональне управління інформаційними потоками, які надходять із зовнішніх джерел. Коректно спроєктоване сховище визначає рівень ефективності функціонування системи, стабільність доступу до даних, продуктивність виконання запитів та

надійність збереження критичної інформації. Рівень організації сховища даних створює основу для масштабованості системи, підвищення якості аналітичної обробки даних і формування інформаційного підґрунтя для прийняття обґрунтованих управлінських рішень у межах загальної архітектури програмного комплексу.

У контексті аналітичної діяльності сховище даних виконує стратегічну роль, оскільки забезпечує можливість побудови звітів, прогнозних моделей і систем підтримки прийняття рішень. Завдяки добре організованій архітектурі сховища знижується навантаження на основні транзакційні системи, що підвищує загальну ефективність роботи програмного комплексу. Саме через цей фактор, грамотне планування, проектування та адміністрування сховища даних виступає фундаментом стабільності, надійності та гнучкості всієї інформаційної системи, забезпечуючи її готовність до масштабування та інтеграції з новими компонентами у майбутньому.

На рисунку 3.7 представлено схему бази даних сервісу, що відповідає за авторизацію користувачів.

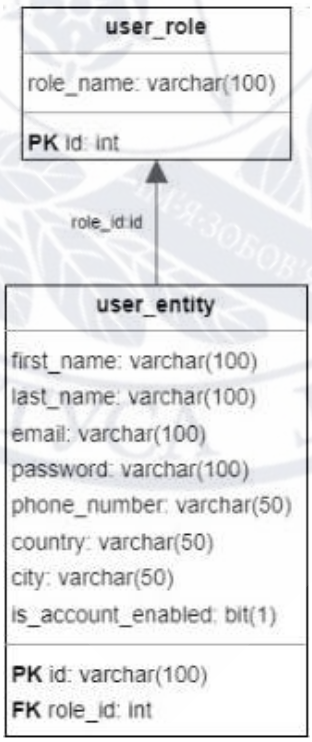

Рисунок 3.7 – Концептуальна схема даних сервісу авторизації та автентифікації користувачів

Дана схема відображає логічну структуру зберігання даних, необхідних для забезпечення процесів реєстрації, автентифікації та управління обліковими записами. Вона містить основні сутності, що відповідають за збереження персональних даних користувачів, їхніх облікових записів та прав доступу.

Для даного сервісу автентифікації розроблено такі сутності:

● user_entity – таблиця, що містить інформацію про всіх користувачів системи. Конфіденційні дані, зокрема паролі, зберігаються у зашифрованому вигляді з метою безпеки доступності даних;

● user_role – таблиця, призначена для зберігання відомостей про ролі користувачів, визначені в системі, що дозволяє реалізувати механізм розмежування прав доступу.

У межах сервісу розташування було спроєктовано відповідну схему бази даних (БД), що відображає структуру зберігання та взаємозв’язки даних, необхідних для коректного функціонування цього модуля. На рисунку 3.8 представлено модель БД сервісу, яка забезпечує ефективне управління інформацією про місцезнаходження посилок, їхній поточний статус, а також взаємодію з іншими компонентами вебпорталу.

Для даного сервісу розташування розроблено такі сутності:

● location_entity – таблиця, що містить інформацію про всі локації, зареєстровані в системі;

● city_entity – таблиця, призначена для зберігання даних про населені пункти, які використовуються в системі;

● country_entity – таблиця, у якій відображено перелік країн, що беруть участь у логістичних операціях;

● placement_entity – допоміжна таблиця, створена для реалізації зв’язку між сутностями міст і країн.

● parcel_destination – таблиця, де буде зберігатись інформація про початкову та кінцеву точки відправки посилики.

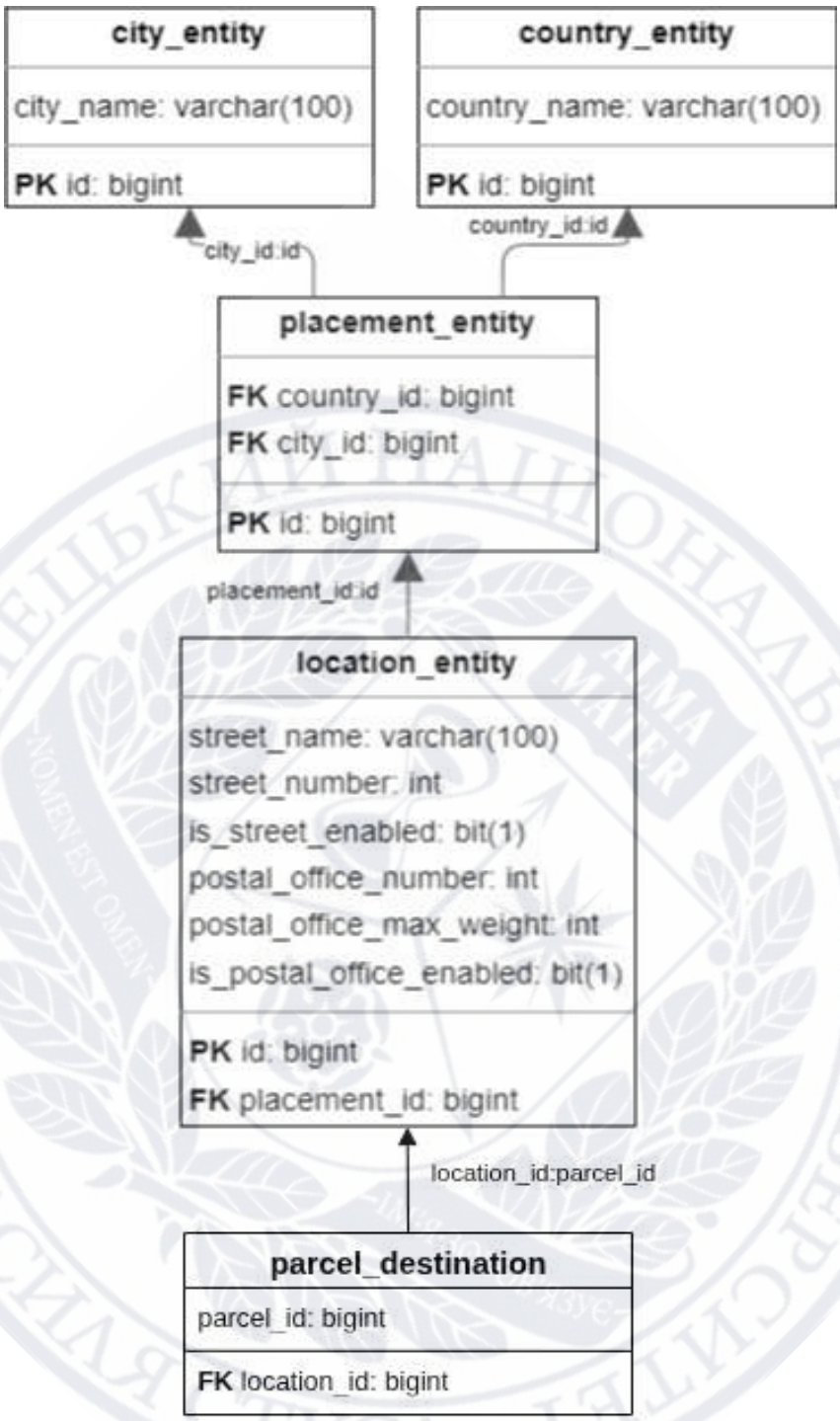

Рисунок 3.8 – Концептуальна схема даних сервісу розташування

У межах сервісу дій з посилками передбачено реалізацію відповідної схеми бази даних (БД), структуру якої представлено на рисунку 3.9. Дана схема відображає логічну організацію даних, що забезпечує зберігання, обробку та відстеження інформації, пов’язаної з життєвим циклом посилки.

Рисунок 3.9 – Концептуальна схема даних сервісу дій з посилками

Для даного дій з посилками розроблено такі сутності:

● parcel_entity – таблиця, що містить інформацію про всі посилки, зареєстровані в системі; зберігає основні атрибути, пов’язані з ідентифікацією, статусом та характеристиками посилки.

● additional_service – таблиця, призначена для фіксації додаткових послуг, що можуть бути застосовані до конкретної посилки (наприклад, страхування, більша кількість днів зберігання чи спеціальне пакування).

3.3. Розробка програмного продукту

Розроблювана система базується на основі клієнт-серверної архітектури. Веб-застосунок побудовано з використанням архітектурного патерна MVC (Model-View-Controller) [39], який є класичним підходом до організації програмного забезпечення мікросервісної архітектури. Даний патерн

проектування передбачає розділення внутрішніх даних програми, користувацького інтерфейсу та керуючої логіки на три окремі компоненти: модель, вигляд та контролер. Така організація дозволяє забезпечити незалежну модифікацію кожного з компонентів, що підвищує гнучкість і підтримуваність програмного рішення.

Фреймворк Spring MVC реалізує принципи патерна Model-View-Controller, надаючи готові слабо пов’язані компоненти для побудови веб-застосунку. Основною перевагою використання архітектурного підходу MVC є можливість змінювати окремі частини програми без істотного впливу на інші, що забезпечує легкість масштабування та модернізації.

Компоненти патерна MVC у контексті розроблюваної системи визначено таким чином:

- Model – зберігає дані програми, які зазвичай представлені у вигляді POJO (Plain Old Java Objects), та забезпечує їхню обробку;
- View – відповідає за клієнтську частину даних моделі, формуючи HTML-сторінки або інші елементи інтерфейсу, що відображаються користувачу;
- Controller – обробляє запити користувача, керує потоками даних між моделлю та виглядом, створюючи відповідні об’єкти моделі та передаючи їх для відображення.

На рисунку 3.10 представлено схему архітектури MVC, яка застосовується у веб-застосунку, що підкреслює логіку взаємодії між компонентами та їхню роль у забезпеченні функціонування системи.

Одним із ключових інструментів під час розроблення програмного забезпечення є UML (Unified Modeling Language) [3] – стандартна мова моделювання, яка забезпечує представлення архітектури системи. За допомогою UML можна відобразити структурні аспекти програмного продукту, включно з компонентами, процесами та їх взаємодією. Використання UML-діаграм полегшує комунікацію між розробниками та дає змогу ефективно

керувати складністю програмної системи.

Сервіс авторизації відіграє ключову роль у побудові всієї системи, оскільки саме він забезпечує механізм ідентифікації користувачів, визначення їхніх ролей і контроль доступу до відповідних функціональних можливостей. Коректна реалізація цього сервісу є критично важливою для гарантування безпеки, цілісності та стабільності роботи системи, адже від неї залежить розмежування прав користувачів, захист персональних даних і запобігання несанкціонованому доступу.

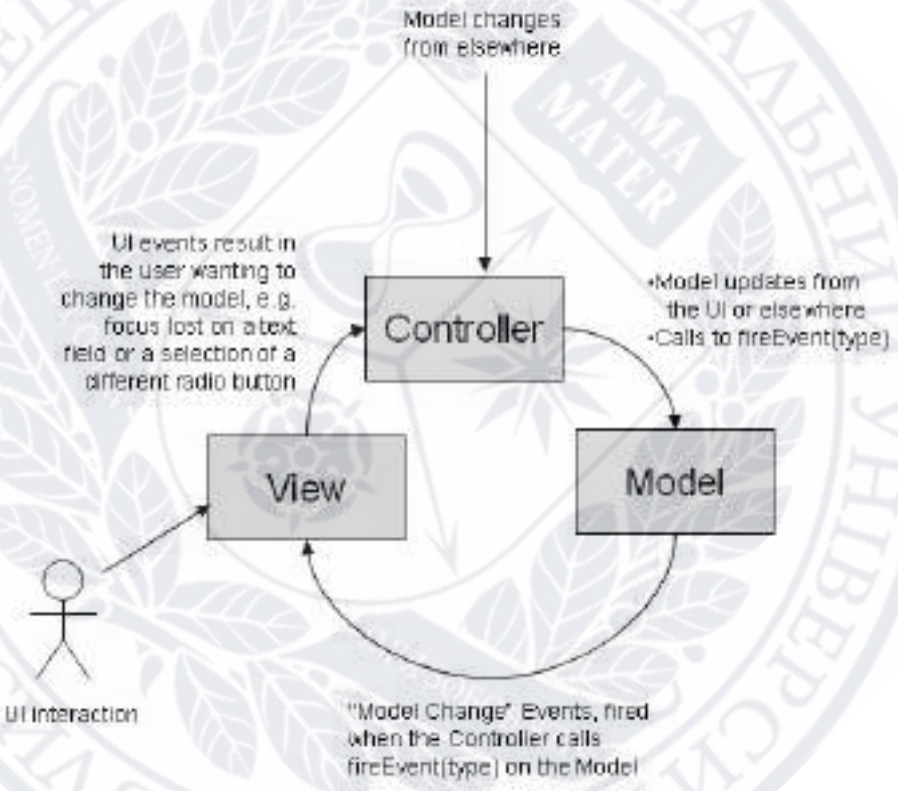

Рисунок 3.10 – Схема компонентів, організованих за принципами патерна Model-View-Controller

На рисунку 3.11 представлено UML-діаграму сервісу авторизації, яка відображає основні сутності, їхні взаємозв’язки та принципи взаємодії компонентів. Дана діаграма ілюструє логіку побудови сервісу, зокрема обробку

даних користувача, перевірку облікових записів та розподіл доступу відповідно до ролей.

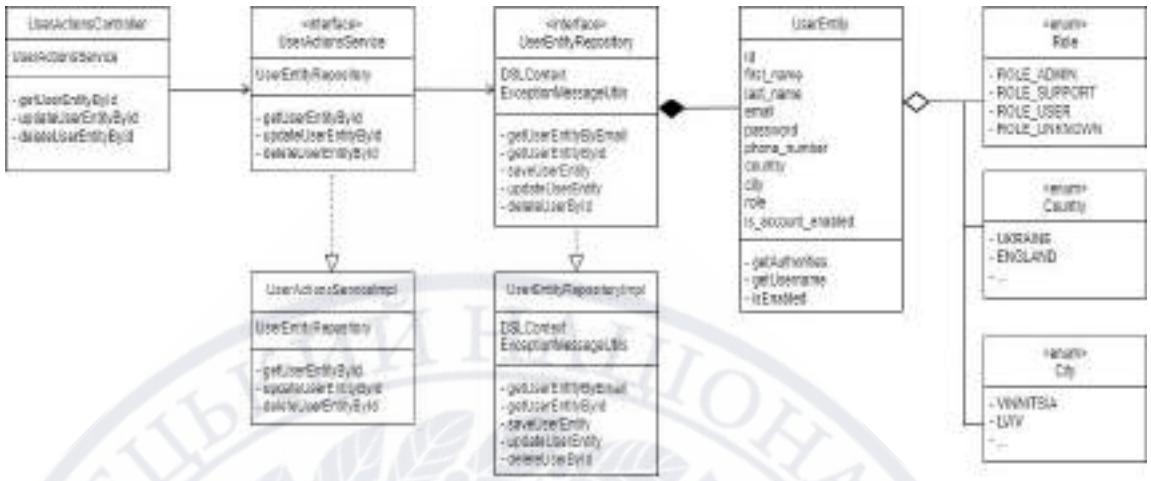

Рисунок 3.11 – Архітектурна UML-діаграма модуля авторизації користувачів

На рисунку 3.11 представлено структуру, згідно з якою всі вхідні запити до системи обробляються через компонент ActionsController, що здійснює взаємодію з ActionsService. Останній відповідає за реалізацію бізнес-логіки, пов’язаної з управлінням користувачами, та використовує UserRepository для виконання операцій із базою даних. Ключовим елементом у цій структурі є сутність User, яка зберігає основну інформацію про користувача системи. Крім того, у моделі визначено перерахування Role, City та Country, що містять фіксовані набори констант, необхідних для забезпечення коректного функціонування сервісу.

Схема організації сервісу розташування відповідає за надання повної та актуальної інформації щодо місцезнаходження логістичних відділень та статусу відправлених посилок у системі. Вона забезпечує узгоджену взаємодію між компонентами сервісу, включаючи обробку запитів користувачів, отримання даних із бази та підготовку результатів для відображення в інтерфейсі. UML-діаграма сервісу розташування, представлена на рисунку 3.12, демонструє ключові елементи архітектури, їхні ролі та взаємозв’язки, що дозволяє чітко уявити логіку роботи даного модуля.

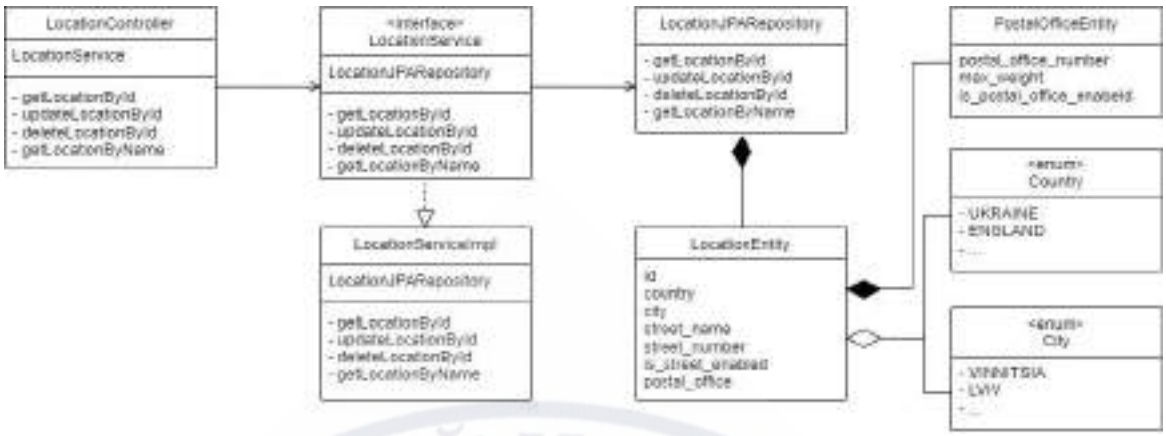

Рисунок 3.12 – Архітектурна UML-діаграма модуля розташування

На рисунку 3.12 представлено схему, що відображає процеси створення, отримання, модифікації та видалення локацій у системі. Компонент LocationController відповідальний за прийом запитів від користувачів, тоді як LocationService реалізує основну бізнес-логіку, забезпечуючи виконання всього необхідного функціоналу сервісу. LocationRepository встановлює зв’язок із базою даних за допомогою ORM технології Hibernate. Ключовою сутністю є Location, яка містить усю необхідну інформацію про локацію. Перерахування City та Country включають визначений набір констант, а сутність PostalOffice надає дані про поштові відділення та інші важливі деталі посилки.

Керування операціями з посилками в межах сервісу забезпечує виконання повного спектру функцій, включно зі створенням, відстеженням, модифікацією та скасуванням надсилання чи отримання посилок для компонентів смарт-систем. На рисунках 3.13 наведено UML-діаграму сервісу дій з посилками. На рисунку 3.13 представлено даіграму, що ілюструє процеси створення, отримання, скасування, зміни та видалення посилки. ParcelController обробляє запити, що надходять до сервісу дій з посилками. ParcelService реалізує основну бізнес-логіку та забезпечує виконання ключових функцій, пов’язаних із керуванням посилками. ParcelRepository встановлює взаємодію з базою даних і є відповідальним за правильність перенесення даних до сутності. ParcelEntity виступає центральною сутністю, що містить усю необхідну інформацію про посилку. Додатково, перерахування AdditionalService, ParcelType та Status

визначають відповідно набір констант для додаткових послуг, статусу посилки та її типу, забезпечуючи стандартизоване відображення цих характеристик у системі.

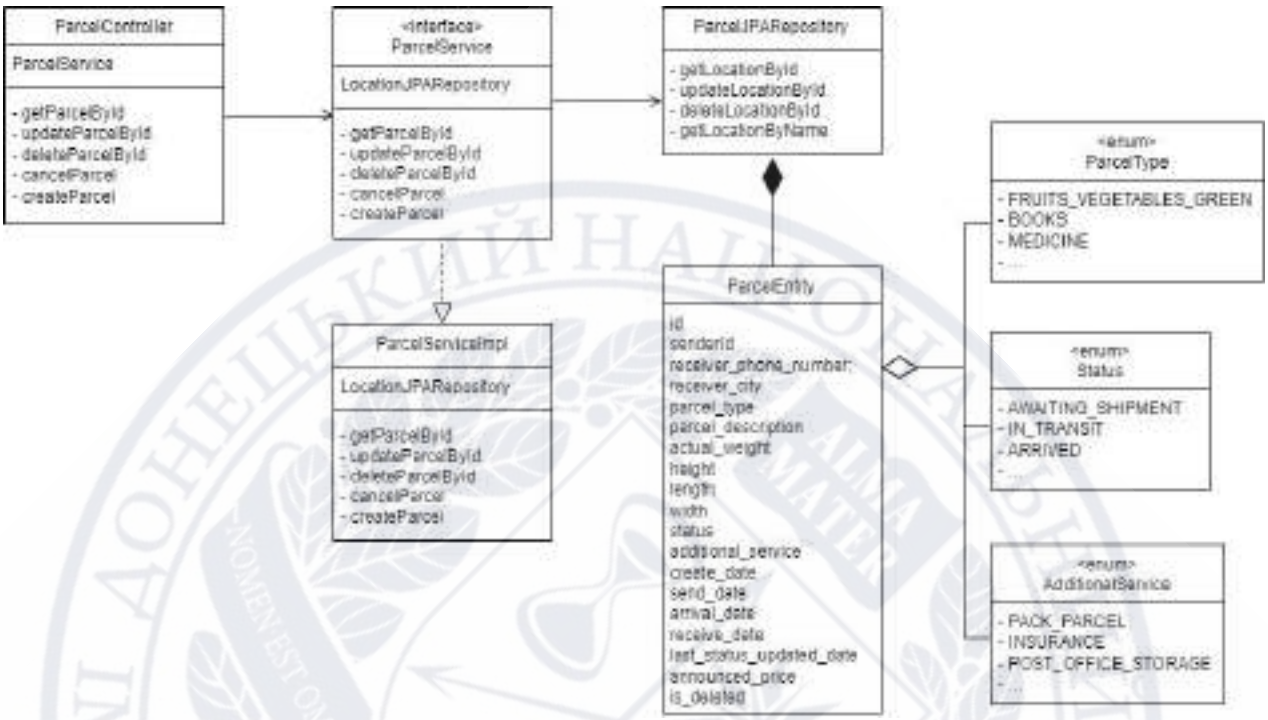

Рисунок 3.13 – Архітектурна UML-діаграма модуля дій з посилками

На рисунку 3.14 наведено UML-діаграму процесів відповідного сервісу з посилками, що ілюструює організацію та послідовність виконання необхідних для доставки операцій.

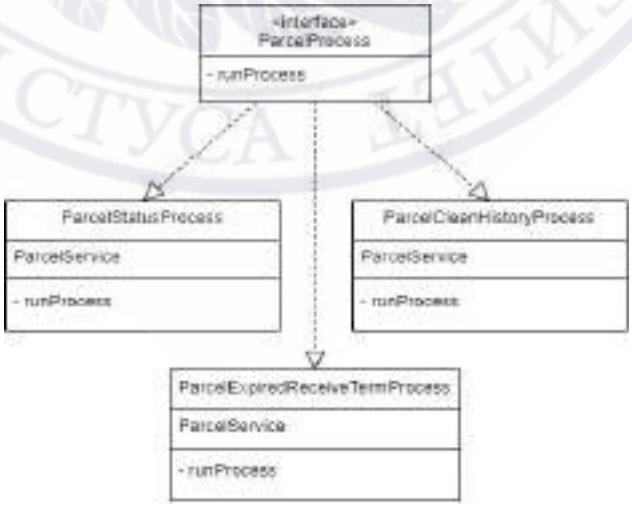

Рисунок 3.14 – Архітектурна UML-діаграма процесів модуля дій з посилками

На рисунку 3.14 представлено діаграму процесів, що виконують відповідні функціональні операції у визначений час. Status запускається наприкінці робочого дня для позначення статусів усіх посилок, які готові до відправки, із можливістю сортування за містом призначення. CleanHistory очищує історичні дані за визначений період, що дозволяє оптимізувати використання простору бази даних та підтримувати її ефективність. ExpiredReceiveTerm позначає усі посилки, які доставили, але не забрали, як ті, які мають повернутись до відправника.

3.4. Складові взаємодії користувача з програмним забезпеченням

Взаємодія користувача з додатком розпочинається з реєстрації акаунту у системі. Даний процес наведено на рисунку 3.15. Заповнення відповідних форм є обов’язковим для ідентифікації особи, яка має намір скористатися функціоналом системи, а також для забезпечення безпеки та персоналізації взаємодії. Після натискання кнопки «Зареєструватися» обліковий запис буде створений у разі коректного заповнення всіх обов’язкових полів форми. У разі виявлення помилок у введених даних система відображає відповідні повідомлення, що дозволяє користувачу внести необхідні виправлення.

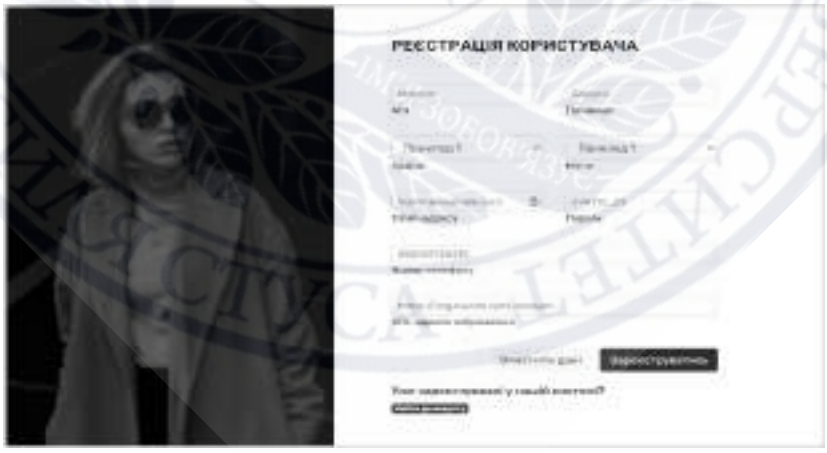

Рисунок 3.15 – Сторінка вебпорталу з реєстрацією акаунту

Авторизація, у свою чергу, передбачає підтвердження облікових даних користувача та надання йому доступу до функцій системи, які потребують автентифікації, включаючи оформлення замовлень, перегляд історії операцій та

налаштування персональних параметрів. Даний процес наведено на рисунку 3.16.

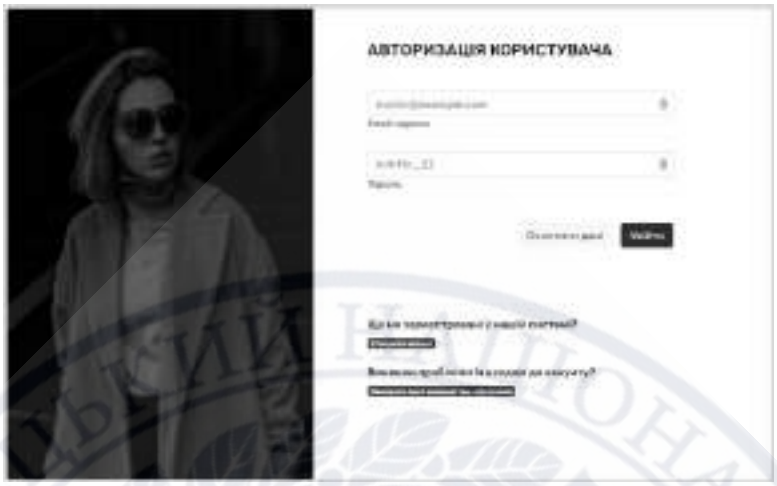

Рисунок 3.16 – Сторінка вебпорталу з авторизацією акаунту

Після того, як був завершений процесу ідентифікації користувача, його буде направлено до інформації акануту, що ілюструється на рисунку 3.17. На цій сторінці користувач має можливість переглядати свої персональні дані, редагувати інформацію профілю, ознайомлюватися з історією відправлень та переглядати перелік створених товарів. Такий функціонал забезпечує централізований доступ до особистої інформації та дозволяє користувачу ефективно керувати своїми даними та взаємодією із системою. Під час редагування даних користувача відбувається перенаправлення до відповідної форми, що містить поля для внесення змін окремих атрибутів, які складають сутність користувача. Кожне поле відповідає певному параметру облікового запису, такому як ім’я, контактні дані або адреса доставки. Система забезпечує перевірку коректності введених даних та відображає повідомлення про помилки у разі некоректного заповнення полів, що дозволяє уникнути помилок та забезпечує достовірність збереженої інформації. Дана функціональність ілюструється на рисунку 3.18, демонструючи користувачу інтуїтивно зрозумілий інтерфейс для внесення змін та контроль за актуальністю особистих даних.

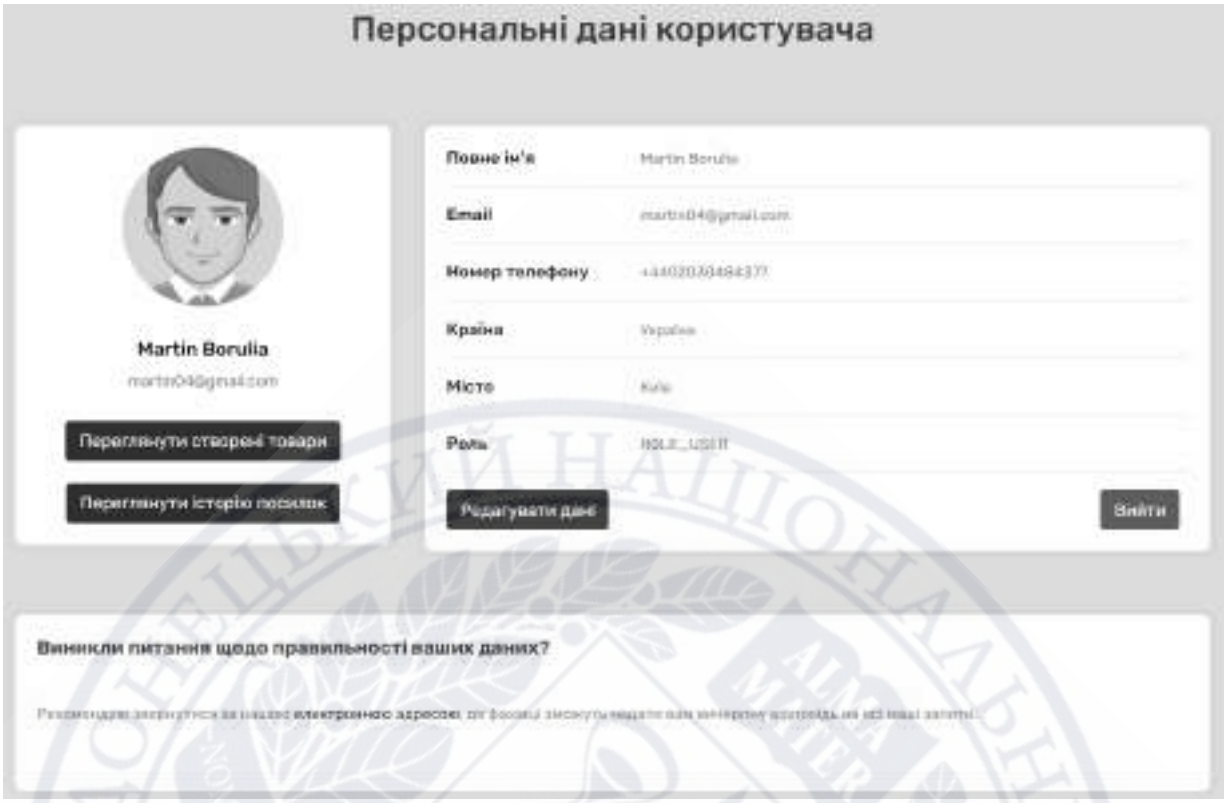

Рисунок 3.17 – Сторінка вебпорталу з інформацією про акаунту

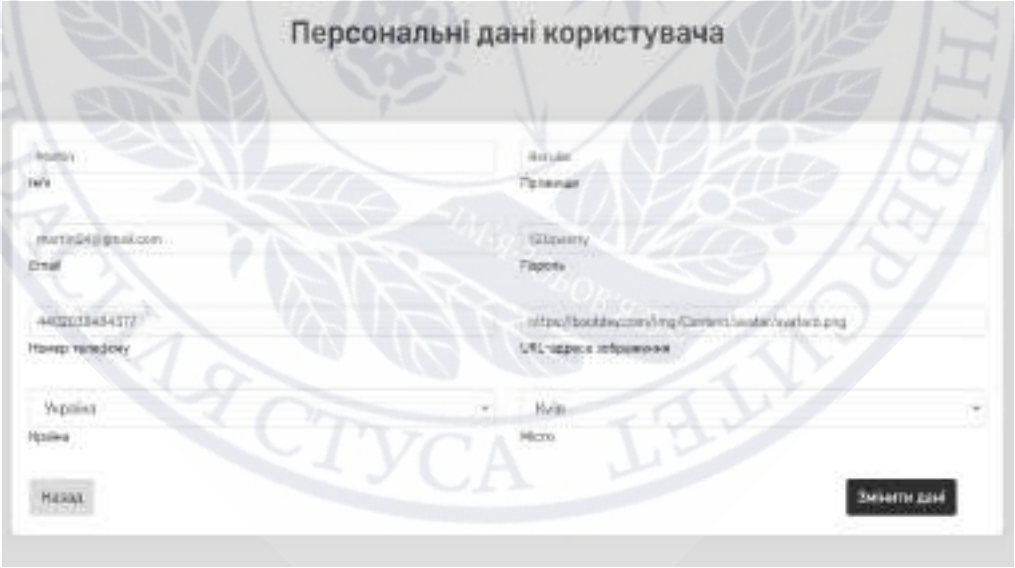

Рисунок 3.18 – Сторінка вебпорталу з редагуванням інформації про акаунту

Знаходячись на сторінці вебпорталу, що показує відповідну кількість посилок, створених користувачем, відображатиметься детальна інформація щодо всіх відправлень. Зокрема, можна переглянути дату створення та

відправлення посилки, маршрут її доставки, включаючи початкове та кінцеве місто, а також інші релевантні дані, що дозволяють відстежувати статус кожного відправлення. Така організація інформації забезпечує зручний і прозорий доступ до історії відправлень, має структуроване та інтуїтивно зрозуміле відображення інформації та дозволяє користувачу ефективно контролювати свої логістичні операції. Даний функціонал ілюструється на рисунку 3.19

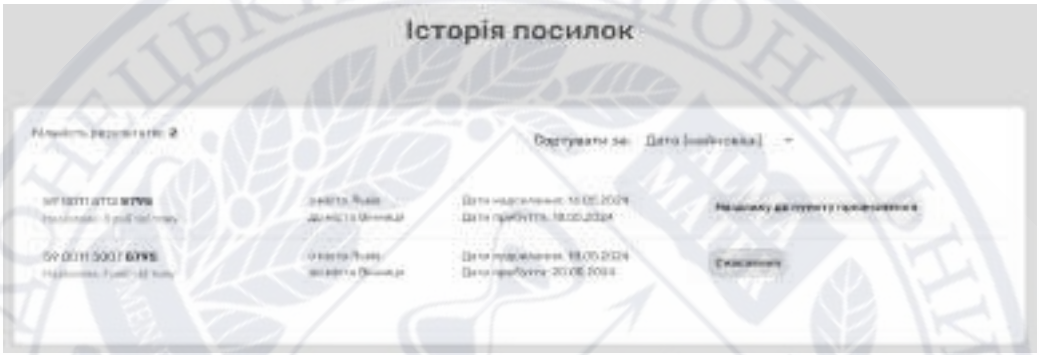

Рисунок 3.19 – Сторінка вебпорталу з історією посилок

Після переходу на сторінку, що показує перелік товарів, які створив користувач, надається можливість переглянути їх повний список разом із детальними характеристиками. Такий підхід дозволяє користувачу ефективно контролювати власний асортимент, відслідковувати зміни в статусі товарів і при необхідності редагувати їхні параметри. Дана функціональність проілюстрована на рис. 3.20.

Рисунок 3.20 – Сторінка вебпорталу з створеними користувачем товарами

При переході на сторінку конкретного товару, створеного користувачем, надається можливість редагувати його характеристики або видалити товар із системи. Подібна функціональність дозволяє користувачу підтримувати актуальність інформації про свої товари та забезпечує контроль за їх представленістю в системі. Даний процес проілюстровано на рис. 3.21.

Рисунок 3.21 – Сторінка вебпорталу із інформацією про товар

Обираючи таку опцію, як редагування товару, користувачеві відкривається форма, що дозволяє змінювати такі характеристики продукту, як назва, наявність у магазині, категорія, зображення, ціна, опис товару, кількість, вага та потужність. Ця форма забезпечує централізоване та структуроване редагування всіх ключових атрибутів, що описують товар, що підвищує точність і узгодженість даних у системі (рис. 3.22).

Після усіх необхідних дій із персональними даними користувачеві відкривається можливість пошуку потрібних товарів. Ілюстрація даної клієнтської частини зображено на рисунку 4.23. Користувач має можливість ознайомитися з повним асортиментом товарів, доступних для замовлення, а також отримати детальну інформацію про кожен товар, включаючи його опис, ціну та інформацію про його наявність.

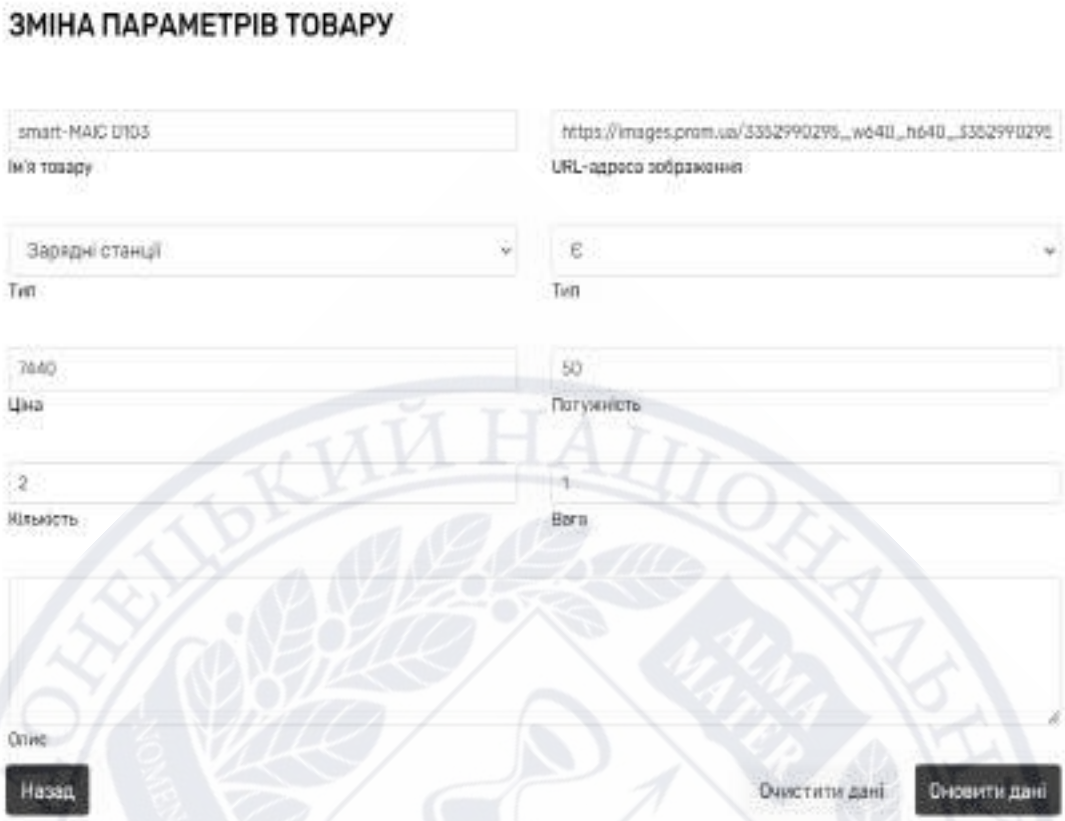

Рисунок 3.22 – Сторінка вебпорталу із редагуванням інформацією про товар

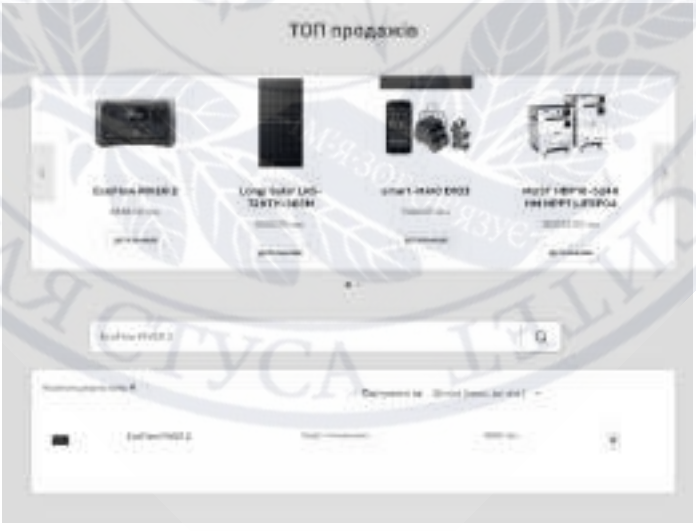

Рисунок 3.23 – Сторінка вебпорталу із пошуком комплектуючих

Після додавання відповідних позицій до кошика користувач отримує можливість сформувати замовлення, визначивши необхідну кількість одиниць

кожного продукту для придбання. Користувач також може видаляти товари з кошика, у результаті чого система відображає повідомлення про успішне видалення відповідного товару зі сторінки формування замовлення. Така організація функціоналу забезпечує гнучке управління складом замовлення, дозволяє коригувати його перед остаточним підтвердженням та сприяє підвищенню зручності користування системою. Дану частину фіункціоналу проілюстровано на рисунку 3.24.

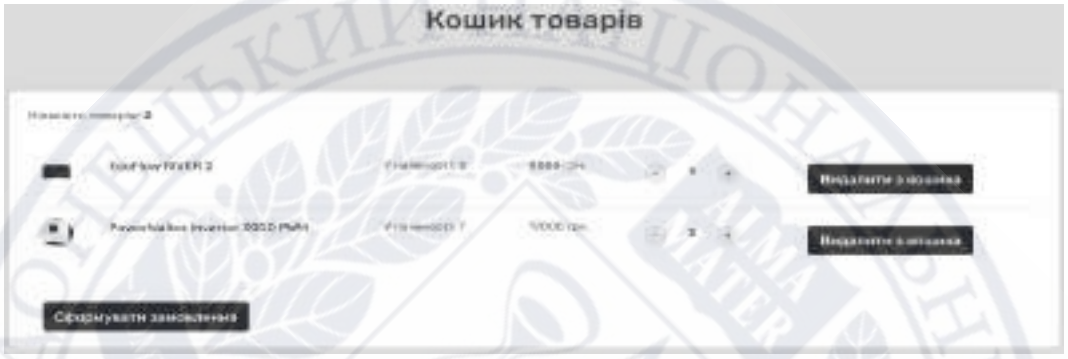

Рисунок 3.24 – Сторінка вебпорталу із кошиком товарів

Після формування замовлення користувач переходить на вебсторінку створення посилки, де місце доставки автоматично прив’язується до номера телефону одержувача, що забезпечує точність і швидкість визначення локації. У формі відображаються товари, які користувач має намір придбати та включити до посилки, з детальною інформацією про кожен з них. Форма створення посилки ілюстрована на рис. 3.25.

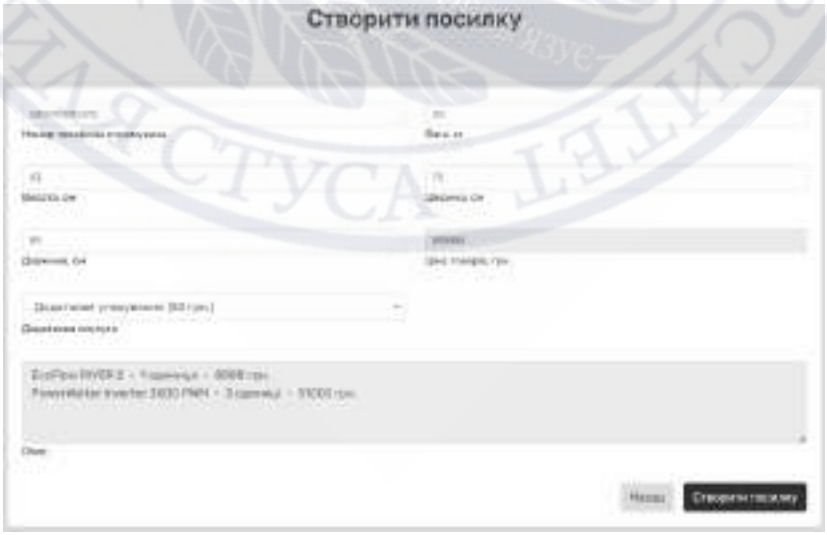

Рисунок 3.25 – Сторінка вебпорталу зі створенням посилки

Після остаточного створення посилки користувач має змогу перейти на сторінку «Пошук посилки» та може знайти відправлення за його унікальним ідентифікаційним номером. Такий підхід дозволяє оперативно відстежувати стан посилки, перевіряти її місцезнаходження та контролювати етапи доставки. Дану частину фіункціоналу проілюстровано на рисунку 3.26.

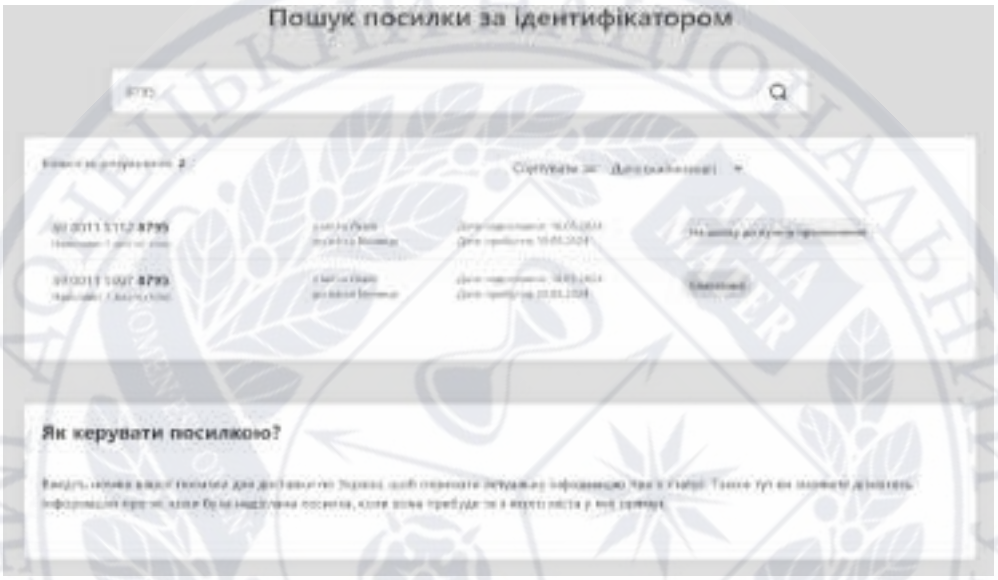

Рисунок 3.226 – Сторінка вебпорталу зі можливістю пошуку посилки

У сучасних умовах цифровізації питання ефективного розгортання веб-застосунків має вирішальне значення, адже невдало зпроєктована архітектура може ускладнити процес впровадження, збільшити тривалість налаштування середовищ та призвести до втрати стабільності системи. Серед провідних платформ для хмарних обчислень, що застосовуються для таких цілей, виділяють Google Cloud Platform [40], Microsoft Azure [41] та Amazon Web Services [42]. Вони надають розробникам можливості масштабування, балансування навантаження, підвищення відмовостійкості та централізованого управління ресурсами.

Розгортання сучасних вебзастосунків зазвичай здійснюється із застосуванням практик безперервної інтеграції та розгортання (CI/CD), які

дозволяють мінімізувати людський фактор, пришвидшити вихід оновлень та гарантувати стабільність релізів. Цей підхід передбачає автоматичну збірку, тестування й публікацію застосунку у вибраному середовищі після кожної зміни у вихідному коді. Контроль над процесом здебільшого здійснюють DevOps-фахівці, які відповідають за інтеграцію сервісів, налаштування середовищ та підтримку стабільності інфраструктури.

В межах даного проєкту для розгортання було обрано Amazon Web Services (AWS) – хмарну платформу, що вирізняється високою продуктивністю, безпекою, масштабованістю та зручністю керування ресурсами. Для створення середовища необхідно авторизуватися в обліковому записі AWS і перейти до керування екземплярами Amazon EC2 [43], після чого встановити Java та MySQL як базові компоненти для роботи застосунку. Далі слід завантажити сервер Apache Tomcat [44] та розгорнути його на EC2, забезпечивши належну інтеграцію з веб-застосунком.

Оптимізацію процесу розгортання можна досягти за допомогою інструменту Jenkins [45] – системи безперервної інтеграції з відкритим вихідним кодом, розробленої на Java. Вона дозволяє описати повний життєвий цикл розгортання, забезпечуючи автоматизацію тестування, побудови артефактів і передачі оновлень на сервер. Завдяки розвиненій системі плагінів Jenkins підтримує інтеграцію з різноманітними платформами та системами керування версіями.

З боку розробника початковим кроком є встановлення інтегрованого середовища IntelliJ IDEA, яке забезпечує повний набір інструментів для створення, тестування та відлагодження програмного забезпечення. Наступним етапом є налаштування системи контролю версій Git [46], що дозволяє зберігати історію змін коду, координувати роботу команди та підтримувати стабільність розробки. Завершальним етапом конфігурації є встановлення платформи Docker [47], що забезпечує контейнеризацію застосунку. Docker дозволяє упакувати

програму разом із її залежностями, забезпечивши відтворюваність середовищ незалежно від операційної системи.

Для роботи з розробленим вебзастосунком управління логістикою магазину комплектуючих до смарт-систем користувачеві необхідно мати активну адресу електронної пошти, номер мобільного телефону та стабільне інтернет-з'єднання. Завдяки кросплатформеній реалізації доступ до функціоналу забезпечується як з персональних комп'ютерів, так і зі смартфонів, що підвищує універсальність і зручність використання системи.

Щоб застосунок працював коректно зі стабільною роботи веб-платформи необхідно дотримуватись визначених системних вимог. Під такими вимогами мають на увазі сукупність характеристик апаратного та програмного середовища пристрою, які забезпечують належне функціонування програмного продукту. Апаратне та програмне забезпечення користувача повинно відповідати мінімальним технічним параметрам, що гарантують працездатність та оптимальну продуктивність веб-застосунку. Нижче наведено мінімальні вимоги різних типів користувачів:

1. Для Windows/Ubuntu користувачів: починаючи із операційної системи, то варто обрати Windows 10,, Linux (Ubuntu 20.04+) для корктної роботи із платформою. У ролі браузера можуть виступати Mozilla Firefox 90+, Google Chrome 95+ та Microsoft Edge 100+. Також, має бути наявне безперебійне з'єднання із мережею Інтернет.

2. Для MacOS користувачів: має стабільну продуктивність через браузер Safari починаючи з версії 3.0, а також із Explorer версії 5.0 та новішими. Для коректного функціонування системи необхідно, щоб пристрій працював під управлінням операційної системи macOS версії 10.4 або вище.

ВИСНОВКИ ДО РОЗДІЛУ 3

У розділі висвітлено питання розробки концептуальну модель системи, яка забезпечує цілісне розуміння архітектури платформи, визначає взаємозв’язки між мікросервісами та користувачами, а також встановлює логіку інформаційних потоків, що підвищує узгодженість, гнучкість і масштабованість системи.

Використання Use-Case-діаграм дозволяє наочно представити сценарії взаємодії користувачів із системою, чітко визначити функціональні можливості та межі застосування кожної ролі, що сприяє зменшенню ризиків під час розробки та підвищує ефективність комунікації між усіма зацікавленими сторонами. Особливу увагу було приділено проєктуванню інформаційного сховища даних. Він є невід’ємним елементом системи, що забезпечує структуроване зберігання, систематизацію та безпечне управління інформацією.

Було детально розглянуто архітектурні, структурні та функціональні особливості вебзастосунку для управління логістичними процесами магазину комплектуючих смарт-систем. Розроблювана система базується на клієнт-серверній архітектурі та реалізує принципи патерна Model-View-Controller (MVC), що забезпечує логічне розділення даних, інтерфейсу та керуючої логіки. Також, було представлено UML-діаграми основних модулів системи, зокрема сервісів авторизації, локації та дій з посилками.

Розглянуто процес розгортання вебзастосунку з використанням сучасних хмарних технологій та сформульовано рекомендації для адміністратора, програміста та користувача щодо встановлення, налаштування та використання системи. Для кожної категорії користувачів визначено необхідні дії, інструменти та послідовність етапів, що сприяють швидкому впровадженню та ефективній роботі із системою.

ВИСНОВКИ

У результаті виконання даної магістерської роботи було реалізовано вебпортал для управління логістикою магазину компонентів смарт-систем. Виконано повний цикл проектування інформаційної системи – від аналізу предметної області та визначення вимог до створення архітектури, розробки програмних модулів і впровадження системи у хмарному середовищі.

1. Досліджено існуючі підходи до управління логістичними механізмами доставки, складування, обліку та розподілу компонентів смарт-систем. Було проведено всебічний аналіз сучасних підходів до управління логістичними процесами та оцінено функціональні можливості наявних рішень на ринку (Logiwa WMS, Deagor WMS, SyncSpider B2B Portal). Це дозволило виокремити дієві методи організації інформатизації логістичних процесів і закласти основу для побудови власного програмного продукту, орієнтованого на удосконалення постачання, обліку та контролю руху товарів.

2. Проведено оцінку методів управління бізнес-процесами у реальних системах і з’ясовано вимоги до інформаційного забезпечення з покроковою інструкцією принципів функціонування вебпорталу.

3. Розроблено загальну архітектуру системи з урахуванням поєднання усіх компонентів та структуру бази даних для відповідних сутностей згідно із характерними особливостями даної предметної області. Архітектурна модель порталу створена за принципами клієнт-серверної взаємодії та застосуванням патерна Model-View-Controller (MVC), що забезпечує чітке розділення даних, логіки бізнес-процесів та інтерфейсу користувача. Серверна частина реалізована на мові Java з фреймворком Spring, для керування залежностями використано Gradle, для роботи з базою даних застосовано ORM-технологію Hibernate та систему контролю версій бази даних Liquibase, а основною СУБД обрано MySQL. На стороні клієнта реалізовано інтерфейс за допомогою HTML, CSS та JavaScript, що забезпечує адаптивність, зручність та функціональність. Для підвищення продуктивності реалізовано кешування даних із використанням

Caffeine Cache. Процес розгортання системи організовано за допомогою AWS та інструментів автоматизації Jenkins і Docker, що значно спрощує адміністрування та підвищує стабільність сервісу.

4. Розроблено інтерфейс, який поєднує інтуїтивне керування, зрозумілу організацію елементів та виконання функцій. Реалізовано функціонал користувачів різних ролей: реєстрація та авторизація, управління товарами, формування та відстеження замовлень. Система забезпечує повний цикл логістичного управління магазином компонентів смарт-систем, дозволяючи автоматизувати ключові бізнес-процеси і зменшити кількість ручних операцій.

5. Вебпортал сприяє автоматизації взаємодії між продавцем і клієнтом, підвищенню прозорості ланцюга постачання та покращенню рівня обслуговування. Перспективи розвитку системи включають створення мобільної версії порталу, розширення функціоналу аналітичного модуля для прогнозування попиту, інтеграцію з системами штучного інтелекту для оптимізації маршрутів доставки, а також підключення додаткових платіжних сервісів і систем управління складом. Такі удосконалення сприятимуть підвищенню рівня автоматизації логістичних процесів і розширенню функціональних можливостей порталу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ТА ЛІТЕРАТУРИ

1. Sinha A., Bernardes E., Calderon R., Wuest T. Digital supply networks : transform your supply chain and gain competitive advantage with disruptive technology and reimagined processes. New York: McGraw-Hill, 2020. 298 p.
2. Greengard S. The Internet of Things. MIT Press. 2015, 73 p.
3. Wang Y., Pettit S. E-logistics : managing digital supply chains for competitive advantage. London: Kogan Page, New York: NY, 2021. 271 p.
4. Freeman E., Robson E. *Head First Design Patterns*. 2nd Edition. O'Reilly, 2020, 634 p.
5. What is a Smart System. URL: https://wattvission.com/smart-system-a-comprehensive-overview-by-watt-vission/.
6. History of smart home from 1963 – 2025: Milestones. URL: https://www.smartest-home.com/en/history-of-smart-home/.
7. What is IoT (Internet of Things)? URL: https://aws.amazon.com/what-is/iot/.
8. Autonomous Systems. URL: https://fulcrumdigital.com/glossary/autonomous-systems/#:~:text=Detailed%20Definition%20%26%20Explanation,operate%20independently%20within%20defined%20parameters.
9. Вебплатформа. URL: https://uk.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B0.
10. Permissions (access control) in web apps. URL: https://wasp.sh/blog/2022/11/29/permissions-in-web-apps.
11. Deagor WMS. URL: https://www.deagor.io/en/.
12. Logiwa WMS. URL: https://www.roimaint.com/en/product/offering-by-product-logia-wms/logia-wms--warehouse-management-system.
13. SyncSpider B2B Portal. URL: https://syncspider.com/.
14. What Is Client-Server Architecture? URL: https://www.coursera.org/articles/client-server-architecture.

15. REST API як спосіб спілкування компонент веб-додатків. URL: https://foxminded.ua/shcho-take-rest-api/.
16. Монолітна архітектура ПЗ. URL: https://qalight.ua/baza-znaniy/shho-take-monolitna-arhitektura/.
17. Про мікросервісну архітектуру. URL: https://qalight.ua/baza-znaniy/shho-take-monolitna-arhitektura/.
18. What is HTML – Definition and Meaning of Hypertext Markup Language. URL: https://www.freecodecamp.org/news/what-is-html-definition-and-meaning/.
19. What is CSS? URL: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Styling_basics/What_is_CSS.
20. What Is JavaScript Used For? URL: https://brainstation.io/learn/javascript/what-is-javascript-used-for.
21. Document Object Model (DOM). URL: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Mode.
22. Cross Site Scripting (XSS). URL: https://owasp.org/www-community/attacks/xss/.
23. What is Java? URL: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-java-programming-language.
24. What exactly is the Java Virtual Machine, and how it does make Java system-neutral? URL: https://www.reddit.com/r/javahelp/comments/8cen3k/what_exactly_is_the_java_virtual_machine_and_how/.
25. JVM Garbage Collectors. URL: https://www.baeldung.com/jvm-garbage-collectors.
26. Gradle Build Tool. URL: https://gradle.org/.
27. Why Choose Spring as Your Java Framework? URL: https://www.baeldung.com/spring-why-to-choose.
28. Inversion of Control (IOC) | Dependency Injection (DI) URL: https://www.apollographql.com/tutorials/caching-subgraph-dgs/03-caching-

with-caffeine.

29. Caching with Caffeine URL: https://www.apollographql.com/tutorials/caching-subgraph-dgs/03-caching-with-caffeine.
30. IntelliJ IDEA Essentials: Build, Test, and Manage Projects Specialization. URL: https://www.coursera.org/specializations/intellij-idea-essentials-build-test-and-manage-java-projects.
31. MySQL programming in Java with JDBC. URL: https://zetcode.com/db/mysqljava/.
32. Getting started Hibernate with Java: A Step-by-Step Guide. URL: https://medium.com/@linkonahad10/getting-started-hibernate-with-java-a-step-by-step-guide-c36c23046f55.
33. How To Use Liquibase with Spring Boot. URL: https://bell-sw.com/blog/how-to-use-liquibase-with-spring-boot/.
34. Conceptual model. URL: https://en.wikipedia.org/wiki/Conceptual_model.
35. JSON Web Tokens. URL: https://auth0.com/docs/secure/tokens/json-web-tokens.
36. Use Case Diagram. URL: https://plantuml.com/use-case-diagram.
37. Алгоритм, як модель алгоритмчного процесу. URL: https://fi.npu.edu.ua/files/Zbirnik_KOSN/6/21.pdf.
38. Entity–relationship model. URL: URL https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model.
39. MVC in Computer Science – The MVC Model. URL: freecodecamp.org/news/what-does-mvc-mean-in-computer-science/.
40. Google Cloud. URL: https://cloud.google.com/.
41. Microsoft Azure. URL: https://azure.microsoft.com/en-us/.
42. AWS. URL: https://aws.amazon.com/.
43. Amazon EC2. URL: https://aws.amazon.com/ec2/.
44. What is Tomcat and What Does It Do? URL: https://medium.com/@mavidev/what-is-tomcat-and-what-does-it-do-

b157ca12ddf1.

45. What is Jenkins? Key Concepts & Tutorial. URL: https://spacelift.io/blog/what-is-jenkins.
46. Git Tutorial. URL: https://www.w3schools.com/git/.
47. A Doker tutorial for beginners. URL: https://docker-curriculum.com/.

ДОДАТКИ

ДОДАТОК А

Class ItemController.java:

```
package ua.donnu.itemdeliveryserivce.controller;

import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.tags.Tag;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PatchMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import ua.donnu.itemdeliveryserivce.model.ItemDeliveryDto;
import ua.donnu.itemdeliveryserivce.model.ItemDeliveryEntity;
import ua.donnu.itemdeliveryserivce.model.ItemType;
import ua.donnu.itemdeliveryserivce.model.ResponseItemEntity;
import ua.donnu.itemdeliveryserivce.service.ItemService;

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

@RestController
@RequestMapping("api/v1/delivery")
@RequiredArgsConstructor
@Tag(name = "Item actions", description = "Action with items")
@Slf4j
public class ItemController {

    private final ItemService itemService;
```

Рисунок А.1 – Ендпоінти для сервісу доставки посилок ч.1

```
@CrossOrigin(origins = "http://localhost:63342")
@GetMapping("/items")
@Operation(summary = "Getting items by name", description = "Returns the items")
public ResponseEntity<ItemDeliveryDto> getItemsByName(@RequestParam("name") String name) {
    log.info("name - {}", name);
    final ItemDeliveryEntity itemDeliveryEntities = itemService.getEntitiesByName(name);
    final ItemDeliveryDto itemDeliveryDtos = castToDto(itemDeliveryEntities);
    return new ResponseEntity<>(itemDeliveryDtos, HttpStatus.OK);
}

@CrossOrigin(origins = "http://localhost:63342")
@PostMapping("/item")
@Operation(summary = "Create item", description = "Return the item")
public ResponseEntity<ItemDeliveryDto> createItem(@RequestBody ItemDeliveryDto itemDeliveryDto) {
    log.info("item to create - {}", itemDeliveryDto);
    final ItemDeliveryEntity itemDeliveryEntity = castFromDto(itemDeliveryDto);
    final ItemDeliveryEntity itemDeliveryCreated = itemService.createItem(itemDeliveryEntity);
    final ItemDeliveryDto itemDeliveryDtoCreated = castToDto(itemDeliveryCreated);
    return new ResponseEntity<>(itemDeliveryDtoCreated, HttpStatus.OK);
}

@CrossOrigin(origins = "http://localhost:63342")
@GetMapping("/item/{itemId}")
@Operation(summary = "Getting items by id", description = "Returns the items")
public ResponseEntity<ItemDeliveryDto> getItemById(@PathVariable(value = "itemId") Long itemId) {
    final ItemDeliveryEntity itemDeliveryEntity = itemService.getEntityById(itemId);
    final ItemDeliveryDto itemDeliveryDto = castToDto(itemDeliveryEntity);
    return new ResponseEntity<>(itemDeliveryDto, HttpStatus.OK);
}

@CrossOrigin(origins = "http://localhost:63342")
@GetMapping("/items/user")
@Operation(summary = "Getting items by user id", description = "Returns the items")
public ResponseEntity<List<ItemDeliveryDto>> getItemsByUserId(@RequestParam(value = "userId") String userId) {
    log.info("Fetching items for user with id - {}", userId);
    final List<ItemDeliveryEntity> itemDeliveryEntities = itemService.getItemsByUserId(userId);
    final List<ItemDeliveryDto> itemDeliveryDtos = itemDeliveryEntities.stream()
            .map(this::castToDto)
            .collect(Collectors.toList());
    return new ResponseEntity<>(itemDeliveryDtos, HttpStatus.OK);
}

@CrossOrigin(origins = "http://localhost:63342")
@DeleteMapping("/item/{itemId}")
@Operation(summary = "Delete item by id", description = "Returns the item")
public ResponseEntity<ResponseItemEntity> deleteItemById(@PathVariable(value = "itemId") Long itemId) {
    final ResponseItemEntity responseItemEntity = itemService.dropEntityById(itemId);
    return new ResponseEntity<>(responseItemEntity, HttpStatus.OK);
}

@CrossOrigin(origins = "http://localhost:63342")
@PatchMapping("/item/{itemId}")
@Operation(summary = "Change item status by id", description = "Returns the item")
public ResponseEntity<ResponseItemEntity> updateItemStatusById(@PathVariable(value = "itemId") Long itemId,
                                                               @RequestParam(value = "itemStatus") Boolean itemStatus) {
    final ResponseItemEntity responseItemEntity = itemService.changeEntityStatusById(itemStatus, itemId);
    return new ResponseEntity<>(responseItemEntity, HttpStatus.OK);
}
```

Рисунок А.2 – Ендпоінти для сервісу доставки посилок ч.2

```
@CrossOrigin(origins = "http://localhost:63342")
@PutMapping("/item")
@Operation(summary = "Update item by id", description = "Returns the item")
public ResponseEntity<ResponseItemEntity> updateItemById(@RequestBody ItemDeliveryDto itemDeliveryDto) {
    final ItemDeliveryEntity itemDeliveryEntity = castFromDto(itemDeliveryDto);
    final ResponseItemEntity responseItemEntity = itemService.updateEntity(itemDeliveryEntity);
    return new ResponseEntity<>(responseItemEntity, HttpStatus.OK);
}

@CrossOrigin(origins = "http://localhost:63342")
@GetMapping("model/items")
@Operation(summary = "Getting items by name", description = "Returns the items")
public List<ItemDeliveryDto> getItems() {
    final List<ItemDeliveryEntity> itemDeliveryEntities = itemService.getAllEntities();
    final List<ItemDeliveryDto> itemDeliveryDtos = itemDeliveryEntities.stream()
            .map(this::castToDto)
            .collect(Collectors.toList());
    return itemDeliveryDtos;
}

private ItemDeliveryDto castToDto(ItemDeliveryEntity itemDeliveryEntity) {
    return ItemDeliveryDto.builder()
            .id(itemDeliveryEntity.getId())
            .name(itemDeliveryEntity.getName())
            .description(itemDeliveryEntity.getDescription())
            .itemType(itemDeliveryEntity.getItemType().getTypeName())
            .imageUrl(itemDeliveryEntity.getImageUrl())
            .price(itemDeliveryEntity.getPrice())
            .quantity(itemDeliveryEntity.getQuantity())
            .ownerId(itemDeliveryEntity.getOwnerId())
            .isItemAvailable(itemDeliveryEntity.getIsItemAvailable())
            .actualWeight(itemDeliveryEntity.getActualWeight())
            .power(itemDeliveryEntity.getPower())
            .reviewEntities(itemDeliveryEntity.getReviewEntities())
            .build();
}

private ItemDeliveryEntity castFromDto(ItemDeliveryDto itemDeliveryDto) {
    return ItemDeliveryEntity.builder()
            .id(itemDeliveryDto.getId())
            .name(itemDeliveryDto.getName())
            .description(itemDeliveryDto.getDescription())
            .itemType(ItemType.getByTypeName(itemDeliveryDto.getItemType()))
            .imageUrl(itemDeliveryDto.getImageUrl())
            .price(itemDeliveryDto.getPrice())
            .quantity(itemDeliveryDto.getQuantity())
            .ownerId(itemDeliveryDto.getOwnerId())
            .isItemAvailable(itemDeliveryDto.getIsItemAvailable())
            .actualWeight(itemDeliveryDto.getActualWeight())
            .power(itemDeliveryDto.getPower())
            .reviewEntities(itemDeliveryDto.getReviewEntities())
            .build();
}
```

Рисунок А.3 – Ендпоінти для сервісу доставки посилок ч.3

Class ItemDeliveryEntity.java:

```
package ua.donnu.itemdeliveryserivce.model;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.ForeignKey;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.JoinTable;
import jakarta.persistence.ManyToMany;
import jakarta.persistence.SecondaryTable;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.math.BigDecimal;
import java.util.HashSet;
import java.util.Set;

@Data 1 usage
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity(name = "delivery_entity")
public class ItemDeliveryEntity {
    @Id no usages
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "delivery_entity_id")
    private Long id;

    @Column(name = "name", no usages
            nullable = false,
            unique = true)
    private String name;

    @Column(name = "description", no usages
            length = 600,
            nullable = false)
    private String description;

    @Column(name = "type", no usages
            nullable = false)
    private ItemType itemType;

    @Column(name = "image_url", no usages
            nullable = false)
    private String imageUrl;

    @Column(name = "price", no usages
            nullable = false)
    private BigDecimal price;

    @Column(name = "quantity", no usages
            nullable = false)
    private Integer quantity;
```

Рисунок А.4 – Головна сутність сервісу доставки посилок ч.1

```
@Column(name = "power",
        nullable = false)
private Double power;

@Column(name = "owner_id",
        nullable = false)
private String ownerId;

@Column(name = "is_item_available",
        nullable = false)
private Boolean isItemAvailable;

@Column(name = "actual_weight",
        nullable = false)
private Double actualWeight;

@ManyToMany
@JoinTable(
        name = "delivery_reviews",
        joinColumns = @JoinColumn(name = "delivery_entity_id"),
        inverseJoinColumns = @JoinColumn(name = "review_id")
)
private Set<ReviewEntity> reviewEntities = new HashSet<>();
```

Рисунок А.5 – Головна сутність сервісу доставки посилок ч.2

Class ItemServiceImpl.java:

```
package ua.dorru.itemdeliveryservice.service;

import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Service;
import ua.dorru.itemdeliveryservice.model.ItemDeliveryEntity;
import ua.dorru.itemdeliveryservice.model.ResponseItemEntity;
import ua.dorru.itemdeliveryservice.model.StatusEntity;
import ua.dorru.itemdeliveryservice.repo.ItemDeliveryRepositoryJpa;

import java.util.List;

@Service
@RequiredArgsConstructor
@Slf4j
public class ItemServiceImpl implements ItemService {

    private final ItemDeliveryRepositoryJpa itemDeliveryRepository;

    @Override
    public ItemDeliveryEntity getEntityByName(String name) {
        log.info("service name - {}", name);
        return itemDeliveryRepository.findByNameContainsIgnoreCase(name);
    }

    @Override
    public ItemDeliveryEntity getEntityById(Long itemId) { return itemDeliveryRepository.findById(itemId).get(); }

    @Override
    public ResponseItemEntity dropEntityById(Long itemId) {
        itemDeliveryRepository.deleteById(itemId);
        return ResponseItemEntity.builder()
                .statusEntity(StatusEntity.SUCCESS)
                .build();
    }
```

Рисунок А.6 – Імплементація сервісу доставки посилок ч.1

```
@Override
public ResponseItemEntity changeEntityStatusById(boolean isActive, Long itemId) {
    itemDeliveryRepository.updateItemStatus(isActive, itemId);
    return ResponseItemEntity.builder()
            .statusEntity(StatusEntity.SUCCESS)
            .build();
}

@Override
public ResponseItemEntity updateEntity(ItemDeliveryEntity itemDeliveryEntity) {
    itemDeliveryRepository.updateItem(
            itemDeliveryEntity.getName(),
            itemDeliveryEntity.getDescription(),
            itemDeliveryEntity.getItemType(),
            itemDeliveryEntity.getImageUrl(),
            itemDeliveryEntity.getPrice(),
            itemDeliveryEntity.getQuantity(),
            itemDeliveryEntity.getActualWeight(),
            itemDeliveryEntity.getPower(),
            itemDeliveryEntity.getId()
            );
    return ResponseItemEntity.builder()
            .statusEntity(StatusEntity.SUCCESS)
            .build();
}

@Override
public List<ItemDeliveryEntity> getAllEntities() { return itemDeliveryRepository.findAll(); }

@Override
public ItemDeliveryEntity createItem(ItemDeliveryEntity itemDeliveryEntity) {
    return itemDeliveryRepository.save(itemDeliveryEntity);
}

@Override
public List<ItemDeliveryEntity> getItemsByUserId(String userId) {
    return itemDeliveryRepository.findItemsByUserId(userId);
}
```

Рисунок А.7 – Імплементація сервісу доставки посилок ч.2

Class ItemDeliveryServiceApplication.java:

```
package ua.donnu.itemdeliveryserivce;

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

@SpringBootApplication
public class ItemDeliverySerivceApplication {

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

}
```

Рисунок А.8 – Клас для запуску сервісу доставки посилок

Class ItemDeliveryDto.java:

```
package ua.donnu.itemdeliveryserivce.model;

import lombok.Builder;
import lombok.Value;
import lombok.extern.jackson.Jacksonized;

import java.math.BigDecimal;
import java.util.Set;

@Value
@Builder
@Jacksonized
public class ItemDeliveryDto {
    Long id;
    String name;
    String description;
    String itemType;
    String imageUrl;
    BigDecimal price;
    Integer quantity;
    String ownerId;
    Boolean isItemAvailable;
    Double actualWeight;
    Double power;
    Set<ReviewEntity> reviewEntities;
}
```

Рисунок А.9 – Клас для транспортування даних у сервісі доставки посилок

Class ItemType.java:

```
package ua.donnu.itemdeliveryserivce.model;

public enum ItemType {
    CHARGING_STATIONS("Зарядні станції"),
    SOLAR_PANELS("Сонячні панелі"),
    WIND_TURBINES("Вітрові турбіни"),
    ENERGY_STORAGE("Системи зберігання енергії"),
    SMART_METERS("Смарт-лічильники"),
    INVERTERS("Інвертори"),
    SMART_LIGHTING("Смарт-освітлення"),
    HEATING_SYSTEMS("Системи опалення"),
    COOLING_SYSTEMS("Системи охолодження"),
    SECURITY_CAMERAS("Камери безпеки"),
    MOTION_SENSORS("Датчики руху"),
    TEMPERATURE_SENSORS("Датчики температури"),
    HUMIDITY_SENSORS("Датчики вологості"),
    AIR_QUALITY_MONITORS("Монітори якості повітря"),
    SMART_LOCKS("Смарт-замки"),
```

Рисунок А.10 – Найменування усіх типів у сервісі доставки посилок ч.1

```
    SMART_THERMOSTATS("Смарт-термостати"),
    VOICE_ASSISTANTS("Голосові помічники"),
    GATEWAY_DEVICES("Пристрої шлюзів"),
    NETWORK_SWITCHES("Мережеві комутатори"),
    SMART_PLUGS("Смарт-розетки");

    ItemType(String typeName) {
        this.typeName = typeName;
    }

    private final String typeName;

    public String getTypeName() {
        return typeName;
    }

    public static ItemType getByTypeName(String typeName) {
        for (ItemType itemType : values()) {
            if (itemType.getTypeName().equals(typeName)) {
                return itemType;
            }
        }
        throw new IllegalArgumentException("No such type with name: " + typeName);
    }
```

Рисунок А.11 – Найменування усіх типів у сервісі доставки посилок ч.2

File application.properties:

```
spring.application.name=item-delivery-serivce

# Spring Context
server.port = 8383

# DB connection
# spring.datasource.username, spring.datasource.password, spring.datasource.url to VAULT
spring.datasource.driver-class-name = com.mysql.cj.jdbc.Driver
spring.datasource.url = jdbc:mysql://localhost:3200/item_delivery_entity_storage
spring.datasource.username = dmytro
spring.datasource.password = item-delivery-entity-secret

# Liquibase
spring.liquibase.enabled = true
spring.liquibase.change-log = classpath:db/changelog/changelog-master.yaml
```

Рисунок А.12 – Файл для запуску сервісу доставки посилок ч.1

```
# Hibernate
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
spring.jpa.database=mysql
```

Рисунок А.13 – Файл для запуску сервісу доставки посилок ч.2

File docker-compose.yml:

```
version: '3.9'

services:

  mysqldb:
    image: mysql:8
    ports:
      - 3200:3306
    volumes:
      - ~/apps/mysql:/location-service-mysql-storage
    environment:
      - MYSQL_USER=dmytro
      - MYSQL_PASSWORD=item-delivery-entity-secret
      - MYSQL_DATABASE=item_delivery_entity_storage
      - MYSQL_ROOT_PASSWORD=qwerty
```

Рисунок А.14 – Файл для запуску докер-еонтейнерів для сервісу доставки посилок

Class ItemDeliveryRepositoryJpa.java:

```
package ua.donnu.itemdeliveryserivce.repo;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Modifying;
import org.springframework.data.jpa.repository.Query;
import org.springframework.lang.NonNull;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;
import ua.donnu.itemdeliveryserivce.model.ItemDeliveryEntity;
import ua.donnu.itemdeliveryserivce.model.ItemType;

import java.math.BigDecimal;
import java.util.List;
```

Рисунок А.15 – Репоиторій для сервісу доставки посилок ч.1

```
@Repository  1 usage
public interface ItemDeliveryRepositoryJpa extends JpaRepository<ItemDeliveryEntity, Long> {
    ItemDeliveryEntity findByNameContainsIgnoreCase(@NonNull String name);  no usages

    @Transactional  no usages
    @Modifying
    @Query("""
            update delivery_entity d set d.name = ?1, d.description = ?2, d.itemType = ?3, d.imageUrl = ?4,
            d.price = ?5, d.quantity = ?6, d.actualWeight = ?7, d.power = ?8
            where d.id = ?9""")
    int updateItem(String name, String description, ItemType itemType, String imageUrl, BigDecimal price,
                   Integer quantity, Double actualWeight, Double power, Long id);

    @Transactional  no usages
    @Modifying
    @Query("update delivery_entity d set d.isItemAvailable = ?1 where d.id = ?2")
    void updateItemStatus(@NonNull Boolean isItemAvailable, @NonNull Long itemId);

    @Query("select d from delivery_entity d where d.ownerId = ?1")  no usages
    List<ItemDeliveryEntity> findItemsByUserId(String userId);
}
```

Рисунок А.16 – Репоиторій для сервісу доставки посилок ч.2

Class ItemService.java:

```
package ua.donnu.itemdeliveryserivce.service;

import ua.donnu.itemdeliveryserivce.model.ItemDeliveryEntity;
import ua.donnu.itemdeliveryserivce.model.ResponseItemEntity;

import java.util.List;

public interface ItemService {
    ItemDeliveryEntity crateItem(ItemDeliveryEntity itemDeliveryEntity);
    ItemDeliveryEntity getEntitiesByName(String name);
    List<ItemDeliveryEntity> getAllEntities();
    ItemDeliveryEntity getEntityById(Long itemId);
    List<ItemDeliveryEntity> getItemsByUserId(String userId);
    ResponseItemEntity dropEntityById(Long itemId);
    ResponseItemEntity updateEntity(ItemDeliveryEntity itemDeliveryEntity);
    ResponseItemEntity changeEntityStatusById(boolean isActive, Long itemId);
}
```

Рисунок А.17 – Інтерфейс для сервісу доставки посилок

ДОДАТОК Б

ДЕКЛАРАЦІЯ

про дотримання академічної доброчесності

Я, _____

_____

Повністю вказується ПІБ та статус (посада для працівників, освітня (освітньо-наукова) програма – для здобувачів вищої освіти)

що нижче підписалась/підписався, розуміючи та підтримуючи загальновизнані засади справедливості, доброчесності та законності,

ЗОБОВ'ЯЗУЮСЬ:

дотримуватися принципів та правил академічної доброчесності, що визначені законодавством України, локальними нормативними актами Донецького національного університету імені Василя Стуса, положеннями, правилами, умовами, визначеними іншими суб'єктами, та не допускати їх порушення.

ПІДТВЕРДЖУЮ:

що мені відомі положення статті 42 Закону України «Про освіту»;

що у даній роботі не представляла/представляв чиїсь роботи повністю або частково як свої власні. Там, де я скористалася/скористався працею інших, я зробила/зробив відповідні посилання на джерела інформації;

що дана робота не передавалась іншим особам і подається вперше, не порушує авторських та суміжних прав закріплених статтями 21-25 Закону України «Про авторське право та суміжні права», а дані та інформація не отримувались в недозволений спосіб.

УСВІДОМЛЮЮ:

що ця робота може бути перевірена університетом на плагіат або інші порушення академічної доброчесності, в тому числі з використанням спеціалізованих сервісів;

що у разі порушення академічної доброчесності, до мене можуть бути застосовані процедури, передбачені законодавством України та Кодексом академічної доброчесності та корпоративної етики Донецького національного університету імені Василя Стуса, іншими локальними нормативними актами університету, та я можу бути притягнута/притягнутий до академічної відповідальності.

_____ _____

(дата) (підпис)