МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯ ОНЛАЙН ТОРГІВЛІ ЗАСОБАМИ FLASK ТА POSTGRESQL»**

Виконав: студент 2 курсу, групи 8.1213-іпз-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
К.О. Кабанов
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, к.ф.-м.н. Кривохата А.Г.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри фундаментальної та прикладної математики, доцент, к.ф.-м.н. Панасенко Є.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Кабанову Костянтину Олексійовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка вебзастосунку для онлайн торгівлі засобами Flask та PostgreSQL

керівник роботи Кривохата Анастасія Григоріївна, к.ф.-м.н.
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » березня 2024 року № 385-с

2. Строк подання студентом роботи 25.11.2024 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Програмна реалізація застосунку та тестування.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

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

7. Дата видачі завдання 07.03.2024 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	25.03.2024	
2.	Збір вихідних даних.	17.04.2024	
3.	Обробка методичних та теоретичних	11.06.2024	
	джерел.		
4.	Розробка першого та другого розділу.	21.08.2024	
5.	Розробка третього розділу.	05.11.2024	
6.	Оформлення та нормоконтроль	20.11.2024	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	11.12.2024	

Студент _____ К.О. Кабанов
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.Г. Кривохата
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка вебзастосунку для онлайн торгівлі засобами Flask та PostgreSQL»: 60 с., 46 рис., 7 табл., 13 джерел.

АСИНХРОННІ ЗАПИТИ, БАЗА ДАНИХ, БЕЗПЕКА, ВЕБЗАСТОСУНОК, ВЕБРОЗРОБКА, ІНТЕГРАЦІЯ, ІНТЕРФЕЙС КОРИСТУВАЧА, ОНЛАЙН-ТОРГІВЛЯ, ПЛАТІЖНІ СИСТЕМИ, ПРОГРАМУВАННЯ, СЕРВЕРНА ЧАСТИНА, ТРАНЗАКЦІЇ, ФРЕЙМВОРК, FLASK, POSTGRESQL.

Об’єкт дослідження: процес розробки вебзастосунку для онлайн-торгівлі, що використовує фреймворк Flask та базу даних PostgreSQL.

Мета роботи: створення функціонального вебзастосунку для онлайн-торгівлі, який дозволяє користувачам здійснювати покупки та продажі товарів, забезпечуючи ефективну взаємодію між продавцями та покупцями, з високим рівнем безпеки та зручним інтерфейсом.

Метод дослідження: аналітичний метод дослідження, проєктувальний метод, експериментальний метод, метод програмної реалізації та метод тестування.

Робота присвячена розробці вебзастосунку для онлайн-торгівлі, який надає можливість користувачам реєструватися, додавати товари, здійснювати покупки та продавати їх через інтуїтивно зрозумілий інтерфейс. Основними компонентами є серверна частина на Flask, що забезпечує обробку запитів та інтеракцію з базою даних PostgreSQL.

SUMMARY

Master’s qualifying paper “Development of a Web Application for Online Trading Using Flask and PostgreSQL”: 60 pages, 46 figures, 7 tables, 13 references.

ASYNCHRONOUS QUERIES, BACKEND, DATABASE, FLASK, FRAMEWORK, INTEGRATION, ONLINE COMMERCE, PAYMENT SYSTEMS, POSTGRESQL, PROGRAMMING, SECURITY, TRANSACTIONS, USER INTERFACE, WEB APPLICATION, WEB DEVELOPMENT.

The object of the study is the development of a web application for online trading using Flask as the backend framework and PostgreSQL as the database system.

The aim of the study is to create a functional and secure web application for online trading, enabling users to buy and sell products while ensuring seamless interaction between sellers and buyers, as well as a high level of security and an intuitive user interface.

The methods of research are system analysis, software architecture design, backend development with Flask, database design and implementation using PostgreSQL, integration of payment systems, as well as testing and performance evaluation of the application.

This paper focuses on the design, development, and testing of a web application that provides users with the ability to register, list products, make purchases, and sell them through an easy-to-use interface. The main components of the system are the Flask-based server-side logic and the PostgreSQL relational database that manages the product catalog, user accounts, and transactions.

ЗМІСТ

Завдання на кваліфікаційну роботу 2
Реферат 4
Summary 5
Вступ 7
1 Огляд засобів для створення сервісів онлайн торгівлі 9
1.1 Аналіз існуючих платформ для онлайн-торгівлі 9
1.2 Особливості роботи з базами даних у вебдодатках 13
1.3 Огляд фреймворків для веброзробки 17
1.3.1 Fastapi 17
1.3.2 Django 18
1.3.3 Flask 18
1.4 Постановка задачі для створення власного сервісу 20
2 Проєктування архітектури вебдодатку 23
2.1 Аналіз вимог до сервісу онлайн-торгівлі 23
2.2 Проєктування архітектури додатку 26
2.3 Структура бази даних для сервісу (postgresql) 30
2.4 Проєктування ключових модулів додатку 34
2.4.1 Модуль реєстрації та авторизації 35
2.4.2 Модуль управління товарами 36
3 Реалізація функціональних модулів вебдодатку та тестування додатку 38
3.1 Реалізація серверної частини додатку за допомогою flask 38
3.2 Інтеграція бази даних postgresql із додатком 42
3.3 Захист даних 45
3.4 Тестування функціональності модулів 50
3.5 Тестування продуктивності сервісу 53
Висновки 59
Перелік посилань 60

ВСТУП

Онлайн-торгівля сьогодні є одним із найбільш динамічно зростаючих сегментів світової економіки, який значно впливає на розвиток сучасного бізнесу та зміну звичок споживачів. Завдяки стрімкому розвитку інформаційних технологій, процес купівлі-продажу став швидшим, зручнішим і доступнішим для широкого кола користувачів. Такі платформи, як Amazon, eBay, OLX та інші, стали невід'ємною частиною життя багатьох людей, дозволяючи здійснювати покупки з будь-якого куточка світу, економлячи час і ресурси.

Розвиток цифрової економіки сприяв зростанню популярності малого та середнього бізнесу, оскільки створення власного онлайн-магазину стало доступним навіть для підприємців із мінімальними ресурсами. У той же час великі корпорації активно впроваджують інноваційні рішення, які дозволяють підвищувати ефективність бізнес-процесів і забезпечувати лояльність клієнтів.

Особливого значення онлайн-торгівля набула в період пандемії COVID-19, коли багато підприємств були змушені перейти в онлайн-середовище через обмеження фізичних контактів. У цих умовах електронна комерція стала основним інструментом для забезпечення економічної діяльності. Це не лише змінило підходи до ведення бізнесу, а й суттєво вплинуло на глобальні ринки, сприяючи зростанню попиту на зручні та функціональні вебдодатки.

Проте, незважаючи на широкий спектр доступних платформ і рішень, створення сучасного сервісу для онлайн-торгівлі все ще залишається складним завданням. Розробники повинні враховувати безліч технічних аспектів: забезпечення надійності роботи додатку, збереження даних користувачів, оптимізацію продуктивності, інтеграцію з платіжними системами, зручність використання для кінцевих користувачів тощо.

Зважаючи на це, вибір оптимальних технологій для створення вебдодатків є важливим етапом розробки. Серед широкого спектра

фреймворків та баз даних виділяються Flask і PostgreSQL. Flask, як мікрофреймворк, дозволяє створювати легкі, але функціональні додатки, які можна легко адаптувати до потреб бізнесу. PostgreSQL, одна з найпотужніших реляційних баз даних, забезпечує високу надійність, підтримку складних запитів, гнучкість і масштабованість.

Таким чином, дослідження, присвячене розробці сервісу для онлайн-торгівлі на базі Flask та PostgreSQL, є актуальним як із технічної, так і з економічної точки зору. Воно дозволяє поєднати найкращі сучасні інструменти для створення продукту, здатного задовольнити потреби як малих підприємців, так і великих компаній.

Метою даної кваліфікаційної роботи є розробка функціонального сервісу для онлайн-торгівлі, що буде базуватися на Flask та PostgreSQL. Сервіс має забезпечити зручний інтерфейс для користувачів, високу продуктивність, надійне збереження даних і можливість масштабування.

1 ОГЛЯД ЗАСОБІВ ДЛЯ СТВОРЕННЯ СЕРВІСІВ ОНЛАЙН ТОРГІВЛІ

1.1 Аналіз існуючих платформ для онлайн-торгівлі

Сьогодні онлайн-торгівля займає важливе місце у сучасній економіці. Завдяки широкому розповсюдженню інтернету і доступності технологій, платформи електронної комерції стали невід'ємною частиною життя мільйонів людей по всьому світу. Вони пропонують широкий спектр функціональних можливостей, таких як покупка і продаж товарів, послуг, пошук пропозицій, обробка транзакцій та забезпечення логістики. У цьому розділі буде розглянуто три найпопулярніші платформи: OLX, Amazon і eBay.

OLX – це міжнародна платформа для оголошень, яка надає користувачам можливість купувати, продавати або обмінюватися товарами та послугами на локальному рівні. OLX функціонує у понад 30 країнах і є одним із лідерів у сфері локальної електронної комерції. OLX розглядається у цій роботі через певну низку причин. По-перше, ця платформа є найпопулярнішою в Україні через можливість детального пошуку, а по-друге, саме такий детальний пошук зроблений за допомоги Elasticsearch [9].

Після ознайомлення із платформою, пропонуємо розглянути функціональність сервісу OLX. Для початку розглянемо процес реєстрацію та авторизацію. Користувачі можуть швидко створити обліковий запис за допомогою електронної пошти або соціальних мереж. Наступне що будемо розглядати, це публікації оголошень. OLX дозволяє розміщувати оголошення з описом товарів, фотографіями та цінами. Користувачі також можуть вибирати категорії для полегшення пошуку. Окрім розгялутих основних функціональних особливостей потрібно ознайомитись із комунікацією між користувачами. Система вбудованих повідомлень дозволяє покупцям і продавцям спілкуватися безпосередньо на платформі. Для більш зручного пошуку розгянемо фільтри пошуку. Пошук товарів за категоріями, ціною, місцем розташування тощо. І

для того, щоб бути впевненим у продавці розгянемо принцип роботи рейтингу користувачів. продавці та покупці можуть оцінювати одне одного, що допомагає підвищувати довіру до платформи.

Після огляду функціональності сервісу, потрібно виділити переваги розглядаємого сервісу. Почнемо із переваг сервісу OLX. Перше що необхідно зазначити це – безкоштовність. Більшість послуг доступна безкоштовно. Окрім цього платформа орієнтована на місцевий ринок, що дозволяє користувачам здійснювати покупки та продажі в межах свого міста. Також платформа має мінімалістичний інтерфейс, який робить платформу зручною навіть для непідготовлених користувачів.

Як і у кожного сервісу, у сервісу, який ми розглядаємо, є недоліки. Для уникнення повторень таких помилок розглянемо їх. Перше та основне – це відсутність гарантій. Тому що платформа не бере участі у процесі купівлі-продажу, тому ризики повністю лежать на користувачах. Також на сервісу має місце шахрайство. Це відбувається через низький рівень перевірки оголошень відкриває можливості для зловживань. Останій недолік, який необхідно розглянути – це обмежена функціональність для бізнесу. Так як OLX більше підходить для приватних осіб, ніж для великих компаній.

Amazon є однією з найбільших платформ електронної комерції у світі. Вона пропонує широкий асортимент товарів і послуг, включаючи ритейл, хмарні сервіси (AWS), цифровий контент та власні лінії продуктів. Amazon розглядається в контексті цієї роботи через те, що при написані цієї платформи були використані Java, C++, Python – для серверної частини [10]. Amazon активно використовує ці мови для створення високопродуктивних систем, а MySQL та PostgreSQL – для деяких застосунків і аналітики [4, 10].

Для детального розуміння процесу розробки власного застосунку, необхідно розглянути функціональність сервісу Amazon. Перше та основна функціональність це широкий асортимент. Мільйони товарів, від електроніки до продуктів харчування, які можна знайти саме на цьому ресурсі. Наступне для розгяду пропонуємо – це prime-членство. Така підписка пропонує

ексклюзивні переваги, такі як швидка доставка, доступ до стрімінгових сервісів, знижки тощо. До основного можна віднести рейтинг та відгуки, які дозволяють покупцям оцінювати товари та продавців. Платформа дозволяє стороннім продавцям використовувати її інфраструктуру для продажу своїх товарів.

Після огляду функціональності розглянемо переваги та недоліки сервісу для торгівлі Amazon. До основних переваг можемо віднести глобальний масштаб, а саме можливість доступу до мільйонів покупців у всьому світі, логістичні центри Amazon забезпечують швидку доставку товарів. Також, що варте уваги – це високий рівень безпеки транзакцій і повернення товарів. Основні недоліки Amazon це великий потік продавців, який створює високий рівень конкуренції. Також платформа стягує значні збори за продаж товарів і використання своїх послуг. І найголовніше, це те, що продавці значною мірою залежать від алгоритмів Amazon, які можуть змінюватися.

eBay – це одна з найстаріших платформ електронної комерції, яка була заснована у 1995 році. Вона спеціалізується на продажу нових і вживаних товарів, а також популяризувала формат онлайн-аукціонів. Ця платформа була обрана для порівння, так як для розробки саме цього сервісу були використанні такі технології, як Java, Python, Ruby – використовуються для серверної розробки та бізнес-логіки, а PostgreSQL – використовується для зберігання даних, що потребують складних запитів і транзакцій [4, 10].

Розглянемо функціональність eBay. По-перше, користувачі можуть продавати товари як через аукціони, так і за фіксованими цінами. Сервіс має в собі велику кількість категорій, завдяки чому, на ньому можна знайти багато чого цікавого та необхідного. Платформа забезпечує підтримку у випадках шахрайства або недобросовісного виконання угоди.

Вона також надає можливість створення індивідуальних магазинів, аналітика продажів тощо.

Перейдемо до *переваг eBay*. Платформа надає можливість продавати як нові, так і вживані товари. Користувачі можуть брати участь в аукціонах або купувати товари відразу. На цій платформі відбувається забезпечення безпеки

транзакцій через програми гарантій. І найголовніше, це легкість входу для нових продавців.

Розглянемо *недоліки платформи eBay*. Більшість товарів доставляються із значними затримками, особливо при міжнародній торгівлі. Через формат аукціонів товари часто продаються за заниженими цінами. Також існують ризики, пов’язані з недобросовісними продавцями.

Після розгляду основних переваг та недоліків кожного з сервісів сформуємо таблицю 1.1, в якій буде розписано порівняльний аналіз цих платформ.

Таблиця 1.1 – Порівняльний аналіз платформ для онлайн торгівлі

Характеристика	**OLX**	**Amazon**	**eBay**
Масштаб	Локальний	Глобальний	Глобальний
Формат продажу	Прямий продаж	Прямий продаж	Аукціони та прямий продаж
Основні клієнти	Приватні особи	Приватні особи та бізнес	Приватні особи
Логістика	Відсутня	Забезпечується Amazon	Забезпечується продавцем
Комісії	Відсутні або мінімальні	Високі	Середні

Кожна з розглянутих платформ має свої переваги та недоліки, що робить їх придатними для різних сегментів ринку. OLX підходить для локальної торгівлі, Amazon є найкращим вибором для глобальної експансії, а eBay пропонує унікальний формат торгів через аукціони. Вивчення цих платформ допомагає зрозуміти ключові аспекти, які слід враховувати при створенні власного сервісу онлайн-торгівлі. У подальшій розробці слід зосередитися на забезпеченні зручності використання, безпеки транзакцій і масштабованості системи. Після огляду функціоналу та розгляду переваг з недоліками

проведемо технічний аналіз конкурентів, таких як OLX, Amazon та eBay, оскільки це дозволить оцінити особливості існуючих платформ та визначити вимоги до функціоналу. OLX орієнтується на зручний інтерфейс для локальних покупок, пропонуючи прості інструменти для публікації оголошень і здійснення транзакцій, що дозволяє інтегрувати систему пошуку та фільтрації за категоріями. Amazon має розвинену інфраструктуру для глобальних продажів з потужною системою управління складом, рекомендаційними алгоритмами та високою швидкістю обробки транзакцій, що вимагає використання надійних баз даних, таких як PostgreSQL для ефективної роботи з великими обсягами інформації. eBay надає платформу для аукціонів і купівлі-продажу товарів з різних категорій, акцентуючи увагу на безпеці платежів і зручності взаємодії між продавцями та покупцями. Розробка вебзастосунку, що конкурує з такими платформами, вимагає інтеграції сучасних технологій для масштабованості, безпеки даних та зручності користувацького досвіду.

1.2 Особливості роботи з базами даних у вебдодатках

Бази даних є ключовим елементом вебдодатків, забезпечуючи ефективне зберігання, організацію, пошук та управління даними. Їхнє використання у веброзробці охоплює широкий спектр завдань: від збереження інформації про користувачів і продукти до підтримки складних операцій, таких як обробка транзакцій чи аналітика. У цьому розділі розглянуто особливості роботи з базами даних у вебдодатках, їхній вплив на архітектуру програмного забезпечення, типи баз даних та вибір оптимальних технологій для різних задач.

Розробка вебдодатків неможлива без інтеграції з базою даних, оскільки більшість бізнес-логіки, наприклад, аутентифікація, зберігання даних про товари чи транзакції, залежить від доступу до інформації, організованої в БД. Тому пропонується розглянути основні відмінності БД, та підтвердити правильність обраної СКБД.

PostgreSQL – це об’єктно-реляційна СУБД з відкритим кодом, яка була розроблена для забезпечення високої продуктивності, надійності та підтримки складних запитів. Вона має довгу історію розвитку та користується великою популярністю серед розробників завдяки своїй стабільності та багатству функціональних можливостей.

Однією з основних переваг PostgreSQL є її здатність працювати з великими обсягами даних і забезпечувати високий рівень цілісності даних. Ця СУБД підтримує стандарт SQL, що забезпечує простоту та зручність у роботі з базою даних, при цьому надаючи ряд додаткових можливостей, які роблять її потужним інструментом для веброзробки.

Основною задачею при роботі з PostgreSQL у вебдодатку є забезпечення ефективної та безпечної взаємодії між серверною частиною додатку та базою даних. Для цього використовується спеціальний драйвер, який надає API для взаємодії з PostgreSQL.

У вебдодатку на Flask для цього зазвичай використовується бібліотека SQLAlchemy або безпосередньо psycopg2 [3, 5]. SQLAlchemy є об’єктно-реляційним маппером (ORM), що дозволяє розробникам працювати з базою даних, використовуючи об’єктно-орієнтовану парадигму програмування. Це дозволяє значно спростити роботу з даними, знижуючи ймовірність помилок і спрощуючи процес підтримки коду. Psycopg2, у свою чергу, є драйвером для роботи з PostgreSQL безпосередньо через SQL-запити [4].

При проєктуванні структури бази даних для вебдодатка важливо враховувати характер даних та логіку взаємодії з ними. Для онлайн-сервісу торгівлі база даних повинна містити кілька ключових таблиць, які зберігають інформацію про користувачів, товари, замовлення та транзакції.

Таблиця користувачів: повинна містити інформацію про зареєстрованих користувачів, їх дані для авторизації (логін, хеш пароля), контактні дані та історію замовлень.

Таблиця товарів: містить інформацію про товари, які пропонуються на платформі: назву, опис, ціну, категорії, наявність на складі, зображення та інші характеристики.

Таблиця замовлень: зберігає інформацію про замовлення, зроблені користувачами, включаючи статус замовлення, дату покупки, зв’язок з користувачем і товарами.

Таблиця транзакцій: забезпечує зберігання даних про фінансові операції, таких як плати за замовлення, способи оплати і статуси транзакцій.

Для забезпечення надійності роботи системи в PostgreSQL використовуються механізми транзакцій. Кожна транзакція гарантує, що або всі операції будуть виконані успішно, або жодна з них не буде застосована. Це дозволяє запобігти виникненню часткових оновлень бази даних, що може призвести до пошкодження даних.

Однією з важливих особливостей роботи з базами даних у вебдодатках є потреба інтеграції з іншими сервісами. У випадку онлайн-торгівлі це можуть бути платіжні системи для обробки платежів, служби доставки для відстеження статусів замовлень, а також сторонні аналітичні інструменти для аналізу даних.

PostgreSQL надає потужні можливості для інтеграції з іншими системами завдяки підтримці RESTful API та можливості працювати з JSON-даними [4, 7]. Це дозволяє ефективно зберігати та обробляти різноманітні типи даних, що надходять від сторонніх сервісів, а також здійснювати запити до зовнішніх API для отримання додаткової інформації.

Оскільки вебдодатки часто працюють з конфіденційними даними користувачів, забезпечення безпеки є важливим аспектом при роботі з базами даних. PostgreSQL має кілька вбудованих механізмів для захисту даних, розглянемо їх далі.

Шифрування даних: для захисту конфіденційних даних, таких як паролі користувачів, PostgreSQL підтримує шифрування на рівні додатку або з використанням сторонніх бібліотек.

Ролі та доступ: PostgreSQL дозволяє створювати різні ролі з обмеженнями доступу до певних таблиць або функцій. Це дає змогу розмежувати доступ до даних для різних користувачів або компонентів системи.

Аудит доступу: PostgreSQL підтримує механізми для ведення журналів доступу до бази даних, що дозволяє відслідковувати всі дії користувачів і адміністраторів.

Інтеграція з сторонніми сервісами безпеки: PostgreSQL може бути інтегровано з системами управління доступом, такими як LDAP або Kerberos, для додаткового рівня захисту.

PostgreSQL є високопродуктивною СУБД, яка підтримує великі обсяги даних та високу пропускну здатність запитів. Важливими для ефективної роботи з великими обсягами даних є наступні можливості:

- індексування: PostgreSQL підтримує різноманітні типи індексів, зокрема B-дерева, хеш-індекси, GIN (Generalized Inverted Index), що дозволяє значно прискорити виконання запитів до великих таблиць [4];
- паралельні запити: для виконання складних запитів PostgreSQL підтримує паралельне виконання, що дозволяє збільшити продуктивність при обробці великих обсягів даних;
- реплікація: для масштабування та забезпечення надійності PostgreSQL підтримує реплікацію, що дозволяє створювати копії бази даних для розподіленого доступу і резервного зберігання даних [4];
- кешування: PostgreSQL використовує вбудоване кешування, що дозволяє значно зменшити час відповіді на повторні запити.

PostgreSQL є потужною та гнучкою СУБД, яка добре підходить для використання у вебдодатках, зокрема в онлайн-торгівлі. Вона надає широкий набір інструментів для забезпечення високої продуктивності, масштабованості та безпеки, що робить її ідеальним вибором для розробки складних систем з великим обсягом даних.

Всі ці особливості дозволяють зручно інтегрувати PostgreSQL з іншими компонентами додатку, а також забезпечити безпечну і ефективну роботу сервісу для онлайн-торгівлі.

1.3 Огляд фреймворків для веброзробки

Розробка вебдодатків вимагає вибору відповідного інструменту, який дозволяє ефективно вирішувати поставлені завдання. Серед Python-фреймворків найбільш популярними є Flask, Djangota FastAPI, кожен із яких має свої особливості, переваги та недоліки [2, 3, 8].

У цьому розділі детально проаналізовано ці три фреймворки з метою визначення їх придатності для розробки різних типів вебдодатків.

1.3.1 FastAPI

FastAPI – це сучасний фреймворк для Python, створений у 2018 році. Його основна мета – забезпечити високу продуктивність і простоту розробки API. FastAPI базується на асинхронності та широко використовує типізацію Python для підвищення надійності.

Розглянемо переваги FastAPI, а саме: FastAPI використовує асинхронний механізм, що дозволяє обробляти велику кількість запитів одночасно [8]. Фреймворк автоматично генерує документацію API у форматі OpenAPI/Swagger.

FastAPI використовує Python-типізацію для валідації вхідних даних, що зменшує кількість помилок. Завдяки асинхронній архітектурі FastAPI підходить для розробки мікросервісів та API.Як і Flask, FastAPI є легким, але забезпечує більше функцій із коробки.

Перейдемо до розглядання недоліків FastAPI: Через молодість FastAPI ще не має такої великої спільноти, як Flask чи Django [2, 8]. Хоча FastAPI добре підходить для API, створення повноцінного вебдодатка може вимагати додаткових бібліотек.

Асинхронність вимагає від розробників розуміння концепцій, що може бути складним для початківців.

1.3.2 Django

Django – це потужний фреймворк для Python, створений у 2005 році. Його основна філософія – “Batteries included” (усе необхідне вже включено). Django надає широкий спектр інструментів для швидкої розробки складних вебдодатків. До переваг Django віднесемо повноцінний функціонал. Це означає, що Django включає ORM, систему шаблонів, аутентифікацію, засоби роботи з формами, адміністраторську панель і багато іншого. Також Django ідеально підходить для створення великих вебдодатків і платформ, наприклад, маркетплейсів, соціальних мереж тощо.Окрім цього, Django має вбудовані інструменти захисту від CSRF-атак, SQL-ін’єкцій, XSS-атак та інших загроз. Завдяки модульній структурі можна легко масштабувати додаток, додаючи нові компоненти. Django має велику спільноту розробників і добре документований функціонал [2].

До недоліків Django віднесемо наступне: через жорстку структуру Django може бути складно налаштувати під специфічні потреби, початківцям може бути складно освоїти всі аспекти фреймворку, зайва складність для простих додатків, також Django може бути надмірним для невеликих проєктів [2].

1.3.3 Flask

Flask – це легкий фреймворк, який характеризується мінімалізмом і гнучкістю. Він був створений у 2010 році і позиціонується як мікрофреймворк для Python [10]. Flask дозволяє розробникам самостійно обирати компоненти, які будуть використані в додатку, надаючи лише базові інструменти [3].

Розглянемо переваги Flask. Найголовніше – це гнучкість. Flask не нав’язує строгих структур і дозволяє розробникам будувати архітектуру додатка відповідно до їхніх потреб [3]. Базовий функціонал включає лише

обробку HTTP-запитів та маршрутизацію, що зменшує складність роботи з фреймворком. Flask має безліч розширень, таких як Flask-SQLAlchemy для роботи з базами даних, Flask-RESTful для створення API тощо [5, 7]. Також Flask ідеальний для стартапів, прототипів та невеликих додатків.

Як і кожен з фреймворків він має певні недоліки, наприклад, Flask не включає інструменти для аутентифікації, адміністративних панелей чи ORM, що вимагає додаткової роботи з інтеграцією. Також у ньому відсутні строгі стандарти, а це, в свою чергу може призводити до хаотичної архітектури в командних проєктах.

Для великих проєктів інтеграція додаткових інструментів може бути трудомісткою.

Тому розглянувши переваги, недоліки та функціональні особливості, через простоту використання та відсутність зайвих функціональних аспектів пропонується обрати цей фреймворк для написання нашого проєкту.

Для наочного порівняння ключових характеристик трьох фреймворків подано таблицю 1.2.

Таблиця 1.2 – Порівняння ключових параметрів фреймворків

Характеристика	**Flask**	**Django**	**FastAPI**
Рік створення	2010	2005	2018
Гнучкість	Висока	Низька	Висока
Продуктивність	Середня	Середня	Висока
Придатність для API	Обмежена	Обмежена	Відмінна
Повнота функціоналу	Мінімальна	Повна	Середня
Документація API	Відсутня	Відсутня	Генерується
Безпека	Ручна конфігурація	Вбудована	Ручна конфігурація
Підтримка мікросервісів	Обмежена	Обмежена	Відмінна

Вибір фреймворку потрібно робити залежно від проєкту, розглянемо кожен з фреймворків для кожного з проєктів: Наприклад, для невеликих додатків Flask буде найкращим вибором завдяки своїй гнучкості та легкості. Якщо проєкт буде великим, то Django ідеально підходить через наявність багатого функціоналу та інструментів для масштабування. FastAPI є оптимальним вибором через його продуктивність, асинхронність та автоматичну генерацію документації.

У результаті порівняння фреймворків Flask, Django та FastAPI, можна зробити наступні висновки, що Flask є чудовим вибором для малих та середніх проєктів, де важлива гнучкість і простота використання. В свою чергу Django підходить для великих проєктів, які потребують вбудованих компонентів для швидкої розробки складних вебдодатків [2]. А FastAPI – це фреймворк, орієнтований на високопродуктивні API з асинхронною обробкою запитів, що є ідеальним варіантом для роботи з великими обсягами даних та високими вимогами до продуктивності [8].

Вибір фреймворку залежить від специфікацій проєкту, рівня вимог до продуктивності, безпеки та складності додатка. Тому, в контексті теми роботи правильним рішенням буде обрати фреймворк Flask [3].

1.4 Постановка задачі для створення власного сервісу

Розробка вебсервісу для онлайн-торгівлі є важливим етапом для створення платформи, яка дозволить користувачам купувати та продавати товари, а також забезпечить ефективну взаємодію між продавцями і покупцями. Вибір правильних технологій, архітектури та інструментів для створення такого сервісу є ключовим аспектом для досягнення високої продуктивності, безпеки та зручності використання. У цьому розділі буде детально описано постановку задачі для розробки власного сервісу для онлайн-торгівлі, враховуючи усі етапи: від визначення цілей і вимог до реалізації та тестування.

З розвитком технологій та глобалізацією інтернет-торгівля стала однією з основних форм комерційної діяльності. Створення онлайн-торговельної платформи дозволяє користувачам купувати та продавати товари без географічних обмежень. Однак, незважаючи на широкий спектр існуючих платформ (OLX, Amazon, eBay тощо), існують певні аспекти, які можна вдосконалити. Власний сервіс дозволить краще відповідати потребам цільової аудиторії, зокрема через інтеграцію новітніх технологій та зручний інтерфейс для користувачів.

Метою цього проєкту є розробка сервісу для онлайн-торгівлі, який забезпечить: створення та управління профілями користувачів (продавців та покупців), можливість публікації товарів для продажу, описів і фото, безпечні транзакції з підтримкою електронних платежів, реалізацію ефективної пошукової системи та фільтрів для товарів. Надання інструментів для оцінки та відгуків про товари та продавців, забезпечення захисту даних користувачів та безпеки операцій.

Цей вебсервіс буде орієнтований на середній та малий бізнес, що прагне розширити свою аудиторію та продавати товари через онлайн-платформи.

Для реалізації сервісу буде обрано наступне технологічне середовище: Flask буде використовуватися як основний фреймворк для побудови сервера. Flask вибраний через свою легкість, гнучкість і підтримку великої кількості розширень, таких як інтеграція з базами даних (через SQLAlchemy), підтримка JWT для аутентифікації, а також здатність працювати з REST API [3,5]. PostgreSQL буде використовуватися як основна реляційна база даних. Вона забезпечує високу цілісність даних, підтримку транзакцій та хорошу продуктивність при високих навантаженнях [4].

Для захисту даних користувачів і здійснення транзакцій буде використовуватися стандарт HTTPS для шифрування з’єднань, а також інструменти для захисту від атак, таких як SQL-ін’єкції та CSRF.

Процес розробки сервісу можна поділити на кілька етапів.

Перший етап – планування та дизайн. На цьому етапі ми формулюємо вимоги до системи, розробляємо технічне завдання та архітектурне рішення та створюємо прототип інтерфейсу користувача.

Другий етап – це розробка серверної частини. На цьому етапі буде відбуватись процес налаштування серверне оточення, реалізація моделі даних та API для управління товарами, користувачами, транзакціями та інтегруємо платіжні системи.

Третій етап – розробка фронтенд частини. На цьому етапі створюється інтерактивний інтерфейсу на основі шаблонізатора та інтегруемо з сервером для взаємодії через API. Передостанній етап – це тестування. Для цього буде організовано проведення юніт-тестування для перевірки функціональності системи. Окрім цього потрібне модульне тестування API та тестування на навантаження для забезпечення продуктивності. Останній етап – розгортання та підтримка. Для цього потрібно виконати розгортання сервісу на сервері (AWS, DigitalOcean) та виконувати постійний моніторинг системи та оновлення.

Постановка задачі для створення власного сервісу для онлайн-торгівлі передбачає комплексний підхід до вибору технологій, проєктування архітектури та реалізації ключових функцій платформи. Це дозволить створити ефективну, безпечну та зручну для користувачів систему для онлайн-продажів, яка буде відповідати потребам сучасного ринку електронної комерції.

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ВЕБДОДАТКУ

2.1 Аналіз вимог до сервісу онлайн-торгівлі

Сервіс онлайн-торгівлі є важливим елементом сучасної економіки. Зростання популярності інтернет-магазинів обумовлене зручністю, доступністю та швидкістю обслуговування. Розробка такого сервісу вимагає чіткого аналізу функціональних та нефункціональних вимог, що забезпечують ефективність, безпеку та масштабованість системи.

У цьому розділі детально розглянуто основні вимоги до сервісу онлайн-торгівлі з урахуванням сучасних тенденцій у веброзробці.

Система повинна бути побудована на основі архітектури клієнт-сервер з поділом на фронтенд і бекенд. Flask слугуватиме бекенд-фреймворком, а PostgreSQL – основною системою управління базами даних (СУБД) [3, 4]. База даних повинна бути нормалізована до таблиць третьої нормальної форми для уникнення надмірності даних [7].

Зберігання структурованих даних, наприклад товари, користувачі, замовлення, тощо та журналу змін. Також додаток повинен бути інтегруємим. Використання RESTful API для взаємодії між фронтендом і бекендом вирішить цю проблему [7].

Наступним кроком при розгляданні архітектури потрібно пригадати обраний технологічний стек. Flask обраний як бекенд-фреймворк завдяки його простоті, розширюваності та активній спільноті. Щодо СКБД обрана була PostgreSQL для зберігання реляційних даних через його потужність у роботі з транзакціями та розширені можливості масштабування [4].

Аналіз вимог до сервісу онлайн-торгівлі дозволяє визначити основні характеристики, які забезпечать його конкурентоспроможність та ефективність.

Розглянемо функціональні та нефункціональні вимоги вебзастосунку.

До функціональних вимог застосунку віднесемо наступні пункти:

а) **реєстрація та аутентифікація користувачів:**
   - реалізація можливості реєстрації користувачів як продавців або покупців,
   - підтримка аутентифікації через логін і пароль, а також через соціальні мережі (Facebook, Google тощо),
   - можливість відновлення пароля через email або SMS;

б) **управління профілями користувачів:**
   - створення профілю продавця з можливістю додавання контактної інформації та опису компанії,
   - можливість для покупців переглядати профілі продавців, їхні рейтинги та відгуки;

в) **додавання та управління товарами:**
   - можливість продавців додавати нові товари з описами, цінами, категоріями та фотографіями,
   - можливість редагування та видалення товарів з особистого кабінету продавця;

г) **пошук і фільтрація товарів:**
   - пошукова система, яка дозволяє покупцям знаходити товари за категоріями, ціною, брендом, рейтингом тощо,
   - можливість застосовувати фільтри для полегшення пошуку;

д) **система рейтингів і відгуків:**
   - можливість для покупців залишати відгуки та ставити оцінки товарам і продавцям,
   - система зворотного зв’язку для забезпечення взаємної довіри;

е) **адміністрування та моніторинг:**
   - панель адміністратора для моніторингу товарів, користувачів і транзакцій,
   - інструменти для модерації контенту, управління порушеннями правил та контролю за процесом транзакцій.

Розглянемо нефункціональні вимоги. Ці вимоги будуть подані у вигляді списку нижче:

а) **продуктивність:** вебсервіс повинен підтримувати великі навантаження, забезпечуючи безперервну роботу навіть при високій кількості одночасних користувачів;

б) **безпека:**
   – зашифровані з’єднання через HTTPS,
   – захист від SQL-ін’єкцій та CSRF-атак,
   – надійне зберігання паролів (наприклад, через хешування за допомогою bcrypt) [12];

в) **масштабованість:** архітектура сервісу повинна передбачати можливість масштабування в разі збільшення навантаження;

г) **інтерфейс користувача:**
   – інтуїтивно зрозумілий та простий інтерфейс, що дозволяє користувачам легко орієнтуватися в сервісі,
   – адаптивний дизайн для коректного відображення на різних пристроях (мобільні телефони, планшети, десктопи);

д) **висока доступність:** система повинна бути доступною 24/7, з мінімальними перервами на обслуговування;

е) **логування та моніторинг:** вебсервіс повинен включати в себе механізми для збору логів, що дозволяють відслідковувати помилки та збої системи.

Врахування функціональних і нефункціональних вимог, а також вибір Flask і PostgreSQL як основи розробки, гарантує надійність, продуктивність і масштабованість системи [3, 4]. Цей аналіз слугуватиме базою для побудови архітектури, проєктування бази даних та вибору відповідних інструментів для реалізації сервісу.

2.2 Проєктування архітектури додатку

Проєктування архітектури вебдодатку є ключовим етапом розробки, що визначає його продуктивність, масштабованість, надійність і зручність використання. У випадку створення сервісу для онлайн-торгівлі важливо забезпечити ефективну взаємодію між компонентами системи, оптимальне управління даними та підтримку інтеграцій з платіжними системами й іншими сервісами. У цьому розділі описано основні принципи проєктування, обрану архітектуру додатку, а також використані технології та їх взаємодію.

Для побудови сервісу онлайн-торгівлі обрано багаторівневу (n-tier) архітектуру, яка передбачає поділ на три рівні.

Рівень презентації (Frontend). Відповідає за взаємодію з користувачем через вебінтерфейс. Фронтенд реалізований на основі HTML, CSS та JavaScript із використанням фреймворку, наприклад, React або Vue.js, що забезпечує динамічність і зручність користування.

Рівень бізнес-логіки (Backend). Реалізований за допомогою Flask, він відповідає за обробку запитів від клієнта, виконання бізнес-логіки та взаємодію з базою даних.

Рівень зберігання даних (Database). Для управління даними використовується PostgreSQL. Цей рівень забезпечує зберігання інформації про користувачів, товари, замовлення та інші важливі дані.

Обраний підхід дозволяє розділити функції додатку, що спрощує його розробку, тестування, підтримку та масштабування.

Архітектура сервісу складається з таких основних компонентів:

а) **Frontend:**
   - відповідає за відображення інформації користувачам,
   - спілкується з бекендом через RESTful API [7];

б) **Backend:**
   - розроблений на Flask, цей шар містить маршрути (endpoints) для обробки HTTP-запитів,

- інкапсулює бізнес-логіку, включаючи управління товарами, кошиком і замовленнями,
- забезпечує аутентифікацію та авторизацію;

в) **Database:**

- PostgreSQL використовується для зберігання структурованих даних [4],
- включає таблиці для користувачів, товарів, замовлень, платежів тощо.

Архітектура передбачає використання RESTful API для зв'язку між фронтендом і бекендом. Основні принципи:

а) **Методи HTTP:**

- GET: отримання даних (наприклад, список товарів),
- POST: створення нових ресурсів (наприклад, замовлення),
- PUT/PATCH: оновлення ресурсів (наприклад, оновлення профілю користувача),
- DELETE: видалення ресурсів (наприклад, товарів із кошика);

б) **Структура URL:**

- /api/v1/users: робота з користувачами,
- /api/v1/products: управління товарами,
- /api/v1/orders: замовлення;

в) **Формат даних**: JSON використовується як стандартний формат для передачі даних між компонентами.

Для реалізації бази даних використовується PostgreSQL. Основні таблиці:

а) **Users:**

- id: первинний ключ,
- email: унікальний ідентифікатор,
- password: хешований пароль,
- role: роль користувача (покупець, адміністратор);

б) **Products:**

- id: первинний ключ,
- name: назва товару,
- description: опис,
- price: ціна,
- stock: кількість на складі;

в) **Orders:**

- id: первинний ключ,
- user_id: зовнішній ключ на таблицю користувачів,
- total_price: загальна сума замовлення,
- status: статус замовлення;

г) **Order_Items:**

- order_id: зовнішній ключ на таблицю замовлень,
- product_id: зовнішній ключ на таблицю товарів,
- quantity: кількість товару в замовленні;

д) **Payments:**

- id: первинний ключ,
- order_id: зовнішній ключ на таблицю замовлень,
- status: статус оплати,
- method: спосіб оплати.

Більш детальна схема таблиць у базі даних буде розглянута згодом в наступних пунктах цього розділу.

Після огляду будови бази даних, пропонується оглянути обраний фреймворк. Flask є мікрофреймворком, який дозволяє створювати легкі, але потужні вебдодатки [3]. Його переваги полягають у тому, що Flask не нав’язує жорстких стандартів, що дозволяє розробникам вільно організовувати код [3].

Підтримка розширень, таких як Flask-SQLAlchemy для роботи з базами даних і Flask-JWT-Extended для авторизації [5, 11]. Flask легко інтегрується з зовнішніми сервісами, що робить його ідеальним для RESTful API [3, 7].

Щоб сервіс міг обробляти зростаючу кількість користувачів і транзакцій, запроваджено такі рішення:

а) **модульність додатку**: логіка розділена на незалежні модулі (наприклад, модуль управління товарами, модуль замовлень);

б) **кешування:** використання Redis для кешування часто запитуваних даних (наприклад, популярних товарів);

в) **горизонтальне масштабування**: розгортання декількох екземплярів бекенду за допомогою контейнерів Docker і балансувальника навантаження;

г) **оптимізація бази даних**: індексація ключових полів, використання розділення (sharding) і реплікації.

Проєктування архітектури сервісу онлайн-торгівлі базується на принципах модульності, масштабованості та безпеки. Використання Flask як бекенду й PostgreSQL для управління даними забезпечує простоту інтеграції та високу продуктивність [4].

Архітектура системи дозволяє легко адаптувати сервіс до змін у вимогах, забезпечуючи зручність як для кінцевих користувачів, так і для адміністраторів. Розглянемо архітектуру за допомоги діаграми компонентів (рис. 2.1).

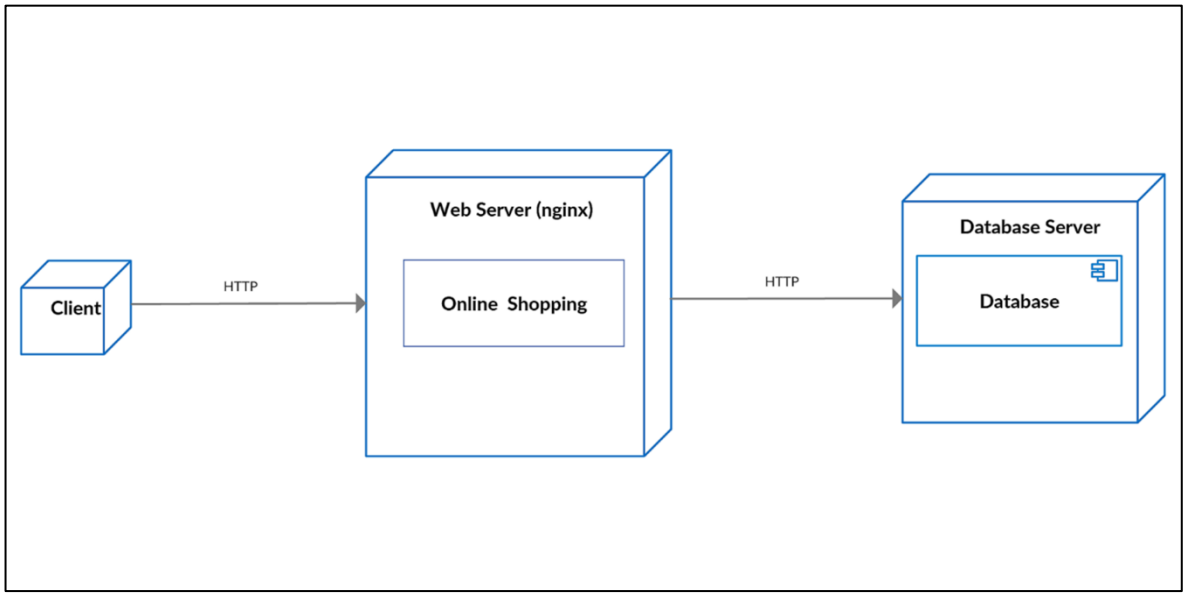


Рисунок 2.1 – Діаграма компонентів застосунку для онлайн торгівлі

2.3 Структура бази даних для сервісу (PostgreSQL)

Система управління базами даних (СУБД) є фундаментальною складовою будь-якого сучасного вебдодатку, особливо у сфері онлайн-торгівлі, де потрібно зберігати великі обсяги інформації, забезпечувати її цілісність і доступність. Для розробки сервісу онлайн-торгівлі було обрано PostgreSQL – потужну реляційну базу даних із відкритим кодом [4]. У цьому розділі розглядається структура бази даних, її основні таблиці, зв’язки між ними, використання нормалізації та оптимізаційні підходи.

Основні вимоги до структури бази даних та таблиць

а) **розділення даних за категоріями**: дані користувачів, товарів, замовлень, платежів та інших сутностей зберігаються в окремих таблицях;

б) **реалізація зв’язків**: таблиці пов’язані через зовнішні ключі для забезпечення цілісності даних;

в) **масштабованість**: дизайн дозволяє розширювати базу даних без значних змін у структурі.

Таблиця Users зберігає інформацію про зареєстрованих користувачів. Розглянемо цю таблицю більш детально, ознайомившись із таблицею 2.1.

Таблиця 2.1 – Опис таблиці Users

Поле	**Тип даних**	**Опис**
id	SERIAL PRIMARY KEY	Унікальний ідентифікатор користувача
email	VARCHAR(255) UNIQUE NOT NULL	Електронна адреса
password	VARCHAR(255) NOT NULL	Хешований пароль
role	VARCHAR(50) DEFAULT 'customer'	Роль користувача (покупець/адміністратор)
created_at	TIMESTAMP DEFAULT CURRENT_TIMESTAMP	Дата створення запису

Таблиця Products зберігає інформацію про товари. Розглянемо цю таблицю більш детально, ознайомившись із таблицею 2.2.

Таблиця 2.2 – Опис таблиці Products

Поле	**Тип даних**	**Опис**
id	SERIAL PRIMARY KEY	Унікальний ідентифікатор товару
name	VARCHAR(255) NOT NULL	Назва товару
description	TEXT	Опис товару
price	DECIMAL(10, 2) NOT NULL	Ціна товару
stock	INTEGER DEFAULT 0	Кількість товарів на складі
created_at	TIMESTAMP DEFAULT CURRENT_TIMESTAMP	Дата створення запису

Таблиця Orders зберігає інформацію про замовлення. Розглянемо цю таблицю більш детально, ознайомившись із таблицею 2.3.

Таблиця 2.3 – Опис таблиці Orders

Поле	**Тип даних**	**Опис**
id	SERIAL PRIMARY KEY	Унікальний ідентифікатор замовлення
user_id	INTEGER NOT NULL REFERENCES Users(id) ON DELETE CASCADE	Покупець (зовнішній ключ)
total_price	DECIMAL(10, 2)	Загальна сума замовлення
status	VARCHAR(50) DEFAULT 'pending'	Статус замовлення
created_at	TIMESTAMP DEFAULT CURRENT_TIMESTAMP	Дата створення замовлення

Таблиця Order_Items зберігає інформацію про товари у замовленні. Розглянемо цю таблицю більш детально, ознайомившись із таблицею 2.4.

Таблиця 2.4 – Опис таблиці Order_Items

Поле	**Тип даних**	**Опис**
id	SERIAL PRIMARY KEY	Унікальний ідентифікатор запису
order_id	INTEGER NOT NULL REFERENCES Orders(id) ON DELETE CASCADE	Замовлення (зовнішній ключ)
product_id	INTEGER NOT NULL REFERENCES Products(id)	Товар (зовнішній ключ)
quantity	INTEGER NOT NULL	Кількість товарів у замовленні
subtotal	DECIMAL(10, 2)	Підсумкова сума за товар

Таблиця Payments зберігає інформацію про оплати. Розглянемо цю таблицю більш детально, ознайомившись із таблицею 2.5.

Таблиця 2.5 – Опис таблиці Payments

Поле	**Тип даних**	**Опис**
id	SERIAL PRIMARY KEY	Унікальний ідентифікатор платежу
order_id	INTEGER NOT NULL REFERENCES Orders(id) ON DELETE CASCADE	Замовлення (зовнішній ключ)
status	VARCHAR(50)	Статус платежу (успішний/неуспішний)
method	VARCHAR(50)	Спосіб оплати

Продовження таблиці 2.5

Поле	**Тип даних**	**Опис**
amount	DECIMAL(10, 2)	Сума платежу
created_at	TIMESTAMP DEFAULT CURRENT_TIMESTAMP	Дата платежу

Система базується на таких типах зв'язків:

а) **один-до-багатьох (1:N):**
   – один користувач може мати багато замовлень (Users → Orders),
   – одне замовлення може містити багато товарів (Orders → Order_Items);

б) **багато-до-багатьох (M:N):**
   – реалізовано через таблицю Order_Items, що зв'язує замовлення і товари.

Після опису декількох таблиць та принципу з'єднань таблиць, пропонується розглянути схему бази даних для створюємого додатку (див. рис. 2.2).

Для швидкої роботи бази даних, були виконанні такі дії з оптимізації бази даних:

– **індексація:** індексація полів, які часто використовуються у фільтрах (email, name);
– **кешування:** використання Redis для кешування популярних запитів, таких як список товарів;
– **агреговані дані:** збереження агрегованих даних, наприклад, total_price у таблиці Orders;
– **шардинг:** поділ таблиць на сегменти (наприклад, за регіоном або типом товарів).

Структура бази даних, спроєктована для сервісу онлайн-торгівлі, базується на принципах реляційного моделювання й нормалізації. PostgreSQL забезпечує надійність, масштабованість і високі показники продуктивності, що дозволяє обробляти великі обсяги транзакцій та даних. Проєктування бази

враховує зручність розширення системи, що є важливим для підтримки змінних вимог бізнесу.

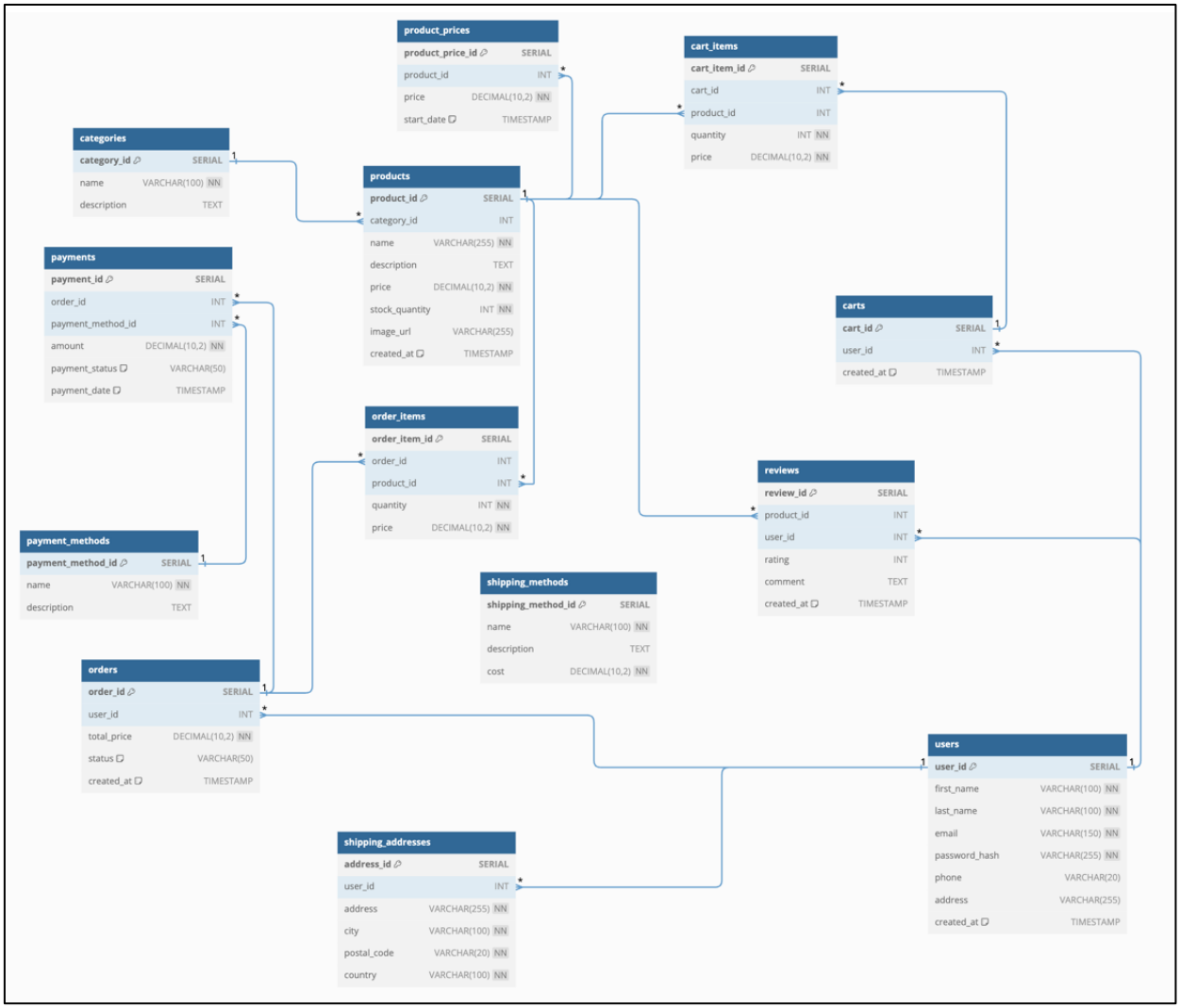


Рисунок 2.2 – Схема бази даних у додатку

2.4 Проєктування ключових модулів додатку

У процесі розробки сервісу для онлайн-торгівлі важливо спроєктувати модулі, які виконують основні функції системи. У цьому розділі буде детально описано проєктування трьох ключових модулів: модуля реєстрації та авторизації, модуля управління товарами та модуля опрацювання замовлень. Кожен модуль розглядається з точки зору його функціональності, структури й взаємодії з іншими компонентами системи.

2.4.1 Модуль реєстрації та авторизації

Опишемо модуль реєстрації та авторизації. Модуль реєстрації та авторизації відповідає за керування доступом до системи, забезпечує ідентифікацію користувачів і захист їхніх даних. Його функції включають реєстрацію нових користувачів, вхід до системи, скидання паролів і керування ролями (наприклад, покупець або адміністратор).

Основна функціональність цього модуля буде полягати в:

а) **реєстрації нових користувачів:**
   – створення нового облікового запису із зазначенням електронної адреси та пароля,
   – хешування пароля перед збереженням у базу даних;

б) **авторизація:**
   – перевірка облікових даних,
   – генерація токена доступу (JWT) для автентифікованих користувачів;

в) **ролі та права доступу:**
   – розмежування функціоналу для звичайних користувачів і адміністраторів;

г) **захист даних:**
   – використання HTTPS для шифрування трафіку,
   – захист від атак типу «перебір паролів» через обмеження кількості спроб входу.

Розглянемо модуль з точки зору проєктування структури.

Маршрути (Endpoints):

– POST /register: реєстрація нового користувача;
– POST /login: вхід до системи;
– POST /logout: вихід користувача;
– POST /reset-password: запит на скидання пароля.

Для цього модуля використовується таблиця Users, а алгоритм роботи цього модуля розглянемо на діаграмі послідовності (рис. 2.3).

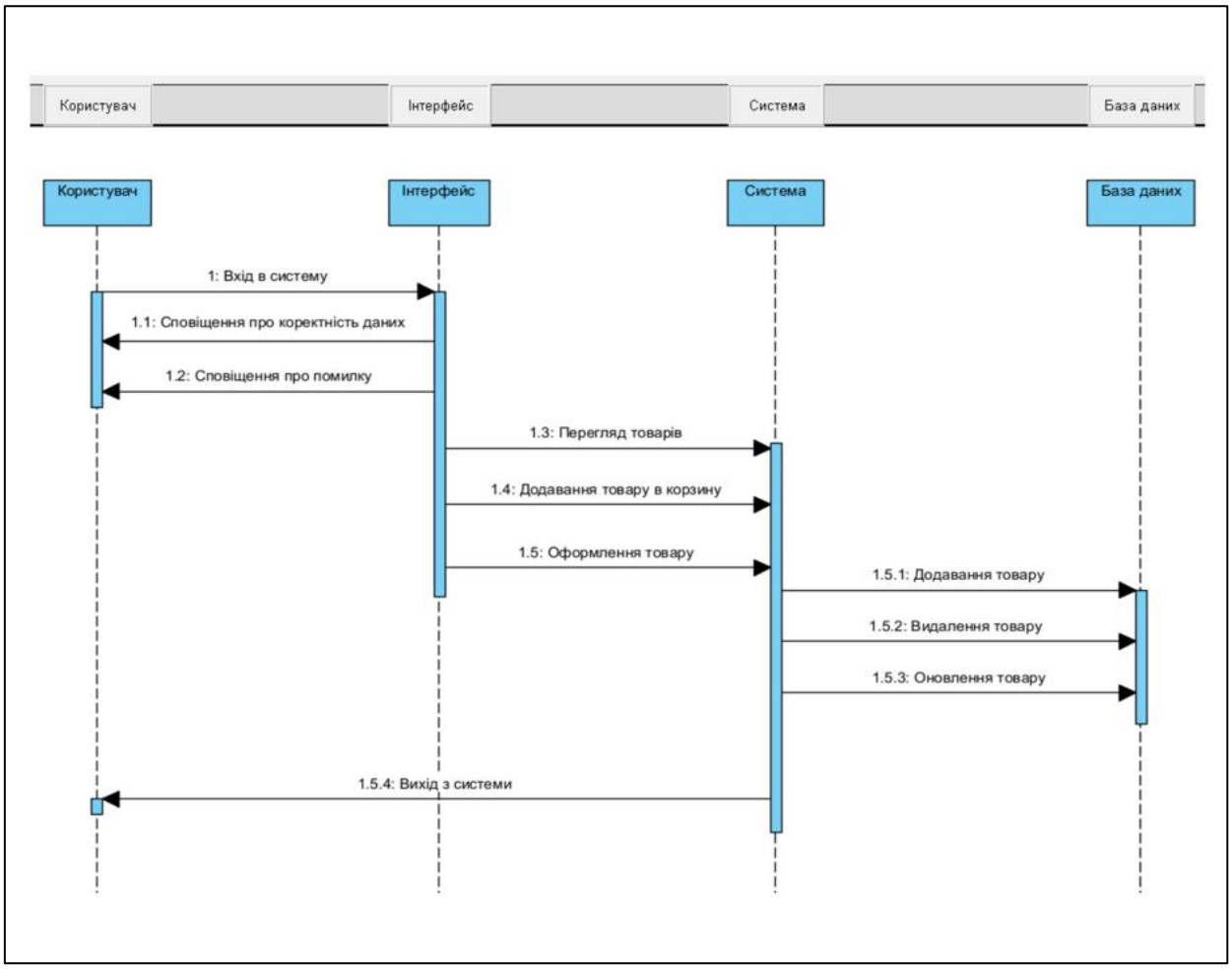


Рисунок 2.3 – Діаграма послідовності вебзастосунку для онлайн торгівлі

Для написання цього модуля були використані такі технології як: Flask-Bcrypt – для хешування паролів. Flask-JWT-Extended – для генерації та перевірки JWT-токенів та Flask-WTF – для валідації форм [1].

2.4.2 Модуль управління товарами

Перейдемо до модуля управління товарами. Опишемо його та розглянемо його більш детально. Модуль управління товарами відповідає за створення, оновлення, видалення та відображення товарів, які доступні для покупки. Основний акцент робиться на зручності роботи для адміністратора та точному відображенні інформації для користувача (рис. 2.3).

Основна функціональність модуля полягає в:

а) **додавання нових товарів:**
   – введення основної інформації про товар (назва, опис, ціна, кількість),
   – завантаження зображень;

б) **редагування товарів:** зміна інформації про існуючі товари (наприклад, оновлення ціни);

в) **видалення товарів:** видалення товарів, які більше не актуальні;

г) **перегляд каталогу:** відображення товарів для кінцевих користувачів із можливістю фільтрації та сортування.

Розглянемо модуль з точки зору проєктування структури.

Маршрути (Endpoints):

– POST /products: додавання нового товару (тільки для адміністраторів);
– GET /products: отримання списку товарів;
– PUT /products/<id>: редагування товару за ідентифікатором;
– DELETE /products/<id>: видалення товару.

Для написання цього модуля були використані такі технології як: Flask-SQLAlchemy – для ORM і взаємодії з базою даних [6]. Flask-Uploads – для завантаження зображень та Flask-RESTful – для побудови REST API [7].

Ключові модулі додатку забезпечують основну функціональність сервісу онлайн-торгівлі. Модуль реєстрації та авторизації гарантує безпеку доступу до системи, модуль управління товарами дозволяє ефективно керувати асортиментом, а модуль опрацювання замовлень забезпечує зручність і надійність процесу купівлі для користувачів. Завдяки використанню Flask та PostgreSQL кожен модуль розроблений із врахуванням масштабованості, безпеки й продуктивності.

3 РЕАЛІЗАЦІЯ ФУНКЦІОНАЛЬНИХ МОДУЛІВ ВЕБДОДАТКУ ТА ТЕСТУВАННЯ ДОДАТКУ

3.1 Реалізація серверної частини додатку за допомогою Flask

Реалізація серверної частини додатку – це один із найважливіших етапів створення сервісу для онлайн-торгівлі. Вона забезпечує обробку запитів користувачів, управління бізнес-логікою, взаємодію з базою даних і відправку відповідей на клієнтську частину. Для реалізації серверної частини обрано мікрофреймворк Flask завдяки його простоті, гнучкості та широким можливостям інтеграції з іншими інструментами. У цьому розділі детально описується процес реалізації серверної частини додатку, структура проєкту, основні компоненти та технології.

Розглянемо архітектуру серверної частини. Проєкт реалізовано із застосуванням підходу **“Blueprints”**, що дозволяє розділити функціонал на окремі модулі. Завдяки їєрархії пропонується розглянути структуру проєкту (рис. 3.1).

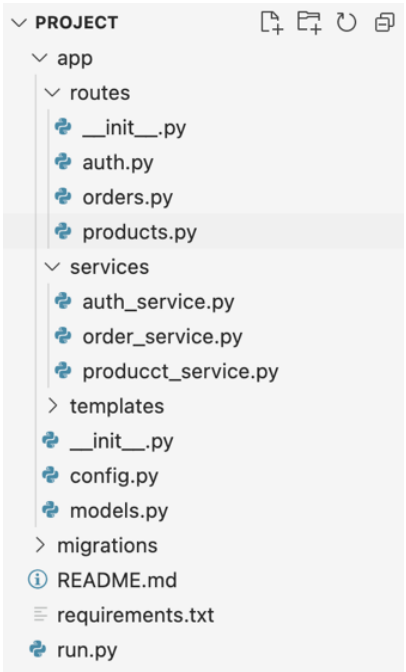


Рисунок 3.1 – Їєрархія проєкту онлайн торгівлі

Для ініціалізації додатку потрібно знайти файл __init__.py у директорії app/ містить код для створення екземпляра Flask-додатку, підключення бази даних та реєстрації маршрутів (рис. 3.2).

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
from flask_migrate import Migrate
from flask_jwt_extended import JWTManager
db = SQLAlchemy()
migrate = Migrate()
jwt = JWTManager()
def create_app(config_class):
    app = Flask(__name__)
    app.config.from_object(config_class)
    db.init_app(app)
    migrate.init_app(app, db)
    jwt.init_app(app)
    from app.routes import auth, products, orders
    app.register_blueprint(auth.bp)
    app.register_blueprint(products.bp)
    app.register_blueprint(orders.bp)
    return app
```

Рисунок 3.2 – Ініціалізація додатку через файл __init__.py

Усі налаштування, включаючи параметри бази даних, секретні ключі, налаштування токенів, зберігаються в окремому файлі, це зображено на рисунку 3.3.

```
class Config:
    SECRET_KEY = 'your-secret-key'
    SQLALCHEMY_DATABASE_URI =
'postgresql://user:password@localhost:5432/online_shop'
    SQLALCHEMY_TRACK_MODIFICATIONS = False
    JWT_SECRET_KEY = 'your-jwt-secret'
```

Рисунок 3.3 – Налаштування додатку у файлі config.py

Моделі визначають структуру даних, які зберігаються в PostgreSQL. Вони реалізовані у файлі models.py за допомогою Flask-SQLAlchemy (див. рис. 3.4).

```
from app import db
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(255), unique=True, nullable=False)
    password = db.Column(db.String(255), nullable=False)
    role = db.Column(db.String(50), default='customer')
    created_at = db.Column(db.DateTime, server_default=db.func.now())
class Product(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(255), nullable=False)
    description = db.Column(db.Text)
    price = db.Column(db.Numeric(10, 2), nullable=False)
    stock = db.Column(db.Integer, default=0)
class Order(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
    total_price = db.Column(db.Numeric(10, 2))
    status = db.Column(db.String(50), default='pending')
    created_at = db.Column(db.DateTime, server_default=db.func.now())
```

Рисунок 3.4 – Опис моделй додатку

Кожен модуль додатку реалізований як окремий Blueprint. Розглянемо файл auth.py на рисунку 3.5.

Цей файл є частиною бекенд-логіки для аутентифікації користувачів в вебдодатку, розробленому за допомогою Flask [3]. Він використовує Flask Blueprint для організації маршруту аутентифікації, а також бібліотеки Flask-Bcrypt для хешування паролів та Flask-JWT-Extended для створення та верифікації JSON Web Tokens (JWT), що використовуються для автентифікації користувачів [11–13]. Цей файл забезпечує механізм аутентифікації користувачів в додатку. Реєстрація нових користувачів та хешування паролів для збереження їх у безпечному вигляді. Авторизація через генерацію JWT

токенів, які дозволяють аутентифікованим користувачам отримувати доступ до захищених ресурсів. Таким чином, цей код є важливою частиною безпеки додатку, забезпечуючи зберігання паролів у зашифрованому вигляді та можливість автентифікації користувачів через токени.

```
from flask import Blueprint, request, jsonify
from app.models import User
from app import db
from flask_bcrypt import Bcrypt
from flask_jwt_extended import create_access_token
bp = Blueprint('auth', __name__, url_prefix='/auth')
bcrypt = Bcrypt()
@bp.route('/register', methods=['POST'])
def register():
    data = request.get_json()
    hashed_password = bcrypt.generate_password_hash(data['password']).decode('utf-8')
    new_user = User(email=data['email'], password=hashed_password)
    db.session.add(new_user)
    db.session.commit()
    return jsonify({'message': 'User registered successfully'}), 201
@bp.route('/login', methods=['POST'])
def login():
    data = request.get_json()
    user = User.query.filter_by(email=data['email']).first()
    if user and bcrypt.check_p1ssword_hash(user.password, data['password']):
        token = create_access_token(identity=user.id)
        return jsonify({'access_token': token})
    return jsonify({'message': 'Invalid credentials'}), 401
```

Рисунок 3.5 – Авторизація користувачів

Розгялнемо файл products.py на рисунку 3.6.

```
@bp.route('/products', methods=['GET'])
def get_products():
    products = Product.query.all()
    return jsonify([{'id': p.id, 'name': p.name, 'price': float(p.price)} for p in products])
```

Рисунок 3.6 – Вміст файлу products.py

Розгялнемо файл orders.py на рисунку 3.7.

```
@bp.route('/orders', methods=['POST'])
def create_order():
    data = request.get_json()
    # Логіка створення замовлення
    return jsonify({'message': 'Order created successfully'}), 201
```

Рисунок 3.7 – Вміст файлу orders.py

Цей файл містить частину бекенд-логіки для обробки замовлень і продуктів в інтернет-магазині, реалізовану за допомогою Flask [3]. Файл описує маршрути для отримання списку продуктів та створення нових замовлень, використовуючи Flask Blueprint для організації маршрутів [13]. Цей файл є частиною функціональності інтернет-магазину, що забезпечує отримання списку доступних продуктів для користувачів та створення нових замовлень через інтерфейс API. Це забезпечує основні операції для інтернет-магазину – перегляд товарів і створення замовлень. Подальше удосконалення включає додавання обробки замовлення в базі даних та валідацію отриманих даних.

Реалізація серверної частини додатку за допомогою Flask забезпечує просту й ефективну організацію бізнес-логіки та взаємодії з базою даних. Використання модульної структури дозволяє легко масштабувати додаток, розширювати його функціональність і підтримувати код. Flask-SQLAlchemy, Flask-JWT-Extended та інші розширення значно полегшують розробку та інтеграцію серверної частини з іншими компонентами системи.

3.2 Інтеграція бази даних PostgreSQL із додатком

Інтеграція бази даних є ключовим етапом у розробці вебдодатку, оскільки саме база даних відповідає за збереження, обробку та доступ до даних. Для реалізації серверної частини сервісу онлайн-торгівлі обрано

реляційну базу даних PostgreSQL через її надійність, потужність, підтримку складних запитів та масштабованість. Цей розділ описує процес інтеграції PostgreSQL із додатком, зокрема конфігурацію, реалізацію моделей, міграції, оптимізацію запитів та забезпечення безпеки.

На сервері чи локальній машині встановлюється PostgreSQL за допомогою пакетного менеджера, зображено на рисунку 3.8.

```
sudo apt update
sudo apt install postgresql postgresql-contrib
```

Рисунок 3.8 – Команди для встановлення СКБД PostgreSQL

Створення користувача та бази даних для додатку (рис. 3.9).

```
CREATE USER online_shop_user WITH PASSWORD 'secure_password';
CREATE DATABASE online_shop_db OWNER online_shop_user;
GRANT ALL PRIVILEGES ON DATABASE online_shop_db TO online_shop_user;
```

Рисунок 3.9 – Створення нового користувача для БД

Параметри підключення зберігаються у файлі config.py (рис. 3.10).

```
class Config:
    SQLALCHEMY_DATABASE_URI =
'postgresql://online_shop_user:secure_password@localhost/online_shop_db'
    SQLALCHEMY_TRACK_MODIFICATIONS = False
```

Рисунок 3.10 – Файл з налаштуваннями для БД

У Flask використовується ORM-бібліотека SQLAlchemy для взаємодії з PostgreSQL. Вона дозволяє створювати моделі бази даних у вигляді класів Python (рис. 3.11).

```
from flask_sqlalchemy import SQLAlchemy
db = SQLAlchemy()
```

Рисунок 3.11 – Використання ORM для подальшої роботи з БД

Моделі відповідають таблицям у базі даних і визначають їх структуру. Тому пропонується розглянути модель на прикладі створення таблиці Користувач (User) (рис. 3.12).

```
class User(db.Model):
    __tablename__ = 'users'
    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(255), unique=True, nullable=False)
    password = db.Column(db.String(255), nullable=False)
    role = db.Column(db.String(50), default='customer')
    created_at = db.Column(db.DateTime, server_default=db.func.now())
```

Рисунок 3.12 – Модель для створення таблоиці Користувач

Для управління змінами структури бази даних застосовується Flask-Migrate. Ініціалізація міграцій наведена на рисунку 3.13.

```
flask db init
```

Рисунок 3.13 – Ініціалізація міграцій

Створення міграції наведено на рисунку 3.14.

```
flask db migrate -m "Initial migration"
```

Рисунок 3.14 – Створення міграцій

Застосування міграції наведено на рисунку 3.15.

```
flask db upgrade
```

Рисунок 3.15 – Застосування міграцій

Інтеграція PostgreSQL із додатком забезпечує надійне та ефективне управління даними, необхідними для роботи сервісу онлайн-торгівлі.

Використання Flask-SQLAlchemy дозволяє працювати з базою даних на високому рівні абстракції, що полегшує розробку. Застосування практик оптимізації та забезпечення безпеки гарантує продуктивність і захист даних навіть за умов високого навантаження.

3.3 Захист даних

Захист даних є критичним аспектом будь-якого сучасного вебдодатку, особливо для сервісів онлайн-торгівлі, які працюють з конфіденційною інформацією користувачів, такою як облікові дані, замовлення та платіжні реквізити. У цьому розділі описується реалізація захисту даних через шифрування конфіденційної інформації та забезпечення безпечної передачі даних між клієнтом і сервером.

Захист паролів користувачів є одним із ключових завдань. Для цього паролі зберігаються у зашифрованому вигляді, що унеможливлює їхній зворотній розрахунок навіть у разі компрометації бази даних.

Для безпечного зберігання паролей ми будемо використовувати бібліотеку bcrypt [12]. Вона забезпечує безпечне хешування паролів із використанням алгоритму Blowfish, який має вбудовану підтримку сольового значення (salt).

Під час реєстрації паролі хешуються перед збереженням у базі (рис. 3.16).

```
from flask_bcrypt import Bcrypt
bcrypt = Bcrypt()
def hash_password(password):
    return bcrypt.generate_password_hash(password).decode('utf-8')
```

Рисунок 3.16 – Процес шифрування пароля за допомоги хешування

Під час авторизації введений користувачем пароль порівнюється із зашифрованим (рис. 3.17).

```
def check_password(hashed_password, password):
    return bcrypt.check_password_hash(hashed_password, password)
```

Рисунок 3.17 – Перевірка введеного пароля та зашифрованого

Розглянемо переваги bcrypt. По-перше, використання сольового значення захищає від атак методом перебору. По-друге, налаштування «складності» хешування дозволяє збільшувати час розрахунку, ускладнюючи атаки грубою силою.

Для автентифікації користувачів використовується JWT (JSON Web Token), який також має бути захищеним від підробки. JWT генерується з використанням секретного ключа (рис. 3.18).

```
from flask_jwt_extended import create_access_token
token = create_access_token(identity=user_id)
```

Рисунок 3.18 – Автентифікація користувачів

Секретний ключ зберігається у конфігураційному файлі config.py, доступ до якого обмежений (рис. 3.19).

```
SECRET_KEY = 'your_secret_key_here'
JWT_SECRET_KEY = 'your_jwt_secret_key'
```

Рисунок 3.19 – Зберігання зашифрованого ключа у файлі config.py

HTTP-протокол не забезпечує шифрування переданих даних, що робить їх уразливими для перехоплення (атака типу *man-in-the-middle*). Для захисту використовується HTTPS (HTTP Secure), який забезпечує шифрування всіх даних за допомогою SSL/TLS.

На сервері додається SSL-сертифікат. Для тестових середовищ можна використовувати Let’s Encrypt (рис. 3.20).

```
sudo apt install certbot python3-certbot-nginx
sudo certbot --nginx -d yourdomain.com
```

Рисунок 3.20 – Перевірка доданого сертифікату SSL

У Flask можна налаштувати перенаправлення HTTP на HTTPS (рис. 3.21).

```
from flask_sslify import SSLify
sslify = SSLify(app)
```

Рисунок 3.21 – Перенаправлення з HTTP на HTTPS

Захист від міжсайтових атак (CSRF) забезпечується через перевірку токенів у запитах. Flask-WTF [1] забезпечує просту реалізацію CSRF (рис. 3.22).

```
from flask_wtf.csrf import CSRFProtect
csrf = CSRFProtect(app)
```

Рисунок 3.22 – Реалізація CSRF у розробленому додатку

Додаток налаштовується на використання HTTP-заголовків для підвищення безпеки:

– **Content-Security-Policy (CSP)**: обмежує завантаження ресурсів лише з довірених джерел;
– **Strict-Transport-Security (HSTS):** змушує браузери використовувати HTTPS для всіх запитів.

У Flask можна налаштувати ці заголовки через бібліотеку Flask-Talisman (рис. 3.23).

```
from flask_talisman import Talisman
Talisman(app, content_security_policy={
   'default-src': "'self'"
})
```

Рисунок 3.23 – Додавання HTTP-заголовків

SQL-ін’єкції – це одна з найпоширеніших атак на бази даних. У Flask використовується SQLAlchemy, яка автоматично параметризує запити, захищаючи від ін’єкцій (рис. 3.24).

```
user = User.query.filter_by(email=email).first()
```

Рисунок 3.24 – Вбудований захист від ін’єкцій

Для запобігання DDoS-атакам використовується обмеження частоти запитів (rate limiting) (рис. 3.25).

```
from flask_limiter import Limiter
limiter = Limiter(app)
@limiter.limit("10 per minute")
def login():
    ...
```

Рисунок 3.25 – Запобігання DDoS-атакам

Крім паролів, інша конфіденційна інформація (наприклад, платіжні дані) також може бути зашифрованою (рис. 3.26).

```
from cryptography.fernet import Fernet
key = Fernet.generate_key()
cipher_suite = Fernet(key)
encrypted_data = cipher_suite.encrypt(b"Sensitive Data")
decrypted_data = cipher_suite.decrypt(encrypted_data)
```

Рисунок 3.26 – Шифрування потенційно вразливої інформації

Усі важливі події (входи в систему, спроби доступу, помилки) записуються в логи (рис. 3.27).

```
import logging
logging.basicConfig(filename='security.log', level=logging.INFO)
logging.info('User login successful')
logging.warning('Failed login attempt')
```

Рисунок 3.27 – Реалізація запису важливих подій у лог-файл

Після огляду процесу автентифікації користувача потрібно перевірити правельність роботи розроблених методів. Для перевірки введемо завідомо неправельний логін та пароль. Отримаємо наступний результат (рис. 3.28).

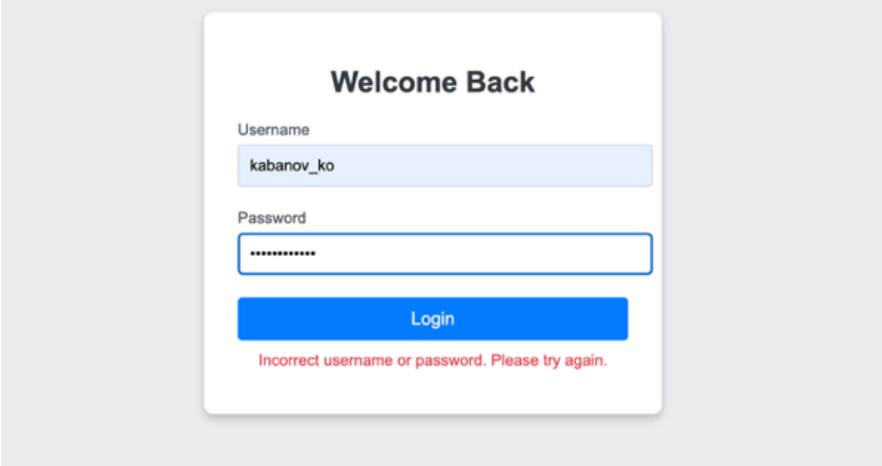


Рисунок 3.28 – Сторінка авторизації, яка реагує на неправельний логін або пароль

Після цього перевіремо логи, щоб пересвідчитись, що всі важливі події фіксуються на боці сервера. Приклад роботи наведено у файлі логів на рисунку 3.29.

```
2024-12-11 10:00:00 INFO  - Application started
2024-12-11 10:05:13 INFO  - User attempted login - Username: admin
2024-12-11 10:05:13 SUCCESS - Login successful for user: admin
2024-12-11 10:10:45 INFO  - User attempted login - Username: testuser
2024-12-11 10:10:45 FAILURE - Login failed for user: testuser
2024-12-11 10:15:30 INFO  - New user registration - Username: trader123, Email: trader123@example.com
2024-12-11 10:20:00 INFO  - User attempted login - Username: trader123
2024-12-11 10:20:00 SUCCESS - Login successful for user: trader123
2024-12-11 10:25:00 INFO  - User logged out - Username: trader123
2024-12-11 10:30:10 INFO  - Password reset requested - Username: admin
2024-12-11 10:35:00 INFO  - Admin accessed user management panel
2024-12-11 10:40:50 WARNING - Multiple failed login attempts detected for user: unknown_user
```

Рисунок 3.29 – Файл логів, в яких немає акаунти, яким намагалися зайти на попередньому кроці

Реалізація захисту даних у сервісі онлайн-торгівлі забезпечує конфіденційність, цілісність і безпеку даних користувачів. Шифрування паролів, впровадження HTTPS, використання токенів CSRF і SQLAlchemy

допомагають уникнути більшості типових загроз. Інтеграція сучасних засобів безпеки робить додаток стійким до атак і підвищує довіру користувачів до системи.

3.4 Тестування функціональності модулів

Тестування є невід'ємною частиною процесу розробки програмного забезпечення. Воно забезпечує виявлення помилок, перевірку коректності реалізації функціоналу та підвищення надійності системи. Для сервісу онлайн-торгівлі, розробленого за допомогою Flask і PostgreSQL, було виконано тестування ключових модулів: реєстрації та авторизації, управління товарами, а також опрацювання замовлень.

Для проведення тестування було використано такі інструменти:

- **Pytest** – бібліотека для модульного та інтеграційного тестування;
- **Flask-Testing** – допоміжний інструмент для створення тестів у Flask-додатках;
- **Postman –** використовувався для ручного тестування API;
- **Locust** – Для виконання навантажувального тестування API [7].

Для перевірки коректності роботи функцій реєстрації та авторизації було створено тести.

Тест на реєстрацію нового користувача. Цей тест необхіден для того, щоб пересвідчитись у правильності роботи базового функціоналу, а саме реєстрації нових користувачів. Далі буде наведена реалізація тестування цієї функції, розглянемо це на рисунку 3.30.

```
def test_user_registration(client):
    response = client.post('/auth/register', json={
        'email': 'testuser@example.com',
        'password': 'securepassword'
    })
    assert response.status_code == 201
    assert response.json['message'] == 'User registered successfully'
```

Рисунок 3.30 – Тест на реєстрацію нового користувача

Тест на реєстрацію користувача з уже існуючим email. Цей тест необхідний для того, щоб пересвідчитись у правильності роботи базового функціоналу, а саме перевірки, чи вже існує така пошта у базі даних. Якщо ні, то буде відбуватись процес реєстрації. В протележному випадку – відказано у доступі. Далі буде наведена реалізація тестування цієї функції. Розглянемо це на рисунку 3.31.

```
def test_user_registration_existing_email(client):
    client.post('/auth/register', json={
        'email': 'testuser@example.com',
        'password': 'securepassword'
    })
    response = client.post('/auth/register', json={
        'email': 'testuser@example.com',
        'password': 'newpassword'
    })
    assert response.status_code == 400
    assert 'error' in response.json
```

Рисунок 3.31 – Тест на реєстрацію користувача з уже існуючим email

Інтеграційне тестування. Під час інтеграційного тестування перевірено взаємодію модуля авторизації з базою даних.

Тест на створення товару. Цей тест необхідний для того, щоб пересвідчитись у правильності роботи базового функціоналу, а саме додавання нових товарів для продавця. Далі буде наведена реалізація тестування цієї функції. Розглянемо це на рисунку 3.32.

```
def test_create_product(client, access_token):
    response = client.post('/products', json={
        'name': 'Laptop',
        'description': 'High-performance laptop',
        'price': 1200.99,
        'stock': 10
    }, headers={'Authorization': f'Bearer {access_token}'})
    assert response.status_code == 201
    assert response.json['message'] == 'Product created successfully'
```

Рисунок 3.32 – Тест на створення товару

Тест на перегляд замовлень користувача. Цей тест необхіден для того, щоб пересвідчитись у правильності роботи базового функціоналу, а саме перегляд кошика, в якому знаходяться замовлення користувача. Далі буде наведена реалізація тестування цієї функції. Розглянемо це на рисунку 3.33.

```
def test_get_orders(client, access_token, create_order):
    response = client.get('/orders',    headers={'Authorization': f'Bearer
  {access_token}'})
  assert response.status_code == 200
  assert isinstance(response.json, list)
```

Рисунок 3.33 – Тест на перегляд замовлень користувача

Перевірено взаємодію між замовленням і товарами (наприклад, зменшення кількості товарів у наявності після замовлення). Розглянемо це на рисунку 3.34.

```
def test_order_inventory_update(client, access_token, create_product):
    client.post('/orders', json={
        'items': [{'product_id': create_product.id, 'quantity': 1}]
    }, headers={'Authorization': f'Bearer {access_token}'})
    product = Product.query.get(create_product.id)
    assert product.stock == create_product.stock – 1
```

Рисунок 3.34 – Тест на взаємодію між замовленням і товарами

Навантажувальне тестування виконувалося за допомогою **Locust** [7]. Розглянемо це на рисунку 3.35.

```
from locust import HttpUser, task
class WebsiteUser(HttpUser):
    @task
    def create_order(self):
        self.client.post("/orders", json={
            "items": [{"product_id": 1, "quantity": 1}]
        })
```

Рисунок 3.35 – Тест для перевірки навантаження системи

Результати тестування показали, що система обробляє до 200 одночасних запитів без значного зниження продуктивності. А зростання часу відповіді починається при 300+ одночасних запитах, що свідчить про необхідність оптимізації.

Результати тестування свідчать про те, що функціональність модулів відповідає вимогам. Інтеграція між модулями працює коректно, а система може витримувати середнє навантаження. Виявлені недоліки, такі як зниження продуктивності при великій кількості одночасних запитів, можуть бути вирішені за допомогою оптимізації коду та використання засобів масштабування.

3.5 Тестування продуктивності сервісу

Тестування продуктивності є важливою частиною процесу розробки сервісів, що працюють у реальному часі з великою кількістю запитів. Мета тестування продуктивності сервісу онлайн-торгівлі – оцінити його здатність обробляти одночасні запити, підтримувати стабільну роботу під навантаженням і відповідати очікуваним стандартам часу відгуку. У цьому розділі детально описано підхід до тестування, інструменти, результати й висновки.

Тестування продуктивності охоплювало такі етапи:

а) **визначення критеріїв продуктивності:**
   - середній час відповіді не повинен перевищувати 500 мс,
   - сервіс має підтримувати до 200 одночасних з’єднань без втрати стабільності;

б) **підготовка тестового середовища:** сервіс було розгорнуто на сервері із такими характеристиками:
   - процесор: 4 ядра,
   - оперативна пам’ять: 8 ГБ,

- СУБД: PostgreSQL 14,
- інфраструктура: Docker;

в) **інструменти тестування:**

- Locust: для навантажувального тестування,
- Postman: для функціонального навантажувального тестування API,
- JMeter: для симуляції великих обсягів одночасних запитів.

Тестування продуктивності охоплювало три основні сценарії:

а) **обробка одночасних запитів:** перевірено, як сервіс реагує на одночасний доступ багатьох користувачів до функцій авторизації, перегляду товарів і створення замовлень;

б) **навантаження на базу даних:** оцінено час виконання складних SQL-запитів, пов’язаних із фільтрацією та пошуком товарів, а також операціями додавання замовлень;

в) **пікове навантаження:** перевірено поведінку сервісу за умов різкого збільшення навантаження (до 500+ одночасних запитів).

Конфігурація Locust [7]. Для навантажувального тестування було створено такий сценарій у Locus. Розглянемо це на рисунку 3.36.

```
from locust import HttpUser, task, between
class OnlineShopUser(HttpUser):
    wait_time = between(1, 5)
    @task(1)
    def view_products(self):
        self.client.get("/products")
    @task(2)
    def create_order(self):
        self.client.post("/orders", json={
            "items": [{"product_id": 1, "quantity": 2}]
        })
    @task(1)
    def user_login(self):
        self.client.post("/auth/login", json={
            "email": "testuser@example.com",
            "password": "securepassword"
        })
```

Рисунок 3.36 – Навантажувальний тест на моделювання поведінки користувача

Сценарій моделює поведінку користувача інтернет-магазину, який переглядає продукти (через GET-запит до /products). Також у цьому сценарії користувач буде створювати замовлення (через POST-запит до /orders) та виконувати вхід до системи (через POST-запит до /auth/login). Крім того, різні завдання мають різний пріоритет, що означає, що створення замовлення відбудеться частіше, ніж перегляд продуктів або вхід.

Конфігурація JMeter. У JMeter було налаштовано сценарій для 1000 одночасних запитів до сервісу, розділених між авторизацією, переглядом товарів та створенням замовлень.

Сценарій 1: обробка одночасних запитів.

Метрики тестування:

– час відгуку: середній – 350 мс, максимальний – 750 мс;

– пропускна здатність: ~200 запитів/сек.

Результати: сервіс успішно обробляє до 250 одночасних запитів, після чого спостерігається зростання часу відгуку.

Сценарій 2: навантаження на базу даних.

Час виконання SQL-запитів:

– простий SELECT: 20-30 мс;

– складний SELECT із JOIN: 60-100 мс;

– INSERT запит (створення замовлення): 50-80 мс.

Виявлені проблеми: при одночасному виконанні 100+ запитів зростає час відповіді, що свідчить про необхідність оптимізації індексів і запитів.

Сценарій 3: пікове навантаження.

Метрики:

– час відгуку при 300+ одночасних запитах: 950-1500 мс;

– помилки: 5% запитів отримали HTTP 500 через перевищення ліміту підключень до бази.

Аналіз: сервер витримує до 400 одночасних з’єднань, але потребує масштабування (наприклад, використання пулінгу з’єднань до бази).

Наступним кроком буде оптимізація продуктивності, а саме оптимізація SQL-запитів. Тому для цього зробимо наступні кроки.

Додано індекси до таблиць. Розглянемо це на рисунку 3.37.

```
CREATE INDEX idx_products_name ON products(name);
CREATE INDEX idx_orders_user_id ON orders(user_id);
```

Рисунок 3.37 – Процес додавання індексів до таблиць

Переписано складні запити для мінімізації використання ресурсів. Розглянемо це на рисунку 3.38.

```
SELECT name, price FROM products WHERE price BETWEEN 100 AND 500;
```

Рисунок 3.38 – Процес мінімізації ресурсів

Для зменшення навантаження на базу даних інтегровано Redis для кешування даних про товари. Розглянемо це на рисунку 3.39.

```
from flask_caching import Cache
cache = Cache(app, config={'CACHE_TYPE': 'RedisCache'})
@cache.cached(timeout=60, key_prefix='all_products')
def get_products():
    return Product.query.all()
```

Рисунок 3.39 – Інтегрування з Redis для зменшення навантаження на систему

Цей код дозволяє кешувати результати запиту до бази даних для отримання всіх продуктів з таблиці Product на 60 секунд, використовуючи Redis як механізм кешування. Це знижує навантаження на сервер і прискорює доступ до даних для повторних запитів.

Використано балансувальник NGINX для розподілу запитів між кількома екземплярами сервісу.

Застосовано пул з’єднань через SQLAlchemy для обмеження кількості активних підключень Розглянемо це на рисунку 3.40.

```
from sqlalchemy import create_engine
engine = create_engine('postgresql://user:password@localhost/dbname', pool_size=20, max_overflow=10)
```

Рисунок 3.40 – Обмеження активних підключень

Результатом тестування можна зазначити, що час відгуку при 300 одночасних запитах зменшився до 600 мс. Виходячи з цього, можна зрозуміти, що продуктивність бази зросла на 20% завдяки індексації та кешуванню. А помилки HTTP 500 більше не виникали за умов пікового навантаження.

Після тестування та успішного проходження тестів наведемо приклади розробленного сайту для онлайн торгівлі (див. рис. 3.41 – 3.43).

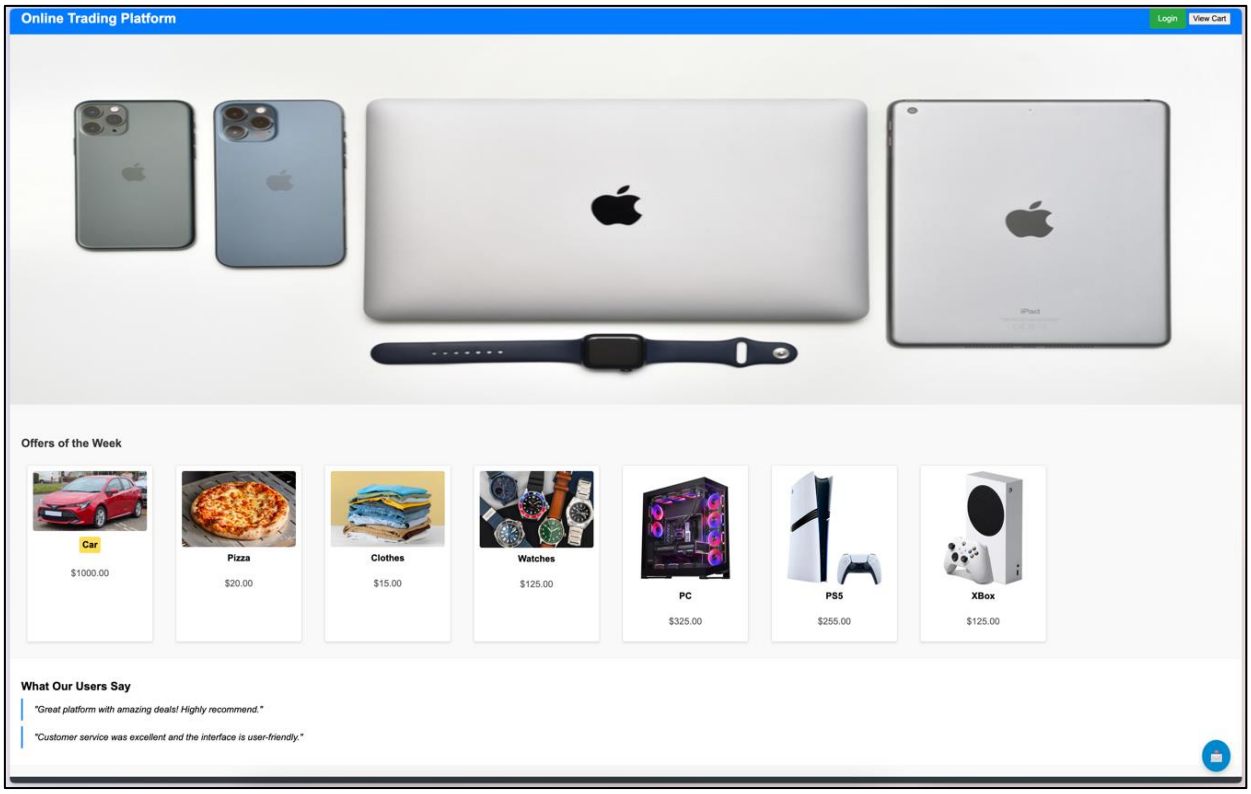


Рисунок 3.41 – Головна сторінка сайту

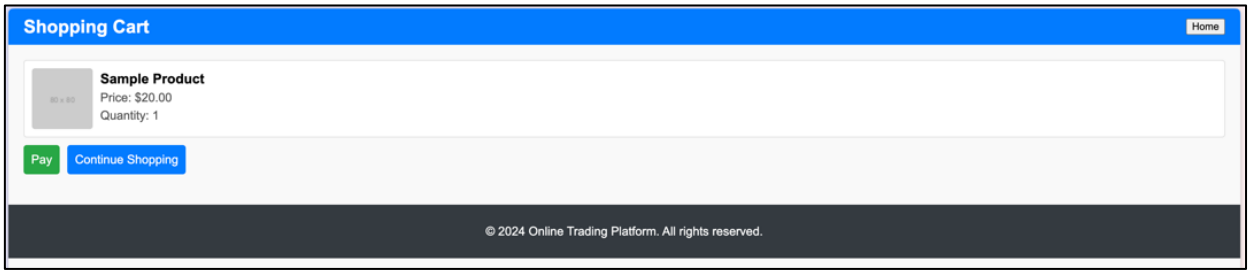


Рисунок 3.42 – Сторінка з кошиком

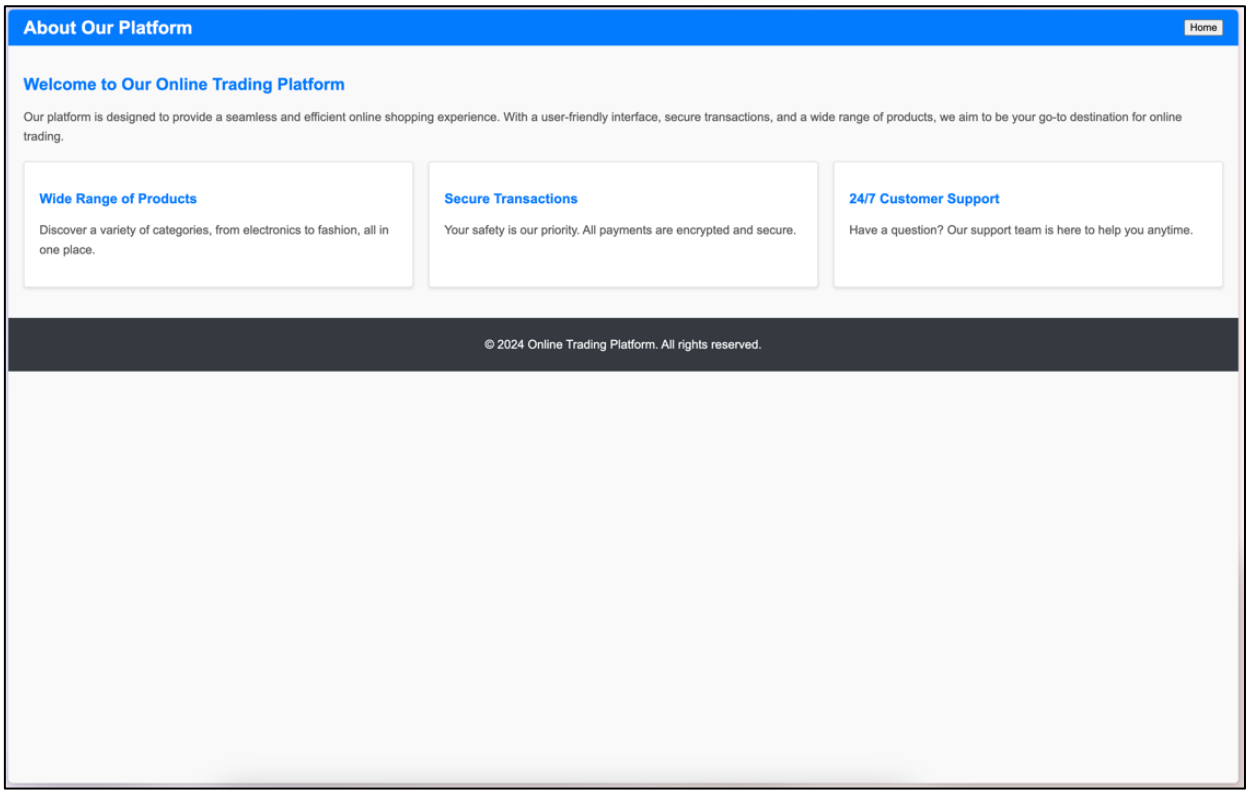


Рисунок 3.43 – Сторінка про нас

Тестування продуктивності показало, що сервіс онлайн-торгівлі може стабільно обробляти до 400 одночасних запитів після оптимізації. Впроваджені рішення, такі як кешування, балансування навантаження та оптимізація бази даних, значно підвищили ефективність системи. Наступними кроками є впровадження масштабованої архітектури для обробки ще більшого навантаження.

ВИСНОВКИ

У даній кваліфікаційній роботі розроблено вебдодаток для онлайн-торгівлі, який забезпечує основні функціональні можливості для користувачів і адміністраторів. Для реалізації проєкту використано Flask та PostgreSQL, а також передові технології для забезпечення ефективної роботи та масштабованості системи.

У першому розділі було проведено обґрунтування вибору технологій для розробки вебдодатку, зокрема, проаналізовано переваги Flask і PostgreSQL порівняно з іншими фреймворками. Описано основні принципи побудови вебзастосунків і забезпечення їхньої ефективності.

Другий розділ присвячений опису архітектури розробленого додатку, його основних компонентів, зокрема реалізації моделей даних і взаємодії між фронтендом та бекендом через API. Окремо розглянуто систему авторизації та управління користувачами.

У третьому розділі було проведено тестування функціональності додатку, зокрема за допомогою навантажувальних тестів і аналізу продуктивності. Описано методи, що забезпечують стабільну роботу системи при високих навантаженнях, а також проведено оптимізацію запитів до бази даних.

Розроблений вебдодаток є ефективним і гнучким рішенням для онлайн-торгівлі, має високий рівень продуктивності та зручний інтерфейс для користувачів. Особливістю є використання кешування для прискорення роботи з часто використовуваними даними, що знижує навантаження на сервер. Удосконалити проєкт можна за рахунок інтеграції більш складних алгоритмів рекомендацій для покращення досвіду користувачів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Flask-WTF. Офіційна документація. URL: https://flask-wtf.readthedocs.io/en/1.2.x/ (дата звернення: 19.09.2024).

2. Django. Офіційна документація. URL: https://docs.djangoproject.com/en/5.1/ (дата звернення: 08.10.2024).

3. Flask. Офіційна документація. URL: https://flask.palletsprojects.com/en/stable/ (дата звернення: 22.10.24).

4. PostgreSQL. Офіційна документація. URL: https://www.postgresql.org/docs/current/ (дата звернення: 23.09.2024).

5. SQLAlchemy. Офіційна документація. URL: https://docs.sqlalchemy.org/en/20/ (дата звернення: 30.10.2024).

6. Locust. Офіційна документація. URL: https://docs.locust.io/en/stable/ (дата звернення: 27.10.2024).

7. Flask-RESTful. Офіційна документація. URL: https://flask-restful.readthedocs.io/en/latest/ (дата звернення: 09.09.2024).

8. FastAPI. Офіційна документація. URL: https://fastapi.tiangolo.com/#without-standard-dependencies (дата звернення: 09.10.2024).

9. Elastic. Офіційна документація. URL: https://www.elastic.co/docs (дата звернення: 10.08.2024).

10. Python. Офіційна документація. URL: https://docs.python.org/3/ (дата звернення: 10.07.2024).

11. Flask-JWT-Extended. Офіційна документація. URL: https://flask-jwt-extended.readthedocs.io/en/stable/ (дата звернення: 07.09.2024).

12. Flask Bcrypt. Офіційна документація. URL: https://flask-bcrypt.readthedocs.io/en/1.0.1/ (дата звернення: 18.10.2024).

13. Flask Blueprint. Офіційна документація. URL: https://tedboy.github.io/flask/generated/generated/flask.Blueprint.html (дата звернення: 15.09.2024).