Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії і управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти перший (бакалаврський)

Веб-застосунок маркетплейсу для продажу вживаних товарів

(тема)

Виконав:
здобувач 4 року навчання,
групи КІУКІ-21-2
Владислав КРИКЛИВЕЦЬ
(власне ім'я, прізвище)

Спеціальність
123 «Комп'ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма
Комп'ютерна інженерія
(повна назва освітньої програми)

Керівник: ас. каф. Артем ГУК
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ _____ Андрій КОВАЛЕНКО
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти перший (бакалаврський)

Спеціальність 123 «Комп’ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп’ютерна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“____” _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Крикливцю Владиславу Віталійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Веб-застосунок маркетплейсу для продажу вживаних товарів

затверджена наказом по університету від “ 26 ” травня 2025 р. № 424 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 17 червня 2025 р.

3. Вхідні дані до роботи Java
Spring Boot
Spring Security
Hibernate
Maven
Thymeleaf
Bootstrap
jQuery

4. Перелік питань, що потрібно опрацювати у роботі
Обґрунтування вибору сучасного технологічного стеку
Проєктування структури бази даних та побудова ER-діаграми
Розробка REST API та серверної логіки
Створення та налаштування реляційної бази даних MySQL
Розробка адаптивного та інтуїтивного інтерфейсу користувача
Інтеграція платіжної системи та обробка замовлень
Підготовка технічної та пояснювальної документації

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій 17 ілюстрацій

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Аналіз проблеми та огляд існуючих рішень	27.05.2025-30.05.2025	
2	Обґрунтування вибору технологічного стеку та	31.05.2025-02.06.2025	
	інструментальних засобів розробки		
3	Проєктування структури бази даних,	03.06.2025-05.06.2025	
	основних сутностей та розробка алгоритмів		
4	Розробка та відлагодження програмного	06.06.2025-09.06.2025	
5	Оформлення матеріалів кваліфікаційної роботи	10.06.2025-11.06.2025	
6	Подання кваліфікаційної роботи керівникові	12.06.2025-13.06.2025	
	та попередній захист		
7	Подання кваліфікаційної роботи на	14.06.2025-16.06.2025	

Дата видачі завдання " 27 " травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ ас. Артем ГУК
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 66 с., 16 рис., 0 табл., 2 дод., 16 джерел.

SECOND-HAND, ВЕБ-ЗАСТОСУНОК, SPRING BOOT, SPRING SECURITY, JPA/HIBERNATE, THYMELEAF, BOOTSTRAP, MYSQL, REST API.

Метою кваліфікаційної роботи є розробка сучасного веб-застосунку для організації маркетплейсу з продажу вживаних товарів. У ході виконання кваліфікаційної роботи розроблено застосунок, що забезпечує автоматизовану публікацію оголошень, пошук, замовлення, оформлення покупок, а також управління користувацькими профілями. Функціональність реалізовано з використанням Java Spring Boot, що дозволило впровадити модульну архітектуру, REST API для взаємодії з клієнтською частиною, Spring Security для захисту даних і авторизації, JPA/Hibernate для роботи з реляційною базою MySQL. Інтерфейс створено за допомогою Thymeleaf та Bootstrap, що гарантує адаптивність і зручність користування на різних пристроях. У роботі розглянуто проєктування архітектури системи, структуру та взаємозв’язки сутностей бази даних, реалізацію ролей та обмежень доступу, а також інтеграцію платіжної системи. Система підтримує окремі модулі для адміністраторів і звичайних користувачів, історію замовлень і захист персональних даних. Запропонований веб-застосунок є гнучким, масштабованим та відповідає актуальним вимогам безпеки й ергономіки, а також може бути базою для подальшого розвитку та впровадження в реальних бізнес-умовах.

ABSTRACT

Bachelor’s thesis: 66 pages, 16 figures, 0 tables, 2 appendices, 16 sources.

SECOND-HAND, WEB APPLICATION, SPRING BOOT, SPRING SECURITY, JPA/HIBERNATE, THYMELEAF, BOOTSTRAP, MYSQL, REST API.

The aim of this bachelor’s thesis is to develop a modern web application for a second-hand marketplace, which enables users to publish listings, search for goods, place orders, and manage their profiles. The application is implemented using Java Spring Boot, providing a modular architecture and a REST API for client-side interaction. Spring Security ensures data protection and user authorization, while JPA/Hibernate is used for working with a relational MySQL database. The user interface is built with Thymeleaf and Bootstrap, guaranteeing adaptability and ease of use across different devices. The thesis covers the design of the system architecture, entity modeling, the structure and relationships of database entities, role-based access control, and integration of a payment system. The platform supports separate modules for administrators and regular users, order history tracking, and secure processing of personal data. The developed web application is flexible, scalable, and meets current standards for security and usability, making it suitable as a basis for further development and deployment in real-world commercial environments.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

АРМ – автоматизоване робоче місце

ІАЦ – інформаційно-аналітичний центр

ПЗ – програмне забезпечення

СУБД – система управління базами даних

API – Application Programming Interface (програмний інтерфейс прикладного програмування)

CRUD – Create, Read, Update, Delete (операції створення, читання, оновлення, видалення даних)

DB – Database (база даних)

HTML – HyperText Markup Language (мова розмітки гіпертексту)

HTTP – HyperText Transfer Protocol (протокол передачі гіпертексту)

MVC – Model-View-Controller (модель представлення та управління даними)

ORM – Object-Relational Mapping (об’єктно-реляційне відображення)

REST – Representational State Transfer (архітектурний стиль для побудови веб-сервісів)

SQL – Structured Query Language (мова структурованих запитів)

UI – User Interface (користувацький інтерфейс)

UX – User Experience (досвід користувача)

WDS – Wireless Distribution System (бездротова розподільча система)

ВСТУП

У сучасних умовах швидкого розвитку електронної комерції вторинний ринок товарів займає дедалі важливіше місце в економіці. Зростання попиту на вживані речі зумовлено економічними, екологічними та соціальними чинниками. Все більше користувачів прагнуть раціонального споживання, що відкриває можливості для платформ, які дозволяють продавати та купувати вживані товари без посередників.

Веб-застосунки для маркетплейсів виступають ключовим інструментом цифрової трансформації такого типу торгівлі. Завдяки сучасним технологіям веб-розробки, зокрема Spring Boot, Spring Security, JPA/Hibernate, Thymeleaf, Maven та MySQL, можна створити ефективні, масштабовані та безпечні онлайн-платформи, які задовольняють потреби як продавців, так і покупців.

Використання структурованого стеку технологій дозволяє автоматизувати основні процеси: реєстрацію користувачів, публікацію товарів, обробку замовлень, інтеграцію з платіжними системами та захист персональних даних. Такі веб-застосунки не лише забезпечують зручний інтерфейс для користувача, а й сприяють розвитку культури відповідального споживання.

Кваліфікаційна робота присвячена розробці маркетплейсу для продажу вживаних товарів, з урахуванням особливостей даної сфери. У межах дослідження буде розглянуто повний цикл створення веб-застосунку – від аналітики предметної області до реалізації та тестування готового продукту.

1 АНАЛІТИЧНЕ ДОСЛІДЖЕННЯ ТЕМИ

1.1 Ретроспектива та сучасні виклики торгівлі вживаними товарами

Торгівля вживаними товарами має багатовікову історію, що бере початок ще з часів натурального обміну. Упродовж століть люди прагнули раціонально використовувати наявні ресурси, повторно застосовуючи речі, які втратили свою цінність для одних, але були корисними для інших. Формати такої торгівлі змінювались: від традиційних ринків, де обмінювались одягом, меблями чи інструментами, до комісійних магазинів і блошиних базарів у XX столітті, які стали символом доступності та споживчої гнучкості.

Зі зростанням масового виробництва та культури споживання в другій половині XX століття інтерес до вживаних речей суттєво зменшився. Новизна, бренд і швидка мода витіснили моделі раціонального користування. Проте з початку XXI століття на тлі загострення екологічних проблем, економічних криз та зростання рівня усвідомленого споживання ситуація змінилася. Вторинний ринок товарів знову почав активно розвиватися.

Цьому сприяє низка факторів. По-перше, екологічна свідомість: повторне використання речей дозволяє зменшити обсяг сміття, уповільнити споживання ресурсів та знизити вуглецевий слід. По-друге, економічна доцільність: купівля вживаних речей дає змогу суттєво зекономити без втрати якості. По-третє, соціальні зміни: молоде покоління дедалі більше орієнтується на практичність, унікальність і індивідуальність, а не лише на "нове".

Цифровізація привела до нового етапу розвитку секонд-хенд торгівлі – появи онлайн-маркетплейсів, які зробили можливим продаж і купівлю вживаних товарів у зручному, автоматизованому форматі. Сайти та мобільні додатки дозволяють продавцям швидко публікувати оголошення, а покупцям

– знаходити потрібні речі з фільтрами, категоріями та пошуком. Це не лише спрощує транзакції, а й розширює потенційну аудиторію до національного або глобального масштабу.

Водночас сучасна торгівля вживаними товарами стикається з низкою викликів. До найважливіших можна віднести.

Низький рівень довіри між користувачами: у зв'язку з відсутністю стандартів контролю якості, гарантій та централізованої перевірки, покупці часто стикаються з ризиками шахрайства або неякісного товару.

Труднощі з верифікацією інформації: неправдиві описи товарів, відсутність фото, приховані дефекти – все це ускладнює чесну торгівлю.

Відсутність спеціалізованого функціоналу: більшість загальних платформ не враховують особливості вторинного ринку, такі як динамічна категоризація, локальні пошуки, гнучка система цін, можливість торгу чи обміну.

Невирішені логістичні аспекти: передача товару часто відбувається в офлайн-форматі, що вимагає координації, фільтрації за містами та інтеграції локальної доставки.

Усе це створює потребу у створенні веб-платформи нового покоління – зручної, прозорої, захищеної та адаптованої саме під специфіку ринку вживаних товарів. Такий застосунок має не лише реалізовувати базовий функціонал маркетплейсу, а й забезпечувати додаткові сервіси: рейтинги користувачів, безпечну оплату, збереження історії покупок і публікацій, підтримку багатомовності та мобільність.

У межах цієї дипломної роботи буде розроблено саме таку систему – second-hand маркетплейс, що враховує зазначені виклики й дозволяє ефективно цифровізувати взаємодію між продавцем і покупцем у вторинному сегменті.

1.2 Проблематика цифровізації вторинного ринку

Цифровізація вторинного ринку товарів відкрила нові горизонти для розвитку комерційної взаємодії між продавцями і покупцями. Проте впровадження цифрових технологій у сферу торгівлі вживаними речами супроводжується низкою суттєвих проблем, які потребують комплексного технічного та організаційного вирішення.

Однією з ключових проблем є відсутність стандартизованих підходів до подання інформації про товари. У традиційній роздрібній торгівлі новими товарами існують чіткі характеристики, сертифікація, уніфіковані фотографії, а також гарантії виробника. На відміну від цього, у вторинному сегменті інформація про товар часто є неповною або суб’єктивною, що ускладнює прийняття рішення для потенційного покупця. Для цифрових маркетплейсів це означає необхідність впровадження спеціальних форм валідації, систем оцінювання, додаткових полів опису та обов’язкових зображень.

Ще одним викликом є низький рівень довіри між користувачами, що особливо відчутно в онлайн-середовищі. Часто бувають випадки шахрайства, маніпуляцій із цінами або підміни опису товару після продажу. У зв’язку з цим важливим завданням розробників є реалізація механізмів аутентифікації користувачів, внутрішніх рейтингів, систем скарг та адміністративної модерації оголошень.

Варто також відзначити складність інтеграції з платіжними системами. На відміну від традиційного інтернет-магазину, де ціни фіксовані й товари нові, у вторинній торгівлі ціна часто узгоджується безпосередньо між сторонами. Це створює технічні труднощі для автоматизації транзакцій: потрібна підтримка оплати після отримання товару, системи резервування коштів, інструменти повернення оплати або затримки переказу до підтвердження покупцем.

Крім того, цифрова трансформація вторинного ринку стикається з проблемою класифікації товарів, адже вживані речі не завжди відповідають

стандартним категоріям. Наприклад, товар може бути відремонтованим, мати унікальні дефекти або бути частиною колекційного зібрання. Це вимагає гнучкої структури бази даних, динамічних фільтрів пошуку та можливості додавання користувацьких міток.

Окремої уваги заслуговує локалізація та логістика. Багато користувачів прагнуть купувати товари в межах свого міста або регіону, що вимагає врахування географічних координат, інтеграції карт, фільтрації оголошень за відстанню, а в ідеалі – підтримки локальних методів доставки (наприклад, самовивіз, передача через кур'єра, зустрічі в публічних місцях).

Нарешті, одним із бар'єрів цифровізації є низький рівень технічної грамотності частини користувачів, які мають труднощі з реєстрацією, публікацією оголошень або взаємодією із сервісом. Це вимагає створення максимально інтуїтивного інтерфейсу, адаптивного дизайну, підказок, інструкцій та простих форм зворотного зв'язку.

Таким чином, хоча цифровізація вторинного ринку відкриває великі можливості для розвитку електронної комерції, вона супроводжується низкою технічних, організаційних та поведінкових проблем. Успішне подолання цих викликів можливе лише за умови поєднання ефективних інструментів веб-розробки з глибоким розумінням специфіки поведінки користувачів у цьому сегменті. Саме на вирішення цих завдань і спрямована дана кваліфікаційна робота.

1.3 Обґрунтування необхідності створення веб-платформи

У сучасному цифровому суспільстві торгівля вживаними товарами дедалі більше переміщується в онлайн-простір. Зростання попиту на такі речі обумовлене економічними, соціальними та екологічними факторами, проте існуючі інструменти для взаємодії продавців і покупців у цьому сегменті залишаються недостатньо ефективними. Значна частина угод і надалі здійснюється через неструктуровані канали – соціальні мережі, форуми,

месенджери – які не забезпечують належного рівня безпеки, зручності й автоматизації процесів.

У зв’язку з цим виникає потреба у створенні спеціалізованої веб-платформи, яка дозволить користувачам здійснювати купівлю та продаж вживаних речей у безпечному, організованому та зручному середовищі. Такий ресурс повинен забезпечувати централізований облік товарів, структурування інформації, зручну систему фільтрації та пошуку, а також захист персональних даних і можливість безпечної оплати.

Існуючі маркетплейси здебільшого орієнтовані на нові товари або мають занадто загальний функціонал, що ускладнює роботу саме з вживаними речами. Наприклад, відсутність гнучких категорій, обмежена можливість прикріплення фото з дефектами, неможливість домовитись про зустріч або використати локальні фільтри. Крім того, на деяких платформах відсутня система репутації користувачів або історії угод, що знижує довіру між учасниками обміну.

Технічна реалізація такої платформи можлива завдяки використанню сучасного стеку веб-технологій: Spring Boot для побудови бекенду, Spring MVC і Spring Security для реалізації контролерів і безпеки, JPA/Hibernate для роботи з базою даних, Thymeleaf для формування динамічних HTML-сторінок, MySQL як надійної СУБД, Bootstrap для адаптивного дизайну, Maven для керування залежностями та DevTools для зручної розробки[5].

Таким чином, створення маркетплейсу вживаних товарів – це не лише відповідь на поточні потреби ринку, а й стратегічне рішення для покращення екосистеми вторинної торгівлі, цифрової трансформації звичних процесів та підвищення якості сервісу для кінцевих користувачів. Реалізація цього проєкту дозволить сформувати ефективну інфраструктуру для безпечної й зручної купівлі-продажу вживаних речей онлайн.

2 ТЕХНОЛОГІЧНЕ ПІДҐРУНТЯ ПРОЄКТУ

2.1 Платформа Spring Boot: функціональність і переваги

Розробка сучасних веб-застосунків вимагає від фреймворків не лише стабільності та розширюваності, а й швидкості запуску, автоматизації конфігурації та можливості масштабування. У цьому контексті Spring Boot став стандартом де-факто для Java-розробки у сфері веб та enterprise-додатків. Створений як спрощення до класичного фреймворку Spring, Spring Boot надає гнучке та швидке рішення для побудови продуктивних застосунків.

Spring Boot дозволяє розробникам створювати повнофункціональні веб-додатки з мінімальною конфігурацією. Його ключові можливості:

Автоматична конфігурація. Spring Boot аналізує структуру класів та залежностей і автоматично налаштовує відповідні компоненти застосунку. Це дозволяє уникнути складних XML-конфігурацій і суттєво прискорює старт проєкту.

Вбудований сервер Tomcat або Jetty. Немає необхідності налаштовувати окремий сервер. Достатньо просто запустити проєкт як Java-додаток (java -jar) – сервер вже інтегрований у збірку.

Інтеграція з базами даних. Spring Boot легко підключається до реляційних СУБД (в, PostgreSQL) через Spring Data JPA або JDBC, підтримуючи автоматичне створення таблиць, міграції та перевірку цілісності даних.

Вбудовані засоби моніторингу. Завдяки Spring Boot Actuator, можна відстежувати стан застосунку (метрики, health-check, environment) без сторонніх інструментів.

Гнучка структура. Spring Boot не нав'язує архітектурний стиль, дозволяючи реалізовувати як монолітні, так і мікросервісні додатки.

У розробці second-hand маркетплейсу Spring Boot виступає основою серверної частини системи. Його застосування дозволило реалізувати такі компоненти.

Створення RESTful API для обробки запитів користувачів (перегляд товарів, додавання оголошень, реєстрація та авторизація).

Інтеграція з базою даних MySQL для збереження інформації про товари, користувачів, кошик і замовлення.

Підключення Spring Security для обмеження доступу до ресурсів згідно з ролями.

Розгортання веб-додатку через вбудований Tomcat без додаткових серверних залежностей.

Зручна обробка шаблонів із використанням Thymeleaf та підтримка багатомовності.

Spring Boot забезпечує такі переваги для команд розробників.

Швидкий запуск проєкту. За лічені хвилини можна створити базовий каркас застосунку через Spring Initializr.

Менше шаблонного коду. Завдяки анотаціям (@RestController, @Service, @Entity) розробник може зосередитися на бізнес-логіці.

Спрощене тестування. Spring Boot Test надає можливість запускати модульні та інтеграційні тести з попередньо налаштованим контекстом.

Масштабованість. Проєкт можна легко адаптувати до зростання навантаження або розділити на мікросервіси.

Сумісність. Велика кількість бібліотек і модулів Spring (Security, Data, Web, AOP) підтримується з коробки.

На рис. 2.1 представлено ключові переваги використання Spring Boot у проєктах.

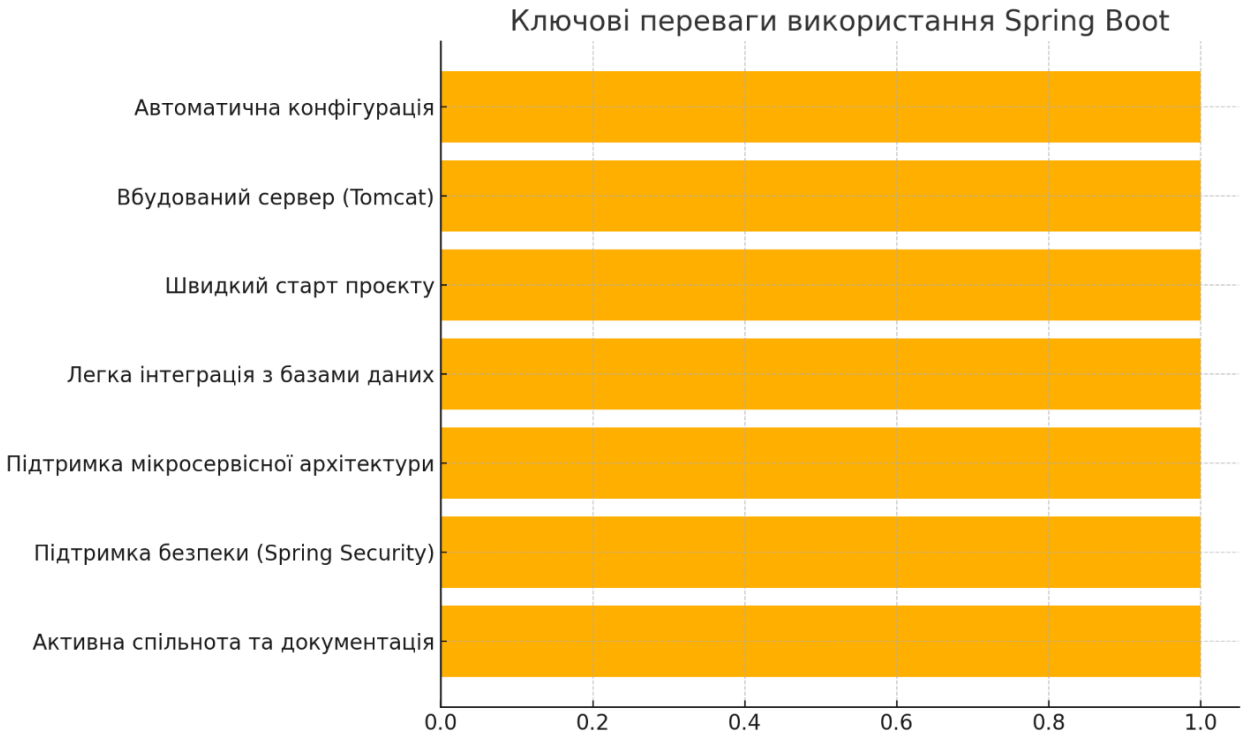

Рисунок 2.1 – Основні переваги Spring Boot

2.2 Архітектурний підхід REST і реалізація через Spring MVC

У сучасній веб-розробці архітектура REST (Representational State Transfer) є одним із найпоширеніших стандартів побудови програмних інтерфейсів для обміну даними між клієнтом і сервером. Її простота, масштабованість і сумісність із HTTP-протоколом роблять REST ідеальним вибором для розробки веб-застосунків, які повинні обробляти численні запити у реальному часі. Під час створення маркетплейсу для продажу вживаних товарів архітектура REST забезпечує зручну взаємодію між фронтендом і бекендом: інтерфейс користувача надсилає запити до серверної частини, яка через контролери Spring MVC обробляє ці запити, викликає відповідні сервіси та повертає структуровані дані, переважно у форматі JSON[1], [15].

Основними принципами REST є ідентифікація кожного ресурсу унікальним URI, наприклад /api/products/5, та використання стандартних HTTP-методів для взаємодії з ресурсами. Зокрема, GET застосовується для

отримання інформації, POST – для створення нового ресурсу, PUT/PATCH – для оновлення, а DELETE – для видалення. Кожен запит є безстановим: він містить всю необхідну інформацію для його обробки, а сервер не зберігає попередній стан користувача між запитами. API має бути уніфікованим, тобто клієнту достатньо знати доступні ендпоінти і правила взаємодії, без необхідності розуміти внутрішню логіку реалізації сервера.

У межах Spring Boot реалізація REST-контролерів здійснюється за допомогою модулю Spring MVC, що забезпечує маршрутизацію HTTP-запитів до відповідних методів контролера. Для цього використовуються анотації: @RestController позначає клас, який обробляє REST-запити та автоматично серіалізує об’єкти Java у JSON; @RequestMapping вказує базовий маршрут; @GetMapping, @PostMapping, @PutMapping, @DeleteMapping конкретизують тип запиту та шлях; а @PathVariable, @RequestBody, @RequestParam дозволяють отримувати дані з URL або тіла запиту.

Типовий REST-контролер для роботи з товарами виглядає так: через анотації задаються маршрути для отримання, створення і видалення товару. Наприклад, метод getProduct дозволяє отримати товар за id, createProduct – додати новий товар, а deleteProduct – видалити існуючий.

Підхід REST надає низку переваг. Він дозволяє легко масштабувати застосунок і підключати до нього різні типи клієнтів, у тому числі веб- і мобільні додатки або зовнішні API. Простота використання стандартного HTTP робить REST-запити інтуїтивно зрозумілими навіть без додаткових інструментів. Безстановість підходить для систем із великою кількістю одночасних користувачів і підвищує масштабованість, а уніфікована структура API спрощує розробку, тестування та документування інтерфейсів.

Таким чином, архітектура REST у поєднанні зі Spring MVC є оптимальним вибором для створення веб-застосунку маркетплейсу. Вона забезпечує зручну маршрутизацію запитів, гнучкий підхід до обробки даних і дозволяє легко розширювати систему новими функціями. Такий підхід

гарантує високий рівень інтегрованості, стабільності та продуктивності платформи, що є особливо важливим у випадку великої кількості користувачів і частих транзакцій.

2.3 Управління даними з JPA та Hibernate

Однією з ключових складових будь-якого веб-застосунку є система зберігання та обробки даних. У випадку маркетплейсу для продажу вживаних товарів необхідно опрацьовувати значні обсяги структурованої інформації, такої як дані про користувачів, товари, категорії, замовлення, кошики, коментарі та інші сутності. Для ефективної роботи з цими об'єктами у проєкті використано стандарт Java Persistence API (JPA) у поєднанні з бібліотекою Hibernate як провайдером ORM (Object-Relational Mapping). JPA, яка є офіційною специфікацією Java EE, визначає механізми для управління збереженням даних у реляційних базах через об'єкти Java та надає абстракцію над SQL, дозволяючи розробникам працювати з таблицями як із класами-сутностями та виконувати CRUD-операції без написання SQL-запитів вручну.

Серед основних переваг JPA – автоматичне створення таблиць на основі Java-класів, використання анотацій для опису зв’язків між сутностями, підтримка транзакційності та каскадності, а також зручна інтеграція зі Spring Data JPA, що значно скорочує обсяг коду. Hibernate як найпопулярніший реалізатор JPA забезпечує додаткові можливості, серед яких гнучке кешування запитів, підтримка як нативного SQL, так і HQL (Hibernate Query Language), автоматичне оновлення схеми бази даних і розширена система валідацій. У межах маркетплейсу Hibernate слугує містком між Java-моделями й базою даних MySQL, забезпечуючи ефективне збереження, пошук, оновлення та видалення записів.

У проєкті реалізовано основні сутності, такі як User, Product, Order, Category, CartItem, Comment, кожна з яких відображає окрему таблицю в базі

даних, а зв’язки між ними описані за допомогою відповідних анотацій JPA. Наприклад, сутність Product містить визначення автоматичної генерації первинного ключа, а також зв’язки багато-до-одного з користувачем (власником) та категорією товару (лістинг 2.1).

Лістинг 2.1 – Опис сутності Product із використанням JPA-анотацій для зв’язків з іншими таблицями

```
@Entity
@Table(name = "products")
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;
    private String description;
    private double price;

    @ManyToOne
    @JoinColumn(name = "user_id")
    private User owner;

    @ManyToOne
    @JoinColumn(name = "category_id")
    private Category category;
}
```

Завдяки такій структурі можна організувати логічні зв’язки між сутностями й підтримувати цілісність даних у базі. Репозиторії, що створені через інтерфейси JpaRepository, дозволяють швидко реалізувати запити до бази даних без ручного написання SQL. Наприклад, для пошуку товарів за категорією можна використати такий репозиторій (лістинг 2.2).

Лістинг 2.2 – Приклад репозиторію ProductRepository для виконання пошуку товарів за назвою категорії

```
public interface ProductRepository extends
JpaRepository<Product, Long> {
    List<Product> findByCategory_Name(String categoryName);

}
```

Використання JPA та Hibernate у поєднанні зі Spring Data JPA значно прискорює розробку, дозволяючи зосередитись на бізнес-логіці, а не на написанні запитів до бази. ORM-підхід забезпечує масштабованість системи, швидку адаптацію структури БД до змін у моделях і підтримку складних пошукових механізмів, які реалізуються або через методи з автоматично генерованими запитами, або за допомогою HQL. Крім того, Spring автоматично обгортає критичні операції у транзакції, що підвищує надійність системи. До переваг також належить можливість розширення функціоналу за рахунок підтримки кастомних репозиторіїв, проекцій, пагінації й сортування, що дозволяє гнучко будувати запити до даних відповідно до потреб користувача та бізнесу.

2.4 Організація безпеки через Spring Security

У процесі розробки веб-застосунку маркетплейсу для вживаних товарів важливим аспектом стало забезпечення високого рівня інформаційної безпеки. Це пов’язано з необхідністю захисту персональних даних користувачів, реалізації автентифікації й авторизації, а також обмеження доступу до певного функціоналу відповідно до ролей. З огляду на це, доцільним і технічно обґрунтованим рішенням стало використання фреймворку Spring Security.

Spring Security є стандартним інструментом у середовищі Java для захисту веб-застосунків. Він надає гнучкий механізм контролю доступу на основі ролей, обробки HTTP-запитів, захисту від атак CSRF та інших поширених загроз. Його інтеграція зі Spring Boot відбувається без додаткових складнощів, що дозволяє швидко й ефективно впровадити політику безпеки у застосунок [11].

У межах дипломного проєкту система безпеки охоплює базову автентифікацію через логін-форму, збереження паролів у зашифрованому вигляді за допомогою алгоритму BCrypt, розмежування доступу до

маршрутів залежно від ролі користувача, а також реалізацію функціоналу виходу із системи. Наприклад, сторінки для публікації оголошень доступні лише авторизованим користувачам з роллю USER, а панель адміністратора – виключно для ролі ADMIN. Це реалізовано за допомогою класу конфігурації безпеки, де визначаються відповідні правила через методи antMatchers() та hasRole().

Перевагою використання Spring Security є його повна сумісність із іншими модулями Spring, а також висока розширюваність. Наприклад, у майбутньому можлива інтеграція двофакторної автентифікації або зовнішніх провайдерів OAuth2. На відміну від альтернатив, таких як Apache Shiro або самописних фільтрів, Spring Security має чітко структуровану документацію, активну спільноту та регулярну підтримку, що є критично важливим для довготривалих проєктів.

Також варто відзначити, що обрана технологія дозволяє уникнути дублювання логіки доступу в контролерах. Завдяки централізованому керуванню політиками доступу зберігається чистота коду й досягається висока гнучкість при зміні бізнес-вимог.

Загалом використання Spring Security у розробці веб-платформи second-hand маркетплейсу є обґрунтованим вибором, який поєднує ефективність, безпеку та простоту інтеграції в межах архітектури проєкту. Це рішення дозволило реалізувати стабільну систему керування правами доступу, забезпечити захист персональних даних і сформувати надійну основу для подальшого розширення функціоналу застосунку.

2.5 Робота з базою даних MySQL

У процесі проєктування серверної частини веб-застосунку ключовим аспектом стала організація надійного, продуктивного та масштабованого сховища даних. Для досягнення цих цілей було проаналізовано кілька сучасних реляційних та нереляційних СУБД, зокрема PostgreSQL, Oracle

Database, MongoDB, а також MySQL. Результати аналізу підтвердили доцільність використання MySQL як основної системи керування базами даних для потреб маркетплейсу вживаних товарів.

MySQL – це зріла, перевірена часом реляційна СУБД, яка поєднує високу продуктивність, відкритий код, зручність розгортання та широку підтримку в екосистемі Java. Вона безпосередньо інтегрується зі Spring Boot за допомогою JDBC-драйверів або через ORM-рішення (JPA/Hibernate), що дозволяє автоматизувати створення та синхронізацію таблиць на основі Java-класів. Наявність вбудованого механізму індексації, ACID-гарантій транзакційності та підтримки складних зв'язків між таблицями робить її особливо зручною для побудови складних структур типу «товар – продавець – категорія – замовлення»[2],[10].

На відміну від PostgreSQL, яка має дещо ширший функціонал (наприклад, повнотекстовий пошук або складні типи даних), MySQL демонструє кращу швидкодію при типових запитах SELECT/INSERT, що є критичним у системах із високою частотою читання та написання. MongoDB, як документна СУБД, хоч і дозволяє гнучкіше працювати з неструктурованими даними, поступається в можливостях реляційного моделювання, яке є основоположним у маркетплейсах.

У межах реалізації дипломного проєкту в MySQL було створено базу даних із кількома таблицями: users, products, categories, orders, order_items, comments, roles. Кожна з таблиць має чітко визначену структуру, зовнішні ключі, індекси та каскадні зв'язки. Це забезпечує узгодженість даних при здійсненні транзакцій, дозволяє реалізувати складені запити для фільтрації, пошуку та сортування, а також мінімізує дублювання даних.

Інтеграція з MySQL у Spring Boot виконується через стандартний драйвер mysql-connector-java, а взаємодія з базою даних реалізована через репозиторії Spring Data JPA. Завдяки цьому розробник може оперувати об'єктами високого рівня (сутностями) замість написання SQL-запитів, що пришвидшує розробку й покращує читаємість коду.

З погляду безпеки, MySQL підтримує контроль доступу за ролями, SSL-шифрування з'єднання та аудит запитів, що є важливим у контексті обробки персональних даних користувачів і транзакційної інформації. До того ж, завдяки широкій підтримці MySQL-хостингів та інструментів адміністрування (phpMyAdmin, DBeaver, MySQL Workbench), розгортання бази в реальному середовищі є інтуїтивно зрозумілим і контрольованим.

На рисунку 2.2 представлена схема сучасної хмарної архітектури розгортання Spring Boot-застосунку та бази даних MySQL у кластері Kubernetes із використанням Docker-контейнерів. Діаграма ілюструє, як взаємодіють між собою сервіси у межах кластера: дані зберігаються у виділених томах (Persistent Volume), а доступ до бази MySQL і серверної частини на Spring Boot здійснюється через окремі сервіси. Секрети та конфігурації розміщуються окремо, забезпечуючи безпеку та централізоване управління налаштуваннями. Вся система масштабована й може інтегруватися з публічними Docker-реєстрами та зовнішніми користувачами через інтерфейс веб-застосунку.

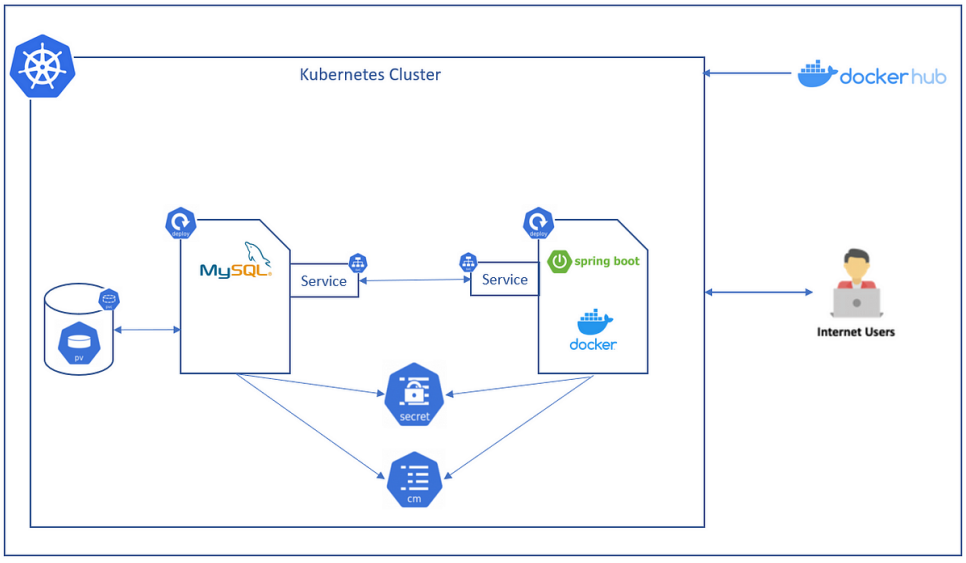

Рисунок 2.2 – Архітектура розгортання Spring Boot і MySQL у Kubernetes-кластері з використанням Docker-контейнерів

З погляду безпеки, MySQL підтримує контроль доступу за ролями,

SSL-шифрування з’єднання та аудит запитів, що є важливим у контексті обробки персональних даних користувачів і транзакційної інформації. До того ж, завдяки широкій підтримці MySQL-хостингів та інструментів адміністрування (phpMyAdmin, DBeaver, MySQL Workbench), розгортання бази в реальному середовищі є інтуїтивно зрозумілим і контрольованим.

На рисунку 2.2 представлена схема сучасної хмарної архітектури розгортання Spring Boot-застосунку та бази даних MySQL у кластері Kubernetes із використанням Docker-контейнерів. Діаграма ілюструє, як взаємодіють між собою сервіси у межах кластера: дані зберігаються у виділених томах (Persistent Volume), а доступ до бази MySQL і серверної частини на Spring Boot здійснюється через окремі сервіси. Секрети та конфігурації розміщуються окремо, забезпечуючи безпеку та централізоване управління налаштуваннями. Вся система масштабована й може інтегруватися з публічними Docker-реєстрами та зовнішніми користувачами через інтерфейс веб-застосунку.

На рисунку 2.3 представлено логічну модель бази даних, розроблену для маркетплейсу вживаних товарів. Вона охоплює такі основні сутності: accounts (користувачі), products (товари), shopping_cart (кошики), cart_item (позиції в кошику), accounts_role (ролі користувачів), users_roles (зв’язки між користувачами та ролями). Зв’язки між таблицями реалізовані через зовнішні ключі, що забезпечує цілісність даних та можливість виконання складених запитів для отримання агрегованої інформації. Зокрема, таблиця cart_item виконує роль проміжної сутності між products та shopping_cart, дозволяючи враховувати кількість кожного товару в конкретному кошику. Таке структурування сприяє масштабованості та гнучкості розробленого рішення.

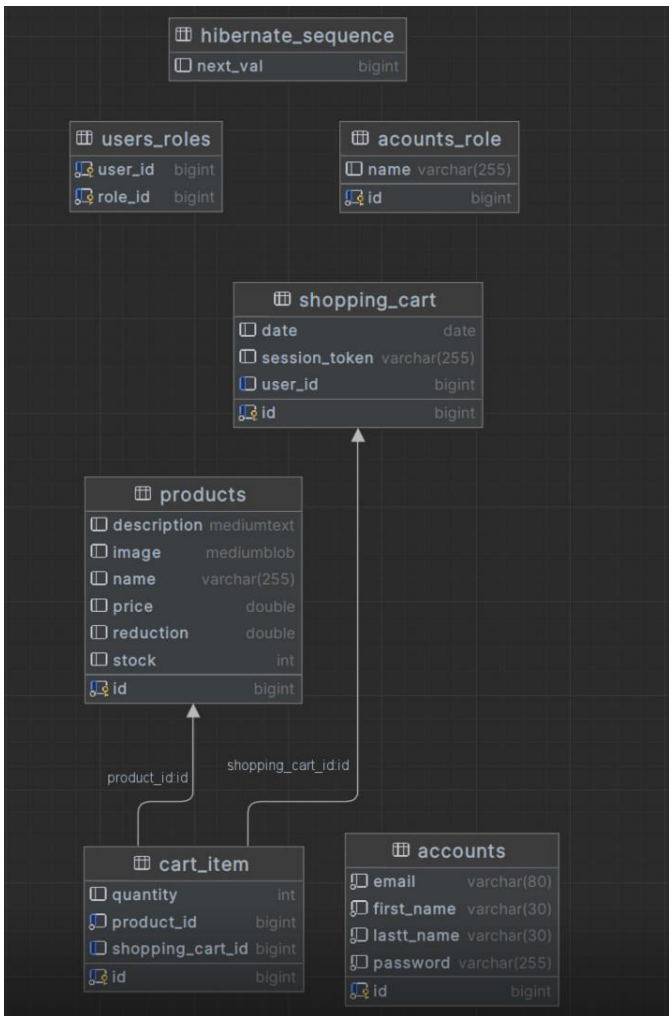

Рисунок 2.3 – Логічна модель бази даних, маркетплейсу вживаних товарів

2.6 Система шаблонів Thymeleaf

У розробці веб-застосунку для маркетплейсу важливим аспектом є забезпечення динамічного формування HTML-сторінок, що дозволяє відображати актуальні дані з бази, обробляти форми, керувати інтерфейсними станами та реагувати на дії користувачів. Для реалізації цього функціоналу було обрано систему шаблонів Thymeleaf, яка органічно інтегрується з екосистемою Spring Boot і забезпечує ефективну генерацію HTML-вмісту на сервері[4].

Thymeleaf – це сучасний Java-шаблонізатор, який дає змогу створювати динамічні веб-сторінки без потреби у клієнтському JavaScript. Його основною перевагою є синтаксис, наближений до звичайного HTML, що

робить шаблони читабельними навіть поза середовищем виконання. Завдяки можливості попереднього перегляду сторінок без запуску серверу, розробники можуть тестувати верстку і логіку шаблонів без складної інфраструктури.

Використання Thymeleaf дозволило реалізувати у маркетплейсі низку критичних для взаємодії з користувачем функцій: виведення списків товарів, заповнення форм публікації оголошень, обробку повідомлень про помилки, відображення повідомлень про успішну дію, навігаційних панелей залежно від ролі користувача тощо. За допомогою виразів у форматі ${} та th:* можна легко прив’язувати дані з моделі Spring до елементів сторінки.

Порівняно з альтернативами, такими як JSP, Freemarker або сучасні клієнтські фреймворки (React, Vue), Thymeleaf має ряд переваг для проєктів, де пріоритетом є серверна логіка. JSP, хоч і традиційна технологія для Java, менш зручна для реалізації складних форм та умовного рендерингу. Freemarker забезпечує ширші можливості форматування, але складніший у синтаксисі й має гіршу інтеграцію з Spring Boot. Використання React або Angular доцільне в клієнт-орієнтованих SPA-застосунках, однак вони потребують окремої збірки, розгортання та підтримки REST API, що не відповідає архітектурі обраного проєкту.

У межах цього дипломного проєкту Thymeleaf забезпечує повноцінну генерацію інтерфейсу з урахуванням ролей користувачів, мовної локалізації, відображення помилок валідації та повідомлень. Він не вимагає додаткових залежностей, легко підключається в Spring Boot через spring-boot-starter-thymeleaf і підтримує конфігурацію як через application.properties, так і на рівні шаблонів.

Таким чином, використання Thymeleaf є обґрунтованим вибором для розробки серверного веб-інтерфейсу маркетплейсу. Ця технологія дозволяє ефективно поєднати бізнес-логіку на стороні сервера з гнучким і адаптивним відображенням на фронтенді, зберігаючи при цьому простоту структури, високу продуктивність і зручність підтримки коду.

На рисунку 2.4 зображено загальну схему роботи шаблонізатора Thymeleaf у зв’язці зі Spring Boot. Користувач надсилає HTTP-запит, який обробляється контролером, далі сервісний шар отримує необхідні дані з бази, і результат у вигляді моделі передається у шаблон Thymeleaf для генерації HTML-сторінки, що повертається у браузер користувача. Наприклад, додати діаграму, яка показує, як HTTP-запит користувача проходить через Spring Controller, передається до сервісного шару, отримує дані з бази, а потім дані передаються у шаблон Thymeleaf для формування HTML-сторінки. Така схема чітко ілюструє місце шаблонізатора у загальній архітектурі застосунку.

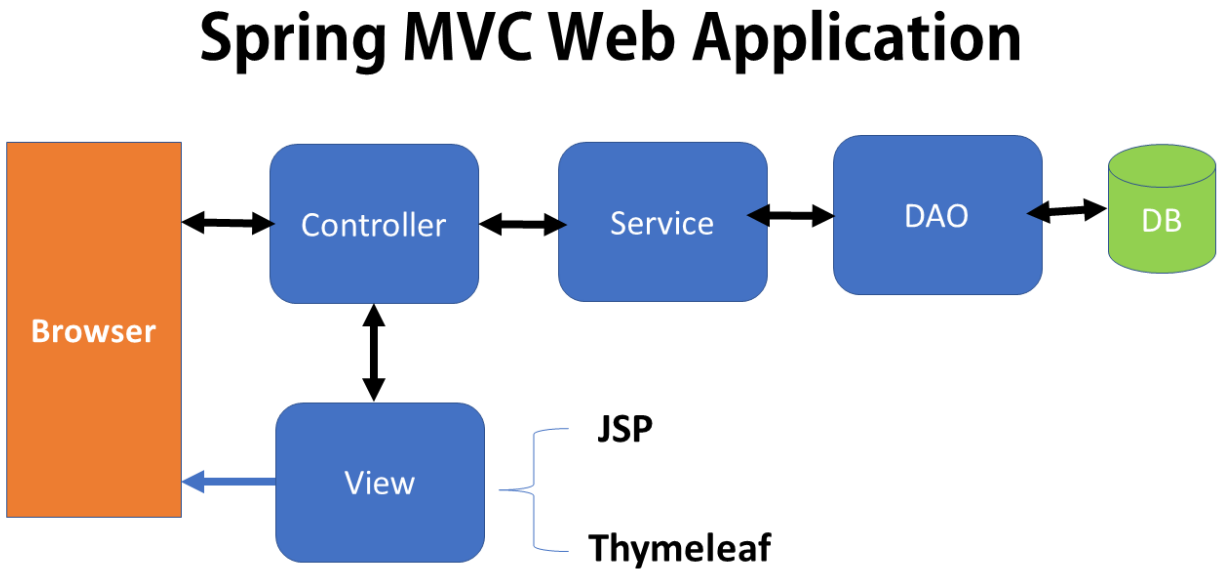

Рисунок 2.4 – Схема роботи Thymeleaf у Spring Boot-проєкті

2.7 Інструменти підтримки розробки: Maven, Lombok, DevTools

У процесі створення веб-застосунку важливе місце займають інструменти, які полегшують і прискорюють розробку, тестування та підтримку коду. Для реалізації маркетплейсу вживаних товарів було використано три ключові засоби підтримки: Maven, Lombok та Spring Boot DevTools. Їх застосування дозволило зробити процес розробки більш

автоматизованим, структурованим і менш залежним від рутинних дій.

Maven виконує роль системи керування збіркою, яка відповідає за автоматичне завантаження бібліотек, компіляцію проєкту, запуск тестів та формування фінального .jar-файлу. Завдяки використанню файлу pom.xml розробник може централізовано керувати версіями залежностей, уникати конфліктів і додавати нові бібліотеки без ризику порушення сумісності. Maven також підтримує профілі збірки для різних середовищ, що спрощує процес деплою. У дипломному проєкті через Maven було підключено такі модулі, як Spring Web, Spring Security, Spring Data JPA, MySQL Connector, Thymeleaf Starter, Lombok та DevTools.

Lombok – це Java-бібліотека, яка автоматично генерує повторювані частини коду, зокрема гетери, сетери, конструктори, методи equals() та toString(). Це значно скорочує обсяг коду моделей, підвищує його читабельність і зменшує кількість помилок. Наприклад, замість ручного написання десятків рядків у класі Product, достатньо додати анотації @Data, @NoArgsConstructor, @AllArgsConstructor, і необхідні методи будуть створені під час компіляції. Це особливо зручно у проєктах з великою кількістю сутностей і DTO-об’єктів, що мають просту структуру[12, 13].

Spring Boot DevTools – це модуль, який значно покращує ефективність під час локальної розробки. Він забезпечує автоматичне перезавантаження застосунку при зміні коду, що дозволяє розробникам миттєво бачити результати своїх змін без повного перезапуску проєкту. Крім того, DevTools вмикає зручні режими кешування, логування та автоперезавантаження шаблонів. У межах розробки маркетплейсу це дало змогу значно пришвидшити цикл правок, особливо на стадії верстки інтерфейсу та налаштування форм.

Порівняно з альтернативами (Gradle, Ant, ручна генерація коду, налаштування через IDE), обрані рішення демонструють високу стабільність, підтримку з боку спільноти та активну інтеграцію з Spring Boot. Gradle, хоч і забезпечує гнучкіший DSL для налаштувань, є складнішим для початківців і

менш передбачуваним у випадку конфліктів залежностей. Натомість Maven надає сувору декларативну модель, яка підходить для академічних і корпоративних проєктів.

Таким чином, комбінація Maven, Lombok та DevTools створює ефективне середовище для розробника, що забезпечує швидке налаштування, зменшення рутинної роботи та оперативну перевірку змін. Їх використання сприяло підвищенню продуктивності на всіх етапах розробки second-hand платформи – від проєктування до тестування.

2.8 Адаптивний інтерфейс з Bootstrap

Забезпечення зручного інтерфейсу користувача є критичним етапом у створенні сучасного веб-застосунку. Особливо це стосується маркетплейсів, де користувачі взаємодіють із системою з різних пристроїв – комп’ютерів, планшетів, смартфонів. У межах розробки second-hand платформи було вирішено використовувати фреймворк Bootstrap як основний інструмент для реалізації адаптивного, стильного та функціонального інтерфейсу.

Bootstrap – це популярна фронтенд-бібліотека з відкритим кодом, що містить набір CSS- і JavaScript-компонентів для швидкої розробки веб-інтерфейсів. Завдяки модульній структурі та готовим класам, Bootstrap дозволяє реалізувати адаптивну верстку без написання великого обсягу кастомного коду. Це значно пришвидшує розробку й спрощує підтримку інтерфейсу.

У дипломному проєкті Bootstrap був використаний для побудови навігаційного меню, форм авторизації та публікації товарів, таблиць замовлень, карток товарів, повідомлень про помилки, повідомлень успіху, кнопок керування та інших стандартних елементів інтерфейсу. Завдяки використанню сіткової системи (grid system), інтерфейс коректно масштабується на різних пристроях, а компоненти автоматично підлаштовуються під ширину вікна браузера.

Окрім адаптивності, Bootstrap забезпечує естетичну узгодженість стилів: усі компоненти мають єдиний візуальний стиль, що формує відчуття цілісності системи. Також у нього вбудовано інструменти для валідації форм, створення модальних вікон, випадаючих меню та повідомлень. Це дозволяє не лише пришвидшити розробку, а й підвищити зручність взаємодії для кінцевого користувача.

Порівняно з альтернативами, такими як Foundation, Bulma або ручна верстка з нуля, Bootstrap демонструє найвищу сумісність із сучасними браузерами, має багаторічну підтримку спільноти та численні приклади реалізації. Використання React або Angular Material потребує складнішої інтеграції й часто застосовується в SPA-додатках, які не відповідають архітектурі класичного серверного рендерингу з Thymeleaf.

Таким чином, Bootstrap дозволив ефективно реалізувати інтерфейс користувача маркетплейсу, забезпечити його адаптивність, функціональність та візуальну привабливість без додаткових витрат часу та ресурсів. Завдяки цьому веб-застосунок відповідає очікуванням сучасного користувача як з точки зору дизайну, так і з погляду зручності користування на різних пристроях.

3 РЕАЛІЗАЦІЯ СИСТЕМИ SECOND-HAND МАРКЕТПЛЕЙСУ

3.1 Концепція та функціональна модель платформи

Розробка веб-застосунку для маркетплейсу вживаних товарів базується на ідеї створення універсального середовища, де будь-який користувач може розмістити товар на продаж або здійснити покупку в іншого користувача без посередників. Концептуальна модель системи передбачає прямий обмін між сторонами, прозору взаємодію, можливість обміну відгуками та безпечну оплату, що відповідає сучасним вимогам цифрової торгівлі у сфері second-hand.

Функціональна модель платформи побудована на ролях користувачів, які взаємодіють із системою. У межах даного застосунку передбачено дві основні ролі: звичайний користувач та адміністратор. Користувач має можливість зареєструватися, увійти до системи, створювати оголошення, переглядати товари інших користувачів, додавати товари до кошика, оформлювати замовлення, залишати відгуки та керувати власним профілем. Адміністратор додатково має змогу модерувати оголошення, переглядати всі зареєстровані акаунти, обмежувати доступ до певного функціоналу та здійснювати загальне адміністрування контенту.

З точки зору архітектури, веб-застосунок реалізовано як класичний монолітний серверно-клієнтський додаток, у якому вся логіка обробки запитів і відображення результатів зосереджена на стороні сервера. Користувацький інтерфейс реалізовано за допомогою Thymeleaf у поєднанні з Bootstrap, що забезпечує адаптивність і зручність взаємодії. Серверна частина побудована з використанням Spring Boot та модулів Spring MVC, Spring Security, Spring Data JPA. Зберігання даних реалізовано на базі MySQL, яка забезпечує цілісність і структурованість даних.

Функціональна модель включає такі ключові модулі:

- Модуль реєстрації та автентифікації користувачів, що реалізує збереження облікових записів, захист паролів, вхід до системи та рольовий розподіл доступу.

- Модуль оголошень, що дозволяє створювати, редагувати, переглядати та видаляти товари. Кожен товар містить опис, фото, категорію, ціну та статус.

- Модуль замовлень та кошика, який забезпечує логіку додавання товарів до кошика, оформлення замовлень, перегляду історії покупок.

- Модуль оплати, для обробки онлайн-платежів.

- Модуль коментарів та оцінок, що дозволяє залишати відгуки про товари або продавців, тим самим підвищуючи довіру між учасниками платформи.

- Модуль багатомовності, який забезпечує локалізацію інтерфейсу на українську та англійську мови.

Взаємодія між модулями організована через контролери Spring MVC, які приймають HTTP-запити, викликають відповідні сервіси та повертають динамічно згенеровані HTML-сторінки. Всі дані зберігаються у відповідних таблицях бази даних, зв’язаних через зовнішні ключі.

Таким чином, концепція застосунку поєднує класичні принципи побудови e-commerce платформ із врахуванням специфіки вторинного ринку: гнучка модель товарів, система репутації, простота взаємодії та мінімізація технічних бар’єрів для користувача. Реалізована функціональна модель підтримує повний цикл взаємодії: від публікації оголошення до завершення замовлення з оплатою.

3.2 Система ролей та управління обліковими записами

Ефективна організація доступу до ресурсів та функціоналу веб-застосунку є критичним компонентом будь-якої системи з участю користувачів. У межах розробки маркетплейсу для продажу вживаних

товарів було реалізовано систему облікових записів з чітко визначеними ролями, яка забезпечує диференційований доступ до функцій платформи, гарантує безпеку та дозволяє масштабувати логіку взаємодії в майбутньому. На рисунку 3.1 зображено інтерфейс реєстрації нового користувача, який дозволяє створити обліковий запис через введення персональних даних.

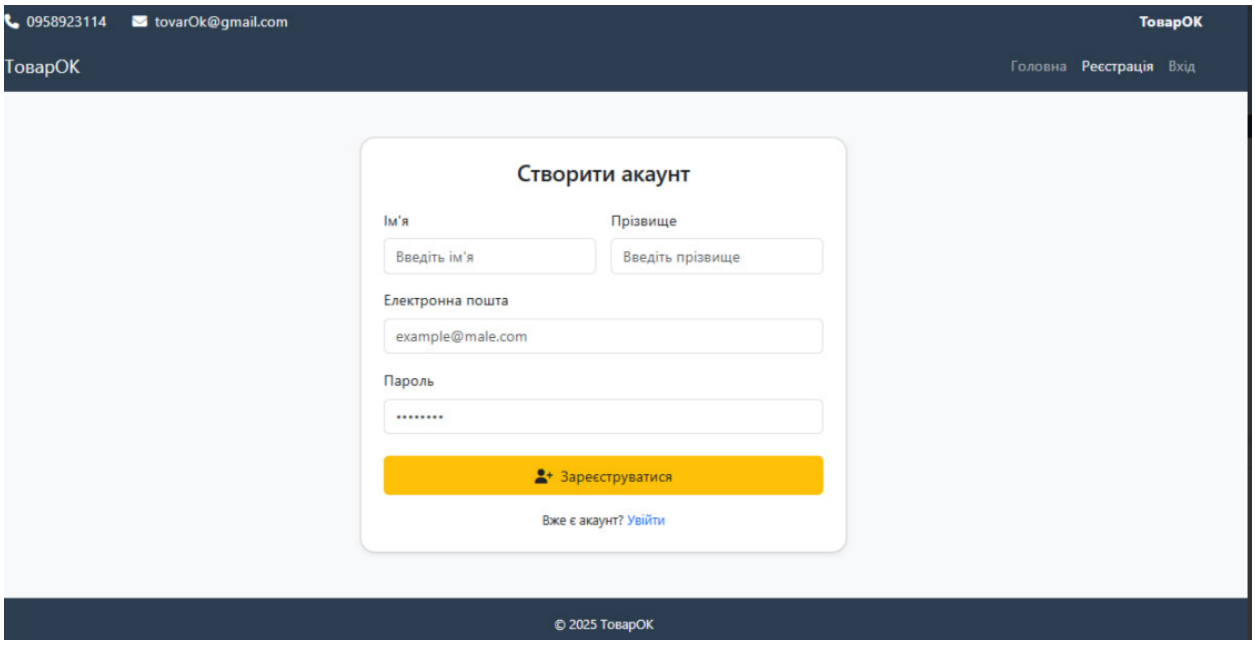

Рисунок 3.1 – Сторінка створення облікового запису

У структурі застосунку передбачено дві основні ролі: користувач (USER) та адміністратор (ADMIN). Користувач – це типовий учасник маркетплейсу, який має право створювати оголошення, переглядати товари, залишати відгуки, додавати товари до кошика, оформлювати замовлення та керувати власним профілем. Адміністратор, у свою чергу, має розширені права: перегляд усіх оголошень, управління користувачами, модерація контенту та доступ до адміністративної панелі.

Кожен користувач має унікальний обліковий запис, який створюється під час реєстрації. З метою захисту даних у системі реалізовано автентифікацію через логін і пароль.

На рисунку 3.2 показано форму входу до системи, яка реалізує авторизацію користувача на основі електронної пошти та паролю.

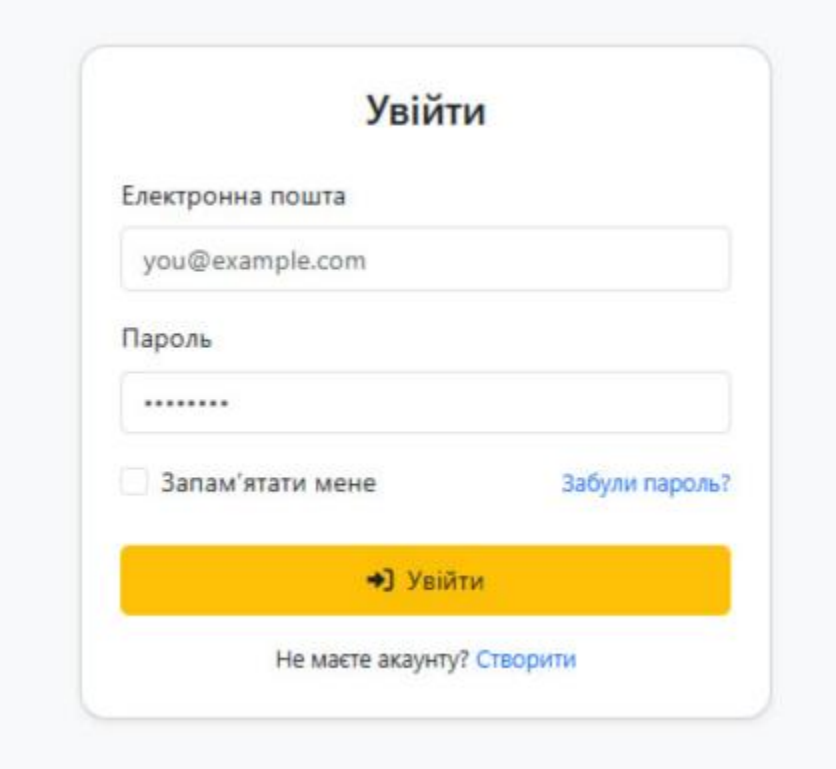

Рисунок 3.2 – Сторінка авторизації користувача

Для безпечного зберігання паролів використовується хешування за допомогою алгоритму BCrypt, що унеможливлює їх розшифрування навіть у разі компрометації бази даних.

Сама система ролей реалізована через сутність Role, яка пов’язана з користувачем відношенням «багато до одного». Це дає змогу легко розширювати набір ролей у майбутньому – наприклад, додати роль «модератор» або «супер-адмін». При автентифікації Spring Security використовує об’єкт UserDetails, до якого автоматично додається інформація про роль користувача, що дозволяє здійснювати перевірку прав доступу у фоновому режимі.

Відповідні обмеження доступу налаштовано у конфігураційному класі безпеки. Наприклад, маршрути /admin/** доступні лише для користувачів із роллю ADMIN, тоді як доступ до створення оголошення чи перегляду замовлень мають лише автентифіковані користувачі.

На рисунку 3.3 представлено інтерфейс для додавання товару, доступний лише зареєстрованим користувачам або продавцям.

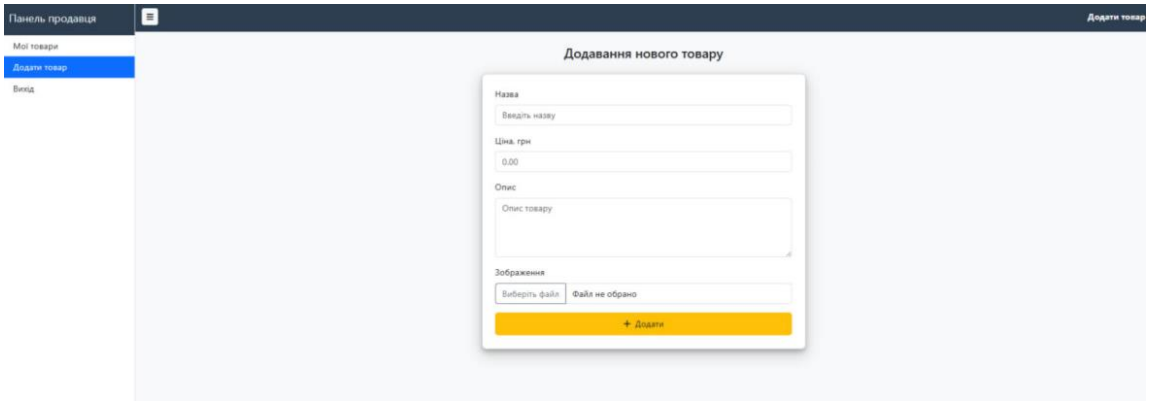

Рисунок 3.3 – Додавання товару у веб-застосунку

Інтерфейс також динамічно адаптується під роль: у навігаційній панелі адміністратора відображаються додаткові елементи, а для звичайних користувачів – лише базовий функціонал.

Усі облікові записи зберігаються у таблиці users, яка містить базову інформацію (логін, ім’я, електронна пошта, роль, статус активності). Через адміністративний інтерфейс передбачено можливість переглядати зареєстрованих користувачів, блокувати підозрілі облікові записи або змінювати їм ролі вручну.

На рисунку 3.4 показано сторінку панелі продавця, де відображаються замовлення разом із контактними даними покупця.

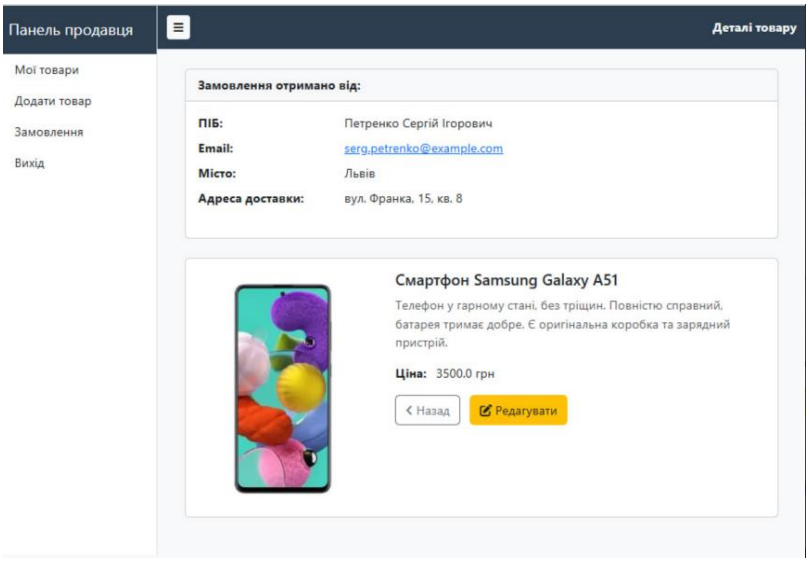

Рисунок 3.4 – Перегляд отриманого замовлення в панелі продавця

З точки зору майбутнього розвитку системи така модель є масштабованою та безпечною. Вона дозволяє легко додати нові ролі, змінити політики доступу без модифікації основної бізнес-логіки та динамічно керувати правами через централізовані правила.

Таким чином, реалізована система авторизації, ролей і управління обліковими записами не лише забезпечує надійну взаємодію користувачів із платформою, а й створює основу для подальшого розвитку застосунку з акцентом на безпеку, гнучкість та контроль.

3.3 Структура даних і логіка взаємодії моделей

Особливу увагу було приділено нормалізації структури: уникнено дублювання інформації, забезпечено розділення відповідальностей між моделями, використано індекси на полях пошуку (назва товару, категорія, email користувача), що забезпечує швидке виконання запитів. Взаємодія моделей із базою даних реалізована через репозиторії Spring Data JPA, які забезпечують доступ до CRUD-операцій без необхідності вручну писати SQL-запити. Для складніших запитів (наприклад, вибірка товарів за категорією та ціновим діапазоном) використовуються спеціалізовані методи з фільтрацією, пагінацією та сортуванням. Завдяки ретельно спроєктованій структурі даних, система підтримує масштабованість і легко адаптується до додавання нових функцій: збережених пошуків, обміну повідомленнями, системи повернень або кешування популярних запитів.

На рисунку 3.5 представлено UML-діаграму структури основних класів застосунку, яка демонструє зв’язки між сутностями, сервісами, контролерами, репозиторіями та конфігураційними компонентами. Вона ілюструє, як реалізовано архітектурні принципи розділення відповідальностей (Separation of Concerns), що підвищує модульність та підтримуваність коду. На діаграмі можна побачити, як сутності Product, CartItem, ShoppingCart, AccountEntity пов’язані між собою та взаємодіють із

відповідними сервісами, а також те, як Spring Security конфігурує авторизацію через SecurityConfiguration, CustomRememberMeServices та інші компоненти.

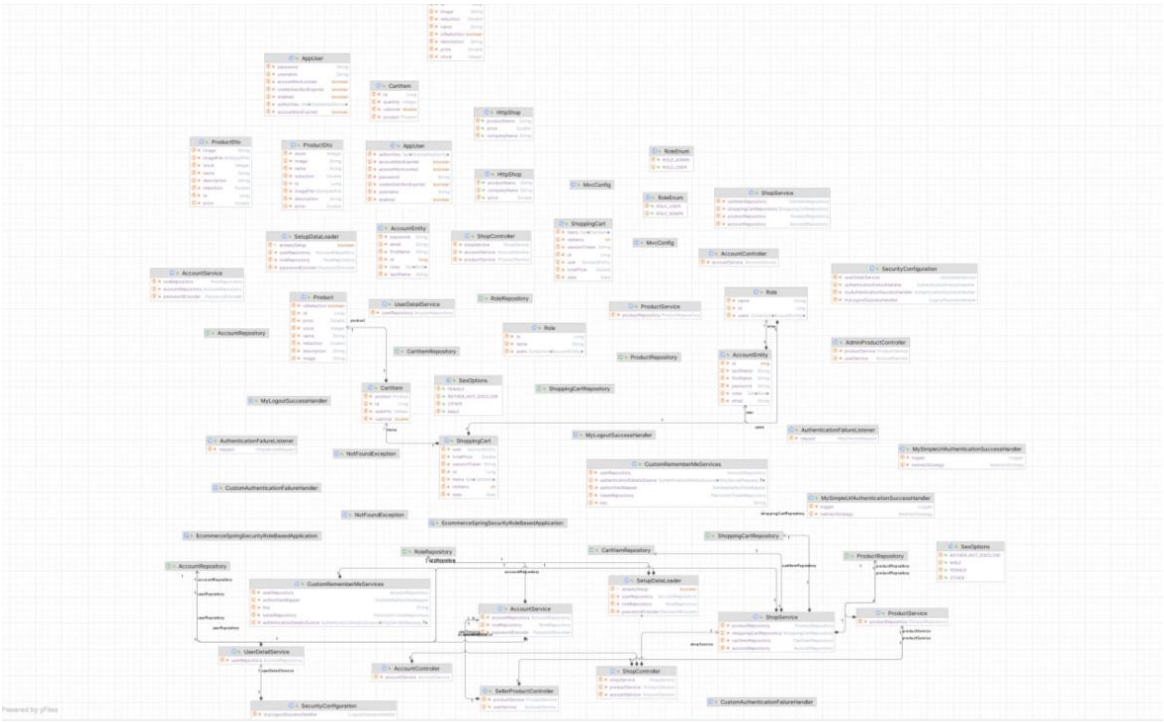

Рисунок 3.5 – UML-діаграма структури класів веб-застосунку на Spring Boot

Таким чином, логічна модель проєкту дозволяє ефективно управляти інформацією, забезпечує швидку взаємодію між об’єктами та служить надійним фундаментом для реалізації всіх функціональних сценаріїв платформи.

3.4 Каталог товарів: подача, перегляд, пошук

Формування каталогу товарів є центральною частиною функціональності маркетплейсу, оскільки саме через нього здійснюється основна взаємодія користувача з платформою. Реалізація механізмів подачі, перегляду та пошуку оголошень дозволяє забезпечити повний цикл розміщення і купівлі вживаних товарів – від додавання продавцем до знаходження покупцем потрібного товару.

У межах проєкту реалізовано інтуїтивно зрозумілу систему подачі товару. Після входу до системи користувач має можливість створити нове оголошення, вказавши його назву, опис, ціну, категорію, стан, контактну інформацію та додавши зображення. На рівні моделі товару (Product) забезпечено перевірку валідності введених даних: обов'язковість полів, числовий формат ціни, обмеження розміру опису. Крім того, впроваджено автоматичне збереження дати створення товару та його початкового статусу (активний/неактивний).

На рисунку 3.6 показано сторінку з особистими оголошеннями продавця, де доступне редагування та видалення кожного товару.

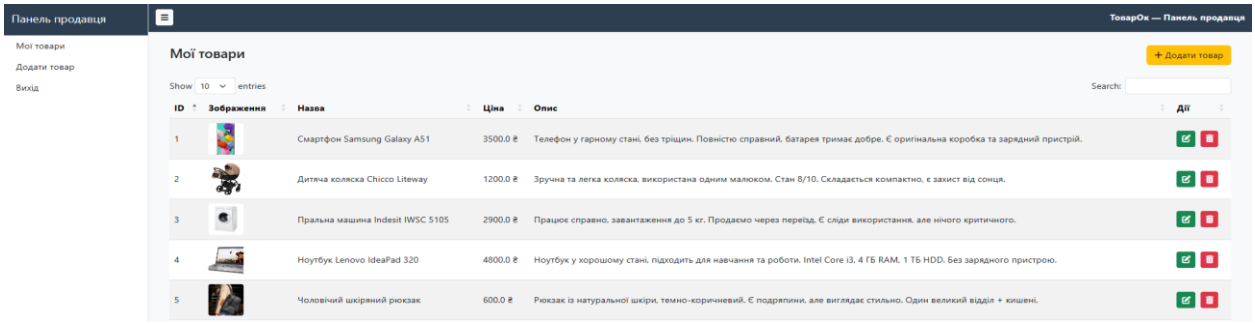

Рисунок 3.6 – Перелік товарів продавця з елементами керування

Каталог товарів реалізовано у вигляді динамічної HTML-сторінки з використанням Thymeleaf і Bootstrap. Для кожного товару створюється картка, яка містить ключову інформацію: фото, назву, ціну, категорію, короткий опис і кнопку для перегляду повної інформації.

На рисунку 3.7 зображено приклад відображення товару в каталозі з кнопкою додавання до кошика.

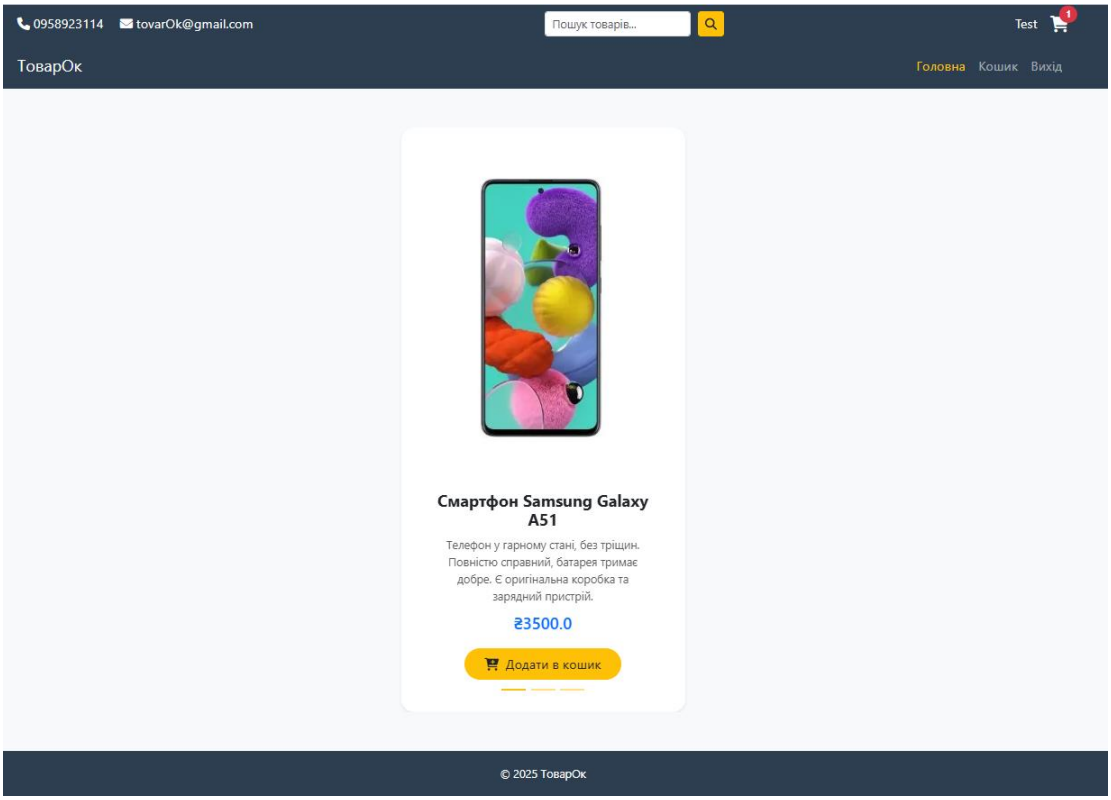

Рисунок 3.7 – Відображення товару в загальному каталозі

На рисунку 3.8 показано приклад картки ще одного товару – пральної машини – у стандартному форматі, аналогічному іншим оголошенням.

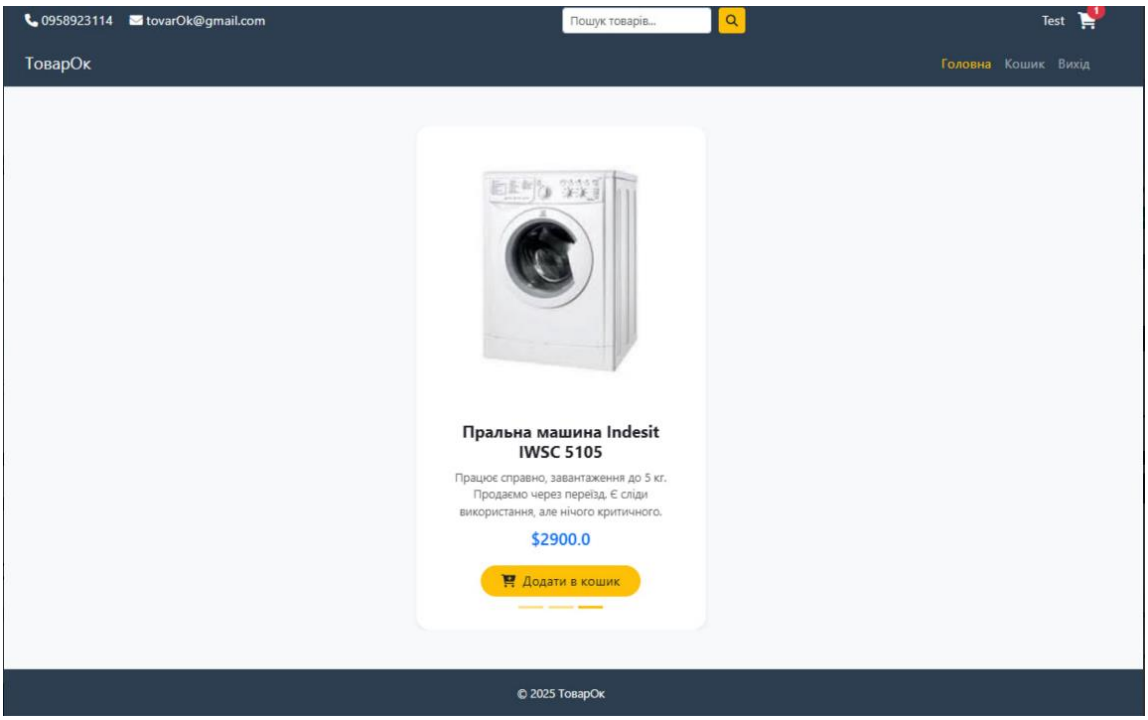

Рисунок 3.8 – Приклад товару “Пральна машина” у каталозі

Відображення каталогу організовано через пагінацію, що дозволяє уникнути перевантаження сторінки при великій кількості товарів.

Кожна картка товару веде до окремої сторінки з детальним описом, контактними даними продавця та кнопкою для додавання до кошика. Власник товару бачить додаткові елементи керування (редагування, видалення), тоді як інші користувачі можуть залишити коментар або оцінити оголошення.

Окрему увагу було приділено реалізації пошуку. У верхній частині каталогу розміщено панель фільтрів, яка дозволяє користувачеві:

- здійснювати пошук за ключовим словом у назві або описі товару;
- обирати категорію з випадаючого списку;
- задавати ціновий діапазон;
- сортувати результати за ціною, датою додавання або популярністю.

Фільтрація реалізована через методи в репозиторії Spring Data JPA, які автоматично формують SQL-запити на основі параметрів, переданих із фронтенду. Це дозволяє ефективно обробляти запити навіть при великому обсязі даних. Усі фільтри зберігаються в URL, що дозволяє користувачам ділитися результатами пошуку або зберігати посилання на відповідні товари.

Для підвищення зручності було реалізовано попереднє завантаження зображень, що оптимізує швидкість завантаження сторінки, та реалізовано перевірку типу файлу й розміру при завантаженні фото під час подачі товару.

Завдяки зазначеному підходу, платформа забезпечує зручну та швидку взаємодію користувачів з каталогом, підтримує адаптивність інтерфейсу та надає всі необхідні інструменти для комфортного пошуку і навігації. Це підвищує залученість користувачів, збільшує тривалість сесії та стимулює до повторного використання платформи.

3.6 Механізм додавання, редагування і видалення оголошень

Ефективне керування контентом є ключовою функціональною вимогою

для будь-якого маркетплейсу. У системах, де користувачі самостійно створюють оголошення, необхідно забезпечити інтуїтивно зрозумілий, стабільний і захищений механізм додавання, редагування та видалення товарів. У межах проєкту платформи для продажу вживаних товарів реалізація цього функціоналу була пріоритетною з точки зору юзабіліті та відповідності реальним сценаріям взаємодії.

Додавання нового оголошення доступне лише для авторизованих користувачів. Після входу до системи, користувач переходить на сторінку створення оголошення, де за допомогою динамічної форми вводить необхідні дані: назву товару, короткий опис, детальний опис, категорію, ціну, стан, контактну інформацію та додає фотографії. Для зручності реалізовано випадаючі списки категорій та автоматичну перевірку правильності введених значень. Наприклад, ціна повинна бути додатнім числом, а зображення – відповідати дозволеним типам (JPG, PNG) і мати обмежений розмір.

Після успішного створення оголошення система автоматично прив’язує товар до поточного користувача (власника) та відображає його у загальному каталозі. Для безпеки та контролю якості оголошення набуває статусу “активний” лише після валідації, яку може проводити адміністратор у майбутніх версіях системи.

Механізм редагування реалізовано через окрему форму, яка відкривається при натисканні кнопки “Редагувати” у власному оголошенні. Важливо, що доступ до цієї функції має лише власник товару. Після внесення змін до форми, дані оновлюються через відповідний контролер, який перевіряє валідність інформації та перезаписує відповідні поля у базі даних. Усі зміни фіксуються у полі “дата оновлення”, що дозволяє відстежувати активність.

Функція видалення оголошення доступна у тому ж інтерфейсі. Після підтвердження дії користувачем оголошення повністю видаляється з бази або змінює свій статус на “неактивне” – залежно від політики платформи. В даному проєкті реалізовано фізичне видалення, однак у майбутньому

можливе впровадження “м’якого видалення” з архівацією.

Усі дії пов’язані з оголошенням – створення, редагування, видалення – реалізовано на основі CRUD-операцій, побудованих із використанням Spring MVC і Spring Data JPA. Кожен запит проходить через шар сервісів, де виконується логіка перевірки прав доступу, валідація даних і взаємодія з репозиторіями.

Інтерфейс реалізовано з використанням Thymeleaf і Bootstrap, що дозволяє створити зручні адаптивні форми з повідомленнями про помилки та успішне збереження. Також впроваджено обробку винятків – у випадку помилки в процесі редагування або видалення користувач отримує відповідне повідомлення.

Таким чином, реалізований механізм керування оголошеннями забезпечує повний життєвий цикл товару – від створення до завершення його актуальності – та створює зручний інструмент для користувачів, сприяючи наповненню платформи якісним контентом.

3.7 Кошик, оформлення замовлення та історія покупок

Функціонал кошика та оформлення замовлень є невід’ємною частиною будь-якого e-commerce рішення. У випадку second-hand маркетплейсу, де користувачі взаємодіють один з одним у форматі прямого продажу, особливо важливо забезпечити зручний, прозорий і стабільний механізм додавання товарів до кошика, підтвердження покупки та перегляду історії операцій. У рамках розробки було реалізовано повноцінну підсистему замовлень, що охоплює всі етапи: від вибору товару до завершення транзакції.

Кошик є тимчасовим сховищем для товарів, які користувач має намір придбати. Його реалізовано як окрему сутність, пов’язану з користувачем, що дозволяє зберігати вміст між сесіями. При додаванні товару до кошика створюється об’єкт CartItem, який містить посилання на товар, кількість (у випадку повторюваних товарів), дату додавання.

На рисунку 3.9 наведено вміст кошика користувача з можливістю оформити замовлення або продовжити покупки.

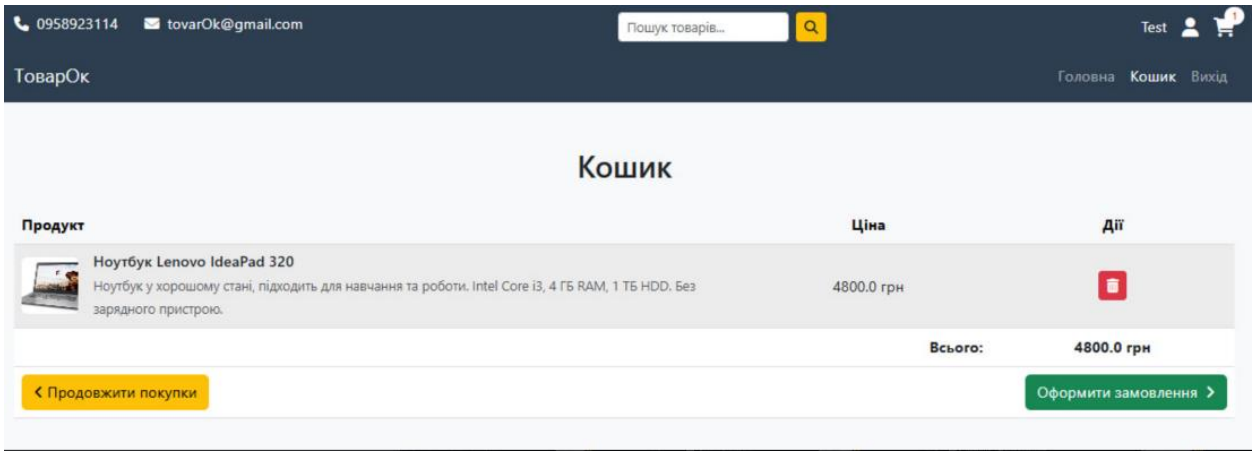

Рисунок 3.9 – Вміст кошика користувача перед оформленням замовлення

Оформлення замовлення реалізовано як окрема дія, що обробляється сервером через відповідний POST-запит. Під час оформлення система створює об’єкт Order, до якого додаються всі позиції з кошика (через сутність OrderItem), а також інформація про покупця, дата створення, загальна сума. Усі розрахунки виконуються автоматично – система підсумовує ціни всіх товарів, перевіряє їхню доступність, фіксує фінальні значення. Після цього користувач перенаправляється до форми оплати або отримує підтвердження створення замовлення.

На рівні бази даних реалізовано зв’язок між User – Order – OrderItem – Product, що дозволяє будувати складні запити й фільтрацію за датами, категоріями, статусами. Цей підхід забезпечує масштабованість системи та можливість розширення – наприклад, додавання статусів “у процесі обробки”, “очікує підтвердження продавця”, “відправлено” тощо.

Важливим елементом є також обробка помилок: якщо користувач намагається оформити замовлення на товар, який вже неактивний або проданий, система видає повідомлення з поясненням та пропонує оновити кошик. Також реалізовано механізм запобігання дублюванню замовлень шляхом перевірки одночасних запитів.

Таким чином, функціонал кошика та історії покупок виконує одразу кілька ключових завдань: забезпечує гнучкість вибору товарів, формує зручний процес оформлення замовлення та дозволяє користувачам повертатися до попередніх транзакцій. Це значно покращує досвід користувача і сприяє повторному використанню платформи.

3.8 Інтеграція платіжної системи

Реалізація онлайн-платежів є ключовим етапом у побудові функціональної e-commerce платформи, особливо у маркетплейсах, де користувачі взаємодіють між собою шляхом купівлі та продажу товарів. У межах дипломного проєкту було реалізовано інтеграцію з сучасною платіжною системою, яка дозволяє здійснювати безпечні розрахунки через банківські картки та популярні електронні платіжні сервіси. На рисунку 3.10 показано інтерфейс сторінки оплати замовлення через LiqPay, де користувач може обрати зручний спосіб – картку, Приват24, Google Pay тощо.

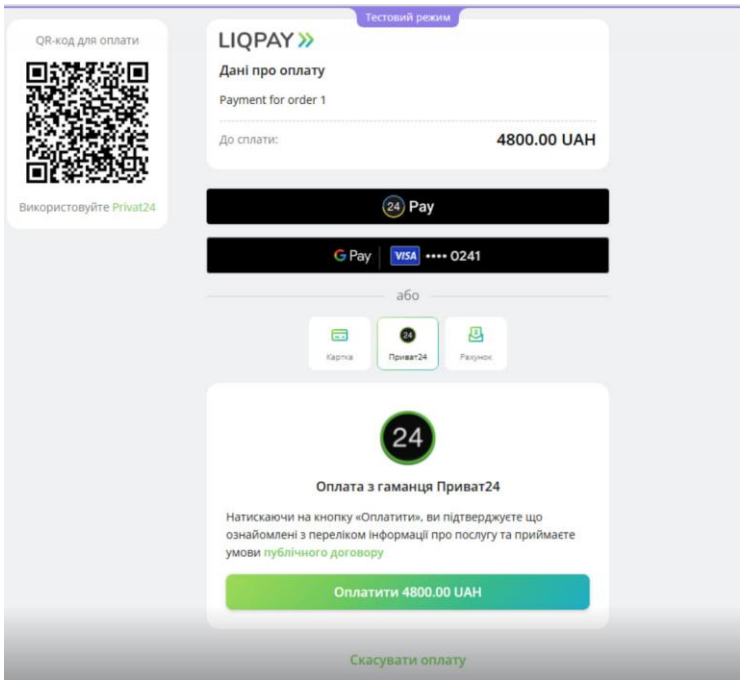

Рисунок 3.10 – Форма оплати замовлення через LiqPay

Вибір платіжної системи визначався низкою важливих факторів: наявністю офіційного API з детальною документацією, підтримкою локальної валюти, можливістю зберігати контактну інформацію покупця, додавати коментарі до транзакцій, а також зручними механізмами налаштування сценаріїв оплати. Крім того, дана система легко інтегрується із застосунками, написаними на сучасних мовах програмування, зокрема Java, завдяки підтримці SDK та наявності прикладів коду.

Процес онлайн-оплати побудовано так, що після підтвердження замовлення система автоматично генерує дані для платіжної форми: ідентифікатор продавця, суму до сплати, опис операції, унікальний номер замовлення, адресу для повернення після оплати, а також цифровий підпис для підтвердження справжності транзакції.

На рисунку 3.11 зображено форму входу у Приват24, яка використовується для підтвердження платежу з особистого кабінету користувача.

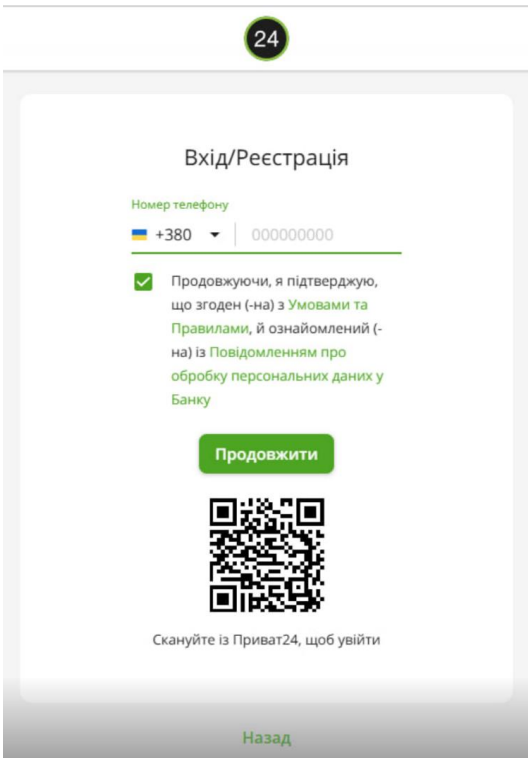

Рисунок 3.11 – Вхід у Приват24 для здійснення оплати

На клієнтській стороні платіжна форма інтегрується у веб-інтерфейс і дозволяє користувачу завершити оплату у зручний спосіб. Після підтвердження платежу платіжна система надсилає серверу callback-запит зі статусом операції. Сервер перевіряє справжність повідомлення, після чого змінює статус замовлення у базі даних залежно від результату оплати.

На рисунку 3.12 наведено приклад електронної квитанції, сформованої після успішної оплати, яка включає ідентифікатори транзакції, суму, дату й реквізити сторін.

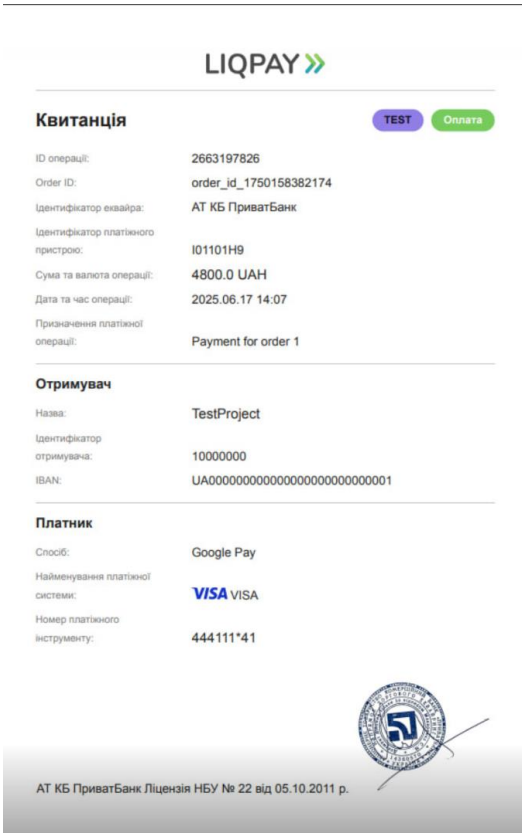

Рисунок 3.12 – Квитанція про оплату через LiqPay

Для забезпечення безпеки перевіряється достовірність отриманого повідомлення, а також здійснюється валідація контрольних сум і унікальних кодів платежу. У випадку відмови від оплати або помилки статус замовлення зберігається як неоплачений, що дозволяє користувачу повторити транзакцію у майбутньому.

На рисунку 3.13 представлено повідомлення про успішне завершення

платежу та можливість завантажити квитанцію або повернутись до сайту.

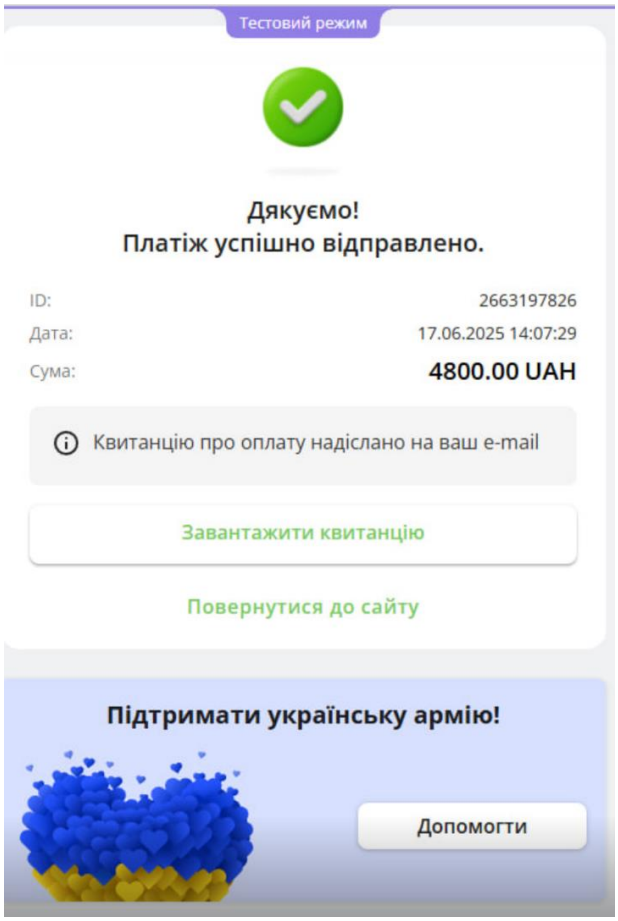

Рисунок 3.13 – Повідомлення про успішну оплату

Веб-застосунок також надає користувачу можливість переглядати історію своїх оплат із зазначенням дати, суми, статусу та номеру кожного замовлення. Це підвищує прозорість сервісу і забезпечує користувачеві додатковий рівень контролю над фінансовими операціями. Крім того, система залишає можливість для подальшого розширення – підключення додаткових платіжних сервісів або впровадження нових сценаріїв оплат, таких як розстрочка чи escrow.

Завдяки інтеграції платіжної системи вдалося реалізувати надійний і безпечний механізм оплати, що відповідає сучасним вимогам до e-commerce платформ, сприяє підвищенню довіри користувачів та забезпечує завершеність бізнес-процесів на платформі.

3.9 Захист даних і валідація введення

Безпека персональних даних і перевірка введеної інформації є критично важливими складовими будь-якого веб-застосунку, особливо в системах, що працюють з реєстрацією користувачів, онлайн-платежами, замовленнями та комунікацією між клієнтами. У межах реалізації second-hand платформи було впроваджено комплекс заходів із валідації даних та забезпечення їхнього захисту на всіх рівнях взаємодії – від інтерфейсу до бази даних.

Перший рівень захисту реалізовано за допомогою серверної валідації. Для всіх форм, які взаємодіють із моделями даних (реєстрація, авторизація, подача товару, оформлення замовлення), використано анотації з пакету javax.validation.constraints, зокрема @NotBlank, @Email, @Size, @Min, @Pattern. Це дозволяє автоматично перевіряти відповідність введених користувачем значень певним правилам ще до потрапляння інформації у базу даних. Результати валідації передаються на рівень представлення через модель, і за допомогою Thymeleaf користувач отримує локалізоване повідомлення про помилку.

Другий рівень – клієнтська валідація, реалізована за допомогою атрибутів HTML5 (required, type, min, maxlength) та скриптів. Це дозволяє попередити більшість очевидних помилок (наприклад, незаповнені поля, неправильний формат e-mail) ще до відправлення форми. Завдяки поєднанню клієнтської та серверної перевірки досягається високий рівень коректності введених даних і зменшується навантаження на сервер.

Окрему увагу приділено захисту персональної інформації. Паролі користувачів хешуються з використанням алгоритму BCrypt перед збереженням у базі даних. Це унеможливлює зчитування або відновлення справжніх паролів навіть у разі доступу до даних. Крім того, система не зберігає критичні платіжні дані – процес оплати здійснюється зовнішньою системою LiqPay, а лише статус і сума транзакції повертаються на сервер.

Для захисту від несанкціонованого доступу реалізовано рольову систему на основі Spring Security, яка обмежує доступ до певних маршрутів залежно від ролі користувача. Всі запити фільтруються через middleware, і лише авторизовані користувачі мають доступ до чутливих частин системи, таких як подача оголошень, перегляд історії покупок або керування акаунтом.

Також у проєкті реалізовано захист від типових веб-уразливостей, зокрема:

- CSRF (Cross-Site Request Forgery) – захист активовано за замовчуванням у Spring Security і реалізується через валідаційні токени в кожній формі;

- XSS (Cross-Site Scripting) – дані, введені користувачами (коментарі, назви товарів тощо), автоматично екрануються у шаблонах, запобігаючи вставці небезпечного коду;

- SQL-ін’єкції – виключені завдяки використанню ORM (JPA/Hibernate), де всі запити до бази даних параметризовані і не будуються вручну з рядків.

Додатково передбачено обмеження кількості запитів, які може виконати один користувач у межах короткого проміжку часу (rate limiting), що дозволяє протидіяти автоматизованим атакам типу brute force.

Таким чином, реалізовані механізми валідації введення та захисту даних створюють надійне середовище для користувачів маркетплейсу, мінімізуючи ризики компрометації інформації, помилок під час вводу і несанкціонованих дій. Це підвищує довіру до платформи та забезпечує відповідність стандартам безпеки сучасних веб-застосунків.

3.10 Інструкція користувача

Щоб розпочати роботу з платформою, відкрийте веб-застосунок у будь-якому сучасному браузері, ввівши його адресу в адресному рядку. Після

завантаження сторінки ви потрапите на головну, де можна переглянути каталог товарів, скористатися пошуком або увійти до облікового запису.

Нові користувачі можуть пройти реєстрацію, заповнивши просту форму з ім’ям, електронною адресою та паролем. Після підтвердження реєстрації можна увійти до системи через стандартну форму авторизації. Після входу відкривається доступ до повного функціоналу: додавання товарів, керування оголошеннями, перегляд замовлень і оформлення покупок.

Додавання товару здійснюється через спеціальну форму, де вказуються назва, опис, ціна, категорія та завантажується фото. На рівні моделі передбачено перевірку правильності введених даних. Після створення оголошення воно з’являється в загальному каталозі платформи. Каталог реалізований у вигляді динамічних карток із фото, короткою інформацією про товар і кнопкою перегляду.

При переході до картки товару можна ознайомитися з усіма деталями та, за бажанням, додати його до кошика. Оформлення замовлення відбувається у кілька кліків: після підтвердження покупки система перенаправляє користувача на платіжну форму LiqPay, де він завершує оплату зручним способом. Успішна транзакція підтверджується повідомленням, а сама квитанція доступна для перегляду або завантаження.

Оплачене замовлення зберігається в особистому кабінеті користувача, а кошик очищується. Також реалізовано можливість перегляду історії покупок і статусів замовлень. Для користувачів із розширеними правами доступна адміністративна панель з керуванням оголошеннями, переглядом користувачів і модерацією контенту.

Загалом інтерфейс платформи є інтуїтивно зрозумілим, а функціонал продуманий таким чином, щоб забезпечити зручну взаємодію на всіх етапах – від реєстрації до оплати.

ВИСНОВКИ

У межах даної кваліфікаційної роботи було спроєктовано та реалізовано повнофункціональний веб-застосунок second-hand маркетплейсу з використанням сучасного Java-стека технологій (Spring Boot, Spring Security, Spring Data JPA, Thymeleaf, MySQL, Bootstrap та ін.). У роботі детально проаналізовано проблематику та виклики цифровізації вторинного ринку, обґрунтовано необхідність створення спеціалізованої веб-платформи для безпечної, структурованої й зручної взаємодії між продавцями та покупцями вживаних речей.

Під час проєктування системи було враховано сучасні підходи до забезпечення надійності, цілісності й безпеки даних у розподілених обчислювальних середовищах, що є особливо важливим з огляду на зростання обсягів оброблюваної інформації, складність архітектури і можливі ризики втрат або пошкодження даних [1]. Зокрема, у процесі створення маркетплейсу приділялася увага питанням ролей користувачів, організації доступу та захисту персональних даних – із використанням таких рішень як Spring Security, валідація на сервері, а також шифрування паролів[15].

В межах проєкту було впроваджено такі сучасні підходи:

- архітектурний стиль REST – для побудови гнучких та масштабованих інтерфейсів взаємодії між клієнтом і сервером;
- ORM-рішення (JPA/Hibernate) – для ефективного управління даними й автоматичного створення таблиць у базі;
- механізми безпеки Spring Security – для надійної автентифікації, авторизації, захисту від найпоширеніших атак та безпечного зберігання паролів;
- інтеграція платіжної системи – для зручної й прозорої реалізації онлайн-оплат;

- адаптивний інтерфейс з Bootstrap – для забезпечення комфортної роботи користувачів на різних пристроях.

Результати роботи продемонстрували можливість створення сучасного, масштабованого та безпечного веб-сервісу для організації торгівлі вживаними товарами з урахуванням реальних потреб і очікувань кінцевих користувачів. Розроблена система може бути використана як основа для подальшого розвитку – впровадження додаткових ролей, інтеграції нових платіжних сервісів, розширення функціоналу мобільного доступу, розробки мобільного додатку або підключення сторонніх сервісів аналітики.

У ході виконання роботи було підтверджено ефективність обраних технологічних рішень та продемонстровано практичне застосування сучасних методів веб-розробки у сфері e-commerce. Запропонований second-hand маркетплейс не лише відповідає актуальним трендам цифрової трансформації, а й підвищує рівень довіри, прозорості й організованості у сфері вторинної торгівлі[1],[16].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Walls C. Spring in Action. Sixth Edition. – Manning Publications, 2022. – 600 p.

2. Bauer C., King G. Java Persistence with Hibernate. Second Edition. – Manning Publications, 2015. – 880 p.

3. MySQL Reference Manual. – [Електронний ресурс]. – Режим доступу: https://dev.mysql.com/doc/

4. Thymeleaf Documentation. – [Електронний ресурс]. – Режим доступу: https://www.thymeleaf.org/documentation.html

5. Bootstrap Documentation. – [Електронний ресурс]. – Режим доступу: https://getbootstrap.com/docs/

6. Черняк О. І. Електронна комерція: підручник. – К.: КНЕУ, 2018. – 272 с.

7. IT-Enterprise. Що таке маркетплейс? – [Електронний ресурс]. – Режим доступу: https://www.it-enterprise.com/blog/marketplace

8. OpenJDK Documentation. – [Електронний ресурс]. – Режим доступу: https://openjdk.org/

9. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. – Addison-Wesley, 2003. – 560 p.

10. JPA Specification. – [Електронний ресурс]. – Режим доступу: https://jakarta.ee/specifications/persistence/3.0/

11. Spring Security Reference. – [Електронний ресурс]. – Режим доступу: https://docs.spring.io/spring-security/reference/

12. Maven Documentation. – [Електронний ресурс]. – Режим доступу: https://maven.apache.org/guides/

13. Lombok Documentation. – [Електронний ресурс]. – Режим доступу: https://projectlombok.org/features/all

14. Binance Academy. What Is a Marketplace**?** – [Електронний ресурс]. –

Режим доступу: https://academy.binance.com/en/articles/what-is-a-marketplace

15. Крикливець В.В., Тимошенко Д.О., ІОТ-ПРИСТРОЇ: ОСНОВНІ СИСТЕМИ УПРАВЛІННЯ Сучасні напрями розвитку інформаційнокомунікаційних технологій та засобів управління: тези доповідей п’ятнадцятої міжнародної науково-технічної конференції. Т.3: секція 3,4. Баку: ІСУ АР; Харків: НТУ «ХПІ»;. Харків: ХНУРЕ; Харків: НАУ «ХАІ»; Жиліна: УмЖ. 2025. С. 48

16. Spring Boot Reference Documentation. – [Електронний ресурс]. – Режим доступу: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/