Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Інститут комп’ютерних наук та інформаційних технологій

Кафедра системного аналізу та інформаційно-аналітичних технологій

Спеціальність Комп’ютерні науки

Освітня програма Комп’ютерні науки

До захисту допускаю

Завідувач кафедри

Юрій ДОРОФЄЄВ
(Ім’я та ПРІЗВИЩЕ)

(підпис, дата)

ДИПЛОМНА РОБОТА

першого (бакалаврського) рівня вищої освіти

Тема роботи РОЗРОБКА МЕРЕЖЕВОЇ СИСТЕМИ ОБЛІКУ ТОВАРІВ НЕВЕЛИКОЇ КОМПАНІЇ

Шифр роботи КН-320В.13
(група, номер теми за наказом)

Виконавець Молчанов Богдан Сергійович
(прізвище, ім’я та по-батькові)

Керівник доцент Роговий Антон Іванович
(посада, прізвище, ім’я та по-батькові)

Харків 2024

Найменування виробу, об’єкта або теми	Найменування документа	Формат	Кільк. арк.	Прим.
	Документи загальні			
	Завдання на виконання ДР	А4	1	
	Пояснювальна записка			
	до ДР	А4	60	
	Ілюстративні матеріали			
Розробка мережевої системи	Презентація	А4	23	
обліку товарів невеликої				
компанії				

				КН-320В .13 ВД
	Прізвище	Підп.	Дата	
Розроб.	Молчанов			Розробка мережевої системи обліку товарів невеликої компанії Відомість документів
Перев.	Роговий			
Н.конт.	Роговий			
Затв.	Дорофєєв			
Реценз.				

Літ.			Аркуш	Аркушів
Д	Р	Б		1

НТУ «ХПІ»
Кафедра САІТ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Інститут комп’ютерних наук та інформаційних технологій

Кафедра системного аналізу та інформаційно-аналітичних технологій

Рівень вищої освіти перший (бакалаврський)

Спеціальність Комп’ютерні науки

Освітня програма Комп’ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри

Юрій ДОРОФЄЄВ

(підпис) (Ім’я та ПРІЗВИЩЕ)

« ____ » _____ 20 ___ року

ЗАВДАННЯ
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Молчанову Богдану Сергійовичу

(прізвище, ім’я та по-батькові)

1. Тема роботи РОЗРОБКА МЕРЕЖЕВОЇ СИСТЕМИ ОБЛІКУ ТОВАРІВ НЕВЕЛИКОЇ КОМПАНІЇ

Керівник роботи Антон Іванович Роговий, к. т. н. , доцент

(прізвище, ім’я та по-батькові, науковий ступень, вчене звання)

затверджена наказом по НТУ «ХПІ» від « 24 » квітня 20 24 р. № 738

2. Строк подання студентом роботи 05.06.2024

3. Вихідні дані для роботи Технічна література по Jakarta EE, MySQL, HTML, CSS; MySQL Connector/J; фреймворк Bootstrap 5

4. Перелік питань, які потрібно розробити у пояснювальній записці

Вступ, 1 Постановка задачі, 2 Обґрунтування вибору інструментів,

3 Методи та алгоритми розв’язання задачі, 4 Опис програмної реалізації,

5 Тестування, 6 Аналіз результатів, Висновки, Список джерел інформації

5. Перелік графічного матеріалу (з точнім зазначенням обов’язкових креслень)

Презентація: 23 слайди

6. Консультанті розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 06 березня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

Номер етапу	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітки
1	Вибір і обґрунтування теми, постановка проблем і завдань	10.03.2024	
2	Аналітичний огляд джерел, вибір методики досліджень	01.04.2024	
3	Підготовка висновків	15.05.2024	
4	Підготовка і виконання пояснювальної записки	25.05.2024	
5	Складання відомості документів оформлення ПЗ	31.05.2024	
6	Виконання демонстраційних матеріалів та доповіді	03.06.2024	
7	Подання ДР на відгук, перевірку ступені авторства	05.06.2024	
8	Подання ДР на допуск до захисту	09.06.2024	
9	Захист ДР	20.06.2024	

Студент _____ Богдан МОЛЧАНОВ

(підпис) (Ім’я та ПРІЗВИЩЕ)

Керівник роботи _____ Антон РОГОВИЙ

(підпис) (Ім’я та ПРІЗВИЩЕ)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Інститут комп’ютерних наук та інформаційних технологій

Кафедра системного аналізу та інформаційно-аналітичних технологій

Спеціальність Комп’ютерні науки

Освітня програма Комп’ютерні науки

ПОЯСНЮВАЛЬНА ЗАПИСКА
до дипломної роботи

першого (бакалаврського) рівня вищої освіти

на тему Розробка мережевої системи обліку товарів невеликої компанії

Виконав студент 4 курсу, групи КН-320
Богдан МОЛЧАНОВ
(підпис, Ім’я та ПРІЗВИЩЕ)

Керівник Антон РОГОВИЙ
(підпис, Ім’я та ПРІЗВИЩЕ)

Рецензент
(підпис, Ім’я та ПРІЗВИЩЕ)

Нормоконтролер Антон РОГОВИЙ
(підпис, Ім’я та ПРІЗВИЩЕ)

Харків 2024

РЕФЕРАТ

Пояснювальна записка до ДР: 67 с., 18 рис., 2 табл., 12 джерел, 2 додатків.

Ключові слова: ФОП, СИСТЕМИ ОБЛІКУ, СУБД, СЕРВЕРНІ ПЛАТФОРМИ, ФРЕЙМВОРКИ

Об’єкт дослідження: Мережева система обліку товарів.

Мета роботи: Розробка мережевої системи обліку товарів для невеликої компанії.

Методи: Обробка HTTP-запитів і реалізація бізнес-логіки на серверній стороні за допомогою Java сервлетів, забезпечення інтерактивності та динамічної взаємодії з користувачем, маніпулювання товарами.

Результати: Програмне забезпечення для мережевого обліку товарів.

Ця дипломна робота присвячена розробці мережевої системи обліку товарів для невеликої компанії. Основна мета проекту полягає в створенні ефективної та інтуїтивно зрозумілої системи, яка забезпечить зручний облік товарів, керування категоріями товарів, облік продажів, управління користувачами та аналітику. Для досягнення цієї мети було проаналізовано існуючі системи обліку товарів, такі як Odoo, Zoho Inventory та TradeGecko (QuickBooks Commerce), щоб визначити їх переваги та недоліки і врахувати їх при розробці власного рішення.

На основі аналізу було обрано інструменти для розробки системи, включаючи MySQL для управління базами даних, Jakarta EE для реалізації серверної частини та Bootstrap 5 для створення адаптивного та сучасного користувацького інтерфейсу. Архітектура клієнт-сервер, з використанням сервлетів та JavaServer Pages (JSP), забезпечує ефективну обробку запитів і виконання бізнес-логіки.

Розробка структури бази даних базується на принципах нормалізації для забезпечення цілісності та уникнення надлишкових даних. ER-діаграми використовувалися для візуалізації структури бази даних, що включає сутності, їх атрибути та взаємозв'язки. Основні SQL-запити були розроблені для створення, модифікації та управління даними в базі даних.

Серверна частина системи включає веб-сервер для обробки HTTP-запитів, сервер додатків для виконання бізнес-логіки та базу даних для зберігання інформації. Сервлети та JSP забезпечують динамічне генерування веб-сторінок і обробку запитів користувачів, що включає управління сесіями та аутентифікацію користувачів для забезпечення безпеки.

Клієнтська частина реалізована за допомогою HTML, CSS та JavaScript, з використанням Bootstrap 5 для створення адаптивного інтерфейсу. Інтерактивність забезпечується за допомогою JavaScript та AJAX, що дозволяє виконувати асинхронні запити до серверної частини, забезпечуючи швидку взаємодію з користувачем. Валідація форм на стороні клієнта забезпечує коректність введених даних перед їх відправкою на сервер.

В результаті роботи було створено ефективну систему обліку товарів, яка відповідає сучасним вимогам та забезпечує зручний і інтуїтивно зрозумілий інтерфейс для користувачів, швидке та надійне управління даними.

ABSTRACT

Explanatory note for DW: 67 p., 18 pic., 2 table, 12 source, 2 appl.

Keywords: SOLE PROPRIETOR, ACCOUNTING SYSTEMS, DBMS, SERVER PLATFORMS, FRAMEWORKS.

Object of the research: Network-based inventory management system.

Goal of the work: Development of a network-based inventory management system for a small company.

Methods: Processing HTTP requests and implementing business logic on the server side using Java servlets, ensuring interactivity and dynamic user interaction, manipulating inventory items.

Results: Software for network-based inventory management.

This thesis is dedicated to the development of a network-based inventory management system for a small company. The main goal of the project is to create an efficient and user-friendly system that provides convenient inventory tracking, category management, sales accounting, user management, and analytics. To achieve this goal, existing inventory management systems such as Odoo, Zoho Inventory, and TradeGecko (QuickBooks Commerce) were analyzed to identify their strengths and weaknesses and incorporate these insights into the development of the new solution.

Based on the analysis, tools for the system development were selected, including MySQL for database management, Jakarta EE for implementing the server-side components, and Bootstrap 5 for creating a responsive and modern user interface. The client-server architecture, utilizing servlets and JavaServer Pages (JSP), ensures efficient request handling and execution of business logic.

The database structure was designed based on normalization principles to ensure integrity and avoid data redundancy. ER diagrams were used to visualize the

database structure, including entities, their attributes, and relationships. Key SQL queries were developed for creating, modifying, and managing the data in the database.

The server-side system components include a web server for handling HTTP requests, an application server for executing business logic, and a database for storing information. Servlets and JSP provide dynamic web page generation and request handling, including session management and user authentication to ensure security.

The client-side implementation uses HTML, CSS, and JavaScript, with Bootstrap 5 for creating a responsive interface. Interactivity is achieved through JavaScript and AJAX, enabling asynchronous requests to the server-side, ensuring fast user interaction. Client-side form validation ensures the correctness of the data before it is sent to the server.

As a result of this work, an efficient inventory management system was created that meets modern requirements and provides a convenient and intuitive user interface, along with fast and reliable data management.

ЗМІСТ

ПЕРЕЛІК ПОЗНАК І СКОРОЧЕНЬ

ФОП	Фізична особа — підприємець
СУБД	Система управління базами даних
IDE	Інтегроване середовище розробки

ВСТУП

У сучасному світі з розвинутими вільного ринку у людей з’являються думки про створення власної справи. Серед безлічі сфер діяльності 25% серед ФОПів займає роздрібна торгівля.

ФОПи часто обирають роздрібну торгівлю через кілька ключових причин:

– низький поріг входу;
– гнучкість та незалежність;
– широкий ринок;
– відносно просте регулювання;
– швидкий оборот коштів;
– можливість масштабування.

Попри численні переваги, власники роздрібних підприємств стикаються з численними викликами, серед яких ключовою проблемою є ефективний облік товарів. Традиційні методи обліку, такі як паперові записи чи прості електронні таблиці, часто є недостатніми для забезпечення точного та своєчасного управління запасами, особливо в умовах зростаючого обсягу даних і динамічних ринкових умов. Помилки в обліку можуть призвести до дефіциту чи надлишку товарів, що спричиняє фінансові втрати, погіршення обслуговування клієнтів та зниження конкурентоспроможності.

Впровадження мережевих систем обліку товарів дозволяє спростити процеси управління запасами, знижуючи ризик помилок і підвищуючи ефективність операцій. Такі системи забезпечують централізоване зберігання даних, доступ до них у реальному часі і аналітику продажів. Це особливо важливо для невеликих компаній, які не можуть дозволити собі великі штати для управління запасами і потребують компактних, але потужних рішень.

Метою цієї дипломної роботи розробка мережевої системи обліку товарів для невеликої компанії і це є актуальним і важливим завданням, яке дозволить

підвищити ефективність управління запасами, знизити операційні витрати і забезпечити конкурентні переваги на ринку.

1 ПОСТАНОВКА ЗАДАЧІ

1.1 **Аналіз існуючих систем обліку товарів**

Для ефективної розробки нової системи обліку товарів необхідно провести аналіз існуючих рішень на ринку, щоб визначити їх переваги та недоліки. Огляд декількох популярних систем допоможе зрозуміти, які функціональні можливості та архітектурні підходи є найбільш ефективними та актуальними для нашої задачі.

1.1.1 **Існуючі системи обліку товарів**

1.1.1.1 **Odoo**

Переваги:

– інтеграція: Одна з найбільших переваг Odoo - це інтеграція з іншими модулями, такими як CRM, бухгалтерія та управління проектами.

– гнучкість: Odoo пропонує велику кількість модулів, що дозволяють налаштовувати систему під конкретні потреби бізнесу.

– відкритий код: Odoo є системою з відкритим кодом, що дозволяє розробникам вносити необхідні зміни.

Недоліки:

– складність: Може бути складною у налаштуванні та використанні, особливо для невеликих компаній без досвіду роботи з ERP-системами.

– вартість: Хоча основні модулі доступні безкоштовно, додаткові функції можуть бути дорогими.

1.1.1.2 **Zoho Inventory**

Переваги:

– інтуїтивний інтерфейс: Легкий у використанні та має дружній інтерфейс, що не потребує значного навчання.

– інтеграція з іншими Zoho продуктами: Добре інтегрується з іншими продуктами Zoho, такими як CRM, бухгалтерія та аналітика.

Недоліки:

– обмежена функціональність: Можливості системи можуть бути недостатніми для компаній зі складними потребами в обліку товарів.

– вартість: Безкоштовна версія має обмежені можливості, а повнофункціональні версії потребують щомісячної абонентської плати.

1.1.1.3 **TradeGecko (тепер QuickBooks Commerce)**

Переваги:

– інтуїтивність: Простий та зручний інтерфейс, що дозволяє швидко навчити персонал.

– функціональність: Містить всі необхідні функції для обліку товарів, управління запасами та інтеграції з e-commerce платформами.

Недоліки:

– ціна: Відносно висока вартість для невеликих компаній.

– обмеження: Деякі функції можуть бути недостатньо гнучкими для специфічних потреб користувачів.

1.2 **Постановка задачі**

Метою цієї дипломної роботи є розробка мережевої системи обліку товарів для невеликої компанії. Для досягнення цієї мети необхідно чітко сформулювати та виконати наступні задачі:

Задачі, що потрібно вирішити для досягнення мети:

– вибір інструментів для розробки мережевої системи обліку товарів;

– розробка структури бази даних;

– розробка серверної частини системи;

– розробка клієнтської частини системи.

До функціональних вимог програмного забезпечення, яке розробляється, відносять:

– облік товарів;

– керування категоріями товарів;

– облік продажів;

– керування користувачами;

– аналітика;

– інтерфейс користувача.

Більш детальний опис функціональних вимог наведений у Таблиця 1.1.

Таблиця 1.1 - Детальний опис функціональних вимог

Функціональні вимоги	Опис
Облік товарів	Введення нових товарів до бази даних, редагування інформації про існуючі товари, видалення товарів з бази даних, відстеження кількості товарів на складі.
Керування категоріями товарів	Створення нових категорій, редагування категорій, видалення категорій.
Облік продажів	Реєстрація продажів товарів клієнтам, відстеження історії постачань та продажів.
Керування користувачами	Реєстрація нових користувачів, аутентифікація користувачів при вході в систему, налаштування прав доступу для різних користувачів, редагування інформації про користувачів, видалення користувачів.
Аналітика	Відображення ключових показників продуктивності
Інтерфейс користувача	Інтуїтивно зрозумілий та зручний веб-інтерфейс для взаємодії з системою.

До переліку вхідних даних мають входити:

– інформація про товари;

– інформація про категорії;

– інформація про продажі;

– інформація про користувачів.

Більш детальний опис функціональних вимог наведений у Таблиця 1.2.

Таблиця 1.2 – Опис вхідних даних

Функціональні вимоги	Опис
Інформація про товари	Назва товару, опис товару, категорія товару, ціна, кількість на складі.
Інформація про категорії	Назва категорії.
Інформація про продажі	Дата продажу, клієнт, перлік проданих товарів, сума продажу.
Інформація про коистувачів	Логін, пароль, роль користувача.

Всі розглянуті системи у попередньому підрозділі мають свої переваги та недоліки. Наприклад, Odoo відрізняється великою кількістю функцій та модулів, але може бути складною у використанні для невеликих компаній. Zoho Inventory зручний та інтегрований з іншими продуктами Zoho, але має обмежену функціональність у безкоштовній версії. TradeGecko пропонує інтуїтивний інтерфейс та потужний функціонал, але за високу ціну.

Для розробки нашої мережевої системи обліку товарів варто врахувати:

– інтуїтивний інтерфейс користувача: Це дозволить знизити витрати на навчання персоналу.

– модульність: Система повинна бути легко масштабованою та адаптивною до змін у бізнес-процесах.

– вартість: Оптимізація витрат на розробку та впровадження системи, особливо для невеликих компаній.

Таким чином, на основі аналізу існуючих систем та врахування специфічних потреб невеликої компанії, розробка нової мережевої системи обліку товарів буде здійснена з урахуванням найкращих практик та врахуванням виявлених недоліків існуючих рішень.

2 ОБҐРУНТУВАННЯ ВИБОРУ ІНСТРУМЕНТІВ

2.1 **Обґрунтування вибору технологій**

Для розробки мережевої системи обліку товарів важливо обрати відповідні інструменти та технології, які забезпечать надійність, продуктивність та масштабованість рішення. В даному проекті обрано середовище розробки IntelliJ IDEA, технологію Jakarta EE для створення серверної частини системи, MySQL для створення бази даних, Bootstrap 5 для створення інтерфейсу користувача і JavaScript для створення клієнтської частини.

2.1.1 **Intellij IDEA**

IntelliJ IDEA було обрано як інтегроване середовище розробки (IDE) з наступних причин:

– IntelliJ IDEA забезпечує високу продуктивність завдяки інтелектуальним підказкам, автоматичному завершенню коду та підтримці рефакторингу;

– IDE повністю підтримує розробку на Java та Jakarta EE, включаючи інтеграцію з популярними серверами додатків та системами управління версіями;

– IntelliJ IDEA легко інтегрується з іншими інструментами розробки, такими як Maven, Gradle та Docker, що робить процес розробки більш гнучким та масштабованим.

2.1.2 **Jakarta EE**

Jakarta EE було обрано для створення серверної частини системи з наступних причин:

– Jakarta EE є стандартом для розробки корпоративних додатків на Java, що забезпечує стабільність та тривалу підтримку;

– платформа забезпечує високий рівень масштабованості та надійності завдяки вбудованим механізмам обробки транзакцій, безпеки та управління ресурсами;

– платформа надає широкий набір API для роботи з базами даних, веб-сервісами, обробки запитів та багато іншого, що спрощує процес розробки.

2.1.3 **MySQL**

Для зберігання даних було обрано MySQL з наступних причин:

– MySQL є однією з найпопулярніших систем управління базами даних, яка забезпечує високу продуктивність та надійність;

– MySQL підтримує великі обсяги даних і високу продуктивність, що важливо для систем обліку товарів;

– MySQL має відкритий код та велику спільноту розробників, що забезпечує регулярні оновлення та підтримку.

2.1.4 **JavaScript**

Для розробки клієнтської частини системи було обрано JavaScript з наступних причин:

– JavaScript є основною мовою програмування для веб-розробки і підтримується всіма сучасними веб-браузерами;

– JavaScript має велику спільноту розробників та широкий набір бібліотек і фреймворків, що полегшує процес розробки та обслуговування додатка;

– JavaScript дозволяє створювати динамічні та інтерактивні інтерфейси користувача, що покращує взаємодію з користувачем та підвищує зручність використання системи.

2.1.5 **Bootstrap 5**

Для створення інтерфейсу користувача було обрано Bootstrap 5 з наступних причин:

– Bootstrap забезпечує швидку та ефективну розробку інтерфейсів завдяки великій кількості готових компонентів та шаблонів;

– фреймворк підтримує адаптивний дизайн, що забезпечує коректне відображення інтерфейсу на різних пристроях та розмірах екрану;

– Bootstrap має велику спільноту розробників та добре задокументовану документацію, що спрощує процес навчання та використання.

2.2 **Аналіз існуючих технологій**

Для розробки мережевої системи обліку товарів було проведено детальний аналіз існуючих технологій, які використовуються в подібних системах. Основні технології включають системи управління базами даних (СУБД), фреймворки для фронтенду та серверні платформи. У кожній категорії було розглянуто кілька популярних технологій з їх перевагами та недоліками.

2.2.1 **Аналіз систем управління базами даних (СУБД)**

2.2.1.1 **MySQL**

MySQL є однією з найпопулярніших систем управління базами даних з відкритим кодом. Вона забезпечує високу продуктивність і масштабованість, що робить її відмінним вибором для багатьох веб-додатків та систем обліку. MySQL підтримує великі обсяги даних і може бути легко інтегрована з різними мовами програмування.

Переваги:

– відкрите програмне забезпечення з великою спільнотою розробників;

– висока продуктивність при обробці великих обсягів даних;

– простота у налаштуванні та використанні;

– широка підтримка з боку інструментів розробки та фреймворків.

Недоліки:

– дещо обмежені можливості порівняно з PostgreSQL, особливо у сфері обробки складних запитів і транзакцій;

– менше можливостей для налаштування та розширення функціональності.

2.2.1.2 **SQLite**

SQLite є легкою системою управління базами даних, яка зазвичай використовується в невеликих проектах або як вбудована база даних. Вона не потребує налаштування сервера і є дуже простою у використанні.

Переваги:

– легкість та простота використання;

– не потребує налаштування сервера;

– дуже добре підходить для прототипів та невеликих проектів.

Недоліки:

– не забезпечує потрібного рівня масштабованості та надійності для більш складних систем;

– обмежені можливості для роботи з великими обсягами даних та одночасними транзакціями.

2.2.2 **Фреймворки для фронтенду**

2.2.2.1 **Angular**

Angular є потужним фреймворком для розробки односторінкових додатків (SPA). Він забезпечує високу продуктивність та пропонує багато вбудованих функцій, що спрощують розробку складних веб-додатків.

Переваги:

– широкий набір функцій для розробки складних SPA;

– підтримка модульності та компонентного підходу;

– активна спільнота та хороша документація.

Недоліки:

– більша крива навчання порівняно з React;

– може бути складним для новачків через свою комплексність.

2.2.2.2 **Vue.js**

Vue.js є легким та швидким фреймворком, який поєднує переваги React та Angular. Він простий у використанні і швидко набирає популярність серед розробників.

Переваги:

– легка крива навчання.

– висока продуктивність та гнучкість.

– можливість поступової інтеграції у проекти.

Недоліки:

– менша спільнота розробників порівняно з Angular та React;

– менше ресурсів та інструментів для розробки великих проектів.

2.2.2.3 **Bootstrap 5**

Bootstrap 5 є популярним фреймворком для розробки адаптивних веб-інтерфейсів. Він надає широкий набір готових компонентів і шаблонів, що дозволяє швидко створювати красиві і функціональні інтерфейси користувача.

Переваги:

– швидка розробка інтерфейсів завдяки готовим компонентам;

– підтримка адаптивного дизайну для різних пристроїв;

– легка інтеграція з JavaScript для створення динамічних елементів.

Недоліки:

– може обмежувати гнучкість дизайну через використання готових шаблонів;

– може бути важким для кастомізації у великих проектах.

2.2.3 **Серверні платформи**

2.2.3.1 **Spring Boot**

Spring Boot є популярним фреймворком для розробки мікросервісів на Java. Він надає готову структуру для швидкої розробки та розгортання додатків.

Переваги:

– підтримка мікросервісної архітектури;

– широкий набір готових модулів та інтеграцій;

– висока продуктивність та гнучкість.

Недоліки:

– складніший у використанні порівняно з Jakarta EE, особливо для новачків;

– потребує більше налаштувань та конфігурацій.

2.2.3.2 **Jakarta EE**

Jakarta EE є стандартом для розробки корпоративних додатків на Java. Він забезпечує стабільність, сумісність та довгострокову підтримку, що робить його ідеальним вибором для розробки надійних і масштабованих серверних додатків.

Переваги:

– стандартність та підтримка;

– висока масштабованість та надійність;

– широкий набір API для роботи з базами даних, веб-сервісами та іншими ресурсами.

Недоліки:

– може бути складним для новачків через свою комплексність;

– потребує налаштування серверів додатків.

2.2.4 **Клієнтські мови програмування**

2.2.4.1 **JavaScript**

JavaScript є основною мовою програмування для розробки динамічних і інтерактивних веб-додатків. Його популярність і широке використання забезпечують доступ до великої кількості бібліотек і інструментів.

Переваги:

– широке використання та підтримка усіма сучасними веб-браузерами;

– велика кількість доступних бібліотек та інструментів;

– легка інтеграція з HTML та CSS.

Недоліки:

– може бути складним для підтримки у великих проектах без використання фреймворків або бібліотек (наприклад, React або Vue.js);

– відсутність типізації, що може призводити до помилок у коді.

2.2.4.2 **TypeScript**

TypeScript є надмножиною JavaScript, яка додає статичну типізацію. Вона дозволяє розробникам писати більш надійний і підтримуваний код.

Переваги:

– статична типізація допомагає виявляти помилки на етапі компіляції;

– краща підтримка для великих проектів;

– підтримує всі можливості JavaScript, оскільки компілюється у чистий JavaScript.

Недоліки:

– потребує додаткового налаштування компіляції;

– може мати більшу криву навчання для розробників, які звикли до чистого JavaScript.

3 МЕТОДИ ТА АЛГОРИТМИ РОЗВ'ЯЗАННЯ ЗАДАЧ РОЗРОБКИ МЕРЕЖЕВОЇ СИСТЕМИ ОБЛІКУ ТОВАРІВ

3.1 **Розробка структури бази даних**

Розробка структури бази даних для системи обліку товарів базується на теорії нормалізації та реляційній моделі даних. Основними принципами є уникнення надлишкових даних та забезпечення цілісності даних через нормалізацію.

3.1.1 **Нормалізація даних**

Нормалізація — це процес організації даних у базі даних для зменшення надлишковості і забезпечення цілісності даних. Основні етапи нормалізації включають приведення структури бази даних до першої нормальної форми (1NF), другої нормальної форми (2NF) та третьої нормальної форми (3NF):

– перша нормальна форма (1NF): Усі атрибути містять атомарні (неділимі) значення. Наприклад, у таблиці Users кожен користувач має окремі поля для імені, електронної пошти та паролю.

– друга нормальна форма (2NF): База даних повинна бути у 1NF, і всі неключові атрибути повинні залежати від первинного ключа.

– третя нормальна форма (3NF): База даних повинна бути у 2NF, і всі неключові атрибути повинні залежати безпосередньо від первинного ключа.

3.1.2 **Реляційна модель даних**

Реляційна модель даних використовує таблиці для представлення даних і відносин між ними. Кожна таблиця складається з рядків (записів) і стовпців (атрибутів). Відносини між таблицями визначаються через первинні та зовнішні ключі.

У нашому проекті MySQL використовувалася для зберігання даних наступних таблиць:

– users (Користувачі): Зберігає інформацію про користувачів;

– clients (Клієнти): Зберігає інформацію про клієнтів;

– products (Товари): Зберігає інформацію про товари;

– categories (Категорії): Зберігає категорії товарів;

– orders (Замовлення): Зберігає інформацію про замовлення;

– order_items (Товари замовлень): Зберігає товари замовлень.

Для забезпечення цілісності даних було використано первинні та зовнішні ключі. Наприклад, у таблиці orders поле fk_order_client_id є зовнішнім ключем, який посилається на первинний ключ у таблиці clients.

3.1.3 **Алгоритми розробки бази даних**

3.1.3.1 **Створення ER-діаграм**

ER-діаграма (Entity-Relationship Diagram) використовується для візуалізації структури бази даних. У нашому проекті ми використовували ER-діаграми для визначення сутностей, їх атрибутів та відносин між ними:

– визначення сутностей: Ми визначили такі сутності як users, clients, products, categories, orders, order_items;

– визначення атрибутів: Кожна сутність має свої атрибути, наприклад, сутність clients має атрибути client_id, client_name, client_email, client_phone;

– визначення відносин: Ми встановили зв'язки між сутностями, наприклад, один користувач може мати кілька замовлень, а одне замовлення може містити кілька товарів.

3.1.3.2 **Розробка SQL-запитів для взаємодії з базою даних**

Для взаємодії з базою даних ми використовували SQL (Structured Query Language). Конкретні приклади SQL-запитів для нашої системи включають:

– запити на створення (DDL - Data Definition Language):

```
create table categories
(
    category_id   int auto_increment
        primary key,
    category_name varchar(256) default 'default' null
);

create table clients
(
    client_id    int auto_increment
        primary key,
    client_name  varchar(256) default 'default' null,
    client_email varchar(256) default 'default' null,
    client_phone varchar(256) default '0'       null
);
```

```
create table order_items
(
    order_items_id            int auto_increment
        primary key,
    fk_order_items_order_id   int           null,
    fk_order_items_product_id int           null,
    order_items_quantity      int default 0 null,
    order_items_price         int default 0 null,
    constraint order_items_orders_order_id_fk
        foreign key (fk_order_items_order_id) references orders (order_id)
            on delete cascade,
    constraint order_items_products_product_id_fk
        foreign key (fk_order_items_product_id) references products
(product_id)
            on delete cascade
);

create table orders
(
    order_id           int auto_increment
        primary key,
    order_date         date         default '1970-01-01' null,
    fk_order_client_id int                               null,
    order_status       varchar(256) default 'default'    null,
    constraint orders_clients_client_id_fk
        foreign key (fk_order_client_id) references clients (client_id)
            on delete cascade
);

create table products
(
    product_id           int auto_increment
        primary key,
    product_name         varchar(256) default 'default'     null,
    product_desc         varchar(256) default 'default'     null,
    product_price        double       default 0             null,
    product_quantity     int          default 0             null,
    product_manufacturer varchar(256) default 'default'     null,
    product_img          varchar(256) default 'default.png' null,
    fk_category_id       int          default 1             null,
    constraint products_fk
        foreign key (fk_category_id) references categories (category_id)
            on update cascade on delete set null
);

create table users
(
    user_id       int auto_increment
        primary key,
    user_name     varchar(256) default 'default'     not null,
    user_surname  varchar(256) default 'default'     not null,
    user_email    varchar(256) default 'default'     not null,
    user_password varchar(256) default 'default'     not null,
    user_role     varchar(256) default 'user'        not null,
    user_login    varchar(256) default 'default'     not null,
    user_avatar   varchar(256) default 'default.png' null,
    constraint user_email
        unique (user_email),
    constraint user_login
        unique (user_login)
);
```

– запити на маніпуляцію даними (DML - Data Manipulation Language):

```
INSERT INTO products () values ();

INSERT INTO orders (order_date, fk_order_client_id) VALUES (CURDATE(), 1);
```

– запити на вибірку даних (DQL - Data Query Language):

```
SELECT COUNT(*) as row_count FROM orders;

SELECT SUM(product_price * product_quantity) AS row_count FROM products;
```

3.2 **Розробка серверної частини**

Серверна частина мережевої системи обліку товарів базується на архітектурі клієнт-сервер, де сервер обробляє запити клієнтів, виконує бізнес-логіку і взаємодіє з базою даних. Основними компонентами є веб-сервер, сервер додатків та база даних. У цьому контексті веб-сервер відповідає за отримання та обробку HTTP-запитів, сервер додатків виконує бізнес-логіку, а база даних зберігає необхідну інформацію.

3.2.1 **Архітектура клієнт-сервер**

– клієнт: Клієнтська частина додатку, написана з використанням HTML, CSS, JavaScript та Bootstrap, надсилає HTTP-запити до сервера через браузер;

– сервер: Отримує запити від клієнтів, обробляє їх та відправляє відповіді назад клієнту. Сервер складається з трьох основних частин:

– веб-сервер: Використовується Apache Tomcat як контейнер сервлетів, який відповідає за отримання та обробку HTTP-запитів;

– сервер додатків: Використовується Jakarta EE для реалізації бізнес-логіки. Це включає створення та налаштування сервлетів, які обробляють запити клієнтів;

– база даних: MySQL використовується для зберігання даних про товари, користувачів та замовлення. MySQL налаштований для забезпечення цілісності та доступності даних;

3.2.2 **API на основі сервлетів та JSP (JavaServer Pages)**

Замість використання RESTful API, в даному проекті застосовуються Java сервлети та JSP. Сервлети - це Java-класи, що відповідають за обробку HTTP-

запитів, а JSP дозволяє створювати динамічні веб-сторінки, що змішують HTML та Java-код.

Сервлети:

– створюються Java-класи, що реалізують інтерфейс HttpServlet. Наприклад, сервлет для обробки реєстрації користувачів реалізує методи doPost() для обробки даних форми реєстрації;

– приклад коду:

```
        public class RegisterServlet extends HttpServlet {
        protected void doPost(HttpServletRequest request,
   HttpServletResponseresponse) throws ServletException, IOException {
                String username = request.getParameter("username");
                String password = request.getParameter("password");
                // Логіка збереження даних у базу даних
                UserDAO userDAO = new UserDAO();
                userDAO.registerUser(username, password);
                response.sendRedirect("register-success.jsp");
            }
        }
```

JSP:

– використовуються для створення динамічних веб-сторінок. JSP-файли містять HTML-код з вставками Java-коду, який генерує динамічний вміст на основі даних, отриманих з сервера;

– приклад коду:

```
<%@ page contentType="text/html;charset=UTF-8" %>
<html>
<head>
<title>Реєстрація користувача</title>
</head>
<body>
<form action="register" method="post">
Ім'я користувача: <input type="text" name="username"><br>
Пароль: <input type="password" name="password"><br>
<input type="submit" value="Зареєструватися">
</form>
</body>
</html>
```

3.2.3 **Алгоритми**

3.2.3.1 **Проектування серверної частини**

Створення структури проекту:

– визначення необхідних сервлетів та JSP-сторінок;

– визначення пакетів для організації коду (наприклад, com.myapp.servlets, com.myapp.models, com.myapp.utils).

Розробка бізнес-логіки

– створення Java-класів, що реалізують бізнес-логіку (наприклад, класів для управління товарами, категоріями, замовленнями).

3.2.3.2 **Реалізація сервлетів**

Сервлети обробляють HTTP-запити, виконують необхідну логіку і повертають відповіді. Основні методи сервлетів включають:

– doGet(HttpServletRequest request, HttpServletResponse response): Обробляє GET-запити;

– doPost(HttpServletRequest request, HttpServletResponse response): Обробляє POST-запити.

3.2.3.3 **Управління сесіями та аутентифікація користувачів**

Для забезпечення безпеки та управління сесіями користувачів використовуються HTTP-сесії.

– створення сесії: Під час входу користувача створюється нова сесія:

```
HttpSession session = request.getSession();
session.setAttribute("user", user);
```

– перевірка сесії: Під час кожного запиту перевіряється, чи є активна сесія користувача:

```
HttpSession session = request.getSession(false);
if (session == null || session.getAttribute("user") == null) {
    response.sendRedirect("login.jsp");
}
```

– завершення сесії: Під час виходу користувача сесія завершується:

```
HttpSession session = request.getSession(false);
if (session != null) {
    session.invalidate();
}
response.sendRedirect("login.jsp");
```

3.3 **Розробка клієнтської частини**

3.3.1 **Алгоритми**

Клієнтська частина системи обліку товарів забезпечує взаємодію з користувачем, відправку запитів до серверної частини, отримання та відображення відповідей. Вона реалізована з використанням HTML, CSS,

JavaScript та Bootstrap 5. Конкретне використання цих технологій у проекті описано нижче.

3.3.1.1 **Створення інтерактивних елементів користувацького інтерфейсу**

Створення HTML-структури:

– використання Bootstrap 5 для створення адаптивного макета сторінки з необхідними компонентами (форми, кнопки, таблиці тощо);

– розміщення елементів і структурування веб-сторінки за допомогою HTML-тегів.

Додавання інтерактивності за допомогою JavaScript:

– використання JavaScript для додавання динамічних елементів та обробки подій;

– реалізація функціоналу взаємодії з елементами сторінки через обробку подій, таких як click, hover тощо;

– забезпечення адаптивності інтерфейсу для різних пристроїв та розмірів екранів.

3.3.1.2 **Використання AJAX для асинхронного обміну даними з сервером**

Відправка AJAX-запиту на сервер:

– використання XMLHttpRequest або fetch API для відправки запитів на сервер та отримання відповідей;

– забезпечення адаптивності інтерфейсу для різних пристроїв та розмірів екранів.

Опрацювання помилок і винятків:

– обробка помилок, які можуть виникнути під час взаємодії з сервером, таких як втрата з'єднання або помилкові відповіді сервера;

– відображення коректних повідомлень про помилки або відновлення з'єднання в разі необхідності.

3.3.1.3 **Валідація форм на стороні клієнта**

Додавання обробника подій для форми:

– обробник події submit прив'язується до форми, щоб перехопити її відправку;

– встановлення обробників подій для валідації введених даних під час взаємодії з окремими полями форми.

Реалізація валідації форм за допомогою JavaScript:

– при натисканні кнопки відправки форми, скрипт перевіряє кожне поле на наявність значення і коректність введених даних;

– встановлення обробників подій для валідації введених даних під час взаємодії з окремими полями форми;

– відображення відповідних повідомлень про помилки поруч із відповідними полями форми.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ МЕРЕЖЕВОЇ СИСТЕМИ ОБЛІКУ ТОВАРІВ

4.1 **Архітектурні рішення**

Архітектура мережевої системи обліку товарів складається з трьох основних компонентів: клієнтської частини, серверної частини та бази даних. Кожен з цих компонентів відіграє важливу роль у забезпеченні функціональності, продуктивності та надійності системи.

4.1.1 **Клієнтська частина**

Клієнтська частина відповідає за взаємодію з користувачем та відображення даних. Вона реалізована за допомогою наступних технологій:

HTML/CSS:

– HTML (HyperText Markup Language): Використовується для створення структури веб-сторінок. В HTML визначаються елементи сторінки, такі як заголовки, абзаци, форми, таблиці та інші елементи інтерфейсу користувача;

– CSS (Cascading Style Sheets): Використовується для стилізації веб-сторінок. CSS дозволяє визначати вигляд елементів HTML, таких як кольори, шрифти, розміри та розташування. Це забезпечує гарний і професійний вигляд веб-додатка.

JavaScript:

– JavaScript - це мова програмування, яка використовується для додавання динамічної функціональності до веб-сторінок. Вона дозволяє створювати інтерактивні елементи, обробляти події (наприклад, кліки на кнопки), маніпулювати DOM (Document Object Model) та виконувати асинхронні запити до серверної частини через AJAX;

– JavaScript також використовується для реалізації валідації форм на стороні клієнта, що покращує користувацький досвід і зменшує навантаження на сервер.

Bootstrap 5:

– Bootstrap 5 - це популярний фреймворк для розробки адаптивних веб-інтерфейсів. Він містить готові компоненти (такі як навігаційні панелі, кнопки, форми, модальні вікна тощо) та стилі, які значно спрощують і прискорюють процес розробки;

– Bootstrap забезпечує адаптивність веб-додатка, що дозволяє йому коректно відображатися на різних пристроях, включаючи комп'ютери, планшети та смартфони.

4.1.2 **Серверна частина**

Серверна частина відповідає за обробку запитів від клієнта, виконання бізнес-логіки та взаємодію з базою даних. Вона реалізована за допомогою наступних технологій:

Jakarta EE:

– Jakarta EE (раніше відома як Java EE) - це набір специфікацій для корпоративних додатків, які працюють на Java. Вона забезпечує стандарти для розробки масштабованих, надійних і безпечних серверних додатків;

– використання Jakarta EE дозволяє розробникам створювати сервіси, які можуть обробляти запити від клієнтів, виконувати бізнес-логіку (наприклад, розрахунок цін, обробку замовлень) і взаємодіяти з базою даних;

– основні компоненти Jakarta EE включають сервлети, EJB (Enterprise JavaBeans), JPA (Java Persistence API) для роботи з базами даних, і JAX-RS для створення RESTful веб-сервісів.

4.1.3 **База даних**

База даних відповідає за зберігання даних, забезпечення їх цілісності та доступності. Для цієї системи використовується MySQL:

MySQL:

– MySQL - це популярна реляційна система управління базами даних (СУБД) з відкритим вихідним кодом. Вона забезпечує зберігання структурованих даних і підтримує мову SQL (Structured Query Language) для виконання запитів;

– MySQL забезпечує ACID-транзакції (Atomicity, Consistency, Isolation, Durability), що гарантує надійність і цілісність даних.

– MySQL також підтримує механізми реплікації і кластеризації, що дозволяє підвищити продуктивність і забезпечити високу доступність системи.

4.1.4 **Архітектурна діаграма**

Для наочного уявлення архітектури мережевої системи обліку товарів можна використати діаграму компонентів (Component Diagram) у нотації UML. Діаграма компонентів зображено на рисунку 4.1.

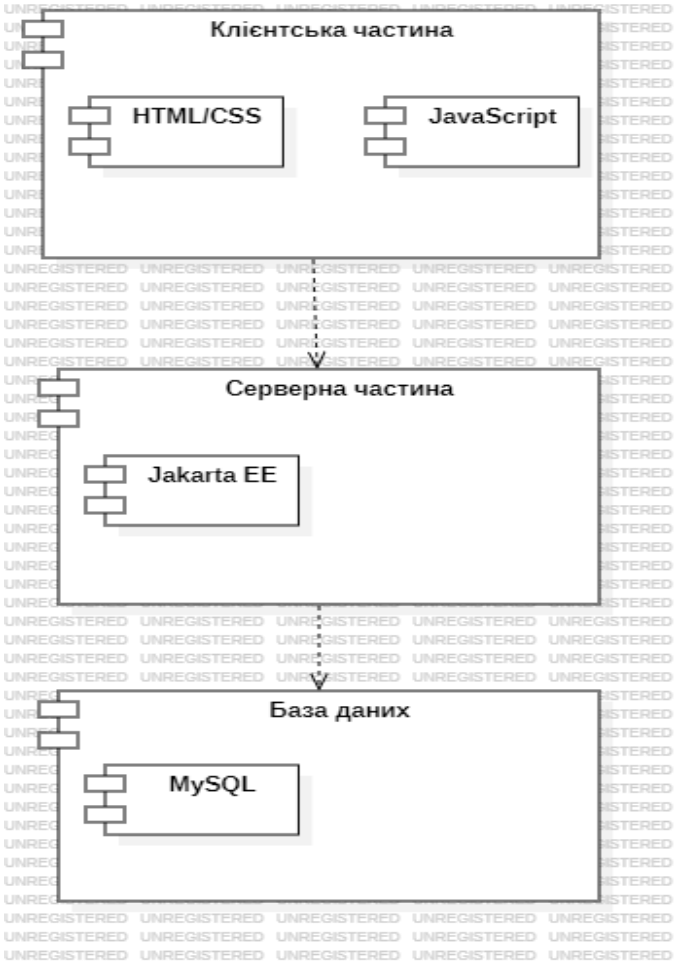


Рисунок 4.1 – Діаграма компонентів

4.2 **Опис алгоритмів програми**

В цьому розділі надається детальний опис алгоритмів, які використовуються в програмі для вирішення конкретних завдань. Кожен алгоритм буде розглянутий окремо, з вказівкою на його функціональне

призначення, послідовність кроків та взаємодію з іншими компонентами системи. Для візуалізації алгоритмів будемо використовувати діаграму послідовності. Діаграма послідовності (Sequence Diagram) є одним із видів діаграм у UML, який використовується для моделювання поведінки системи, зокрема для відображення порядку взаємодії між різними об'єктами або компонентами у процесі виконання певної функції або сценарію.

Основні елементи діаграми послідовності включають:

Актори (Actors):

– відображають зовнішні сутності, що взаємодіють із системою (користувачі, інші системи тощо);

– представлені як фігури людини або стандартні UML-ікони на лівому краю діаграми.

Об'єкти (Objects):

– відображають учасників, які взаємодіють у процесі сценарію;

– представлені прямокутниками з підкресленими іменами, які розташовані вздовж верхнього краю діаграми.

Лінії життя (Lifelines):

– вертикальні пунктирні лінії, що виходять від об'єктів, які показують існування об'єктів протягом часу;

– лінія життя починається від об'єкта і продовжується вниз, позначаючи період часу, протягом якого об'єкт існує і може взаємодіяти.

Повідомлення (Messages):

– горизонтальні стрілки між лініями життя, що відображають обмін повідомленнями (виклики методів, відповіді тощо) між об'єктами;

– містять ім'я повідомлення та, за потреби, аргументи.

Типи повідомлень:

– синхронні повідомлення: Стрілка з заповненою голівкою, яка вказує на виклик методу, що очікує на відповідь;

– асинхронні повідомлення: Стрілка з відкритою голівкою, що вказує на виклик методу, який не очікує негайної відповіді;

– ретурн-повідомлення (Return messages): Пунктирна стрілка з відкритою голівкою, що вказує на повернення значення з методу.

– активаційні бари (Activation Bars):

– товсті вертикальні прямокутники на лініях життя, що відображають періоди, коли об'єкти активно виконують операції або методи;

– вказують на тривалість виконання методу або операції.

4.2.1 **Алгоритм додавання товару**

Функціональні призначення додавання товару:

– надання можливості адміністраторам або авторизованим користувачам додавати нові товари до системи;

– збереження даних про новий товар у базі даних.

Актори та об'єкти:

– користувач (User);

– клієнтська частина (Client);

– серверна частина (Server).

– база даних (Database)

Детальний опис алгоритму:

Користувач натискає кнопку "Add product":

– користувач натискає кнопку "Додати товар", що викликає JavaScript функцію для обробки події.

Клієнтська частина надсилає запит на сервер:

– JavaScript надсилає AJAX запит на сервер, щоб створити новий товар;

– запит може бути реалізований за допомогою XMLHttpRequest або fetch API, причому запит не містить даних про товар.

Серверна частина обробляє запит:

– сервер, побудований на основі Jakarta EE, приймає запит, обробляє його і створює новий запис з дефолтними значеннями (наприклад, назва товару - "Новий товар", кількість - 0, ціна - 0);

– сервер взаємодіє з базою даних MySQL для збереження інформації про новий товар.

База даних зберігає новий товар:

– MySQL база даних отримує запит на додавання нового запису з дефолтними значеннями, виконує його та повертає результат операції.

Серверна частина надсилає відповідь клієнту:

– після успішного збереження даних сервер надсилає відповідь клієнтській частині, вказуючи, що товар успішно додано.

Клієнтська частина оновлює інтерфейс:

– JavaScript обробляє відповідь сервера, оновлюючи інтерфейс користувача для відображення нового товару.

Діаграма послідовності для цього алгоритму наведена на рисунку 4.2.

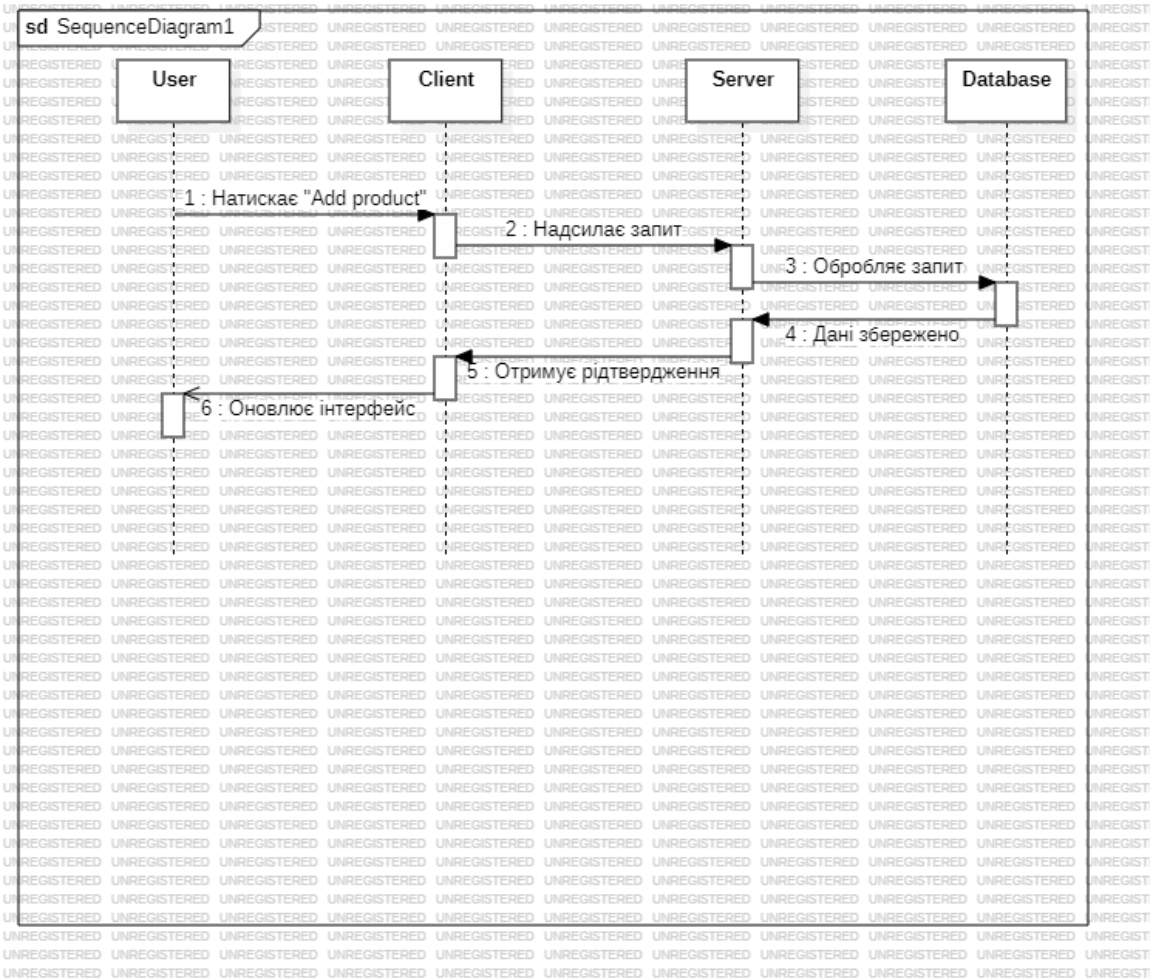


Рисунок 4.2 – Діаграма послідовності додавання товарів

4.2.2 **Алгоритм редагування товару**

Функціональні призначення редагування товару:

– надання можливості коригувати існуючі дані про товар (назва, опис, ціна тощо);

– збереження змін у базі даних після редагування товару;

– перевірка введених даних на коректність та валідація перед збереженням змін.

Актори та об'єкти:

– користувач (User);

– клієнтська частина (Client);

– керверна частина (Server).

Детальний опис алгоритму:

Користувач натискає кнопку "Edit" на карточці товару:

– користувач натискає кнопку "Edit", що викликає JavaScript функцію для обробки події.

Клієнтська частина відображає форму редагування:

– JavaScript формує та відображає форму з поточними значеннями товару, які можуть бути редаговані.

Користувач змінює необхідні поля та натискає кнопку "Зберегти":

– користувач вводить нові значення для товару у форму і натискає кнопку "Save".

Клієнтська частина надсилає запит на сервер:

– JavaScript надсилає AJAX запит на сервер з новими значеннями товару;

– запит може бути реалізований за допомогою XMLHttpRequest або fetch API.

Серверна частина обробляє запит:

– сервер, побудований на основі Jakarta EE, приймає запит, обробляє його та оновлює відповідний запис товару у базі даних MySQL.

База даних зберігає оновлені дані товару:

– MySQL база даних отримує запит на оновлення запису, виконує його та повертає результат операції.

Серверна частина надсилає відповідь клієнту:

– після успішного оновлення даних сервер надсилає відповідь клієнтській частині, вказуючи, що товар успішно оновлено.

Клієнтська частина оновлює інтерфейс:

– JavaScript обробляє відповідь сервера, оновлюючи інтерфейс користувача для відображення оновлених даних товару.

Діаграма послідовності для цього алгоритму наведена на рисунку 4.3.

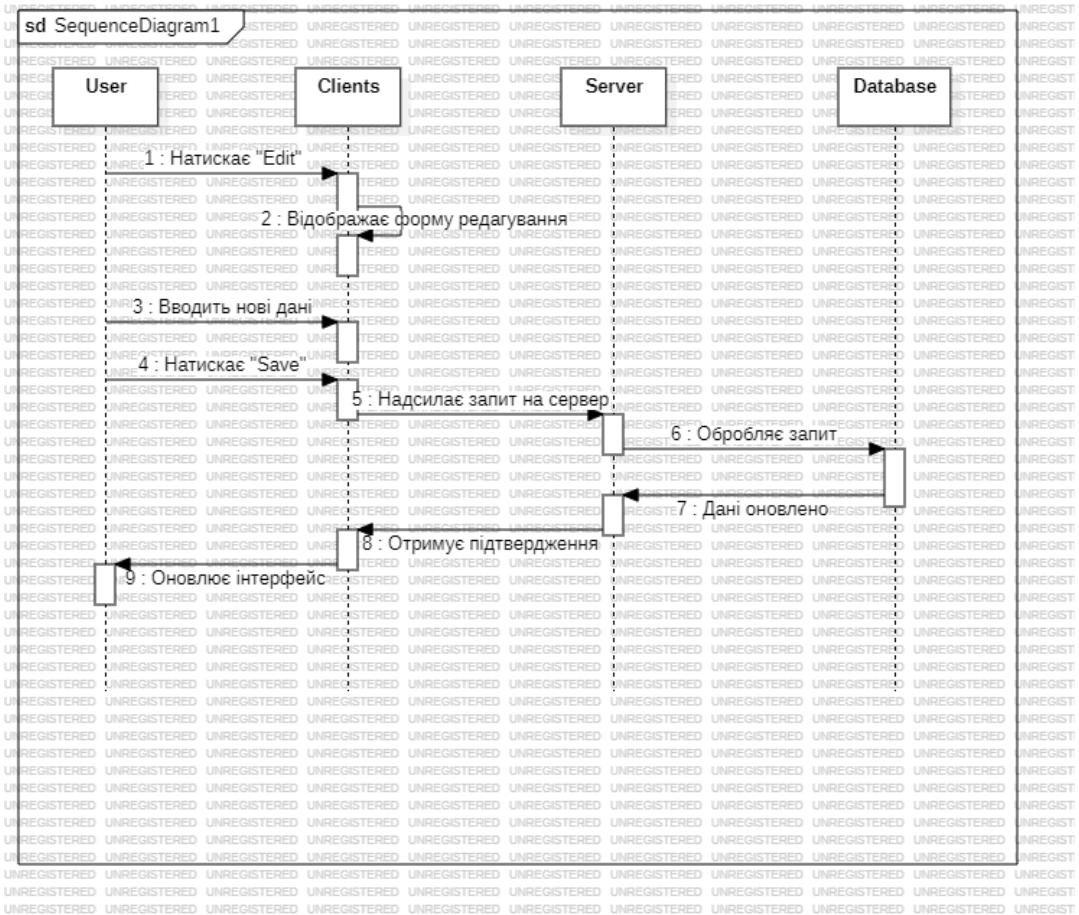


Рисунок 4.3 – Діаграма послідовності редагування товарів

4.2.3 **Алгоритм реєстрації**

Функціональні призначення реєстрації:

– забезпечення можливості користувачам створювати облікові записи у системі;

– збереження основної інформації про користувача (ім'я, email, пароль) у базі даних;

– перевірка унікальності інформації та забезпечення цілісності даних під час реєстрації.

Актори та об'єкти:

– користувач (User);

– клієнтська частина (Client);

– серверна частина (Server).

Детальний опис алгоритму:

Користувач натискає кнопку "Register":

– користувач натискає кнопку "Register", що викликає JavaScript функцію для відкриття форми реєстрації.

Клієнтська частина відкриває форму реєстрації:

– JavaScript відкриває форму реєстрації, яка дозволяє користувачеві ввести необхідні дані (ім'я, email, пароль тощо).

Користувач заповнює форму реєстрації та натискає кнопку "Submit":

– користувач заповнює форму та натискає кнопку "Submit", що викликає JavaScript функцію для обробки події.

Клієнтська частина надсилає запит на сервер з даними користувача:

– JavaScript надсилає AJAX запит на сервер з даними користувача;

– запит може бути реалізований за допомогою XMLHttpRequest або fetch API.

Серверна частина обробляє запит:

– сервер, побудований на основі Jakarta EE, приймає запит, обробляє його і створює новий запис у базі даних відповідно до отриманих даних.

База даних зберігає новий запис користувача:

– MySQL база даних отримує запит на створення нового запису, виконує його та повертає результат операції.

Серверна частина надсилає відповідь клієнту:

– після успішного збереження даних сервер надсилає відповідь клієнтській частині, вказуючи, що користувач успішно зареєстрований.

Клієнтська частина оновлює інтерфейс:

– JavaScript обробляє відповідь сервера, оновлюючи інтерфейс користувача для відображення повідомлення про успішну реєстрацію.

Діаграма послідовності для цього алгоритму наведена на рисунку 4.4.

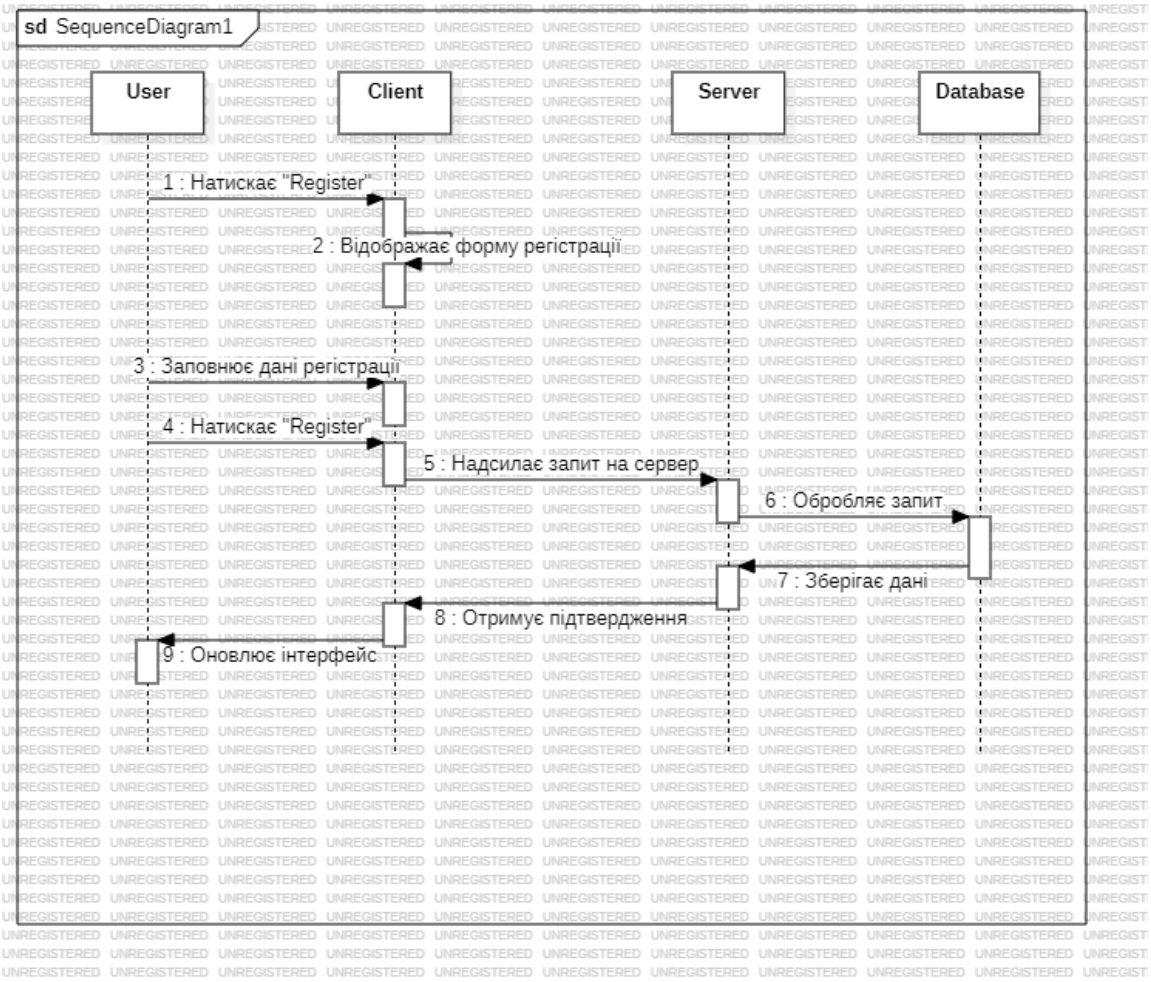


Рисунок 4.4 – Діаграма послідовності реєстрації користувача

4.2.4 **Алгоритм логіну**

Функціональні призначення логіну:

– забезпечення можливості користувачам входити до системи під своїми обліковими записами;

– перевірка автентичності введених даних (email та пароль);

– надання доступу до захищених ресурсів системи після успішної автентифікації.

Актори та об'єкти:

– користувач (User);

– клієнтська частина (Client);

– серверна частина (Server).

Детальний опис алгоритму:

Користувач відкриває форму входу:

– користувач переходить на сторінку входу, де відображається форма входу.

Користувач заповнює форму входу:

– користувач вводить свою електронну пошту та пароль у відповідні поля форми.

Користувач натискає кнопку "Login":

– користувач натискає кнопку "Login", що викликає JavaScript функцію для обробки події.

Клієнтська частина надсилає запит на сервер з даними для входу:

– JavaScript надсилає AJAX запит на сервер з даними форми входу;

– запит може бути реалізований за допомогою XMLHttpRequest або fetch API.

Серверна частина обробляє запит:

– сервер, побудований на основі Jakarta EE, приймає запит, обробляє його і перевіряє дані користувача в базі даних.

База даних перевіряє наявність користувача та відповідність пароля:

– MySQL база даних отримує запит на перевірку даних користувача, виконує його та повертає результат операції.

Серверна частина надсилає відповідь клієнту:

– після перевірки даних сервер надсилає відповідь клієнтській частині, вказуючи, що вхід успішний або невдалий.

Клієнтська частина оновлює інтерфейс:

– JavaScript обробляє відповідь сервера, оновлюючи інтерфейс користувача для відображення повідомлення про успішний або невдалий вхід.

Діаграма послідовності для цього алгоритму наведена на рисунку 4.5.

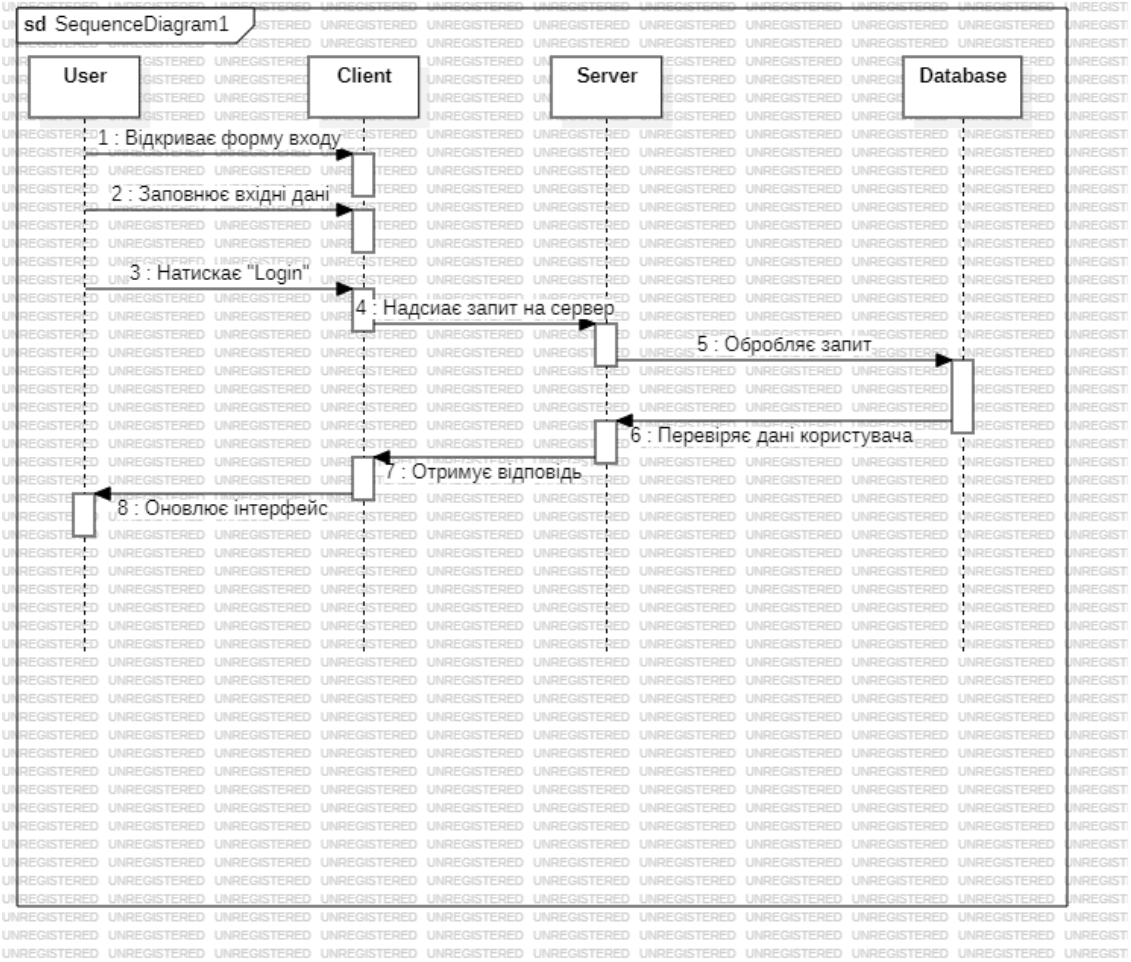


Рисунок 4.5 – Діаграма послідовності логіну користувача

4.2.5 **Алгоритм створення замовлення**

Функціональні призначення створення замовлення:

– дозвіл клієнту вибирати товари та кількість для замовлення;

– збереження інформації про замовлення у базі даних;

– перевірка доступності товарів на складі та розрахунок загальної вартості замовлення.

Актори та об'єкти:

– користувач (User);

– клієнтська частина (Client);

– серверна частина (Server).

Детальний опис алгоритму:

Користувач відкриває форму створення замовлення:

– користувач переходить на сторінку створення замовлення, де відображається форма для вибору товару та клієнта.

Користувач вибирає товар зі списку:

– користувач обирає потрібний товар зі списку доступних товарів.

Користувач вибирає існуючого клієнта або додає нового клієнта:

– користувач може вибрати існуючого клієнта зі списку або додати нового клієнта, заповнюючи відповідну форму.

Користувач натискає кнопку "Create Order":

– користувач натискає кнопку "Create Order", що викликає JavaScript функцію для обробки події.

Клієнтська частина надсилає запит на сервер з даними замовлення:

– JavaScript надсилає AJAX запит на сервер з даними форми замовлення;

– запит може бути реалізований за допомогою XMLHttpRequest або fetch API.

Серверна частина обробляє запит:

– сервер, побудований на основі Jakarta EE, приймає запит, обробляє його і створює новий запис замовлення в базі даних.

База даних зберігає дані нового замовлення:

– MySQL база даних отримує запит на створення нового замовлення, виконує його та повертає результат операції.

Серверна частина надсилає відповідь клієнту:

– після створення нового замовлення сервер надсилає відповідь клієнтській частині, вказуючи на успішне створення замовлення.

Клієнтська частина оновлює інтерфейс:

– JavaScript обробляє відповідь сервера, оновлюючи інтерфейс користувача для відображення повідомлення про успішне створення замовлення.

Діаграма послідовності для цього алгоритму наведена на рисунку 4.6.

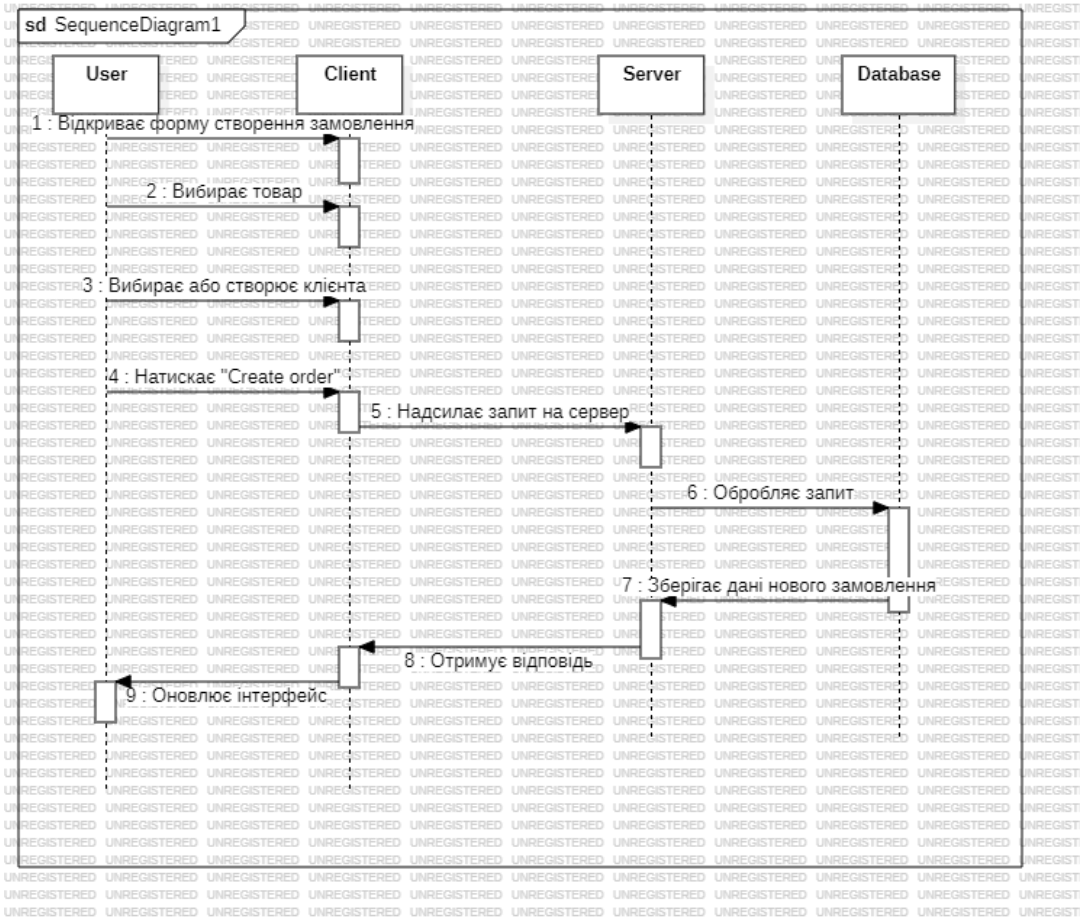


Рисунок 4.6 – Діаграма послідовності створення замовлення

4.3 **Діаграми структур даних**

4.3.1 **ER-діаграма**

ER-діаграма використовується для моделювання структури бази даних і показує зв'язки між різними сутностями (entity). Вона допомагає зрозуміти, як дані пов'язані один з одним і як вони зберігаються в базі даних.

Основні компоненти ER-діаграми:

– сутність (Entity): Це об'єкт або поняття, про яке зберігається інформація в базі даних. Наприклад, Користувач, Товар, Замовлення;

– атрибути (Attributes): Це характеристики або властивості сутності. Наприклад, атрибути сутності Користувач можуть включати ID користувача, Ім'я, Email, Пароль;

– зв'язки (Relationships): Це відносини між сутностями. Вони показують, як одна сутність пов'язана з іншою.

ER-діаграма зображена на рисунку 4.7.

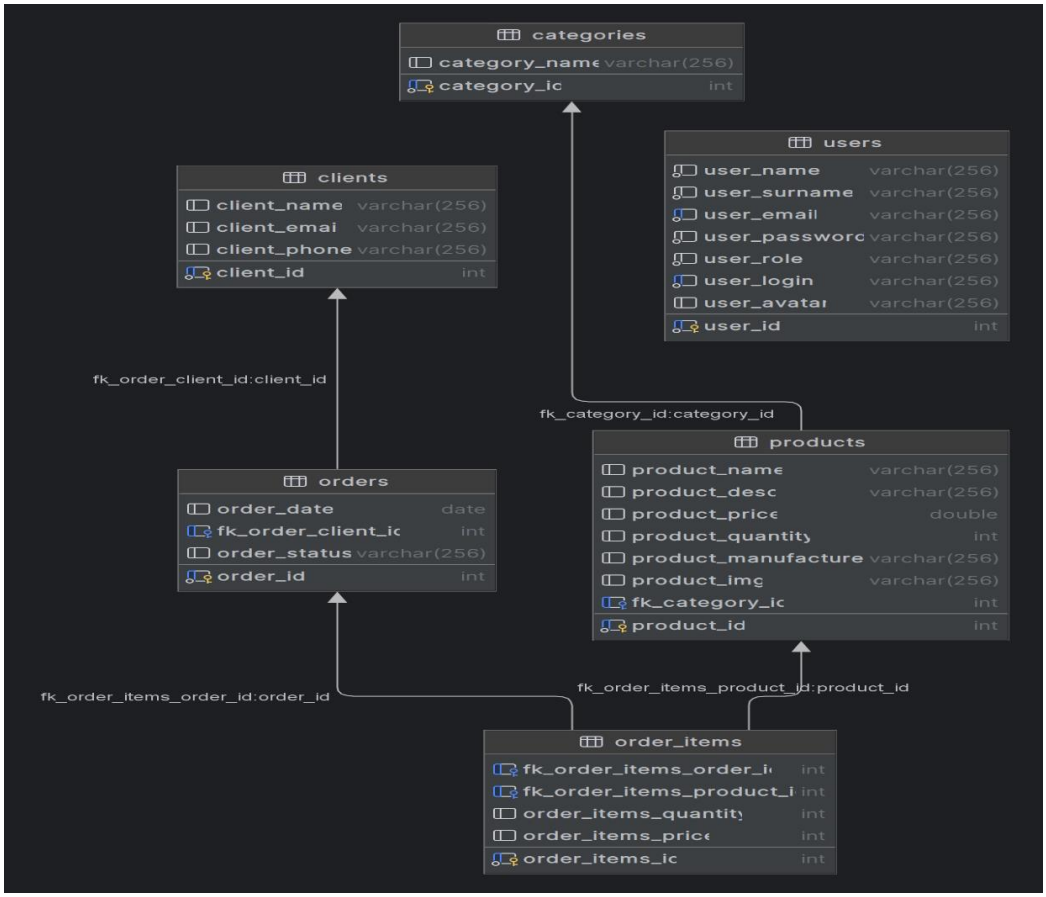


Рисунок 4.7 – ER-діаграма бази даних

4.3.2 **Діаграма розподілення**

Діаграма розподілення (Deployment Diagram) показує фізичну архітектуру системи, розташування компонентів та їх зв'язки між фізичними вузлами (сервери, клієнти, мережеві пристрої). Ця діаграма корисна для розуміння, як програмне забезпечення працює в реальному середовищі і як різні компоненти системи взаємодіють один з одним.

Основні елементи діаграми розподілення:

Вузли (Nodes):

– фізичні або віртуальні машини, де виконуються компоненти системи;

– відображаються у вигляді тривимірних коробок.

Компоненти (Components):

– програмні елементи, що виконуються на вузлах;

– відображаються у вигляді прямокутників всередині вузлів.

Зв'язки (Connections):

– лінії між вузлами, що вказують на комунікацію між ними;

– можуть бути мітками для уточнення типу зв'язку (наприклад, HTTP, SQL).

Діаграма розподілення зображена на рисунку 4.8.

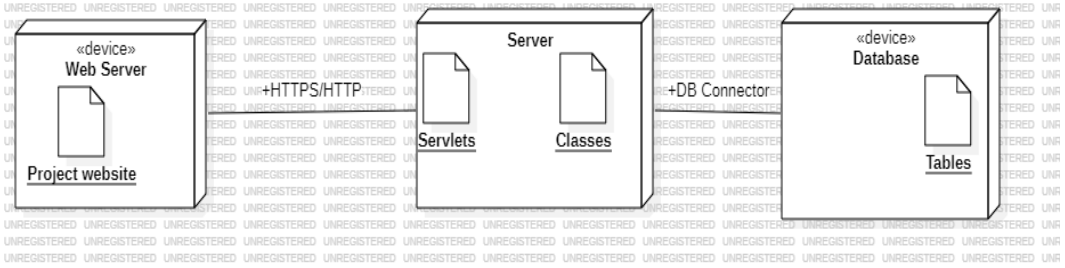


Рисунок 4.8 – Діаграма розподілення

5 ТЕСТУВАННЯ МЕРЕЖОВОЇ СИСТЕМИ ОБЛІКУ ТОВАРІВ

Тестування програмного забезпечення є важливим етапом розробки, який дозволяє виявити помилки та перевірити коректність функціонування програми. Для нашої мережевої системи обліку товарів було проведено тестування з метою перевірки логіки роботи, обробки виключних ситуацій та забезпечення стабільної роботи на різних пристроях і в різних умовах. Нижче наведено кілька тестових сценаріїв.

5.1 **Тестові сценарії**

5.1.1 **Вхід користувача в систему**

Вхідні дані:

– логін: "user";

– пароль: "password".

Очікуваний результат:

– успішний вхід до системи;

– відображення головної сторінки користувача.

Реальний результат:

– успішний вхід;

– відображена головна сторінка.

Результат роботи програми зображено на рисунку 5.1.

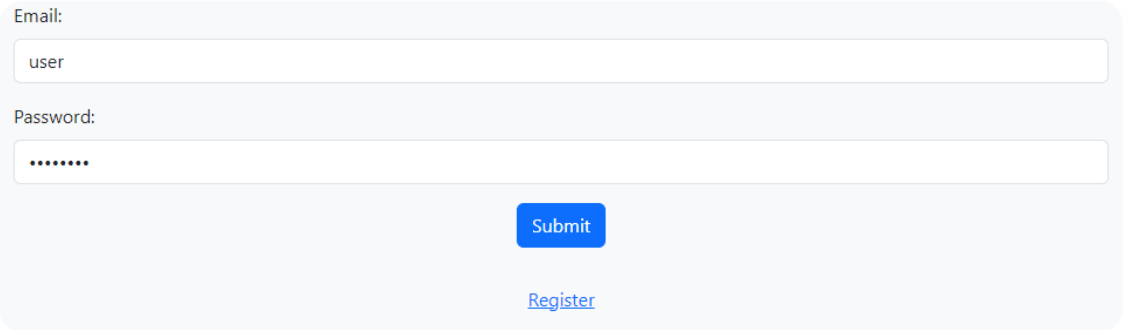


Рисунок 5.1 – Логін до аккаунту

5.1.2 **Вхід з неправильним паролем**

Вхідні дані:

– логін: "user";

– пароль: "wrongpassword".

Очікуваний результат:

– відображення повідомлення про неправильний пароль;

– вхід до системи не здійснено.

Реальний результат:

– відображено повідомлення про помилку;

– вхід не здійснено.

Результат роботи програми зображено рисунку 5.2.

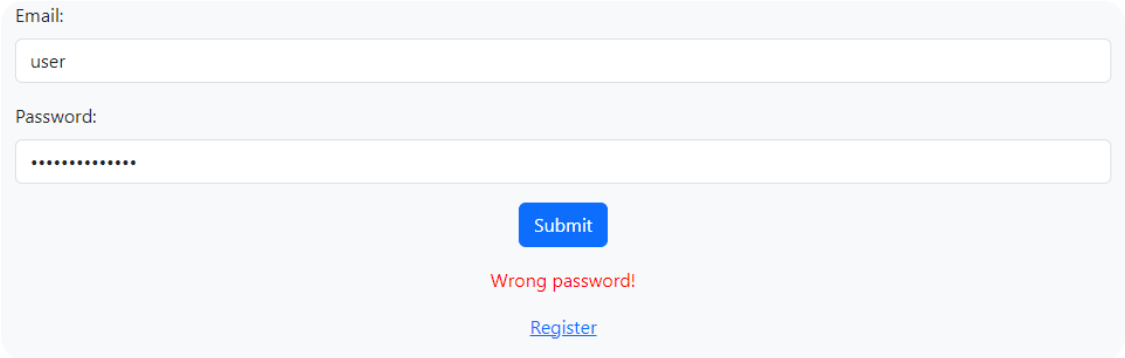


Рисунок 5.2 – Логін з неправильним паролем

5.1.3 **Реєсрація нового користувача**

Вхідні дані:

– ім’я: "Богдан";

– прізвище: "Молчанов";

– логін: "user";

– пароль: "password";

– пошта: "test@gmail.com".

Очікуваний результат:

– реєстрація нового користувача;

– перехід на сторінку логіну.

Реальний результат:

– новий користувач зареєстрований;

– перехід на сторінку логіну здійснено.

– вхід не здійснено.

Результат роботи програми зображено рисунку 5.3.

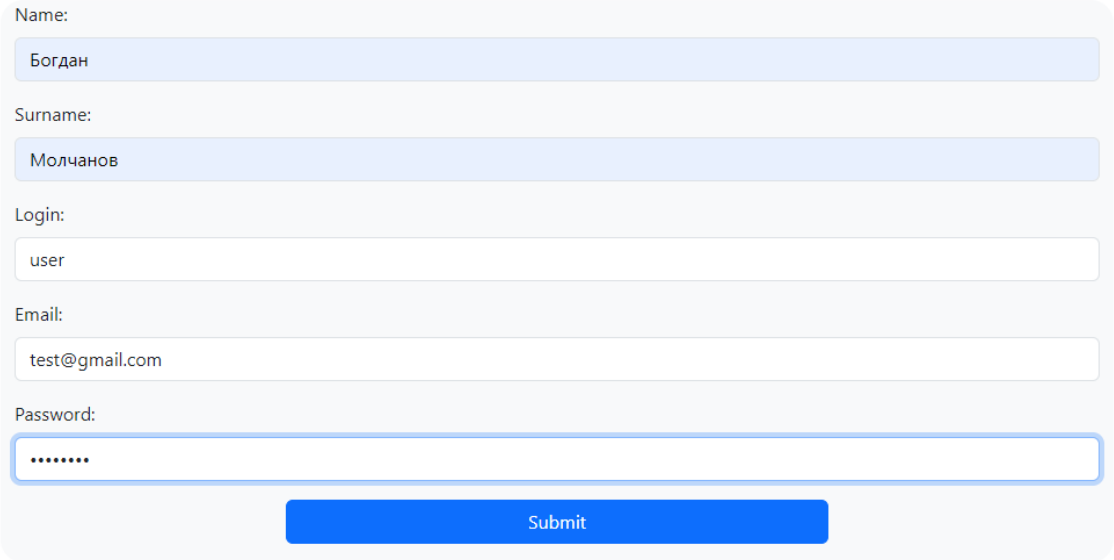


Рисунок 5.3 – Реєстрація користувача

5.1.4 **Реєсрація нового користувача з використаним логіном чи поштою**

Вхідні дані:

– ім’я: "Богдан";

– прізвище: "Молчанов";

– логін: "user";

– пароль: "password";

– пошта: "test@gmail.com".

Очікуваний результат:

– реєстрація нового користувача не здійснена;

– відображено повідомлення про помилку.

Реальний результат:

– новий користувач не зареєстрований;

– повідомлення відображено.

Результат роботи програми зображено рисунку 5.4.

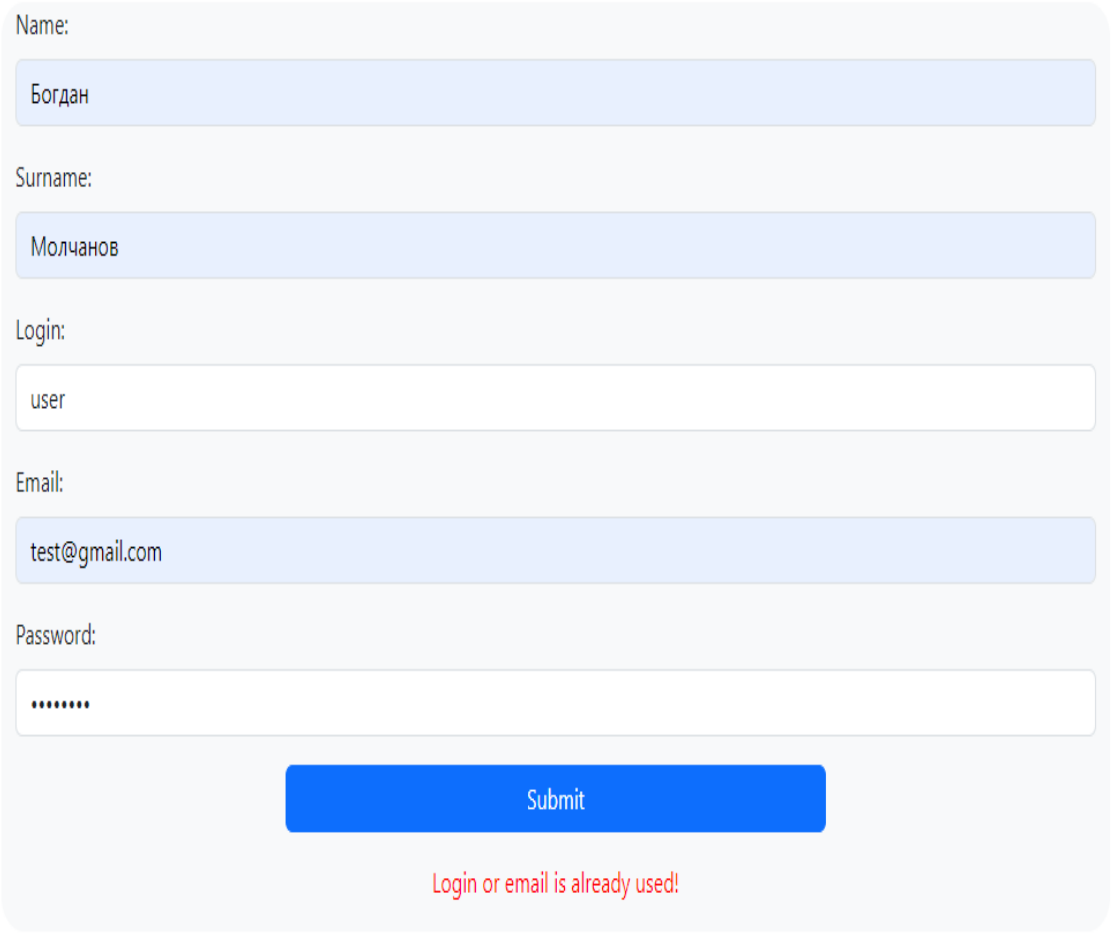


Рисунок 5.4 – Реєстрація з використаним логіном і поштою

5.1.5 **Додавання товару**

Вхідні дані:

– додавання товару здійснюється через натискання кнопки "Add product" без заповнення полів форми (всі поля заповнені за замовчуванням);

Очікуваний результат:

– товар успішно додається до бази даних;

– товар відображається у списку товарів.

Реальний результат:

– товар додано до бази даних успішно;

– товар відображається успішно.

Результат роботи програми зображено рисунку 5.5 та рисунку 5.6.

Рисунок 5.5 - Кнопка додавання товару

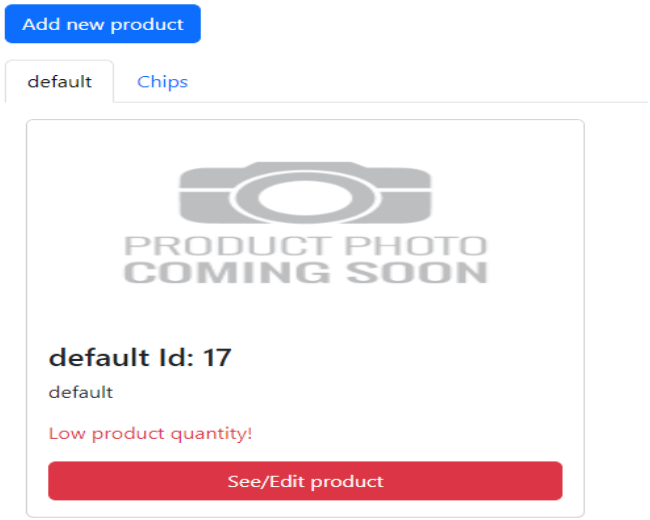


Рисунок 5.6 - Результат додавання товару

5.1.6 **Видалення товару**

Вхідні дані:

– видалення товару здійснюється через натискання кнопки "Edit product" та "Delete product".

Очікуваний результат:

– товар успішно видаляється з бази даних;

– товар зникає зі списку товарів.

Реальний результат:

– товар видалено з бази даних успішно;

– товар зникає успішно.

Результат роботи програми зображено рисунку 5.7 та рисунку 5.8.

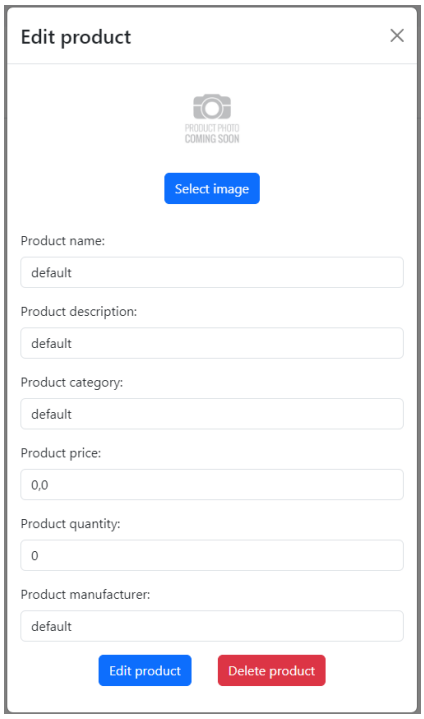


Рисунок 5.7 – Кнопка видалення товару

Рисунок 5.8 – Результат видалення товару

5.1.7 **Зміна паролю акаунту**

Вхідні дані:

– старий пароль;

– новий пароль.

Очікуваний результат:

– пароль змінюється і зміна заноситься до бази даних;

– відображено повідомлення зміну паролю.

Реальний результат:

– пароль змінився і оновився у базі даних;

– повідомлення про успіх відображено.

Результат роботи програми зображено на рисунку 5.9.

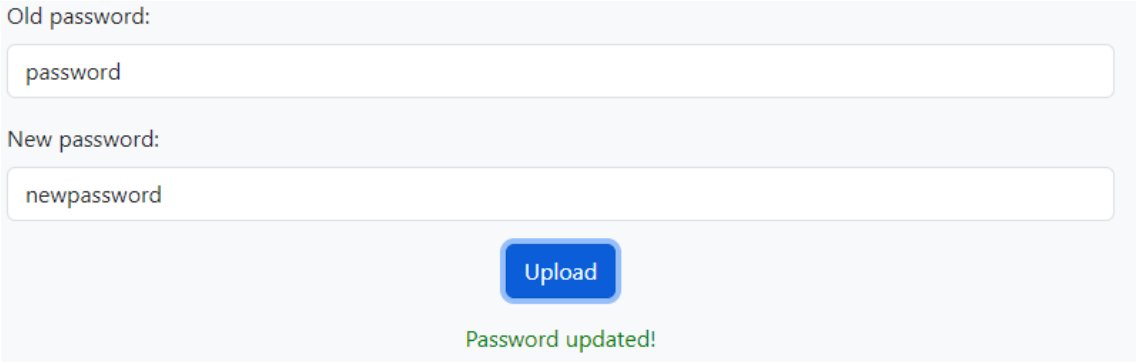


Рисунок 5.9 – Зміна паролю

5.1.8 **Зміна паролю акаунту (старий пароль неправильний)**

Вхідні дані:

– старий неправльний пароль;

– новий пароль.

Очікуваний результат:

– пароль не змінюється;

– відображено повідомлення про помилку.

Реальний результат:

– пароль не змінився;

– повідомлення про помилку відображено.

Результат роботи програми зображено на рисунку 5.10

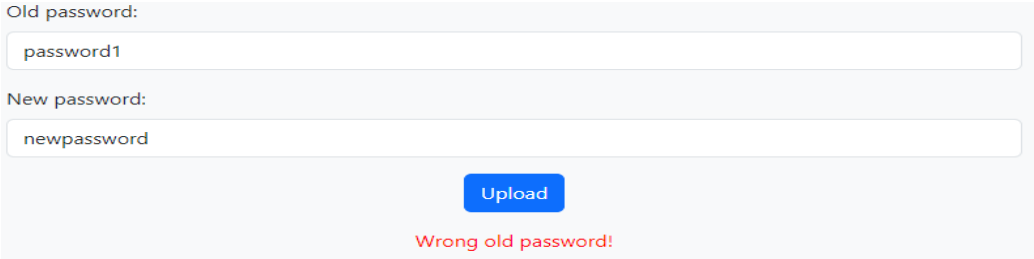


Рисунок 5.10 – Невдала зміна паролю

6 АНАЛІЗ РЕЗУЛЬТАТІВ СТВОРЕННЯ МЕРЕЖЕВОЇ СИСТЕМИ ОЛІКУ ТОВАРІВ

У цьому розділі розглядаються результати виконання дипломної роботи з точки зору досягнення поставлених задач та задоволення функціональних вимог, визначених у розділі "Постановка задачі". Представлений текстовий опис з аналізом кожного результату, а також надано оцінки та рекомендації.

6.1 **Досягнення поставлених задач**

6.1.1 **Вибір інструментів для розробки мережевої системи обліку товарів**

Для розробки системи було обрано такі інструменти:

– веб-сервер: Apache Tomcat, який забезпечує стабільну роботу та обробку HTTP-запитів;

– сервер додатків: Jakarta EE, що дозволяє реалізувати бізнес-логіку на Java.;

– база даних: MySQL, яка забезпечує надійне зберігання та управління даними.;

– клієнтська частина: HTML, CSS, JavaScript з використанням Bootstrap 5 для адаптивного дизайну.

Ці інструменти були обрані через їхню стабільність, гнучкість та здатність інтегруватися, що дозволило створити ефективну систему обліку товарів.

6.1.2 **Розробка структури бази даних**

Структура бази даних була розроблена таким чином, щоб забезпечити ефективне зберігання даних і підтримку всіх необхідних функцій системи. Основні таблиці включають:

– users (Користувачі): Зберігає інформацію про користувачів;

– clients (Клієнти): Зберігає інформацію про клієнтів;

– products (Товари): Зберігає інформацію про товари;

– categories (Категорії): Зберігає категорії товарів;

– orders (Замовлення): Зберігає інформацію про замовлення;

– order_items (Товари замовлень): Зберігає товари замовлень.

Розроблена структура бази даних забезпечує оптимальну організацію та швидкий доступ до даних, що сприяє ефективній роботі системи.

6.1.3 **Розробка серверної частини системи**

Серверна частина була реалізована на основі архітектури клієнт-сервер. Основні компоненти включають:

– сервлети: Обробляють HTTP-запити, виконують бізнес-логіку і взаємодіють з базою даних;

– JSP (JavaServer Pages): Забезпечують динамічне створення веб-сторінок.

Серверна частина реалізує всі необхідні функції для управління товарами, категоріями, користувачами та продажами. Вона також забезпечує безпечну аутентифікацію та управління сесіями. Застосування Jakarta EE дозволило ефективно розділити бізнес-логіку від презентаційного шару, що сприяє легкості підтримки та розширення функціональності системи.

6.1.4 **Розробка клієнтської частини системи**

Клієнтська частина включає:

– HTML та CSS: Для створення структури та стилю веб-сторінок;

– JavaScript: Для забезпечення динамічної взаємодії з користувачем;

– Bootstrap 5: Для адаптивного дизайну та швидкої розробки інтерфейсу.

Це забезпечує зручний та інтуїтивний інтерфейс користувача, який легко адаптується до різних пристроїв і розмірів екранів. Використання Bootstrap 5 дозволило створити сучасний, адаптивний дизайн, який забезпечує позитивний користувацький досвід на будь-якому пристрої.

6.1.5 **Аналіз функціональних вимог**

6.1.5.1 **Облік товарів**

Система дозволяє додавати, редагувати та видаляти товари. Всі операції з товарами виконуються коректно, що підтверджується результатами

тестування. Наприклад, товари можуть бути додані як зі значеннями за замовчуванням, так і з введеними користувачем даними.

6.1.5.2 **Керування категоріями товарів**

Система забезпечує можливість додавання, редагування та видалення категорій товарів. Це дозволяє ефективно класифікувати товари та спрощує їх пошук і управління.

6.1.5.3 **Облік продажів**

Функціонал обліку продажів дозволяє реєструвати нові продажі, переглядати історію продажів та аналізувати дані. Система коректно обробляє всі операції з продажами, включаючи облік доходів та повернення товарів.

6.1.5.4 **Керування користувачами**

Система підтримує функції додавання, редагування та видалення користувачів. Забезпечується безпечна аутентифікація та управління ролями користувачів, що дозволяє контролювати доступ до різних частин системи.

6.1.5.5 **Аналітика**

Система надає можливості для аналізу даних про товари та продажі. Включено звіти та графічні відображення, що допомагають у прийнятті рішень щодо управління запасами та планування продажів.

6.1.5.6 **Інтерфейс користувача**

Інтерфейс розроблений з урахуванням вимог зручності та інтуїтивності. Використання Bootstrap 5 забезпечує адаптивний дизайн, який добре відображається на різних пристроях, таких як комп'ютери, планшети та смартфони.

6.1.6 **Результати тестування**

Проведено тестування ключових функцій системи, включаючи обробку виключних ситуацій та роботу з некоректними даними. Всі функції працюють коректно, система обробляє помилки та забезпечує стабільну роботу.

Перелік тестових сценаріїв наступний:

– логін до акаунту: Перевірка успішного логіну з правильними даними та виводу повідомлення про помилку логіну з неправильними даними

– реєстрація акаунту: Перевірка успішної реєстрації акаунту з унікальним логіном та поштою, та виводу повідомлення про помилку реєстрації з повторним логіном та поштою

– керування товарами: Перевірка успішного додавання та видалення товарів

– зміна паролю: Перевірка успішної зміни паролю акаунту

ВИСНОВКИ

Основна мета дипломної роботи полягала у розробці мережевої системи обліку товарів, яка дозволяє ефективно керувати товарними запасами, продажами та користувачами. На основі проведеного дослідження та розробки можна впевнено стверджувати, що поставлена мета досягнута. Система відповідає всім визначеним функціональним вимогам та забезпечує необхідну продуктивність і стабільність.

Розв'язання задач дослідження

У процесі роботи були розв'язані наступні задачі:

Вибір інструментів для розробки системи:

– як розв’язана: Обрані Apache Tomcat, Jakarta EE, MySQL, HTML, CSS, JavaScript та Bootstrap 5;

– що отримано в результаті: Стабільна, гнучка та ефективна система, що легко інтегрується і забезпечує необхідну функціональність.

Розробка структури бази даних:

– як розв’язана: Створено таблиці для користувачів, клієнтів, товарів, категорій, замовлень та товарів замовлень;

– що отримано в результаті: Оптимальна організація даних, що забезпечує швидкий доступ та ефективне управління даними.

Розробка серверної частини системи:

– як розв’язана: Використано сервлети та JSP для обробки HTTP-запитів і виконання бізнес-логіки;

– що отримано в результаті: Надійна серверна частина, яка забезпечує всі необхідні функції управління та безпечну аутентифікацію.

Розробка клієнтської частини системи:

– як розв’язана: Використано HTML, CSS, JavaScript та Bootstrap 5 для створення інтуїтивного та адаптивного інтерфейсу;

– що отримано в результаті: Зручний інтерфейс, який добре відображається на різних пристроях.

Реалізація функціональних вимог:

– як розв’язана: Впроваджено функції обліку товарів, керування категоріями, обліку продажів, керування користувачами та аналітики;

– що отримано в результаті: Система коректно виконує всі необхідні операції, забезпечуючи ефективне управління та аналіз даних.

Тестування системи:

– як розв’язана: Проведено тестування ключових функцій та обробки виключних ситуацій;

– що отримано в результаті: Всі функції працюють коректно, система обробляє помилки та забезпечує стабільну роботу.

Рекомендації щодо практичного використання здобутих результатів

Впровадження у підприємствах роздрібної торгівлі:

– розроблену систему можна використовувати для автоматизації обліку товарів, управління продажами та користувачами в магазинах та супермаркетах.

Підтримка та розвиток системи:

– рекомендується регулярно оновлювати систему та впроваджувати нові функції для покращення її продуктивності та функціональності.

Інтеграція з іншими системами:

– систему можна інтегрувати з існуючими ERP-системами для забезпечення комплексного управління бізнес-процесами.

Навчання персоналу:

– для ефективного використання системи необхідно провести навчання персоналу, що дозволить максимізувати користь від впровадження системи.

Розроблена мережева система обліку товарів демонструє високу ефективність та відповідає сучасним вимогам. Її впровадження може значно спростити процеси управління товарами та продажами, підвищуючи загальну продуктивність та конкурентоспроможність підприємства.

СПИСОК ДЖЕРЕЛ ІНФОРМАЦІЇ

1 Блош, Джошуа. Effective Java. - 3rd ed. - Addison-Wesley Professional, 2017.

2 Дукетт, Джон. "HTML and CSS: Design and Build Websites." - Wiley, 2011.

3 Траверсі, Бред. Bootstrap 5 From Scratch. - Packt, 2023.

4 Геффельфінгер, Девід Р. Java EE 8 Application Development. - Packt Publishing, 2017.

5 MySQL 8.0 Reference Manual. - Oracle, 2018

6 Хантер, Джейсон. Java Servlet Programming. - O'Reilly Media, 2001.

7 Ангхелд, Леонард. Mastering JavaServer Faces 2.2. - Wiley, 2014.

8 Вінанд, Маркус. SQL Performance Explained. - Scalability Experts, 2010.

9 Браун, Саймон. Pro JSP 2. - Apress, 2004.

10 Шилдт, Герберт. Java: The Complete Reference. - 11th ed. - McGraw-Hill Education, 2018.

11 Фаулер, Мартін. Patterns of Enterprise Application Architecture. - Addison-Wesley Professional, 2002.

12 Харрис. Енді. HTML5 and CSS3 All-in-One For Dummies. - For Dummies, 2014.

ДОДАТОК А

ТЕКСТ СЕРВЕРНОЇ ЧАСТИНИ

```
package Servlets;

import Classes.DatabaseConnection;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

import java.io.IOException;
import java.sql.*;

@WebServlet(name = "RegisterServlet", value = "/RegisterServlet")
public class RegisterServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws IOException {
        Connection connection;
        PreparedStatement statement;
        String name;
        String surname;
        String email;
        String login;
        String password;
        try {
            name = request.getParameter("name");
            surname = request.getParameter("surname");
            email = request.getParameter("email");
            login = request.getParameter("login");
            password = request.getParameter("password");
            connection = DatabaseConnection.getInstance().getConnection();
            statement = connection.prepareStatement("INSERT INTO users
(user_name, user_surname, user_email, user_login, user_password, user_role) values (?,
?, ?, ?, ?, ?)");
            statement.setString(1, name);
            statement.setString(2, surname);
            statement.setString(3, email);
            statement.setString(4, login);
            statement.setString(5, password);
            statement.setString(6, "user");
            statement.executeUpdate();
            response.setContentType("text/plain");
            response.setCharacterEncoding("UTF-8");
            response.getWriter().write("Success");
        } catch (SQLException | ClassNotFoundException e) {
            response.setContentType("text/plain");
            response.setCharacterEncoding("UTF-8");
            response.getWriter().write("Fail");
        }
    }
}
package Servlets;

import Classes.DatabaseConnection;
import Classes.Users;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
```

```
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.http.HttpSession;

import java.io.IOException;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Objects;

@WebServlet(name = "LoginServlet", value = "/LoginServlet")
public class LoginServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws IOException {
        Connection connection;
        PreparedStatement statement;
        String login;
        String password;
        try {
            login = request.getParameter("login");
            password = request.getParameter("password");
            connection = DatabaseConnection.getInstance().getConnection();
            statement = connection.prepareStatement("SELECT * FROM users WHERE
user_login = ?");
            statement.setString(1, login);
            ResultSet resultSet = statement.executeQuery();
            if(resultSet.next()){
                if(Objects.equals(resultSet.getString("user_password"),
password)){
                    Users user = new Users();
                    user.setId(resultSet.getInt("user_id"));
                    user.setName(resultSet.getString("user_name"));
                    user.setSurname(resultSet.getString("user_surname"));
                    user.setEmail(resultSet.getString("user_email"));
                    user.setPassword(resultSet.getString("user_password"));
                    user.setRole(resultSet.getString("user_role"));
                    user.setLogin(resultSet.getString("user_login"));
                    user.setAvatar(resultSet.getString("user_avatar"));
                    HttpSession session = request.getSession();
                    session.setAttribute("user", user);
                    response.setContentType("text/plain");
                    response.setCharacterEncoding("UTF-8");
                    response.getWriter().write("Success");
                }
                else{
                    response.setContentType("text/plain");
                    response.setCharacterEncoding("UTF-8");
                    response.getWriter().write("PassFail");
                }
            }else{
                response.setContentType("text/plain");
                response.setCharacterEncoding("UTF-8");
                response.getWriter().write("LoginFail");
            }
        } catch (SQLException | ClassNotFoundException e) {
            throw new RuntimeException(e);
        }
    }
}
package Servlets;
```

```
import Classes.DatabaseConnection;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

import java.io.IOException;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;

@WebServlet(name = "AddProductServlet", value = "/AddProductServlet")
public class AddProductServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws IOException {
        Connection connection;
        PreparedStatement statement;
        try {
            connection = DatabaseConnection.getInstance().getConnection();
            statement = connection.prepareStatement("INSERT INTO products ()
values ()");
            statement.executeUpdate();
        } catch (SQLException | ClassNotFoundException e) {
            throw new RuntimeException(e);
        }
        response.sendRedirect(request.getContextPath() + "/products.jsp");
    }
}
package Servlets;

import Classes.DatabaseConnection;
import Classes.Users;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

import java.io.IOException;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Objects;

@WebServlet(name = "ChangePasswordServlet", value = "/ChangePasswordServlet")
public class ChangePasswordServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws IOException {
        String oldpass = request.getParameter("oldpass");
        String newpass = request.getParameter("newpass");
        int id = ((Users) request.getSession().getAttribute("user")).getId();
        Connection connection;
        PreparedStatement statement;
        try {
            connection = DatabaseConnection.getInstance().getConnection();
            statement = connection.prepareStatement("SELECT user_password FROM
users WHERE user_id = ?");
```

```
                    statement.setInt(1, id);
                    ResultSet resultSet = statement.executeQuery();
                    if(resultSet.next()){
                        if (Objects.equals(resultSet.getString(1), oldpass)){
                            statement = connection.prepareStatement("UPDATE users SET
user_password = ? WHERE user_id = ?");
                            statement.setString(1, newpass);
                            statement.setInt(2, id);
                            statement.executeUpdate();
                            response.setContentType("text/plain");
                            response.setCharacterEncoding("UTF-8");
                            response.getWriter().write("Success");
                        }else{
                            response.setContentType("text/plain");
                            response.setCharacterEncoding("UTF-8");
                            response.getWriter().write("Fail");
                        }
                    }
                } catch (SQLException | ClassNotFoundException e) {
                    throw new RuntimeException(e);
                }
            }
        }
        package Servlets;

        import Classes.DatabaseConnection;
        import jakarta.servlet.ServletException;
        import jakarta.servlet.annotation.WebServlet;
        import jakarta.servlet.http.HttpServlet;
        import jakarta.servlet.http.HttpServletRequest;
        import jakarta.servlet.http.HttpServletResponse;

        import java.io.File;
        import java.io.IOException;
        import java.sql.Connection;
        import java.sql.PreparedStatement;
        import java.sql.SQLException;
        import java.util.Objects;

        @WebServlet(name = "DeleteProductServlet", value = "/DeleteProductServlet")
        public class DeleteProductServlet extends HttpServlet {
            @Override
            protected void doPost(HttpServletRequest request, HttpServletResponse
response) {
                Connection connection;
                PreparedStatement statement;
                int id = Integer.parseInt(request.getParameter("id"));
                String img = request.getParameter("img");
                if(!Objects.equals(img, "default.png")){
                    File file = new
File(getServletContext().getRealPath("/ProductsImages/")+img);
                    boolean deleted = file.delete();
                }
                    try {
                        connection = DatabaseConnection.getInstance().getConnection();
                        statement = connection.prepareStatement("DELETE FROM products
WHERE product_id = ?");
                        statement.setInt(1, id);
                        statement.executeUpdate();
                    } catch (SQLException | ClassNotFoundException e) {
                        throw new RuntimeException(e);
                    }
```

```
        }
    }
    package Servlets;

    import Classes.DatabaseConnection;
    import jakarta.servlet.ServletException;
    import jakarta.servlet.annotation.WebServlet;
    import jakarta.servlet.http.HttpServlet;
    import jakarta.servlet.http.HttpServletRequest;
    import jakarta.servlet.http.HttpServletResponse;

    import java.io.IOException;
    import java.sql.Connection;
    import java.sql.PreparedStatement;
    import java.sql.SQLException;
    @WebServlet(name = "EditProductServlet", value = "/EditProductServlet")
    public class EditProductServlet extends HttpServlet {
        @Override
        protected void doPost(HttpServletRequest request, HttpServletResponse
response) {
            Connection connection;
            PreparedStatement statement;
            int id = Integer.parseInt(request.getParameter("id"));
            String name = request.getParameter("name");
            String desc = request.getParameter("desc");
            double price = Double.parseDouble((request.getParameter("price")));
            int quantity = Integer.parseInt(request.getParameter("quantity"));
            String manufacturer = request.getParameter("manufacturer");
            String img = request.getParameter("img");
            int category = Integer.parseInt(request.getParameter("category"));
            try {
                connection = DatabaseConnection.getInstance().getConnection();
                statement = connection.prepareStatement("UPDATE products SET
product_name = ?,  product_desc = ?, product_price = ?, product_quantity = ?,
product_manufacturer = ?, product_img=?, fk_category_id = ?  WHERE product_id = ?");
                statement.setString(1, name);
                statement.setString(2, desc);
                statement.setDouble(3, price);
                statement.setInt(4, quantity);
                statement.setString(5, manufacturer);
                statement.setString(6, img);
                statement.setInt(7, category);
                statement.setInt(8, id);
                statement.executeUpdate();
            } catch (SQLException | ClassNotFoundException e) {
                throw new RuntimeException(e);
            }
        }
    }
```

ДОДАТОК Б

ТЕКСТ КЛІЄНТСКОЇ ЧАСТИНИ

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head>
    <title>Login</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
rel="stylesheet">
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"></sc
ript>
    <script src="Scripts/ajaxLogin.js"></script>
</head>
<body>
<div class="d-flex align-items-center min-vh-100">
<div class="container" style="background-color: rgb(248,249,250); border-radius:
20px; width: 50vw">
<form>
    <div class="mb-3">
        <label for="login" class="form-label">Login:</label>
        <input type="text" class="form-control" id="login" placeholder="Enter
login" name="login">
    </div>
    <div class="mb-3">
        <label for="password" class="form-label">Password:</label>
        <input type="password" class="form-control" id="password"
placeholder="Enter password" name="password">
    </div>
    <div class="d-flex justify-content-center mb-3">
        <button type="button" class="btn btn-primary"
onclick="userLogin()">Submit</button>
    </div>
    <div class="d-none justify-content-center mb-3" id="result">
        <span class="mx-auto" style="color: red; " id="resinfo"></span>
    </div>
    <div class="d-flex justify-content-center mb-3">
        <a href="register.jsp">Register</a>
    </div>
</form>
</div>
</div>
</body>
</html>
<%@ page contentType="text/html;charset=UTF-8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
    <title>Products</title>
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
rel="stylesheet">
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"></sc
ript>
    <link rel="stylesheet" href="CSS/sidebar.css">
```

```
            <link rel="stylesheet" href="CSS/avatar.css">
            <script src="Scripts/showProductInfo.js"></script>
        </head>
        <body>
        <jsp:include page="/LoadProductsServlet"/>
        <div id="wrapper">
            <%@ include file="sidebar.jsp" %>
            <%@ include file="Modals/modal_addproduct.jsp" %>
            <div id="page-content-wrapper">
                <%@ include file="navbar.jsp" %>
                <div class="container-fluid">
                    <div class="d-flex">
                        <form action="AddProductServlet" method="post" class="my-3">
                            <button class="btn btn-primary" type="submit">Add new
product</button>
                        </form>
                    </div>
                    <ul class="nav nav-tabs" id="productTabs" role="tablist">
                        <c:forEach var="category" items="${productsByCategory.keySet()}"
varStatus="status">
                            <li class="nav-item" role="presentation">
                                <a class="nav-link ${status.index == 0 ? 'active' : ''}"
id="${category}-tab" data-bs-toggle="tab" href="#${category}" role="tab" aria-
controls="${category}" aria-selected="${status.index == 0 ? 'true' :
'false'}">${category}</a>
                            </li>
                        </c:forEach>
                    </ul>

                    <div class="tab-content" id="productTabsContent">
                        <c:forEach var="category" items="${productsByCategory.keySet()}"
varStatus="status">
                            <div class="tab-pane fade ${status.index == 0 ? 'show active'
: ''}" id="${category}" role="tabpanel" aria-labelledby="${category}-tab">
                                <ul class="d-flex flex-wrap" style="list-style-type:none;
padding: 0">
                                    <c:forEach var="product"
items="${productsByCategory[category]}" varStatus="loop">
                                        <li style="width: 22.8%" class="m-3"
id="${loop.index}">
                                            <div class="card">
                                                <img class="card-img-top"
src="${pageContext.request.contextPath}/ProductsImages/${product.img}" alt="Изображение
товара" style="height: 200px;width:100%">
                                                <div class="card-body d-flex flex-
column">
                                                    <h4 class="card-
title">${product.name} Id: ${product.id}</h4>
                                                    <p class="card-
text">${product.desc}</p>
                                                    <c:if test="${product.quantity <=
5}">
                                                        <p class="text-danger">Low
product quantity!</p>
                                                    </c:if>
                                                    <a href="#" class="btn btn-danger"
data-bs-toggle="modal" data-bs-target="#modalAddProduct"
onclick="showProductInfo('${product.category_id}','${product.id}','${product.name}',
'${product.desc}', '${product.price}', '${product.quantity}',
'${product.manufacturer}', '${product.img}')">See/Edit product</a>
                                                </div>
                                            </div>
```

```
                                    </li>
                                </c:forEach>
                            </ul>
                        </div>
                    </c:forEach>
                </div>
            </div>
        </div>
    </div>
    </body>
    </html>
    <%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
    <!DOCTYPE html>
    <html>
    <head>
        <title>Register</title>
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
rel="stylesheet">
        <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"></sc
ript>
        <script src="Scripts/ajaxRegister.js"></script>
    </head>
    <body>
    <div class="d-flex align-items-center min-vh-100">
    <div class="container" style="background-color: rgb(248,249,250); border-radius:
20px; width: 50vw">
        <form id="myForm">
            <div class="mb-3">
                <label for="name" class="form-label">Name:</label>
                <input type="text" class="form-control" id="name" placeholder="Enter
surname" name="name" required value="${name_val}">
            </div>
            <div class="mb-3">
                <label for="surname" class="form-label">Surname:</label>
                <input type="text" class="form-control" id="surname"
placeholder="Enter surname" name="surname" required value="${surname_val}">
            </div>
            <div class="mb-3">
                <label for="login" class="form-label">Login:</label>
                <input type="text" class="form-control" id="login" placeholder="Enter
login" name="login" required value="${login_val}">
            </div>
            <div class="mb-3">
                <label for="email" class="form-label">Email:</label>
                <input type="email" class="form-control" id="email"
placeholder="Enter email" name="email" required value="${email_val}">
            </div>
            <div class="mb-3">
                <label for="password" class="form-label">Password:</label>
                <input type="password" class="form-control" id="password"
placeholder="Enter password" name="password" required value="${password_val}">
            </div>
            <div class="d-grid gap-3 col-6 mx-auto">
                <button type="button" class="btn btn-primary mb-3" id="submitBtn"
onclick="userRegister()">Submit</button>
            </div>
            <div class="d-none justify-content-center mb-3" id="result">
                <span class="mx-auto" style="color: red; " id="resinfo">Login or
email is already used!</span>
```

```
            </div>
        </form>
    </div>
    </div>
    </body>
    </html>
    <%@ page contentType="text/html;charset=UTF-8" %>
    <html>
    <head>
        <title>Account</title>
        <script
src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>
        <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
rel="stylesheet">
        <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"></sc
ript>
        <link rel="stylesheet" href="CSS/sidebar.css">
        <link rel="stylesheet" href="CSS/avatar.css">
        <script src="Scripts/ajaxDeleteAccount.js"></script>
        <script src="Scripts/ajaxChangePass.js"></script>
    </head>
    <body>
    <div id="wrapper" >
    <%@ include file="sidebar.jsp" %>
    <%@ include file="Modals/modal_delete.jsp" %>
    <div id="page-content-wrapper" style="height: 90%">
        <%@ include file="navbar.jsp" %>
        <div class="container-fluid d-flex mt-3" style="background-color:
rgb(248,249,250); border-radius: 10px; width: 60%">
            <div class="d-flex flex-column" style="width: 70%">
                <h2 class="d-flex">Account information</h2>
                <form class="">
                    <div class="mb-3">
                        <label for="fullname" class="form-label">Name:</label>
                        <input type="text" class="form-control" id="fullname"
placeholder="Enter login" name="fullname" value="<%out.print(((Users)
session.getAttribute("user")).getName()+" "+ ((Users)
session.getAttribute("user")).getSurname());%>" readonly>
                    </div>
                    <div class="">
                        <label for="email" class="form-label">Email:</label>
                        <input type="text" class="form-control" id="email"
placeholder="Enter password" name="email" value="<%out.print(((Users)
session.getAttribute("user")).getEmail());%>" readonly>
                    </div>
                </form>
                <h2 class="d-flex">Change password</h2>
                <form class="d-flex flex-column">
                    <div class="mb-3">
                        <label for="oldpass" class="form-label">Old password:</label>
                        <input type="text" class="form-control" id="oldpass"
placeholder="Enter old password" name="oldpass" required>
                    </div>
                    <div class="mb-3">
                        <label for="newpass" class="form-label">New password:</label>
                        <input type="text" class="form-control" id="newpass"
placeholder="Enter new password" name="newpass" required>
                    </div>
                    <div class="d-flex justify-content-center">
```

```
                            <button type="button" class="btn btn-primary"
onclick="passChange()">Upload</button>
                        </div>
                        <div class="d-none justify-content-center mt-3" id="changepas">
                            <span id="changeres" style="color: red;"></span>
                        </div>
                    </form>
                    <div class="d-flex justify-content-center mb-3">
                        <button type="button" class="btn btn-danger" data-bs-
toggle="modal" data-bs-target="#modalDelete">
                            Delete account
                        </button>
                    </div>
                </div>
                <div class="d-flex justify-content-center mt-1" style="width: 30%">
                    <form action="UploadServlet" method="post" enctype="multipart/form-
data" class="d-flex flex-column">
                        <div class="d-flex justify-content-center mb-3">
                            <img
src="${pageContext.request.contextPath}/UsersAvatars/<%=((Users)
session.getAttribute("user")).getAvatar()%>" title="Select avatar" alt="Default"
id="avatar" style="width:100px; height: 100px;" class="rounded-pill">
                        </div>
                        <div class="d-flex justify-content-center mb-3">
                            <input type="file" class="form-control" id="image"
name="image" accept="image/*" required hidden="hidden">
                            <button type="submit" class="btn btn-primary">Upload</button>
                        </div>
                        <div class="d-flex justify-content-center mb-3">
                            <span id="result" style="color: green;">${res}</span>
                        </div>
                    </form>
                </div>
            </div>
            </div>
        </div>
        <script>
            var avatarInput = document.getElementById('image');

            var selectFileButton = document.getElementById('avatar');

            selectFileButton.addEventListener('click', function() {
                avatarInput.click();
            });
        </script>
        </body>
</html>
```

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Інститут комп’ютерних наук та інформаційних технологій

Кафедра системного аналізу та інформаційно-аналітичних технологій

Спеціальність Комп’ютерні науки

Освітня програма Комп’ютерні науки

ІЛЮСТРАТИВНІ МАТЕРІАЛИ

Тема роботи РОЗРОБКА МЕРЕЖЕВОЇ СИСТЕМИ ОБЛІКУ ТОВАРІВ НЕВЕЛИКОЇ КОМПАНІЇ

Виконавець Молчанов Богдан Сергійович
(прізвище, ім’я та по-батькові)

Керівник Доцент Роговий Антон Іванович
(посада, прізвище, ім’я та по-батькові)

Харків 2024

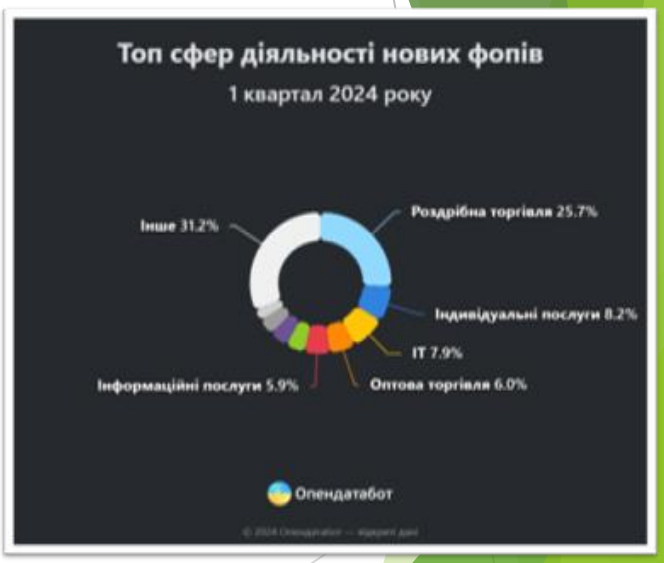
Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Розробка мережової системи обліку товарів

Виконав: ст. групи КН-320в
Молчанов. Б. С.
Перевірив: к. т. н., доцент
Роговий А. І.

Серед усіх відкритих ФОПів 25% займаються роздібною торгівлею. ФОПи часто обирають роздрібну торгівлю через кілька ключових причин:

- низький поріг входу;
- гнучкість та незалежність;
- широкий ринок;
- відносно просте регулювання;
- швидкий оборот коштів;
- можливість масштабування.



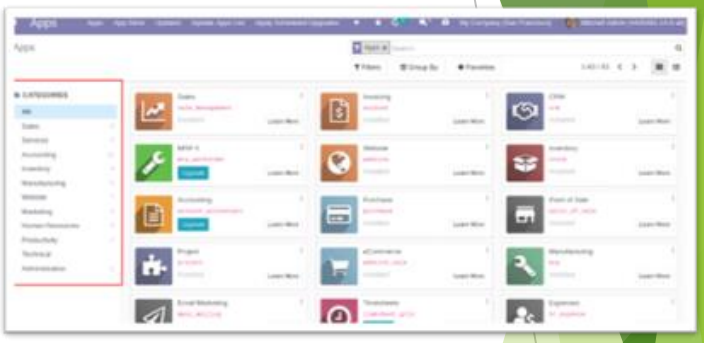
Додаток “Odoo”

Переваги:

- інтеграція;
- гнучкість;
- відкритий код.

Недоліки:

- складність;
- вартість.



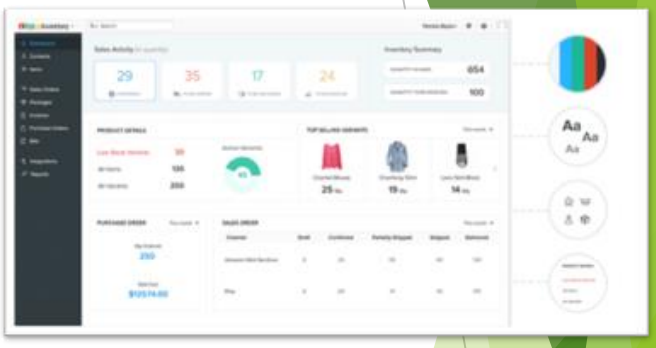
Додаток “Zoho Inventory”

Переваги:

- інтуїтивний інтерфейс;
- інтеграція з іншими Zoho продуктами.

Недоліки:

- обмежена функціональність; вартість.



Додаток “TradeGecko”

Переваги:

- інтуїтивність;
- функціональність.

Недоліки:

- ціна;
- обмеження.

Мета роботи – розробка мережевої системи обліку товарів для невеликої компанії.

Задачі, що потрібно вирішити для досягнення мети:

- вибір інструментів для розробки
- розробка структури бази даних
- реалізація серверної та клієнтської частини

До функціональних вимог програмного забезпечення, яке розробляється , відносять:

- облік товарів;
- керування категоріями товарів;
- облік продажів;
- керування користувачами;
- аналітика;
- інтерфейс користувача.

Intellij IDEA

IntelliJ IDEA було обрано як інтегроване середовище розробки (IDE) з наступних причин:

- IntelliJ IDEA забезпечує високу продуктивність завдяки інтелектуальним підказкам, автоматичному завершенню коду та підтримці рефакторингу;
- IDE повністю підтримує розробку на Java та Jakarta EE, включаючи інтеграцію з популярними серверами додатків та системами управління версіями;
- IntelliJ IDEA легко інтегрується з іншими інструментами розробки, такими як Maven, Gradle та Docker, що робить процес розробки більш гнучким та масштабованим.

Jakarta EE

Jakarta EE було обрано для створення серверної частини системи з наступних причин:

- Jakarta EE є стандартом для розробки корпоративних додатків на Java, що забезпечує стабільність та тривалу підтримку;
- платформа забезпечує високий рівень масштабованості та надійності завдяки вбудованим механізмам обробки транзакцій, безпеки та управління ресурсами;
- платформа надає широкий набір API для роботи з базами даних, веб-сервісами, обробки запитів та багато іншого, що спрощує процес розробки.

JavaScript

Для розробки клієнтської частини системи було обрано JavaScript з наступних причин:

- JavaScript є основною мовою програмування для веб-розробки і підтримується всіма сучасними веб-браузерами;
- JavaScript має велику спільноту розробників та широкий набір бібліотек і фреймворків, що полегшує процес розробки та обслуговування додатка;
- JavaScript дозволяє створювати динамічні та інтерактивні інтерфейси користувача, що покращує взаємодію з користувачем та підвищує зручність використання системи.

Bootstrap 5

Для створення інтерфейсу користувача було обрано Bootstrap 5 з наступних причин:

- Bootstrap забезпечує швидку та ефективну розробку інтерфейсів завдяки великій кількості готових компонентів та шаблонів;
- фреймворк підтримує адаптивний дизайн, що забезпечує коректне відображення інтерфейсу на різних пристроях та розмірах екрану;
- Bootstrap має велику спільноту розробників та добре задокументовану документацію, що спрощує процес навчання та використання.

Структура баз даних

Основні таблиці включають:

- users (Користувачі): Зберігає інформацію про користувачів;
- clients (Клієнти): Зберігає інформацію про клієнтів;
- products (Товари): Зберігає інформацію про товари;
- categories (Категорії): Зберігає категорії товарів;
- orders (Замовлення): Зберігає інформацію про замовлення;
- order_items (Товари замовлень): Зберігає товари замовлень.

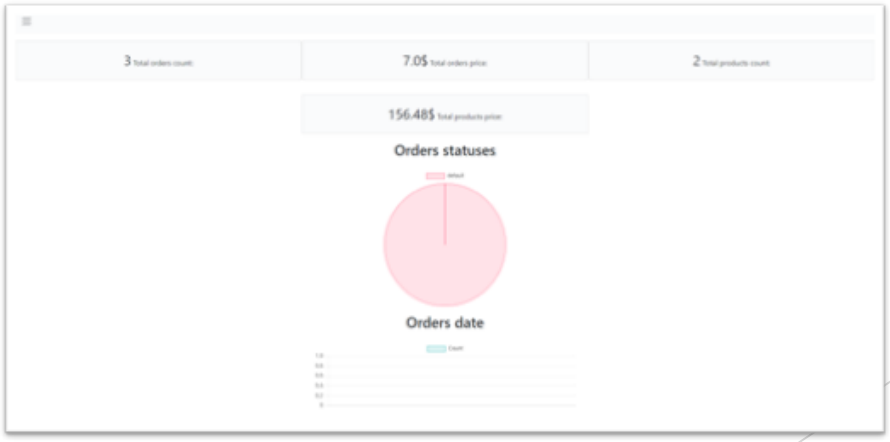
Реалізація серверної частини

- Використання сервлетів та JSP для обробки запитів
- Архітектура клієнт-сервер

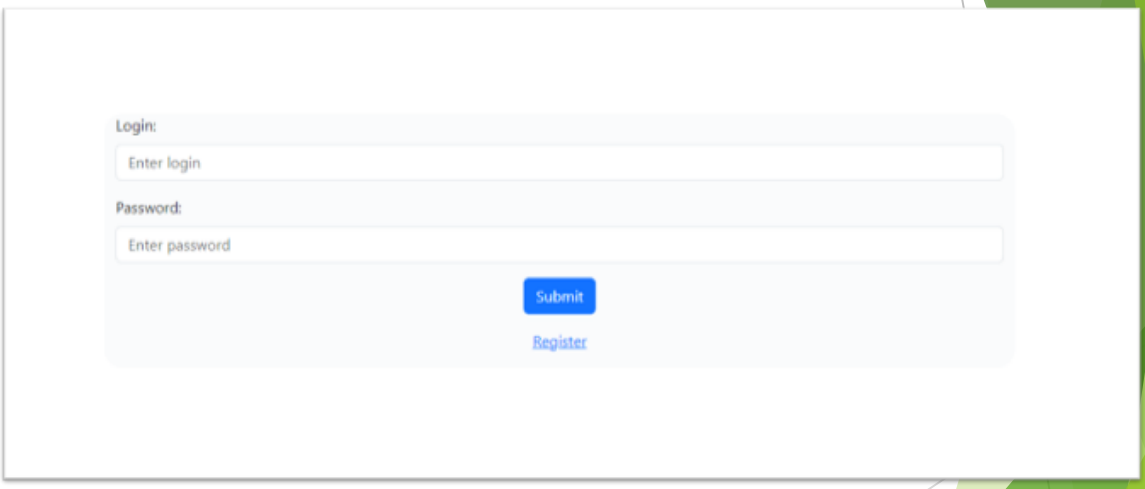
```java
package Servlets;

import ...

@WebServlet(name = "LogoutServlet", value = "/LogoutServlet")
public class LogoutServlet extends HttpServlet {
    no usages
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException {
        HttpSession session = request.getSession( b: false);

        if (session != null) {
            session.invalidate();
        }
        response.sendRedirect( s: request.getContextPath() + "/index.jsp");
    }
}
```

Реалізація клієнтської частини

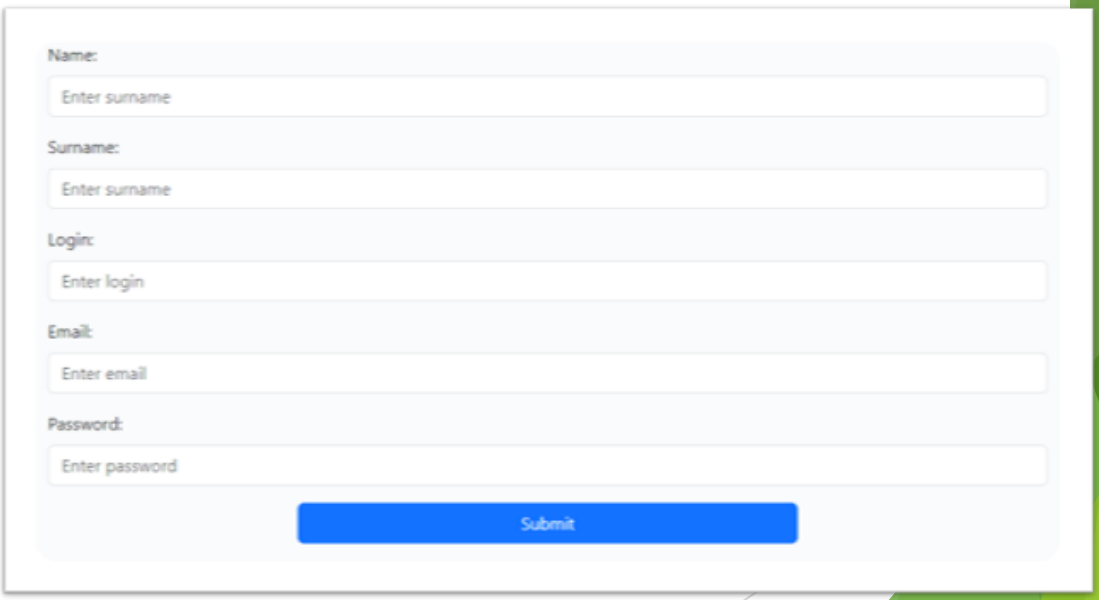
- HTML, CSS, JavaScript для створення інтерфейсу;
- Використання Bootstrap 5 для адаптивного дизайну.



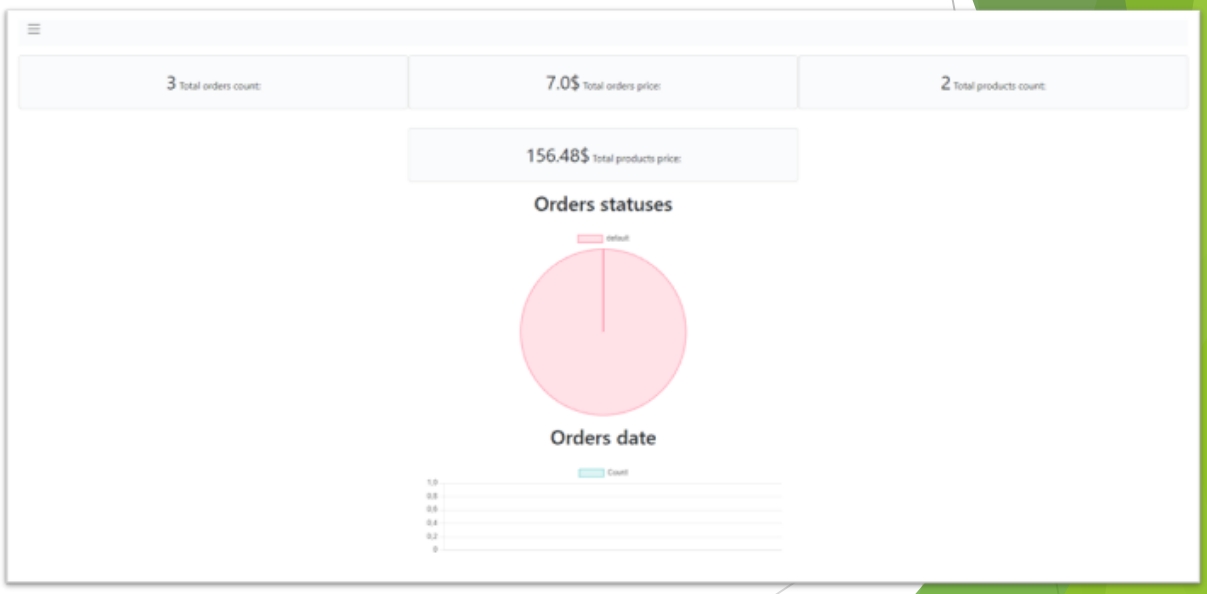
Логін



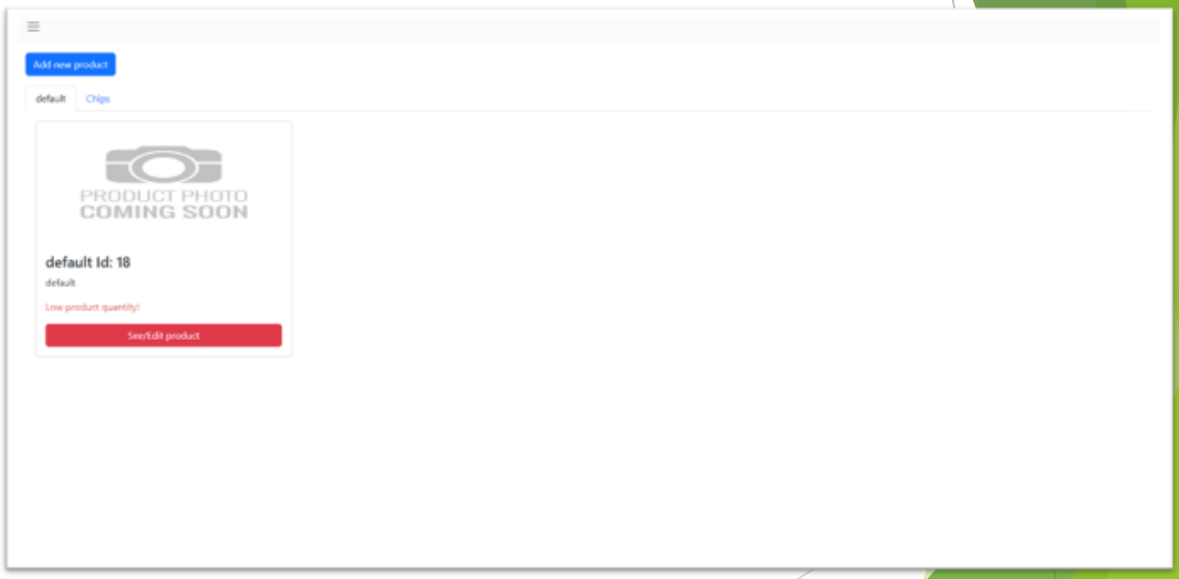
Реєстрація



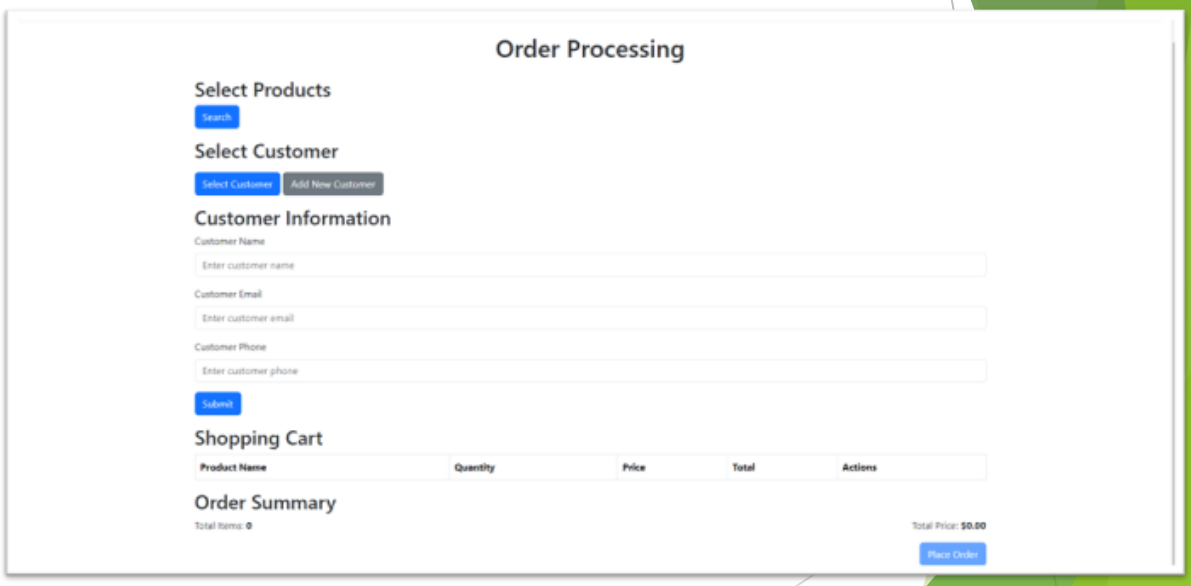
Головне меню



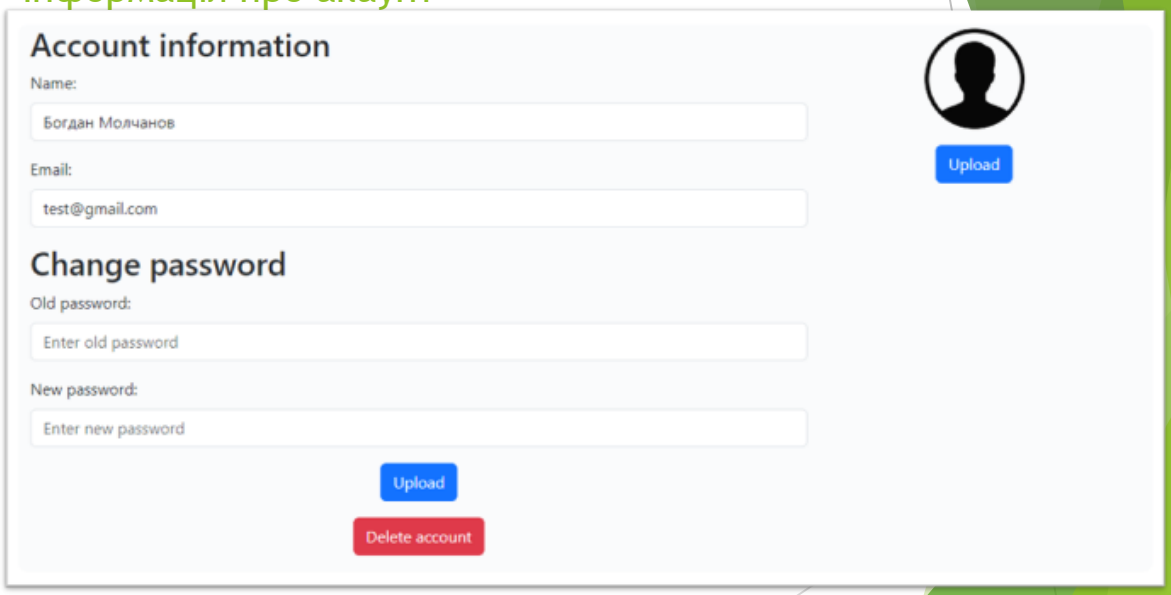
Товари



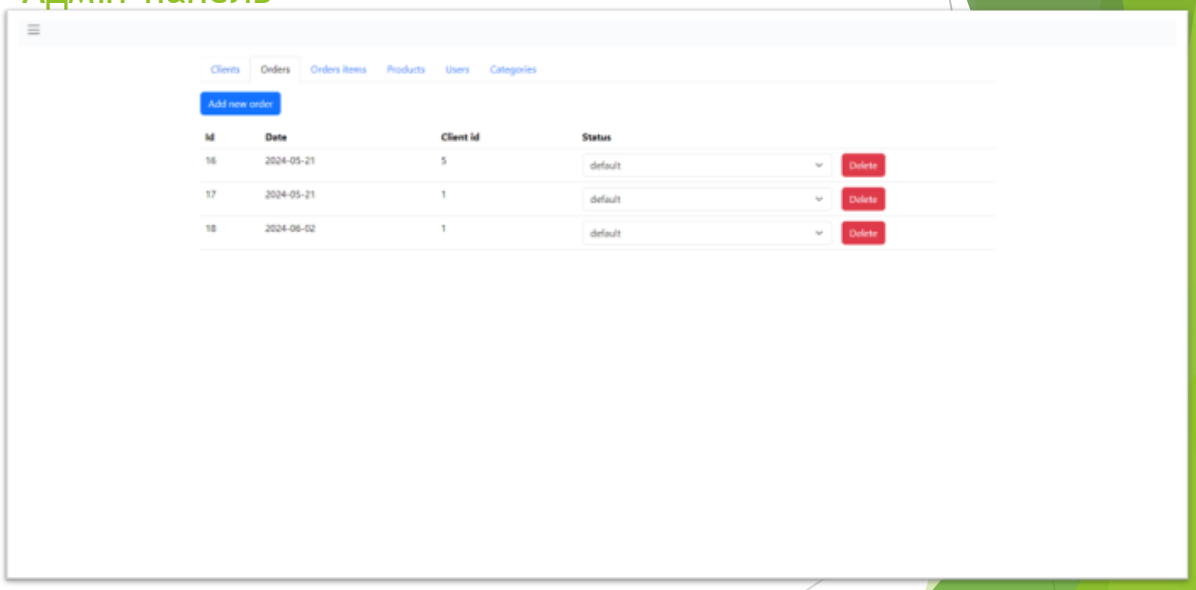
Створення замовлення



Інформація про акаунт



Адмін-панель



Висновки

Основна мета дипломної роботи полягала у розробці мережевої системи обліку товарів, яка дозволяє ефективно керувати товарними запасами, продажами та користувачами. На основі проведеного дослідження та розробки можна впевнено стверджувати, що поставлена мета досягнута. Система відповідає всім визначеним функціональним вимогам та забезпечує необхідну продуктивність і стабільність.

Дякую за увагу!