ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Онлайн-магазин на базі технології Spring Boot: проектування та реалізація»

на здобуття освітнього ступеня бакалавра
зі спеціальності 126 Інформаційні системи та технології
(код, найменування спеціальності)
освітньо-професійної програми 126 Інформаційні системи та технології
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Максим ЛІНСЬКИЙ
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач вищої освіти гр. ІСД-43
Максим ЛІНСЬКИЙ
Ім'я, ПРІЗВИЩЕ

Керівник: Вікторія ЖЕБКА, д. т. н., професор
науковий ступінь, вчене звання
Ім'я, ПРІЗВИЩЕ

Рецензент: _____
науковий ступінь, вчене звання
Ім'я, ПРІЗВИЩЕ

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інформаційних систем та технологій
Ступінь вищої освіти бакалавр
Спеціальність 126 Інформаційні системи та технології
Освітньо-професійна програма 126 Інформаційні системи та технології

ЗАТВЕРДЖУЮ
Завідувач кафедрою ІСТ
_____Каміла СТОРЧАК
«____» _____2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Лінський Максим Олександрович
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Онлайн-магазин на базі технології Spring Boot: проектування та реалізація
керівник кваліфікаційної роботи Вікторія ЖЕБКА, д. т. н., професор,
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. No 56.

2. Строк подання кваліфікаційної роботи «30» травня 2025 р.

3. Вихідні дані до кваліфікаційної роботи:
   1. Науково-технічна література по технології Spring Boot.
   2. Офіційна документація Java, Spring Boot, MySQL, HTML.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
   1. Огляд технічних особливостей технології Spring Boot
   2. Архітектури системи та бази даних онлайн-магазину
   3. Розробка прототипу онлайн-магазину на основі технології Spring Boot

5. Перелік ілюстративного матеріалу: *презентація*
6. Дата видачі завдання «24» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	01.03.2025 р.	
2	Дослідження існуючих систем керування транспортним трафіком у розумних містах	15.03.2025 р.	
3	Проектування архітектури системи та бази даних	17.03.2025 р.	
4	Реалізація функціональних можливостей онлайн-магазину	4.04.2025 р.	
5	Тестування та виправлення помилок у роботі онлайн-магазину	16.05.2025 р.	
6	Розробка демонстраційних матеріалів	26.05.2025 р.	
7	Попередній захист дипломної роботи	27.05.2025 р.	
8	Подання роботи в деканат	30.05.2025 р.	

Здобувач вищої освіти _____ Максим ЛІНСЬКИЙ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Вікторія ЖЕБКА
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 77 стор., 92 рис., 20 джерел.

Мета роботи: розробка прототипу онлайн-магазину на основі технології Spring Boot, що забезпечує ефективну взаємодію між клієнтом та сервером у клієнт-серверній архітектурі.
Об'єкт дослідження: веб-додаток, створений із використанням фреймворку Spring Boot.
Предмет дослідження: програмна архітектура та реалізація функціональних можливостей онлайн-магазину.
Короткий зміст роботи:
1. У першому розділі було проведено огляд технології Spring Boot, переваги цієї технології та особливості інтеграції з іншими фреймворками як Spring Security, JPA, JDBC, MySQL.
2. Другий розділ описує архітектуру онлайн-магазину, що реалізована за моделлю "клієнт-сервер", дворівневу структуру системи, складові її компонентів як контролери, репозиторії та сервіси.
3. Третій розділ представляє процес створення прототипу онлайн-магазину, налаштування середовища розробки, набір використаних інструментів, а саме – Spring Initializr, Maven, Java. Розглянуто головні функціональні можливості: автентифікація користувачів, адміністрування замовлень, керування товарами, оформлення замовлень та робота бази даних.

КЛЮЧОВІ СЛОВА: SPRING BOOT, SPRING INITIALIZR, MAVEN, JPA, MYSQL, JAVA.

ABSTRACT

Text part of the qualification work for obtaining a bachelor's degree: 77 pages, 92 figures, 20 sources.

The purpose of the work: to develop a prototype of an online store based on Spring Boot technology, which ensures effective interaction between the client and the server in the client-server architecture.

Object of study: a web application created using the Spring Boot framework.

Subject of research: software architecture and implementation of the functionality of an online store.

Summary of the work:

1. In the first section, an overview of Spring Boot technology, the advantages of this technology and the features of integration with other frameworks such as Spring Security, JPA, JDBC, MySQL was conducted.
2. The second section describes the architecture of the online store, which is implemented according to the "client-server" model, the two-level structure of the system, the components of its components as controllers, repositories and services.
3. The third section presents the process of creating a prototype of an online store, setting up the development environment, a set of tools used, namely Spring Initializr, Maven, Java. The main functionalities are considered: user authentication, order administration, product management, ordering and database operation.

KEYWORDS: SPRING BOOT, SPRING INITIALIZR, MAVEN, JPA, MYSQL, JAVA.

ЗМІСТ

ВСТУП

Сучасна веб-розробка потребує гнучких, масштабних та надійних рішень, що можуть бути адаптовані до будь-яких стрімких змін у IT сфері. Spring Boot набуває особливого значення як інструмент, що сприяє спрощенню створення мікросервісних систем. Маючи потужну продуктивність, зручну модель структури та легкість обслуговування, ця технологія посідає одне із провідних місць серед інших інструментів, що використовуються професійними розробниками у фінансовому, медичному та логістичному секторах для комерційного програмного забезпечення.

Від інших популярних технологій, таких як Node.Js чи Django, Spring Boot відрізняється більшою стабільністю роботи системи.

Технологія Spring Boot – це фреймворк Java, який використовується для програмування програм, що працюють автономно. Він спрощує і прискорює розробку веб-застосунків та різних мікросервісних, комерційних додатків на базі Spring Framework. Надає інструменти для максимально швидкого старту проєкту, мінімізує будь-які конфігурування та інтеграції з основним фреймворком Spring.

Метою роботи є розробка прототипу онлайн-магазину із використанням технології Spring Boot, який надає ефективну взаємодію між клієнтами, користувачами та сервером у клієнт-серверній архітектурі.

Об'єктом дослідження є веб-застосунок, що розроблявся за допомогою технології Spring Boot.

Предметом дослідження виступає програмна архітектура із реалізацією функціональних можливостей онлайн-магазину.

Апробація дослідження здійснювалась у формі участі в II всеукраїнській науково-технічній конференції «Технологічні горизонти: дослідження та застосування інформаційних технологій для технологічного прогресу України і світу» з поданням тез на тему «Використання Spring Boot для прискорення розробки веб-додатків за допомогою автоконфігурації та вбудованих інструментів».

1 ОГЛЯД ТЕХНІЧНИХ ОСОБЛИВОСТЕЙ ТЕХНОЛОГІЇ SPRING BOOT ДЛЯ РОЗРОБКИ ВЕБ-РЕСУРСІВ

1.1 Основні концепції та переваги використання технології Spring Boot у веб-розробці

Однією з ключових переваг використання Spring Boot є його надійна система. Spring Boot створено на основі популярного середовища Spring, яке надає широкий спектр функцій та інструментів для створення програм корпоративного рівня. За допомогою Spring Boot розробники можуть скористатися перевагами попередньо налаштованих шаблонів, автоматичної конфігурації та безлічі вбудованих бібліотек, які спрощують процес розробки.

Spring Boot пропонує бездоганну інтеграцію з іншими проектами Spring, такими як Spring Security для автентифікації та авторизації, Spring Data для доступу до даних і Spring Cloud для створення хмарних програм. Ця інтегрована екосистема дозволяє розробникам зосередитися на написанні коду та створенні функцій, а не на налаштуванні та налаштуванні інфраструктури.

Найпопулярнішим вибором для платформи виконання є Spring Boot із 83%(рис 1.1). Spring Boot вже кілька років є кращим фреймворком Java. Це пов’язано, принаймні частково, із збільшенням впровадження мікросервісів для програм Java за останні роки. З 2% від загальної кількості респондентів Spring посіла друге місце за популярністю, а респонденти, які не використовують платформу виконання, також повідомили про 2% від загальної кількості. Респонденти, які використовують Dropwizard, Micronaut (одну з найновіших технологій Java, яка набирає обертів), Vert.x або користувальницькі платформи, отримали по 1% кожного.

Більше 86% респондентів працювали зі Spring (рис 1.2). 51% респондентів повідомили про роботу з такими технологіями збереження, як Hibernate, OpenJPA або EclipseLink. 27% повідомили про використання серверних технологій візуалізації, таких як JSP, JSF, Thymeleaf, FreeMarker або GWT. 26% повідомили

про використання таких технологій JAX-RS або JAX-WS, як Jersey, RESTEasy, CXF або Axis. Для реактивних фреймворків 11% респондентів повідомили про використання таких технологій, як Vert.x, Akka, RxJava або Project Reactor. Нарешті, 7% респондентів повідомили про використання корпоративних JavaBeans у своєму основному проекті[1].

Ці опитування свідчать про силу та універсальність Spring Boot.

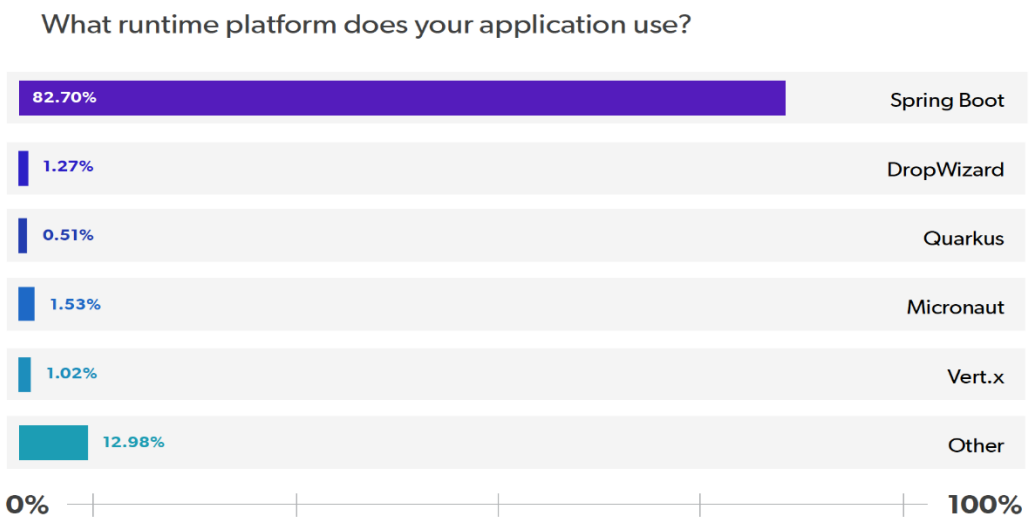


Рис. 1.1 Опитування проведене RebelLabs та JRebel на найпопулярнішу платформу[1]

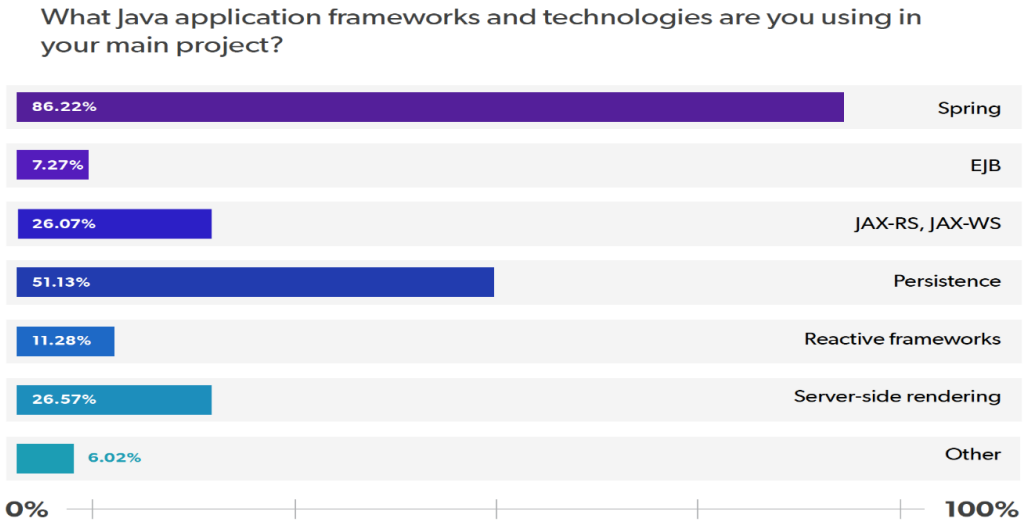


Рис. 1.2 Опитування проведене ReberLabs та JRebel на найпопулярніший фреймворк[1]

Другою перевагою використання Spring Boot для веб-розробки є підвищення продуктивності. Функція автоматичної конфігурації Spring Boot позбавляє від необхідності ручного налаштування та конфігурації, дозволяючи розробникам швидко розпочати створення додатків, не загрузаючи у виснажливих завданнях налаштування. Завдяки потужному інтерфейсу командного рядка (CLI) Spring Boot розробники можуть легко створювати нові проекти, додавати залежності та запускати тести за допомогою лише кількох простих команд. Цей спрощений робочий процес прискорює час розробки та дозволяє швидко створювати прототипи та ітерації, допомагаючи командам швидше розробляти високоякісне програмне забезпечення.

У дослідженні, проведеному Stack Overflow, популярною онлайн-спільнотою розробників, 72% респондентів назвали підвищення продуктивності основною перевагою використання Spring Boot для своїх проектів. Простота використання та ефективність фреймворку роблять його найкращим вибором для розробників, які хочуть створювати складні веб-програми з мінімальними зусиллями.

Також, однією з сильних сторін використання Spring Boot для веб-розробки є потужна підтримка спільноти. Спільнота Spring є однією з найбільших і найактивніших в Java, з живим онлайн-форумом, великою документацією та регулярними оновленнями та випусками. Розробники, які використовують Spring Boot, можуть використати знання та досвід спільноти для вирішення проблем, обміну найкращими практиками та бути в курсі останніх розробок, легко знаходити рішення типових проблем, брати участь у форумах і обговореннях, а також мати доступ до великої кількості ресурсів і документації, які допоможуть їм розпочати роботу з фреймворком. Велика документація, надана командою Spring, охоплює все, починаючи від базових концепцій і закінчуючи складними темами, що полегшує розробникам вивчення та опанування фреймворку. Це середовище для співпраці сприяє інноваціям і навчанню, забезпечуючи розробникам ресурси та підтримку, необхідні для досягнення успіху.

Відповідно до звіту RedMonk, провідної аналітичної компанії, що спеціалізується на технологіях розробників, Spring Boot визнано одним із

найкращих фреймворків Java на основі залучення спільноти та популярності (рис. 1.3). Активна та підтримуюча спільнота, що оточує Spring Boot, є цінним активом для розробників, які прагнуть створювати масштабовані веб-програми, які можна підтримувати[3].

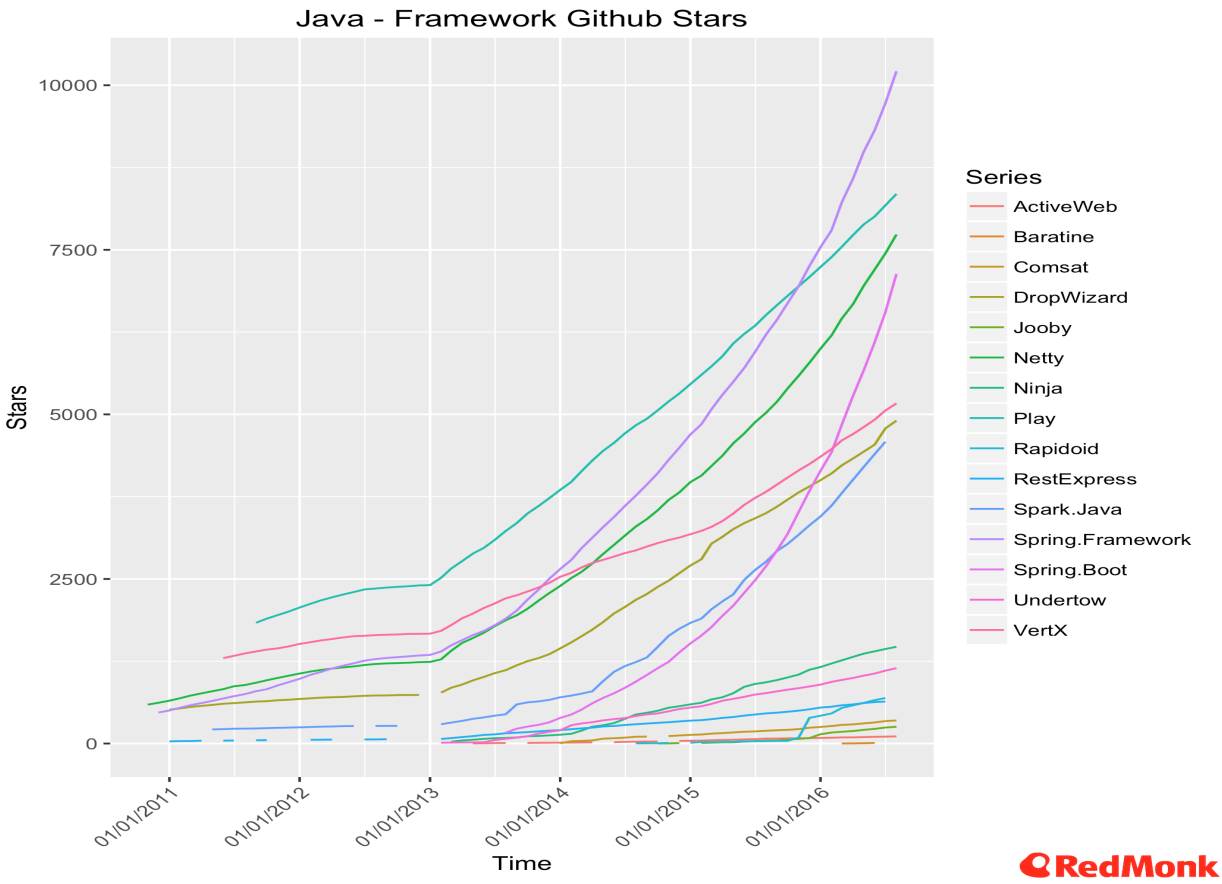


Рис. 1.3 Дані зі звіту RedMonk про фреймворки та їх популярність[3]

Spring Boot відомий своєю швидкістю та легкістю, що робить його ідеальним вибором для створення мікросервісів і хмарних програм. Фреймворк розроблений мінімалістичним та ефективним, має невелику площу та швидкий час запуску. Використовуючи Spring Boot, розробники можуть створювати автономні програми, які не вимагають окремого сервера програм для запуску. Ця гнучкість дозволяє легко розгортати та масштабувати, полегшуючи адаптацію до мінливих вимог бізнесу та вимог користувачів.

Ключова перевага використання Spring Boot є спрощений процес розробки. З його допомогою розробники можуть швидко налаштувати новий проект і почати

кодування, не витрачаючи час на налаштування проекту. Spring Boot надає низку функцій із коробки, включаючи автоматичне налаштування, яке допомагає зменшити шаблонний код і прискорити розробку. Крім того, Spring Boot дотримується принципу конвенції над конфігурацією, що означає, що розробникам потрібно надати конфігурацію лише для тих частин програми, які відрізняються від налаштувань за замовчуванням. Це спрощує процес розробки та дозволяє розробникам зосередитися на написанні коду, а не на налаштуванні проекту. Spring Boot надає ряд вбудованих серверів, включаючи Tomcat, Jetty і Undertow, що дозволяє розробникам легко пакувати свої програми як автономні файли JAR і розгортати їх без необхідності зовнішнього сервера. Це допомагає зменшити складність розгортання та полегшує розробникам розгортання своїх програм у різних середовищах.

Spring Boot спрощує процес розробки, надаючи набір стандартних конфігурацій і потужний набір інструментів для створення веб-додатків. Він дотримується принципу «кодування за домовленістю», що означає, що розробники можуть дотримуватися певних шаблонів кодування та найкращих практик без необхідності налаштовувати все вручну. Це забезпечує більш спрощений і ефективний процес розробки, дозволяючи розробникам зосередитися на створенні надійних і масштабованих програм.

Безпека є головним пріоритетом для веб-додатків, і Spring Boot надає вбудовані функції безпеки, які допомагають розробникам захистити свої програми від загроз і вразливостей. За допомогою Spring Security розробники можуть реалізувати автентифікацію, авторизацію та безпечні протоколи зв’язку лише за допомогою кількох рядків коду. Це гарантує, що конфіденційні дані захищені, а програма відповідає галузевим стандартам і нормам.

Spring Boot пропонує початкові POM (об’єктну модель проекту), тобто файл pom.xml, який спрощує конфігурацію збірки. Ці параметри за замовчуванням охоплюють усе: від зовнішнього сервера, який вбудовує сервери Tomcat, Jetty або Undertow, до налаштування JPA Spring. Він пропонує готові до використання

параметри для швидшого налаштування, також є можливість вручну додати залежності у файл «pom.xml» відповідно до необхідних вимог(рис. 1.4).

Рис. 1.4 Приклад використання готового шаблону POM[5]

З прикладу рисунку 1.4 – усе, що потрібно це знайти необхідні елементи та заповнити значення цих елементів.

Слід не забувати, що змінення налаштування можливе у будь-який час, коли це буде потрібно. Розробники можуть змінювати свої різні частини додатків, або ж компонентів програми.

Підсумовуючи, Spring Boot пропонує багато переваг для веб-розробки, включаючи спрощену розробку, легку конфігурацію та повну інтеграцію з іншими інструментами та фреймворками. Використовуючи можливості Spring Boot, розробники можуть легко створювати надійні та масштабовані веб-додатки.

Незалежно від того, початківець чи досвідчений розробник, Spring Boot може допомогти оптимізувати процес розробки та надати високоякісні рішення. Spring Boot став вибором для розробників, які прагнуть ефективно створювати сучасні веб-додатки. Завдяки спрощеному процесу розробки, легкому налаштуванню та можливостям повної інтеграції Spring Boot дає розробникам змогу зосередитися на створенні інноваційних рішень, а не турбуватися про складність процесу розробки.

1.2 Інтеграція Spring Boot із базами даних та іншими сервісами

Spring Boot має бездоганну інтеграцію з іншими інструментами та фреймворками, забезпечує інтеграцію з такими популярними інструментами, як Spring Security, JPA(Java/Jakarta Persistence API), JDBC(Java Database Connectivity) та Thymeleaf, що полегшує розробникам створення безпечних і багатофункціональних програм. Крім того, Spring Boot підтримує архітектуру мікросервісів, дозволяючи розробникам створювати масштабовані та гнучкі програми, розбиваючи їх на менші служби, які можна розгортати незалежно, надає низку плагінів для популярних інструментів збірки, таких як Maven і Gradle, які допомагають автоматизувати процес збирання та полегшують розробникам керування залежностями. Це спрощує робочий процес розробки та дозволяє розробникам зосередитися на написанні коду, а не на управлінні збіркою проекту.

Однією із найбільш використовуваних баз даних є – MySQL. Інтеграція бази даних MySQL у програму на платформі Spring Boot починається з налаштування проекту та включення необхідних залежностей. Завдяки Spring Boot цей процес значно спрощений, оскільки фреймворк пропонує інтуїтивно зрозумілий підхід до підключення всіх необхідних компонентів для взаємодії з MySQL. Щоб розпочати, можна скористатися Spring Initializr — веб-інструментом, який допомагає створити основу проекту. Через цей інструмент можна вибрати між Maven чи Gradle, визначити мову програмування (у цьому випадку Java), вибрати останню стабільну версію Spring Boot, а також додати залежності, необхідні для роботи з базою даних, такі як Spring Web, Spring Data JPA та драйвер MySQL. Після налаштування параметрів проекту можна створити архівний файл, завантажити його та розпакувати, отримавши структуру проекту, готову до подальшої розробки. Цей процес дозволяє швидко налаштувати проект для інтеграції з MySQL, що є важливим кроком для розробки сучасних веб-додатків.

JDBC — це Java API, який забезпечує стандартний спосіб взаємодії з реляційними базами даних. Він дозволяє розробникам виконувати запити SQL, виконувати операції CRUD (Створення, читання, оновлення, видалення) і керувати

транзакціями бази даних. Щоб працювати з JDBC у програмі Spring Boot потрібно включити відповідні залежності у файл pom.xml проєкту (рис. 1.5).

```
<!-- JDBC Dependencies -->
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-jdbc</artifactId>
    </dependency>
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
    </dependency>
</dependencies>
```

Рис. 1.5 Залежності JDBC

JDBC спочатку задумувався як API на стороні клієнта, що дозволяє клієнту Java взаємодіяти з джерелом даних. Це змінилося з JDBC 2.0, який включав додатковий пакет, що підтримує підключення JDBC на стороні сервера. Відтоді кожен новий випуск JDBC містив оновлення як для пакета на стороні клієнта (java.sql), так і пакета на стороні сервера (javax.sql). JDBC 4.3, остання версія на момент написання цієї статті, була випущена як частина Java SE 9 у вересні 2017 року під назвою JSR 221.

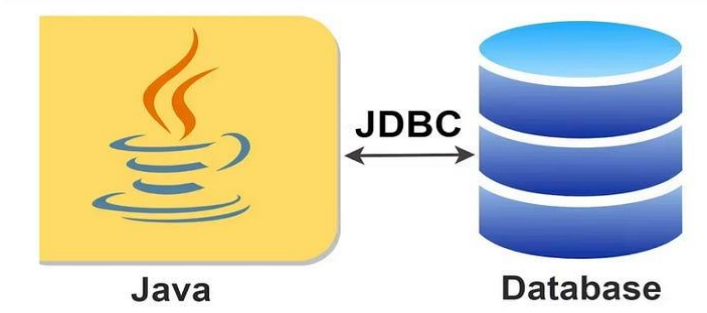


Рис. 1.6 Робота Java з базою даних через JDBC

Як розробник, можна використовувати JDBC для взаємодії з базою даних у програмі Java. JDBC діє як міст між вашим кодом і базою даних, як показано на рис. 1.7.

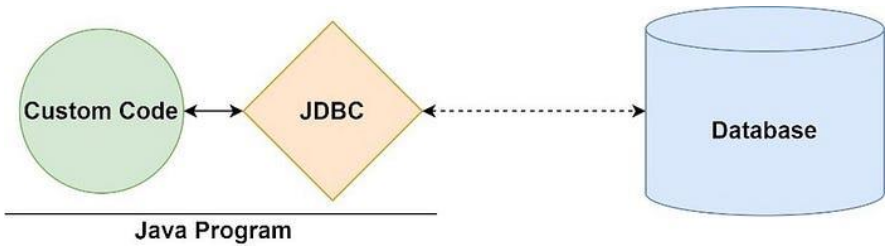


Рис. 1.7 JDBC підключає програми Java до баз даних

Інтерфейс JDBC складається з двох основних рівнів. Перший рівень, JDBC API, забезпечує взаємодію між програмою, написаною на мові Java, та менеджером JDBC. Другий рівень, JDBC Driver, відповідає за зв'язок між менеджером JDBC та драйвером конкретної бази даних(рис. 1.8).

Протягом останніх років API JDBC і драйвер JDBC зазнали значних удосконалень, що сприяло їх трансформації у багатофункціональну, продуктивну та надійну бібліотеку для роботи з базами даних.

JDBC представляє собою універсальний інтерфейс API, який використовується для взаємодії з базами даних на рівні програмного коду. У підґрунті цієї архітектури розташований драйвер, сумісний із JDBC, який слугує містком між API та конкретною системою управління базами даних, забезпечуючи ефективну інтеграцію.

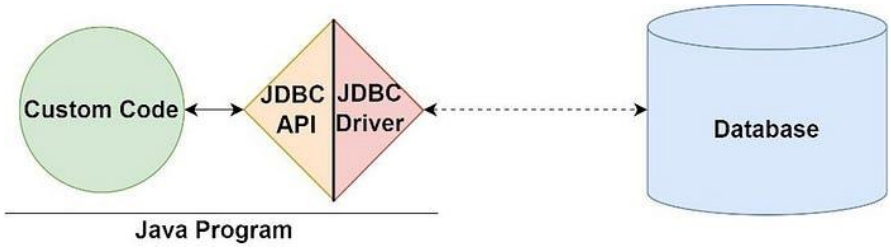


Рис. 1.8 JDBC архітектура

Програмістам додатків, як правило, немає необхідності одразу занурюватися в деталі реалізації драйвера JDBC, за умови, що використовується безпечний та офіційний драйвер. Проте для глибшого розуміння архітектури додатка та оптимізації продуктивності корисно знати, що існують чотири основні типи драйверів JDBC:

1. Драйвер моста JDBC-ODBC — це тонкий рівень Java, який використовує драйвер ODBC як основу для взаємодії з базою даних.

2. Власний драйвер API — забезпечує інтерфейс між Java та рідним клієнтом бази даних, використовуючи специфічний API бази даних.

3. Драйвер проміжного програмного забезпечення — функціонує як універсальний інтерфейс («проміжне програмне забезпечення»), який забезпечує взаємодію між Java та протоколом постачальника RDBMS (реляційної системи управління базами даних).

4. Чистий драйвер Java — реалізує специфічний протокол постачальника бази даних безпосередньо в Java, забезпечуючи незалежність від платформних компонентів.

Під час проєктування архітектури додатка та аналізу його продуктивності доцільно враховувати тип драйвера JDBC, що використовується. Вибір оптимального драйвера може суттєво вплинути на ефективність взаємодії додатка з базою даних та спрощення процесу розробки.

Jakarta Persistence API (раніше Java Persistence API) стосується стійкості, що у загальній формі означає будь-який механізм, за допомогою якого об’єкти Java переживають процес програми, який їх створив. Не всі об’єкти Java потрібно зберігати, але більшість програм зберігають ключові бізнес-об’єкти. Специфікація JPA дозволяє визначити, які об’єкти мають зберігатися та як вони зберігаються у ваших програмах Java.

Сам по собі JPA не є інструментом або структурою; скоріше, він визначає набір концепцій, якими керуються виконавці. Хоча модель об’єктно-реляційного відображення (ORM) JPA спочатку базувалася на Hibernate, згодом вона розвивалася. Подібним чином, хоча JPA спочатку призначався для використання з реляційними базами даних, деякі реалізації JPA були розширені для використання зі сховищами даних NoSQL. Популярним фреймворком, який підтримує JPA з NoSQL, є EclipseLink, еталонна реалізація для JPA 3.

Основна ідея JPA на відміну від JDBC полягає в тому, що здебільшого JPA дозволяє уникнути необхідності «мислити реляційно». У JPA визначаються свої

правила збереження в області коду та об’єктів Java, тоді як JDBC вимагає ручного перекладу з коду в реляційні таблиці та назад. Кожна реалізація JPA забезпечує певний рівень ORM. Щоб зрозуміти JPA та сумісні з JPA інструменти потрібно добре розуміти ORM.

Об’єктно-реляційне відображення(ORM) – це завдання, яке розробники мають вагомі причини уникати виконання вручну. Такий фреймворк, як Hibernate ORM або EclipseLink, кодує це завдання в бібліотеку або фреймворк, рівень ORM. Як частина прикладної архітектури рівень ORM відповідає за керування перетворенням програмних об’єктів для взаємодії з таблицями та стовпцями в реляційній базі даних. У Java рівень ORM перетворює класи та об’єкти Java, щоб їх можна було зберігати та керувати ними в реляційній базі даних.

За замовчуванням ім’я об’єкта, що зберігається, стає ім’ям таблиці, а поля – стовпцями. Після налаштування таблиці кожен рядок таблиці відповідає об’єкту в програмі. Відображення об’єктів можна налаштувати, але стандартні налаштування, як правило, працюють, і, дотримуючись стандартних значень, Користувач уникає необхідності підтримувати метадані конфігурації.

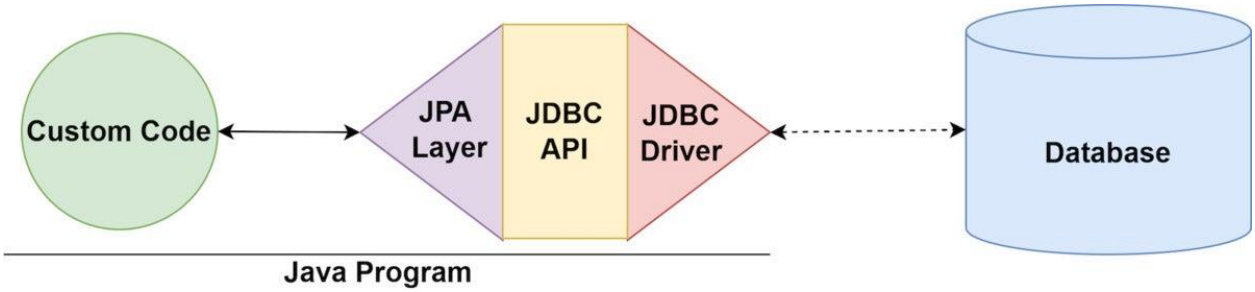


Рис. 1.9 Роль JPA та ORM у програмах

Налаштування рівня Java ORM:

Під час налаштування нового проєкту для використання Java Persistence API (JPA) необхідно здійснити конфігурацію сховища даних і постачальника JPA. Насамперед слід налаштувати конектор сховища даних, який забезпечує підключення до обраної бази даних, що може бути реляційною (SQL) або нереляційною (NoSQL). Окрім цього, потрібно інтегрувати та налаштувати

постачальника JPA — фреймворк, що реалізує специфікацію JPA, наприклад, Hibernate або EclipseLink. Незважаючи на те, що JPA можна налаштувати вручну, більшість розробників віддають перевагу використанню готової інтеграції, яку надає фреймворк Spring.

З точки зору програмування рівень Object-Relational Mapping (ORM) виконує функцію адаптера, перетворюючи мову графів об'єктів на мову SQL і реляційних таблиць. Рівень ORM дозволяє об'єктно-орієнтованим розробникам створювати програмне забезпечення, яке зберігає дані, зберігаючи об'єктно-орієнтовану парадигму без необхідності вручну опрацьовувати реляційні структури.

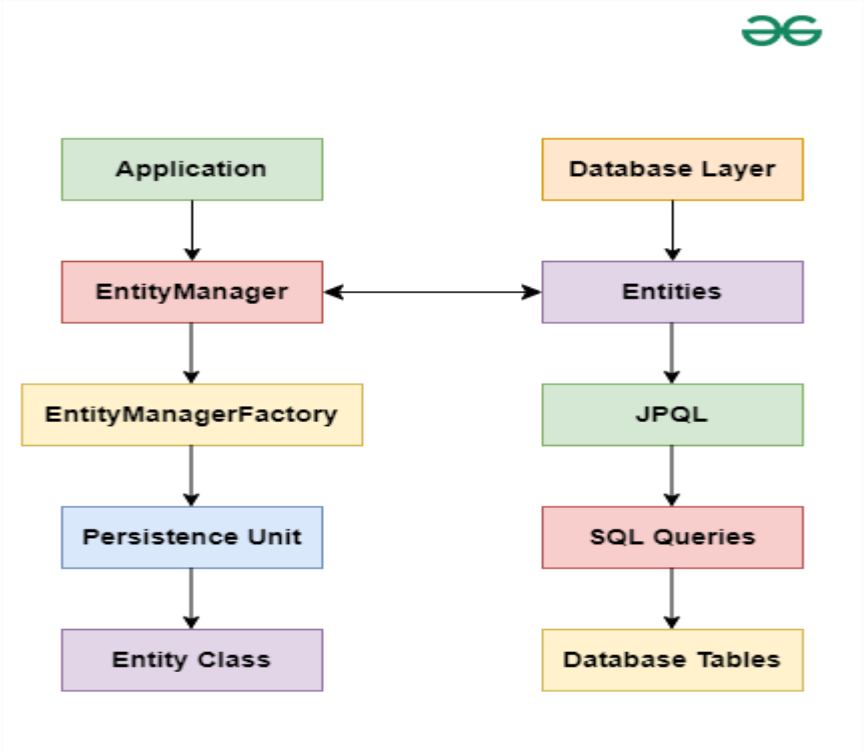


Рис. 1.10 Архітектура рівнів у JPA[8]

Користуючись Java Persistence API (JPA), розробник створює карту відображення сховища даних до об'єктів моделі даних програми. Замість того, щоб визначати специфіку зберігання та витягування об'єктів, програміст визначає відображення між об'єктами та базою даних, після чого використовує JPA для збереження цих об'єктів. У випадку з реляційною базою даних більша частина

фактичного зв'язку між кодом програми та базою даних обробляється через Java Database Connectivity (JDBC).

Як специфікація, JPA надає набір анотацій метаданих, які використовуються для визначення відображення між об'єктами програми та реляційною базою даних. Кожна реалізація JPA надає власний механізм обробки цих анотацій. Специфікація JPA також передбачає наявність PersistenceManager або EntityManager, які є основними точками взаємодії з JPA-системою. Ці компоненти служать інтерфейсами, через які бізнес-логіка програми взаємодіє з відображеними об'єктами, визначаючи операції, які мають бути виконані на них у контексті бази даних.

Кожна програма, яка працює з базою даних, повинна визначати прикладний рівень, єдиною метою якого є ізоляція постійного коду. Jakarta Persistence API надає низку можливостей і підтримку збереження об’єктів Java. Прості програми можуть не вимагати всіх можливостей JPA, а в деяких випадках накладні витрати на налаштування фреймворку можуть бути невиправданими. Однак у міру зростання програми структура та інкапсуляція JPA справді заслуговують на збереження. Використання JPA робить об’єктний код простим і надає звичайну структуру для доступу до даних у програмах Java.

Thymeleaf — це сучасний механізм шаблонів Java на стороні сервера, який наголошує на шаблонах HTML, які можна попередньо переглянути у браузері подвійним клацанням миші, що дуже корисно для самостійної роботи над шаблонами інтерфейсу користувача (наприклад, дизайнером) без необхідності запущений сервер (рис. 1.11).

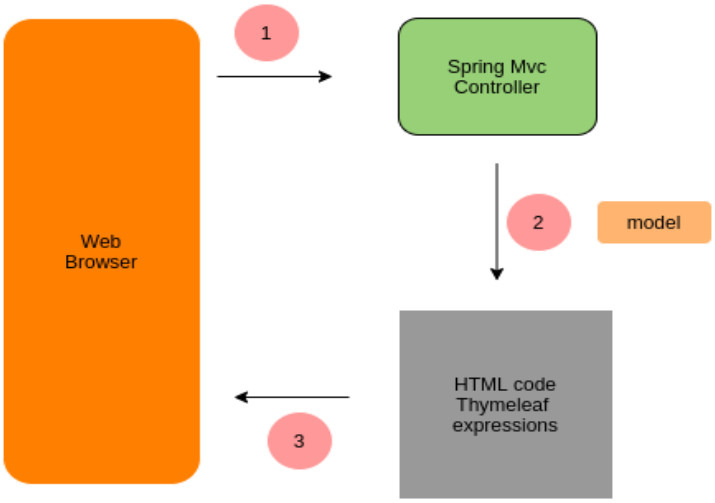


Рис. 1.11 Схематична робота Thymeleaf

Jakarta Server Pages (раніше JavaServer Pages) — це стандартна технологія Java, яку розробники використовують для створення динамічних веб-сторінок на основі даних для веб-програм Java. JSP побудовано на основі специфікації Java Servlet (також відомої як Jakarta Servlet) і є однією з веб-технологій Java, включених для постійної підтримки та оновлень у Jakarta EE.

JSP і сервлети зазвичай працюють разом, особливо в старих веб-додатках Java. З точки зору програмування, найочевидніша відмінність між JSP і сервлетами полягає в тому, що за допомогою сервлетів користувач пишете код Java, а потім вставляє розмітку клієнта (наприклад, HTML) у цей код. JSP починає зі сценарію або розмітки на стороні клієнта, а потім вставляє теги JSP, щоб підключити сторінку до серверної частини Java.

JSP використовується як спосіб написання розмітки з надзвичайними можливостями для взаємодії з серверною частиною. Зазвичай така розмітка, як HTML, надсилається клієнту, де вона взаємодіє з внутрішнім сервером через JavaScript. JSP попередньо обробляє HTML за допомогою спеціальних команд для доступу та використання можливостей сервера, а потім надсилає скомпільовану сторінку клієнту.

1.3 Особливості побудови REST API за допомогою Spring Boot

REST (Representational State Transfer) швидко став стандартом де-факто для створення веб-сервісів в Інтернеті завдяки своїй простоті у розробці та використанні. Цей підхід орієнтований на архітектурні принципи веб, що включають використання стандартних протоколів та методів взаємодії. Автором концепції REST є Рой Філдінг, що має значний внесок у розробку багатьох специфікацій, що регулюють роботу Інтернету, зокрема протоколу HTTP, що є основою для REST-сервісів.

Переваги REST у контексті веб-сервісів можна охарактеризувати через використання низки функцій, властивих веб та його основному протоколу. Серед них варто виділити такі ключові можливості:

1. Відповідні дії: REST API підтримує основні HTTP методи, такі як GET, POST, PUT, DELETE та інші, що забезпечує чітке розмежування операцій на сервері.
2. Кешування: можливість кешувати ресурси зменшує навантаження на сервери та підвищує продуктивність.
3. Переадресація та переадресація: REST дозволяє ефективно реалізувати редиректи, забезпечуючи додаткову гнучкість у маршрутизації запитів.
4. Безпека: використання стандартних механізмів безпеки, таких як шифрування та аутентифікація, критично важливо для створення надійних веб-сервісів.

Завдяки цим характеристикам архітектура REST дозволяє створювати стійкі та ефективні веб-системи. Однак це не єдині переваги, які пропонує REST. Архітектурна концепція REST, заснована на безлічі дрібних специфікацій, дозволяє легко масштабувати та адаптувати системи, уникаючи при цьому «війни стандартів» і забезпечуючи розвиток без значних обмежень. Розробники можуть застосовувати сторонні набори інструментів, які реалізують ці специфікації, що дозволяє швидко створювати як клієнтські, так і серверні рішення.

Створюючи REST API, можна досягти низки важливих цілей, зокрема:

1. Зворотна сумісність API: забезпечення збереження сумісності між різними версіями API.

2. Розширюваність API: можливість поступового розвитку та адаптації API до змінних вимог.

3. Масштабованість сервісів: побудова масштабованих веб-сервісів для обробки великої кількості запитів.

4. Безпека сервісів: використання перевірених методів для забезпечення безпеки даних та взаємодії.

5. Гнучкість між безстиковими та станними сервісами: можливість реалізації як безстикових (stateless), так і станних (stateful) сервісів, залежно від потреб проєкту.

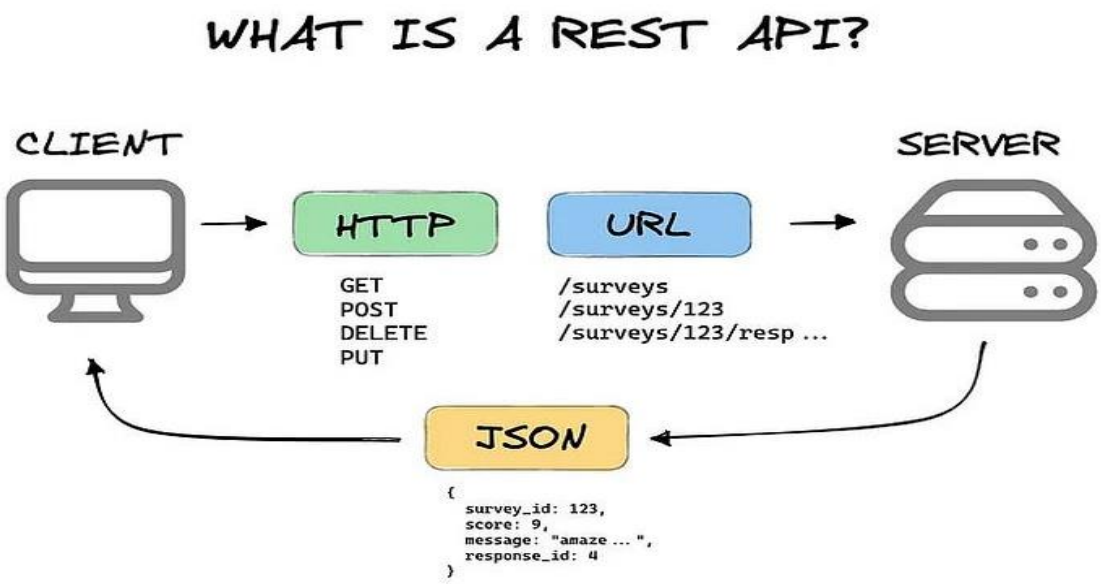


Рис. 1.12 Архітектура REST API[12]

Однак варто зазначити, що REST сам по собі не є стандартом, а радше є підходом або стилем архітектури, що обмежує певні аспекти взаємодії між компонентами системи. Ці обмеження дозволяють створювати веб-масштабні системи, які можуть ефективно взаємодіяти через HTTP, використовуючи переваги архітектури Інтернету.

У цьому контексті Spring Framework є потужним інструментом для розробки RESTful сервісів, забезпечуючи широкі можливості для реалізації переваг безстикових функцій REST.

Для REST API важлим буде знання клієнт-серверної архітектури (рис.1.13)

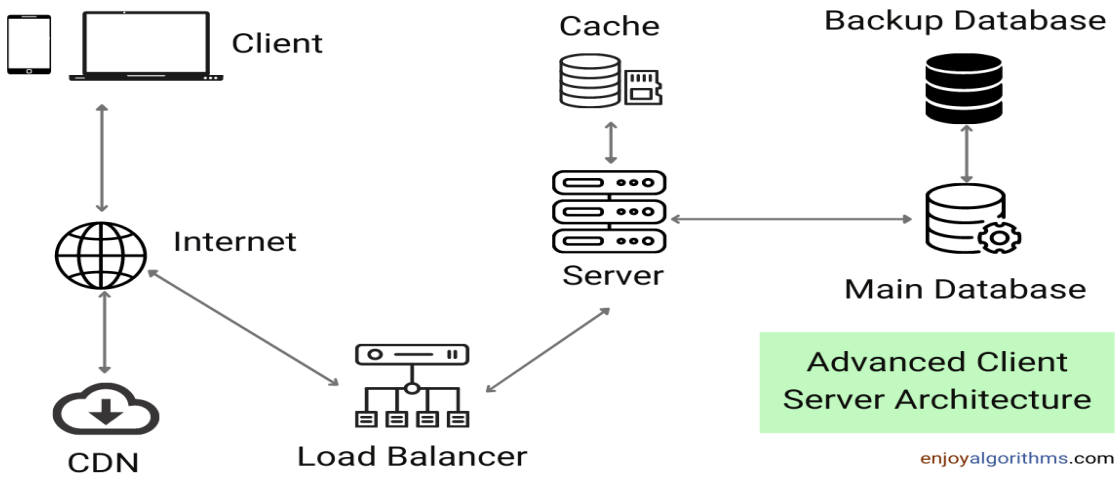


Рис. 1.13 Клієнт-серверна архітектура[12]

Така архітектура забезпечує розділення обов’язків між клієнтом і сервером, підвищуючи масштабованість і керованість системи.

2 АРХІТЕКТУРА СИСТЕМИ ТА СТРУКТУРА БАЗИ ДАНИХ ОНЛАЙН-МАГАЗИНУ НА ОСНОВІ ТЕХНОЛОГІЇ SPRING BOOT

2.1. Загальна архітектура онлайн-магазину: клієнт-серверна модель

Модель «Клієнт-сервер» — це розподілена структура програми, яка розподіляє завдання або робочі навантаження між постачальниками ресурсів або послуг, які називаються серверами, і запитувачами послуг, які називаються клієнтами (рис. 2.1). В архітектурі клієнт-сервер, коли клієнтський комп'ютер надсилає запит на дані на сервер через Інтернет, сервер приймає запитуваний процес і доставляє запитувані пакети даних назад клієнту. Клієнти не діляться своїми ресурсами. Прикладами клієнт-серверної моделі є електронна пошта, Всесвітня павутина. У сфері інформаційних технологій терміни «клієнт» і «сервер» мають специфічне значення, що відображає їхню роль у клієнт-серверній архітектурі.

Термін «клієнт» у загальному значенні означає особу або організацію, яка користується певною послугою. В інформаційних системах клієнтом називають комп'ютер (хост), що здійснює запити до сервера для отримання інформації або доступу до певних сервісів. Клієнт може бути як кінцевим користувачем, що взаємодіє з веб-застосунками через браузер, так і програмним забезпеченням, що надсилає запити через API.

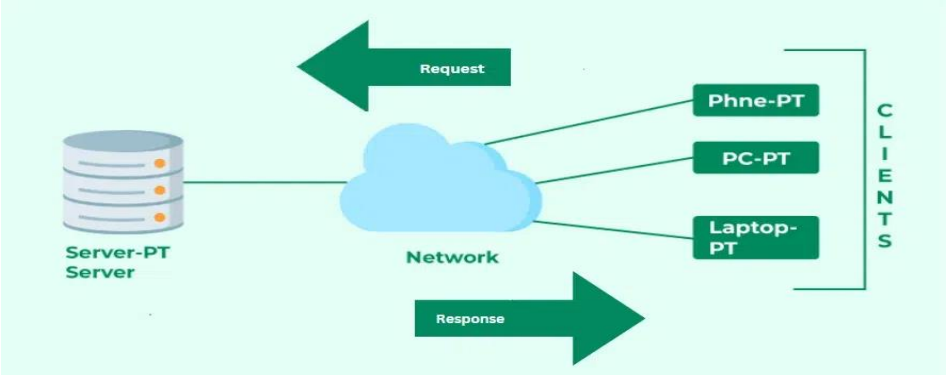


Рис. 2.1 Загальний приклад клієнт-серверної моделі[13]

Термін «сервер» загалом позначає особу або механізм, що забезпечує надання певної послуги. У цифровому контексті сервером називають віддалений комп’ютер або програмне забезпечення, яке обробляє запити клієнтів і надає їм необхідні дані чи сервіси. Сервери можуть виконувати різні функції, зокрема зберігання та обробку даних, автентифікацію користувачів, управління мережею або надання веб-сторінок.

Для здійснення взаємодії між веб-браузером та серверами необхідно виконати низку етапів, що забезпечують доступ користувача до веб-ресурсів:

1. Введення URL - адреси користувач вводить у веб-браузері уніфікований покажчик ресурсу (URL), який вказує на веб-сайт або конкретний файл у мережі.
2. Запит до DNS-сервера - після введення URL браузер надсилає запит до системи доменних імен (DNS) для отримання відповідної IP-адреси веб-сервера, на якому розміщено запитуваний ресурс.
3. Пошук DNS-сервера та отримання IP-адреси - DNS-сервер здійснює пошук доменного імені та повертає IP-адресу відповідного веб-сервера, що дозволяє браузеру встановити з’єднання.
4. Надсилання HTTP/HTTPS-запиту - браузер формує та надсилає HTTP або HTTPS-запит на отриману IP-адресу веб-сервера, запитуючи доступ до веб-сторінки або файлу.
5. Обробка запиту сервером - веб-сервер отримує запит, обробляє його та надсилає браузеру необхідні файли (HTML, CSS, JavaScript, зображення тощо), необхідні для коректного відображення веб-ресурсу.
6. Обробка та відтворення вмісту браузером - браузер отримує передані сервером файли та відтворює їх для користувача. Цей процес включає:
    - 6.1. Інтерпретацію DOM (Document Object Model) – структурування HTML-документу.
    - 6.2. Обробку CSS – застосування стилів для коректного відображення контенту.
    - 6.3. Виконання JavaScript-коду за допомогою JIT-компілятора (Just-In-Time), що оптимізує виконання скриптів.

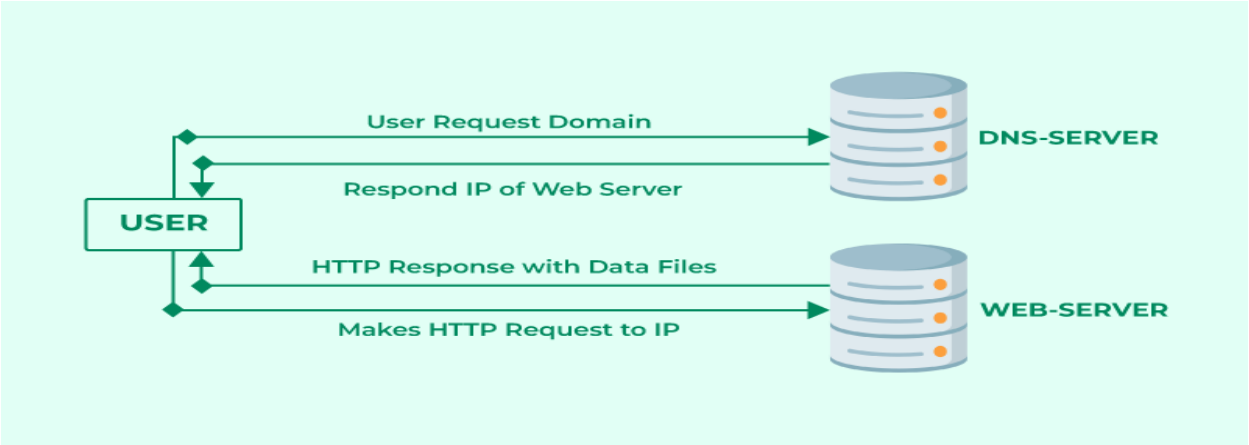


Рис. 2.2 Запит та відповідь клієнт-сервера[13]

Дворівнева архітектура - дворівнева клієнт-серверна архітектура є найпростішою формою взаємодії між клієнтом і сервером, за якої клієнтський пристрій безпосередньо обмінюється даними із сервером (рис. 2.3). У цій моделі клієнт надсилає запит безпосередньо на сервер, який здійснює його обробку та повертає відповідь. Такий підхід є ефективним для невеликих інформаційних систем, оскільки мінімізує затримку під час обміну даними та спрощує реалізацію взаємодії. Водночас значне збільшення кількості клієнтських запитів може спричинити підвищене навантаження на сервер, що знижує продуктивність системи та може вимагати додаткової оптимізації або масштабування.

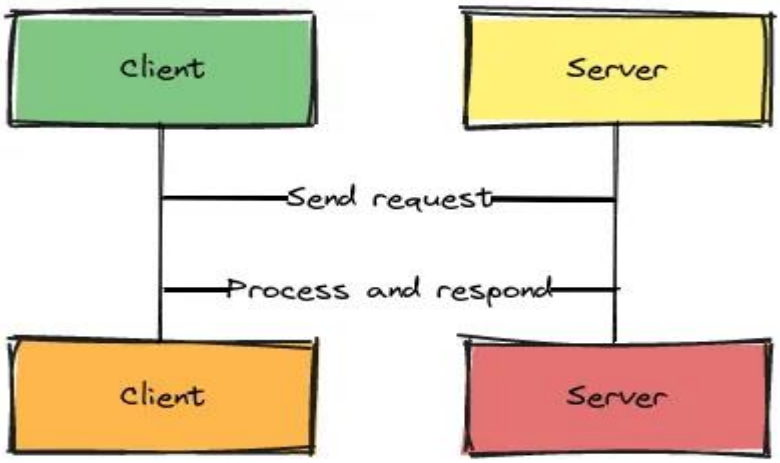


Рис. 2.3 Дворівнева архітектура [14]

Трирівнева архітектура - трирівнева архітектура передбачає наявність проміжного рівня (middleware) між клієнтом і сервером (рис. 2.4). Даний рівень виконує функції обробки бізнес-логіки, автентифікації користувачів, управління доступом або кешування даних, що дозволяє оптимізувати запити до бази даних та зменшити навантаження на основний сервер. Такий підхід підвищує масштабованість системи та її продуктивність, забезпечуючи розподілений обробіток інформації. Використання проміжного рівня також сприяє підвищенню рівня безпеки, оскільки обмежує прямий доступ клієнтів до критично важливих даних.

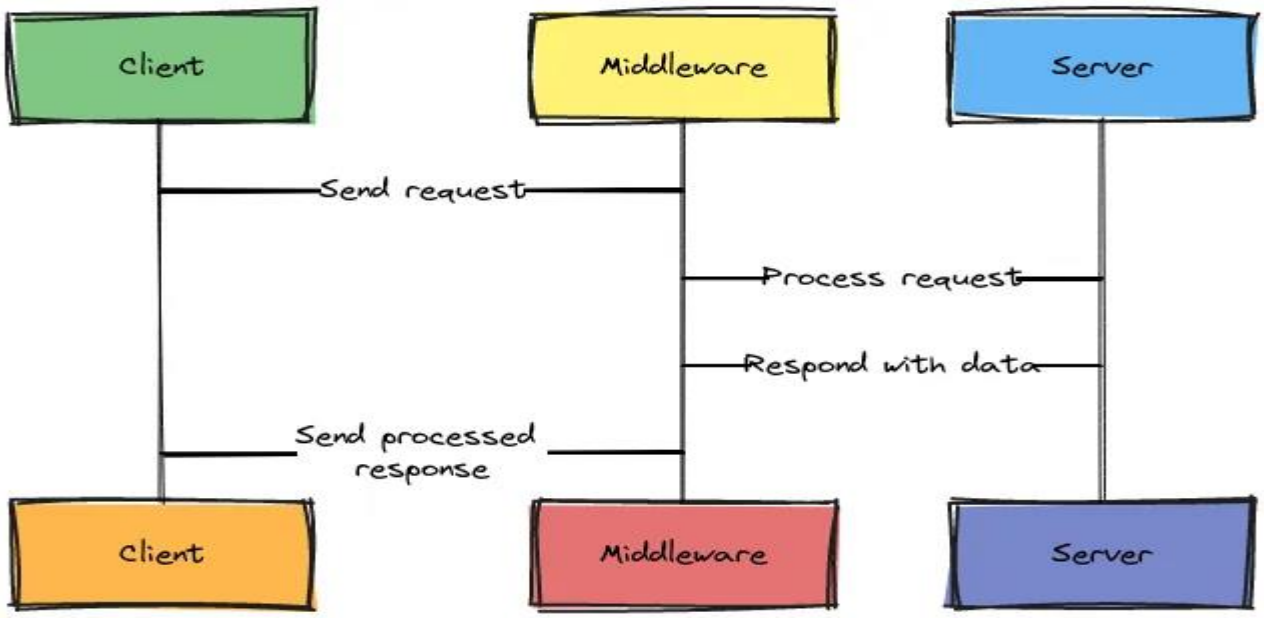


Рис. 2.4 Трирівнева архітектура [14]

Багаторівнева архітектура - багаторівнева клієнт-серверна модель передбачає наявність декількох проміжних рівнів, що забезпечують виконання спеціалізованих функцій, таких як управління безпекою, бізнес-логіка, балансування навантаження та обробка великих обсягів даних (рис. 2.5). Ця архітектура є оптимальним рішенням для високонавантажених інформаційних систем, оскільки дозволяє ефективно розподіляти запити між різними серверами, запобігаючи надмірному навантаженню на єдиний сервер.

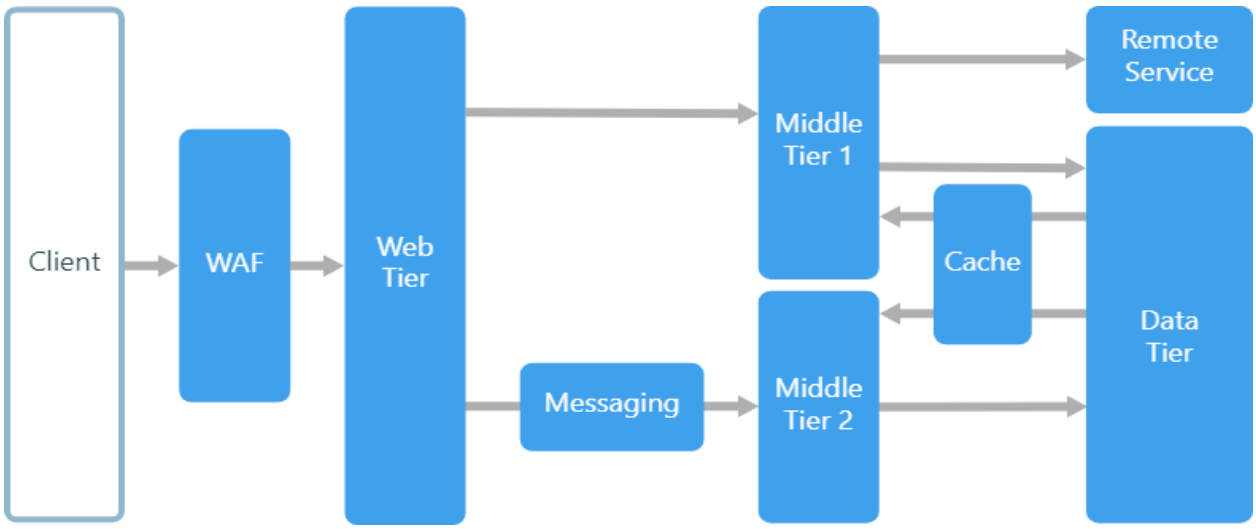


Рис. 2.5 Багаторівнева архітектура [15]

Зазначений підхід широко застосовується у великих корпоративних рішеннях, таких як банківські системи, платіжні платформи, електронна комерція та хмарні сервіси. Наприклад, у фінансових системах використання багаторівневої архітектури дозволяє забезпечити безпечну обробку транзакцій, контроль доступу до чутливих даних та ефективний розподіл навантаження між серверами.

Таким чином, вибір відповідної клієнт-серверної архітектури залежить від масштабності, продуктивності та вимог до безпеки конкретної інформаційної системи, а багаторівневі моделі є найбільш придатними для складних, високонавантажених середовищ.

Клієнт-серверна архітектура передбачає взаємодію між клієнтом (веб-браузером) та сервером, який обробляє запити та надає відповідь у вигляді веб-сторінки. Нижче представлено детальну структурну схему обміну даними між компонентами веб-додатку, що використовує Spring Boot як бекенд, FreeMarker як механізм відображення сторінок та MySQL як систему управління базами даних.

Основні етапи взаємодії клієнта та сервера:

1. Формування HTTP-запиту клієнтом:

- Користувач у браузері вводить URL-адресу веб-ресурсу або натискає на посилання, що ініціює HTTP-запит до веб-сервера.
- Запит надсилається за протоколом HTTP або HTTPS для отримання певних даних.

2. Передача запиту на рівень Frontend:

- Браузер отримує відповідь від сервера, яка містить HTML-код, стилі CSS та можливі клієнтські скрипти JavaScript.
- У даній архітектурі використовується шаблонізатор FreeMarker, який динамічно формує веб-сторінку, заповнюючи її відповідними даними.

3. Передача запиту на сервер (Backend):

- Коли користувач взаємодіє із веб-сторінкою (наприклад, відкриває список товарів), Frontend передає запит до контролера на сервері.
- Контролер обробляє отримані дані, визначає логіку їх обробки та передає їх на наступний рівень системи.

4. Обробка запиту на рівні серверної частини (Spring Boot):

- Бекенд, розроблений на основі Spring Boot, отримує запит та виконує його обробку через бізнес-логіку.
- Сервер звертається до бази даних для отримання необхідної інформації.
- Отримані результати передаються назад у бізнес-логіку для подальшої обробки.

5. Запит до бази даних (MySQL) та отримання результату:

- База даних MySQL виконує обробку SQL-запиту та формує відповідь у вигляді набору даних, що містить необхідні відомості про товари.
- Отримані результати передаються назад на рівень бекенду.
- Формування відповіді та передача даних клієнту
- Отримані сервером дані перетворюються у структурований формат (наприклад, JSON або безпосередньо в HTML).
- Веб-сторінка передається клієнту для відображення.

6. Відображення списку товарів у браузері:

- Браузер отримує згенеровану сторінку та відображає її для користувача.
- Використовуючи механізми обробки DOM (Document Object Model), CSS та JavaScript, браузер формує фінальний вигляд сторінки.

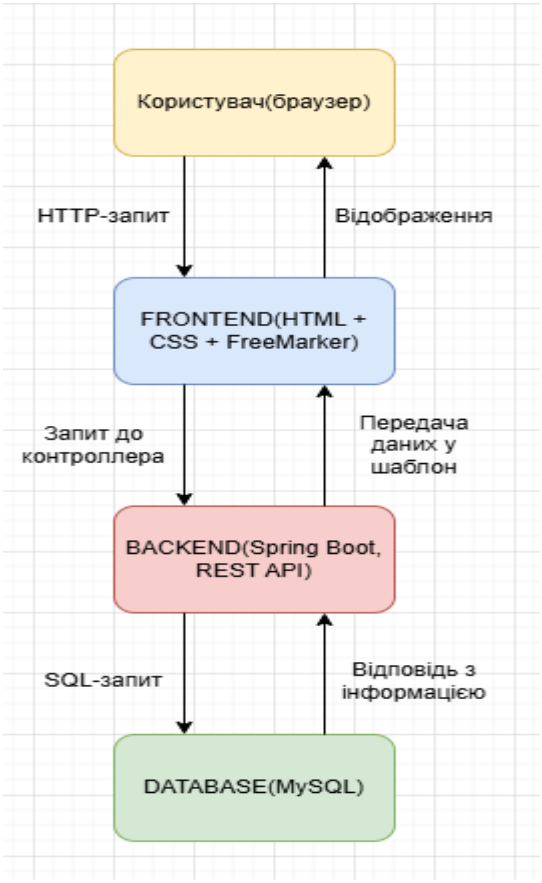


Рис. 2.6 Клієнт-серверна модель онлайн-магазину

Перевагами такої моделі як на рис. 2.6 є:

- Масштабованість – можливість розширення система незалежно від клієнта.
- Гнучкість – можливість змінювати backend або frontend без впливу один на одного.
- Безпека – повний контроль доступу сервера до даних, в той час як клієнт не має прямого доступу.
- Швидка оптимізована робота між сервером через API та клієнтом.

2.2. Компоненти системи: взаємодія сервісів, контролерів, репозиторіїв та бази даних

Основна увага приділяється взаємодії основних компонентів системи, а саме контролерів, сервісів та бази даних. Ці компоненти працюють у тісній взаємодії,

забезпечуючи обробку користувацьких запитів, управління бізнес-логікою та збереження інформації в системі керування базами даних (MySQL).

Основними рівнями системи є:

- Рівень представлення (Frontend) – відповідає за взаємодію з користувачем, формування запитів до сервера та відображення отриманих даних.
- Рівень бізнес-логіки (Backend, сервісний рівень) – обробляє запити клієнта, виконує бізнес-операції та звертається до бази даних.
- Рівень зберігання даних (Database) – відповідає за збереження, оновлення та отримання інформації про товари, користувачів та інші об’єкти системи.

Контролери у Spring Boot відповідають за обробку запитів, які надходять від клієнта. Вони реалізують REST API та викликають відповідні сервіси для виконання необхідних дій.

Основні контролери у системі:

ProductController – керує товарами в магазині, обробляє запити на отримання, додавання та видалення товарів.

```
@GetMapping
public String listProducts(@RequestParam(value = "query", required = false) String query, Model model) {
    List<Product> products;

    if (query != null && !query.isEmpty()) {
        products = productService.searchProducts(query);
    } else {
        products = productService.getAllProducts();
    }
```

Рис. 2.7 Приклад обробки запиту у ProductController

UserController – обробляє запити, пов’язані з управлінням користувачами.

```
@GetMapping("/register")
public String showRegistrationForm(Model model) {
    model.addAttribute( attributeName: "user", new User());
    return "register";
}
```

Рис. 2.8 Приклад обробки запиту у UserController

LoginController – відповідає за автентифікацію користувачів.

```
@GetMapping("/login")
public String loginPage(@RequestParam(value = "error", required = false) String error, Model model) {
    if (error != null) {
        model.addAttribute( attributeName: "error",  attributeValue: "Невірний логін або пароль!");
    }
    return "login";
}
```

Рис. 2.9 Приклад обробки запиту у LoginController

AccountController – відповідає за контроль аккаунта користувача.

```
@GetMapping("/account")
public String myAccount(@AuthenticationPrincipal UserDetails userDetails, Model model) {
    User user = userRepository.findByEmail(userDetails.getUsername()).orElseThrow();
    model.addAttribute( attributeName: "user", user);
    model.addAttribute( attributeName: "orders", orderRepository.findByUser(user));
    return "account";
}
```

Рис. 2.10 Приклад обробки запиту у AccountController

AdminController – надає можливість користувача-адміну переглядати наявні замовлення, помічати їх як виконані та скасовані, додавати та видаляти товари.

```
@PostMapping("/products/delete/{id}")
public String deleteProduct(@PathVariable Long id, HttpSession session) {
    if (!isAdmin(session)) {
        return "redirect:/admin/login";
    }
```

Рис. 2.11 Приклад видалення продукту із бази даних

```
@PostMapping("/orders/complete/{id}")
public String markAsCompleted(@PathVariable Long id, HttpSession session) {
    if (!isAdmin(session)) return "redirect:/admin/login";

    Order order = orderRepository.findById(id).orElseThrow();
    order.setStatus(OrderStatus.COMPLETED);
    orderRepository.save(order);
    return "redirect:/admin/panel";
}
```

Рис. 2.12 Приклад помічення замовлення як виконане

OrderController – надає користувачеві можливість оформити замовлення та підтвердити його заповнивши необхнідні поля з інформацією.

```
@PostMapping("/submit")
public String submitOrder(@ModelAttribute Order order,
                          @RequestParam Long productId,
                          @AuthenticationPrincipal UserDetails userDetails) {

    Product product = productRepository.findById(productId).orElseThrow();
    User user = userRepository.findByEmail(userDetails.getUsername()).orElseThrow();

    order.setProduct(product);
    order.setUser(user);
    orderRepository.save(order);

    return "redirect:/account";
}
```

Рис. 2.13 Приклад підтвердження замовлення

Сервіси виконують бізнес-логіку та обробляють запити від контролерів. Вони працюють із даними, звертаючись до репозиторіїв, і виконують складні операції перед тим, як повернути результат контролеру.

Нижче розглянуто основні сервіси у системи.

ProductController – взаємодіє з ProductRepository для отримання та збереження товарів у базі даних.

```
public List<Product> searchProducts(String query) {  1 usage
    List<Product> products = productRepository.findByNameContainingIgnoreCase(query);

    return products.stream().map( Product product -> {
        if (product.getImage() != null) {
            String base64Image = Base64.getEncoder().encodeToString(product.getImage());
            product.setBase64Image(base64Image);
        }
        return product;
    }).collect(Collectors.toList());
}
```

Рис. 2.14 Приклад конвертації зображення із бази даних у Base64 для відображення на сторінках типу ftlh

UserController – взаємодіє із UserRepository для отримання та збереження email та паролю користувача

```
public User registerUser(User user) {  1 usage
    if (userRepository.findByEmail(user.getEmail()).isPresent()) {
        throw new RuntimeException("Email вже зайнятий!");
    }
    if (userRepository.findByPhoneNumber(user.getPhoneNumber()).isPresent()) {
        throw new RuntimeException("Телефон вже зайнятий!");
    }
```

Рис. 2.15 Приклад реєстрації нового клієнта

Рівень збереження даних (Database, Репозиторії) - є частиною Spring Data JPA і забезпечують зв’язок між сервісним рівнем та базою даних. Spring Data JPA є частиною системи Spring Data і забезпечує зручний механізм впровадження репозиторіїв на основі Java Persistence API (JPA). Ця технологія спрощує створення додатків у межах Spring Framework, що використовують сучасні методи доступу до даних.

Основною метою Spring Data JPA є оптимізація реалізації рівнів доступу до даних, що дозволяє значно зменшити обсяг коду, необхідного для роботи з базою даних. Це досягається за рахунок автоматизації створення репозиторіїв, динамічного генерації запитів та використання зручних засобів для виконання операцій із базою даних.

Розробник, використовуючи Spring Data JPA, визначає інтерфейси репозиторіїв, у яких може оголошувати методи доступу до даних без необхідності їхньої імплементації. Spring автоматично генерує відповідний код та підключає його до системи. Це значно зменшує складність розробки, оскільки розробнику не потрібно вручну реалізовувати операції читання, запису, оновлення чи видалення даних. Spring автоматично створює відповідний SQL-запит для пошуку товарів, що містять введене значення у полі та забезпечує його виконання.

```
public interface ProductRepository extends JpaRepository<Product, Long> { 3 usages
    List<Product> findByNameContainingIgnoreCase(String name); 1 usage
    List<Product> findByScreenSizeAndDisplayTechnologyAndResolution(Double screenSize, String displayTechnology, String resolution);
}
```

Рис. 2.16 ProductRepository, приклад запиту до таблиці даних, знаходження товарів за збігом у назві

```
public interface UserRepository extends JpaRepository<User, Long> { 5 usages
    Optional<User> findByEmail(String email); 2 usages
    Optional<User> findByPhoneNumber(String phoneNumber); 1 usage
}
```

Рис. 2.17 Приклад UserRepository, запит до таблиці даних, знаходження email та номера телефону

```
public interface OrderRepository extends JpaRepository<Order, Long> {  6 usages
    List<Order> findByUser(User user);  1 usage
}
```

Рис. 2.18 Пошук замовлення за користувачем у OrderRepository

Завдяки чіткій структурі взаємодії компонентів система забезпечує логічне розділення обов'язків між рівнями, що сприяє покращенню її підтримки, розширюваності та ефективності. Кожен рівень системи виконує свою визначену роль: контролери приймають і обробляють запити, сервіси реалізують бізнес-логіку, а репозиторії виконують операції взаємодії з базою даних. Це дозволяє забезпечити гнучкість та незалежність компонентів, що сприяє їхньому легкому розширенню та модифікації без порушення загальної роботи системи.

Використання Spring Boot дозволяє значно прискорити процес розробки та підвищити продуктивність додатку завдяки автоматичній конфігурації, зручному управлінню залежностями та інтеграції з іншими технологіями Spring. Завдяки Spring Data JPA усувається необхідність написання складного SQL-коду, що забезпечує автоматизацію створення запитів та зменшує ризик помилок у процесі взаємодії з базою даних. Крім того, Spring Data JPA дозволяє спростити реалізацію операцій доступу до даних через репозиторії, що підтримують CRUD-операції, а також забезпечує можливість використання методів-запитів без необхідності написання додаткового SQL-коду, що значно підвищує зручність роботи з даними. Автоматична підтримка транзакційності гарантує цілісність і узгодженість даних під час виконання складних бізнес-операцій.

Безпека та надійність також є важливими характеристиками системи. Контролери реалізують REST API з обмеженим доступом для різних рівнів користувачів, що дозволяє запобігти несанкціонованому доступу до критично важливих частин системи. Додатковий рівень захисту може бути забезпечений за допомогою Spring Security, що надає можливість автентифікації користувачів, контролю доступу до ресурсів та управління ролями.

Архітектура, що базується на Spring Boot, забезпечує можливість ефективного масштабування системи, що дозволяє збільшувати її продуктивність відповідно до зростання кількості користувачів та навантаження. Масштабування може здійснюватися шляхом додавання нових серверів або балансування навантаження між кількома інстанціями. Використання механізму кешування, такого як Redis або інші кеш-сервіси, дозволяє значно зменшити навантаження на базу даних та підвищити швидкість виконання запитів. Крім того, впровадження реплікації бази даних може суттєво покращити доступність сервісу та зменшити час відповіді у разі високих навантажень.

Подальша оптимізація системи може бути досягнута за рахунок використання індексів у базі даних, що значно прискорить виконання запитів до великих обсягів даних. Запровадження асинхронної обробки запитів дозволить підвищити продуктивність додатку, оскільки тривалі операції будуть виконуватися у фоновому режимі, не блокуючи основний потік обробки даних. Також можливий перехід до мікросервісної архітектури, що дозволить ще більше підвищити гнучкість, незалежність компонентів та масштабованість системи, забезпечуючи її адаптивність до змінних вимог бізнесу та високий рівень надійності.

2.3. Проектування структури бази даних для онлайн-магазину

Розробка бази даних є одним із найважливіших етапів створення інформаційної системи для онлайн-магазину, оскільки вона забезпечує надійне збереження, швидкий доступ та ефективне управління інформацією, зокрема даними про товари, користувачів, замовлення та інші сутності. Використання реляційної моделі бази даних сприяє структурованому збереженню даних, що дозволяє зменшити ризик дублювання інформації, забезпечити узгодженість даних та підвищити продуктивність системи. У цьому контексті застосування MySQL як системи керування базами даних є доцільним рішенням, оскільки вона підтримує ефективне зберігання, обробку та маніпуляцію великими обсягами інформації. Дана технологія дозволяє реалізувати взаємозв’язки між сутностями через

реляційну модель, що сприяє оптимізації запитів, підвищенню швидкодії системи та забезпеченню її масштабованості.

Проектування бази даних базується на фундаментальних принципах, які сприяють оптимальному використанню ресурсів, узгодженості інформації та ефективному виконанню операцій. Одним із ключових аспектів є нормалізація даних, яка передбачає структурований підхід до організації таблиць, усунення надмірності даних та запобігання можливим аномаліям оновлення. Завдяки цьому забезпечується логічна узгодженість інформації та зменшується ймовірність появи конфліктних записів у системі.

Важливим принципом є також цілісність і узгодженість даних, що досягається через встановлення реляційних зв’язків між сутностями. Це дозволяє гарантувати коректність збереження інформації, підтримку унікальності записів та узгодженість при виконанні транзакцій. Даний підхід особливо актуальний для онлайн-магазину, де необхідно забезпечити точне відстеження замовлень, управління товарними позиціями та збереження історії взаємодій користувачів із системою.

Продуктивність бази даних є ще одним критично важливим аспектом проектування, оскільки від її оптимізації залежить загальна швидкодія інформаційної системи. Використання індексів на критичних полях таблиць значно підвищує швидкість виконання пошукових запитів, що є необхідною умовою для ефективного функціонування онлайн-магазину. Окрім того, застосування реляційних зв’язків дозволяє структурувати збереження даних таким чином, щоб мінімізувати кількість необхідних операцій з обробки інформації, тим самим забезпечуючи оптимальну продуктивність навіть при значному обсязі даних.

Масштабованість бази даних відіграє ключову роль у забезпеченні стабільності та гнучкості роботи інформаційної системи. Ефективна структура бази даних повинна дозволяти збільшувати обсяг збережених даних без втрати продуктивності, що досягається через використання стратегій розподілу навантаження, оптимізацію схеми даних та можливість застосування механізмів кешування. Це є важливим фактором для онлайн-магазину, оскільки при

збільшенні кількості користувачів, замовлень та товарів система має зберігати високу швидкість обробки інформації та стабільність роботи.

Проектування структури бази даних для онлайн-магазину базується на фундаментальних принципах нормалізації, підтримки цілісності, продуктивності та масштабованості, що дозволяє забезпечити ефективне управління інформацією, оптимізовану взаємодію між сутностями та високу продуктивність обробки запитів. Це створює надійну основу для розгортання масштабованої, безпечної та продуктивної інформаційної системи, яка відповідає сучасним вимогам електронної комерції.

В якості бази даних виступає MySQL з наступними таблицями:

- Products
- Users

Таблиця Products є ключовою складовою бази даних онлайн-магазину, оскільки вона містить інформацію про наявні товари та їх характеристики. Дана таблиця структурована відповідно до принципів реляційного моделювання, що забезпечує логічну організацію даних, оптимальну продуктивність запитів та цілісність інформації (рис. 2.19).

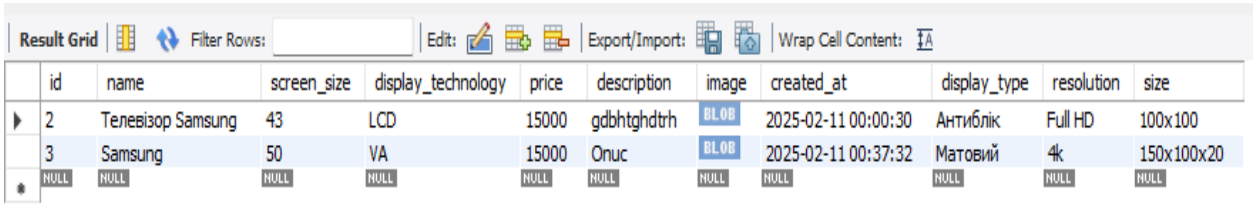
```
CREATE TABLE Products (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    screen_size DECIMAL(5,2) NOT NULL,
    display_technology VARCHAR(50) NOT NULL,
    price DECIMAL(10,2) NOT NULL,
    description TEXT,
    image LONGBLOB,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    display_type VARCHAR(50) NOT NULL,
    resolution VARCHAR(50) NOT NULL,
    size VARCHAR(100) NOT NULL
);
```

Рис. 2.19 Створення таблиці Products

Структура таблиці передбачає використання наступних полів (рис. 2.20):

- id – унікальний ідентифікатор товару, що слугує первинним ключем таблиці. Це поле має автоматично генероване значення (AUTO_INCREMENT), що гарантує унікальність кожного запису та спрощує операції збереження та пошуку товарів у базі даних.
- name – текстове поле, що містить назву товару. Це одне з ключових полів, оскільки воно дозволяє користувачам та адміністраторам ідентифікувати товар у системі.
- screen_size – числове значення, яке визначає розмір екрану товару в дюймах. Це поле є важливим для товарів, що належать до категорії телевізорів або дисплеїв, оскільки розмір екрану є одним із основних критеріїв вибору покупця.
- display_technology – текстове поле, що вказує на тип технології дисплею (наприклад, VA, IPS, OLED). Дане поле дозволяє класифікувати товари за їх візуальними характеристиками та формувати відповідні фільтри для пошуку в системі.
- price – числове поле, що містить вартість товару у грошовому еквіваленті. Поле реалізоване з використанням десяткового формату (DECIMAL) для забезпечення точності обчислень при проведенні фінансових операцій.
- description – текстове поле великого розміру, що містить детальний опис товару. Це поле забезпечує можливість надання розгорнутої характеристики товару, що є важливим для інформування користувачів та покращення досвіду взаємодії із системою.
- image – поле типу BLOB (Binary Large Object), яке використовується для збереження зображень товару без необхідності використання зовнішніх сховищ. Це забезпечує інтегроване управління мультимедійним контентом у межах бази даних.
- created_at – поле з типом TIMESTAMP, яке автоматично зберігає дату та час додавання товару до бази даних. Це дозволяє відстежувати історію змін у каталозі товарів та реалізовувати функціонал сортування за часом додавання.
- display_type – текстове поле, що визначає тип дисплею (наприклад, LCD, LED). Дане поле дозволяє класифікувати товари за особливостями їх екранних технологій та полегшує вибір товарів у межах однієї категорії.

- resolution – текстове поле, яке містить інформацію про роздільну здатність екрану (наприклад, Full HD, 4K, 8K). Це поле є критично важливим для користувачів, які обирають товар за критерієм якості зображення.
- size – текстове поле, що містить загальні параметри розміру товару, включаючи габарити корпусу.



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	name	screen_size	display_technology	price	description	image	created_at	display_type	resolution	size
2	Телевізор Samsung	43	LCD	15000	gdbhtghdtrh	BLOB	2025-02-11 00:00:30	Антиблік	Full HD	100x100
3	Samsung	50	VA	15000	Onuc	BLOB	2025-02-11 00:37:32	Матовий	4k	150x100x20
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Рис. 2.20 Приклад роботи таблиці Products

Таблиця Users також є невід’ємною складовою бази даних онлайн-магазину, оскільки вона містить критично важливу інформацію про зареєстрованих користувачів системи. Її структура розроблена відповідно до принципів реляційного моделювання, що забезпечує ефективне управління даними, високий рівень безпеки та цілісність інформації. Завдяки оптимізованій організації полів таблиця підтримує швидкий доступ до даних користувачів, автентифікацію та управління правами доступу в межах системи.

```
CREATE TABLE Users (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    email VARCHAR(100) UNIQUE NOT NULL,
    phone_number VARCHAR(20) UNIQUE,
    name VARCHAR(100) NOT NULL,
    password VARCHAR(255) NOT NULL,
    active BOOLEAN DEFAULT TRUE,
    role ENUM('USER', 'ADMIN') NOT NULL DEFAULT 'USER'
);
```

Рис. 2.21 Створення таблиці Users

Структура таблиці включає кілька основних полів, кожне з яких виконує конкретну функцію. Унікальний ідентифікатор користувача (id) реалізований у вигляді первинного ключа, що забезпечує унікальність кожного запису та автоматично генерується за допомогою механізму AUTO_INCREMENT. Це рішення дозволяє ефективно здійснювати операції з ідентифікації користувачів у системі.

Одним із ключових атрибутів таблиці є електронна пошта (email), що використовується як унікальний ідентифікатор під час автентифікації користувачів (рис. 2.22). Це поле є критично важливим, оскільки забезпечує можливість ідентифікації користувача в системі, а також використовується для надсилання повідомлень та підтверджень реєстрації. Окрім цього, таблиця містить номер телефону (phone_number), який слугує додатковим засобом комунікації та може бути використаний для двофакторної автентифікації або відновлення доступу.

Ім'я користувача (name) представлено у вигляді текстового поля, яке містить персональні дані власника облікового запису (рис. 2.22). Дане поле дозволяє ідентифікувати користувача в межах інтерфейсу системи, забезпечуючи зручну персоналізацію у процесі взаємодії з онлайн-магазином.

Окремим елементом таблиці є пароль (password), що зберігається у зашифрованому форматі (рис. 2.22). Це критично важливий компонент системи безпеки, оскільки він забезпечує захист конфіденційних даних користувача. Використання сучасних криптографічних методів хешування дозволяє запобігти несанкціонованому доступу до облікових записів та підвищує рівень інформаційної безпеки системи.

Функціональність керування статусом користувачів реалізована за допомогою булевого поля active, яке вказує, чи є обліковий запис активним (рис. 2.22). Це поле використовується для керування доступом до системи, що дозволяє адміністраторам тимчасово блокувати користувачів або відновлювати їхній доступ без необхідності видалення записів.

Окреме значення в структурі таблиці має поле role, яке визначає рівень доступу користувача в системі (рис. 2.22). Це поле використовується для реалізації

механізму контролю доступу (RBAC – Role-Based Access Control), що дозволяє надавати різні права доступу адміністраторам, звичайним користувачам або іншим ролям. Завдяки цьому система забезпечує гнучке управління привілеями та обмежує доступ до конфіденційної інформації залежно від рівня прав користувача.

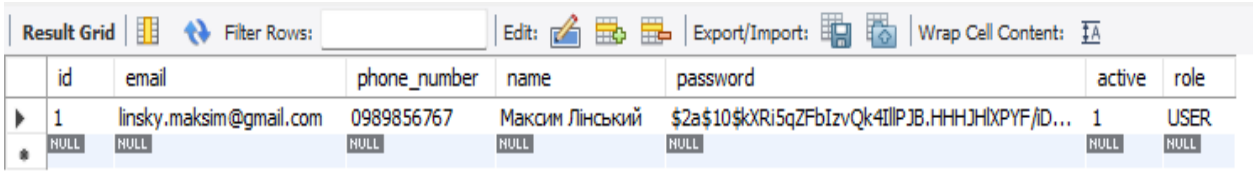


Рис. 2.22 Приклад роботи таблиці Users

Структура таблиці Users побудована відповідно до принципів логічної організації, інформаційної безпеки та оптимального управління доступом, що забезпечує ефективну автентифікацію користувачів, персоналізацію взаємодії та високий рівень безпеки даних у системі (рис. 2.22).

Таблиця Order є прикладом реляційної структури даних, що зберігає інформацію про клієнтів, їхні замовлення та відповідні статуси (рис. 2.23). Вона складається з таких атрибутів:

- id – первинний ключ запису (унікальний ідентифікатор);
- city – назва міста, де був зафіксований запис або здійснене замовлення;
- full_name – повне ім’я клієнта;
- phone – контактний номер телефону;
- product_id – ідентифікатор товару або послуги;
- user_id – ідентифікатор користувача системи;
- status – статус замовлення (наприклад, COMPLETED, CANCELLED, ACTIVE).

```sql
CREATE TABLE orders (
    id INT PRIMARY KEY,
    city VARCHAR(100),
    full_name VARCHAR(255),
    phone VARCHAR(20),
    product_id INT,
    user_id INT,
    status VARCHAR(50)
);
```

Рис. 2.23 Приклад створення таблиці Orders у SQL

```java
public class Order {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne
    private User user;

    @ManyToOne
    private Product product;

    private String fullName;
    private String phone;
    private String city;

    @Enumerated(EnumType.STRING)
    private OrderStatus status = OrderStatus.ACTIVE;
}
```

Рис. 2.24 Створення таблиці Orders у Intellij IDEA

Такі таблиці використовуються в інформаційних системах для моніторингу стану замовлень, аналізу активності клієнтів та оптимізації сервісних процесів. Вони легко інтегруються з backend-логікою застосунків, особливо у контексті Spring Boot та JPA/Hibernate (рис. 2.24).

id	city	full_name	phone	product_id	user_id	status
4	Київ	Максим Лінський	0989856767	3	1	COMPLETED
5	Обухів	Максим Лінський	0685811070	2	1	CANCELLED
6	Львів	Олександр Лінський	0989856767	2	1	ACTIVE
7	Київ	Олександр Лінський	0685811070	3	3	ACTIVE
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Рис. 2.25 Приклад роботи таблиці Orders

Ця таблиця демонструє типову ситуацію, коли один клієнт може мати кілька записів (рядків) з різними параметрами — містом, товаром або статусом виконання (рис. 2.25). Наприклад, користувач Максим Лінський має два записи з різними номерами телефонів, містами та статусами замовлень (COMPLETED і CANCELLED). Аналогічно, користувач Олександр Лінський також представлений двома записами зі статусом ACTIVE.

3 РОЗРОБКА ПРОТОТИПУ ОНЛАЙН-МАГАЗИНУ НА ОСНОВІ SPRING BOOT

3.1 Налаштування середовища розробки

Робота над онлайн-магазин починається із Spring Initializr. Spring Initializr надає розширюваний API для створення проектів на основі JVM із реалізаціями кількох поширених концепцій:

- Генерація базової мови для Java, Kotlin і Groovy.
- Створіть абстракцію системи з реалізаціями для Apache Maven і Gradle.
- Підтримка .gitignore.
- Кілька точок підключення для генерації власних ресурсів.

Spring Initializr — це розширювана платформа, призначена для генерації початкової структури проєктів на базі Spring Boot. Вона включає низку модулів, кожен з яких виконує свою функціональну роль у процесі створення та обслуговування проєктів (рис. 3.1).

Модулі Spring Initializr:

Модуль initializr-actuator відповідає за надання діагностичної інформації та статистики, що стосуються процесу генерації проєктів. Це дозволяє здійснювати моніторинг ефективності використання сервісу Spring Initializr і виявляти потенційні вузькі місця в його роботі.

Модуль initializr-bom реалізує концепцію «bill of materials» (BOM), яка забезпечує централізоване керування сумісними версіями залежностей. Це суттєво полегшує підтримку проєктів, унеможливлюючи конфлікти версій між бібліотеками, що часто виникають під час оновлення або інтеграції нових компонентів.

Документаційний модуль initializr-docs містить повний опис принципів функціонування Spring Initializr. Він також надає рекомендації щодо адаптації й розширення цього інструменту в межах індивідуальних або корпоративних

проєктів, що особливо корисно для розробників, які бажають реалізувати власні шаблони генерації.

Ключову роль у системі відіграє модуль initializr-generator — основна бібліотека, що безпосередньо відповідає за створення структури проєкту. Вона формує кодову базу відповідно до обраних користувачем параметрів, таких як мова програмування, версія Spring Boot, набір залежностей тощо.

Доповнюючим є модуль initializr-generator-spring, який задає типовий шаблон конфігурації для проєктів, заснованих на Spring Boot. Цей модуль орієнтований на дотримання загальноприйнятих практик Spring-екосистеми, але також передбачає можливість внесення кастомних змін відповідно до вимог конкретного середовища розробки.

Для забезпечення високої якості генерації та стабільності проєктів використовується модуль initializr-generator-test. Він надає інфраструктуру для тестування процесу створення проєктів, що дозволяє виявляти помилки на ранніх етапах розробки.

Модуль initializr-metadata відіграє важливу роль в управлінні метаданими, які описують доступні опції конфігурації — перелік залежностей, підтримувані мови, сумісні версії та інші параметри, які формують кінцеву структуру проєкту.

Для демонстрації можливостей створення власного сервісу генерації слугує initializr-service-sample — приклад реалізації настроюваного веб-сервісу. Цей модуль є відправною точкою для побудови кастомізованого рішення, що відповідає специфічним вимогам організації чи команди розробників.

Ще одним важливим компонентом є initializr-version-resolver — утиліта, яка дозволяє автоматично отримувати актуальні версії залежностей із зовнішніх джерел, зокрема з POM-файлів. Це сприяє підтримці проєкту в актуальному стані та забезпечує його сумісність із останніми релізами бібліотек.

Модуль initializr-web відповідає за веб-інтерфейс та API, які використовуються як для взаємодії з користувачем, так і для інтеграції з іншими сервісами. Цей модуль реалізує механізми прийому параметрів конфігурації, запуску генерації та передачі згенерованих проєктів клієнтам.

Ці модулі разом створюють гнучку та потужну систему для швидкого старту та стандартизованої ініціалізації Java-проєктів у середовищі Spring.

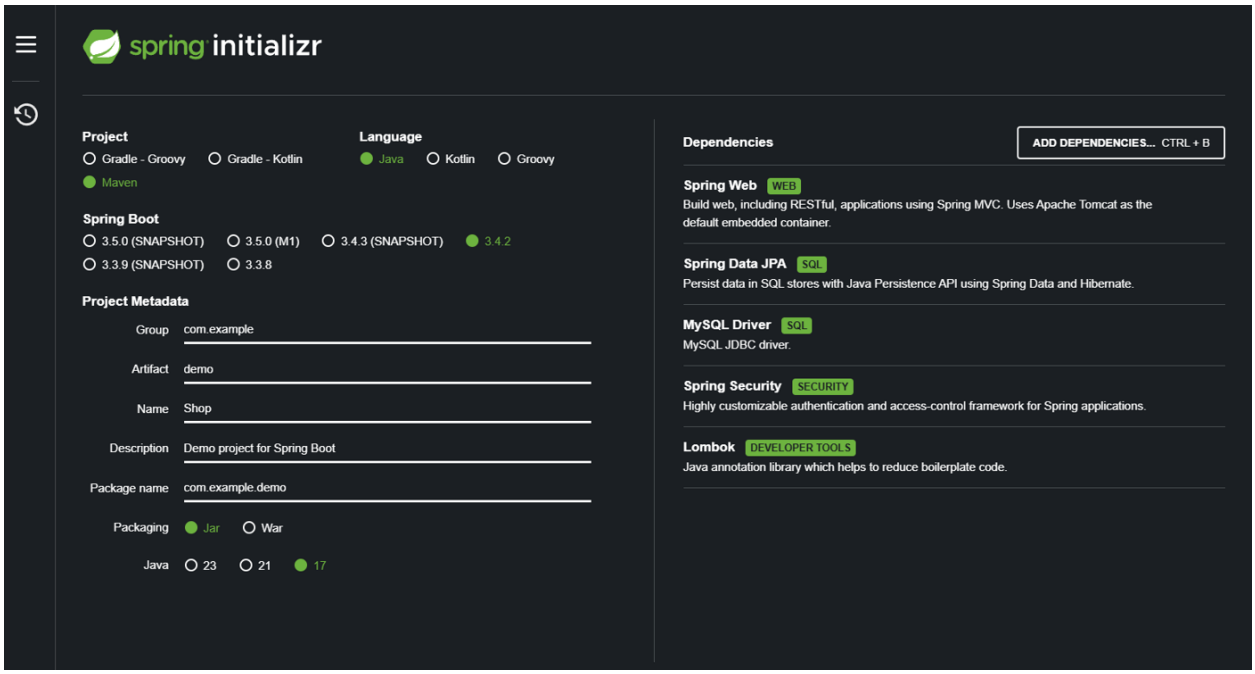


Рис. 3.1 Налаштування Spring Initializr для проєкту

Типів проєкту загалом чотири - Gradle – Groovy, Gradle – Kotlin та Maven.

Gradle – Groovy - плагін Groovy у середовищі Gradle розширює функціональність плагіна Java, додаючи підтримку для розробки проєктів на мові Groovy. Його функціональність охоплює як обробку чистого Groovy-коду, так і змішаних проєктів, що містять компоненти як на Groovy, так і на Java. Хоч він може компілювати й виключно Java-код, така практика не є рекомендованою через наявність спеціалізованих інструментів, орієнтованих на Java. Однією з переваг плагіна є підтримка спільної компіляції, що забезпечує прозоре поєднання класів, реалізованих на обох мовах. Це дає змогу створювати двосторонні залежності між класами, реалізованими на Groovy та Java, що значно розширює можливості архітектурного проєктування. Наприклад, клас на Groovy може успадковувати функціональність класу на Java, який, у свою чергу, наслідує інший клас на Groovy.

Gradle – Kotlin - Kotlin DSL (Domain-Specific Language) у середовищі Gradle представляє собою сучасну альтернативу традиційному Groovy DSL для

конфігурації проєктів. Його застосування забезпечує значно розширеніші можливості редагування конфігураційних файлів завдяки повній інтеграції з сучасними середовищами розробки (IDE), такими як IntelliJ IDEA. Вбудований компілятор Kotlin працює на Linux, macOS, Windows, Cygwin, FreeBSD і Solaris на архітектурах x86-64.

Maven - Apache Maven — це інструмент управління та розуміння програмного проекту. Базуючись на концепції об’єктної моделі проекту (POM), Maven може керувати збіркою проекту, звітністю та документацією з центральної частини інформації.

Для роботи із проєктом використовується Maven з його перевагами:

- Полегшення процесу створення (build process) - Maven автоматизує низку повторюваних завдань, таких як компіляція, тестування, пакування та розгортання. При цьому він абстрагує багато внутрішніх деталей, дозволяючи розробникам зосередитися на логіці застосунку, а не на технічних нюансах збірки.
- Забезпечення єдиної системи побудови - Maven пропонує стандартизовану структуру проєкту та фіксований життєвий цикл збірки, що забезпечує узгодженість між різними проєктами та командами.
- Надання якісної інформації про проєкт - завдяки інтеграції з різними плагінами, Maven генерує багатий набір метрик, звітів та документації, що сприяє кращому моніторингу якості та прогресу розробки.

У конфігурації проєкту як основну мову програмування було обрано Java сімнадцятої версії, що відповідає сучасним стандартам розробки та забезпечує підтримку новітніх можливостей платформи Java.

Тип пакування застосунку — JAR (Java ARchive), що дозволяє створювати автономні виконувані архіви, зручні для розгортання та інтеграції у хмарні середовища.

У якості основи фреймворку використовується Spring Boot версії 3.4.2, що забезпечує спрощену конфігурацію, високу продуктивність та сумісність із найновішими версіями Spring екосистеми.

Важливою частину роботи із Spring Boot є правильне використання та додавання необхнідних залежностей як Spring Web - Spring Web є традиційним веб-фреймворком, що побудований на основі Servlet API та реалізує модель блокувального вводу-виводу (blocking I/O). Основна парадигма цього підходу полягає у тому, що під час виконання запиту потік обробки блокується доти, доки не буде отримана відповідь від зовнішнього джерела, зокрема, бази даних або стороннього веб-сервісу. Цей фреймворк є оптимальним рішенням для створення синхронних веб-додатків, у яких взаємодія із зовнішніми компонентами здійснюється у послідовному порядку. Зокрема, у типовому випадку використання, коли необхідно отримати інформацію з погодного API, контролер Spring Web надсилатиме запит та очікуватиме на завершення операції, перш ніж повертати відповідь користувачу. Такий підхід є інтуїтивно зрозумілим і простим у реалізації, однак він менш ефективний при високих навантаженнях або за умов інтенсивного паралельного доступу.

Spring Data JPA - це підсистема у межах більш широкого проєкту Spring Data, призначена для спрощення реалізації доступу до даних на основі Java Persistence API (JPA). Її основна мета полягає у зменшенні обсягу шаблонного (template) коду, який традиційно потрібен для розробки рівня доступу до даних у застосунках, що використовують JPA. Створення навіть простих SQL-запитів вимагало значної кількості низькорівневого коду — від ініціалізації транзакцій до обробки результатів запиту. Коли до цього додаються такі вимоги, як пагінація, аудит, або фільтрація за умовами, складність і обсяг реалізації зростають експоненційно. Spring Data JPA пропонує високий рівень абстракції, дозволяючи розробнику зосередитись виключно на функціональній логіці. Інтерфейси репозиторіїв, створені розробником, можуть містити лише сигнатури методів, тоді як їх реалізація автоматично генерується фреймворком на основі правил іменування або за допомогою анотацій.

MySQL Driver - (Structured Query Language) є системою керування реляційними базами даних (RDBMS) з відкритим вихідним кодом, яка широко використовується для зберігання, організації та ефективного управління даними.

Завдяки своїй надійності, високій продуктивності, масштабованості та простоті у використанні, MySQL стала однією з найпопулярніших СУБД серед розробників програмного забезпечення різного рівня — від індивідуальних проєктів до великих корпоративних систем.

Spring Security - є потужним та гнучким фреймворком для забезпечення безпеки в Java-застосунках, який виконує функції автентифікації (authentication) та авторизації (authorization) користувачів. Він розроблений як частина системи Spring і на сьогодні вважається стандартом для впровадження механізмів контролю доступу у Spring-додатках. Основна функціональність Spring Security зосереджена на створенні захищеного середовища для взаємодії користувача з додатком, дозволяючи визначати політики доступу, обмеження за ролями, правила сесій, а також захист від поширених векторів атак, таких як CSRF або XSS. Перевагою цього фреймворку є його висока розширюваність - архітектура дозволяє розробнику адаптувати механізми безпеки до специфічних бізнес-вимог, реалізуючи кастомні фільтри, сервіси користувачів, схеми входу.

Lombok - є спеціалізованою бібліотекою для мови програмування Java, яка використовує механізм анотацій для скорочення обсягу шаблонного (template) коду. Основною метою цієї бібліотеки є спрощення та підвищення ефективності процесу розробки шляхом автоматичної генерації часто використовуваних конструкцій у коді. Lombok надає низку анотацій, які дозволяють усунути потребу в ручному написанні таких елементів, як конструктори без аргументів, методи toString(), equals(), hashCode(), а також гетери та сетери. Замість цього, розробнику достатньо додати відповідні анотації до класу, і під час компіляції Lombok автоматично згенерує необхідний байт-код у файлах .class. Цей підхід забезпечує зменшення дублювання коду, покращення читабельності та підтримуваності програмного забезпечення. З технічного погляду, трансформації виконуються на етапі компіляції, що дозволяє зберегти чистоту вихідного коду без шкоди для функціональності застосунку.

У процесі розробки онлайн-бібліотеки з використанням технології Spring Boot були застосовані ключові анотації, що значно полегшили конфігурування та організацію компонентів програми. Однією з основних анотацій є @Controller, застосовується для оголошення класу як контролера в рамках Spring MVC. Такі класи переважно використовуються у веб-застосунках, де є потреба у поверненні представлень користувачу.

Завдяки анотації @Autowired, яку надає Spring Framework, реалізується автоматичне впровадження залежностей у класи. Це звільняє розробника від ручної ініціалізації об’єктів і дозволяє Spring самостійно впроваджувати потрібні залежності у поля, методи або конструктори класів.

Анотація @GetMapping відповідає за обробку запитів, що надсилаються з використанням методу GET. Вона є актуальною у випадках, коли потрібно лише отримати певну інформацію, наприклад, під час завантаження сторінки користувачем[20].

Для доступу до поточного автентифікованого користувача у Spring Security використовується анотація @AuthenticationPrincipal. Вона дозволяє звернутися до об'єкта, який реалізує інтерфейс Authentication#getPrincipal(), безпосередньо в методах контролера.

Анотація @Configuration, що з’явилась у Spring 3.0, слугує для позначення класів, які є джерелами конфігурацій для контейнера IoC. Вона є сучасною заміною XML-конфігурацій, надаючи програмісту можливість описувати метадані контексту у вигляді Java-коду.

Застосування анотації @EnableWebSecurity вмикає у Spring Boot підтримку функціоналу веб-безпеки, що дозволяє реалізувати політику доступу, автентифікацію та авторизацію в межах застосунку.

Анотація @Bean слугує для позначення методів, що повертають об'єкти, які слід зареєструвати як компоненти (bean) у контейнері Spring IoC. Вона тісно пов’язана з @Configuration і використовується у класах конфігурації.

Анотація @RequestMapping забезпечує обробку запитів до веб-контролерів. Вона є гнучкою та дозволяє обробляти запити різних типів: GET, POST, PUT, DELETE, на відміну від більш спеціалізованих анотацій.

@PostMapping — це анотація, що використовується для обробки POST-запитів. Вона вважається безпечнішою у порівнянні з @GetMapping, оскільки передає дані у тілі запиту (HttpBody), і часто застосовується під час надсилання форм або створення нових об’єктів[20].

Анотація @RequestParam дозволяє отримувати значення параметрів запиту з URL. Такі параметри передаються після символу "?" в адресному рядку та розділяються знаком “&”.

Застосування анотації @PathVariable дає змогу отримувати значення змінних, які вбудовано безпосередньо в URL. На відміну від @RequestParam, такі змінні є частиною маршруту, а не параметрами.

Анотація @ModelAttribute виконує роль механізму, що забезпечує зіставлення даних запиту з об’єктами моделі. Вона може застосовуватись як до параметрів методу, так і до самих методів, забезпечуючи ефективну передачу даних між клієнтом і сервером.

Позначення класу анотацією @Entity сигналізує JPA про те, що даний клас представляє собою сутність, яка буде відповідати таблиці у базі даних. Це дозволяє здійснювати перетворення екземплярів класу на записи в таблиці[19].

Анотація @Table використовується разом із @Entity і дозволяє вказати додаткову інформацію про таблицю бази даних, зокрема її назву, схему або інші характеристики, необхідні для коректного зіставлення.

Завдяки анотації @Column розробник може детально описати стовпці таблиці, які відповідають полям класу. Ця анотація дозволяє задавати параметри зберігання даних, їхню назву, довжину тощо.

@Lob позначає ті поля сутності, які мають зберігатися як великі об’єкти (Large Objects) — наприклад, зображення або відео. ORM-фреймворк, побачивши цю анотацію, зберігатиме такі поля як CLOB або BLOB у відповідній базі даних.

Анотація @Data забезпечує автоматичну генерацію важливих методів класу — геттерів, сеттерів, toString, equals, hashCode, а також конструктора з обов’язковими параметрами. Це дозволяє значно скоротити обсяг шаблонного коду в POJO-класах.

Анотація @Transient виключає поле з процесу збереження у базі даних. Така анотація корисна, коли певне значення розраховується динамічно або використовується лише у процесі обчислення на рівні застосунку.

Анотація @NoArgsConstructor дозволяє згенерувати конструктор без параметрів, який ініціалізує поля стандартними значеннями. Це корисно при створенні об’єктів без попередньої ініціалізації їхніх властивостей.

Анотація @AllArgsConstructor створює конструктор, який приймає значення для всіх полів класу. Це дозволяє швидко створити об'єкт із повністю заданими параметрами при його ініціалізації[18].

Анотація @Id використовується для позначення поля, що виконує роль первинного ключа у сутності JPA. Це поле забезпечує унікальність кожного запису у відповідній таблиці.

@GeneratedValue вказує на те, що значення поля з анотацією @Id повинно генеруватися автоматично. Тип генерації можна налаштувати за допомогою відповідної стратегії, що дозволяє, наприклад, використовувати послідовності або автозаповнення[17].

Анотація @ManyToOne визначає зв'язок між двома сутностями у форматі "багато до одного", де декілька об’єктів можуть бути пов’язані з одним елементом іншої сутності. Це забезпечує нормалізацію бази даних і зручність у роботі з відношеннями[16].

Анотація @Enumerated дає змогу зберігати перерахункові типи (enum) у базі даних у вигляді строки або порядкового числа. Якщо не вказано інший варіант, тип зберігається у вигляді числового значення, яке відповідає позиції елемента у переліку.

3.2 Реалізація функціональних можливостей онлайн-магазину

У межах реалізації функціонального модуля облікового запису в онлайн-магазині, побудованому за допомогою технології Spring Boot, особливу роль відіграє контролер AccountController. Його призначенням є обробка HTTP GET-запиту за маршрутом /account, що забезпечує доступ користувача до персоніфікованої сторінки облікового запису. Вказана сторінка слугує ключовим елементом взаємодії користувача з системою, оскільки на ній зосереджено основні відомості щодо його профілю, зокрема ім’я, контактні дані, історію замовлень та інші персональні параметри.

Однією з особливостей функціональності AccountController є механізм аутентифікації, який реалізується засобами Spring Security (рис. 3.2). У методі myAccount використовується анотація @AuthenticationPrincipal, що дозволяє автоматично впроваджувати до контролера об'єкт, який містить інформацію про поточного автентифікованого користувача. Це реалізовано шляхом передачі екземпляра інтерфейсу UserDetails як параметра методу, що, у свою чергу, дає змогу отримувати дані про особу, яка на даний момент здійснила вхід у систему. Завдяки цьому забезпечується надійна і безпечна робота з персональними даними, оскільки доступ до сторінки облікового запису можливий виключно для авторизованих користувачів.

```
@GetMapping("/account")
public String myAccount(@AuthenticationPrincipal UserDetails userDetails, Model model) {
    User user = userRepository.findByEmail(userDetails.getUsername()).orElseThrow();
    model.addAttribute( attributeName: "user", user);
    model.addAttribute( attributeName: "orders", orderRepository.findByUser(user));
    return "account";
}
```

Рис. 3.2 Аутентифікація користувача

Для отримання даних користувача з бази даних використовується репозиторій UserRepository. Він здійснює пошук об'єкта User за його електронною поштою, який збігається із ім'ям користувача у UserDetails (рис. 3.3).

```
User user = userRepository.findByEmail(userDetails.getUsername()).orElseThrow();
```

Рис. 3.3 Отримання користувача з бази даних

За завантаження замовлень користувача, що належать даному користувачеві, відповідає OrderRepository. Після отримання об’єкта користувача (user) за допомогою UserRepository, OrderRepository здійснює вибірку всіх пов’язаних з ним замовлень (orders) (рис. 3.4). Після чого об'єкти user та orders додаються до моделі (Model model) і передаються у шаблон account.ftlh.

```
model.addAttribute( attributeName: "orders", orderRepository.findByUser(user));
```

Рис. 3.4 Завантаження замовлень користувача

AdminController відповідає за авторизацію адміністратора для доступу до адміністративної панелі (рис. 3.5). Під час обробки запиту виконується перевірка введеного пароля, і лише у разі його правильності користувач отримує доступ до адміністративного інтерфейсу.

```
@PostMapping("/login")
public String login(@RequestParam String password, HttpSession session, Model model) {
    if (ADMIN_PASSWORD.equals(password)) {
        session.setAttribute( s: "admin",  o: true);
        return "redirect:/admin/panel";
    } else {
        model.addAttribute( attributeName: "error",  attributeValue: "❌ Неправильний пароль!");
        return "adminLogin";
    }
}
```

Рис. 3.5 Авторизація адміністратора

У системі управління онлайн-магазином панель адміністратора виступає важливим інструментом для моніторингу та управління основними бізнес-процесами (рис. 3.6). Метод showAdminPanel, анотаційований як @GetMapping("/panel"), реалізує логіку відображення цієї панелі. Він забезпечує авторизаційний контроль, перевіряючи, чи має користувач адміністративні повноваження. Якщо перевірка не проходить, користувача автоматично перенаправляють на сторінку входу адміністратора.

Після успішної авторизації метод виконує вибірку всіх наявних замовлень за допомогою orderRepository та всіх товарів через productRepository. Для зручності перегляду та керування, замовлення класифікуються за статусами: активні (ACTIVE), завершені (COMPLETED) та скасовані (CANCELLED). Кожна з цих груп додається до моделі з відповідними іменами атрибутів — activeOrders, completedOrders і cancelledOrders, що дозволяє шаблонізатору відобразити їх окремими списками у відповідних секціях сторінки.

```
@GetMapping("/panel")
public String showAdminPanel(HttpSession session, Model model) {
    if (!isAdmin(session)) {
        return "redirect:/admin/login";
    }

    List<Order> allOrders = orderRepository.findAll();
    List<Product> products = productRepository.findAll();

    model.addAttribute( attributeName: "activeOrders", allOrders.stream()
            .filter( Order o -> o.getStatus() == OrderStatus.ACTIVE)
            .toList());

    model.addAttribute( attributeName: "completedOrders", allOrders.stream()
            .filter( Order o -> o.getStatus() == OrderStatus.COMPLETED)
            .toList());

    model.addAttribute( attributeName: "cancelledOrders", allOrders.stream()
            .filter( Order o -> o.getStatus() == OrderStatus.CANCELLED)
            .toList());

    model.addAttribute( attributeName: "products", products);

    return "adminPanel";
}
```

Рис. 3.6 Реалізація панелі адміністратора

У даному фрагменті Java-коду реалізовано функціонал видалення замовлення в системі керування замовленнями з використанням фреймворку Spring. Метод deleteOrder позначений анотацією @PostMapping, що вказує на обробку HTTP POST-запиту за маршрутом /orders/delete/{id} (рис. 3.7). У якості параметрів метод приймає ідентифікатор замовлення (id) та об'єкт сесії користувача (HttpSession session). Перед виконанням видалення проводиться перевірка прав доступу через виклик методу isAdmin(session). Якщо користувач не є адміністратором, його буде перенаправлено на сторінку авторизації адміністратора (redirect:/admin/login). У протилежному випадку відбувається виклик orderRepository.deleteById(id), який видаляє запис замовлення з бази даних за переданим ідентифікатором. Після успішного видалення користувача перенаправляють на адміністративну панель (redirect:/admin/panel).

```
@PostMapping("/orders/delete/{id}")
public String deleteOrder(@PathVariable Long id, HttpSession session) {
    if (!isAdmin(session)) {
        return "redirect:/admin/login";
    }

    orderRepository.deleteById(id);
    return "redirect:/admin/panel";
}
```

Рис. 3.7 Реалізація видалення замовлення

Реалізовано механізм видалення товару з бази даних у веб-додатку, створеному за допомогою Spring Framework (рис. 3.8). Метод deleteProduct, анотований як @PostMapping("/products/delete/{id}"), відповідає за обробку POST-запиту на вказаний маршрут і приймає два параметри: ідентифікатор товару (id) та сесію користувача (HttpSession session).

```java
@PostMapping("/products/delete/{id}")
public String deleteProduct(@PathVariable Long id, HttpSession session) {
    if (!isAdmin(session)) {
        return "redirect:/admin/login";
    }

    productRepository.deleteById(id);
    return "redirect:/admin/panel";
}
```

Рис. 3.8 Реалізація видалення товару

Два окремих методи у Spring-додатку, що дозволяють адміністратору змінювати статус замовлення (рис. 3.9), (рис. 3.10). Це важлива частина бізнес-логіки для систем керування онлайн-замовленнями, яка забезпечує контроль за життєвим циклом замовлення. Метод markAsCompleted відповідає за встановлення статусу замовлення як COMPLETED. Аналогічно, метод markAsCancelled змінює статус на CANCELLED. Обидва методи починаються з перевірки адміністративного доступу через isAdmin(session). Якщо перевірка не проходить, користувач перенаправляється на сторінку входу адміністратора.

```java
@PostMapping("/orders/complete/{id}")
public String markAsCompleted(@PathVariable Long id, HttpSession session) {
    if (!isAdmin(session)) return "redirect:/admin/login";

    Order order = orderRepository.findById(id).orElseThrow();
    order.setStatus(OrderStatus.COMPLETED);
    orderRepository.save(order);
    return "redirect:/admin/panel";
}
```

Рис. 3.9 Зміна статусу замовлення на завершене

```
@PostMapping("/orders/cancel/{id}")
public String markAsCancelled(@PathVariable Long id, HttpSession session) {
    if (!isAdmin(session)) return "redirect:/admin/login";

    Order order = orderRepository.findById(id).orElseThrow();
    order.setStatus(OrderStatus.CANCELLED);
    orderRepository.save(order);
    return "redirect:/admin/panel";
}
```

Рис. 3.10 Зміна статусу замовлення на скасоване

Метод isAdmin, який відповідає за перевірку прав доступу користувача на основі його сесії (рис. 3.11). Метод має модифікатор доступу private і повертає логічне значення (boolean), що вказує, чи є користувач адміністратором. Усередині методу з сесії витягується атрибут з ключем "admin", який очікується як об'єкт типу Boolean. Далі здійснюється перевірка, чи атрибут не є null і має значення true. Якщо обидві умови виконуються, метод повертає true, що означає наявність адміністративних прав.

```
private boolean isAdmin(HttpSession session) {  5 usages
    Boolean isAdmin = (Boolean) session.getAttribute( s: "admin");
    return isAdmin != null && isAdmin;
}
```

Рис. 3.11 Перевірка прав адміністратора

Функціонально LoginController відповідає за обробку GET-запиту на сторінку входу /login (рис. 3.12).

```
@GetMapping("/login")
public String loginPage(@RequestParam(value = "error", required = false) String error, Model model) {
    if (error != null) {
        model.addAttribute( attributeName: "error",  attributeValue: "❌ Невірний логін або пароль!");
    }
    return "login"; // Повертає шаблон login.ftlh
}
```

Рис. 3.12 Обробка запиту /login

@GetMapping("/login") – обробляє запит на сторінку входу.

@RequestParam(value = "error", required = false) – перевіряє параметр error, якщо він є.

Якщо error != null, модель отримає атрибут error, що виведеться у шаблоні login.ftlh та на сторінці буде показано помилку входу (рис. 3.12).

Функціональні можливостями OrderController є відображення форми замовлення. Даний фрагмент коду показує сторінку із формою оформлення замовлення вибраного товару (рис. 3.13).

```
@GetMapping("/create/{productId}")
public String showOrderForm(@PathVariable Long productId, Model model) {
    Product product = productRepository.findById(productId).orElseThrow();
    model.addAttribute( attributeName: "product", product);
    model.addAttribute( attributeName: "order", new Order());
    return "orderForm";
}
```

Рис. 3.13 Відображення форми замовлення

Метод приймає ідентифікатор товару (productId) як параметр шляху, знаходить відповідний товар у базі даних за допомогою productRepository.findById(productId).orElseThrow() та додає його до об'єкта Model під іменем "product" (рис. 3.14). Крім того, до моделі додається новий екземпляр класу Order, який буде використовуватись у формі для заповнення замовлення.

```
@PostMapping("/submit")
public String submitOrder(@ModelAttribute Order order,
                          @RequestParam Long productId,
                          @AuthenticationPrincipal UserDetails userDetails) {

    Product product = productRepository.findById(productId).orElseThrow();
    User user = userRepository.findByEmail(userDetails.getUsername()).orElseThrow();

    order.setProduct(product);
    order.setUser(user);
    orderRepository.save(order);

    return "redirect:/account";
}
```

Рис. 3.14 Надсилання форми замовлення

На даному етапі роботи контролера реалізується логіка надсилання форми замовлення, яка обробляє дані, заповнені користувачем, і зберігає нове замовлення у базі даних. Метод, що відповідає за цю функцію, зазвичай анотований як @PostMapping, що вказує на обробку POST-запиту з клієнтської сторони. Методі приймає модель об'єкта Order, яка автоматично заповнюється з даних HTML-форми за допомогою механізмів Spring MVC. Далі, використовуючи productId, виконується пошук відповідного товару з бази даних через productRepository.findById(...). Паралельно з цим із об'єкта UserDetails, що зазвичай надається Spring Security, визначається користувач, який надіслав замовлення.

Контролер ProductController відповідає за відображення списку товарів із можливістю пошуку. Метод обробляє GET-запит за маршрутом /products та надає перелік усіх наявних товарів (рис. 3.15). У разі, якщо у запиті присутній параметр query, відбувається фільтрація результатів і повертаються лише ті товари, що відповідають критеріям пошуку.

```
@GetMapping
public String listProducts(@RequestParam(value = "query", required = false) String query, Model model) {
    List<Product> products;

    if (query != null && !query.isEmpty()) {
        products = productService.searchProducts(query);
    } else {
        products = productService.getAllProducts();
    }

    model.addAttribute( attributeName: "products", products);
    return "products";
}
```

Рис. 3.15 Відображення списку товарів

Контролер також реалізує функцію відображення форми для створення нового товару. У методі створюється порожній об'єкт Product, який передається у модель для заповнення відповідними даними через форму. Після цього повертається шаблон представлення createProduct.ftlh, що містить інтерфейс введення інформації про новий товар (рис. 3.16).

```
@GetMapping("/create")
public String showCreateProductForm(Model model) {
    model.addAttribute( attributeName: "product", new Product());
    return "createProduct";
}
```

Рис. 3.16 Відображення форми створення товару

Функціонал додавання нового товару реалізовано у методі createProduct, який обробляє POST-запит, надісланий з форми створення товару. Метод приймає об’єкт із заповненими даними, а також обробляє завантажене зображення. Після цього викликається сервісний метод productService.addProduct(), що відповідає за збереження нового товару в базі даних (рис. 3.17).

```
@PostMapping("/create")
public String createProduct(@RequestParam String name,
                            @RequestParam double screenSize,
                            @RequestParam String displayTechnology,
                            @RequestParam double price,
                            @RequestParam String description,
                            @RequestParam MultipartFile image,
                            @RequestParam String displayType,
                            @RequestParam String resolution,
                            @RequestParam String size,
                            Model model) {
    try {
        productService.addProduct(name, screenSize, displayTechnology, price, description, image, displayType, resolution, size);
        model.addAttribute( attributeName: "message",  attributeValue: "✅ Товар успішно додано!");
    } catch (IOException e) {
        model.addAttribute( attributeName: "error",  attributeValue: "❌ Помилка завантаження зображення!");
    } catch (RuntimeException e) {
        model.addAttribute( attributeName: "error", e.getMessage());
    }

    return "createProduct";
}
```

Рис. 3.17 Функція додавання нового товару

Контролер UserController містить функцію відображення форми реєстрації нового користувача (рис. 3.18). У відповідному методі створюється новий об’єкт User, який додається до моделі з метою подальшого заповнення через форму. Після цього користувачу повертається шаблон представлення register.ftlh, що містить інтерфейс для введення реєстраційних даних.

```
@GetMapping("/register")
public String showRegistrationForm(Model model) {
    model.addAttribute( attributeName: "user", new User());
    return "register"; // FreeMarker автоматично шукає register.ftlh
}
```

Рис. 3.18 Реалізація відображення форми реєстрації

При реєстрація користувача POST-запит обробляється із форми реєстрації, реєструє нового користувача за допомогою сервісу UserService (рис. 3.19).

```
@PostMapping("/register")
public String registerUser(@ModelAttribute User user, Model model) {
    userService.registerUser(user);
    model.addAttribute( attributeName: "message",  attributeValue: "Користувач успішно зареєстрований!");
    return "register";
}
```

Рис. 3.19 Реєстрація користувача

У системі онлайн-магазину реалізовано модель Order, яка представляє собою сутність бази даних. Клас позначений анотацією @Entity, що вказує на його відповідність окремій таблиці у базі даних. Поле id у цій моделі є первинним ключем (@Id) і забезпечує унікальну ідентифікацію кожного запису (рис. 3.20).

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;
```

Рис. 3.20 ID замовлення

```
@ManyToOne
private User user;
```

Рис. 3.21 Користувач (User)

У моделі Order зв’язок типу ManyToOne з користувачем, що вказує на відношення багато-до-одного (рис. 3.21). Це означає, що кожне замовлення

належить одному конкретному користувачу, але один користувач може мати кілька замовлень.

```
@ManyToOne
private Product product;
```

Рис. 3.22 Продукт (Product)

Також зв’язок ManyToOne із сутністю Product, що означає: багато замовлень можуть стосуватися одного й того ж самого товару (рис. 3.22).

```
private String fullName;
private String phone;
private String city;
```

Рис. 3.23 Контактна інформація

Для моделі Order створено окремі поля для збереження повного імені замовника, номера телефону та міста доставки (рис. 3.23). Ці поля необхідні для правильної обробки замовлення, забезпечення зв’язку з клієнтом і організації доставки товару.

```
@Enumerated(EnumType.STRING)
private OrderStatus status = OrderStatus.ACTIVE;
```

Рис. 3.24 Статус замовлення

Поле статусу зберігається у базі даних зі значенням «активний» за замовченням (рис. 3.24). Це означає, що кожне нове замовлення створюється з початковим статусом, який вказує на те, що воно ще перебуває в обробці. У процесі роботи адміністратора статус замовлення може бути змінено на «скасований» або «завершений» залежно від результату виконання замовлення.

Реалізовано модель OrderStatus – це enum, який представляє можливі стани замовлення.

```
public enum OrderStatus {  8 usages
    ACTIVE,  3 usages
    COMPLETED,  3 usages
    CANCELLED  3 usages
}
```

Рис. 3.25 Модель OrderStatus

У системі існує три статуси для обробки замовлень: Active, Completed та Cancelled (рис. 3.25). Статус Active означає, що замовлення щойно створене і перебуває в очікуванні обробки та доставки. Статус Completed позначає успішно виконане та доставлене замовлення, тоді як Cancelled вказує на скасоване замовлення, яке не буде виконане. Для реалізації роботи з цими статусами використовується відповідний фрагмент коду, представлений на рисунку 3.24, який дозволяє змінювати статус замовлення відповідно до його поточного стану в процесі обробки.

Модель Product – сутність бази даних, що відповідає за збереження інформації про всі товари в базі даних.

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;

@Column(nullable = false, unique = true)
private String name;

@Column(nullable = false)
private double screenSize;

@Column(nullable = false)
private String displayTechnology;

@Column(nullable = false)
private double price;

@Column(columnDefinition = "TEXT")
private String description;
```

Рис. 3.26 Головні поля про товар

Модель Product містить основні характеристики товару, необхідні для його ідентифікації та представлення в онлайн-магазині. До таких характеристик належать: унікальний ідентифікатор (id), назва товару, розмір екрану, технологія дисплею, ціна та текстовий опис (рис. 3.26).

```
@Lob
@Column(columnDefinition = "LONGBLOB")
private byte[] image;

@Transient
private String base64Image; // Не зберігається в БД, використовується тільки для відображення
```

Рис. 3.27 Зображення до товару

У моделі Product зображення товару зберігається у вигляді байтового масиву з типом LONGBLOB, що дозволяє зберігати у базі даних великі графічні файли без втрати якості. Для відображення зображення на стороні клієнта використовується поле base64Image, яке містить зображення у форматі Base64 (рис. 3.27).

Ще однією моделлю є User – сутність користувача, використовується для аутентифікації, авторизації та збережння профілю користувача.

```
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true, nullable = false)
    private String email;

    @Column(unique = true, nullable = false)
    private String phoneNumber;

    @Column(nullable = false)
    private String name;

    @Column(nullable = false)
    private String password;
```

Рис. 3.28 Головна інформаці користувача

Модель User містить основні дані, необхідні для ідентифікації та авторизації користувача в системі (рис. 3.28). До таких даних належать: унікальний ідентифікатор (id), електронна пошта, номер телефону, ім’я та пароль.

```
@Enumerated(EnumType.STRING)
@Column(nullable = false)
private UserRole role = UserRole.USER;
```

Рис. 3.29 Ролі користувача

Існує поле для збереження ролі користувача, що визначає його рівень доступу в системі. За замовчуванням кожному новому користувачу присвоюється роль USER, яка надає базові права для перегляду та оформлення замовлень (рис. 3.29). У разі необхідності роль може бути змінена на ADMIN, що надає розширені можливості, зокрема керування товарами, замовленнями та доступ до адміністративної панелі.

Останньою моделлю є UserRole – це enum, який представляє наявні ролі користувача.

```
public enum UserRole {  6 usages
    USER, ADMIN  2 usages
}
```

Рис. 3.30 Модель ролей користувача

У структурі онлайн-магазину реалізовано інтерфейс OrderRepository, який забезпечує доступ до даних замовлень (рис. 3.30). Цей інтерфейс розширює функціонал Spring Data JPA і дозволяє виконувати стандартні операції з сутністю Order, такі як збереження, оновлення, пошук та видалення.

```
public interface OrderRepository extends JpaRepository<Order, Long> {  6 usages
    List<Order> findByUser(User user);  1 usage
}
```

Рис. 3.31 OrderRepository

Інтерфейс OrderRepository використовує можливості Spring Data JPA для виконання базових CRUD-операцій над сутністю Order (рис. 3.31). Серед

доступних методів — findAll() для отримання всіх замовлень, findById(Long id) для пошуку конкретного замовлення за ідентифікатором, save(Order order) для збереження або оновлення замовлення, а також deleteById(Long id) для його видалення. Остання строка коду повертає список замовлень, що зробив користувач.

Другим репозиторієм є ProductRepository – реалізовано для роботи із сутністю Product.

```
public interface ProductRepository extends JpaRepository<Product, Long> {  7 usages
    List<Product> findByNameContainingIgnoreCase(String name);  1 usage
```

Рис. 3.32 ProductRepository

Інтерфейс ProductRepository аналогічно до OrderRepository надає набір базових CRUD-операцій для роботи з сутністю Product, таких як findAll(), findById(Long id), save(Product product) та deleteById(Long id) (рис. 3.32). Окрім стандартних функцій, репозиторій також підтримує можливість пошуку товарів за повною або частковою назвою

Третім репозиторієм є UserRepository – репозиторій створений для роботи із користувачами в базі даних.

```
public interface UserRepository extends JpaRepository<User, Long> {  9 usages
    Optional<User> findByEmail(String email);  4 usages
    Optional<User> findByPhoneNumber(String phoneNumber);  1 usage
}
```

Рис. 3.33 UserRepository

Інтерфейс UserRepository забезпечує виконання базових CRUD-операцій для роботи з сутністю User, таких як створення, зчитування, оновлення та видалення користувачів (рис. 3.33). Крім стандартних методів, репозиторій реалізує пошук користувачів за електронною поштою та номером телефону.

У структурі онлайн-магазину реалізовано клас SecurityConfig (рис. 3.34), який відповідає за налаштування безпеки додатку. Це Java-конфігурація, що

активує механізми Spring Security. Завдяки цьому класу визначаються правила доступу до сторінок, обмеження для різних ролей користувачів (наприклад, USER чи ADMIN), а також налаштовуються параметри автентифікації та авторизації. Конфігурація забезпечує захист веб-додатку від несанкціонованого доступу та контролює безпеку всього процесу взаємодії з користувачем.

```
http
        .csrf( CsrfConfigurer<HttpSecurity> csrf -> csrf.disable())
        .authorizeHttpRequests( AuthorizationManagerRequestMat... auth -> auth
                .requestMatchers("/", "/register", "/users/register", "/login").permitAll()
                .requestMatchers("/home").authenticated()
                .anyRequest().authenticated()
        )
```

Рис. 3.34 Головні правила доступу

Конфігурація SecurityConfig передбачає відкритий доступ до певних маршрутів, зокрема /register, /users/register та /login, що дозволяє користувачам реєструватися та входити в систему без попередньої авторизації. Усі інші маршрути додатку захищені та стають доступними лише після успішної автентифікації користувача.

```
.formLogin( FormLoginConfigurer<HttpSecurity> login -> login
        .loginPage("/login")
        .defaultSuccessUrl( defaultSuccessUrl: "/products",  alwaysUse: true)
        .failureUrl( authenticationFailureUrl: "/login?error=true")
        .permitAll()
)
```

Рис. 3.35 Реалізації форми логіну

У конфігурації SecurityConfig задається власна сторінка входу за маршрутом /login, яка використовується для автентифікації користувачів (рис. 3.35). У разі успішної авторизації система автоматично перенаправляє користувача на сторінку з товарами — /products. Якщо ж під час входу виникає помилка (наприклад, неправильний логін або пароль), користувача повертає на сторінку входу з

параметром /login?error=true, що дозволяє відобразити відповідне повідомлення про помилку.

```
.logout( LogoutConfigurer<HttpSecurity> logout -> logout
        .logoutUrl("/logout")
        .logoutSuccessUrl("/login?logout")
        .permitAll()
);
```

Рис. 3.36 Реалізація виходу із аккаунту

Також визначено маршрут для виходу з системи — /logout. Після завершення сесії користувача система автоматично виконує перенаправлення на сторінку входу з параметром /login?logout (рис. 3.36).

```
@Bean
public PasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(); }
```

Рис. 3.37 PasswordEncoder

PasswordEncoder відповідає за шифрування паролів для безпеки від крадіжки бази даних користувачі використовується BCrypt, що зашифровує паролі (рис. 3.37).

```
@Bean
public UserDetailsService userDetailsService(UserRepository userRepository) {
    return String email -> userRepository.findByEmail(email) Optional<User>
            .map( User user -> org.springframework.security.core.userdetails.User.withUsername(user.getEmail())
                    .password(user.getPassword())
                    .roles(user.getRole().name())
                    .build()) Optional<UserDetails>
            .orElseThrow(() -> new UsernameNotFoundException("Користувача не знайдено"));
}
```

Рис. 3.38 Деталі користувача

У процесі автентифікації користувача реалізовано механізм пошуку облікового запису за електронною поштою у базі даних (рис. 3.38). Знайдений

користувач перетворюється на об’єкт UserDetails, який містить усю необхідну інформацію для роботи Spring Security — зокрема логін, пароль та набір прав доступу. Роль користувача визначається на основі значення поля UserRole і включається до списку авторитетів, що дозволяє системі обмежувати або надавати доступ до відповідних ресурсів згідно з призначеними правами.

У межах розробки онлайн-магазину важливою складовою є зручний, інтуїтивно зрозумілий інтерфейс користувача, який забезпечує швидкий доступ до основних функцій системи.

Початковим елементом взаємодії користувача з системою є сторінка авторизації (рис. 3.39), яка розташована за адресою localhost:8080/login. Дизайн реалізовано у мінімалістичному стилі: елементи інтерфейсу розміщено по центру екрана, що сприяє зосередженню уваги на основному завданні – введенні облікових даних. Заголовок сторінки – «Вхід» – відображено чітко та лаконічно. Форма авторизації містить два текстові поля: для введення логіна та пароля. Кнопка «Увійти» стилізована зеленим кольором, що символізує позитивну дію та підтвердження. Нижче розташована синя кнопка «Реєстрація», яка веде користувача на сторінку створення нового облікового запису.

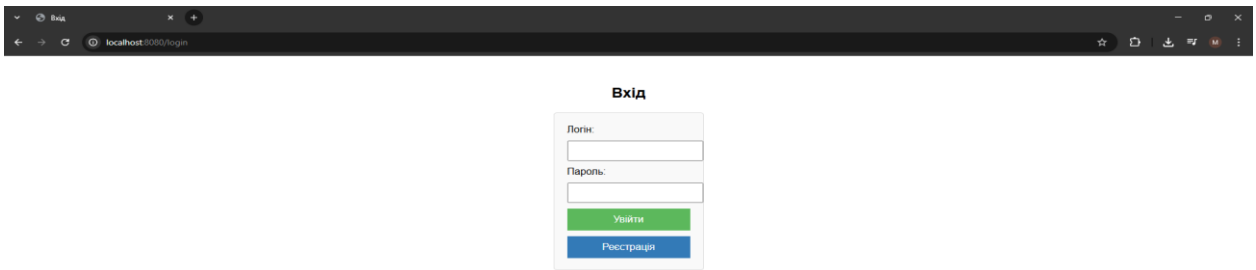


Рис. 3.39 Сторінка входу до системи

Сторінка реєстрації доступна за маршрутом localhost:8080/users/register (рис. 3.40), виконує функцію створення нового облікового запису для користувача. З точки зору архітектури взаємодії з користувачем, ця форма є ключовою для залучення нових відвідувачів до системи та забезпечення персоналізованого доступу до функціоналу інтернет-магазину.

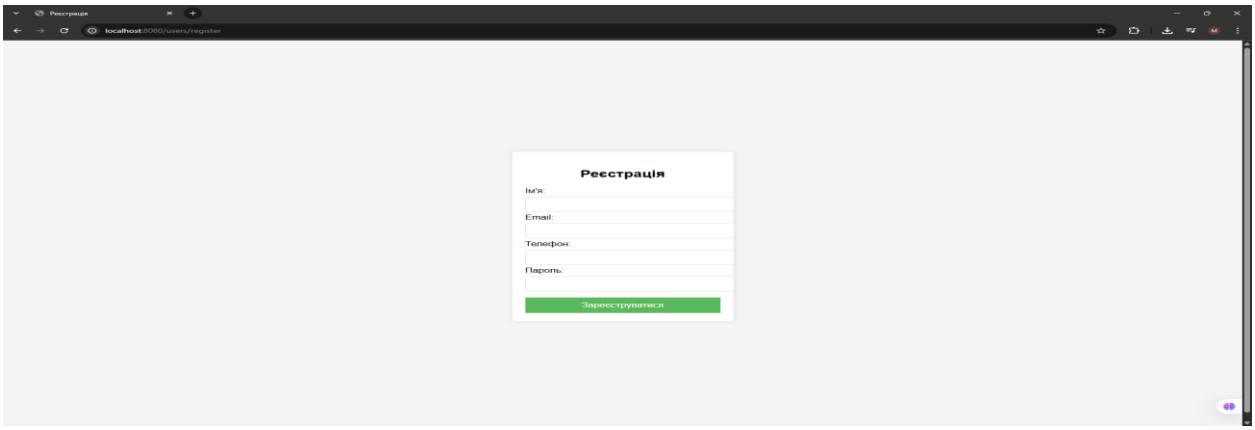


Рис. 3.40 Сторінка реєстрації

Після автентифікації користувач потрапляє на основну сторінку перегляду товарів (рис. 3.41), яка виконує роль вітрини інтернет-магазину. Вона реалізована у вигляді адаптивного вебінтерфейсу, що дозволяє зручно ознайомитися з асортиментом продукції. У верхній частині сторінки розміщено логотип магазину під назвою «DieselShop», що виконує функцію візуального маркера бренду, а також сприяє загальній впізнаваності ресурсу. Поряд із логотипом інтегровано пошуковий інтерфейс, за допомогою якого користувач може оперативно здійснювати фільтрацію за назвою товару. Праворуч знаходиться посилання «Мій акаунт», яке забезпечує перехід до персонального профілю користувача. Основну площину інтерфейсу займають картки товарів, реалізовані у вигляді окремих елементів сітки. Кожна з них містить структуровано подану інформацію про продукт: назву, основні технічні параметри, ціну, детальний опис та зображення. Кожна картка завершується кнопкою «Замовити», що надає можливість для замовлення.

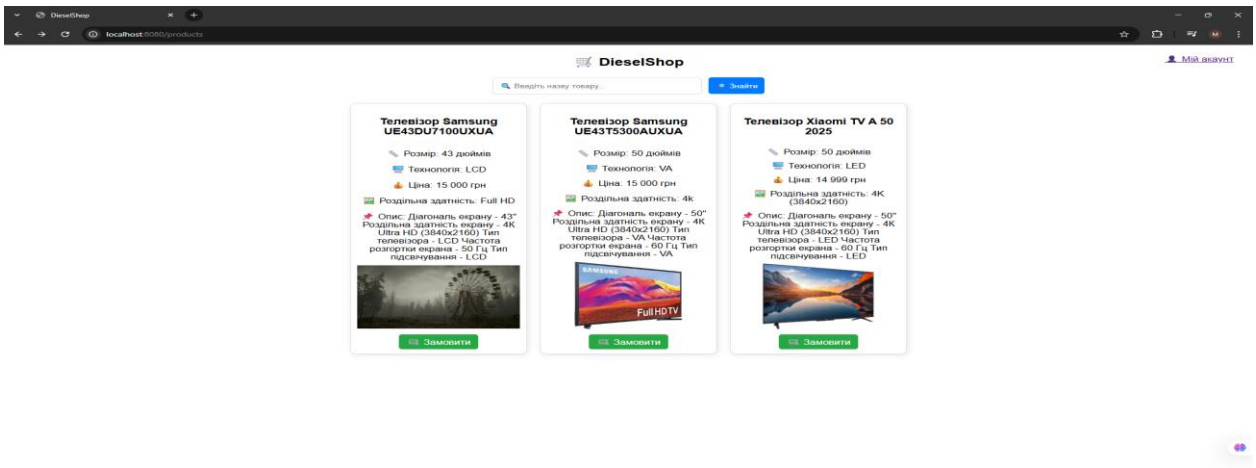


Рис. 3.41 Сторінка вітрини інтернет-магазину

Сторінка оформлення замовлення є завершальним етапом взаємодії користувача з каталогом товарів інтернет-магазину. Форма містить три базові поля для заповнення: «Ім’я та прізвище», «Телефон» та «Місто». Така структура забезпечує збір мінімально необхідної контактної інформації для здійснення доставки й комунікації з клієнтом.

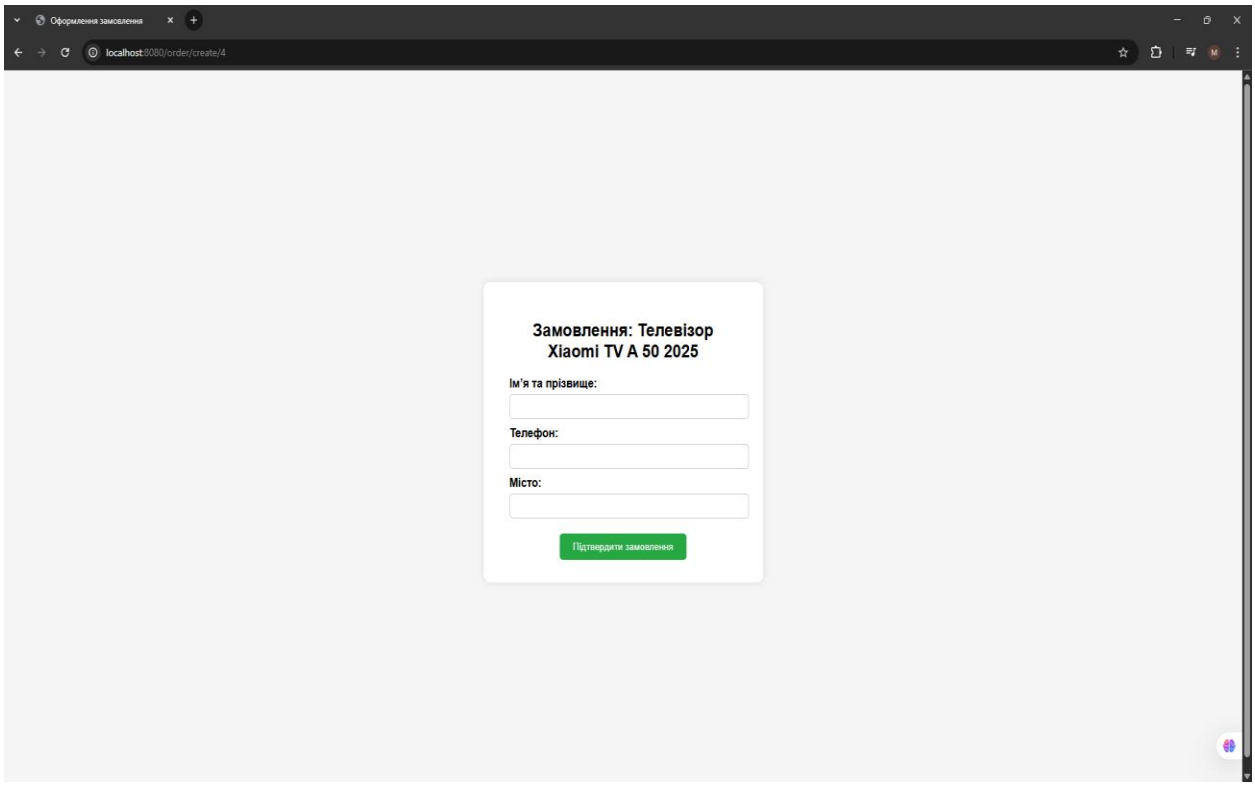


Рис. 3.42 Сторінка замовлення товару

Сторінка особистого кабінету користувача, що розташована за маршрутом /account (рис. 3.43) на якій розташована вся інформація по замовленням. У верхній частині блоку наведено персональні дані користувача — ім’я та прізвище, адресу електронної пошти і контактний номер телефону. Нижче по сторінці представлено таблицю із заголовком «Мої замовлення», яка виконує функцію архіву транзакційної активності користувача. У таблиці систематизовано перелік усіх здійснених замовлень, включаючи ключові параметри: найменування товару, місто доставки, контактний номер телефону та поточний статус виконання. Статус замовлення маркується текстовими значеннями, такими як «Виконане», «Скасоване» або «Активне», що дозволяє користувачеві оперативно орієнтуватися у своїй взаємодії з магазином.

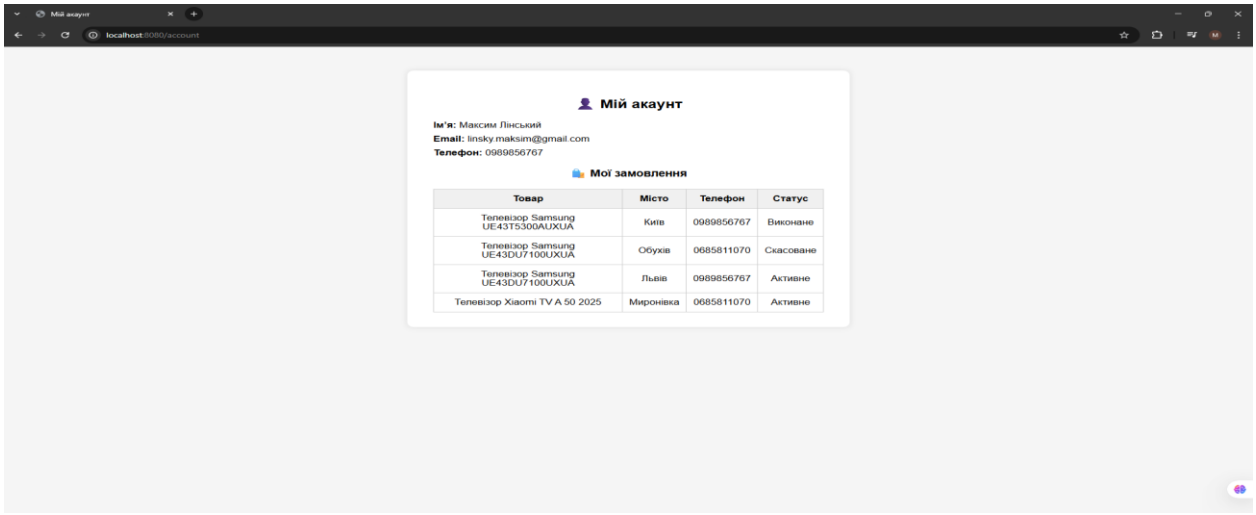


Рис. 3.43 Сторінка аккаунта користувача

Адміністративна панель (рис. 3.44) є центральним інструментом керування внутрішніми процесами інтернет-магазину та призначена виключно для адміністраторів, які здійснюють модерацію вмісту, контроль за виконанням замовлень і управління асортиментом товарів. Вона розташована за маршрутом /admin/panel. У верхній частині інтерфейсу розміщено кнопку «Додати товар», що активує механізм розширення товарного асортименту та дозволяє швидко інтегрувати нові позиції до бази даних. Основний інформаційний блок поділено на чотири функціональні розділи: активні, виконані та скасовані замовлення, а також

перелік усіх товарів. У кожного замовлення три функціональні кнопки: «Виконано», «Відмінено» та «Видалити», що дозволяє адміністратору змінювати статуси замовлень або повністю видаляти записи з бази.

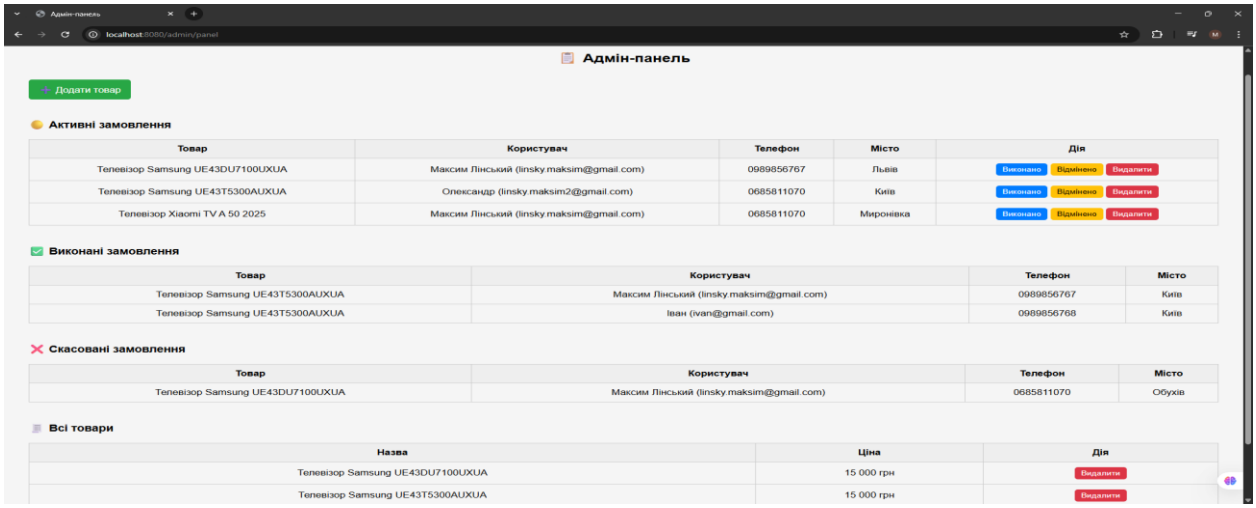


Рис. 3.44 Сторінка адміністративної панелі

У останньому блоці «Всі товари» виведено список наявних у базі товарів з вказанними назвами, цінами та кнопкою «Видалити» біля кожної позиції, яка забезпечує швидке вилучення товару із системи.

3.3 Тестування прототипу онлайн-магазину

З метою оцінки зручності застосунку було проведено тестування неавторизованого користувача.

Якщо користувач не зареєстрований та спробує перейти на будь-яку сторінку із магазину, наприклад, /products, то він має отримувати помилку сторінки, або перехід на сторінку до моменту авторизації (рис. 3.46).

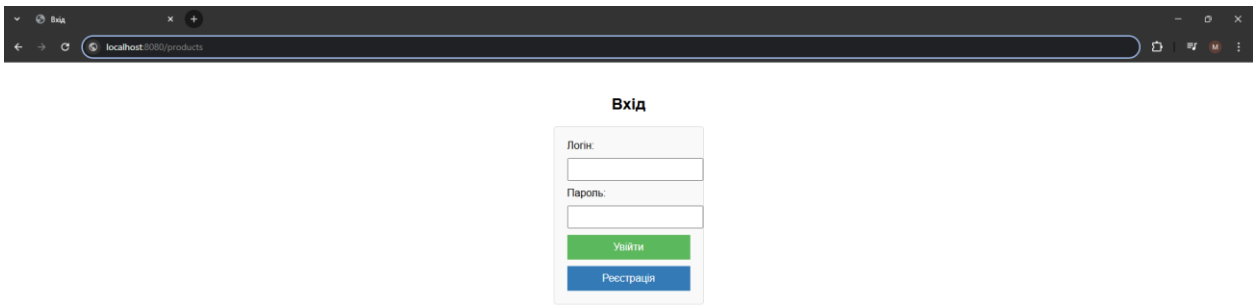


Рис. 3.45 Перехід із сторінки авторизації до наступної

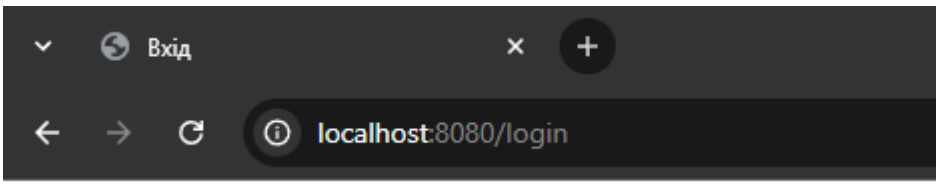


Рис. 3.46 Успішка перевірка неавторизованого користувача

Неавторизований користувач повернувся на сторінку авторизації та не перейшов на наступну, а саме /products.

Також було проведено тестування DDos-атаки.

Для даного тесту було створено файл dos_attack.py.

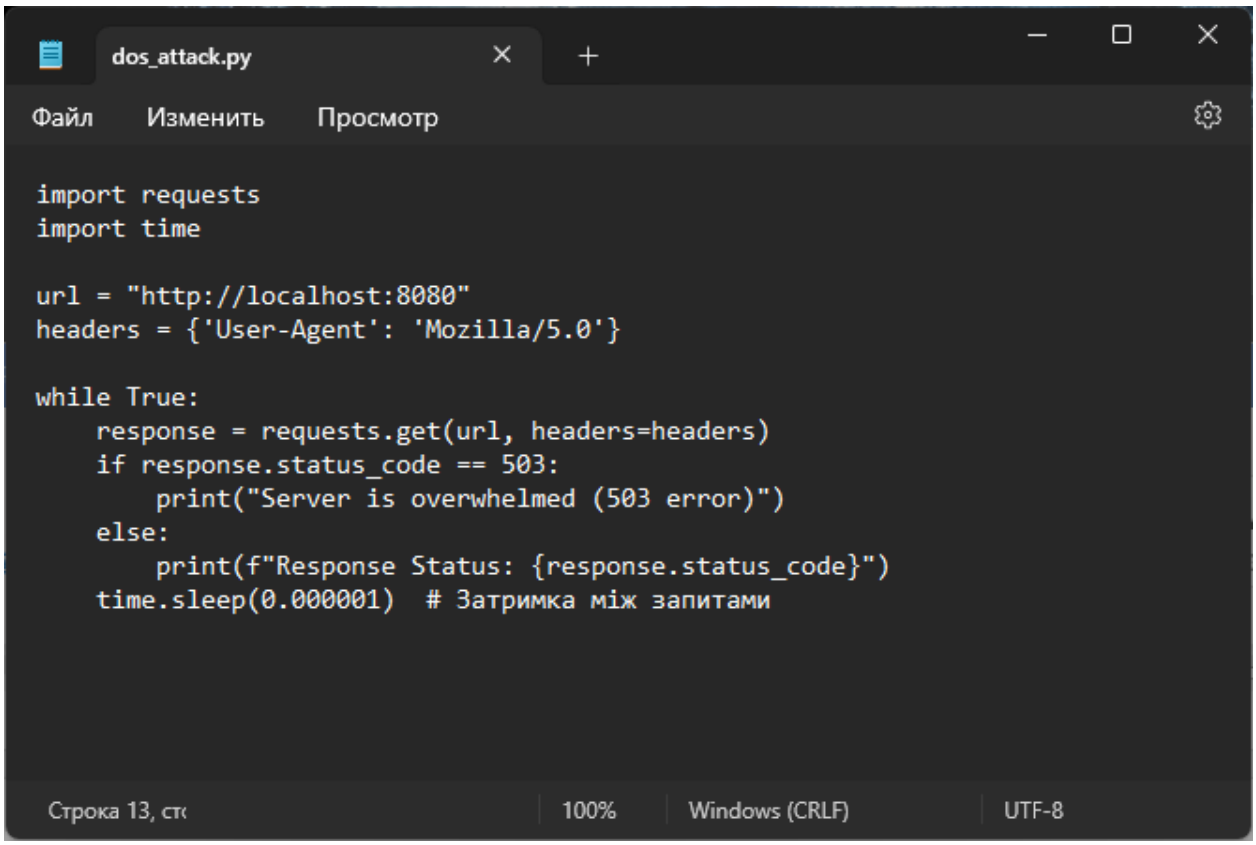


Рис. 3.47 Файл dos_attack.py

Вказаний інтервал – time.sleep(0.000001) на рис. 3.47, означає, що в секунду будуть надсилатись один мільйон запитів.

Нагрузка на онлайн-магазин до проведення DDos-атаки менша за 1% (рис 3.48).

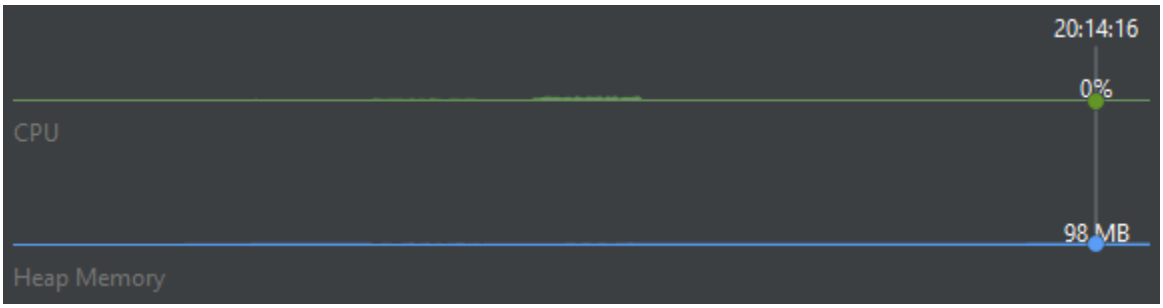


Рис. 3.48 Навантаження серверу до DDos-атаки

Після активації файлу спостерігається незначне збільшення навантаження на сервер (рис. 3.49), однак воно не досягає рівня, який би загрожував стабільному функціонуванню системи.

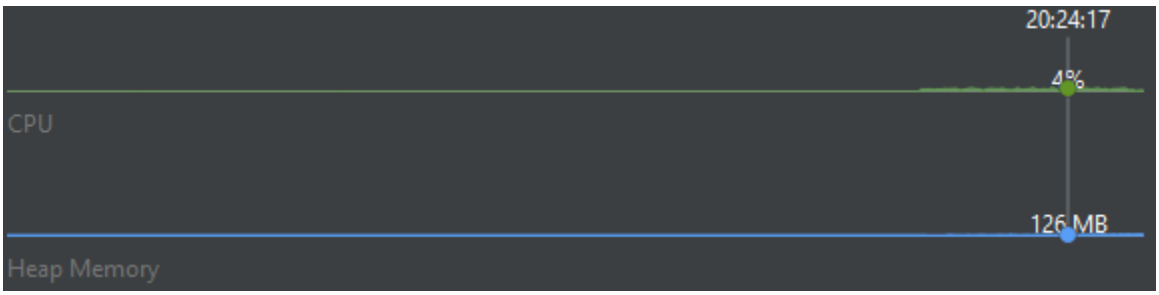


Рис. 3.49 Навантаженя серверу під час DDos-атаки

Також було проведене тестування хешування паролю.

У процесі реєстрації аккаунта на рис. 3.50, створено користувача з іменем Дмитро, електронною поштою dmitro@gmail.com, номером телефону 0989853131 та паролем Dmitro53421.

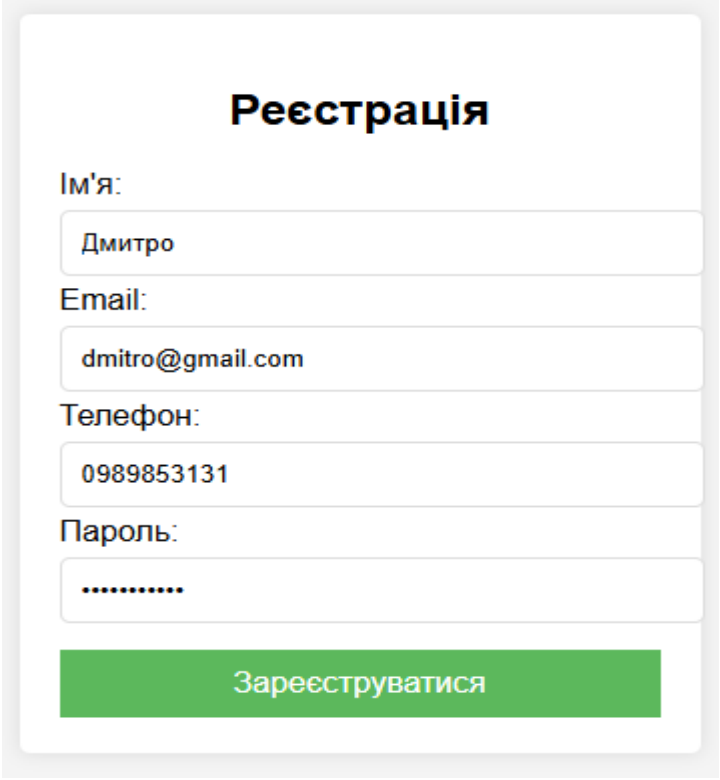


Рис. 3.50 Створення аккаунта користувача

В базі даних, таблиця users, додано нового користувача із вказаними даними (рис. 3.51).

id	email	phone_number	name	password	active	role
1	linsky.maksim@gmail.com	0989856767	Максим Лінський	$2a$10$kXRi5qZFbIzvQk4IllPJB.HHHJHlXPYF/iD...	1	USER
2	petro@gmail.com	0685811071	Петро	$2a$10$wj/UxpFOuzFP2oh6VStWBObmpSpKfVf...	1	USER
3	linsky.maksim2@gmail.com	0685811070	Олександр	$2a$10$Q9TcUQq9ofHDXpzmpgLzg.0xj5aka.Aa...	1	USER
4	ivan@gmail.com	0989856768	Іван	$2a$10$mhq3CJRV3D0WV2cSNXPWo.4XwsG/7...	1	USER
5	dmitro@gmail.com	0989853131	Дмитро	$2a$10$K1taXDP4BrNorpWJiYQTRuj9t.SNw0G...	1	USER
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Рис. 3.51 Таблиця users

Користувач отримав свій ідентифікатор, роль та статус. Колонка password містить хешований пароль користувача - пароль "Dmitro53421", що забезпечує високий рівень захисту від крадіжки облікових записів. Оскільки паролі зберігаються у вигляді хешів, навіть у випадку компрометації бази даних, зловмисники не зможуть відновити оригінальні паролі користувачів, що значно знижує ризик несанкціонованого доступу до облікових записів. Система надає ефективний захист від атак на паролі – алгоритм BCrypt, забезпечуючи конфіденційність та безпеку даних користувачів.

Було проведено тестування XSS (Cross-Site Scripting).

XSS (Cross-Site Scripting) – це перевірка на вразливість системи, яка дозволяє зловмисникам вставляти шкідливий код у поля на веб-сторінках, наприклад, при реєстрації або ж замовленні товару.

Для тестування задіяно наступний код: <script>alert('XSS')</script> (рис 3.52).

Рис. 3.52 Тестування XSS у полі пошуку

Цей код має інтерпретуватись та викликати функцію alert(), що викликає на сторінці, де був введений, спливаюче вікно і містити текст “XSS”.

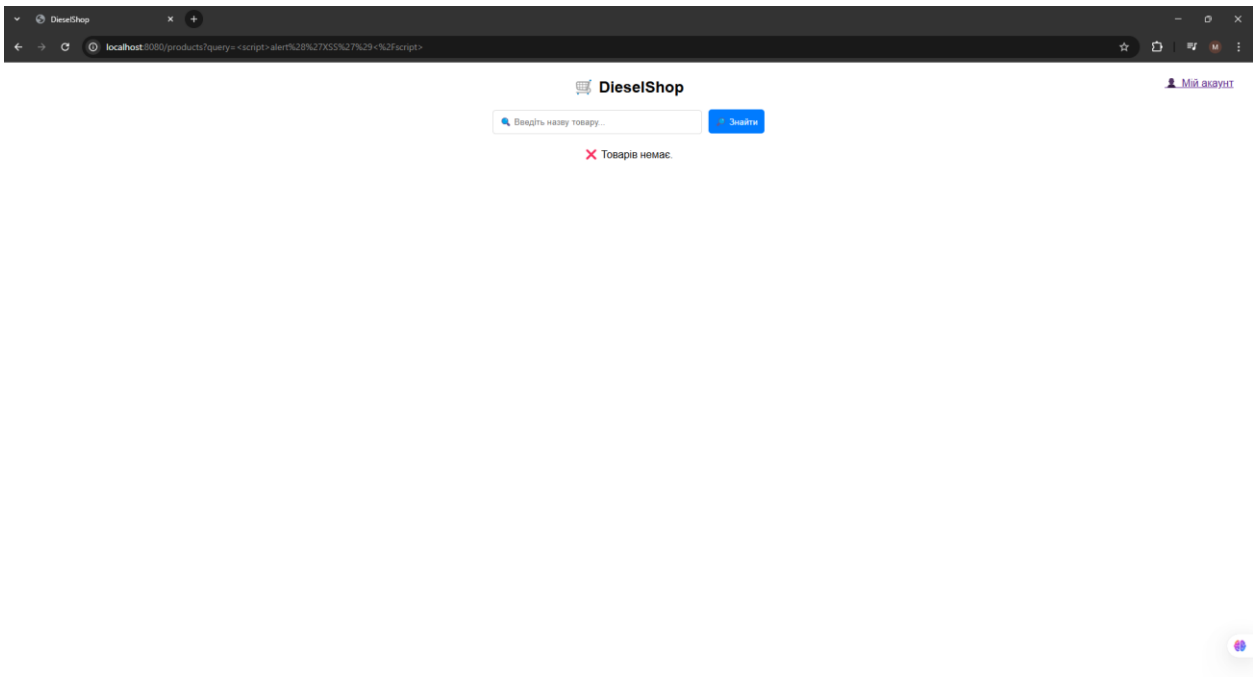


Рис. 3.53 Тестування XSS у полі пошуку пройшло успішно

Поле пошуку спрацювало коректно та показало, що товарів із такою назвою немає (рис. 3.54).

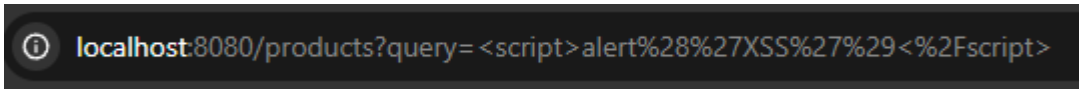


Рис. 3.54 URL-адреса після пошуку

Захист від даної вразливості включений у Spring Boot Security, програма виконала успішне URL-енкодування та не виконала зловмисний код:

- %3C — це закодоване значення символу <.
- %3E — це закодоване значення символу >.
- %28 — це закодоване значення символу (.
- %29 — це закодоване значення символу).
- %27 — це закодоване значення символу '.

Відповідно у решті місць, де є можливість вводити текст, програма спрацює аналогічно та декодує подібні шкідливі коди.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було успішно реалізовано прототип онлайн-магазину на основі фреймворку Spring Boot, що підтвердило доцільність вибору зазначеної технології для побудови сучасних веб-додатків. Проведене дослідження технічних особливостей Spring Boot, його інтеграції з базами даних та розробки REST API засвідчило високий потенціал даного інструментарію для створення масштабованих, гнучких і продуктивних інформаційних систем у сфері електронної комерції.

Розроблений застосунок відповідає архітектурі клієнт-сервер, реалізує дворівневу модель взаємодії компонентів та містить усі необхідні функціональні можливості для базової роботи онлайн-магазину — автентифікацію користувачів, керування товарами, оформлення замовлень і адміністрування. Проєктування структури бази даних із використанням MySQL забезпечило ефективне збереження й обробку інформації, що є критично важливим для стабільної роботи системи.

Результати розробки свідчать про практичну цінність обраного підходу, а також демонструють можливості подальшої модернізації системи — зокрема, впровадження мікросервісної архітектури, масштабування, інтеграції із зовнішніми платіжними системами та використання інструментів кешування.

Запропоноване рішення не лише задовольняє вимоги до типового онлайн-магазину, а й створює надійну основу для подальшого розширення та комерційного впровадження.

ПЕРЕЛІК ПОСИЛАНЬ

1. 2020 Java Technology Report | JRebel by Perforce. JRebel by Perforce. URL: https://www.jrebel.com/blog/2020-java-technology-report.

2. 2024 Stack Overflow Developer Survey. Stack Overflow Insights - Developer Hiring, Marketing, and User Research. URL: https://survey.stackoverflow.co/2024/.

3. Raible M. Redmonk Analyzes Java Framework Popularity. InfoQ. URL: https://www.infoq.com/news/2016/09/redmonk-java-frameworks/.

4. What are the advantages of using Spring Boot for web development?. Custom software development company. URL: https://moldstud.com/articles/p-what-are-the-advantages-of-using-spring-boot-for-web-development.

5. Сторінка Object Model & Factory в Selenium. Guru99. URL: https://www.guru99.com/uk/page-object-model-pom-page-factory-in-selenium-ultimate-guide.html.

6. GeeksforGeeks. Spring Boot - JDBC - GeeksforGeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/spring-boot-jdbc/.

7. What is JPA? Introduction to Java persistence. InfoWorld. URL: https://www.infoworld.com/article/2259807/what-is-jpa-introduction-to-the-java-persistence-api.html.

8. GeeksforGeeks. JPA - Architecture - GeeksforGeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/jpa-architecture/.

9. Umakant. Thymeleaf. Medium. URL: https://medium.com/thymeleaf-basics-concepts/thymeleaf-1ef952db0740.

10. What is JSP? Introduction to Jakarta Server Pages. InfoWorld. URL: https://www.infoworld.com/article/2258815/what-is-jsp-introduction-to-javaserver-pages.html.

11. Getting Started | Building REST services with Spring. Getting Started | Building REST services with Spring. URL: https://spring.io/guides/tutorials/rest.

12. Shikha R. Rest API Architecture. Medium. URL: https://medium.com/@shikha.ritu17/rest-api-architecture-6f1c3c99f0d3.

13. GeeksforGeeks. Client-Server Model - GeeksforGeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/client-server-model/.

14. Gupta H. Client-Server Architecture Explained with Examples, Diagrams, and Real-World Applications. Medium. URL: https://medium.com/nerd-for-tech/client-server-architecture-explained-with-examples-diagrams-and-real-world-applications-407e9e04e2d1.

15. N-tier architecture style - Azure Architecture Center. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/n-tier.

16. Koung R. Attributes in JPA annotations like @OneToMany, @ManyToOne, @ManyToMany, and @OneToOne. Medium. URL: https://medium.com/@raksmey_koung/attributes-in-jpa-annotations-like-onetomany-manytoone-manytomany-and-onetoone-3eb9ba754352.

17. Byte Wise 010. Spring Boot Identifiers. Medium. URL: https://medium.com/@bytewise010/spring-boot-identifiers-30454276449a.

18. Lahmidi M. T. @NoArgsConstructor, @AllArgsConstructor, and @RequiredArgsConstructor. Medium. URL: https://medium.com/@mtl98/noargsconstructor-allargsconstructor-and-requiredargsconstructor-60e11b338a6c.

19. Prasad H. JPA Annotations. Medium. URL: https://medium.com/@himani.prasad016/jpa-annotations-e3b5d749b547.

20. Jeong S. @GetMapping @PutMapping @PostMapping @DeleteMapping @PatchMapping Difference. Medium. URL: https://medium.com/@seonggil/spring-getmapping-putmapping-postmapping-deletemapping-patchmapping-difference-984a86520aad.

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)

Державний університет інформаційно-комунікаційних технологій
Кафедра інформаційних систем та технологій

КВАЛІФІКАЦІЙНА РОБОТА
на тему: «Онлайн-магазин на базі технології Spring Boot: проектування та реалізація»

на здобуття освітнього ступеня бакалавра
ні спеціальності 126 Інформаційні системи та технології
нсвітньо-професійної програми Інформаційні системи та технології

Виконав: здобувач вищої освіти гр. ІСД-43
Максим ЛІНСЬКИЙ
Науковий керівник:
Вікторія ЖЕБКА

Київ 2025

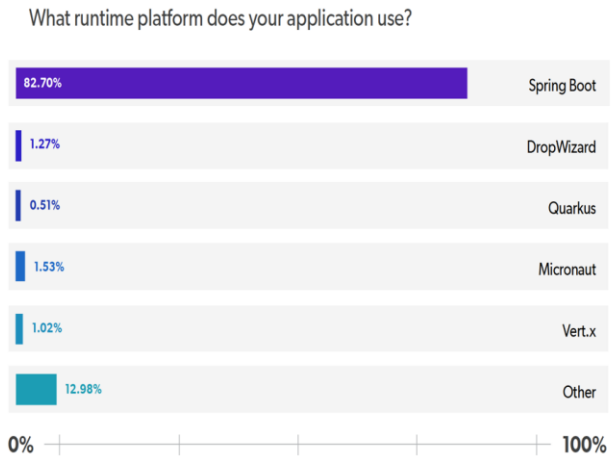
Мета, об’єкт та предмет дослідження

- **Мета роботи** - розробка прототипу онлайн-магазину на основі технології Spring Boot, що забезпечує ефективну взаємодію між клієнтом та сервером у клієнт-серверній архітектурі.
- **Об’єкт дослідження**: веб-додаток, створений із використанням фреймворку Spring Boot.
- **Предмет дослідження**: програмна архітектура та реалізація функціональних можливостей онлайн-магазину.

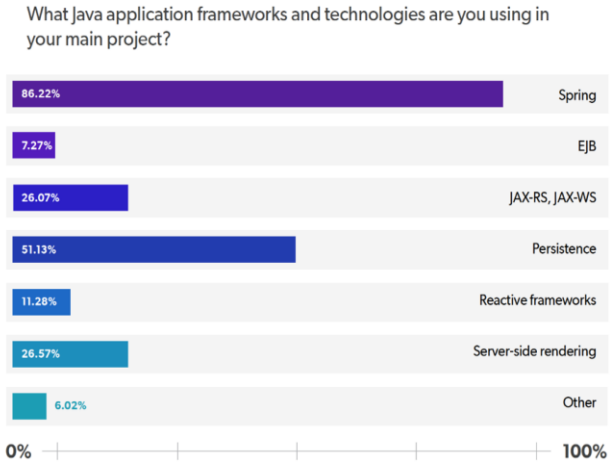
Актуальність теми

- Сучасна веб-розробка потребує гнучких, масштабних та надійних рішень, що можуть бути адаптовані до будь-яких стрімких змін у IT сфері. Spring Boot набуває особливого значення як інструмент, що сприяє спрощенню створення мікросервісних систем.
- Серед вагомих переваг Spring Boot: потужна продуктивність, простота використання, швидке розгортання проєкту, вбудована система захисту.

Результати опитувань розробників

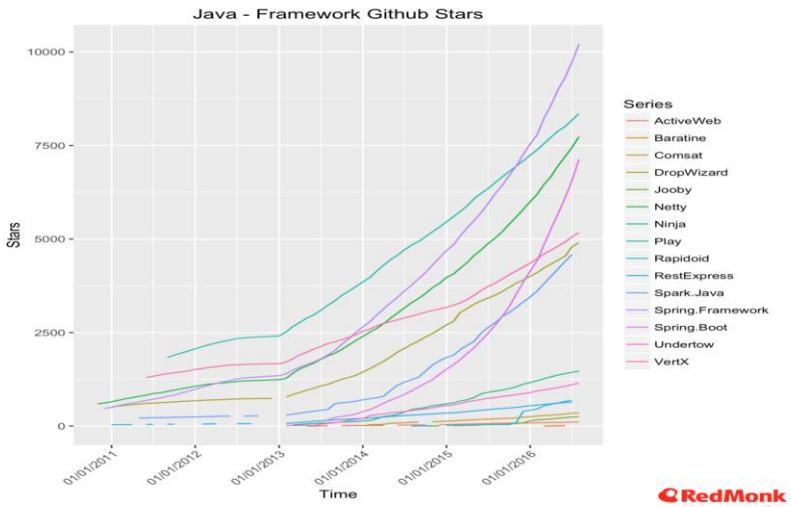


Опитування, проведене RebelLabs та JRebel на найпопулярнішу платформу розробки



Опитування, проведене ReberLabs та JRebel на найпопулярніший фреймворк

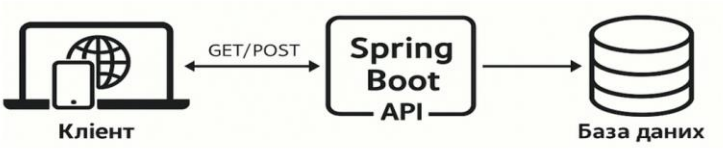
Результати опитувань розробників



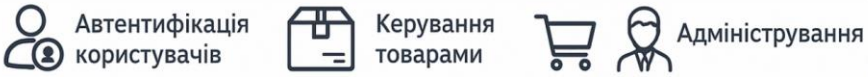
Із звіту RedMonk про фреймворки та їх популярність

Результат виконання кваліфікаційної роботи

- Результатом виконання кваліфікаційної роботи є успішно реалізований прототип онлайн-магазину на основі фреймворку Spring Boot, що підтвердило доцільність вибору зазначеної технології для побудови сучасних веб-додатків.



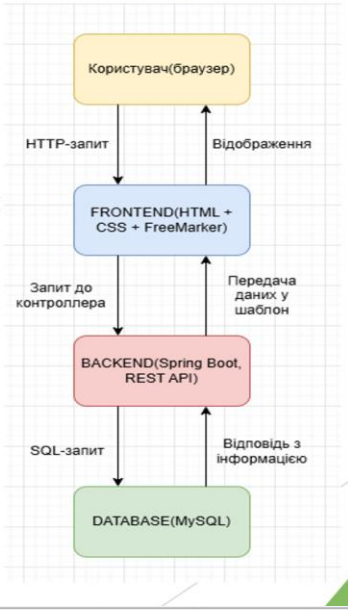
- Розроблений застосунок відповідає архітектурі клієнт-сервер, реалізує дворівневу модель взаємодії компонентів та містить усі необхідні функціональні можливості для базової роботи онлайн-магазину — автентифікацію користувачів, керування товарами, оформлення замовлень і адміністрування.



Використана модель клієнт-серверу

Перевагами такої моделі є:

- Масштабованість – можливість розширення система незалежно від клієнта.
- Гнучкість – можливість змінювати backend або frontend без впливу один на одного.
- Безпека – повний контроль доступу сервера до даних, в той час як клієнт не має прямого доступу.
- Швидка оптимізована робота між сервером через API та клієнтом.



База даних - MySQL

id	email	phone_number	name	password	active	role
1	linsky.maksim@gmail.com	0989856767	Максим Лінський	$2a$10$kXRi5qZFbIzvQk4IIlPJB.HHHJHlXPYF/lD...	1	USER
2	petro@gmail.com	0685811071	Петро	$2a$10$wj/UxpFOuzFP2oh6VStWBObmpSpKfVf...	1	USER
3	linsky.maksim2@gmail.com	0685811070	Олександр	$2a$10$Q9TcUQq9ofHDXpzmpgLzg.0xj5aka.Aa...	1	USER
4	ivan@gmail.com	0989856768	Іван	$2a$10$mhq3CJRV3D0WV2cSNXPWo.4XwsG/7...	1	USER
5	dmitro@gmail.com	0989853131	Дмитро	$2a$10$K1taXDP4BrNorpWJiYQTRuj9t.SNw0G...	1	USER
NULL	NULL	NULL	NULL	NULL	NULL	NULL

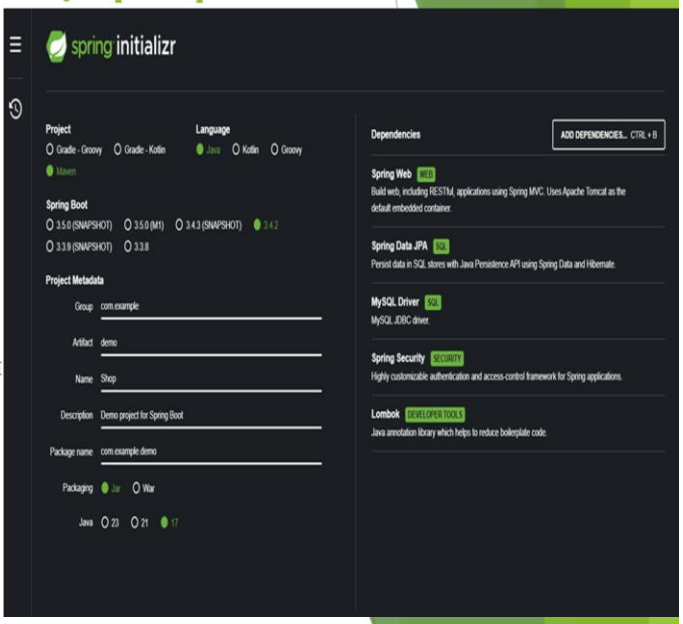
Приклад однієї з таблиць (Users) у базі даних MySQL

- **Для прототипу інтернет-магазину створено три таблиці бази даних:**

Orders	Products	Users
• ID • Місто • Ім'я • Номер телефону • ID товару • ID користувача • Статус	• ID • Назва товару • Розмір екрану • Технологія дисплею • Ціна • Опис • Колонка з фото • Тип дисплею • Роздільна здатність • Розмір	• ID • Електронна пошта • Номер телефону • Ім'я • Пароль • Активність користувача • Роль

Налаштування середовища розробки

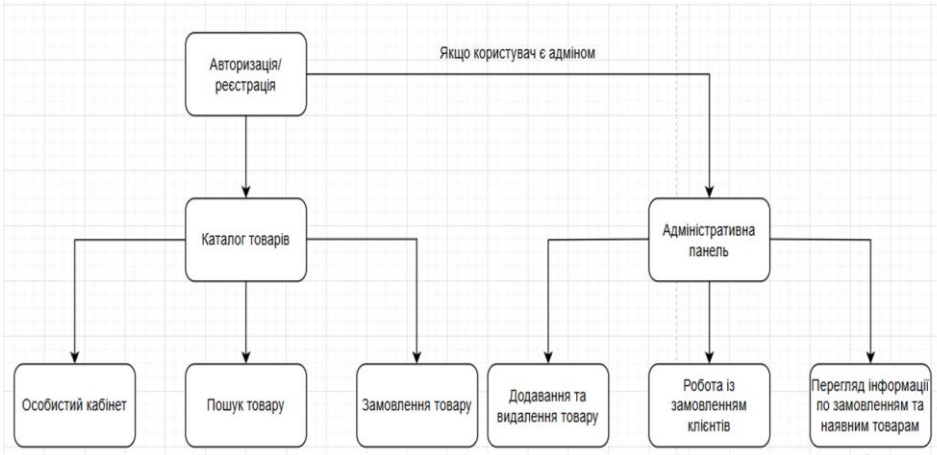
- Одним з головних етапів налаштування середовища розробки є налаштування Spring Initializr.
- Додані залежності:
  - Spring Web – для створення веб-застосунку з REST API (Spring MVC).
  - Spring Data JPA – для роботи з базами даних через ORM.
  - MySQL Driver – драйвер для підключення до бази даних MySQL.
  - Spring Security – для реалізації автентифікації й авторизації.
  - Lombok – для зменшення шаблонного коду за допомогою анотацій.



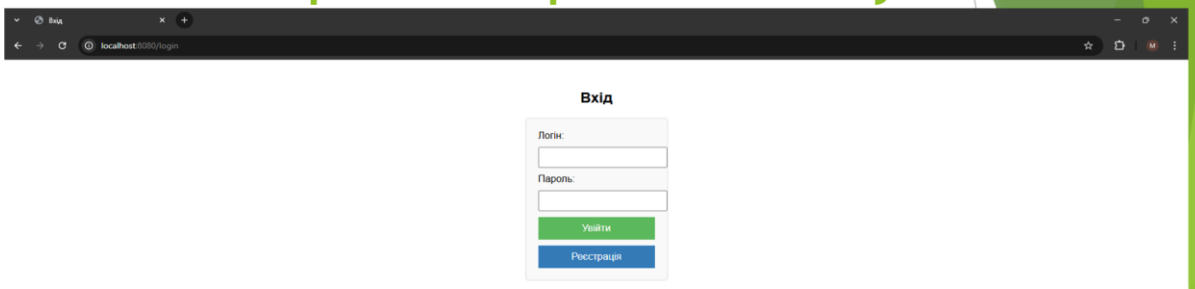
Функціональність системи

Користувач	Товари	Замовлення	Адміністратор
• Реєстрація аккаунту • Вхід(автентифікація) через Spring Security • Особистий кабінет із переглядом замовлень	• Перегляд списку товарів із зображенням, описом та цінами • Додавання товарів до замовлення	• Формування та підтвердження замовлення із заповненням контактної інформації • Перегляд історії замовлень для кожного користувача через особистий кабінет	• Додавання, редагування та видалення товарів • Перегляд усіх замовлень та позначення їх як виконаних або скасованих

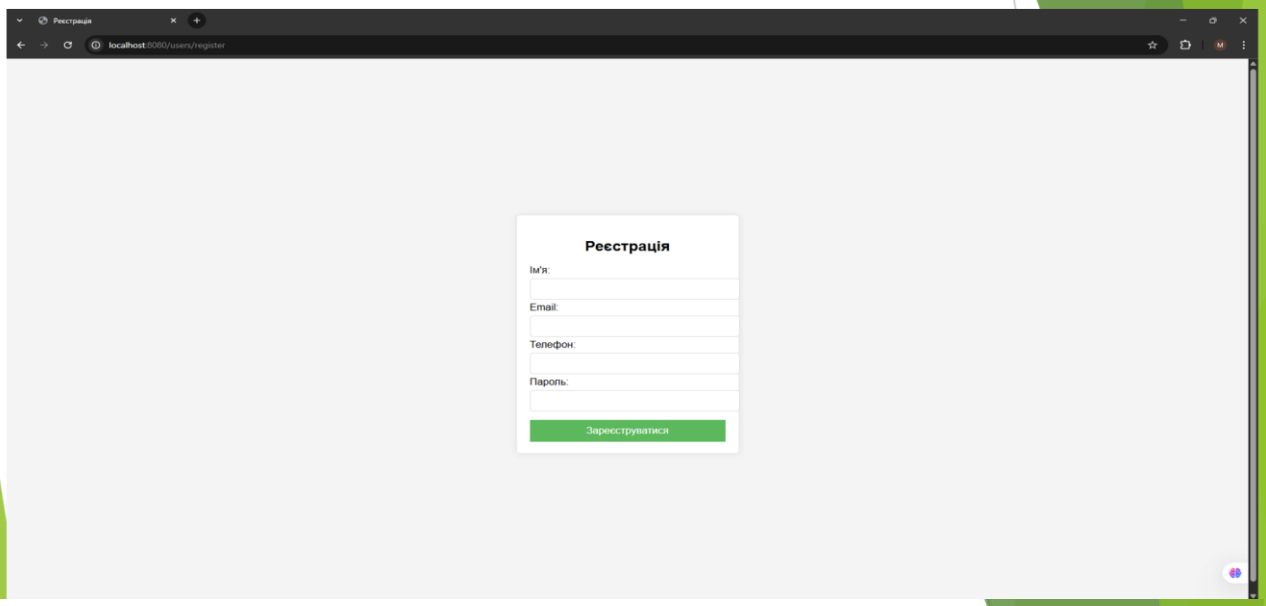
Структура онлайн-магазину



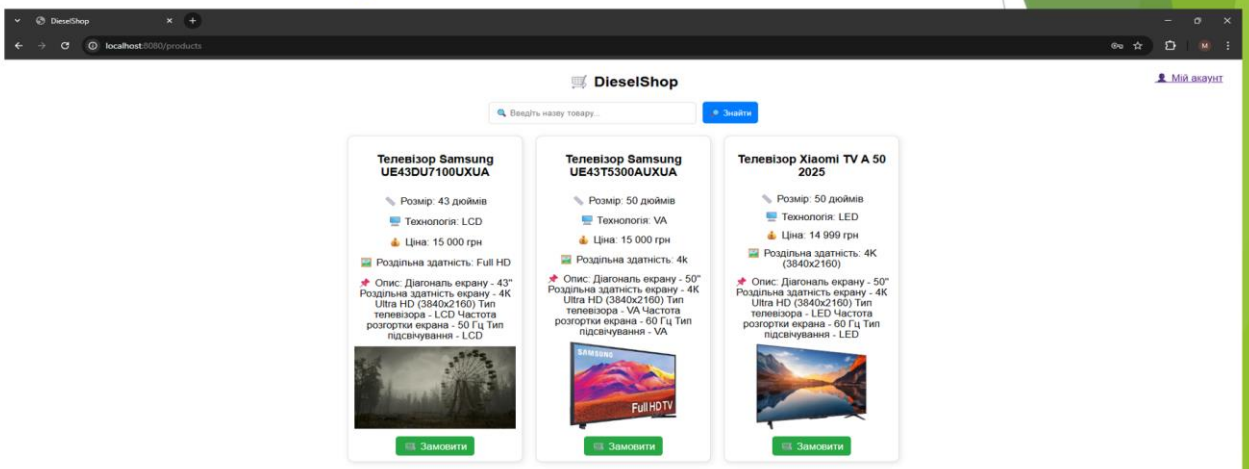
Головна сторінка інтернет-магазину



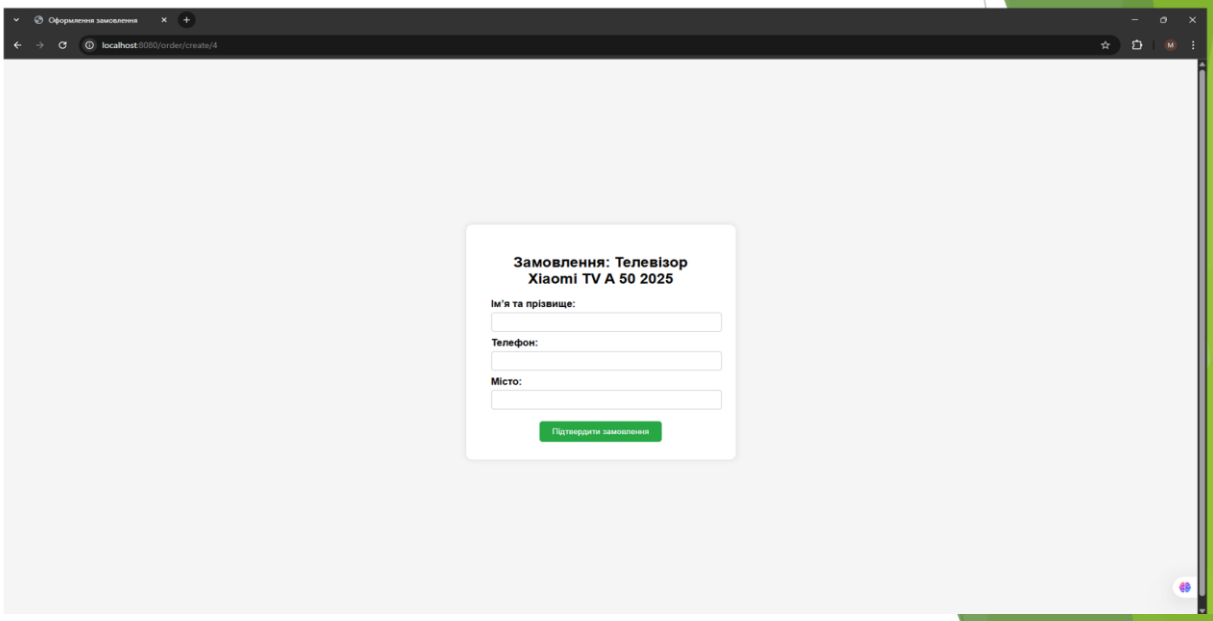
Сторінка реєстрації



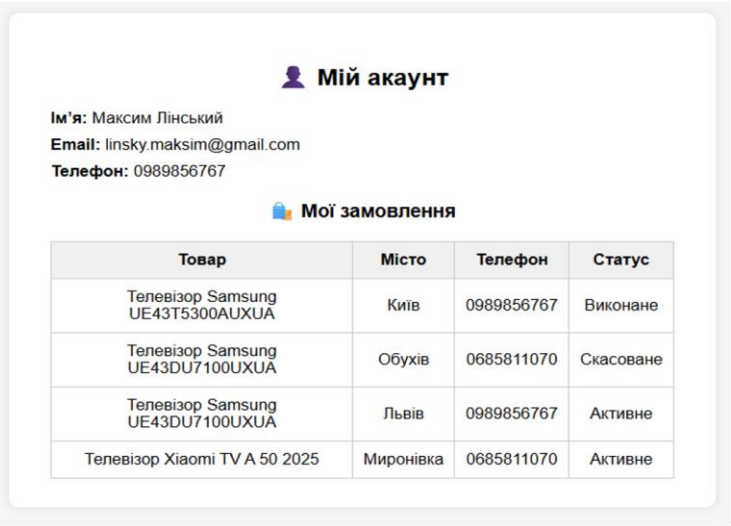
Каталог товарів



Сторінка оформлення замовлення



Особистий кабінет користувача



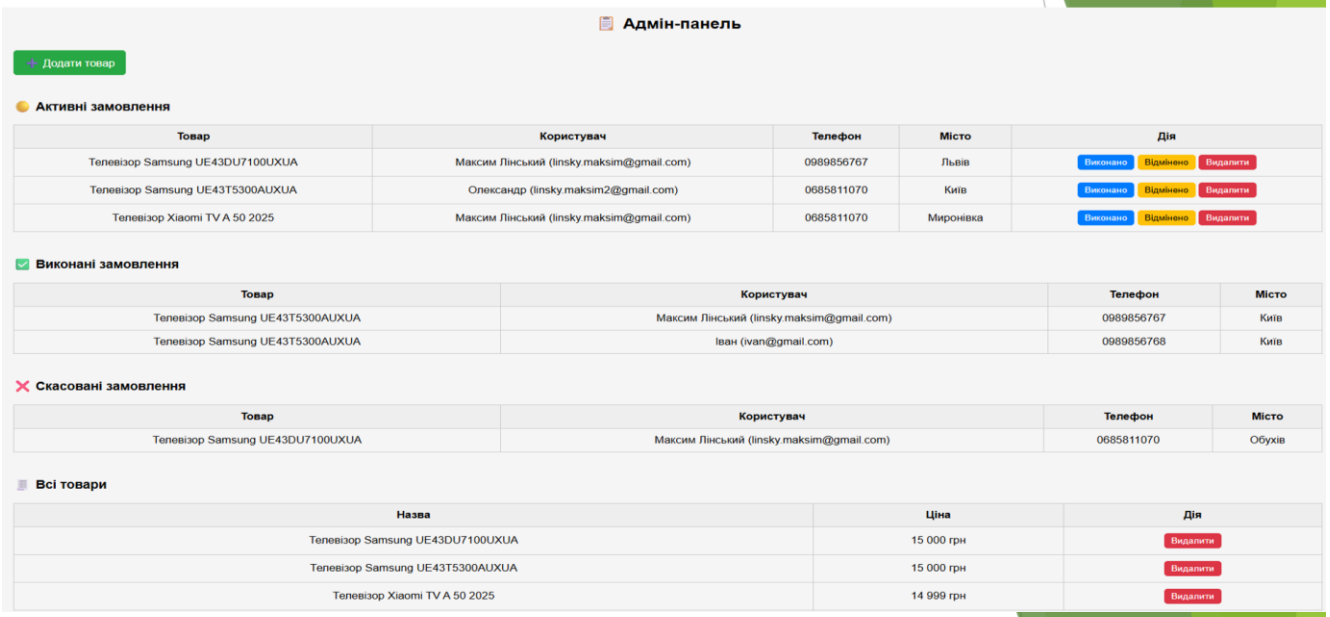
Мій акаунт

Ім'я: Максим Лінський
Email: linsky.maksim@gmail.com
Телефон: 0989856767

Мої замовлення

Товар	Місто	Телефон	Статус
Телевізор Samsung UE43T5300AUXUA	Київ	0989856767	Виконане
Телевізор Samsung UE43DU7100UXUA	Обухів	0685811070	Скасоване
Телевізор Samsung UE43DU7100UXUA	Львів	0989856767	Активне
Телевізор Xiaomi TV A 50 2025	Миронівка	0685811070	Активне

Адміністративна панель



Адмін-панель

Додати товар

Активні замовлення

Товар	Користувач	Телефон	Місто	Дія
Телевізор Samsung UE43DU7100UXUA	Максим Лінський (linsky.maksim@gmail.com)	0989856767	Львів	Виконано Відмінено Видалити
Телевізор Samsung UE43T5300AUXUA	Олександр (linsky.maksim2@gmail.com)	0685811070	Київ	Виконано Відмінено Видалити
Телевізор Xiaomi TV A 50 2025	Максим Лінський (linsky.maksim@gmail.com)	0685811070	Миронівка	Виконано Відмінено Видалити

Виконані замовлення

Товар	Користувач	Телефон	Місто
Телевізор Samsung UE43T5300AUXUA	Максим Лінський (linsky.maksim@gmail.com)	0989856767	Київ
Телевізор Samsung UE43T5300AUXUA	Іван (ivan@gmail.com)	0989856768	Київ

Скасовані замовлення

Товар	Користувач	Телефон	Місто
Телевізор Samsung UE43DU7100UXUA	Максим Лінський (linsky.maksim@gmail.com)	0685811070	Обухів

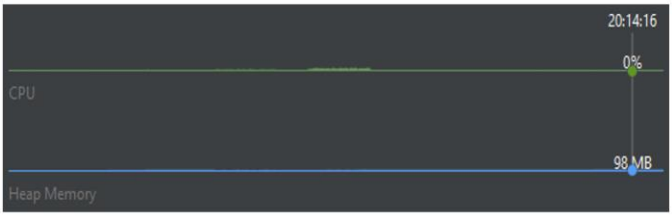
Всі товари

Назва	Ціна	Дія
Телевізор Samsung UE43DU7100UXUA	15 000 грн	Видалити
Телевізор Samsung UE43T5300AUXUA	15 000 грн	Видалити
Телевізор Xiaomi TV A 50 2025	14 999 грн	Видалити

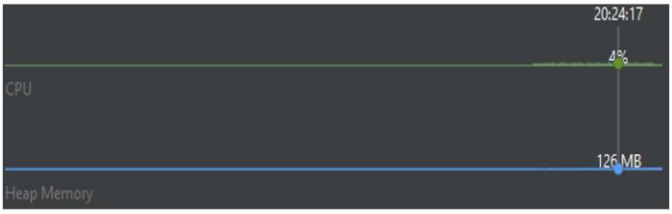
Тестування інтернет-магазину

- Проведено такі тестування як: DDos-атака, XSS-коди та головні наявні функції, що може використовувати користувач.

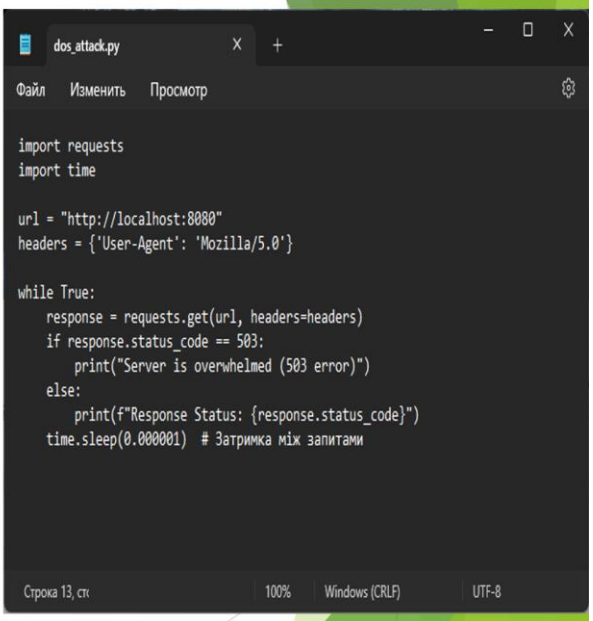
Тестування навантаженості системи під час DDos-атаки



Показники навантаженості до атаки

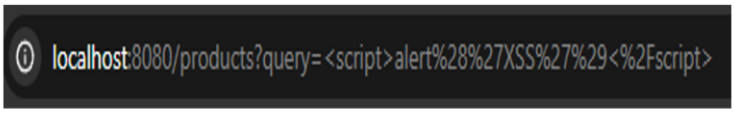


Показники навантаженості під час атаки



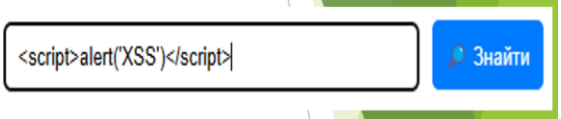
Використаний файл для тестування DDos-атаки

Тестування XSS (Cross-Site Scripting)



Захист від даної вразливості включений у Spring Boot Security, програма виконала успішне URL-енкодування та не виконала зловмисний код:

- %3C — це закодоване значення символу <.
- %3E — це закодоване значення символу >.
- %28 — це закодоване значення символу (.
- %29 — це закодоване значення символу).
- %27 — це закодоване значення символу '.



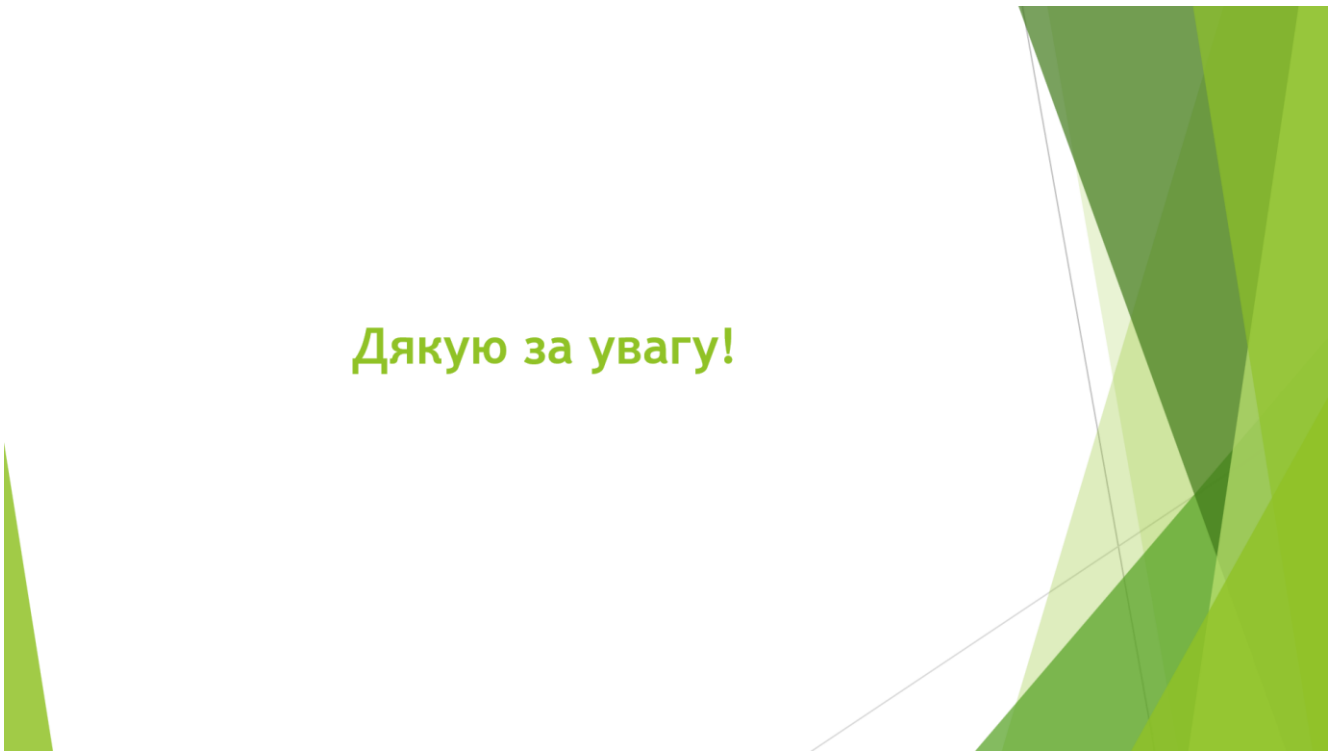
Скрипт вставлений у поле пошуку

Апробація результатів роботи

- II Всеукраїнська науково-технічна конференція «Технологічні горизонти: дослідження та застосування інформаційних технологій для технологічного прогресу України і світу»
- ТЕЗИ “ВИКОРИСТАННЯ SPRING BOOT ДЛЯ ПРИСКОРЕННЯ РОЗРОБКИ ВЕБ-ДОДАТКІВ ЗА ДОПОМОГОЮ АВТОКОНФІГУРАЦІЇ ТА ВБУДОВАНИХ ІНСТРУМЕНТІВ”

Висновки

- Розроблений веб-додаток — це прототип інтернет-магазину, реалізований із використанням сучасного фреймворку Spring Boot, який забезпечив гнучкість, масштабованість та ефективність взаємодії в архітектурі клієнт-сервер.
- У межах проєкту була реалізована повна функціональність: реєстрація та автентифікація користувачів, управління товарами, обробка замовлень, а також адміністрування.В якості бази даних використано MySQL, що продемонструвала свою надійність у поєднанні з Spring Data JPA.
- Особливу увагу було приділено безпеці: система успішно пройшла тестування на захист від XSS-атак, а також витримала навантаження під час імітації DDoS-атаки.Проведені тести підтвердили надійність реалізованої архітектури та функціональність кожного з компонентів.


Дякую за увагу!