НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

«На правах рукопису»
УДК_____

«До захисту допущено»
Завідувач кафедри
_____ Наталія АУШЕВА
“___”_____2025 р.

Магістерська дисертація

на здобуття ступеня другого (магістерського) рівня вищої освіти
за освітньою програмою “Цифрові технології в енергетиці”
зі спеціальності 122 “Комп’ютерні науки”

на тему_____Автоматизація процесу закупівель_____
_____енергетичного обладнання_____
_____

Виконав: студент _2_ курсу, групи ТР-43мп
__Середа Владислав Олександрович_____ _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник к.ф.-м.н. , доцент Юрій ТАРНАВСЬКИЙ _____
(науковий ступінь, вчене звання, ім’я ПРІЗВИЩЕ) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, ім’я ПРІЗВИЩЕ) (підпис)

Н.контроль __асистент Володимир РУДИК_____ _____
(посада, ім’я ПРІЗВИЩЕ) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ — 2025

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти другий (магістерський)
спеціальність 122 “Комп’ютерні науки”
Освітньо-наукова програма “Цифрові технології в енергетиці”

ЗАТВЕРДЖУЮ
Завідувач кафедри ЦТЕ
_____ Наталія АУШЕВА_

«___»_____2025 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Середі Владиславу Олександровичу
(прізвище, ім’я, по батькові)

1. Тема роботи Автоматизація процесу закупівель енергетичного обладнання

керівник роботи *Тарнавський Юрій Адамович, к.ф.-м.н., доцент*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «__» листопада 2025 р. № _____

2. Термін подання студентом дисертації 06.12.2025

3. Об’єкт дослідження : Процес організації та управління закупівлями енергетичного обладнання в умовах цифровізації

4. Вихідні дані: Мова програмування Java, фреймворк Spring Boot, технології Spring Data JPA, Thymeleaf, база даних PostgreSQL, платіжний сервіс LiqPay, UML-діаграми, вимоги бізнес-процесів закупівель, REST-архітектура, протоколи інтеграції з платіжними сервісами.

5. Перелік завдань: Дослідити ключові бізнес-процеси закупівель енергетичного обладнання. Розробити архітектуру веб-системи автоматизації закупівель із розмежуванням ролей користувачів. Реалізувати модулі: оформлення закупівель,

управління кошиком, каталог обладнання та історія замовлень. Інтегрувати безпечний сервіс онлайн-платежів LiqPay, включно з підтримкою підписок. Провести тестування продуктивності, оптимізувати роботу системи.

6. Орієнтований перелік ілюстративного матеріалу : Архітектура веб-системи закупівель; діаграми UML (діаграма варіантів використання, ER-діаграма БД, діаграма класів); схеми бізнес-процесів закупівель; інтерфейси користувацьких модулів; блок-схема інтеграції з LiqPay; результати тестування навантажень.

7. Орієнтований перелік публікацій: роботу представлено на XXII-й міжнародній науково-практичній конференції молодих вчених та студентів “Сучасні проблеми наукового забезпечення енергетики”

8. Дата видачі завдання «16» вересня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів підготовки магістерської дисертації	Термін виконання	Примітка
1	Затвердження теми роботи	16.09.24р.	Виконано
2	Вивчення та аналіз задачі	01.09.25р.-10.09.25р.	Виконано
3	Розробка архітектури та загальної структури програмного забезпечення	01.09.25р.-20.09.25р.	Виконано
4	Розробка структур окремих підсистем	20.09.25р.-05.10.25р.	Виконано
5	Програмна реалізація системи	01.10.25р.-19.10.25р.	Виконано
6	Комплексне тестування програмного забезпечення	13.10.25р.-19.10.25р.	Виконано
7	Захист створеного програмного забезпечення	20.10.25р.	Виконано
8	Оформлення магістерської дисертації	20.10.25р.-20.11.25р.	Виконано
9	Передзахист	27.11.25р.	Виконано
10	Нормоконтроль	24.11.25р.-28.11.25р.	Виконано
11	Захист		Виконано

Студент _____ **Владислав СЕРЕДА**
(підпис) (ім’я, ПРІЗВИЩЕ)

Керівник роботи _____ **Юрій ТАРНАВСЬКИЙ**
(підпис) (ім’я, ПРІЗВИЩЕ)

РЕФЕРАТ

Сфера закупівель енергетичного обладнання стрімко переходить до цифрових рішень, оскільки традиційні паперові або напівавтоматизовані процеси не забезпечують достатньої швидкості, прозорості та точності під час взаємодії підприємств із постачальниками. Актуальність теми обумовлена зростанням потреби в ефективних IT-інструментах, здатних оптимізувати формування замовлень, контроль залишків, облік транзакцій, аналітику закупівель та подальшу взаємодію з постачальниками. Автоматизація цих процесів дає змогу зменшити людський фактор, прискорити ухвалення рішень та підвищити економічну ефективність підприємств енергетичного сектору.

Мета роботи полягає у розробленні веб-системи для автоматизації процесу закупівель енергетичного обладнання з використанням сучасних веб-технологій, механізмів обробки даних та інтеграції безпечних онлайн-платежів.

Завдання дослідження включають:

— визначення ключових бізнес-процесів у сфері закупівель енергообладнання;

— вибір технологій та інструментів для побудови надійної веб-платформи;

— розроблення модулів оформлення замовлень, аналітики, обліку та взаємодії з постачальниками;

— інтеграцію сервісу онлайн-платежів LiqPay;

— проведення тестування продуктивності та оптимізацію системи під реальні навантаження.

Об’єктом дослідження є процес організації та виконання закупівель енергетичного обладнання.

Предметом дослідження є методи та програмні засоби автоматизації закупівельних процесів у веб-середовищі.

Методи дослідження включають аналіз бізнес-процесів, проєктування архітектури інформаційних систем, застосування фреймворку Spring, моделювання

бази даних, інтеграцію зовнішніх API, а також тестування продуктивності та функціональне тестування веб-додатку.

Апробація результатів дослідження. Основні положення роботи представлено на XXII-й міжнародній науково-практичній конференції молодих вчених та студентів “Сучасні проблеми наукового забезпечення енергетики”.

Робота складається зі вступу, п’яти розділів, загальних висновків, списку використаних джерел і додатків. Загальний обсяг роботи становить 96 сторінок, включає 3 таблиці, 34 рисунка і 27 використаних джерел.

Ключові слова: автоматизація закупівель, енергетичне обладнання, веб-система, Spring Framework, PostgreSQL, LiqPay, бізнес-процеси.

ABSTRACT

The sphere of procurement of energy equipment will quickly move to digital solutions, as traditional paper or semi-automated processes do not provide sufficient speed, transparency and accuracy during the interaction of enterprises with suppliers. The relevance of the topic is due to the growing need for effective IT tools that can optimize order formation, balance control, transaction accounting, procurement analytics and further interaction with suppliers. Automation of these processes makes it possible to reduce the human factor, accelerate decision-making and increase the economic efficiency of enterprises in the energy sector.

Objective of the work is to develop a web system for automating the process of procurement of energy equipment using modern web technologies, data processing mechanisms and integration of secure online payments.

Research tasks include:

— identification of key business processes in the sphere of procurement of energy equipment;

— selection of technologies and tools for building a reliable web platform;

— development of modules for ordering, analytics, accounting and interaction with suppliers;

— integration of the LiqPay online payment service;

— performance testing and system optimization for real loads.

The object of the study is the process of organizing and executing purchases of energy equipment.

The subject of the study is methods and software tools for automating procurement processes in a web environment.

The research methods include business process analysis, information system architecture design, application of the Spring framework, database modeling, integration of external APIs, as well as performance testing and functional testing of the web application.

Approbation of the research results. The main provisions of the work were presented at the XXII International Scientific and Practical Conference of Young Scientists and Students “Modern Problems of Scientific Support of Energy”.

The work consists of an introduction, five sections, general conclusions, a list of sources used and appendices. The total volume of the work is 96 pages, includes 3 tables, 34 figures and 27 references.

Keywords: procurement automation, energy equipment, web system, Spring Framework, PostgreSQL, LiqPay, business processes.

ЗМІСТ

СПИСОК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

Фреймворк — це комплект інструментів та концепцій розробки програмного забезпечення, метою яких є прискорення та спрощення процесу створення нових проектів, використовуючи готовий стандарт. Такий підхід дозволяє багаторазове використання готового коду, утворені структурні шаблони проектування полегшують командну працю, велика частина функціоналу вже прихована та адаптована до реальних випробувань.

API (Application Programming Interface) — це визначені протоколи, метою яких є стандартизація взаємодії модулів програмного забезпечення, задля можливості використання функціоналу та взаємодії з іншими програмними компонентами.

БД — база даних.

СКБД — система керування базами даних.

DNS (система доменних імен) — розподілена ієрархічна система перетворення імен хостів в IP-адреси.

HTTP (Hypertext Transport Protocol) — це протокол прикладного рівня для передачі гіпермедійних документів, наприклад HTML. Його розроблено для зв’язку між веб-браузерами та веб-серверами, але його також можна використовувати для інших цілей.

SQL — мова структурованих запитів. SQL використовується для зв’язку з базою даних. Згідно з ANSI (American National Standards Institute) це стандартна мова для систем управління реляційними базами даних.

REST — підхід до архітектури мережевих протоколів, які надають доступ до інформаційних ресурсів.

ВСТУП

Кожного дня інтернет стає все дедалі орієнтованим на потреби користувача. Купівля поглиблюється у онлайн середовищі. Цьому існує величезна кількість пояснень, серед яких: доступність товарів з усіх куточків світу, зміцнення транспортного сполучення поміж країнами, перегляд відгуків до якості продукції та його постачальника, а найголовніше це економія часу без необхідності прямого прибуття за місцем перебування точки видачі. Головним аспектом успішності та конкурентоспроможності торгівлі є коефіцієнт «ціна/якість» наданих послуг. Цей показник невпинно покращується на користь клієнта, адже це і є дорученням цифрового світу.

Мета проєкту полягає у створенні веб-системи, яка інтегрує сучасні веб-технології, інструменти обробки даних та безпечні сервіси онлайн-платежів для автоматизації процесу купівлі енергетичного обладнання.

До **завдання дослідження** віднесено: розробка модулів для закупівель, аналітики, бухгалтерського обліку та взаємодії з постачальниками; вибір передових технологій та інструментів для створення надійної веб-платформи, інтеграція безпечного сервісу онлайн-платежів, проведення тестування продуктивності та оптимізація системи для фактичних навантажень.

Об’єкт дослідження — процес організації та виконання закупівель енергетичного обладнання.

Предмет дослідження — методи та програмні засоби автоматизації закупівельних процесів у веб-середовищі.

Методи дослідження включають: аналіз бізнес-процесів, функціональне та навантажувальне тестування, методи об’єктно-орієнтованого проєктування, інтеграцію зовнішніх API, проєктування архітектури веб-систем, використання фреймворку Spring та його модулів, моделювання баз даних у PostgreSQL.

Практичне значення отриманих результатів має розробка веб-системи, яку підприємства можуть використовувати для оптимізації процедур закупівель,

скорочення використання часу та ресурсів, зменшення людського фактору та покращення взаємодії з постачальниками. Розроблене рішення може бути використане в різних секторах, коли необхідно автоматизувати операції купівлі-продажу.

Робота була представлена на XXII міжнародній науково-практичній конференції молодих учених та студентів «Сучасні проблеми наукового забезпечення енергетики».

У першому розділі проведено аналіз предметної області, визначено ключові бізнес-процеси закупівель енергетичного обладнання, описано вимоги до системи та обґрунтовано актуальність автоматизації цих процесів.

У другому розділі детально розглянуто запропоновані методи вирішення прикладної задачі застосовані у проектуванні системи.

У третьому розділі розроблено основні програмні модулі платформи: оформлення замовлень, інструменти перегляду попередніх, реєстрація підписок тамодулі обліку та механізми редагування існуючого асортименту товарів. Також зміст розділу включає огляд архітектури програмних компонентів з елементами БД.

У четвертому розділі наведені приклади використання застосунку для наявних ролей у проекті.

П’ятий розділ демонструє особливості реалізації стартап-проєкту у проекції готового програмного продукту.

1 РОЗРОБКА ВЕБ-СИСТЕМИ АВТОМАТИЗАЦІЇ ПРОЦЕСУ ЗАКУПІВЕЛЬ ЕНЕРГЕТИЧНОГО ОБЛАДНАННЯ

Метою даної роботи є створення платформи автоматизації процесу закупівель енергетичного обладнання, розгляд специфікації та загальних характеристики додатку, методи впровадження та поточні моделі в цій галузі.

З кожним днем процес автоматизації зростає, і вся ручна праця, яка залежить від заздалегідь визначеного порядку кроків, тепер є стандартом для корпоративних операцій. Без автоматизованого програмного забезпечення та простого у використанні інтерфейсу важко уявити роботу більшості підприємств. Учасники та представники торгівлі генерують, обробляють та зберігають замовлення через портативні пристрої в репозиторії за допомогою спеціалізованих програм. Більшість замовлень з платформ отримання надходять у форматі, який можна архівувати та додатково документувати в протоколах.

1.1 Формування прикладної задачі

Мета роботи — створення платформи з автоматизованим платіжним функціоналом замовлення товарів електрообладнання. Система надає можливість реєструвати одноразове чи регулярне замовлення, списувати кошти інтегруючи сторонню платформу платежів, підтримувати адміністрування наявного асортименту товарів, архівацію замовлень у хроніці активності клієнта та фільтрацію компонентів відносно сумісності між собою.

1.1.1 Огляд характеристик системи

Сформовано загальні вимоги до програмного продукту, які стосуються основних характеристик, необхідних для його надійної та ефективної роботи. Стабільність системи є однією з першочергових вимог. Навіть у ситуаціях з

великими навантаженнями розроблене рішення повинно гарантувати відмовостійкість та безперебійну роботу. Для запобігання потенційним небезпекам вкрай важливо постійно проводити стрес-тестування та оптимізувати архітектурні методи, оскільки впровадження нових модулів може призвести до несумісності, порушень потоку або інших системних проблем.

Швидкість обробки запитів критично важлива для користувацького досвіду використання. Швидке формування представлень на стороні веб-сервера вимагає використання ефективних баз даних, методів кешування та оптимізації програмного коду. Це дозволяє гарантувати комфортну взаємодію користувачів з платформою та скоротити час відгуку системи.

Особлива увага зосереджена до безпеки обробки та передачі даних. Система повинна гарантувати цілісність, доступність та конфіденційність інформації. Використання сучасних методів моніторингу загроз, шифрування та автентифікації гарантує захист користувачів та зменшує можливість небажаного доступу.

Зручність використання інтерфейсу користувача є важливим фактором. Швидкому освоєнню функціональності сприяє логічна структура сайту, інтуїтивно зрозуміла навігація та уніфіковане представлення інформації. Загальна ефективність роботи користувачів з платформою підвищується завдяки передбачуваності та зрозумілості представлення елементів інтерфейсу.

Крім того, система приймає низку варіантів онлайн-оплати. Швидкі та безпечні транзакції забезпечуються завдяки інтеграції з надійними зовнішніми платіжними системами, що також підвищує безпеку та покращує зручність для клієнтів.

Відстеження активності користувачів є ще однією обов’язковою умовою. Щоб користувачі могли переглядати власні транзакції та оптимізувати процес відновлення даних у разі виникнення проблем, система повинна реєструвати дії, виконані на платформі. Насамперед, цей метод спрощує моніторинг стану системи та робить її роботу більш прозорою.

1.1.2 Існуючі рішення в мережі

Веб-ресурси зараз є одними з найбільш широко використовуваних технологій у цифровому світі. Швидкий розвиток інтернет-технологій, удосконалення браузерів та ефективність пошукових систем є причинами їх активного використання.

Через низку зовнішніх факторів попит на електротовари постійно зростає. У мережі вже є багато майданчиків, які спеціалізуються на продажу електроприладів. Приклади цих методів торгівлі, а також їхні переваги та недоліки, наведено нижче.

EVO заснувала Prom.ua, український торговельний майданчик, який об’єднує продавців з різних секторів для торгівлі товарами онлайн. Без потреби у спеціалізованих технологічних навичках, платформа пропонує підприємцям простий та економічно ефективний варіант для початку онлайн-торгівлі.

Однак, як і більшість торговельних майданчиків, Prom.ua має низку спільних недоліків, таких як високий рівень конкуренції, значна плата за успішні продажі, вимога додаткових витрат на розміщення товарів та складність виділення серед величезної кількості продавців. Крім того, немає багато можливостей для налаштування сайту магазину.

Мережа Volti та Elektrovoz є сучасними інтернет-магазинами електрообладнання, на відміну від попереднього ринку. Насправді мережа є домівкою для низки таких комп’ютеризованих торгових платформ, кожна з яких прагне максимізувати прибуток. Існує ряд відмінностей між цими платформами та власним сервісом. Наш сервіс припускає, що клієнту потрібно лише налаштувати веб-додаток на сервері, який має публічний домен і може обробляти запити зовнішніх користувачів. І навпаки, ці платформи призначені лише для окремих осіб; вони не пропонують можливості підключити згенероване програмне забезпечення до власного хоста, створити зручну конфігурацію, додати певні інновації або змінити вже існуючі компоненти.

Підсумовуючи, можна зазначити, що розроблена система належить до типу проміжних торгових веб-застосунків, які вже готові до практичного використання,

але водночас передбачають можливість подальшого вдосконалення та розширення функціональності відповідно до потреб користувача.

1.2 Методи розробки та форма представлення

Першочергово важливо подумати про обраний тип впровадження системи, а також про потенційні можливості презентації. Існує кілька методів взаємодії з програмною моделлю в корпоративному світі, а сучасна розробка програмного забезпечення розділена на сфери використання.

Розробка додатків, що підходять для настільних та мобільних операційних систем, є одним із популярних рішень. З іншого боку, створення веб-додатків вважається більш масштабною та адаптивною стратегією з кількома перевагами.

Встановлення програмного забезпечення, розробленого для певної операційної системи, вимагає часу та спеціалізованого обладнання. Веб-додатки працюють безпосередньо через браузер, який пропонує доступ до кожної сторінки в мережі, незважаючи на споживання певної оперативної пам’яті. Хоча кешування використовується обома типами додатків для зберігання даних сеансу, веб-браузери виділяються завдяки своїй більшій функціональності та адаптивності.

Методика розробки програмного забезпечення є досить широкою, і не всі підходи слід враховувати, оскільки деякі з них адаптовані до певного набору завдань. Найвідоміші методи включають каскадний, ітеративний та стандартний покроковий код, написаний та налагоджений.

Також важливо позначити, як користувач взаємодіє з платформою. Цей програмний продукт використовує техніку «клієнт-сервер» для обміну даними, тоді як «сервер-сервер» використовується під час взаємодії зі стороннім сервісом. Оскільки сучасні програми постійно розвиваються, їх важко віднести до певного типу дизайну. В результаті виникають нові гібридні моделі залежно від характеру завдання та рівня складності написання.

1.3 Огляд програмних засобів

Цей веб-застосунок засновано з використанням архітектури «клієнт-сервер». Для реалізації фронтенду використовується JavaScript разом з розміткою HTML та стилізацією CSS. Рішення екосистеми Spring, такі як Spring Framework, Spring Boot, Spring Web, Spring Data JPA та Spring Session, використовуються для побудови логіки сервера мовою Java.

1.3.1 Мова програмування Java

Мова програмування Java стала глобальним інструментом для розробки різноманітних програмних рішень, включаючи серверні платформи та системи автоматизації [1]. Її методологія розробки побудована на модульності та чіткому розподілі логіки, що робить код багаторазовим та дозволяє легше підтримувати величезні проекти.

Особливу увагу слід приділити методу виконання програм Java, який використовує проміжний формат, що обробляється середовищем виконання, а не пряму компіляцію для конкретної операційної системи. Це означає, що програми не потрібно переписувати для кожної платформи, щоб вони працювали на багатьох пристроях [2].

Широка екосистема інструментів є одним із елементів, що призвели до успіху мови [3]. Розробники мають доступ до сотень офіційних та сторонніх бібліотек, інструментів асемблерування з широкими можливостями налаштування та багатьох фреймворків, розроблених для пришвидшення розробки корпоративних систем.

Граматика Java поступово змінювалася з моменту своїх перших ітерацій, набуваючи нових функцій та конструкцій. Незважаючи на це, розробники все ще можуть виконувати старіші програми на найновіших версіях платформи. Широке використання Java у секторі фінансових послуг, телекомунікаціях, аналітичних системах та високонавантажених серверних програмах, які приділяють велику

увагу керованості та надійності виконання, значною мірою зумовлене цією стабільністю.

1.3.2 Фреймворк Spring

Протягом багатьох років створення корпоративних додатків на базі Java значною мірою отримувало підтримку фреймворку Spring. Його основою є розробка адаптивної архітектури, яка дозволяє створювати складні інформаційні системи без необхідності людського вирішення низькорівневих технічних деталей. Модульність дозволяє розробнику вибирати, які компоненти екосистеми використовуватимуться в даному проекті [4].

В архітектурі Spring об’єднані кілька окремих модулів, що виконують різні функції:

— IoC контейнер. Інфраструктурний контейнер, який керує терміном служби елементів та їх взаємозалежностями, надає програмі попередньо створені об’єкти, а не вимагає їх створення. Цей метод спрощує тестування та значно скорочує обсяг шаблонного коду;

— Spring Boot. Розроблений для пришвидшення запуску додатків. Програма може функціонувати майже без початкового налаштування завдяки попередньо створеним наборам параметрів та автоматичному налаштуванню, що значно скорочує час, необхідний для підготовки середовища;

— Spring Data. Незалежно від того, чи використовується JPA, чи інший метод доступу, цей модуль уніфікує метод обробки даних. Більшість дій з базою даних виконуються автоматично, а структура сховища даних визначається декларативно;

— Spring MVC. Він дозволяє інтегрувати серверні компоненти веб-застосунків, розділяючи отримання запитів, обробку та створення відповідей. Такий метод спрощує масштабованість та пропонує наочну логічну архітектуру;

— Spring Security. Від фільтрації запитів до контролю прав користувачів, це рішення пропонує багаторівневий захист програм. Воно використовується в

сервісах, де дотримання правил безпеки та контроль доступу є критично важливими;

— Spring AOP. Можливо підключати незалежні модулі, які не взаємодіють з бізнес-логікою, для допоміжних функцій, таких як облік викликів або ведення журналу. В результаті код стає менш складним і повторюваним.

Завдяки своїй універсальності, широкій екосистемі та здатності організовувати складні рішення, Spring здобув популярність. Тим не менш, використання цього фреймворку іноді вимагає роботи з досить складними налаштуваннями та заглиблення у значну кількість документації, незважаючи на його численні переваги [5].

Загалом, Spring інтегрує широкий спектр технічних можливостей, але здатність розробника належним чином створити архітектуру та вибрати необхідні модулі визначає, наскільки він буде успішним.

Завдяки своїм характеристикам, що значно спрощують та пришвидшують розробку програмного забезпечення, Spring продовжує залишатися однією з найпопулярніших платформ для створення серверних додатків. Можливість швидко розпочати новий проект є однією з головних переваг. Попередньо налаштовані механізми платформи, автоматизоване управління залежностями та готові інструменти значно скорочують обсяг рутинного коду та час, необхідний для початкової розробки.

Високий ступінь модульності Spring є ключовим аспектом. Широкий спектр бібліотек та підсистем, які можна зібрати відповідно до вимог конкретного додатку, включено до екосистеми фреймворку. Це дозволяє створювати функціональність без додавання сторонніх компонентів до архітектури, включаючи лише модулі, необхідні для виконання певних дій.

Крім того, Spring демонструє високий ступінь гнучкості з точки зору розширення та масштабування системи. Проекти, які потребують обробки вищих навантажень або розробки бізнес-процесів, добре підходять для цієї платформи. Завдяки забезпеченню взаємодії з розподіленими інфраструктурами, ефективного

кешування та використання інших методів оптимізації швидкості, системи на основі Spring зберігають стабільність.

Фреймворк Spring має багато переваг, але є й недоліки. Початкова конфігурація системи є однією з головних проблем. Щоб гарантувати належну роботу базової інфраструктури, зв'язок між компонентами програми, який контролюється контейнером залежностей, має бути добре описаний. Встановлення таких комбінацій може вимагати багато роботи та досвіду, особливо в проектах зі складними архітектурними обмеженнями.

Тісні зв'язки з середовищем Spring створюють ще одну складність. Численні модулі фреймворку покладаються на внутрішні інструменти Spring та методи інверсії керування. Через це перенесення проекту на інший фреймворк або інтеграція альтернативних технологій часто вимагає значних змін. Ця залежність може обмежувати адаптивність та підвищувати вартість модифікації системи для нових технологічних контекстів.

Незважаючи на певні недоліки, екосистема Spring продовжує залишатися одним із найпотужніших інструментів розробки на Java, оскільки вона інтегрує різноманітні функції та значно скорочує час, необхідний для створення веб-систем. Завдяки своїй модульності розробник може створити повноцінну архітектуру застосунку лише з тими частинами, які потрібні для виконання певного завдання.

Spring Web є одним із основних компонентів цієї екосистеми. Він пропонує платформу для розробки серверної частини онлайн-ресурсів, включаючи обробку маршрутів, надсилання та отримання HTTP-запитів та створення REST-сервісів. Натомість платформа надає чіткі інтерфейси, які значно пришвидшують розробку інтернет-застосунків, усуваючи необхідність для розробника мати справу з низькорівневими мережевими операціями.

Контроль точності вхідних даних зазвичай необхідний у складних системах. Spring Web легко взаємодіє з іншими підсистемами, такими як Spring Data або Spring Security, і вже має інструменти попередньої обробки та валідації. Це

дозволяє розробляти логіку лише в рамках Spring Framework та уникати інтеграції інших сторонніх рішень.

Spring Boot, платформа, яка спрощує початок нових проектів та мінімізує кількість параметрів, які розробники повинні налаштовувати вручну, є ще одним важливим модулем. Фреймворк сам вибирає найкращі параметри для програми, встановлює автоматичне підключення до системи керування базами даних, налаштовує обробку помилок та допомагає запобігти конфліктам між різними бібліотеками, замість того, щоб обробляти тисячі файлів конфігурації [6].

Крім того, Spring Boot пропонує попередньо створені конфігурації, щоб застосунок можна було запустити одразу після його створення, що усуває необхідність налаштування додаткової інфраструктури. Цей метод особливо корисний на етапі розробки, коли швидке тестування функціональності є критично важливим.

Взаємодія з даними здійснюється за допомогою модуля Spring Data JPA. Використовуючи інтерфейси та анотації, ви можете вказати доступ до бази даних, не пишучи багато повторюваного коду [7]. Хоча Hibernate є постачальником ORM за замовчуванням, конфігурацію можна змінити або розширити залежно від вимог проекту.

Spring Data JPA підтримує різноманітні методи запитів, включаючи автоматично створені методи, конструктори критеріїв та анотовані запити. Розробнику не потрібно вручну генерувати SQL-запити, оскільки дії представлені на рівні об’єктів, що значно спрощує та робить взаємодію з базою даних безпечнішою.

1.3.3 Мова розмітки HTML

Структура будь-якої веб-сторінки базується на HTML, фундаментальній мові розмітки. Вона визначає розташування елементів інтерфейсу на сайті та те, як браузер повинен їх розуміти. Структура документа, який може містити текстові

блоки, посилання навігації, зображення або інші види контенту, формується певними тегами та атрибутами.

Створення структур веб-документів — одне з багатьох застосувань HTML, фундаментальної технології веб-розробки. Розробник використовує розмітку для визначення логічного порядку представлення інформації, упорядкування письмового контенту, розташування зображень та мультимедійних елементів, а також для керування тим, як сторінка відображається в браузері. Це гарантує, що веб-ресурс відображається однаково на всіх платформах та пристроях.

Мережева навігація — ще одне важливе застосування HTML. Ви можете пов’язати окремі сторінки або зовнішні ресурси в єдину інформаційну структуру, створюючи гіперпосилання за допомогою мови розмітки. Завдяки легкості, з якою користувач може переміщатися між веб-елементами, HTML є невід’ємною частиною сучасної архітектури Інтернету.

1.3.4 Каскадні таблиці стилів

CSS визначає, як повинні виглядати елементи, створені за допомогою HTML або інших мов розмітки. Вона робить процес створення веб-інтерфейсів більш гнучким та керованим, дозволяючи відокремити візуальний вигляд документа від його структури. Стилі сторінки — кольори, шрифти, макет блоків та інші косметичні елементи — обробляються CSS, тоді як HTML служить її основою.

Існує чітка різниця між сторінкою, яка використовує CSS, і сторінкою, яка його не використовує: веб-документ без стилів виглядає як сирий текст з невеликою кількістю форматування, переважно монохромний, та невпорядкованим розміщенням елементів. Такий вигляд часто означає, що стилів немає або що сталася помилка завантаження.

Іншими словами, за наявності набору стилів не потрібно весь час описувати зовнішній вигляд різних компонентів. Це скорочує код і зменшує ризик помилок на додаток до економії часу [8].

Раніше колір, розмір та інші характеристики кожного компонента доводилося вказувати вручну, а елементи стилю потрібно було вставляти безпосередньо в HTML-код. Тепер стилі зберігаються у зовнішніх CSS-файлах. Це дозволяє встановити узгоджені стилістичні рекомендації, які застосовуються до кількох елементів одночасно.

Змінюючи стиль в одному файлі, розробник миттєво змінює всі підключені сторінки, що пришвидшує оновлення дизайну, одночасно зменшуючи обсяг коду. Крім того, централізоване управління стилями значно знижує ймовірність помилок та покращує читабельність і зручність обслуговування HTML-структури [9].

1.3.5 Мова програмування JavaScript

Динамічний характер веб-сторінок зумовлений мовою JavaScript. Завдяки ній веб-сайти можуть робити більше, ніж просто відображати інформацію та зображення у статичному вигляді. Сторінка, найімовірніше, використовує JavaScript, якщо вона реагує на введення користувача, змінює вміст без необхідності оновлення або має інтерактивні компоненти.

Інтерактивні карти, відтворення мультимедіа, 2D та 3D анімація, обробка подій у реальному часі та динамічне оновлення даних — типові приклади функцій, реалізованих цією мовою. Через це JavaScript є важливим компонентом практично всіх онлайн-додатків та вирішальним інструментом у сучасній фронтенд-розробці[10].

1.3.6 Середовище розробки IntelliJ IDEA

IntelliJ IDEA — інтегроване середовище розробки (IDE) для мов програмування, що працюють на Java Virtual Machine. Завдяки інтелектуальному аналізу коду, автоматичному виявленню помилок та допомозі з рефакторингом можливо покращити структуру програми, не впливаючи на її логіку, пришвидшивши загальний процес розробки.

Однією з основних переваг середовища є автоматичний аналіз коду та виявлення будь-яких проблем, що дозволяє розробнику швидко виправляти

недоліки. Як результат, IDE усуває частину організаційного навантаження, звільняючи вас, щоб зосередитися на створенні важливих функцій та виконанні основних цілей проекту.

1.3.7 Система керування базами даних PostgreSQL

Однією з найпопулярніших та найнадійніших систем керування базами даних (СКБД) у світі є PostgreSQL, яка є безкоштовною та має відкритий вихідний код [11]. Завдяки своїй довговічності, адаптивності та відповідності сучасним правилам зберігання даних, вона широко використовується як малими, так і великими підприємствами.

Як об’єктно-реляційна база даних, PostgreSQL поєднує традиційний реляційний метод з більш складними об’єктними методами, які дозволяють розширювати структуру даних та логіку обробки [12].

PostgreSQL не контролюється однією фірмою, на відміну від інших ініціатив з відкритим вихідним кодом, таких як Apache, FreeBSD або MySQL. За розробку системи відповідає багатонаціональна спільнота шанувальників та компаній, які використовують цю СКБД у власних продуктах та зацікавлені в розширенні її можливостей.

Серед ключових можливостей СКБД PostgreSQL варто виділити такі:

— підтримка транзакцій із гарантіями ACID, що забезпечує надійність, цілісність і узгодженість даних навіть у разі збоїв;

— оновлювані представлення (views), які дають змогу користувачам отримувати актуальну інформацію безпосередньо з даних бази в режимі реального часу;

— матеріалізовані представлення, що дають можливість зберігати заздалегідь сформовану копію запиту, пришвидшуючи доступ до складних вибірок;

— механізм тригерів, який автоматично виконує певні дії у відповідь на зміни в таблицях, що спрощує реалізацію логіки контролю та обробки даних.

Загальні характеристики СКБД наведені в таблиці 1.1.

Таблиця 1.1 — Характеристики та обмеження СКБД

Характеристика	**Верхня межа**	**Коментар**
розмір БД	необмежений	
кількість баз даних	4,294,950,911	
зв’язків на базу даних	1,431,650,303	
розмір відношення	32 TB	з стандартним BLCKSZ 8192 байт
рядків на таблицю	обмежено кількістю кортежів, які можуть вміститися на 4 294 967 295 сторінках	
індексів на таблицю	необмежено	обмежений максимальними відношеннями на базу даних
стовпців на індекс	32	можна збільшити шляхом перекомпіляції PostgreSQL
ключі розділів	32	можна збільшити шляхом перекомпіляції PostgreSQL
довжина ідентифікатора	63 байтів	можна збільшити шляхом перекомпіляції PostgreSQL
аргументи функції	100	можна збільшити шляхом перекомпіляції PostgreSQL

Шляхом перекомпіляції можна змінити конфігурацію та підняти продуктивність для власної бази даних.

2 АЛГОРИТМИ СТВОРЕННЯ СУЧАСНОЇ ПЛАТФОРМИ

У цьому розділі розглядаються методи та алгоритми, що використовуються для створення сучасного сервісу автоматизації закупівлі електрообладнанням, проблеми, які вони створюють, та рішення.

2.1 Облікові записи користувачів

Правильне зберігання та обробка даних користувачів є критично важливими в рамках дослідження функціональних підзавдань платформи, оскільки саме ці дані надають користувачам доступ до розширених можливостей системи та персоналізованого контенту. Коли платформі потрібно ідентифікувати нового відвідувача та створити для нього інший обліковий запис, дозвіл стає проблемою.

Профіль користувача створюється під час початкового процесу реєстрації та містить основну особисту інформацію, необхідну для майбутньої взаємодії з системою. Прикладами таких даних є окреме ім’я користувача, пароль доступу, повне ім’я, адреса електронної пошти та номер телефону. Оскільки ці дані використовуються для ідентифікації, автентифікації та гарантування безпеки облікового запису, їх належний збір та перевірка є важливими кроками.

Кожен із вищезгаданих компонентів пов’язаний з ім’ям користувача або логіном, а дані перевіряються на відповідність унікальній ідентифікації. Оскільки ім’я користувача, адреса електронної пошти та номер телефону кожного облікового запису явно відрізняються, користувача слід попередити, якщо він спробує ввести неправильну інформацію. Кожне поле перевіряється на відповідність вимогам виконання шаблону, що включає номер телефону з початковим кодом українських мобільних операторів та ім’я користувача, що складається виключно з латинських літер або цифр. Також підтримується перевірка паролів, електронних адрес та повних імен (принаймні двох слів).

Дані з форми мають бути передані до програмного коду якимось чином після натискання кнопки «Submit». Цей програмний компонент, який включає всі доступні дані в модель, обробляється генератором динамічних шаблонів. Потім контролери отримують останні як об’єкт фреймворку. Немає проблем із поверненням даних, оскільки всі записи зберігаються у форматі «ключ-значення».

Клієнт-серверна програма продовжуватиме працювати у своїй поточній формі та буде марною, якщо дані не будуть збережені належним чином. База даних SQL містить усі необхідні дані.

Ключовим компонентом інформаційної безпеки в сучасних веб-системах є механізм авторизації клієнтів, який дозволяє контрольований доступ до ресурсів, бізнес-логіки та персональних даних. Стандарт OAuth 2.0/2.1, міжнародний протокол делегування доступу, та інфраструктура, що пропонується Spring Security, Spring Authorization Server та Spring OAuth2, слугують основою для авторизації в розробленій системі. Цей метод гарантує масштабованість, уніфікований зв’язок, сумісність з іншими сервісами та дотримання сучасних правил безпеки.

Механізм авторизації під назвою OAuth 2.0 дозволяє одній системі надавати іншій обмежений доступ до ресурсів без необхідності передачі облікових даних користувача. У цьому сенсі авторизація стосується того, що клієнт може робити від імені користувача, а не безпосередньої автентифікації користувача через сторонній застосунок. Оскільки OAuth пропонує логічний розподіл ролей, надійну інфраструктуру токенів та процедури перевірки дозволів, він став стандартом у хмарних та мікросервісних розробках.

Хоча система може здаватися комплексною, концептуально вона складається з 4 ключових компонентів, кожен з яких має власну алгоритмічну роль:

— resource owner — власник ресурсу (користувач), який може надати додатку обмежений доступ до своїх даних;

— client — застосунок, що хоче отримати доступ до захищених ресурсів;

— authorization server — сервер, відповідальний за автентифікацію користувача, видачу токенів доступу та оновлення;

— resource server — сервер, на якому розміщені захищені API та який приймає токени для верифікації доступу.

Ці елементи реалізуються відповідними підпроектами в екосистемі Spring. Створення та підпис токенів обробляються сервером авторизації Spring; належний потік авторизації обробляється клієнтом Spring OAuth2; а перевірка доступу до API здійснюється сервером ресурсів безпеки Spring за допомогою JWT або непрозорих токенів.

Кілька потоків авторизації, що відповідають різним контекстам взаємодії, описані в OAuth 2.0. Архітектура застосунку, ступінь довіри між його компонентами та вимоги безпеки впливають на прийняття певного потоку. Потік коду авторизації, потік облікових даних клієнта та потік токенів оновлення є одними з найважливіших.

Потік кодів авторизації — один з основних процесів авторизації користувачів у браузері. Наразі він рекомендується для будь-яких онлайн-застосунків, де фронтенд та бекенд взаємодіють через захищені канали, і формує основу OAuth 2.1. У цьому потоці користувач спочатку потрапляє на сторінку входу на сервер авторизації, де він надає свою інформацію. Потім сервер надає клієнту тимчасовий номер авторизації. Цей код лише перевіряє, чи пройшов користувач автентифікацію; це не токен доступу. Справжній токен доступу та, за потреби, токен оновлення отримуються клієнтом після надсилання цього коду на сервер авторизації на другому етапі.

Потік токенів оновлення дозволяє автоматично продовжувати сеанси без повторної автентифікації, тоді як потік облікових даних клієнта використовується, коли доступ здійснюється між сервісами без втручання користувача. Як результат, колекція потоків OAuth пропонує гнучкість та можливість змінювати процес авторизації для різних типів користувачів або сервісів.

У системі застосовуються JWT-токени (JSON Web Tokens). Це стиснені структури даних, закодовані у Base64, що містять такі частини у наступному порядку:

1) header, який описує алгоритм підпису та тип токена;

2) payload, де зберігається інформація про клієнта, термін дії, ролі та інші атрибути;

3) signature, що підписується секретним ключем або асиметричною парою ключів (RSA, EC).

JWT дозволяє серверу ресурсів виконувати перевірку токенів локально, без необхідності звертатися до сервера авторизації. Це значно пришвидшує процес авторизації, особливо в мікросервісних системах, де велика кількість сервісів повинна виконати перевірку прав доступу.

Оскільки токени JWT не можна централізовано відкликати, вони зазвичай мають короткий термін дії. Токен оновлення, який має довший термін дії та зберігається на стороні клієнта, використовується для продовження сеансу. Він ніколи не зберігається в API браузера та завжди надсилається лише через захищені канали.

Ключовим компонентом у реалізації логіки OAuth, такої як випуск токенів, управління клієнтами та автентифікація, є сервер авторизації Spring. Він виконує загальні вимоги OpenID Connect та OAuth 2.0, такі як:

— підтримку PKCE (Proof Key for Code Exchange) для підвищення безпеки;

— ведення конфігурацій OAuth-клієнтів;

— генерацію криптографічно підписаних JWT-токенів;

— захист від атак типу replay та CSRF;

— перевірку облікових даних користувача.

Сервер авторизації Spring функціонує як єдина точка прийняття рішень для автентифікації за допомогою алгоритмів. Сервер перевіряє, чи відповідає клієнт своїм дозволим, реєстраційній інформації та дозволеному URI перенаправлення,

коли клієнт надсилає запит на авторизацію. Потім користувача просять пройти автентифікацію, перш ніж операція продовжиться.

Щоб запобігти перехопленню токенів та підробці даних, сервер використовує протокол HTTPS для всіх видів зв’язку. Коли автентифікація користувача успішна, створюються токени, а потім обмінюються ними разом із кодом авторизації. Як результат, кожен рівень має унікальний контроль безпеки.

Бібліотека під назвою Spring OAuth2 Client реалізує поведінку клієнтської програми всередині процесу OAuth. Вона відповідає за створення запитів до сервера авторизації, відстеження стану, обмін кодами на токени та їх подальше використання.

Ключовою її особливістю є дотримання принципів безпеки:

— виконання PKCE-challenge та verifier;

— перевірка redirect-URI;

— автоматичне оновлення access tokens через refresh tokens;

— зберігання access tokens у серверній сесії або в бекенді.

Оскільки клієнт Spring OAuth2 інтегрується зі Spring MVC, користувач сприймає всю процедуру як типовий перехід між сторінками; проте, параметри, токени та зашифровані канали насправді використовуються для зв’язку між компонентами.

Сервер ресурсів, сервер, який надає доступ до API та підтверджує токени доступу для кожного запиту, є останнім компонентом методу авторизації. Він працює на основі декодерів JWT та Spring Security, які автоматично перевіряють токени для:

— строку придатності;

— коректності підпису;

— правильності видавця (issuer) та аудиторії (audience);

— відповідності набору дозволів (scopes).

Запит переходить до бізнес-логіки, яка пропонує надійний контроль доступу до даних користувача, лише після завершення всіх перевірок. У системах, що складаються з кількох незалежних сервісів, цей метод дозволяє централізовано контролювати дозволи та автентифікацію.

Для характеристики всього плану комунікації між учасниками процесу авторизації можна використовувати ряд кроків:

1) користувач отримує доступ до клієнтського застосунку;

2) використовуючи параметри запиту на авторизацію, client перенаправляє користувача на authorization server;

3) користувача проходить автентифікується;

4) клієнт отримує код авторизації від authorization server;

5) через серверну частину client обмінює цей код на токени;

6) authorization server повертає як access token, так і refresh token;

7) client отримує доступ до resource server за допомогою access token;

8) після перевірки токена resource server або схвалює, або відхиляє запит.

Оскільки механізм авторизації є одним із найважливіших компонентів системи, він включає низку сучасних алгоритмів та рекомендацій, таких як вимога HTTPS для шифрування всіх переданих даних, використання PKCE для запобігання перехопленню коду авторизації, використання короткострокових токенів доступу для зменшення ризику компрометації, підтримка довгострокових токенів оновлення з можливістю їх ротації, логічне відділення сервера авторизації від ресурсного для підвищення безпеки та реалізація захисту API через авторизацію ролей та областей. У поєднанні ці процеси створюють повну парадигму безпеки, яка пропонує надійний захист як для кінцевих користувачів, так і для серверних компонентів.

Таким чином, механізм авторизації клієнтів, який базується на технологіях OAuth 2.0 та Spring, пропонує стандартизований, безпечний та масштабований метод контролю доступу. Розділяючи автентифікацію, авторизацію та доступ до

ресурсів між незалежними компонентами, система в цілому підвищує надійність та знижує ризик компрометації даних.

Важливо зазначити, що окреслена стратегія авторизації дозволяє системі легко адаптуватися до змінних вимог та специфікацій безпеки. Подальший розвиток механізмів автентифікації, таких як взаємодія із зовнішніми постачальниками ідентифікації, багатофакторна автентифікація або підтримка нових криптографічних алгоритмів, можливий завдяки модульності архітектури та дотриманню відкритих стандартів.

2.2 Актуалізація сесії

Перевірка облікових даних — це лише один аспект процесу авторизації користувача; зберігання точних та достовірних даних про активного користувача є не менш важливим. Це особливо критично для систем, яким потрібно обробляти великі навантаження та одночасно обслуговувати багато клієнтів, що вимагає спеціального тестування продуктивності. Загалом кажучи, існує два способи організації зберігання даних поточного користувача.

Варто пояснити функцію HTTP-сеансу, перш ніж переходити до його опису. Усі запити відвідувача до сервера супроводжуються набором параметрів, відомих як сеанс. Щоразу, коли браузер взаємодіє із сервером, до браузера надсилається файл cookie, що містить унікальний ідентифікатор сеансу. Після отримання цього ідентифікатора програма знаходить відповідний набір характеристик користувача.

Зберігання поточних сесій безпосередньо в оперативній пам’яті веб-застосунку — це перший спосіб взаємодії з даними. Незважаючи на простоту використання, ця стратегія має серйозні недоліки. Кожен користувач автоматично втрачає свій авторизований статус, якщо JVM перезапускається, оскільки весь вміст кешу оперативної пам’яті видаляється. Планові оновлення сервера, встановлення нових версій програмного забезпечення, непередбачені збої обладнання або численні перезапуски під час активної фази розробки, коли

програма регулярно генерується та тестується, можуть призвести до таких обставин.

Другим суттєвим недоліком локального зберігання сесій у пам’яті є те, що воно по суті несумісне з подальшим переходом до розподіленої архітектури. Важко належним чином розподілити дані між серверами, якщо система має працювати над кількома екземплярами програм одночасно та розподіляти навантаження порівну між ними, просто зберігаючи сесії на кожному окремому вузлі (рис. 2.1). У цьому сценарії, коли балансувальник навантаження перемикається між вузлами, користувач може «втратити» свій статус.

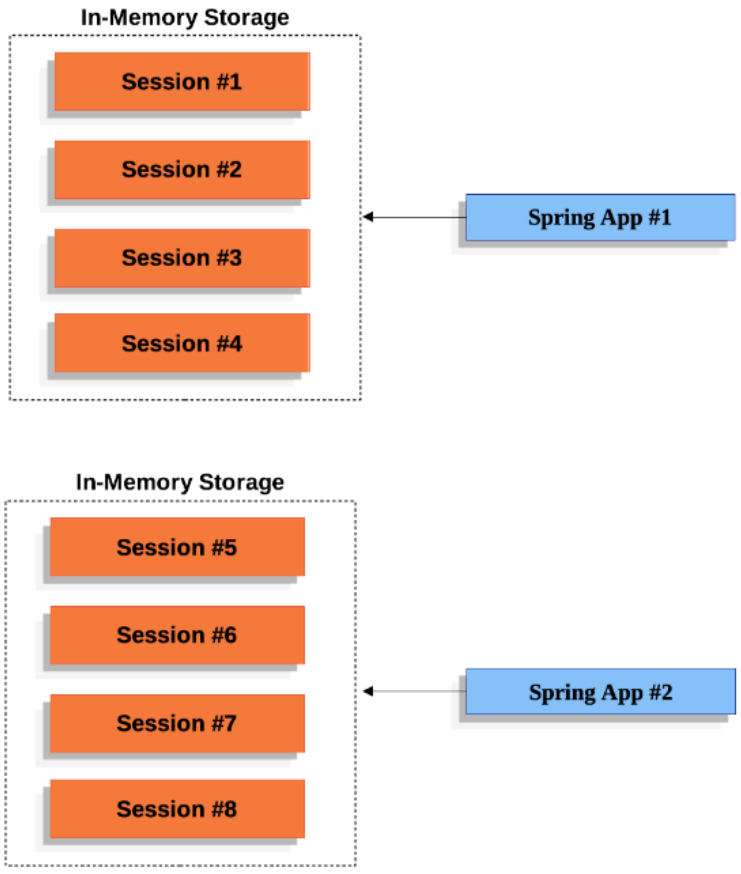

Рисунок 2.1 — Відокремлення пам’яті двох додатків

Spring пропонує готовий механізм під назвою Spring Session, щоб подолати ці недоліки. Він усуває необхідність складних ручних налаштувань і дозволяє

розробнику переносити сховище сесій до централізованого зовнішнього сервісу. Redis, сховище JDBC, Hazelcast, MongoDB та багато інших технологій легко інтегруються зі Spring Session. Цей метод повертає дані в єдиному форматі для кожного екземпляра сервера, зберігаючи всі дані користувача в одному спільному сховищі. Це усуває всі попередні обмеження та дозволяє безболісно масштабувати логіку програми.

На рисунку 2.2 подано спрощену схему взаємодії. Кілька незалежних серверів додатка звертаються до єдиного централізованого сховища, у якому зберігаються всі активні сесії.

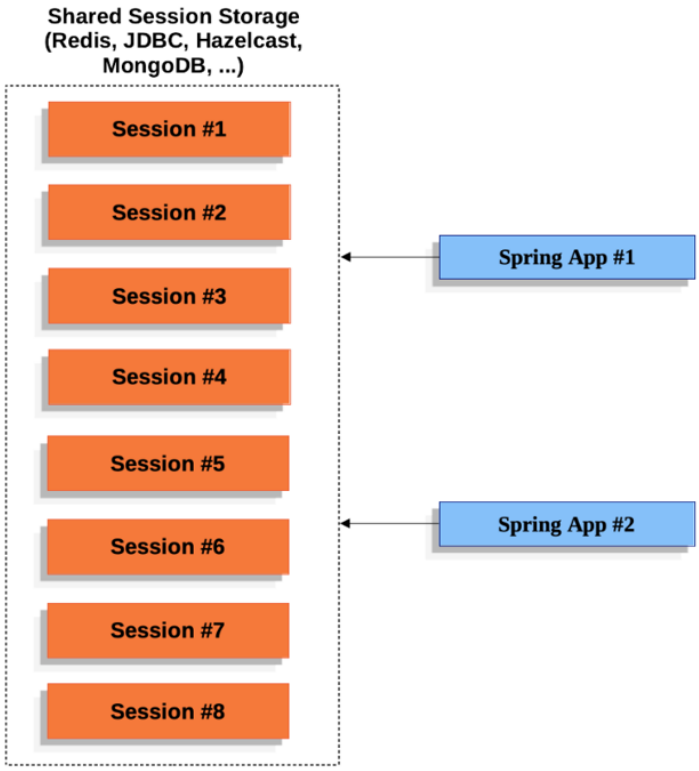

Рисунок 2.2 — Об’єднане сховище для двох додатків

Перевага роботи зі Spring Session полягає також у тому, що незалежно від того, який тип сховища застосовується, розробник працює з уніфікованим API.

2.3 Функціонування кошика

Логіку типового шопінгу можна пов’язати з процесом вибору товарів в інтернет-магазині, але все відбувається в цифровому середовищі.

Користувач бачить створений каталог доступних товарів після відвідування сторінки магазину. Він може отримати більше деталей про певний товар, натиснувши на нього, включаючи його опис, категорію, інформацію про виробника, кількість на складі та поточну ціну.

На сторінці товару користувач може вибрати, скільки одиниць він хоче додати до свого кошика. За потреби будь-який з раніше вибраних товарів можна легко видалити.

На основі кількості замовлених товарів система постійно оновлює загальну вартість та автоматично розраховує суму замовлення. У будь-який час клієнт може перевірити, що знаходиться в його кошику.

2.4 Оплата товарів

Після вибору покупцем набору товарів замовлення необхідно зареєструвати та оплатити. Оскільки обробкою транзакцій займається спеціалізований сервіс LiqPay, не потрібно налаштовувати власну платіжну інфраструктуру.

Надається можливість швидко здійснювати електронні платежі з будь-якого комп’ютера або мобільного пристрою за допомогою LiqPay. Програма пропонує безпечну та практичну взаємодію з фінансовими транзакціями та призначена як для фізичних, так і для юридичних осіб.

2.4.1 Одноразовий платіж

Для того, щоб метод інтеграції працював, користувач повинен спочатку отримати своє замовлення з кошика онлайн-застосунку. Система створює кнопку оплати на основі цієї інформації. Після натискання на неї клієнт перенаправляється на веб-сайт LiqPay, де з’являється унікальна форма оплати. Кошти списуються, і

замовлення вважається оплаченим після успішної перевірки всіх введених даних на стороні сервісу [13].

Після цього користувач миттєво повертається на сайт магазину, де його повідомляють про успішне завершення транзакції. LiqPay одночасно надсилає до нашої системи HTTP POST-запит у форматі JSON, який обробляється сервером. Згенерований об’єкт замовлення вноситься до бази даних після перевірки повідомлення.

Узагальнений алгоритм роботи представлений на рисунку 2.3.

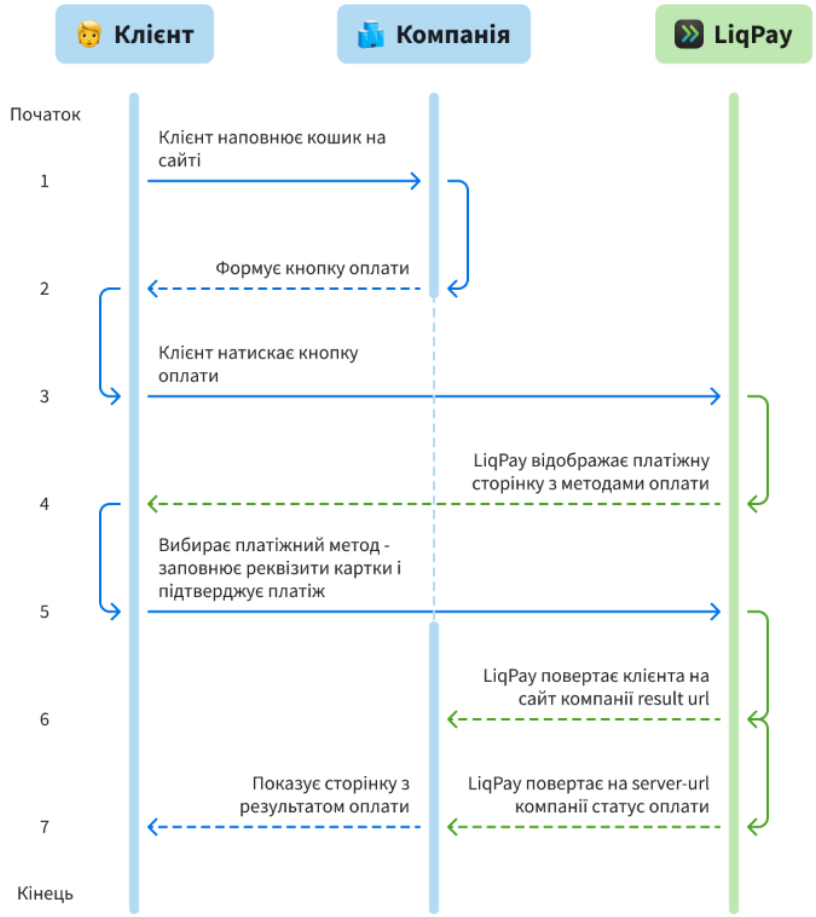

Рисунок 2.3 — Схема роботи checkout

Liqpay також підтримує інші способи взаємодії з додатка, але був обраний метод checkout, що повноцінно задовольняє цілі проекту.

Доступні методи оплати через checkout зображені на рисунку 2.4. Цих методів має бути достатньо для потреб користувачів.

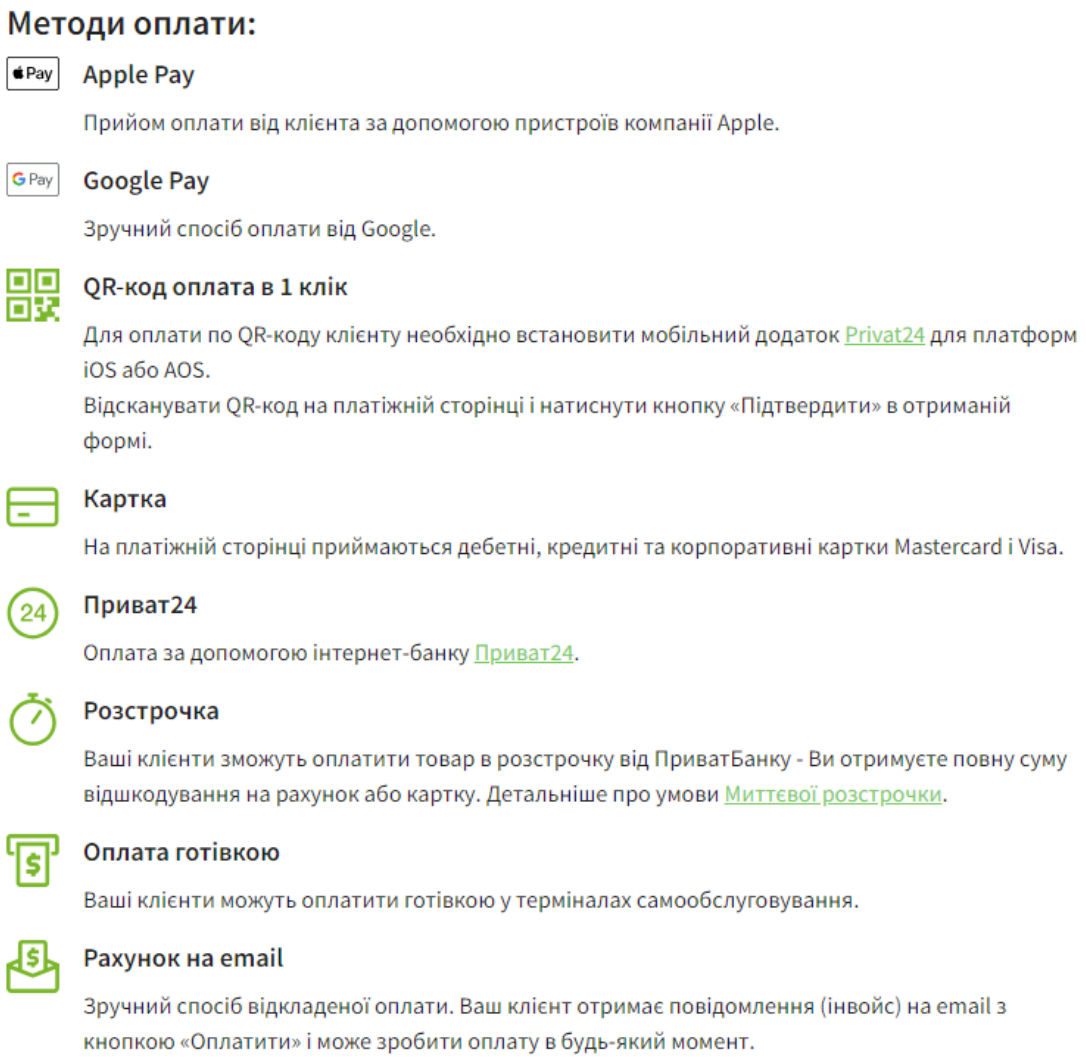

Рисунок 2.4 — Доступні методи оплати

Згідно з загальнодоступними аналітичними даними, Google Pay та Apple Pay наразі обробляють більшість онлайн-покупок. Використання цих способів оплати покращує зручність для споживачів та значно спрощує процедуру оплати електронної комерції.

2.4.2 Періодичне списання коштів

Однією з головних особливостей сучасних електронних платіжних систем є періодичне списання коштів або рекурентні платежі, що дозволяє здійснювати повторювані транзакції без необхідності кожного разу вводити інформацію про платіжну картку. У системі LiqPay для реалізації моделі рекурентних платежів використовуються підписки. Ці підписки формуються під час першого платежу, а потім використовуються для автоматичного зняття коштів через певні проміжки часу. Використовуючи цей метод, компанії можуть налаштувати надійну систему оплати послуг, а клієнти можуть уникнути повторюваних завдань, пов’язаних із здійсненням частих платежів.

Оформлення підписки завжди є першим кроком у процедурі періодичного списання коштів. На веб-сайті компанії клієнт обирає послугу регулярного платежу (наприклад, щомісячне замовлення запчастин, лізинг обладнання або користування послугою). Потім створюється сторінка для введення платіжної інформації. Клієнт підтверджує, що хоче налаштувати регулярний платіж, надаючи номер картки, термін дії та код CVV.

Зі свого боку, компанія надсилає LiqPay запит між серверами з параметрами, що визначають умови підписки, включаючи суму, частоту, валюту, потенційну тривалість та адресу свого сервера-url, на який LiqPay повідомлятиме про статус операцій. Система LiqPay відповідає на запит унікальним ідентифікатором підписки, який потім використовується як ключ для здійснення подальших списань (рис. 2.5).

Компанія зберігає інформацію про підписку у внутрішній базі даних на основі отриманої ідентифікації. Потім клієнт бачить результат початкової транзакції на сторінці, який включає повідомлення про формування підписки, підтвердження успішного попереднього платежу та будь-які потенційні сповіщення щодо технічних несправностей або неточностей.

LiqPay завершує початкове списання коштів після завершення реєстрації підписки. Цей важливий крок підтверджує фінансову стабільність клієнта та

ініціює послугу; підписка вважається повністю активованою після початкового списання коштів.

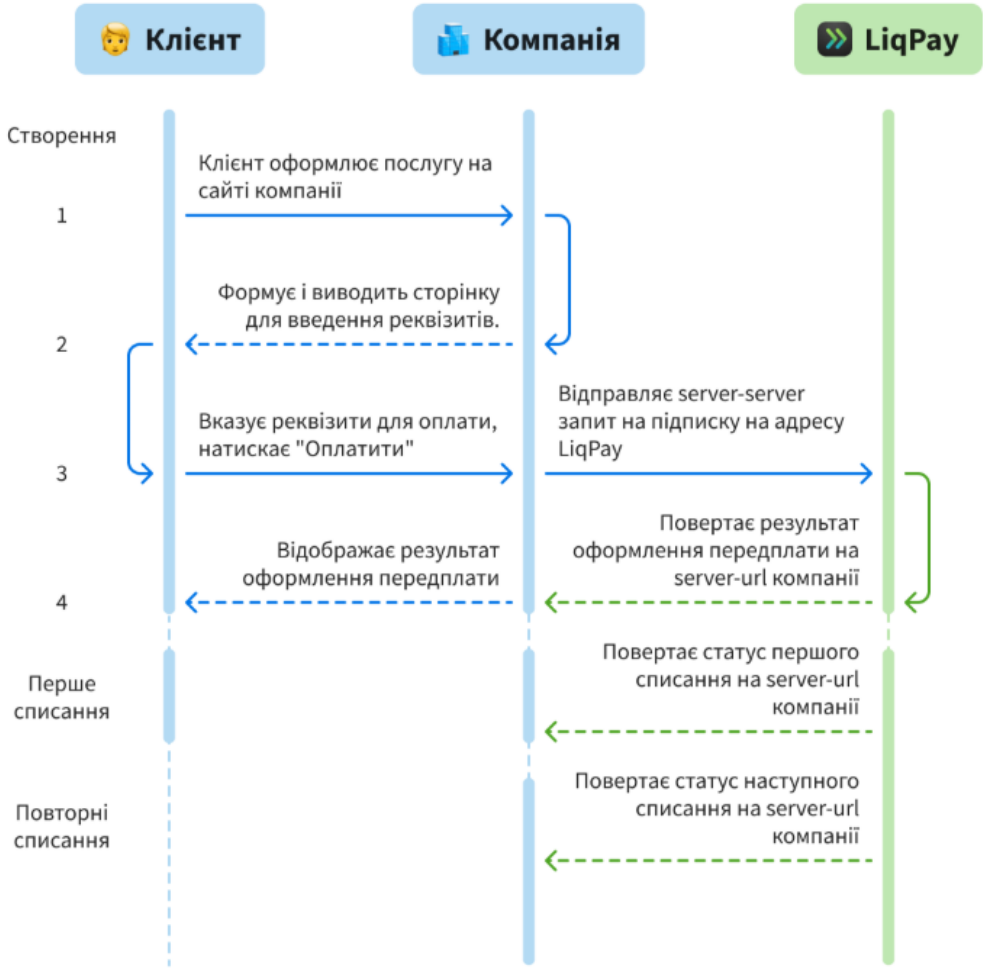

Рисунок 2.5 — Схема роботи підписок

Алгоритм взаємодії складається з багатьох послідовних кроків:

1) бізнес використовує запит між серверами, щоб надати LiqPay дані про підписку;

2) LiqPay оцінює інформацію, перевіряє автентичність картки (включаючи використання алгоритму Luna) та завершує початкове списання коштів;

3) LiqPay надає бізнесу результат операції на адресу сервера;

4) на стороні інтерфейсу клієнт отримує або повідомлення про помилку, або підтвердження успішного платежу.

Залежно від типу бізнес-процесу, корпорація створює замовлення або надає доступ до запитуваної послуги після успішного списання коштів.

LiqPay виконує планові періодичні списання коштів після першого платежу. Вони відбуваються автоматично відповідно до налаштувань, вибраних вами під час створення підписки, яка може бути щомісячною, щотижневою, річною або з іншою заздалегідь визначеною частотою. Повторне введення інформації або підтвердження транзакції від клієнта не вимагається.

З кожним автоматичним списанням пов’язаний наступний цикл:

1) використовуючи збережений токен картки, LiqPay починає списання коштів;

2) коли списання успішне, LiqPay повідомляє бізнес про хід операції за допомогою запиту зворотного виклику, надісланого на адресу сервера;

3) бізнес реєструє транзакцію, поновлює ліміт послуги або розширює доступ до послуги, щоб оновити внутрішні дані системи.

За потреби клієнту надається електронна квитанція або повідомлення щодо успішного списання коштів.

Процедура повністю автоматизована технологією, що зменшує залучення клієнта. Це дозволяє клієнту користуватися послугою без зайвих кроків, а бізнес працює надійно.

Автоматичне списання може бути неможливим за деяких обставин. Серед найпоширеніших причин:

— закінчення строку дії картки;

— недостатня кількість коштів на її балансі;

— блокування або заміна карти банком;

— технічні неполадки у платіжній інфраструктурі.

LiqPay повідомляє бізнес про відповідний статус транзакції в таких ситуаціях. З огляду на це, компанія може:

— через деякий час спробувати списання коштів ще раз;

— повідомити клієнта про необхідність оновлення інформації;

— призупинити підписку на невизначений термін;

— обмежити доступ до сервісу до успішного завершення списання коштів.

Прозора система обробки помилок гарантує стабільність сервісу та запобігає помилковим уявленням користувачів.

Гнучкість управління підписками є однією з ключових характеристик моделі повторюваної оплати [14]. Користувач має можливість відмовитися від автоматичного скасування в будь-який момент. Зазвичай скасування підписки передбачає:

1) перехід до особистого кабінету;

2) вибір поточної підписки;

3) скасувати її.

Потім компанія використовує унікальний ідентифікатор підписки, щоб зробити запит на сервер LiqPay для її деактивації. У відповідь LiqPay запобігає подальшим скасуванням і надає компанії підтвердження успішного скасування.

Після того, як клієнт отримує повідомлення про скасування підписки, автоматичне списання коштів припиняється. Якщо правилами надання послуг не передбачено інше, доступ до послуги часто може бути збережений до закінчення раніше оплаченого періоду. Це гарантує правильне надання послуг і запобігає втраті клієнтом доступу до послуг, за які він уже сплатив.

Отримавши повідомлення про скасування підписки, автоматичні списання коштів клієнта припиняються. Якщо правилами надання послуг не передбачено інше, доступ до послуги часто може бути збережений до закінчення раніше оплаченого періоду. Це гарантує правильне надання послуг і запобігає втраті клієнтом доступу до послуг, за які він вже сплатив.

Користувачеві може знадобитися змінити умови своєї підписки за певних обставин. Це включає, наприклад, зміну пов’язаної платіжної картки, зміну тарифного плану, зміну частоти списань коштів або перехід на інший пакет послуг.

Такі зміни можна внести в системі двома різними способами. Компанія може оновити налаштування без створення нової підписки, якщо зміна не впливає на основну фінансову логіку підписки, особливо якщо клієнт змінює лише частоту списань коштів. Після цього сервіс LiqPay продовжуватиме функціонувати з оновленими даними. Попереднє членство може бути припинене, а нове генерується у разі зміни важливих фінансових характеристик, таких як сума регулярних списань, валюта платежу або платіжна картка. У цьому випадку клієнт ще раз вводить дані картки, забезпечуючи дотримання оновлених умов та підвищуючи безпеку транзакцій.

Безпеці регулярних платежів приділяється особлива увага. LiqPay пропонує високий рівень безпеки під час обробки підписок, оскільки платіжна інформація клієнта не надсилається компанії та не зберігається на її серверах. Натомість використовується унікальний платіжний токен, що дозволяє здійснювати списання коштів без розкриття конфіденційної інформації. Усі учасники процесу використовують зашифровану передачу даних. Крім того, дані картки перевіряються, а базова верифікація виконується за допомогою таких методів, як алгоритм Luna. Токени мають обмежений термін дії, і результат кожної операції можна точно встановити, а потенційних незаконних списань можна уникнути завдяки добре розробленій системі статусу. Навіть при тривалому використанні надійність регулярних платежів гарантується такою ретельною системою.

Як для компаній, так і для кінцевих клієнтів періодичне списання платежів пропонує кілька суттєвих переваг. Перш за все, це стабільний і передбачуваний грошовий потік, який дозволяє підприємствам краще планувати свою діяльність. Автоматизація повторюваних завдань зменшує навантаження на співробітників, знижує ймовірність помилок і покращує процедури обслуговування клієнтів. В результаті підвищується як загальна продуктивність, так і якість обслуговування. Оскільки клієнтам не потрібно щоразу вводити дані своєї картки, вони отримують підвищену зручність, а своєчасне списання коштів гарантує, що вони зможуть продовжувати користуватися послугою. Гнучкість забезпечується можливістю

швидкого скасування або зміни підписки, а ви можете керувати власними витратами завдяки історії транзакцій.

Таким чином, періодичне списання коштів платіжною системою LiqPay є корисним інструментом для автоматизації фінансових операцій. Вона поєднує високий ступінь безпеки даних, надійність для бізнесу та зручність для користувачів. Ця система є адаптивною та безпечною для довгострокового використання завдяки гнучким функціям управління підписками, можливостям редагування та скасування, токенізації та зашифрованим каналам зв’язку. Система здатна забезпечити безперервну роботу послуг, пов’язаних з регулярними платежами, оскільки повідомлення зворотного зв’язку надають їй найактуальнішу інформацію про статус кожної транзакції.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

У цьому розділі розглядається архітектура системи, як вона розділена на функціональні рівні, СУБД, що використовувалася, та сформовані таблиці.

3.1 Патерн MVC

Архітектура додатка збудована за патерном проектування MVC (рис. 3.1), ґрунтуючись на вище згаданому фреймворку Spring [15].

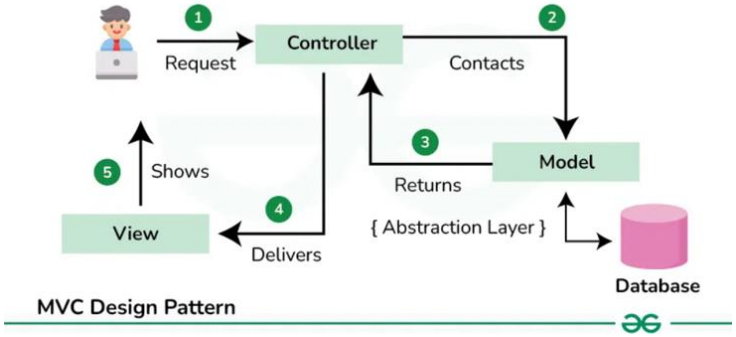

Рисунок 3.1 — Схематичне зображення патерну MVC

Основна концепція архітектурного підходу MVC полягає в тому, щоб поділити програмне забезпечення на дві частини: одну, яка обробляє логіку роботи, та іншу, яка взаємодіє з користувачем.

Основним компонентом системи є модель, яка є важливою частиною загальної архітектури та включає правила обробки даних, функції з бізнес-логікою та визначає структуру предметної області.

Модуль створюється незалежно та відповідає за візуальне представлення даних та отримання введених даних від користувача. Він відповідає за те, як виглядає інтерфейс та як користувач взаємодіє з функціями програми.

Створення незалежного ядра — моделі, яку можна побудувати та протестувати незалежно від решти системи — є метою поділу програми на окремі компоненти. Не впливаючи на інтерфейс чи допоміжні компоненти, цей метод спрощує тестування та гарантує стабільність логіки [16].

Архітектура побудована на багатьох рівнях, кожен з яких має окрему функцію. Принцип єдиної відповідальності, який пропонує сегментувати програму таким чином, щоб кожен модуль мав одну область відповідальності, відповідає цьому. Ролі кожного з цих рівнів будуть ретельно розглянуті далі.

3.1.1 Model

Дані застосунку та бізнес-логіка представлені компонентом Model в архітектурному шаблоні Model-View-Controller (MVC). Він відповідає за підтримку даних застосунку, виконання бізнес-правил та відповіді на запити інформації від інших компонентів

Система містить два пакети з вмістом, пов’язаних з моделлю застосунку. Один з них має відповідну назву, класи зосереджено на рисунку 3.2. Його обов’язки включають реалізацію бізнес-логіки та зберігання сутностей з певними станами та поведінкою.

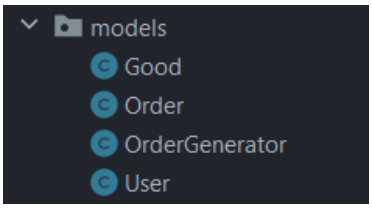

Рисунок 3.2 — Вміст пакету models

Як видно зі структури пакета, він містить чотири основні класи:

— Good: відповідає за зберігання властивостей окремого товару;

— Order: описує характеристики сформованого замовлення;

— OrderGenerator: забезпечує створення нових замовлень;

— User: визначає модель облікового запису користувача, що зберігається у базі даних.

Детальніше розглянемо клас OrderGenerator. Він відповідає за генерацію нових замовлень у системі, як було сказано раніше, проте цей опис занадто широкий. Метод createPaymentFormHtml, який створює HTML-код кнопки оплати, насправді є основним компонентом його функціонування.

Поточний HTTP-сеанс та сума до сплати в гривнях — це два аргументи, які приймає метод. Потім у створений мапу включаються такі аргументи:

— action — тип платіжної операції;

— amount — сума оплати, отримана з параметра методу;

— currency — валюта транзакції (гривня);

— description — текстовий опис платежу;

— order_id — унікальний ідентифікатор замовлення;

— version — актуальна (третя) версія API сервісу LiqPay;

— sandbox — режим роботи (тестовий або бойовий);

— result_url — адреса повернення після успішної оплати;

— server_url — адреса, на яку LiqPay надсилає POST-запит з деталями транзакції для подальшого збереження в базі даних.

Функція cnb_form отримує згенеровану колекцію та повертає HTML-текст заповненої платіжної форми. Потім динамічний шаблонізатор Thymeleaf отримує цей рядок, забезпечуючи коректне відображення кнопки оплати на стороні клієнта.

Офіційна документація платіжної системи LiqPay відповідає алгоритму формування колекції. Для взаємодії з сервісом до проекту було додано бібліотеку з ідентифікатором групи com.liqpay та назвою артефакту liqpay-sdk, яка пропонується на офіційному сайті LiqPay.

Статичні поля класу використовуються для зберігання початкових даних зареєстрованого магазину в системі LiqPay. Якщо підприємець використовує цей проєкт як модель, йому потрібно буде визначити власні налаштування у файлі

application.properties, а саме відкритий та закритий ключі. Щоб модифікувати систему відповідно до власної бази даних та платіжного середовища, це єдині необхідні налаштування [17].

Інший пакет, названий «dao», відповідає за доступ до бази даних, зміну її записів, а також створення або видалення наявних елементів (рис. 3.3). Саме цей пакет забезпечує взаємодію між програмною логікою та фізичним сховищем даних, реалізуючи операції читання та модифікації інформації.

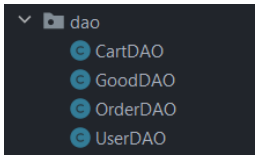

Рисунок 3.3 — Вміст пакету dao

Чотири класи, що надають доступ до бази даних, включені до пакета «dao». Щоб змінити поточний стан системи, кожен клас на цьому рівні взаємодіє з даними, виконуючи такі завдання, як додавання, видалення або оновлення записів.

Особливої уваги заслуговує клас UserDAO, який відповідає за алгоритмізацію процедур перевірки форм користувачів. Зокрема, було впроваджено метод підтвердження інформації, надісланої у реєстраційній формі; аналогічна стратегія використовується для перевірки входу.

Метод працює таким чином, що спочатку отримує дані поточного користувача з активного http-з’єднання, а потім перевіряє дані на наявність серйозних конфліктів, таких як дублікати облікових даних для входу або адрес електронної пошти.

Відповідність призначеним форматам введення та шаблонам перевіряється повторно лише після того, як буде підтверджено, що жоден користувач у системі не має подібних полів.

3.1.2 View

Представлення відповідає за спілкування з користувачем; воно приймає дані, які вводить користувач, і використовує контролер для відображення інформації, яку надіслала модель. Контролер — єдиний спосіб доступу до даних; немає прямого зв'язку між представленням і моделлю.

Кожен екран інтерфейсу реалізовано як HTML-файл, а його вміст створюється динамічно з використанням даних, що надаються об'єктом Model, який є компонентом фреймворку Spring. Насправді цей об'єкт служить контейнером, до якого контролер додає параметри, необхідні для додаткового рендерингу, використовуючи шаблонний механізм Thymeleaf. Цей метод також використовується для зворотної передачі даних з HTML-форм [18].

На рисунку 3.4 показано всі файли перегляду, які організовані в підкаталоги на основі логіки переходу користувачів застосунку. Призначення основних сторінок буде обговорено далі.

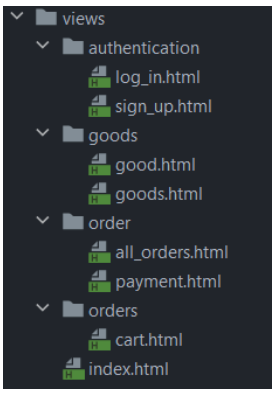

Рисунок 3.4 — Вміст каталогу views

Каталоги, пов'язані з оформленням замовлень, містять три основні представлення: «all_orders» відображає хронологію попередніх успішних замовлень користувача, «payment» — сторінка успішної оплати після перенаправлення сервісу «liqpay», «cart» — сторінка поточного стану кошика з можливістю його очистити чи олатити обрані товари.

3.1.3 Controller

Важливо розуміти, як Spring призначає та керує завданнями у веб-застосунку. DispatcherServlet, основний «диспетчер» системи, є ключовим компонентом обробки HTTP-запитів. Кожен клієнтський запит спочатку проходить через серію фільтрів, які можуть виконувати завдання попередньої обробки, такі як ведення журналу, автентифікація або розбір інформації запиту. Потім фільтри надсилають запит до DispatcherServlet.

DispatcherServlet вирішує, який контролер слід додатково обробити після розбору URL-адреси та аргументів запиту. Цей метод спрощує масштабування та підтримку застосунку, пропонуючи чітку маршрутизацію.

Контролер служить мостом між представленням та моделлю. Після того, як користувач надає дані (наприклад, через форму або параметри запиту), він: ідентифікує представлення для повернення користувачеві, перевіряє вхідні дані, викликає відповідні елементи моделі для зміни стану системи та надсилає необхідні атрибути до цього представлення.

В результаті, контролер гарантує узгодженість даних між моделлю та інтерфейсом користувача та організовує логіку взаємодії.

На рисунку 3.5 показано всі системні контролери. Кожен з них відповідає за свою сферу функціональності, залежно від своєї ролі та важливості в логіці веб-застосунку.

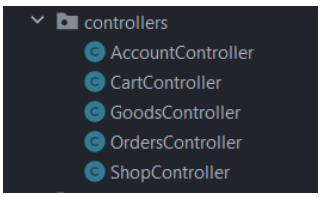

Рисунок 3.5 — Вміст пакету controllers

Найпростіший у списку ShopController. Він просто повертає головну сторінку веб-додатку.

GoodsController, який має два основні методи, є наступним за складністю. У той час як функція showById() створює сторінку з вичерпною інформацією про певний продукт, такою як його характеристики, ціна та наявність на складі, метод showAll() відповідає за надання представлення всього асортименту продуктів.

Усі представлення, що стосуються розміщення та перегляду замовлень, обробляються OrdersController. Це відповідає його основній функціональній меті. PaymentAnalyze(), який обробляє результати платежів, отриманих від платіжної системи LiqPay, є найважливішим методом цього контролера.

Метод paymentAnalyze() очікує два аргументи:

— data — рядок JSON, який вже закодовано в Base64;

— signature — унікальний підпис, що використовується для підтвердження легітимності запиту.

Під час виконання процедура:

1) декодує параметр даних Base64;

2) використовуючи ідентифікатор групи org.json та артефакт json, допоміжна бібліотека аналізує об’єкт JSON;

3) зчитує необхідні параметри (такі як сума платежу, ідентифікатор замовлення, статус транзакції тощо) з отриманого об’єкта JSON;

4) створює новий екземпляр об’єкта Order з пакета моделі, використовуючи ці дані;

5) зберігає створене замовлення в серіалізованому вигляді в базі даних як успішно оплачене.

Функція paymentAnalyze() забезпечує цілісність та достовірність отриманих даних, виконуючи додаткові перевірки безпеки на додаток до первинної обробки платіжних даних. Після розшифрування даних контролер використовує секретний ключ магазину для створення власного контрольного підпису, який потім перевіряє за допомогою параметра підпису, наданого сервісом LiqPay.

Операція ігнорується або підозрілі дані записуються для додаткового аналізу, якщо підписи не збігаються, і в такому разі запит вважається автентичним і може

бути продовжений. Цей метод гарантує стійкість системи до «підроблених запитів» та захищає історію замовлень від зовнішнього втручання [19].

Приклад дешифрованої колекції з json вмістом зображено на рисунку 3.6.


```
{
    "payment_id":2460231630, "action":"pay",
    "status":"sandbox", "version":3,
    "type":"buy", "paytype":"card",
    "public_key":"sandbox_i48897120598",
    "acq_id":414963, "order_id":"-99979", "liqpay_order_id":"372OO6XK1715333166936842",
    "description":"Оплата обраних товарів",
    "sender_first_name":"Name", "sender_last_name":"Surname",
    "sender_card_mask2":"424242*42",
    "sender_card_bank":"Test",
    "sender_card_type":"visa",
    "sender_card_country":804,
    "ip":"194.55.90.33",
    "amount":1360.74,
    "currency":"UAH",
    "sender_commission":0.0,
    "receiver_commission":20.41,
    "agent_commission":0.0,
    "amount_debit":1360.74,
    "amount_credit":1360.74,
    "commission_debit":0.0,
    "commission_credit":20.41,
    "currency_debit":"UAH",
    "currency_credit":"UAH",
    "sender_bonus":0.0,
    "amount_bonus":0.0,
    "mpi_eci":"7",
    "is_3ds":false,
    "language":"uk",
    "create_date":1715333166940,
    "end_date":1715333167092,
    "transaction_id":2460231630
}
```


Рисунок 3.6 — Дані замовлення у json форматі

Кожна властивість є критично важливою в реальних ситуаціях обробки, і кожне значення залежить від даних платника, виду та часу платежу, мережевої інформації пристрою тощо.

3.2 Конфігурація фреймворку Spring

Контейнер IoC створює внутрішній контекст, де кожен об’єкт існує як попередньо створений екземпляр класу після аналізу всіх доступних компонентів,

анотованих відповідними анотаціями під час налаштування програми. Завдяки цьому методу система виграє від прогнозованої ініціалізації необхідних компонентів та централізованого управління залежностями [20]. Протягом життєвого циклу програми створюється лише один екземпляр контролерів та компонентів рівня доступу до даних, оскільки вони функціонують в режимі singleton (рис. 3.7). Цей метод гарантує однакову поведінку для всіх компонентів системи, які взаємодіють з цими службами, одночасно знижуючи накладні витрати на створення об’єктів.

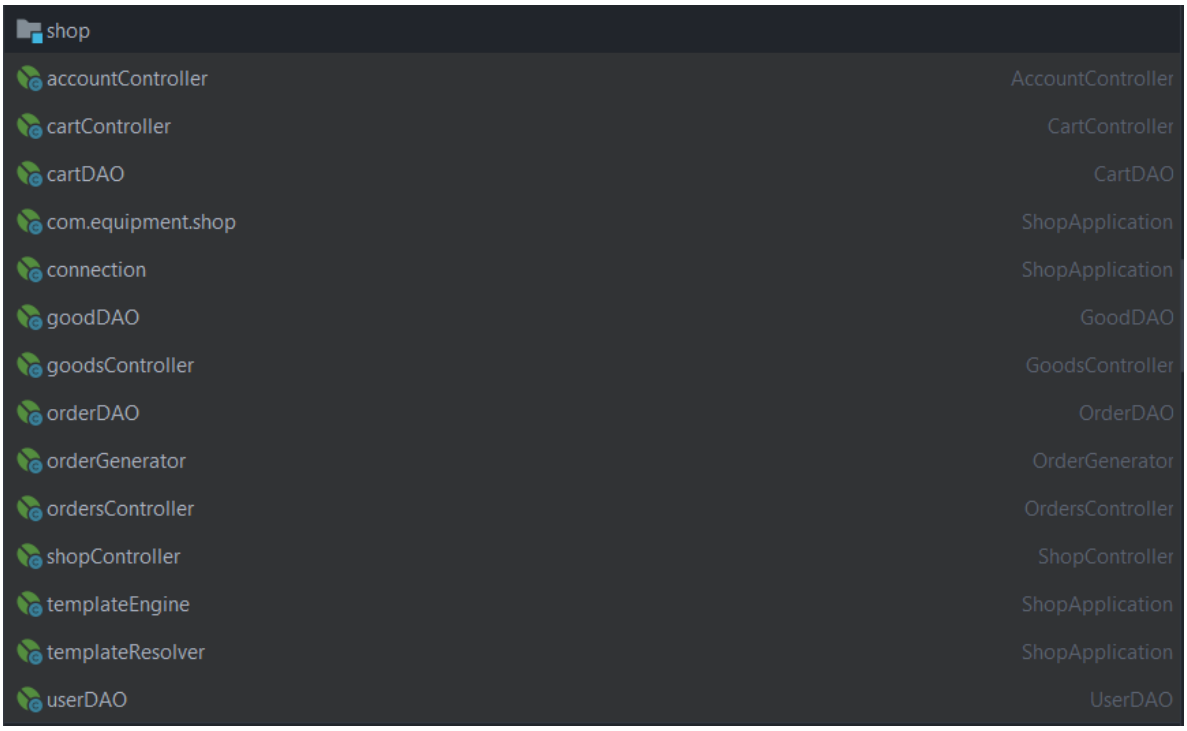

Рисунок 3.7 — Вихідний контекст компонентів додатку

Компоненти отримуються з контейнера IoC під час обробки запиту. Основний клас застосунку ShopApplication містить інші об’єкти зі значним функціональним значенням.

3.3 Опис бази даних

Для налаштування постійного сховища даних проекту було використано систему керування базами даних PostgreSQL. Для візуального відображення структури таблиці було використано офіційні інструменти, що входять до комплекту дистрибутиву, зокрема графічне програмне забезпечення pgAdmin [21].

Спочатку розглянемо таблиці spring_session та spring_session_attributes, які відображені на рисунках 3.8 та 3.9. Після підключення модуля Spring Session, який замінює реалізацію, що відповідає специфікації Jakarta EE, для стандартного сховища сесій HTTP, ці структури генеруються автоматично.

	primary_id [PK] character	session_id character	creation_time bigint	last_access_time bigint	max_inactive_interval integer	expiry_time bigint	principal_name character varying (100)
1	a120b882-82f3-403...	c3f8b7e6-a56e-4b8b-8f1...	1716130319495	1716130325820	60000	1716190325820	[null]

Рисунок 3.8 — Таблиця spring_session в БД

Вміст цих таблиць оновлюється автоматично: при створенні нових сесій у них додаються відповідні записи, під час зміни атрибутів сесії — оновлюються пов’язані значення, а за завершення терміну дії або примусовому завершенню сесії — застарілі записи видаляються.

	session_primary_id [PK] character	attribute_name [PK] character varying (200)	attribute_bytes bytea
1	a120b882-82f3-403c-b4da-e3bc741c793e	currentUser	[binary data]

Рисунок 3.9 — Таблиця spring_session_attributes в БД

Без подальшої участі розробника фреймворк забезпечує належне зберігання характеристик сесій користувачів у базі даних, незалежно генеруючи необхідні таблиці та підтримуючи їхню актуальність [22].

Ім’я та значення атрибута, серіалізовані у вигляді байтового масиву, включені до таблиці атрибутів. Ідентифікатор кожного запису, який служить зовнішнім ключем, пов’язує його з певним сеансом. Як результат, ця таблиця безпосередньо залежить від основної таблиці сеансів; коли застарілий або неактуальний сеанс видаляється, всі пов’язані з ним характеристики миттєво видаляються.

Таблиця users, показана на рисунку 3.10, містить опис поточних даних користувача в системі. Включено унікальний ідентифікатор користувача, ім’я користувача, пароль для входу, адресу електронної пошти, повне ім’я, серіалізований об’єкт кошика та номер мобільного телефону. Як автентифікація користувача в системі, так і створення HTTP-сеансу залежать від цієї інформації.

У таблиці зберігаються саме ті поля, які користувач вводить у відповідну форму під час реєстрації [23]. Щоб запобігти конфліктам або неналежним вставкам у таблицю під час створення нових записів, кожне поле перевіряється на відповідність заздалегідь визначеним шаблонам.

Усі наведені дані в поточній та інших таблицях є тестовими та не будуть задіяні у реальній симуляції процесів.

	user_id [PK] integer	username text	password text	email text	full_name text	cart bytea	phone_number text
1	1	seredaVO	12345678	sv@gmail.com	Середа Владислав Олександрович	[null]	+380669999999
2	2	vladyslav	12345678	vl@gmail.com	Середа Владислав Олександрович	[null]	+380969999999
3	3	seredaOO	12345678	soo@gmail.com	Середа Олександр	[null]	+380639999999

Рисунок 3.10 — Таблиця users в БД

У поточній версії проекту таблиця атрибутів може містити дані про останній переглянутий продукт, сповіщення щодо проблем автентифікації та серіалізований об’єкт користувача з назвою атрибута currentUser. Як результат, ця база даних служить гнучким сховищем для станів користувачів, що дозволяє системі відтворювати контекст взаємодії в пізніших запитах.

Усі належним чином оплачені замовлення, які були підтверджені та автентифіковані сервісом LiqPay, зберігаються в останній, але не менш значущій, системній таблиці замовлення [24]. Унікальний ідентифікатор замовлення, сума транзакції, дата та час оплати, спосіб оплати, ідентифікатор користувача, якому належить замовлення, та інші метадані, зібрані із запиту зворотного виклику після завершення транзакції, включені до структури таблиці, яка показана на рисунку 3.11. Це дозволяє створювати історію замовлень, проводити подальший аудит та представляти інформацію в індивідуальному обліковому записі користувача.

	order_id [PK] integer	user_id integer	information bytea
1	1	1	[binary data]
2	2	1	[binary data]
3	3	3	[binary data]
4	4	1	[binary data]
5	5	3	[binary data]
6	6	1	[binary data]
7	7	1	[binary data]
8	8	1	[binary data]

Рисунок 3.11 — Таблиця orders в БД

Ідея зберігання замовлень полягає у використанні унікального ідентифікатора для пов’язування кожного запису з певним зареєстрованим користувачем. Це встановлює зв’язок «один до багатьох», в якому один користувач може розмістити кілька замовлень, кожне з яких безпомилково пов’язане з його власником.

На рисунку 3.12 показано загальну структуру бази даних. Ви можете оцінити архітектуру системи та взаємодію між сутностями, переглянувши діаграму, яка відображає всі таблиці та їх асоціації, первинні та зовнішні ключі, логічний табличний простір та іншу інформацію.

Таке розділення даних забезпечує ефективне керування інформацією та полегшує виконання запитів до бази. Наприклад, можна швидко отримати всю історію замовлень конкретного користувача або проаналізувати активність користувачів за певний період. Крім того, завдяки чіткій структурі таблиць та встановленим зовнішнім ключам забезпечується цілісність даних, що критично важливо для коректної роботи електронного магазину та подальшого формування звітності.

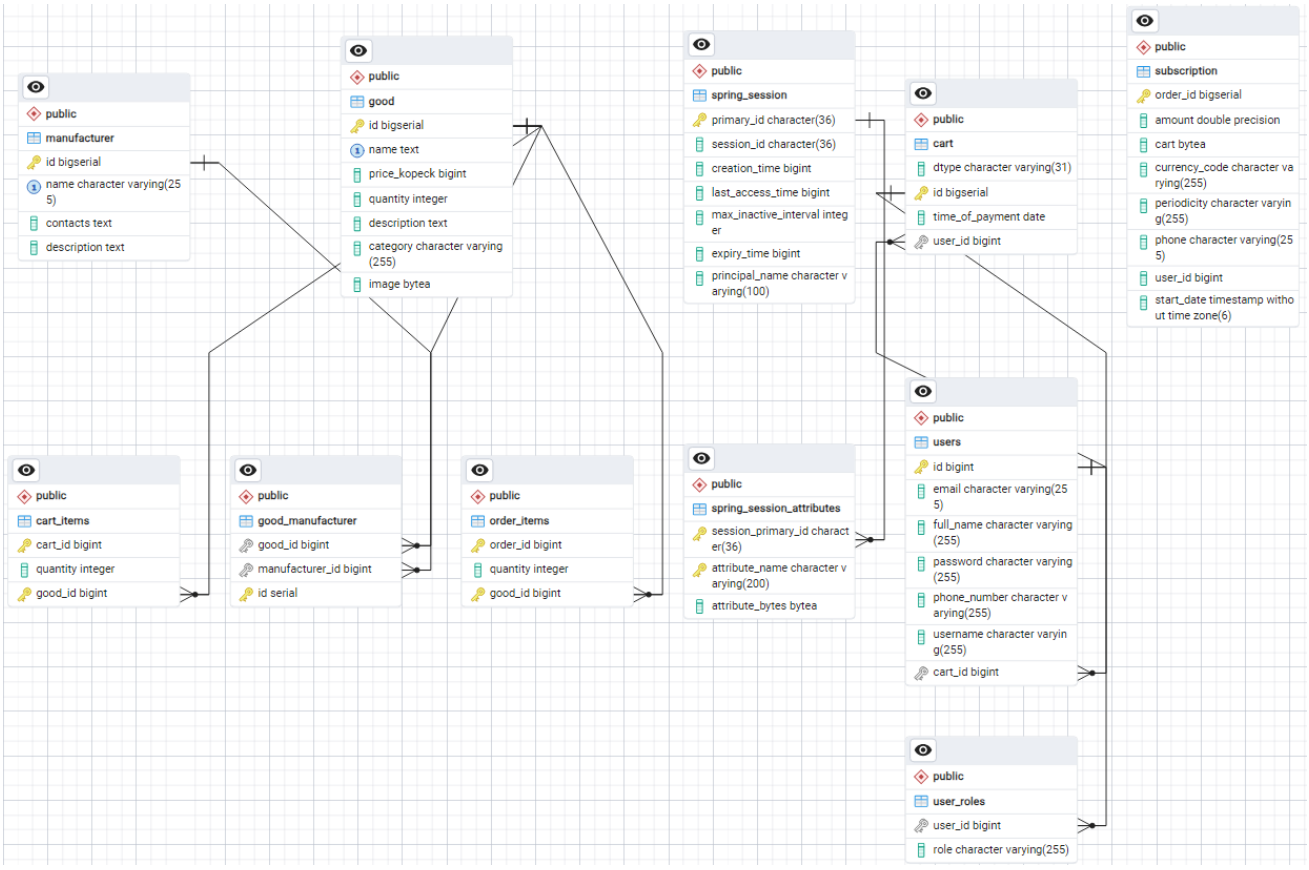

Рисунок 3.12 — Діаграма ERD бази даних

За потреби, систему поточних зв’язків можна покращити, наприклад, додавши посаду адміністратора. Такий крок розширить можливості проекту, але водночас вимагатиме складнішої логіки для заповнення та регулювання даних у репозиторії.

Для зберігання списку електротехнічної продукції використовуються чотири основні таблиці:

— Goods: містить характеристики, спільні для всіх лінійок продуктів;

— проміжна таблиця goods_manufacturers створює зв’язок багато-до-багатьох між сутностями «товари» та «виробники»;

— Manufacturers: список доступних у системі виробників;

— Categories: список можливих категорій продукції.

Найкращий підхід полягає у використанні двох окремих таблиць для представлення сутностей та однієї проміжної таблиці, яка гарантує їхню залежність, оскільки один продукт може вироблятися кількома виробниками, а один виробник може охоплювати широкий спектр товарів у своєму асортименті.

3.4 Додаткові відомості

Більшість сучасних програмних систем створені на основі об’єктно-орієнтованого підходу (ООП). Модель, властива йому, дозволяє природним чином імітувати атрибути та поведінку реальних речей. Важливо зазначити, що такі елементи не існують ізольовано — вони взаємодіють один з одним, створюючи логічну структуру, спрямовану на виконання цілей програми.

Стандартна бібліотека будь-якої мови програмування пропонує широкий спектр готових інструментів, проте реальні завдання настільки різноманітні, що її можливостей іноді недостатньо. В результаті розробники повинні писати власні розширення, які можуть бути окремими класами, колекціями модулів або цілими бібліотеками, що реалізують певні функції, необхідні для їхнього проекту.

3.4.1 Діаграма класів

Структура програмної системи та зв’язки між її частинами зображені на діаграмі класів. Така діаграма вважається найкращою практикою у великих програмних проектах, оскільки вона спрощує подальший розвиток програми,

полегшує розуміння її внутрішньої структури та пришвидшує процес інтеграції нових розробників. Діаграма класів показана на рисунках 3.13 та 3.14 відповідно в контексті цієї роботи.

Під час розробки ви можете додати ще один рівень абстракції, створюючи власні класи. Це робить програму більш адаптивною та готовою до майбутнього зростання, дозволяючи точно описувати роботу системи, приховувати зайву складність та запобігати повторенню логіки [25].

Ім’я класу розташоване у верхній частині прямокутника в структурі діаграми UML. У центральній частині перераховані атрибути (поля класу), які позначені малою літерою. Методи класу, що визначають поведінку, розташовані внизу контейнера, вирівняні по лівому краю та написані малими літерами.

Повна діаграма розділена на два окремі малюнки для зручності перегляду, що дозволяє вам глибше розглянути кожну частину, не перевантажуючись деталями.

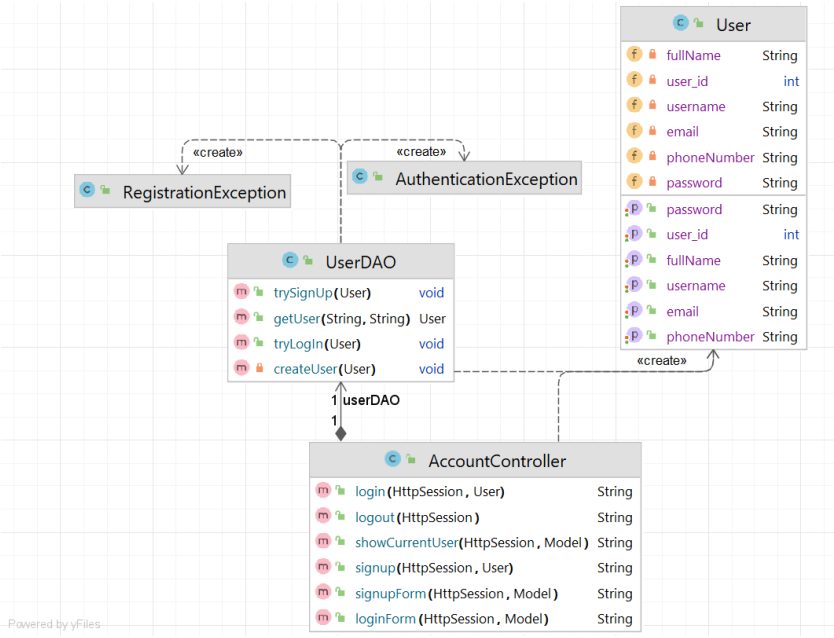

Рисунок 3.13 — Перша частина діаграми класів

Оскільки деякі класи виходять за рамки традиційної структури MVC, найкраще розглядати їх окремо. Початкове налаштування веб-рівня застосунку

обробляється класом MySpringMvcDispatcherServletInitializer. Він визначає класи конфігурації контексту застосунку, ініціалізує властивості фільтра та вирішує, як система обробляє HTTP-запити. Він визначає правила маршрутизації, кореляцію між сервлетами та інші параметри, необхідні для належної роботи механізму DispatcherServlet.

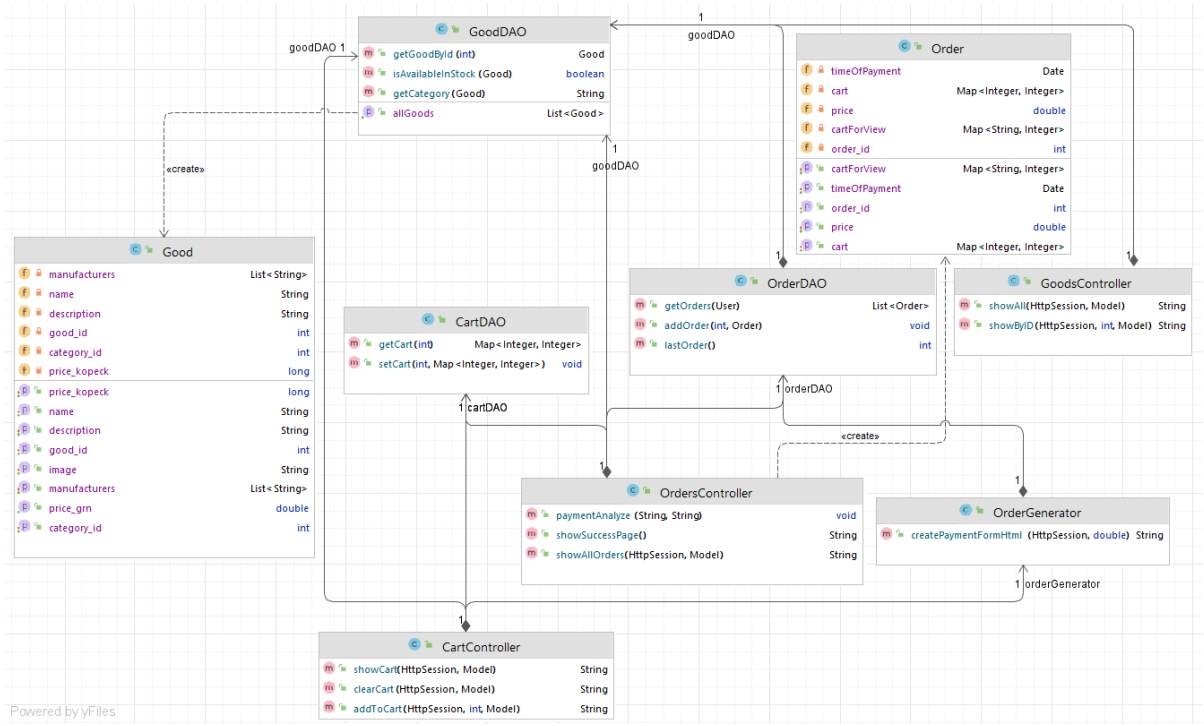

Рисунок 3.14 — Друга частина діаграми класів

Клас ShopApplication слугує основною точкою входу застосунку. Окрім запуску контейнера Spring, він містить додаткову інформацію про конфігурацію, а саме про налаштування генератора динамічних шаблонів Thymeleaf, підключення до бази даних та ініціалізацію допоміжних компонентів, необхідних для функціонування системи. В результаті ShopApplication слугує центром конфігурації, який інтегрує кілька налаштувань проєкту та гарантує їх правильне завантаження під час запуску.

Діаграми класів роблять зв’язки між частинами та загальною структурою системи більш очевидними. Вони роблять MVC-дизайн дуже очевидним, особливо те, як модель, презентація та контролери логічно мають різні обов’язки. Основна частина зв’язків наданих елементів стосується асоціації та її підвидів — агрегації та композиції. Це свідчить про високу структуру коду та добре виконану взаємодію компонентів, що робить систему легко керованою та масштабованою.

3.4.2 Кінцеві точки

Усі актуальні кінцеві точки веб-застосунку зібрано та структуровано на рисунку 3.15. Вони наочно демонструють логіку взаємодії клієнта з API та межі відповідальності кожного контролера.

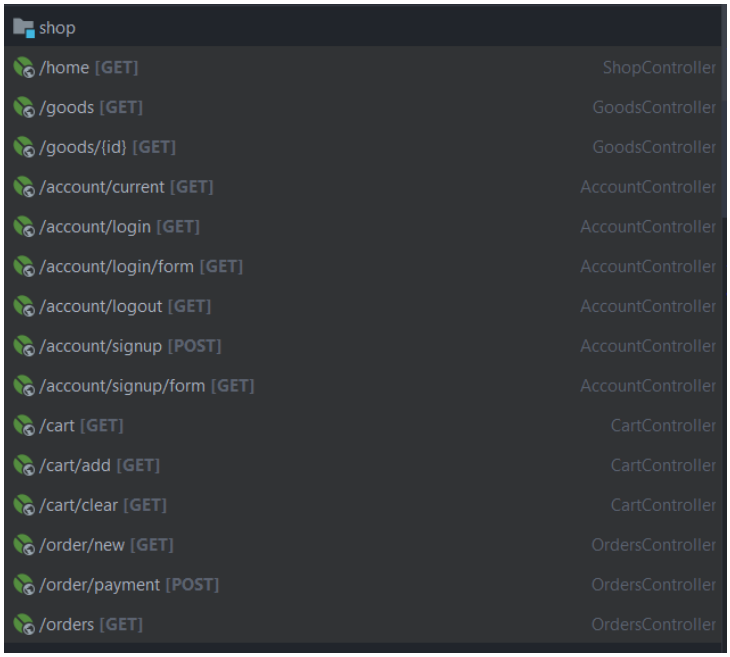

Рисунок 3.15 — Кінцеві точки системи

HTTP-метод визначає дозволені взаємодії з ресурсом (наприклад, GET, POST, PUT, DELETE), тоді як кінцева точка (endpoint) описує конкретний шлях доступу до цього ресурсу та правила його отримання або зміни. Один і той самий

ресурс зазвичай має декілька кінцевих точок, кожна з яких повертає різний обсяг інформації або виконує певну дію, що відрізняється шляхом та HTTP-методом.

Опис кінцевої точки, як правило, стислий і нагадує узагальнений опис функціоналу, який вона виконує. Важливо, що кінцева точка не містить у собі базового маршруту, спільного для всієї групи маршрутів контролера; вона визначає лише кінцеву URL-адресу, за якою доступний ресурс або операція [26].

Оскільки кінцеві точки API визначають взаємодію компонентів та зовнішніх служб, вони мають вирішальне значення для розробки та інтеграції програмних систем. Для пояснення їхньої важливості можна використовувати кілька ключових факторів:

— інтеграція та комунікація. Кінцеві точки пропонують визначений метод передачі даних між різними компонентами системи або зовнішніми службами;

— адаптивність та модульність. Чітко визначені кінцеві точки спрощують додавання нових функцій до системи, не втручаючись у поточні операції;

— сумісність та контроль версій. За допомогою API ви можете зберегти функціональність застарілих клієнтів, підтримуючи при цьому численні версії інтерфейсу;

— розподіл відповідальності. Оскільки кожна кінцева точка виконує окрему функцію, розробка, тестування та підтримка системи спрощуються [27].

Шляхи організовано відповідно до принципів RESTful-проектування, а побудований застосунок здебільшого використовує HTTP-запити GET та POST. Це пояснюється тим, що взаємодія з користувачами включає відвідування різних сайтів, вивчення товарів та внесення змін до бази даних шляхом додавання нових записів, особливо під час здійснення покупок або реєстрації користувачів.

4 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

У цьому розділі розглядається веб-застосунку, потенційні можливості розробки, запропоновані дії та взаємодія користувача з програмною системою.

4.1 Автентифікація користувача

Усі HTTP-запити до веб-сервера надходять з доменного імені electropoint.hopto.org, де встановлено систему. Сторінка, до якої можна отримати доступ через маршрут «-home», служить початковою точкою для взаємодії користувача з програмою. З цього моменту користувач може отримати доступ до додаткових функцій веб-сайту. На рисунку 4.1 зображено початкову структуру логіки доступу та навігації.

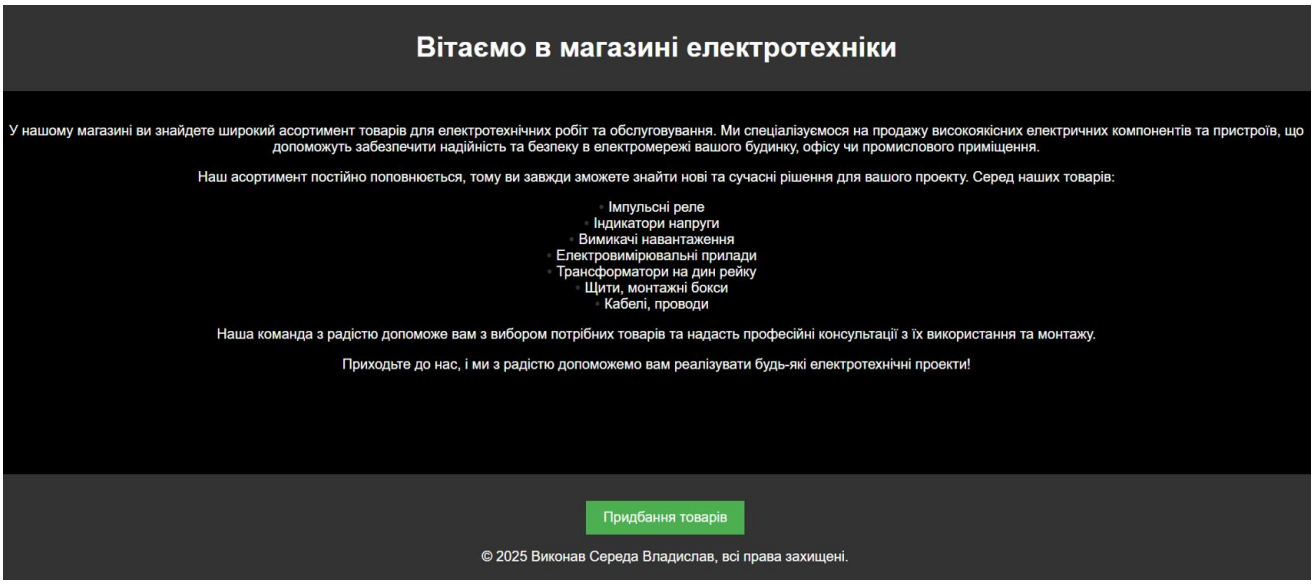

Рисунок 4.1 — Гостьова сторінка

Після натискання кнопки «Придбання товарів» відвідувач миттєво переходить на сторінку входу. Користувач повинен створити новий обліковий

запис, якщо він входить у систему вперше (рис. 4.2). Якщо ні, він може використовувати свої поточні дані для входу (рис. 4.3).

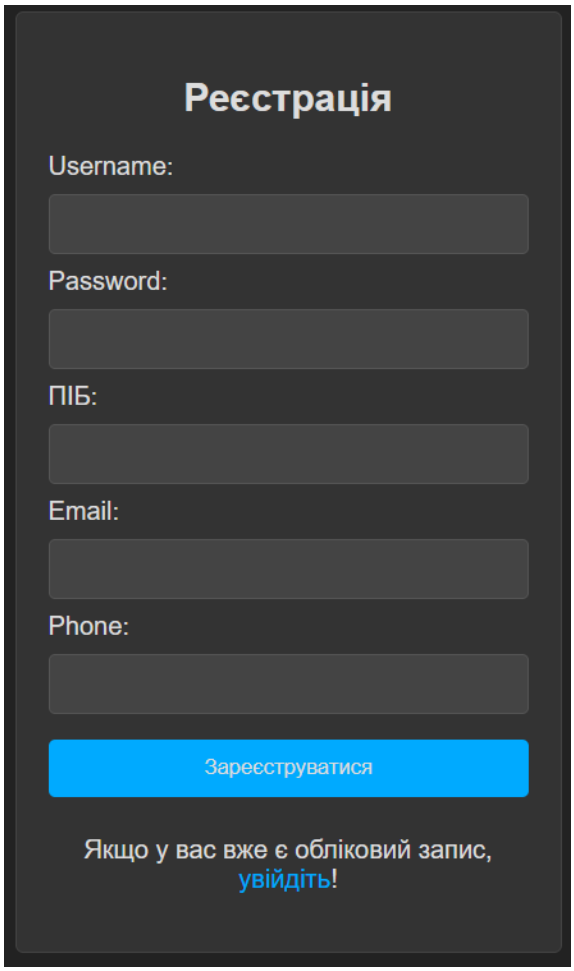

Рисунок 4.2 — Функціонал реєстрації

У разі раніше створеного аккаунту користувачу потрібно ввести справні дані для процесу автентифікації. В разі неуспішності гостя також буде повідомлено.

Кожне значення, яке користувач вводить у форму під час реєстрації, надсилається до об’єкта Model фреймворку Spring. Надалі всі параметри нового профілю ретельно оцінюються на стороні сервера програми.

Процес підтвердження введених даних складається з кількох кроків. Спочатку система перевіряє, чи зареєстровано користувача в базі даних з таким самим іменем облікового запису або адресою електронної пошти. Відвідувач

отримує сповіщення про параметри, які не відповідають вимогам, якщо виявлено збіг.

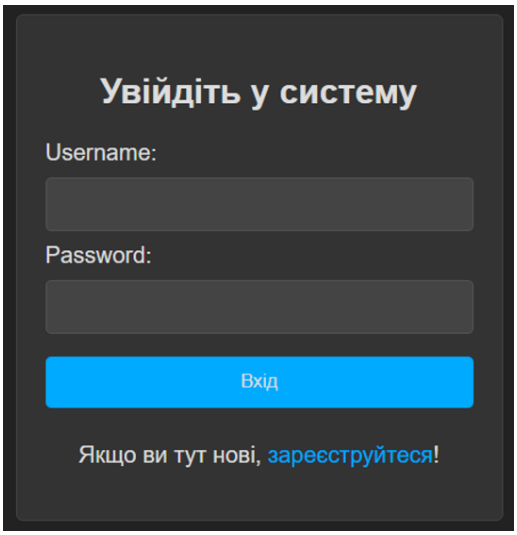

Рисунок 4.3 — Функціонал входу в аккаунт

Наступним етапом є перевірка наданих даних у необхідних форматах. Наприклад, це включає перевірку дійсності номера телефону та його відповідності вимогам українських мобільних операторів.

Крім технічної перевірки даних, важливим аспектом є забезпечення безпеки користувацької інформації. Всі чутливі дані, такі як паролі та реквізити платежів, зберігаються у зашифрованому вигляді та передаються через захищені протоколи, що мінімізує ризики несанкціонованого доступу та підвищує довіру користувачів до системи.

4.2 Обрання товарів

Після успішної автентифікації користувач направляється на сторінку зі списком товарів, які наразі є в наявності (рис. 4.4).

Щоб додати товари до власного кошика (рис. 4.5), користувач натискає на зображення товару на сторінці асортименту, щоб переглянути його поточну ціну та детальний опис.

За допомогою перемикача користувач може вибрати кількість одиниць товару, перш ніж натиснути кнопку «Додати до кошика». Технологія робить процес покупки простим та швидким, автоматично оновлюючи товари в кошику та відображаючи поточну загальну суму транзакції.

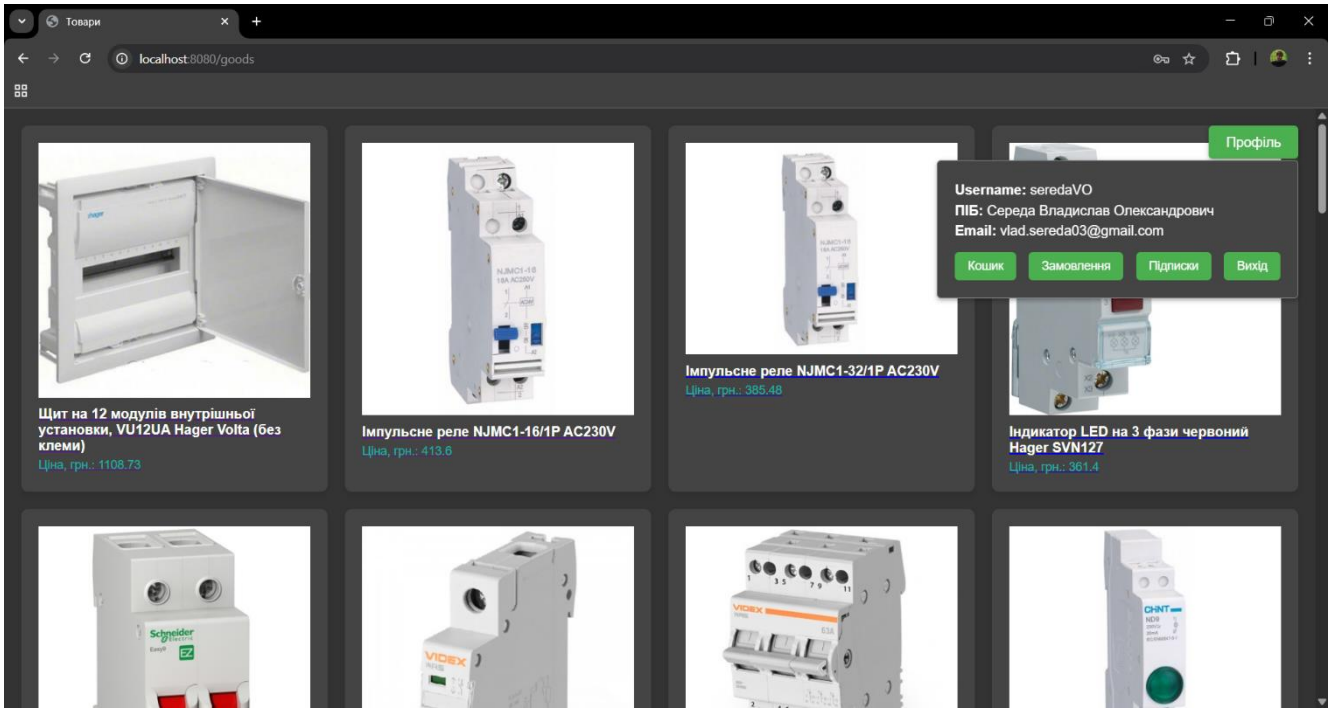

Рисунок 4.4 — Сторінка з асортиментом товарів

Назви та кількість вибраних товарів перелічені в окремих рядках на сторінці перегляду кошика. Відвідувач має можливість скористатися платіжним сервісом LiqPay або очистити кошик. Для завершення дії достатньо вибрати потрібний варіант і натиснути відповідну кнопку у відповідному контейнері.

На сторінці перегляду кошика назви та кількість обраних товарів відображаються в окремих рядках. Відвідувач може або очистити кошик, або

скористатися платіжним сервісом LiqPay. Просто виберіть правильний варіант і натисніть відповідну кнопку у відповідному контейнері, щоб завершити дію.

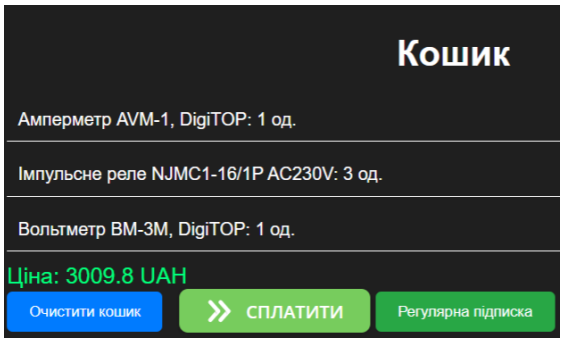

Рисунок 4.5 — Відображення кошика

Приклад сторінки з товаром та його детальною інформацією представлено на рисунку 4.6.

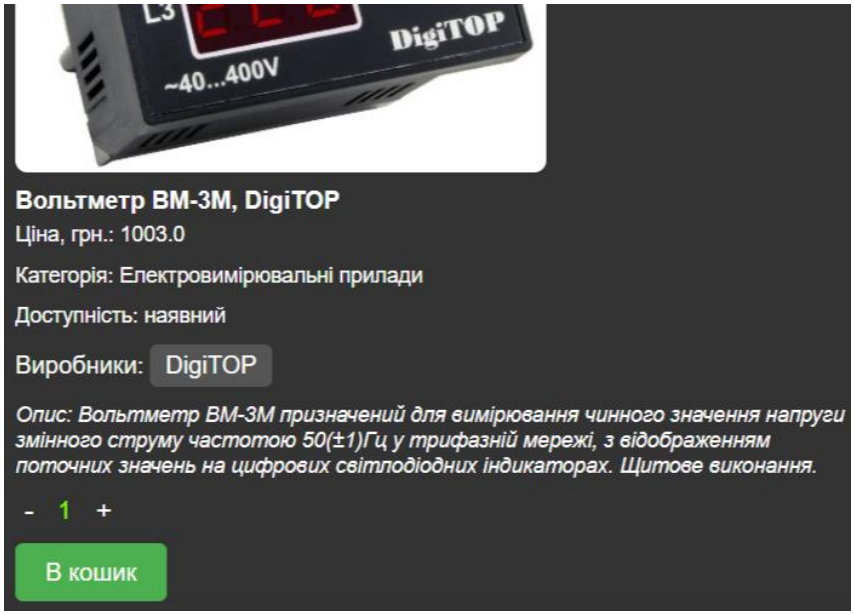

Рисунок 4.6 — Опис товару

Назва товару, ціна, категорія, виробник, наявність, а також детальний опис його характеристик і потенційного застосування відображаються на веб-сайті.

Перемикач товарів — це зручний спосіб додавати дрібні товари до кошика. Користувач натискає кнопку «в кошик» після визначення необхідної кількості одиниць. Повна колекція відображається в єдиному стилі та форматі, оскільки веб-сайт створюється динамічно.

4.3 Перегляд хронології замовлень

Натиснувши кнопку профілю та вибравши пункт «Замовлення» з випадаючого меню, користувач може переглянути історію попередніх замовлень, зроблених через головну сторінку з товарами. Кожне замовлення відображається в хронологічному порядку та зберігається окремо для користувача (рис. 4.7).

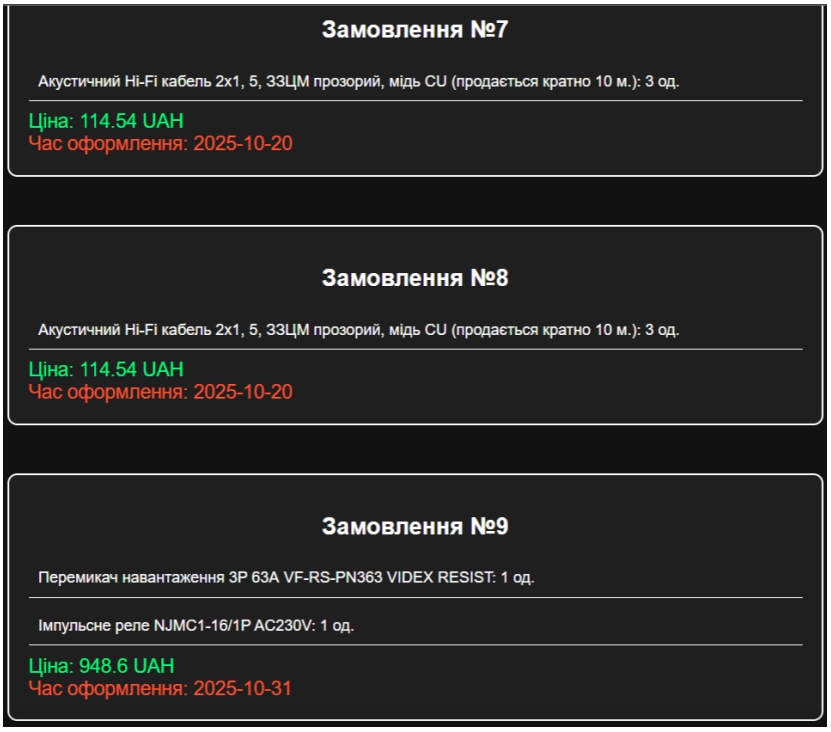

Рисунок 4.7 — Список успішних замовлень

У кожному блоці відображається кількість замовлених одиниць, загальна сума та час розміщення замовлення.

4.4 Оплата товарів

Платіжний сервіс LiqPay бере на себе процедуру, коли ви натискаєте кнопку «Сплатити» у вашому кошику. Користувачеві відображається спливаюче вікно вибору способу оплати. За потреби ви можете ввімкнути опцію надсилання чека на електронну пошту, ввівши правильну адресу. Схема роботи зображена на рисунку 4.8.

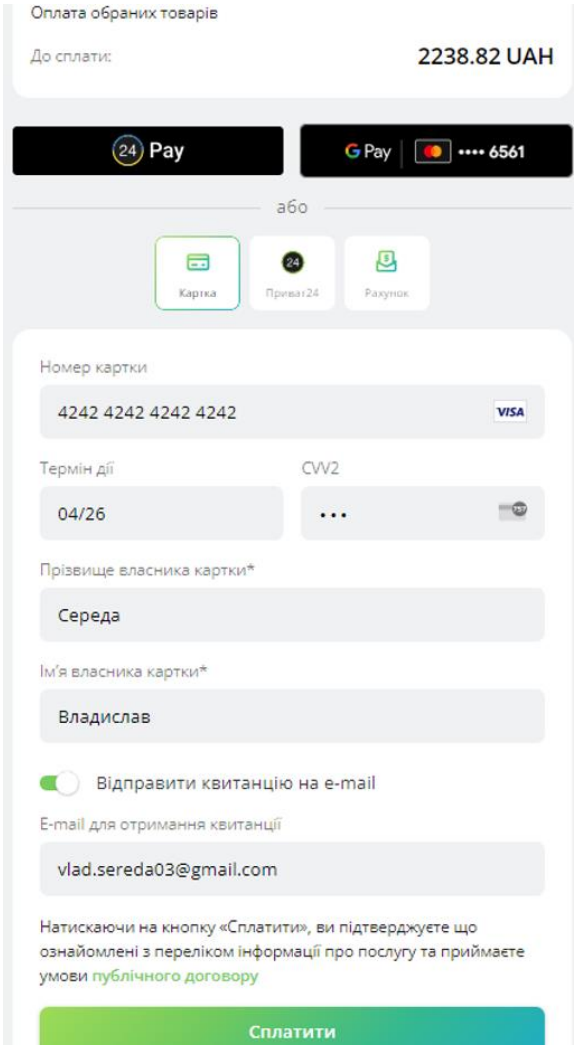

Рисунок 4.8 — Сплата замовлення

За винятком базової перевірки номера картки за допомогою алгоритму Luna та відповідності шаблону полів, надані дані не перевіряються на точність у

тестовому режимі. Винятком є електронна пошта; разом із результатом транзакції надсилається чек (рис. 4.9).

Після успішного завершення транзакції створюється квитанція, яка містить суму переказу, унікальний ідентифікатор транзакції, призначення платежу та дату отримання коштів. Користувач може завантажити цю квитанцію у форматі PDF для подальшого друку або зберігання.

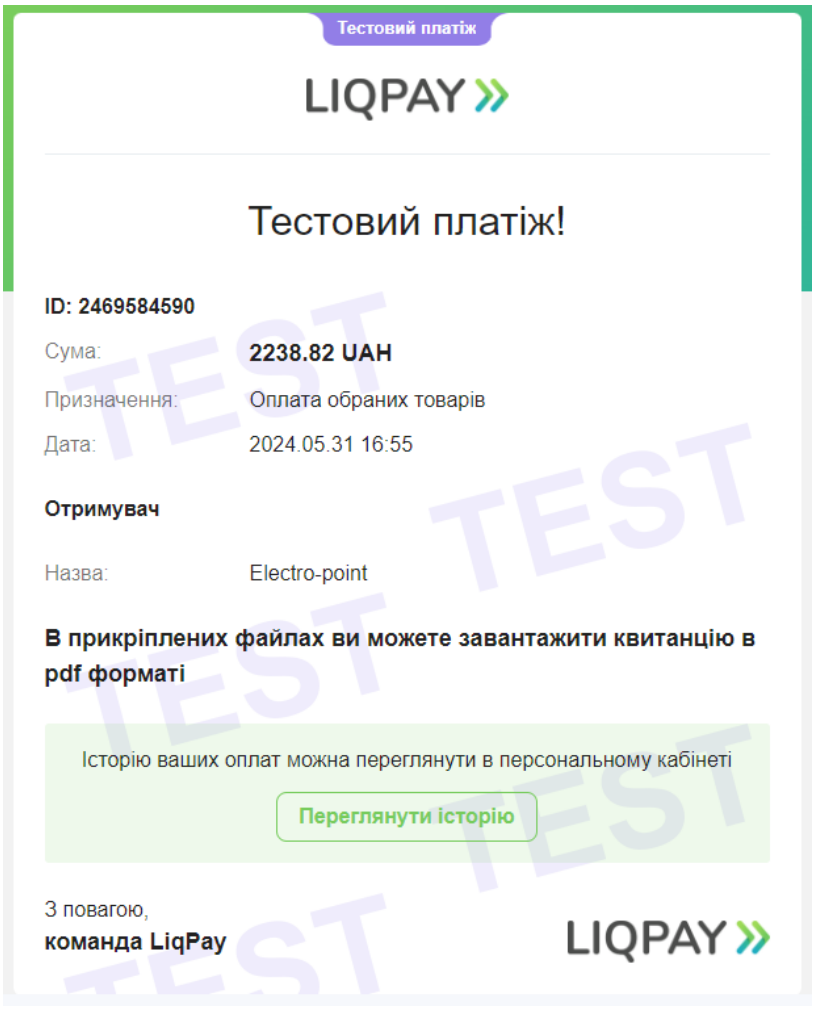

Рисунок 4.9 — Квитанція платежу

Система повідомляє користувача про результат транзакції після натискання кнопки «Сплатити» (рис. 4.10).

Повернення до сторінки з товарами передбачено відповідним елементом вікна.

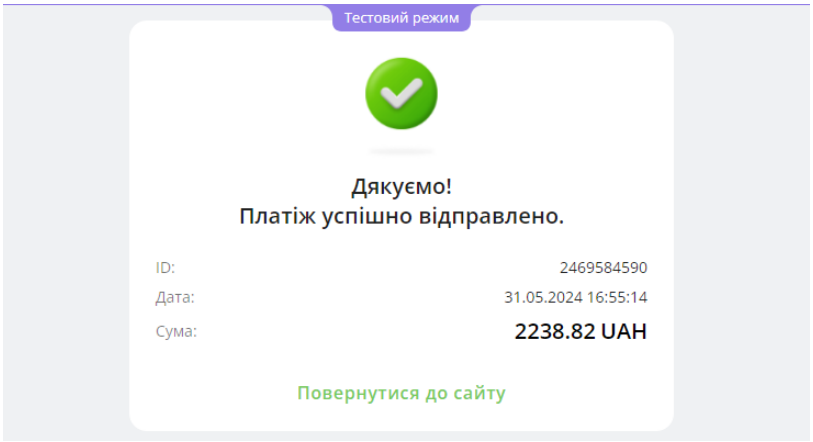

Рисунок 4.10 — Успішність платежу у тестовому режимі

Після відстеження та реєстрації нового платежу система видає нову сторінку із зазначенням успішного здійснення платежу (рис. 4.11).

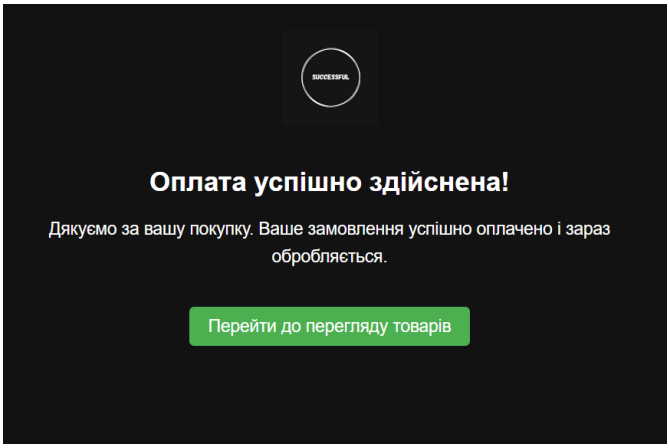

Рисунок 4.11 — Успішність платежу у тестовому режимі

Клієнт повертається на головну сторінку з різноманітними товарами та можливістю додавати товари до кошика та здійснювати повторні покупки після успішного завершення оплати.

4.5 Оформлення регулярної підписки

Система дозволяє створювати періодичні підписки на певні товари, окрім разових покупок (рис. 4.12) . Користувачі, які купують витратні матеріали або товари, які потрібно регулярно оновлювати, вважатимуть цей підхід дуже корисним. Процес підписки починається на сторінці кошика. Після того, як обрано потрібні речі, сторінка з кошиком пропонує перехід до створення регулярного платежу.

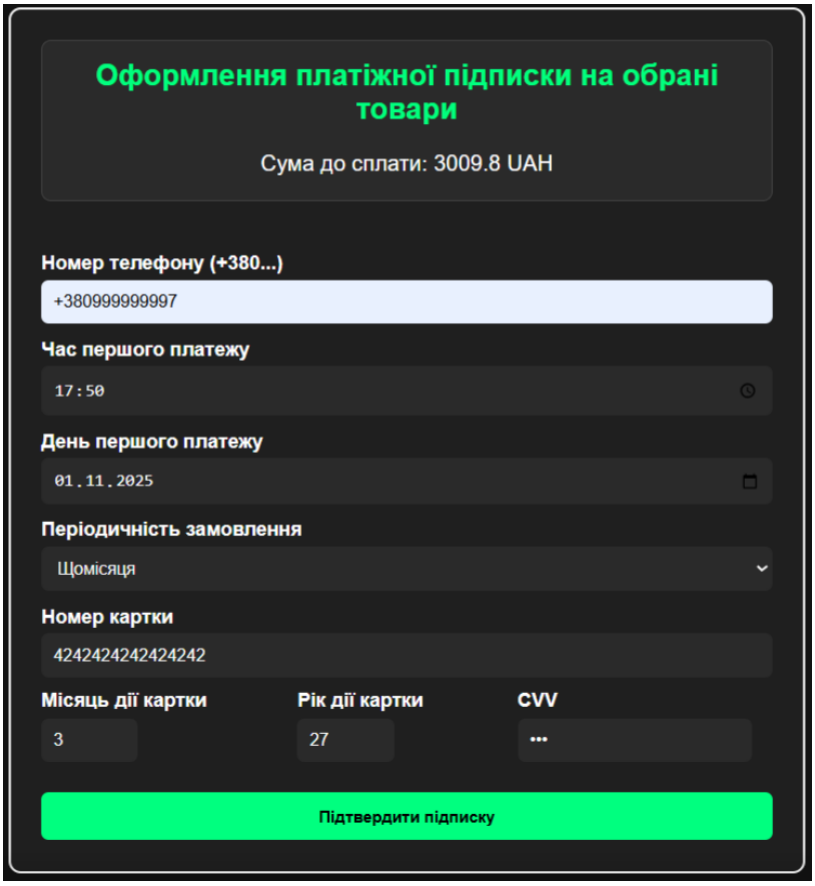

Рисунок 4.12 — Сторінка створення нової підписки

На сторінці підписки користувачеві відображається вся сума майбутніх регулярних платежів та набір полів, необхідних для точної обробки періодичних транзакцій. До них належать номер телефону, дата та час початкового платежу,

частота списання (щоденно, щомісяця тощо), інформація про банківську картку та код CVV. Завдяки дизайну інтерфейсу форми користувач може швидко ввести необхідну інформацію та перевірити її дійсність перед підтвердженням.

Після успішної реєстрації платіжний процесор LiqPay отримує дані про підписку, реєструє регулярний платіж та сповіщає систему про створення періодичного замовлення. База даних зберігає вхідні параметри, що дозволяє автоматично генерувати наступні списання відповідно до обраного графіка.

Друга функція під назвою «Мої підписки» дозволяє користувачам переглядати всі активні підписки. Пов’язаний номер телефону, частота платежів, дата початку та сума, обрана для кожного циклу дебетування, відображаються на веб-сайті (рис. 4.13). Кожна підписка відображається як окремий блок із чітко визначеними характеристиками для зручності використання.

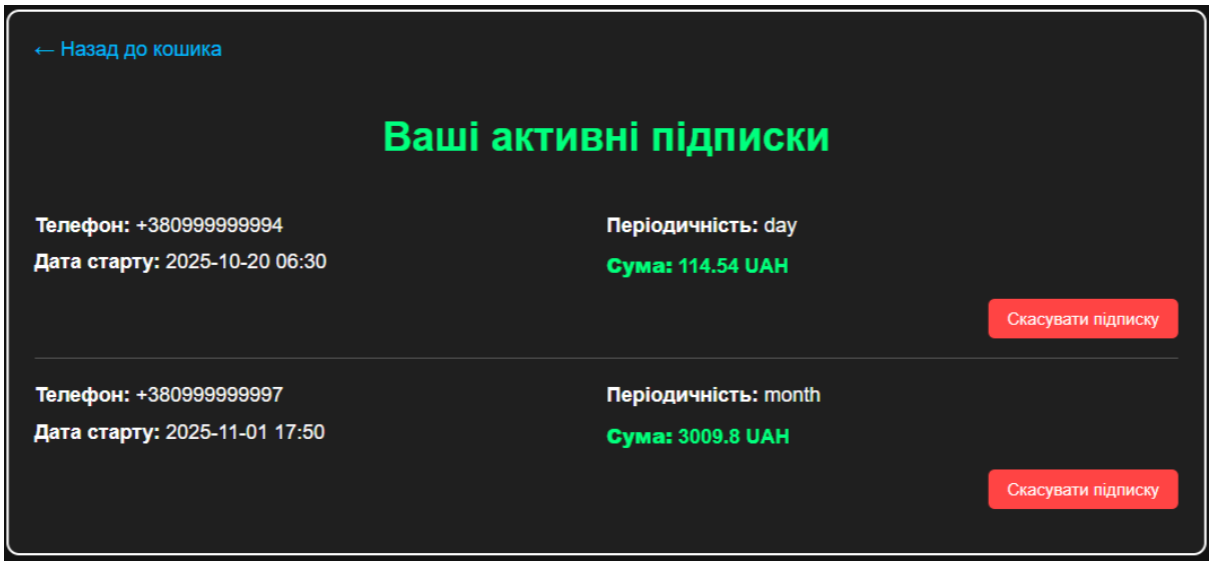

Рисунок 4.13 — Активні підписки

Крім того, користувач може скасувати будь-які поточні підписки. Для цього пропонується окрема кнопка, яка запитує LiqPay припинити здійснювати регулярні дебетування. Підписка більше не обробляється, а її статус видаляється зі списку активних після підтвердження дії.

Клієнти, які бажають автоматизувати процес покупки, вважатимуть систему набагато функціональнішою завдяки встановленню механізму регулярних платежів. Це підвищує комфорт користувачів та сприяє лояльності аудиторії, дозволяючи перетворити сервіс на більш адаптивне та довгострокове рішення.

4.6 Функції адміністратора

Адміністратор має доступ до іншого набору можливостей у поточній версії системи, які гарантують адміністрування асортименту продукції та збереження актуальності бази даних. Доступ до адміністративної частини можуть отримати лише користувачі з відповідними правами доступу, що забезпечує безпеку життєво важливих процесів та запобігає небажаним змінам вмісту.

Додавання нових товарів до каталогу є одним з основних обов'язків адміністратора. Ви можете ввести всі необхідні атрибути одиниць товару у спеціальній формі, наданій для цієї мети. Адміністратор вибирає виробників зі списку та вводить назву товару, ціну, повний опис, категорію та доступну кількість на сторінці «Додати товар» (рис. 4.14). Швидке введення інформації можливе завдяки зручному макету форми, що усуває необхідність додаткових технічних кроків.

Також додано опцію завантаження зображення товару. Адміністратор може додати графічний файл, який буде збережено та пов’язано з відповідним товаром після вибору опції «Вибрати фото». Це дозволяє створити привабливий каталог та зробити картки товарів більш інформативними. Як підтвердження завантаження, вибране зображення відображається під кнопкою.

Система перевіряє точність наданих даних перед додаванням товару до каталогу, звертаючи особливу увагу на формат числових полів, необхідність заповнення основних атрибутів та наявність принаймні одного виробника. Товар буде додано до бази даних після того, як адміністратор натисне кнопку «Додати» після успішної перевірки.

Крім того, адміністративний модуль спрощує контроль якості контенту, дозволяючи адміністратору оперативно виправляти будь-які неточності або застарілий матеріал. Це гарантує точне представлення інформації для кінцевих користувачів і дозволяє підтримувати каталог актуальним.

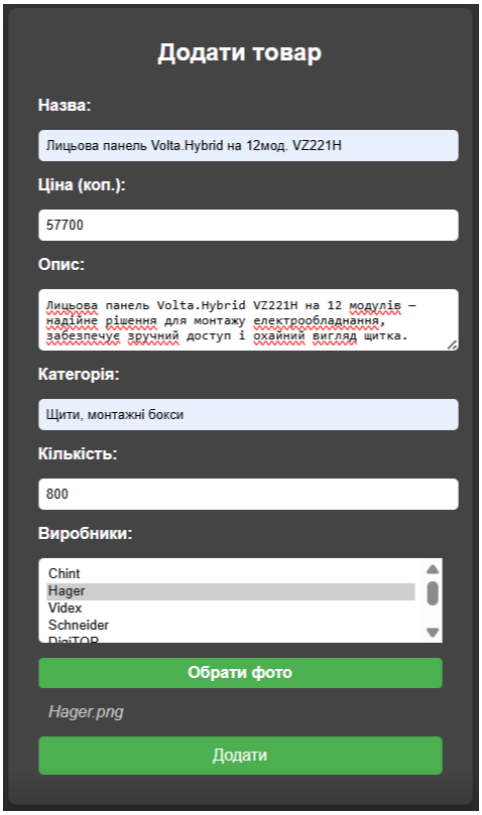

Рисунок 4.14 — Доповнення асортименту новим товаром

Підтримка каталогу продукції максимально універсальна завдяки наявності панелі адміністрування. Адміністратор може швидко оновлювати інформацію, додаючи нові товари електрообладнання, оновлюючи їх кількість, а також змінюючи ціни чи описи. Це гарантує своєчасне оновлення асортименту та підвищує загальну ефективність системи.

5 РОЗРОБЛЕННЯ СТАРТАП-ПРОЄКТУ

Інноваційні технічні рішення є особливо важливими на сучасному етапі цифрового переходу економіки, оскільки вони можуть підвищити ефективність корпоративних процесів, оптимізувати витрати та покращити комунікацію між бізнесом та кінцевими споживачами. Розвиток місцевих та міжнародних ринків все частіше зумовлений стартапами, що виникають на стику ІТ-технологій та реальних потреб ринку. У контексті представленої магістерської роботи необхідно проаналізувати ринковий потенціал, конкурентоспроможність та можливості практичного впровадження створеного програмного продукту, окрім його технічної реалізації.

Розробка стартап-проекту, що базується на розробленій інформаційній системі та відображає маркетингові та фінансові аспекти його реалізації, є основною метою цієї частини. Основними цілями є оцінка інноваційної складової проєкту, визначення його ринкових можливостей, вивчення конкурентного та споживчого середовища, а також запропонування пропозиції щодо комерціалізації розробки. Таке дослідження дозволяє нам визначити економічну життєздатність рішення на додаток до його технічної життєздатності, що є вирішальною передумовою для майбутнього зростання та виходу на ринок у форматі стартапу.

5.1 Загальна характеристика стартап-ідеї

Метою стартап-проекту, створеного в рамках цієї магістерської роботи, є розробка передової автоматизованої онлайн-системи торгівлі енергетичним обладнанням, яка задовольнятиме потреби малого бізнесу, монтажників, приватних клієнтів та організацій, що надають послуги з електротехніки. Однією з основних тенденцій зростання ринку електротехнічних товарів та послуг є створення єдиної цифрової платформи, яка об’єднує каталог спеціалізованих

продуктів, гнучку систему управління замовленнями та інтегровані механізми оплати, включаючи періодичні платежі.

Необхідність прозорих, швидких та автоматизованих каналів комунікації з постачальниками обладнання зростає через поступовий цифровий перехід сегментів B2C та B2B. Поточні рішення ринку в першу чергу орієнтовані на широкий сегмент споживачів або загальну онлайн-роздрібну торгівлю, яка ігнорує унікальні характеристики енергетичного сектору, діапазон технічних параметрів товарів, вимогу частих покупок та необхідність підтримки розширених сценаріїв оплати. Розробляючи унікальну платформу, адаптовану до потреб електротехнічної галузі, запропонована фірма вирішує ці проблеми.

Основною особливістю концепції є інтеграція сучасних методів оплати, таких як розробка та адміністрування періодичних підписок на постачання товарів або часті оновлення компонентів. Клієнти отримують зручний сервіс з автоматичним списанням коштів та історією платежів, а бізнес виграє від очікуваного грошового потоку та більшої лояльності аудиторії. Це створює нові можливості як для користувачів, так і для постачальників. Така концепція особливо актуальна в енергетичній галузі, де численним клієнтам часто потрібні кабелі, автоматичні вимикачі, розподільчі щити, компоненти освітлення та інші продукти, які підлягають плановому оновленню або періодичній заміні.

Окрім створення магазину, основною метою стартапу є розробка всеохоплюючої цифрової екосистеми, яка може автоматизувати кожен етап контакту клієнта з енергетичним обладнанням, від вибору продуктів до здійснення повторних покупок, від перегляду історії замовлень до управління підписками. Функція персоналізації, яка полегшує швидкий пошук відповідного обладнання на основі минулих замовлень, стандартних налаштувань та пропозицій виробника, також є основою платформи.

Масштабованість стартапу є вирішальною особливістю на додаток до його функціональності. Дизайн системи поділяє її на окремі модулі, кожен з яких можна оновлювати незалежно від платформи в цілому. Це робить рішення адаптованим

до різних ринкових категорій, включаючи інженерні системи, техніку для розумного дому, інструментальні матеріали та навіть складні IoT-рішення для управління енергією. Це відкриває стратегічні можливості для зростання стартапу та розширення бізнес-моделі.

Скорочено ідея проекту та пов’язана інформація зведена в таблиці 5.1.

Таблиця 5.1 — Загальні дані стартап-проекту

Зміст ідеї (що пропонується)	Напрямки застосування	Вигоди для користувача
Розробка хмарної платформи для управління цифровими підписками та рекурентними платежами з вбудованим механізмом авторизації на основі OAuth 2.0, аналітикою транзакцій, гнучкими бізнес-налаштуваннями та автоматизацією білінгових процесів. Система орієнтована на SaaS-компанії, онлайн-сервіси та бізнеси, які використовують регулярні платежі.	Інтернет-платформи з моделлю підписки (освітні сервіси, медіа-платформи, онлайн-курси).	Зручне управління підписками без ручних операцій.
	SaaS-продукти з оплатою за місяць/рік.	Автоматизація списань і мінімізація людських помилок.
	Маркетплейси цифрових товарів.	Прозора історія платежів та аналітика.
	Сервіси автоматизації бізнес-процесів.	Безпечна авторизація та захист платіжних даних.
	Мобільні застосунки з преміум-функціями.	Зменшення витрат на розробку власного білінгового рішення.
	Фінтех-проекти з механізмом регулярних транзакцій.	Швидка інтеграція з існуючими додатками.
	Платформи, що потребують централізованого управління користувачами та ролями.	Висока надійність і масштабованість сервісу.

Отже, загальні риси концепції показують, що стартап зосереджений на задоволенні поточної потреби в цифровізації в галузі продажу енергетичного обладнання та розробці сервісу, який поєднує високий ступінь автоматизації, зручність для кінцевого користувача та креативний метод організації регулярних

покупок. На ринку, де більшість підприємств досі взаємодіють з клієнтами традиційними засобами, цей формат дає вам конкурентну перевагу.

5.2 Аналіз ринкової потреби

Низка ринкових елементів, що визначають сучасні тенденції розвитку електротехнічної галузі, перетинаються, щоб створити попит на спеціалізовану онлайн-платформу для продажу енергетичного обладнання. По-перше, важливо зазначити, що ринок електрообладнання швидко розширюється як у комерційному, так і в приватному секторах. Це пов’язано з великим обсягом будівельних та ремонтних робіт, зростаючою потребою в енергоефективних рішеннях, зростанням ринку відновлюваної енергії та постійним відродженням інженерних мереж у будинках та на підприємствах. Одночасно зростає кількість користувачів, які хочуть, щоб обладнання постачалося швидко, прозоро та без зайвих посередників.

Ринок загальних інтернет-магазинів зріс, але сектор спеціалізованих електротоварів все ще фрагментований. Магазини зазвичай зосереджуються на широкому асортименті продукції, не пропонують вичерпної технічної інформації або не спрощують перемикання між сумісними деталями. Це ускладнює роботу монтажників, сервісних інженерів та власників бізнесу, оскільки затримує пошук компонентів та збільшує ймовірність придбання невідповідного обладнання. Робочі процеси ускладнюються та забирають багато часу через відсутність регулярних закупівель або інтегрованих рішень з управління проектами.

Водночас, український ринок демонструє сильну тенденцію до цифровізації міжбізнесових обмінів. Після 2020 року більшість корпоративних клієнтів очікують, що постачальники матимуть обліковий запис клієнта або веб-портал, де вони зможуть легко здійснювати покупки, контролювати хід замовлення, повторювати попередній запит або налаштовувати автоматичні списання. Однак традиційні електротехнічні підприємства часто працюють, використовуючи застарілі моделі обслуговування, такі як телефонні дзвінки, паперові рахунки,

ручні розрахунки кошторисів або офлайн-каталоги продукції. Клієнти обирають постачальників, які пропонують швидкість, прозорість та операційну автоматизацію, що різко знижує їхню конкурентоспроможність.

Ще однією тенденцією, яка підсилює створену потребу, є модернізація житлових електричних мереж та розширення культури електробезпеки. Сьогодні пересічні споживачі активно порівнюють продукти, шукають моделі від різних виробників та оцінюють технічні параметри, а не покладаються лише на поради фахівців. У цій ситуації особливо важливо надати користувачам доступ до вичерпних технічних даних, фільтрації на основі параметрів, пропозицій щодо сумісності та швидкого пошуку запасних частин. Через свою зосередженість на масовій роздрібній торгівлі, а не на технічній аудиторії, більшість масових онлайн-бізнесів цього не пропонують.

Особливу увагу слід приділити попиту на регулярні закупівлі. Кабелі, автоматичні вимикачі, шинопроводи, технічні коробки та інші компоненти завжди потрібні підприємствам, що займаються інженерними установками, сервісним обслуговуванням або електромонтажними роботами. Ремісники витрачають забагато часу на пошук відкритих позицій та розміщення нових замовлень через відсутність автоматизованих систем замовлення. Цю проблему можна вирішити за допомогою методу підписки стартап-платформи, який гарантує надійність поставок та економить час бізнес-користувачам.

У світлі цих подій очевидно, що на ринку існує справжня потреба в спеціалізованому сервісі, який одночасно пропонуватиме технічну глибину каталогу, зручність замовлення, швидкі варіанти оплати та адаптивні формати взаємодії з користувачем. Ця потреба служить основою для концепції стартапу та підтверджує здатність платформи конкурувати, забезпечуючи складність, автоматизацію та увагу до галузевих стандартів, яких немає в сучасних рішеннях.

5.3 Аналіз аналогів та конкурентного середовища

Важливим компонентом розвитку стартап-проекту є аналіз аналогів та конкурентного середовища, що дозволяє оцінити застосовність ідеї, вирішити, як позиціонувати майбутній продукт, та виявити можливості для отримання конкурентної переваги. Ринок електрообладнання в Україні представляють численні компанії, які пропонують як широкий асортимент продукції, так і спеціалізовані рішення для професійних користувачів. Структура ринку різноманітна, але не повністю насичена, оскільки кожен із поточних конкурентів має свою бізнес-модель, рівень обслуговування, технологічну зрілість та стратегічні пріоритети.

Основних конкурентів можна розділити на три категорії: офлайн-бізнес з частковою онлайн-присутністю, спеціалізовані інтернет-магазини електротоварів та загальні торговельні майданчики. Хоча вони мають великий асортимент електротоварів, загальні ринки, такі як Rozetka або Prom.ua, в першу чергу розроблені для масових покупців. Цим системам бракує інструментів для закупівель проектів, професійних фільтрів, комплексних технічних вимог та можливості об’єднання товарів у цілі системи чи рішення. Як результат, вони не відповідають потребам клієнтів B2B, установників та техніків, яким потрібен точніший вибір обладнання та глибша технічна інформація.

Друга категорія складається зі спеціалізованих магазинів електротоварів, які спеціалізуються на продажу освітлювальних приладів, електротоварів та аксесуарів для встановлення. Ці підприємства зазвичай мають найглибші каталоги у своїй галузі, але більшість із них не автоматизують процес продажу за допомогою сучасних цифрових інструментів. Багато магазинів працюють як традиційні інтернет-каталоги, пропонуючи інформацію у вигляді статичних списків товарів без автоматичних оцінок, рекомендацій щодо сумісності або особистого облікового запису для повторюваних транзакцій. Важливо також зазначити, що значна частина

цих роздрібних торговців має невеликий вибір, оскільки вони залежать від певного дистриб'ютора.

Офлайн-мережі, такі як спеціалізовані склади електротоварів або магазини «зроби сам», забезпечують чітку конкуренцію. Фізична доступність та огляд продукції є їхніми перевагами, але онлайн-можливості зазвичай обмежені. Багато з них не мають повних профілів користувачів, не використовують цифрові інструменти для аналітики закупівель та не пропонують актуальний баланс товарів онлайн. Оскільки сучасний клієнт очікує потенціалу постійної взаємодії через цифрові канали, це дозволяє їм маневрувати конкурентами.

Результати аналізу ринку вказують на низку суттєвих недоліків існуючих аналогів, зокрема відсутність автоматизованих рішень для повторюваних покупок, низький ступінь інтеграції експертних інструментів для вибору обладнання, відсутність персоналізації пропозицій та можливості для управління проектами та створення кошторисів. Замість розвитку довгострокових B2B-зв'язків більшість конкурентів зосереджені на разових продажах. Також слід виділити недостатній ступінь мобільної адаптації та відсутність складних систем рекомендацій, які могли б покращити користувацький досвід та пришвидшити процес вибору продукту.

Для позиціонування стартап-платформи на такому конкурентному ринку важливо підкреслити важливі переваги, які можуть значно підвищити якість взаємодії з користувачами на кожному рівні, від пошуку продукту до оформлення замовлення та повторної покупки. Ці функції включають повну прозорість залишків продуктів та витрат, зручний особистий кабінет для бізнес-клієнтів, динамічну систему рекомендацій на основі сумісності та технічних характеристик, автоматизацію покупок через механізм підписки та складні інструменти фільтрації та порівняння. Забезпечуючи професійний рівень обслуговування в сучасному цифровому форматі, бізнес може таким чином заповнити нішу між широкими торговими майданчиками та вузькоспеціалізованими магазинами.

На завершення можна сказати, що хоча ринок електротоварів є висококонкурентним, існує низка незадоволених потреб, які нова платформа може

задовольнити. Реальними критеріями успішного виходу на ринок та створення довготривалої конкурентної переваги є ефективне позиціонування, зосередження на професійному сегменті та ретельна каталогізація продукції.

Скорочено характеристики проекту подані в таблиці 5.2.

Таблиця 5.2 — Визначення характеристик ідеї проекту

№ з/п	Економ. характеристики	Концепція (стратегія) конкурентів				Слабкі (W), нейтр. (N) та сильні (S) сторони		
		Розроблений проєкт	Stripe Billing	Paddle	Recurly	W	N	S
1	Сегментування моделі підписки	Гнучке налаштування	Є	Є	Є			+
2	Вартість інтеграції	Низька	Середня	Середня	Висока		+	
3	Комісія за транзакції	Низька	Помірна	Помірна/ висока	Висока			+
4	Локальні платежі (LiqPay)	Повна підтримка	Обмеж.	Обмеж.	Обмеж.			+
5	Аналітика платежів	Розширена	Базова	Розшир.	Розшир.		+	
6	Технічна підтримка	Персоналізована	Стандарт	Стандарт	Розшир.		+	
7	Кастомізація	Дуже висока	Середня	Низька	Середня			+
8	Захист даних (OAuth 2.1)	Сучасні механізми	Високий	Високий	Високий		+	
9	Швидкість інтеграції	Висока	Середня	Висока	Середня			+
10	Масштабованість	Висока	Висока	Середня	Висока		+	

5.4 Технологічний аудит

Вирішальним кроком в оцінці життєздатності стартап-проекту є технологічний аудит, який дозволяє з технічної точки зору визначити реальну можливість втілення запропонованої ідеї на практиці. Основна мета аудиту — оцінити сучасні технології, щоб визначити їх ступінь зрілості, відповідність галузевим стандартам та здатність гарантувати масштабованість, надійність та безперервність системи. Проводячи технологічний аудит, ви також можете визначити області для майбутнього вдосконалення технологічної платформи та скласти перелік ризиків і технічних перешкод, які можуть вплинути на швидкість та якість впровадження стартапу.

Під час розробки програмного забезпечення стартапу використовуються сучасні серверні та клієнтські технології, що забезпечує високу швидкість роботи системи, безпеку та адаптивність. Технологічний стек Java та фреймворк Spring, які пропонують модульність, структуру та потенціал для ефективної взаємодії з системою, формують основу серверної архітектури. Помітною перевагою є використання Spring Boot, що спрощує налаштування програми, а також Spring Security, що гарантує надійну взаємодію зі службами авторизації та безпеку персональних даних користувачів за допомогою сучасної криптографії та мережевих протоколів.

Оцінка використання OAuth 2.0 як основного механізму дозволів та автентифікації користувачів є окремою сферою технологічного аудиту. Цей стандарт допомагає відокремити важливі компоненти системи та гарантує їх незалежне масштабування при використанні разом із власним сервером авторизації та сервером ресурсів. Проект також використовує PKCE, токенізацію, токени короткострокового доступу та ротацію токенів оновлення, що відповідає міжнародним стандартам безпеки та значно знижує ймовірність компрометації облікових даних. Таким чином, незалежно від кількості активних клієнтів,

технологічна платформа може запропонувати повний цикл безпечного доступу до ресурсів.

База даних стартап-проекту використовує реляційний формат для максимально ефективного зберігання структурованих даних про продукти, користувачів, замовлення та періодичні підписки. Обрані ORM-рішення пропонують швидку взаємодію з даними та гнучке розширення структури таблиць у міру зростання потреб бізнесу. Аудит демонструє, що поточна архітектура дозволяє вирішити проблему підвищення продуктивності як за рахунок вертикального, так і горизонтального масштабування, а також підключити мікросервіси в майбутньому.

Високий рівень технологічної зрілості також спостерігається в клієнтському сегменті стартапу. Адаптивне макетування дозволяє зберігати функціональність на різних типах пристроїв, а використання сучасних фреймворків динамічного рендерингу гарантує хороший користувацький досвід. LiqPay, який використовує токенізацію та управління статусом транзакцій для обробки як одноразових платежів, так і регулярних підписок, інтегрований з платіжною системою. Технологічна оцінка демонструє, що цей підхід до інтеграції повністю відповідає світовим стандартам безпеки PCI DSS і може бути розширений додатковими постачальниками платежів за потреби.

Аудит масштабованості також дуже важливий. Гнучкість та вільний зв’язок компонентів, що лежать в основі запропонованої архітектури стартапу, дозволяють поступово модифікувати систему для розміщення зростаючої бази користувачів. Інтеграція інструментів оркестрації, таких як Kubernetes або Docker Swarm, спрощується завдяки використанню контейнеризації та можливості перенесення сервісів у хмарне середовище. Це створює основу для розробки високонавантаженої платформи, яка може обробляти величезні обсяги даних, гарантуючи безперервність процесів та зменшуючи час простою.

За результатами технологічного аудиту, стартап-проект має високий ступінь технологічної гнучкості та може зростати та масштабуватися. Обрана архітектура

пропонує гнучкість, безпеку та ефективність, а використані інструменти задовольняють сучасні потреби ринку. Це призводить нас до висновку, що технологічна база стартапу є достатньо міцною, щоб гарантувати успішне впровадження продукту в реальних конкурентних умовах та закласти основу для майбутніх удосконалень та розвитку функціональності.

5.5 Ринкові можливості та загрози

Ключовою складовою стратегічного планування стартап-проекту є аналіз ринкових можливостей і загроз, який дає змогу оцінити зовнішнє середовище, визначити найбільш перспективні напрямки розвитку та перерахувати можливі перепони на шляху реалізації бізнес-ідеї. Успішний маркетинговий план можна розробити за допомогою такого дослідження, яке також полегшує розуміння рис цільового ринку та модифікацію проекту відповідно до мінливих ринкових умов.

Зростаюча потреба в цифрових послугах, автоматизації корпоративних процесів та безпечних онлайн-платформах, що пропонують швидку авторизацію, контроль доступу та підключення до сучасних методів оплати, є єдиною з основних ринкових перспектив запуску. Ринок цифрових послуг постійно розширюється, що робить застосування креативних рішень прибутковим та актуальним. Крім того, завдяки зростанню цифровізації та популярності онлайн-підписок створюється сприятливе середовище для товарів, спрямованих на пропонування зручного контенту чи послуг монетизації.

Іншу можливість відкриває активне зростання малих і середніх підприємств, яким потрібні готові технологічні рішення для захисту даних, організації доступу та автоматизації процесів. Багато компаній хочуть інтегрованих готових рішень, які легко запровадити та не коштують багато грошей, оскільки їм бракує ресурсів чи IT-інфраструктури, необхідних для розробки складних програмних систем. Продукти із сучасною архітектурою, високою надійністю та масштабованістю, як-от запропонований стартап, можуть заповнити життєздатну нішу.

Наростаюче значення інформаційної безпеки в сучасній цифровій економіці — це ще одна проблема, яка представляє бізнес-потенціал. Постійні кіберзагрози змушують компанії впроваджувати нові рекомендації щодо безпеки та використовувати сервіси з перевіреними методами авторизації, контролю доступу та секретного зберігання даних. Як результат, існує постійна потреба в рішеннях, таких як OAuth 2.0, PKCE, багаторівнева авторизація та сучасні криптографічні методи, які відповідають міжнародним нормам та забезпечують повну безпеку.

Однак жорстка конкуренція в бізнесі цифрових технологій створює низку ризиків, які необхідно враховувати. Швидкий технічний розвиток галузі є одним із найважливіших факторів; нові інструменти, стандарти та конкуренти можуть з’являтися дуже швидко. Це означає, що фірма повинна постійно інвестувати в оновлення своєї архітектури, модернізацію своїх продуктів та адаптацію до нових правил. Конкурентоспроможність проекту може бути значно знижена через застарілі платформи або рідкісні оновлення.

Надзвичайна чутливість ринку до проблем конфіденційності даних є ще однією небезпекою. Будь-який інцидент, пов’язаний з витоком інформації або порушенням правил, що регулюють використання персональних даних, може серйозно зашкодити репутації стартапу та призвести до втрати користувачів. Як наслідок, важливо ретельно дотримуватися рекомендацій щодо кібербезпеки, регулярно проводити аудити та тести на проникнення, а також стежити за будь-якими потенційними вразливостями в системі.

Нестабільні фінанси потенційних клієнтів у невизначені економічні часи становлять ще один ризик. Через скорочення бюджету або зниження попиту на власні послуги, підприємства можуть відмовитися від креативних рішень, що може вплинути на швидкість виходу компанії на ринок. Під час створення цінової стратегії, надання різних тарифних рівнів, пробних періодів або гнучких варіантів оплати необхідно враховувати цей фактор.

Нарешті, важливо пам’ятати, що зростання глобальних компаній зі значними фінансовими ресурсами та домінуючими позиціями на ринку може бути пов’язане

з ринковими ризиками. Їхнє існування може ускладнити вихід на деякі ринки, особливо якщо вони пропонують складні рішення з різноманітним використанням. Однак місцеві компанії мають шанс конкурувати завдяки гнучкості своєї бізнес-моделі, глибшій адаптації до місцевих потреб та нішевій експертизі.

З огляду на вищезазначене, можна зробити висновок, що існує значний ринковий потенціал для реалізації стартапу. Ці шанси випливають з необхідності для бізнесу використовувати нові технічні рішення, зростання попиту на цифрові послуги та зростання ролі інформаційної безпеки. Однак на ринку існують інші ризики, якими необхідно методично керувати, такі як фінансова нестабільність, зміни в регулюванні та тиск з боку конкурентів. Ви можете створити стратегію розвитку, яка зробить фірму ефективною та стійкою на швидкозмінному, переповненому ринку, ретельно зваживши потенціал та ризики.

5.6 Можливості комерціалізації

Здатність запропонованого продукту вийти на ринок, генерувати економічні вигоди та гарантувати довгостроковий сталий розвиток визначає потенціал комерціалізації стартап-проекту. Потенціал комерціалізації особливо високий для цифрових послуг, що зосереджені на автоматизації процесів компанії, інтеграції з платіжними системами та підвищенні інформаційної безпеки. Це пояснюється тим, що бізнес загалом рухається до цифрової трансформації, моделі монетизації за передплатою стають все більш популярними, а також зростає потреба в технічних рішеннях, які знижують витрати та одночасно підвищують операційну ефективність.

Широка цільова аудиторія проекту, яка охоплює кілька ринкових категорій, таких як малий та середній бізнес, великі організації, онлайн-сервіси та цифрові платформи, що використовують модель передплати, є однією з його головних переваг. Як результат, продукт має здатність одночасно обслуговувати кілька галузей, включаючи фінансово-технологічні фірми, онлайн-освітні платформи,

SaaS-сервіси, медичні інформаційні системи, CRM/ERP-рішення та торговельні майданчики. Кожна з цих галузей має вузькоспеціалізоване суперництво, що дозволяє добре позиціонованому продукту не лише заповнити свою нішу, але й перетворитися на ключову частину IT-інфраструктури організації.

Використання моделі регулярних платежів, яка пропонує стабільний та передбачуваний грошовий потік, є значною можливістю комерціалізації на додаток до її широкої доступності на ринку. На відміну від разових продажів, модель передплати забезпечує довгострокові зв’язки зі споживачами та дозволяє стабільно збільшувати дохід шляхом покращення функцій, продажу додаткових пакетів або збільшення кількості користувачів в одній фірмі. Це зміцнює фінансову стабільність стартапу та робить його привабливим для інвесторів, оскільки бізнес-моделі передплати демонструють більшу передбачуваність та меншу залежність від коливань ринку.

Технологічна сучасність продукту є одним із ключових елементів, що підвищує ймовірність комерціалізації. Оскільки підприємства все більше зосереджуються на наданні послуг, що відповідають найновішим нормам кібербезпеки, використання Spring Authorization Server, OAuth 2.1, PKCE та інших сучасних стандартів надає їм конкурентну перевагу. Наявність таких технологічних рішень підвищує привабливість продукту як для вітчизняних, так і для іноземних споживачів, створюючи можливості для масштабування та розширення за межі локальної території.

Простота інтерфейсу стартап-рішення з існуючими системами клієнтів також значною мірою сприяє комерціалізації. Багато підприємств обирають сервіси з добре документованими API, здатністю швидко впроваджувати та підтримувати загальні протоколи обміну даними, щоб уникнути складних та дорогих процесів адаптації. У цій ситуації запропоноване рішення має потенціал перетворитися на універсальну платформу, яку можна інтегрувати в різноманітні інформаційні системи, не вимагаючи від клієнта значних ресурсів.

Перспектива створення партнерських програм та співпраці з іншими IT-фірмами, такими як виробники CRM, ERP, LMS та інших платформ, також пов’язана з потенційним зростанням компанії. Завдяки рекомендаціям, попереднім інтеграціям і навіть співпраці за принципом «white label», таке партнерство може запропонувати більше каналів збуту та допомогти в консолідації продукту на ринку. Продукт зрештою може бути включений до складних цифрових екосистем, де низка взаємопов’язаних послуг додає цінності для бізнесу.

Крім того, високий ступінь інноваційності продукту гарантує доступність венчурних інвестицій, грантів та участі в бізнес-акселераторах. Проекти, що вирішують актуальні бізнес-проблеми, легко масштабуються та мають технологічну зрілість, традиційно приваблювали інвесторів. Цим вимогам відповідає компанія, що базується на сучасній авторизації, безпеці та автоматизованих фінансових операціях, що підвищує ймовірність успішного фінансування.

Потенціал проникнення на світовий ринок — це ще один аспект, який необхідно враховувати. Технологічні рішення проекту не потребують фундаментальних змін для адаптації до міжнародних ринків, оскільки вони відповідають загальновизнаним міжнародним стандартам. Це дозволяє розширити пропозицію на ЄС, Північну Америку та Азію, регіони з високим рівнем використання моделей передплати та автоматичної авторизації. Крім того, існує більший потенціал комерціалізації завдяки встановленому попиту на безпечні та стандартизовані рішення на ринках США та ЄС.

В результаті, стартап-проект має значні та різноманітні перспективи монетизації. Вони базуються на потребі ринку в цифрових послугах, адаптивності продукту, простоті інтеграції, дотриманні сучасних правил кібербезпеки, масштабованості та великому потенціалі моделі монетизації передплати. У поєднанні ці елементи створюють сприятливі умови для дебюту продукту на ринку, а також для його подальшого розвитку, вдосконалення та глобальної експансії.

ВИСНОВКИ

В рамках роботи було оцінено поточні ринкові тенденції в обраній галузі, розглянуто сучасні методи впровадження таких платформ, а також проведено аналіз існуючих онлайн-сервісів для замовлення електротоварів.

На основі зібраних даних було визначено найкращий формат впровадження — веб-додаток з простим, але достатньо функціональним інтерфейсом для обслуговування клієнтів.

У розробці програмного продукту були використані сучасні технології, зокрема мови програмування Java та JavaScript, система керування залежностями Maven, фреймворк Spring та його компоненти, а також додаткові бібліотеки для обробки даних, логування та інтеграції сторонніх сервісів. Контент відображався з використанням стандартів HTML та CSS.

Автоматизована система онлайн-продажів електрообладнання пропонує можливості для підвищення продуктивності малих та середніх підприємств. Завдяки своїй величезній масштабованості, ви можете модифікувати програму відповідно до потреб різних користувачів та секторів. Впровадження адміністративних функцій та розширеного механізму пошуку та фільтрації товарів є частиною постійного розвитку проекту.

У результаті розроблена система відповідає критеріям, має можливості для вдосконалення та може бути ефективно використана для укладання угод та розміщення замовлень на електрообладнання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Download Java. *Java.* URL: https://www.java.com/en/download/ (date of access 15.11.2025).

2. Maven download page. *Apache.* URL: https://maven.apache.org/download.cgi (date of access 15.11.2025).

3. Heckler, M. Spring Boot: Up and Running: Building Cloud Native Java and Kotlin Applications. USA, Sebastopol : O'Reilly Media, 2021. 328 с.

4. Walls C. Spring in Action. Київ : Manning Publications, 2018. 520 с.

5. Spring Framework Documentation. *Spring.* URL: https://docs.spring.io/spring-framework/docs/ (date of access 12.11.2025).

6. Spring Boot. *Spring.* URL: https://spring.io/projects/spring-boot (date of access 11.11.2025).

7. Spring Data JPA. *Spring.* URL: https://spring.io/projects/spring-data-jpa (date of access 9.11.2025).

8. Jon Duckett. HTML & CSS: Design and Build Websites. Published by Wiley, 2011. 512 p.

9. JavaScript introduction. *Mozilla.* URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction (date of access 14.11.2025).

10. David Flanagan. JavaScript: The Definitive Guide: Master the World's Most-Used Programming Language 7th Edition. Published by O'Reilly Media, 2020. 706 p.

11. PostgreSQL documentation. *Postgresql.* URL: https://www.postgresql.org/docs/current (date of access 8.11.2025).

12. Шаховська Н. Б., Пасічник В. В. Сховища та простори даних: монографія. Львів: Видавництво Львівської політехніки, 2009. 244 с.

13. LiqPay документація. *Liqpay.* URL: https://www.liqpay.ua/doc/api/internet_acquiring /checkout?tab=1 (дата звернення 5.11.2025).

14. MVC pattern description. *Geeksforgeeks.* URL: https://www.geeksforgeeks.org/mvc-design-pattern/ (date of access 8.11.2025).

15. Class diagram definition. *Geeksforgeeks.* URL: https://www.geeksforgeeks.org/unified-modeling-language-uml-class-diagrams/ (date of access 3.11.2025).

16. Endpoint meaning. *Hubspot.* URL: https://blog.hubspot.com/website/api-endpoint (date of access 20.11.2025).

17. DDNS description. *Amazon.* URL: https://aws.amazon.com/ru/what-is/dynamic-dns/ (date of access 18.11.2025).

18. Spring Security Documentation. *Spring.* URL: https://docs.spring.io/spring-security/reference/ (date of access 5.11.2025).

19. OAuth 2.1 Authorization Framework. *Oauth.* URL: https://oauth.net/2.1/ (date of access 1.11.2025).

20. Richardson C. Microservices Patterns: With examples in Java. New York: Manning Publications, 2019. 520 c.

21. Kubernetes Documentation. *Kubernetes.* URL: https://kubernetes.io/docs/home (date of access 19.11.2025).

22. OpenAPI Specification. *Swagger.* URL: https://swagger.io/specification/ (date of access 18.11.2025).

23. Nginx Documentation. *Nginx.* URL: https://nginx.org/en/docs/ (date of access 17.11.2025).

24. OWASP API Security Top 10. *Owasp.* URL: https://owasp.org/API-Security/ (date of access 15.11.2025).

25. Stripe API Reference. *Stripe.* URL: https://stripe.com/docs/api (date of access 5.11.2025).

26. Cloud Design Patterns — Microsoft Azure. *Microsoft.* URL: https://learn.microsoft.com/en-us/azure/architecture/patterns/ (date of access 6.11.2025).

27. Newman S. Building Microservices: Designing Fine-Grained Systems. O'Reilly Media, 2021. 412 c.

ДОДАТОК А

Тези XXII міжнародної науково-практичної конференції молодих вчених та студентів «Сучасні проблеми наукового забезпечення енергетики»

УКР.НТУУ”КПІ ім. Ігоря Сікорського”_ІАТЕ_ЦТЕ_ТР-43мп

Аркушів 3

Київ — 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

СУЧАСНІ ПРОБЛЕМИ НАУКОВОГО ЗАБЕЗПЕЧЕННЯ ЕНЕРГЕТИКИ

У 2-х томах

Том 2

Матеріали XXII Міжнародної
науково-практичної конференції
молодих вчених і студентів

м. Київ, 22–25 квітня 2025 року

Київ – 2025

УДК 004.432

[1] Магістрант 1 курсу Середа В.О.
[1] Доц., к.ф.-м.н. Тарнавський Ю.А.
https://scholar.google.com.ua/citations?user=g3-qVSgAAAAJ&hl=en

[1] КПІ ім. Ігоря Сікорського

АВТОМАТИЗАЦІЯ ПРОЦЕСУ ЗАКУПІВЕЛЬ ЕНЕРГЕТИЧНОГО ОБЛАДНАННЯ

Постановка проблеми та її актуальність. Сучасні компанії, що працюють у сфері енергетики, стикаються з необхідністю ефективного управління процесами закупівель енергетичного обладнання. Традиційні методи закупівель, що базуються на паперовій або напівавтоматизованій обробці даних, мають ряд суттєвих недоліків, таких як затримки в узгодженні, помилки через людський фактор та низька прозорість процесів. Автоматизація закупівель здатна значно підвищити ефективність цього процесу, скоротити витрати та забезпечити більш прозоре управління ланцюгами постачання.

Недостатня автоматизація закупівель енергетичного обладнання призводить до значних втрат часу та ресурсів. Ручна обробка замовлень ускладнює контроль за виконанням угод, аналіз цінових пропозицій та управління постачальниками. Відсутність централізованої системи унеможливлює швидке узгодження та оптимізацію закупівельних процесів. На сучасному ринку компанії, що не використовують автоматизовані рішення, втрачають конкурентні переваги.

Формулювання мети. Метою даної роботи є розробка та впровадження веб-системи для автоматизації закупівель енергетичного обладнання, що дозволяє зменшити ризики людського фактору, підвищити швидкість та ефективність закупівельного процесу, а також забезпечити аналітичний контроль за витратами.

Основна частина. Автоматизована система закупівель реалізована у вигляді веб-додатку, що базується на таких технологіях:

- Мова програмування: Java.
- Фреймворк: Spring Framework (Spring Boot, Spring Data, Spring Session).
- Фронтенд: HTML, CSS, JavaScript.
- База даних: PostgreSQL.
- ORM: Jakarta Persistence API (JPA), Hibernate.
- Контейнеризація: Docker.

Система включає наступні функціональні можливості:

- Автоматизований процес обробки заявок на закупівлю.
- Інтеграція з базами даних постачальників для збору актуальних цінових пропозицій.
- Аналітичний модуль для оцінки ефективності закупівель.
- Контроль виконання замовлень та відстеження етапів поставки.
- Захищений доступ до системи з використанням механізмів аутентифікації та авторизації.

Для забезпечення зручної та безпечної оплати було реалізовано інтеграцію зі стороннім сервісом LiqPay. Даний сервіс надає можливість здійснювати платежі через банківські картки, Apple Pay та Google Pay. Інтеграція виконана за допомогою REST API LiqPay, що дозволяє автоматично генерувати платіжні посилання та відстежувати статус транзакцій. Реалізовано механізм шифрування даних для безпечної передачі платіжної інформації. Успішні транзакції автоматично додаються до історії платежів користувача, а в разі помилки система сповіщає його про необхідність повторної спроби. На рис.1 зображено вікно оплати замовлення.

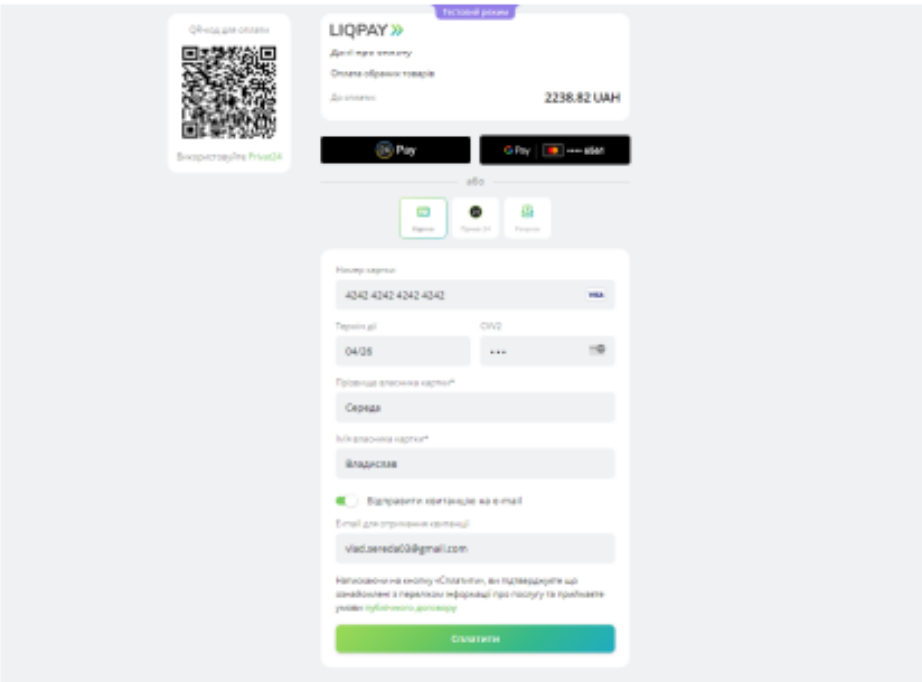

Рисунок 1 – Вікно оплати товарів

Впровадження даної системи надає цілий ряд переваг. По-перше, значно скорочуються витрати часу на узгодження, обробку та виконання замовлень завдяки автоматизації рутинних процесів, що дозволяє співробітникам зосередитися на стратегічно важливих завданнях. По-друге, підвищення точності обробки даних щодо закупівель мінімізує ризик помилок, що можуть виникнути внаслідок людського фактора, і забезпечує більш якісний аналіз витрат. Крім того, централізоване управління закупівлями не тільки сприяє прозорості операцій, а й дозволяє компанії максимально ефективно використовувати фінансові ресурси, завдяки чому забезпечується раціональне планування і контроль витрат. Це також створює умови для більш тісної співпраці з надійними постачальниками, що сприяє довгостроковій стабільності бізнесу.

Висновки. Отже, можна зробити висновок, що автоматизація закупівель енергетичного обладнання є ключовим фактором цифрової трансформації підприємств енергетичного сектору. Вона забезпечує не лише підвищення конкурентоспроможності завдяки ефективнішому управлінню ресурсами, але й сприяє значній оптимізації витрат та прискоренню процесів прийняття рішень. Така автоматизація дозволяє компаніям швидше адаптуватися до змін на ринку, впроваджувати інновації та підвищувати якість обслуговування клієнтів, що в свою чергу забезпечує стійкий розвиток галузі в цілому.

Перелік посилань:

1. The future of energy procurement and energy trading: change, trends and the role of IT service providers. URL: https://www.adesso.de/en/news/blog/the-future-of-energy-procurement-and-energy-trading-change-trends-and-the-role-of-it-service-providers.jsp (дата звернення 28.02.2025).
2. Spring Framework Documentation. URL: https://spring.io (дата звернення 28.02.2025).
3. PostgreSQL Official Documentation. URL: https://www.postgresql.org/docs (дата звернення 28.02.2025).
4. Docker Documentation. URL: https://docs.docker.com (дата звернення 28.02.2025).