ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Проектування та розробка повноцінного REST API для створення онлайн-магазину»

на здобуття освітнього ступеня магістра
зі спеціальності 122 Комп'ютерні науки
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерні науки
(назва)

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

_____ Кирило БОНДАРЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ здобувача)*

Виконав:
здобувач вищої освіти Кирило БОНДАРЕНКО
група КНДМ-62

Керівник: Олена ШИКУЛА
науковий ступінь, вчене звання д.т.н., професор

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

Київ 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Комп'ютерних наук

Ступінь вищої освіти Магістр

Спеціальність Комп'ютерні науки

Освітньо-професійна програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедрою Комп'ютерних наук

_____ Віктор ВИШНІВСЬКИЙ
«_____» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Бондаренко Кирилу Ігоровичу
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Проектування та розробка повноцінного REST API для створення онлайн-магазину

керівник кваліфікаційної роботи Олена ШИКУЛА д.т.н., професор
(Ім'я, ПРІЗВИЩЕ науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «19» жовтня 2023р. №. 145

2. Строк подання кваліфікаційної роботи «29» грудня 2023р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, вимоги Node.js та TypeScript, практики програмування

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Дослідження проблем розробки серверної частини сайту
Аналіз проблеми захисту

Розробка функціонуючої частини серверу використовуючи найкращі практики Node.js

5. Перелік графічного матеріалу: *презентація*
    1. Задачі у електронній комерції.
    2. Виклики у сфері електронної комерції.
    3. Аутентифікація та авторизація.
    4. Node.js для серверної частини
    5. Архітектура бази даних.

6. Дата видачі завдання «19» жовтня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	19.10-05.11.23	
2	Дослідження проблем та задач у електронній комерції	05.11-12.11.23	
3	Дослідження аутентифікації та авторизації використовуючи Node.js	13.11-19.11.23	
4	Аналіз новітніх практик у TypeScript	20.11-25.11.23	
5	Розробка додатку	27.11-03.12.23	
6	Аналіз захисту та роботоспроможності додатку	04.12-10.12.23	
7	Оформлення роботи: вступ, висновки, реферат	11.12-20.12.23	
8	Розробка демонстраційних матеріалів	21.12-29.12.23	

Здобувач вищої освіти _____ Кирило БОНДАРЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Олена ШИКУЛА
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 65 стор., 25 рис., 28 джерел.

Наукове завдання – Подолання проблем у процесі проектування та розробки повноцінного REST API для створення онлайн-магазину з фокусом на високий рівень безпеки та захисту даних користувачів.

Мета роботи – Дослідження та порівняння ефективності різних стратегій та методів захисту в контексті проектування REST API для електронного магазину. Розробка та впровадження найоптимальніших рішень щодо безпеки та конфіденційності.

Об'єкт дослідження – Процес створення та функціонування додатку.

Предмет дослідження – Проектування та розробка безпечного та надійного REST API для ефективного управління онлайн-магазином, з основним акцентом на захист особистих даних.

Короткий зміст роботи: Аналізовано основні виклики та завдання при створенні REST API для онлайн-магазину з високим ступенем уваги до безпеки. Проведено порівняльний аналіз різних стратегій захисту, враховуючи специфіку електронної комерції. Розроблено та впроваджено заходи для підвищення рівня безпеки API, такі як шифрування даних, автентифікація та контроль доступу.

КЛЮЧОВІ СЛОВА: REST API, БЕЗПЕКА, ОНЛАЙН-МАГАЗИН, ПРОЕКТУВАННЯ, РОЗРОБКА, ЗАХИСТ ДАНИХ, NODE.JS, TYPESCRIPT, АУТЕНТИФІКАЦІЯ

ABSTRACT

Text part of the master's qualification work: 65 pages, 25 pictures, 28 sources.

The scientific task - Overcoming problems in the process of designing and developing a full-fledged REST API for creating an online store with a focus on a high level of security and protection of user data.

The objective - The purpose of the work is to study and compare the effectiveness of various protection strategies and methods in the context of designing a REST API for an e-store. Development and implementation of the most optimal security and privacy solutions.

Research object – process of creating and functioning of the application.

Research subject – design and development of a safe and reliable REST API for the effective management of an online store, with the main emphasis on the protection of personal data.

Summary of the work: The main challenges and tasks in creating a REST API for an online store with a high degree of attention to security are analyzed. A comparative analysis of various protection strategies was carried out, taking into account the specifics of electronic commerce. Developed and implemented measures to improve API security, such as data encryption, authentication, and access control.

KEYWORDS: REST API, SECURITY, ONLINE STORE, DESIGN, DEVELOPMENT, DATA PROTECTION, NODE.JS, TYPESCRIPT, AUTHENTICATION

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ..... 9
ВСТУП.....10
1 ОГЛЯД ТА АНАЛІЗ ОБЛАСТІ ДОСЛІДЖЕННЯ..... 12
1.1 Огляд предметної області..... 12
1.2 Огляд цифрових сервісів..... 16
1.3 Перспективи розвитку сайту..... 20
1.4 Актуальні проблеми в галузі..... 26
2 АНАЛІЗИ ТА ШЛЯХИ РОЗВИТКУ ПРОБЛЕМИ..... 31
2.1 Інструментальні засоби та технології створення..... 31
2.1.1 Мова програмування TypeScript у контексті Node.js..... 31
2.1.2 Середовище розробки JetBrains WebStorm..... 33
2.1.3 СУБД PostgreSql..... 34
2.1.4 ОРМ Typeorm для PostgreSql.....36
2.1.5 Postman для автоматизації тестування..... 40
2.2 Технічні виклики та рішення.....44
3 ОПИС ПРОГРАМНОГО ПРОДУКТУ.....48
3.1 Вхідні та вихідні дані.....48
3.2 Проектування бази даних..... 49
3.3 Опис роботи та тестування додатку..... 55
ВИСНОВКИ..... 68
ПЕРЕЛІК ПОСИЛАНЬ.....70
ДОДАТОК А. КОДИ ДОДАТКУ.....72
ДОДАТОК Б. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ.....102

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API - інтерфейс прикладного програмування

БД – база даних

СУБД – система управління базами даних

SQL – декларативна мова програмування для взаємодії користувача

CMS - системи управління контентом

ОРМ - об'єктно-реляційне відображення

JWT - стандарт токена доступ

ВСТУП

У сучасному світі електронна комерція є важливим інструментом розширення бізнесу та забезпечення доступу до товарів та послуг на глобальному рівні. Щоб забезпечити конкурентоспроможність та задоволення вимог споживачів, онлайн-магазини повинні оперативно реагувати на зміни у ринкових умовах та ефективно взаємодіяти з різними сервісами та платформами.

Актуальність розробки API для онлайн-магазину визначається потребою в створенні гнучких, масштабованих та ефективних інфраструктур. За допомогою API можна оптимізувати обробку замовлень, автоматизувати взаємодію з платіжними системами, а також покращити аналітику та взаємодію з клієнтами.[1]

Сучасні тенденції електронної комерції вимагають не тільки створення ефективних магазинів, але й швидкого реагування на нові можливості та вимоги ринку. Розробка API стає стратегічним інструментом для забезпечення легкої інтеграції з іншими сервісами, зокрема з системами управління запасами, системами доставки та іншими, що дозволяє підтримувати високий рівень обслуговування та задоволення клієнтів.

Таким чином, розробка API для онлайн-магазинів є актуальною та стратегічно важливою темою, яка відкриває можливості для поліпшення ефективності бізнесу, забезпечуючи одночасно високий рівень зручності та задоволення від користування для клієнтів.

Основною метою даного дослідження є створення та впровадження ефективного та функціонального API для онлайн-магазину з метою підвищення його ефективності та конкурентоспроможності.

1. Проектування API: Розробити детальну архітектуру та дизайн API, враховуючи специфіку магазину. Забезпечити максимальну гнучкість та розширюваність інтерфейсу для взаємодії з різними компонентами системи.

2. Реалізація API: Розробити програмний код API, враховуючи рекомендації та стандарти, визначені на етапі проектування. Забезпечити

ефективну роботу з базою даних, обробку замовлень, а також взаємодію з іншими сервісами магазину.

3. Тестування та оптимізація: Провести комплексне тестування розробленого API для переконливої стабільності та надійності. Оптимізувати роботу інтерфейсу для максимальної продуктивності та забезпечення швидкого доступу до даних.

4. Документація: Ретельно підготувати та представити документацію для розробленого API. Забезпечити зрозумілість та легкість використання для інших розробників, які можуть інтегрувати чи розширювати функціонал API.

5. Впровадження та підтримка: Забезпечити ефективне впровадження розробленого API в роботу магазину та регулярну технічну підтримку для вирішення можливих проблем та розширення функціоналу.

Мета дослідження полягає в створенні високопродуктивного та гнучкого API, яке відповідає потребам конкретного онлайн-магазину, сприяючи покращенню ефективності бізнес-процесів та підвищенню задоволення клієнтів.

1 ОГЛЯД ТА АНАЛІЗ ОБЛАСТІ ДОСЛІДЖЕННЯ

1.1 Огляд предметної області

Електронна комерція (e-commerce) не лише впроваджується в сучасний бізнес, але й перетворює його обличчя, ставлячи підприємства в нові умови конкуренції та спілкування з клієнтами. Цей метод купівлі та продажу товарів і послуг через Інтернет визначається своєрідною гнучкістю та глобальністю.

Електронні магазини дозволяють клієнтам не лише швидко та зручно здійснювати покупки в будь-який час, але і вибирати з різноманітної асортиментної лінійки товарів та послуг. Ця динаміка призводить до збільшення конкуренції серед бізнесів, оскільки вони прагнуть пристосовуватися до змінних вимог сучасного споживача.

Однак разом із зростанням популярності електронної комерції виникають нові виклики для підприємств. Крім створення привабливих інтернет-магазинів, важливо забезпечити їхню ефективну інтеграцію з іншими системами та сервісами.

API, або інтерфейс програмування застосунків, виступає як мостик, що сполучає різні компоненти електронного магазину та дозволяє їм ефективно взаємодіяти. Це важливий інструмент для створення цілісної інфраструктури, яка сприяє швидкій обробці замовлень та розширенню функціоналу. API в електронній комерції використовується для підключення платіжних систем, інтеграції з системами управління запасами, аналітичними інструментами та іншими сервісами. Він забезпечує автоматизацію багатьох бізнес-процесів, що веде до ефективнішого управління та більшої задоволеності клієнтів.

Однією з ключових тенденцій в розробці API для онлайн-магазинів є використання RESTful архітектури. Вона дозволяє створювати легкі та гнучкі інтерфейси, які легко інтегруються з різноманітними клієнтами та іншими системами.[2]

Безпека також стає пріоритетом у розробці API. Використання засобів аутентифікації та авторизації забезпечує захист конфіденційності та цілісності даних, що особливо важливо в електронній комерції.

Розробка API напряму впливає на функціональність та продуктивність онлайн-магазину. Правильно спроектоване та ефективно реалізоване API дозволяє магазину швидше реагувати на зміни в попиті, оптимізувати обробку замовлень та поліпшувати загальний користувацький досвід. Оптимізація інтерфейсу API та впровадження кращих практик у розробці дозволяють забезпечити високий рівень продуктивності та швидкий доступ до даних. Тестування та оптимізація стають важливим етапом для забезпечення стабільності та ефективності API в умовах реального електронного бізнесу.

Розробка API для онлайн-магазину визначає ключові аспекти функціональності та продуктивності, суттєво вдосконалюючи його бізнес-процеси та взаємодію з клієнтами:

1. Покращення функціональності:

Справжній успіх електронного магазину полягає в його здатності ефективно відповідати на різноманітні потреби клієнтів. Розробка API дозволяє розширити функціональні можливості магазину, надаючи зручний та гнучкий інтерфейс для інтеграції з різними додатками та сервісами. Відділи замовлень, інвентаризації та аналітики можуть взаємодіяти із системами сторонніх розробників, що забезпечить гладку та автоматизовану роботу магазину.

2. Підвищення продуктивності:

Швидкість обробки замовлень та реакція на зміни в попиті є критичними для успішності онлайн-магазину. Правильно розроблене API дозволяє оптимізувати ці процеси, забезпечуючи ефективне взаємодію з базою даних, системами управління запасами та іншими компонентами магазину. Це призводить до скорочення часу обробки замовлень, зменшення ймовірності помилок та вищої продуктивності управління.

3. Покращення користувацького досвіду:

Забезпечення оптимального користувацького досвіду є важливою метою для будь-якого електронного магазину. Розробка API дозволяє створити інтегровані та гнучкі рішення, які відповідають потребам сучасних покупців. Відстеження замовлень, персоналізація пропозицій та швидкий доступ до інформації – це всі елементи, які можуть бути оптимізовані завдяки впровадженню ефективного API.[3]

Інтернет-маркетинг в сфері електронної комерції відіграє ключову роль у забезпеченні видимості та привертанні уваги цільової аудиторії. В умовах зростання конкуренції важливою стає не лише наявність електронного магазину, але і ефективна стратегія його просування в Інтернеті.

Стратегії інтернет-маркетингу в електронній комерції включають в себе різноманітні підходи, такі як контент-маркетинг, пошукова оптимізація (SEO), соціальні медіа кампанії, email-маркетинг та інші. Кожен з цих інструментів сприяє підвищенню обігу та залученню цільової аудиторії.[4]

Особливу увагу слід приділити використанню аналітики для вимірювання ефективності маркетингових кампаній. Збір та аналіз даних дозволяє визначити успішні стратегії, а також вчасно коригувати невдалий контент чи рекламні заходи.

Зростання популярності мобільних пристроїв ставить завдання перед маркетологами електронної комерції в адаптації своїх стратегій до мобільного середовища. Мобільний маркетинг, мобільні додатки та оптимізовані версії веб-сайтів стають важливими елементами успішної маркетингової стратегії.

Також вплив екологічних трендів на електронну комерцію є надзвичайно актуальним та значущим в контексті сучасних вимог споживачів та сталого розвитку бізнесу. Зростання свідомості споживачів щодо екологічних питань призводить до змін у їхньому способі споживання та вибору постачальників товарів і послуг. В цьому контексті електронні магазини, які включають екологічно відповідальні практики у свою діяльність, можуть отримати суттєву конкурентну перевагу.

Одним з ключових аспектів, який може впливати на успіх електронного магазину в екологічно освідомленому ринковому середовищі, є використання екологічно чистих упаковок. Споживачі все більше оцінюють продукти, які не лише відповідають їхнім потребам, але й мають мінімальний вплив на довкілля. Використання вторинної упаковки, біорозкладаючих матеріалів та зменшення використання пластику може зробити продукти електронного магазину більш привабливими для цільової аудиторії.

Додатковою стратегією, яка впливає на екологічну стійкість електронного магазину, є зменшення вуглецевого сліду. Це може включати оптимізацію логістичних процесів, використання ефективних методів транспорту та виробництва, а також перехід до відновлюваних джерел енергії. Публічне сповіщення про зусилля у зменшенні вуглецевого сліду може вразити та залучити споживачів, які прагнуть підтримувати екологічно відповідальних брендів.[5]

Підтримка еко-ініціатив та участь у програмах екологічного захисту також може сприяти позитивному сприйняттю бренду. Електронні магазини можуть співпрацювати з організаціями, що займаються екологічними питаннями, або власноруч впроваджувати програми відновлення лісів, збирання сміття або відновлення екосистем.

Враховуючи ці аспекти, екологічна відповідальність може стати не лише складовою маркетингової стратегії електронного магазину, але й важливим елементом його корпоративної соціальної відповідальності. Такий підхід сприяє створенню позитивного іміджу компанії, забезпечує конкурентні переваги та сприяє розвитку сталого бізнесу в умовах зростаючого екологічного середовища.

Також забезпечення швидкої та надійної доставки стає ключовим фактором конкурентоспроможності на міжнародному ринку. Важливим елементом є впровадження систем відстеження вантажів у реальному часі. Це не лише надає клієнтам можливість моніторити свої замовлення, але й створює додаткову впевненість у тому, що їх товари перебувають на шляху до них.

Оптимізація шляхів доставки також відіграє важливу роль. Адаптація маршрутів до регіональних особливостей та плідна співпраця з логістичними партнерами дозволяють знизити витрати та збільшити швидкість доставки.[6]

Ефективне управління запасами - ще один важливий аспект. Застосування технологій прогнозування попиту, зокрема, за допомогою машинного навчання, дозволяє уникнути надлишкового або недостатнього товарного запасу, що впливає на оперативність та вартість бізнес-процесів.

Таким чином, оптимізація логістики та доставки в електронній комерції не тільки впливає на задоволеність клієнтів, але й стає стратегічним фактором для забезпечення конкурентоспроможності на ринку. За умови правильної імплементації цих підходів, електронні магазини отримують можливість не лише задовольняти, але й перевершувати очікування своїх клієнтів.

1.2 Огляд цифрових сервісів

З розвитком технологій та зростанням вимог сучасного споживача важливо не лише пропонувати високоякісні товари та послуги, але й забезпечити їхню максимальну доступність та зручність використання. У цьому контексті цифрові сервіси виявляються ключовим елементом електронної комерції, що покликані відповісти на сучасні виклики та очікування споживачів.

Веб-сайт. Це візитівка бізнесу, де покупці можуть не лише переглядати асортимент товарів, але й легко та зручно здійснювати покупки через онлайн-платформу. Інтуїтивний інтерфейс та зручна навігація дозволяють користувачам швидко знаходити необхідну інформацію та здійснювати покупки за кілька кліків.[7]

Окрім функціональності електронної вітрини, веб-сайт служить інформаційним ресурсом. Тут клієнти можуть детально дізнатися про товари, ознайомитися з умовами доставки та оплати, отримати вичерпну відповідь на свої питання. Системи відгуків та рейтингів роблять процес вибору товару більш прозорим. Користувачі можуть ділитися своїм досвідом та оцінювати товари, що

підвищує довіру до бренду та допомагає іншим покупцям у виборі. Таким чином, веб-сайт виступає як ключовий елемент успішного електронного бізнесу, забезпечуючи зручність, доступність та надійність для клієнтів.[8]

Веб-сайт є не лише електронною вітриною бізнесу, але й ключовим інструментом в сфері електронної комерції. Він виконує роль візитівки, де потенційні покупці можуть не лише ознайомитися з асортиментом товарів, але й легко та зручно здійснювати покупки через онлайн-платформу. Інтуїтивний інтерфейс та зручна навігація дозволяють користувачам швидко знаходити необхідну інформацію та здійснювати покупки за кілька кліків.

Окрім функціональності електронної вітрини, веб-сайт виконує роль інформаційного ресурсу. Тут клієнти можуть детально дізнатися про товари, вивчити умови доставки та оплати, отримати вичерпні відповіді на свої питання. Системи відгуків та рейтингів роблять процес вибору товару більш прозорим. Користувачі можуть ділитися своїм досвідом та оцінювати товари, що підвищує довіру до бренду та допомагає іншим покупцям у виборі. Таким чином, веб-сайт виступає як ключовий елемент успішного електронного бізнесу, забезпечуючи зручність, доступність та надійність для клієнтів.

Зокрема, важливо підкреслити, що веб-сайт не лише спрощує процес покупки, але і служить інформаційним центром. Детальна інформація про товари, умови обслуговування та часті питання дозволяють клієнтам взяти максимум із свого онлайн-шопінгу. Безпосередній зворотній зв'язок через систему відгуків та рейтингів створює прозорий простір для обміну думками та досвідом між споживачами, що сприяє розвитку довіри до бренду та полегшує вибір товарів для інших покупців.

Веб-додаток. В еру мобільності та постійного руху веб-додатки стають невід'ємною частиною сучасного електронного бізнесу. Їхня роль полягає в наданні додаткового рівня зручності та доступності для клієнтів, незалежно від їхнього місця перебування.

Веб-додатки дозволяють користувачам звертатися до магазину через веб-браузер, уникнувши необхідності завантаження та інсталяції додаткових програм на свої пристрої. Оптимізовані для роботи на різних пристроях, вони забезпечують безперебійну роботу як на комп'ютерах, так і на смартфонах чи планшетах.[9]

Основна перевага веб-додатків полягає в тому, що вони підтримують основні функціональні можливості, які доступні на веб-сайті. Клієнти можуть легко та зручно здійснювати покупки, додавати товари до кошика, відстежувати статус своїх замовлень та отримувати повідомлення про акції, навіть перебуваючи в дорозі чи на роботі.[10]

Враховуючи ритм сучасного життя, веб-додатки стають надійним компаньйоном для клієнтів, які цінують зручність та швидкість у використанні електронних послуг. Вони забезпечують безперервний доступ до свого улюбленого магазину, роблячи процес покупок ще більш приємним та доступним.

Веб-додатки, на відміну від традиційних мобільних додатків, не вимагають встановлення на пристрої користувачів, що робить їх особливо привабливими. Це економить не лише час, а й обмежує необхідність великого обсягу внутрішньої пам'яті для зберігання додатків.

Завдяки використанню технологій, таких як Progressive Web Apps (PWA), веб-додатки можуть працювати в офлайн-режимі, забезпечуючи користувачів можливістю переглядати товари та здійснювати покупки навіть при відсутності інтернет-з'єднання. Це стає важливим аспектом для тих, хто подорожує або часто опиняється в умовах обмеженого доступу до мережі.[11]

Зростання популярності веб-додатків також пов'язане з їхньою можливістю працювати на різних операційних системах без значних модифікацій. Це полегшує розгортання та обслуговування, а також робить їх більш універсальними для широкого кола користувачів.

Застосування технологій штучного інтелекту та аналізу даних у веб-додатках дозволяє персоналізувати пропозиції та рекомендації для кожного

користувача, підвищуючи рівень задоволення від покупок та сприяючи збільшенню конверсії.

Отже, веб-додатки не лише забезпечують зручність та доступність, але і стають відмінним інструментом для вдосконалення користувацького досвіду та підвищення ефективності електронного бізнесу в цілому.

Мобільний додаток. У світі, де швидкість та мобільність грають ключову роль, мобільні додатки стають невід'ємною частиною сучасного електронного бізнесу. Вони виступають як важливий інструмент, забезпечуючи клієнтам доступ до магазину зі своїх смартфонів та планшетів. Мобільні додатки дарують користувачам не лише можливість швидко та зручно здійснювати покупки в будь-якому місці, але й володіють рядом додаткових функцій. Серед них – сповіщення про акції та знижки, персоналізовані рекомендації, а також зручний інтерфейс, оптимізований для використання на рухомих пристроях.[12]

Мобільні додатки стають вірними спутниками сучасних клієнтів, що цінують свою мобільність. Завдяки їм, вони можуть бути на зв'язку з улюбленим магазином у будь-який момент, отримуючи найсвіжіші новини, швидко реагуючи на акції та здійснюючи покупки в руху. Мобільні додатки не лише забезпечують доступ до електронного магазину, але і надають додатковий рівень зручності та персоналізації для задоволення потреб сучасних споживачів.

Мобільні додатки в сучасному електронному бізнесі також активно впроваджують технології розширеної реальності (Augmented Reality, AR) та віртуальної реальності (Virtual Reality, VR). Це дозволяє користувачам отримувати унікальний іммерсивний досвід під час онлайн-шопінгу. Наприклад, за допомогою AR можливо випробувати вироби або переглядати, як вони виглядатимуть в конкретному просторі перед покупкою. VR може створити віртуальні магазини зі своїми унікальними атмосферами та можливістю взаємодії з продуктами.[13]

Окрім того, застосування технологій штучного інтелекту в мобільних додатках дозволяє створювати персоналізовані пропозиції на основі покупкового

поведінки та інших даних користувачів. Аналіз великих обсягів даних допомагає підвищити ефективність маркетингових кампаній та забезпечує більш точне спрямування акцій на індивідуальних користувачів.

Також важливою складовою сучасних мобільних додатків є використання безпеки на вищому рівні. Застосування технологій шифрування та біометричних ідентифікаторів (відбитків пальців, розпізнавання обличчя) забезпечує захист конфіденційності даних користувачів та безпечні транзакції під час онлайн-покупок.

Усі ці інновації роблять мобільні додатки не лише інструментом для зручних покупок, але і захоплюючим інтерактивним середовищем, яке допомагає створювати унікальний та запам'ятовуваний досвід для клієнтів.

1.3 Перспективи розвитку сайту

У світі стрімкого технологічного прогресу, перспективи розвитку онлайн-магазину визначаються не лише його товарним асортиментом, але й цифровим середовищем, яке надається його веб-сайтом.

1. Мобільна оптимізація:

З урахуванням зростаючого використання смартфонів, наданням зручностей для мобільних користувачів стає важливою стратегією. Розвиток мобільної версії сайту та мобільного додатку дозволить залучити більше аудиторії та покращити їхній користувацький досвід.[14]

Вдосконалення мобільної версії сайту забезпечує не лише адаптацію контенту до різних розмірів екранів, але й оптимізацію швидкості завантаження. Це має важливе значення, оскільки швидкість реакції впливає на задоволеність користувачів та їхню готовність здійснювати покупки через мобільні платформи.

Успішна мобільна стратегія включає в себе не лише технічні аспекти, але й акцент на інновації в користувацькому інтерфейсі, персоналізованому маркетингу та використанні технологій розширеної реальності для збільшення залучення аудиторії.

2. Удосконалення функціональності:

Постійне вдосконалення функціональності сайту сприятиме полегшенню процесу покупок, введенню нових сервісів та опцій для клієнтів. Додаткові інструменти, такі як розширена фільтрація товарів чи персоналізовані рекомендації, можуть підняти рівень зручності та залучення користувачів.

Розширена фільтрація товарів є важливим елементом поліпшення функціональності, що сприяє зручності вибору для покупців. Можливість швидко та ефективно фільтрувати продукцію за різними параметрами дозволяє клієнтам знаходити необхідні товари швидше та легше.

Персоналізовані рекомендації – це ще один аспект, який може значно поліпшити користувацький досвід. Шляхом використання алгоритмів машинного навчання та аналізу покупкового поведінки клієнтів, електронні магазини можуть пропонувати індивідуально підібрані товари та послуги, що підвищує ймовірність успішних покупок.

Удосконалення функціональності сайту – це не лише стратегічний хід для залучення нових клієнтів, але і інструмент для утримання існуючої аудиторії. Все більше користувачів вимагають не лише широкого асортименту, але й інтуїтивно зрозумілої та ефективної платформи для їхніх онлайн-покупок.[15]

3. Розвиток системи доставки та сплати:

Стратегічний розвиток системи доставки та оплати є ключовим фактором для подальшого успіху електронного магазину. Це включає в себе не лише підвищення зручності для клієнтів, а й підвищення ефективності процесів та конкурентоспроможності бізнесу в цілому.

Впровадження нових варіантів доставки – це стратегічний крок, який може відзначити електронний магазин серед конкурентів. Вибір між різними способами доставки, такими як експрес-доставка, самовивіз або пункти видачі, надає клієнтам можливість обирати той варіант, який найбільше відповідає їхнім потребам та графіку.

Оптимізація часу доставки – це ще один аспект, який має значний вплив на задоволеність клієнтів. Зменшення часу між моментом замовлення та його отриманням може бути вирішальним фактором у виборі покупця.

Розширення способів оплати – це також стратегічна ініціатива, яка спрощує процес покупок для клієнтів. Забезпечення різноманіття варіантів оплати, таких як кредитні картки, електронні гроші чи платіжні системи, дозволяє покупцям вибирати той метод, який найбільше їм підходить.

В цілому, розвиток системи доставки та оплати є невід'ємною частиною стратегії електронного магазину, спрямованої на створення оптимального та зручного середовища для покупок, що веде до задоволення клієнтів та підвищення його конкурентоспроможності на ринку.

4. Експансія асортименту:

Розширення товарного асортименту відповідно до змін у попиті споживачів та ринкових тенденцій дозволить магазину залишатися актуальним та цікавим для своєї аудиторії. Введення нових категорій товарів чи брендів розширить можливості для покупців.

Стратегічне розширення товарного асортименту є суттєвим елементом успішної діяльності електронного магазину. Зміни в попиті споживачів та динаміка ринкових тенденцій створюють унікальні можливості для бізнесу не лише відповідати на зміни, але й визначати їх. Введення нових категорій товарів чи брендів відкриває широкі горизонти для покупців, створюючи для них додаткові варіанти вибору. Це не лише збільшує кількість доступних продуктів, але й дозволяє магазину відзначитися своєю унікальністю та інноваційністю.

Підтримка актуальності та цікавості для аудиторії стає ключовою умовою у конкурентному середовищі електронної комерції. Оновлення товарного асортименту відповідно до динаміки ринку дозволяє магазинам ефективно конкурувати та привертати нових клієнтів. Стратегічне планування включає в себе аналіз попиту, вивчення поведінки споживачів та прогнозування трендів, що дозволяє магазинам адаптуватися та вносити необхідні зміни у свій асортимент.

Такий підхід не лише забезпечує конкурентоспроможність, але й сприяє створенню витонченого та персоналізованого досвіду для кожного клієнта.[16]

Розширення товарного асортименту в електронній комерції визначається не лише кількістю, але і якістю продуктів, що пропонуються, що дозволяє магазинам активно взаємодіяти з ринком та впевнено рухатися вперед у світлі постійних змін і викликів.

5. Соціальна взаємодія та загальнотематичні акції:

Використання соціальних мереж та проведення загальнотематичних акцій сприяє залученню нових клієнтів та підтримці існуючої аудиторії. Інтерактивність та участь у великих подіях можуть позитивно вплинути на імідж та популярність магазину.

Споживачі активно перебувають в соціальних мережах, і електронні магазини можуть використовувати цей канал для підвищення своєї видимості та привертання нових клієнтів. Платформи, такі як Instagram, Facebook, Twitter, надають можливості для реклами, взаємодії та створення унікального контенту, що привертає увагу цільової аудиторії.

Соціальні мережі також є ідеальним майданчиком для підтримки вже існуючих клієнтів. Взаємодія через коментарі, особисті повідомлення та обговорення може покращити відносини з покупцями. Ретельне слідкування за фідбеком та реакцією на запитання або скарги створює позитивне сприйняття магазину серед споживачів.

Проведення загальнотематичних акцій, конкурсів чи розіграшів через соціальні мережі сприяє взаємодії з аудиторією. Креативний та зацікавлюючий контент може розширити охоплення та створити позитивне сприйняття магазину. Залучення споживачів до участі в подіях часто збільшує їхню лояльність та залученість. Все це разом формує ефективну стратегію соціальної взаємодії та проведення акцій, що додає важливий аспект до успішного функціонування електронного магазину в умовах сучасного ринку електронної комерції.

Усі ці напрямки розвитку взаємодіють, сприяючи створенню цілісного та сучасного цифрового середовища для клієнтів. Збереження відкритості до інновацій та активне апгрейдування веб-сайту дозволять забезпечити йому стабільний розвиток та високий рівень конкурентоспроможності на ринку електронної комерції.

Розвиток веб-сайту для онлайн-магазину необхідний з кількох причин. Це ключовий елемент стратегії, спрямованої на успішну експлуатацію в умовах електронної комерції.[17]

Важливість постійного вдосконалення веб-сайту полягає в:

- Збереженні Конкурентоспроможності: Актуальний та функціональний веб-сайт допомагає магазину залишатися конкурентоспроможним. Регулярне оновлення та впровадження нових можливостей привертають та утримують клієнтів.
- Покращенні Взаємодії з Клієнтами: Сучасні функції та інтерактивність веб-сайту сприяють зручності та задоволенню клієнтів. Це важливо для збереження та залучення нових покупців.
- Пристосуванні до Ринкових Тенденцій: Зміни в електронній комерції та споживацьких практиках вимагають постійного апгрейду. Розвиток відповідає на нові вимоги ринку та сприяє адаптації до змін.
- Підвищенні Іміджу Бренду: Модернізація веб-сайту сприяє створенню сучасного та привабливого іміджу бренду. Це робить магазин більш привабливим для клієнтів та партнерів.

Електронні магазини повинні постійно адаптуватися до змін, щоб залишатися конкурентоспроможними та забезпечувати задоволення клієнтів. Однією з ключових аспектів розвитку є створення цифрового середовища, яке буде не лише зручним для покупців, але й відповідатиме останнім технологічним тенденціям.

У цьому контексті, мобільна оптимізація стає стратегічно важливою. З ростом використання смартфонів важливо забезпечити зручність для мобільних

користувачів. Розробка мобільної версії веб-сайту та мобільного додатку дозволяє залучити більше аудиторії та покращити користувацький досвід.

До того ж, постійне вдосконалення функціональності веб-сайту грає важливу роль у спрощенні процесу покупок та наданні нових сервісів для клієнтів. Розширена фільтрація товарів, персоналізовані рекомендації та інші інструменти можуть значно поліпшити рівень зручності та взаємодії.

Окрім цього, розвиток системи доставки та оплати грає ключову роль у покращенні процесу покупок для клієнтів. Введення нових варіантів доставки, оптимізація часу доставки та розширення способів оплати може позитивно позначитися на конкурентоспроможності магазину.

Однак, розвиток електронного магазину не обмежується лише технічними аспектами. Застосування інновацій у маркетингових стратегіях також відіграє важливу роль. Використання аналітики та великих обсягів даних дозволяє легше розуміти поведінку покупців та прогнозувати їхні вимоги. Зокрема, персоналізований маркетинг, голосовий пошук, інтеграція з іншими платформами та використання технологій віртуальної примірки розширюють можливості електронного магазину та роблять його привабливим для сучасного споживача.

У контексті постійного розвитку електронного магазину, активна інтеграція технологій Інтернету речей (IoT) може стати важливим етапом. Завдяки цьому, магазин може оптимізувати управління запасами, автоматизувати бізнес-процеси та надавати клієнтам персоналізовані послуги на основі аналізу зібраних даних.[18]

У світлі стрімкого росту застосування штучного інтелекту, важливо враховувати можливості використання цієї технології для поліпшення клієнтського сервісу. Чат-боти, персоналізовані рекомендації та аналіз відгуків можуть значно покращити взаємодію з покупцями, роблячи процес покупок більш ефективним та задовільним.

Аналітика та використання великих обсягів даних можуть бути не лише інструментами для розуміння поведінки клієнтів, але й для прогнозування

тенденцій ринку. Ефективне використання аналітики дозволяє адаптуватися до змін в ринкових умовах та підтримувати стратегії магазину на високому рівні конкурентоспроможності. Використання технології блокчейн може відігравати ключову роль у покращенні безпеки та прозорості у електронному магазині, забезпечуючи надійність та цілісність транзакцій, що особливо важливо у сфері онлайн-платежів.

Приділення уваги екологічним питанням та соціальній відповідальності також може зробити магазин більш привабливим для сучасних споживачів, які докладають зусиль до вибору екологічно чистих та етичних брендів.

Усі ці напрямки розвитку, взаємодіючи між собою, формують інноваційне та гнучке цифрове середовище для клієнтів електронного магазину. Співробітництво з новітніми технологіями та постійне апгрейдування бізнес-процесів дозволять магазину не лише виживати в умовах швидких змін, але й тримати провідні позиції в електронній комерції.

1.4 Актуальні проблеми в галузі

Актуальні проблеми в галузі є ключовим елементом розділу, який спрямований на визначення та розкриття викликів, що стоять перед сучасними проектами у сфері дослідження. В ході аналізу будуть розглянуті найбільш пильні теми та проблеми, які визначають контекст розробки та впливають на подальший розвиток та вдосконалення інформаційних систем.

Розгляд актуальних викликів у галузі визначає необхідність подальших досліджень та розвитку, а також обґрунтовує вибір обраного проекту як важливого внеску у вирішення цих проблем. Приділення уваги найбільш актуальним темам дозволить ефективно спрямувати зусилля на розв'язання проблем, що мають визначальне значення для розвитку предметної області.

У сучасному середовищі електронної комерції та цифрових сервісів існує ряд актуальних проблем, які визначають необхідність подальших досліджень та вдосконалення. Деякі з них включають:

Безпека та конфіденційність. Безпека та конфіденційність є одними з найбільш критичних аспектів в електронній комерції та цифрових сервісах. Забезпечення високого рівня безпеки є обов'язковим завданням для будь-якої онлайн-платформи. Це стосується як особистої інформації користувачів, так і фінансових транзакцій. Однією з ключових вимог є ефективне шифрування даних, особливо під час передачі через Інтернет. Технології шифрування, такі як SSL (Secure Sockets Layer) та його спадкоємець TLS (Transport Layer Security), гарантують захищений обмін інформацією між користувачем і сервером.[19]

Крім того, системи автентифікації та авторизації грають важливу роль у забезпеченні безпеки. Застосування двофакторної аутентифікації, складних паролів та інших методів перевірки особи допомагає запобігти несанкціонованому доступу до особистого кабінету користувача.

Окрім цього, важливо регулярно оновлювати програмне забезпечення та застосовувати патчі для усунення вразливостей. Постійний моніторинг системи та виявлення можливих загроз є частиною стратегії забезпечення безпеки. Щодо конфіденційності, важливо визначити, як інформація зберігається, обробляється та передається. Це може включати застосування політик конфіденційності, обмеження доступу до конфіденційних даних, а також регулярні аудити для визначення відповідності стандартам безпеки.

Ефективність маркетингу. Ефективність маркетингу та реклами в електронній комерції та цифрових сервісах визначається низкою факторів, які взаємодіють для привертання уваги аудиторії та забезпечення конверсії. У сучасному бізнес-середовищі важливо використовувати стратегії маркетингу, що відповідають вимогам та змінам у споживацькому поведінці.[20]

Один із ключових елементів — це персоналізація маркетингових кампаній. Аналіз даних про покупців дозволяє створювати індивідуалізовані пропозиції та рекомендації, забезпечуючи більший рівень залучення та задоволення клієнтів. Основна увага також приділяється використанню соціальних мереж як ефективного інструменту реклами та взаємодії з аудиторією. Рекламні кампанії на

платформах, таких як Facebook, Instagram чи Twitter, можуть значно збільшити обсяги продажів та покращити взаємодію бренду зі споживачами.

Також важливо розвивати мобільний маркетинг, оскільки все більше користувачів здійснюють покупки через мобільні пристрої. Мобільні додатки та адаптивні веб-сайти дозволяють надавати персоналізований досвід та ефективно взаємодіяти з аудиторією в режимі реального часу. Спеціальні промоакції, знижки та лояльність також є важливими стратегіями для привертання нових клієнтів та утримання існуючих. Інтеграція із системами електронної комерції та використання аналітики дозволяють оцінювати ефективність кампаній та вносити необхідні корективи.

Узагальнюючи, ефективність маркетингу та реклами в електронній комерції ґрунтується на глибокому розумінні цільової аудиторії, використанні різноманітних каналів залучення та постійному вдосконаленні стратегій на основі аналізу результатів.

Лояльність та утримання клієнтів. Лояльність та утримання клієнтів стали критичними аспектами в електронній комерції, де конкуренція велика, а можливості перейти до іншого бренду завжди на вагу. Утримання клієнтів вимагає не лише залучення нових, але й створення позитивного досвіду для існуючих.

Один з ключових елементів цього процесу — це персоналізація обслуговування. Споживачі цінують індивідуальний підхід та особисті підходи. Адаптація рекламних кампаній, знижок та спеціальних пропозицій до індивідуальних потреб клієнтів дозволяє підтримувати високий рівень лояльності. Програми лояльності, що надають знижки, бонуси чи ексклюзивні привілеї, стають суттєвим стимулом для повторних покупок. Важливо вдосконалювати такі програми, адаптуючи їх до змін у споживацькому підході та враховуючи змінюючіться умови ринку.

Слід також визнати значення високої якості обслуговування клієнтів. Швидке та ефективне вирішення проблем, оперативний відгук на запитання та

запити, а також позитивний діалог в соціальних мережах допомагають створювати позитивний імідж бренду.

Утримання клієнтів також пов'язане із створенням якісного контенту та інформаційної цінності. Це може включати блоги, статті, відео, які допомагають клієнтам зрозуміти вироби чи послуги, а також надають корисні поради та інсайти.

Узагальнюючи, лояльність та утримання клієнтів в електронній комерції вимагають комплексного підходу, орієнтованого на індивідуальні потреби клієнтів, створення позитивного враження та підтримку довгострокових відносин.

Технологічна інфраструктура. Технологічна інфраструктура є розгалуженою та невід'ємною частиною електронної комерції. У сучасному бізнесі та торгівлі важко уявити успішну діяльність без ефективної технологічної основи. Ця інфраструктура охоплює різноманітні компоненти, спрямовані на підтримку різних аспектів електронного бізнесу.

Системи управління контентом (CMS) відіграють важливу роль у розміщенні та оновленні вмісту на веб-сайтах електронних магазинів. CMS надає зручний інтерфейс для додавання нових товарів, описів, зображень та інших важливих даних, що полегшує процес управління вмістом.[21]

Інтернет-хмарні рішення використовуються для забезпечення надійності та доступності веб-сайтів. Це включає в себе хмарні хостингові платформи, які забезпечують стабільну роботу віртуальних серверів та зменшують час відновлення в разі виникнення проблем. Електронні платіжні системи та шлюзи є ключовим компонентом для здійснення безпечних та зручних транзакцій в електронному бізнесі. Ці системи інтегруються з веб-сайтами для обробки платежів, підтримки різних методів оплати та забезпечення безпеки фінансових операцій.

Аналітичні інструменти та системи бізнес-аналітики допомагають власникам електронних магазинів збирати та аналізувати дані про поведінку клієнтів, ефективність продажів та інші ключові метрики. Це дозволяє приймати

обґрунтовані рішення для оптимізації бізнес-процесів та підвищення результативності. Інтеграція з соціальними мережами та іншими маркетинговими каналами є важливим елементом для розширення аудиторії та залучення нових клієнтів. Забезпечення простоти взаємодії з різними медіа-платформами сприяє підвищенню обізнаності та популярності бренду.

Узагальнюючи, технологічна інфраструктура є невід'ємною частиною успішного функціонування електронної комерції, надаючи потужні та інноваційні інструменти для підтримки бізнес-процесів та забезпечення задоволення клієнтів.

Гнучкість та адаптивність. Гнучкість та адаптивність в електронній комерції визначають спроможність бізнесу ефективно реагувати на зміни в ринкових умовах, технологічному прогресі та вимогах споживачів. У контексті електронної комерції, гнучкість означає здатність швидко змінювати стратегії, процеси та пропозиції відповідно до ринкових тенденцій та змін у споживацьких уподобаннях. Гнучкий електронний бізнес може легко адаптуватися до нових умов, вчасно впроваджувати інновації та швидко реагувати на конкурентні виклики.

Адаптивність, з іншого боку, вказує на здатність електронного бізнесу пристосовуватися до різноманітних умов і змінюватися відповідно до нових вимог. Це може включати в себе гнучкі стратегії маркетингу, зміни в продуктовому портфелі, адаптацію до технологічних інновацій та вдосконалення бізнес-процесів.

Гнучкість та адаптивність є ключовими факторами успіху в електронній комерції, особливо в умовах швидко змінюючогося економічного та технологічного середовища. Підприємства, які здатні ефективно адаптуватися до нових умов та вчасно реагувати на виклики, можуть забезпечити стабільний розвиток та зберегти конкурентні переваги.

2 АНАЛІЗИ ТА ШЛЯХИ РОЗВИТКУ ПРОБЛЕМИ

2.1 Інструментальні засоби та технології створення

2.1.1 Мова програмування TypeScript у контексті Node.js

JavaScript та його розширення TypeScript виявляються потужними інструментами для розробки серверної частини додатків у середовищі Node.js. TypeScript, який є суперсетом JavaScript з додатковими можливостями статичного типізації, дозволяє розробникам створювати більш безпечний та читабельний код.[22]

Особливості JavaScript та TypeScript у контексті Node.js

1. Асинхронна Неблокуюча Модель:

Мова програмування JavaScript і TypeScript підтримують асинхронну та неблокуючу модель програмування. Це дозволяє ефективно обробляти багато операцій одночасно та реагувати на події без заблокування виконання коду.

2. Використання Callbacks та Promises:

В обидві мови використовуються callback-функції та Promises для роботи з асинхронним кодом. Це робить код більш зручним та читабельним, сприяючи уникненню пасток "callback hell".

3. Модульність та Екосистема:

JavaScript та TypeScript базуються на модульній системі, що дозволяє розробникам використовувати різноманітні пакети та бібліотеки. Node Package Manager (NPM) стає потужною екосистемою для управління залежностями та розширення функціональності.

4. Типізація в TypeScript:

TypeScript додає до JavaScript статичну типізацію, що полегшує виявлення та усунення помилок на етапі розробки. Це робить код більш надійним та сприяє підтримці великих проектів.

5. Розширена Підтримка Об'єктно-Орієнтованого Програмування:

TypeScript надає більше можливостей для об'єктно-орієнтованого програмування, включаючи підтримку класів, інтерфейсів та модифікаторів доступу.

6. Підтримка Сучасних Версій ECMAScript:

Як розширення JavaScript, TypeScript підтримує сучасні версії ECMAScript, що дозволяє використовувати нові функції та покращення мови.

Node.js — це відкрите серверне середовище виконання, яке дозволяє використовувати мову програмування JavaScript для створення ефективних та масштабованих серверних додатків. Однією з ключових особливостей Node.js є його асинхронна та неблокуюча модель, яка дозволяє обробляти багато операцій одночасно без блокування виконання коду. Node.js був розроблений з врахуванням потреб веб-розробників, які використовують JavaScript для розробки клієнтської частини веб-застосунків. Завдяки Node.js цю мову можна використовувати і на серверній стороні, що створює єдиний стек технологій для розробки.[23]

Однією з переваг Node.js є його висока швидкість виконання завдяки використанню движка V8 від Google. Це робить його ідеальним вибором для розробки застосунків, які вимагають обробки великої кількості одночасних підключень, таких як чат-сервери, стрімінгові сервіси або застосунки реального часу. За допомогою Node.js розробники можуть ефективно створювати серверні додатки, обробляти запити, взаємодіяти з базами даних та іншими ресурсами. Інтеграція з платформою npm (Node Package Manager) робить його потужним інструментом для управління залежностями та розширення функціональності.

Однією з ключових областей використання Node.js є розробка API для взаємодії між клієнтською та серверною частинами веб-застосунків. Node.js забезпечує можливість швидкої та ефективної обробки запитів, що робить його ідеальним вибором для створення API. Створення API за допомогою Node.js дозволяє розробникам легко взаємодіяти з базами даних, обробляти запити від клієнтів та надавати їм необхідні дані. Асинхронний характер Node.js виявляється

особливо корисним при обробці багатьох паралельних запитів, що є частим сценарієм у роботі з API.

Використання TypeScript у поєднанні з Node.js надає додаткові переваги в розробці. TypeScript дозволяє визначати статичні типи даних, що полегшує виявлення та усунення помилок ще на етапі розробки. Це стає особливо важливим у великих проектах, де підтримка коду та його рефакторинг вирішально впливають на продуктивність розробки.

За допомогою TypeScript можна визначити інтерфейси, використовувати класи та модулі, що сприяє покращенню структури коду та забезпеченню більшої читабельності. Крім того, TypeScript легко інтегрується з іншими інструментами розробки, забезпечуючи розробникам зручний інструментарій.

2.1.2 Середовище розробки JetBrains WebStorm

Розробка веб-додатків вимагає потужного та зручного середовища розробки, яке враховує особливості мов програмування та технологій, використовуваних у проекті. Один із таких інструментів - JetBrains WebStorm, який забезпечує найвищий рівень продуктивності та зручності для розробників TypeScript та Node.js (рис. 2.1).

JetBrains WebStorm був представлений компанією JetBrains як спеціалізоване інтегроване середовище розробки для веб-проектів. Відтоді він зазнав значних покращень та адаптацій, щоб відповідати зростаючим вимогам веб-розробки.

1. Інтеграція з TypeScript: WebStorm забезпечує відмінну інтеграцію з TypeScript, що дозволяє з легкістю використовувати всі переваги цієї мови програмування. Від автодоповнення до аналізу коду - всі ці функції роблять розробку в TypeScript більш приємною та продуктивною.

2. Підтримка Node.js: Середовище WebStorm також повністю орієнтоване на підтримку Node.js. Інтегровані інструменти для відлагодження,

автоматичного перезавантаження сервера та аналізу продуктивності роблять розробку серверних застосунків на Node.js простішою та ефективнішою.

3. Автоматизація Завдань: WebStorm включає широкий спектр інструментів для автоматизації рутинних завдань. Від автоматичної компіляції TypeScript до конфігурації системи збірки - ці можливості роблять розробку більш ефективною.

4. Вбудована Підтримка Git: Інтеграція з системою контролю версій Git дозволяє командам легко спільпрацювати та ведення історії змін. WebStorm надає зручний інтерфейс для роботи з гілками, злиття та іншими Git-операціями.

5. Аналіз Та Відлагодження: WebStorm має вбудовані інструменти для аналізу коду та відлагодження, що значно полегшує виявлення та усунення помилок. Розширені можливості відлагодження допомагають вивчати роботу коду на будь-якому етапі розробки.

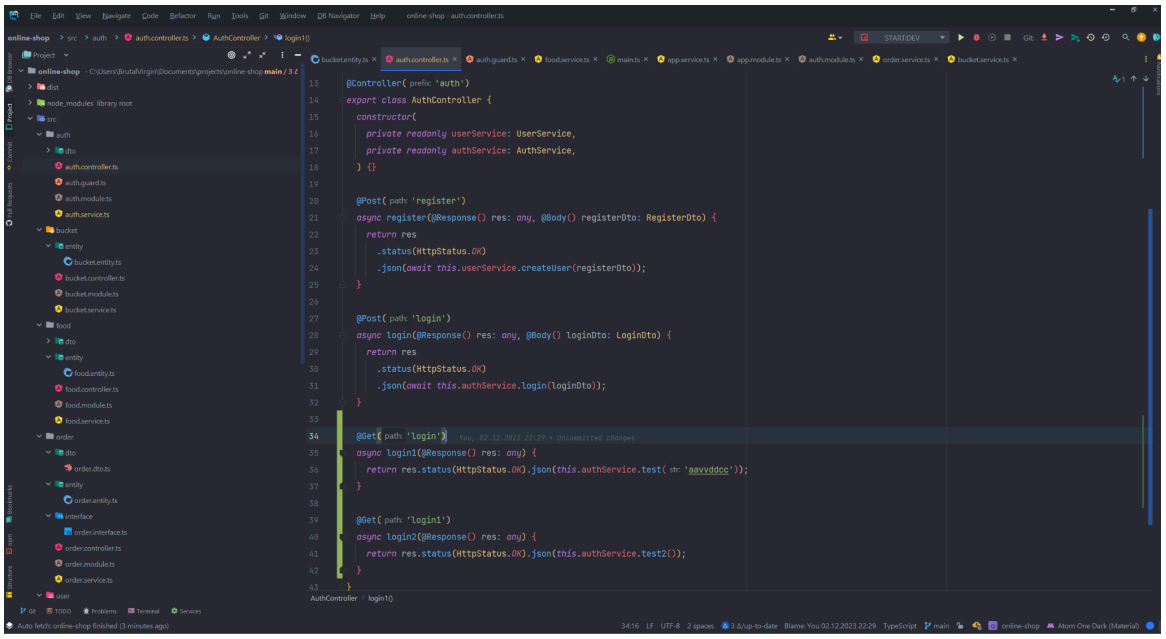


Рисунок 2.1 - Інтерфейс JetBrains WebStorm

2.1.3 СУБД PostgreSql

PostgreSQL, часто визначена як "Postgres," є відкритою та безкоштовною системою управління базами даних (СУБД), яка здобула широку популярність у

розробників та підприємств. Її висока надійність, масштабованість та розширюваність роблять її привабливим вибором для різноманітних проектів.

PostgreSQL була розроблена в університеті Каліфорнії в Берклі в 1986 році і була базовою системою для проекту Ingres. З тих пір вона пройшла значний шлях розвитку та стала однією з найпотужніших та найбільш функціональних СУБД. Принципові характеристики PostgreSQL включають високий рівень стандартності SQL, підтримку ACID (Atomicity, Consistency, Isolation, Durability), а також розширені можливості роботи зі складними типами даних, такими як JSON та XML. [24]

ACID є абревіатурою, що означає чотири основні властивості транзакцій у базі даних: Atomicity (Атомарність), Consistency (Узгодженість), Isolation (Ізоляція) та Durability (Тривалість). Ці принципи є важливими для забезпечення надійності та цілісності даних.[25]

1. Атомарність (Atomicity): Цей принцип гарантує, що транзакції виконуються атомарно, тобто як єдиний, недільний блок. Якщо яка-небудь частина транзакції не може бути виконана, вся транзакція відміняється, і база даних залишається у стані, якщо нічого не трапилося.

2. Узгодженість (Consistency): Цей принцип забезпечує, що транзакція переводить базу даних з одного стабільного стану в інший. Тобто, якщо база даних була у коректному стані перед транзакцією, вона залишається коректною після транзакції.

3. Ізоляція (Isolation): Принцип ізоляції гарантує, що виконання однієї транзакції не впливає на інші транзакції, які відбуваються одночасно. Це забезпечує віртуальне виключення транзакцій одна від одної, щоб уникнути взаємного впливу.

4. Тривалість (Durability): Цей принцип гарантує, що зміни, внесені в базу даних після успішного завершення транзакції, залишаються незмінними і не втрачаються. Це означає, що дані залишаються стійкими навіть у випадку аварії або перезавантаження системи.

Використання ACID-властивостей забезпечує надійність бази даних та виключає ймовірність втрати або пошкодження даних під час обробки транзакцій, роблячи PostgreSQL відмінним вибором для додатків, де критично важлива цілісність та надійність даних.

PostgreSQL має такі можливості:

- Розширена Підтримка SQL: PostgreSQL надає багатий набір функцій SQL, що включає в себе велику кількість операцій та функцій для обробки та аналізу даних.
- Масштабованість та Розширюваність: Здатність працювати з великими обсягами даних і легко розширюватися роблять PostgreSQL ідеальним вибором для проектів різного масштабу.
- JSON та XML Підтримка: Вбудована підтримка для роботи з документ-орієнтованими данними у форматах JSON та XML дозволяє зберігати та опрацьовувати неструктуровані дані.
- Розширюваність за Допомогою Розширень: PostgreSQL дозволяє створювати власні розширення та функції, розширюючи функціональність за потребою конкретного проекту.

PostgreSQL використовується у різноманітних галузях, включаючи веб-розробку, телекомунікації, фінанси та багато інших. Велика спільнота розробників по всьому світу активно підтримує та розвиває цю систему. PostgreSQL продовжує залишатися важливим інструментом для тих, хто цінує надійність та гнучкість у роботі з базами даних. Її активна спільнота та постійні оновлення роблять PostgreSQL перспективним вибором для різноманітних проектів та завдань у сфері розробки програмного забезпечення.

2.1.4 OPM Typeorm для PostgreSql

OPM є технологічним підходом до взаємодії з базами даних, де дані в базі представлені об'єктами програмного коду. Замість традиційних SQL-запитів

розробник використовує об'єктно-орієнтований код для маніпулювання даними в базі.

ОРМ дозволяє взаємодіяти з базою даних, використовуючи об'єктно-орієнтовані концепції, що робить код більш зрозумілим та зменшує кількість необхідних SQL-запитів. Роблячи взаємодію з базою даних більш абстрактною, ОРМ приховує деталі роботи з конкретною СУБД (системою управління базами даних), що дозволяє легше мігрувати або змінювати СУБД без значних змін у коді. ОРМ автоматизує багато рутинних операцій, таких як створення таблиць, взаємодія з записами та керування зв'язками між таблицями (рис. 2.2).

```
// Використання TypeORM
await userRepository.update(userId, {
  email: newEmail,
  username: newUsername,
});

// Використання raw SQL
await pool.query(
  'UPDATE users SET email = $1, username = $2 WHERE id = $3 RETURNING *
  [newEmail, newUsername, userId],
);
```

Рисунок 2.2 - Typeorm запит і чистий SQL

Як можна побачити, використання TypeORM дозволяє розробнику писати менше коду для досягнення того самого результату порівняно з справжнім SQL. Наприклад, для оновлення даних користувача код стає більш компактним і зрозумілим.[26]

TypeORM вигідно виділяється завдяки своїй здатності автоматично генерувати SQL-запити для багатьох типових операцій, таких як вставка, оновлення та видалення даних. Це зменшує кількість рутинної роботи для розробника та спрощує взаємодію з базою даних. Крім того, використання TypeORM дозволяє отримати переваги TypeScript, надаючи типізацію та

інтеграцію з іншими функціональностями мови. Це робить код більш стійким до помилок та зручним для обслуговування.

Однією з важливих особливостей TypeORM є підтримка різних систем управління базами даних, що робить його гнучким інструментом для роботи з різноманітними проектами. Зручний синтаксис та активне співтовариство розробників роблять TypeORM популярним вибором для розробки додатків, особливо тих, які використовують TypeScript та JavaScript.

Основні Ідеї ORM:

- Сутності: Об'єкти, що представляють сутності бази даних, такі як користувачі, товари чи замовлення.
- Зв'язки: Взаємозв'язки між сутностями, які визначають, як об'єкти пов'язані один з одним.
- Автоматична Генерація SQL: ORM автоматично генерує SQL-запити для виконання операцій з базою даних, що дозволяє розробнику уникати написання великої кількості SQL-коду.

Для цього проекту була вибрана TypeORM. TypeORM є одним із популярних ORM-фреймворків, призначених для роботи з TypeScript та JavaScript. Його створив і утримує команда розробників з Node.js. Перша версія була випущена у 2014 році, і з того часу TypeORM набув значної популярності завдяки своїй зручності та потужним можливостям.

Основні Ідеї та Особливості TypeORM:

1. Підтримка TypeScript:TypeORM спрощує взаємодію з базою даних під час розробки на TypeScript, надаючи типізацію та інтеграцію з іншими функціональностями мови.
2. Підтримка Різних СУБД: TypeORM підтримує різні системи управління базами даних, такі як PostgreSQL, MySQL, SQLite та багато інших, що робить його універсальним інструментом.

3. Зручний Синтаксис: Лаконічний та зрозумілий синтаксис TypeORM дозволяє легко взаємодіяти з базою даних, створюючи абстракцію над SQL-запитами.

4. Міграції та Генерація Схеми: TypeORM дозволяє здійснювати міграції для зручного керування змінами в базі даних та автоматично генерує схему бази даних з існуючого коду.

5. Активне Співтовариство: Багата документація та активна спільнота розробників роблять TypeORM дуже популярним інструментом, який постійно вдосконалюється та оновлюється.

TypeORM використовується у широкому спектрі проектів, від невеликих веб-додатків до великих корпоративних систем. Його зручний інтерфейс та гнучкість дозволяють розробникам ефективно працювати з базами даних, незалежно від їхнього розміру та складності.

Міграції та збереження схеми є ключовою частиною використання ОРМ TypeORM для PostgreSQL:

Міграції в ОРМ представляють собою спосіб автоматизованого внесення змін у структуру бази даних з плином часу. Зазвичай, проект розвивається, і може знадобитися додавання нових таблиць, полів чи внесення інших змін в базу даних.

TypeORM дозволяє створювати міграції, які автоматично визначають та виконують необхідні SQL-запити для внесення змін. Це дозволяє легко керувати розвитком бази даних без необхідності вручну писати SQL-запити для кожної зміни.

Збереження схеми - це процес, який дозволяє зберегти структуру бази даних та всі визначені в неї зв'язки та обмеження. Збереження схеми важливо для того, щоб гарантувати цілісність даних та забезпечити, що вони відповідають поточним вимогам системи. TypeORM допомагає автоматизувати цей процес, надаючи інструменти для генерації та збереження схеми бази даних. Це забезпечує консистентність структури даних та дозволяє легко керувати розвитком проекту.

Інтеграція міграцій та збереження схеми в TypeORM дозволяє ефективно впроваджувати нові функції та зміни в проекті, зберігаючи при цьому надійність та цілісність бази даних.

2.1.5 Postman для автоматизації тестування

В сучасному програмуванні і розробці програмного забезпечення, автоматизація тестування є ключовим елементом для забезпечення високої якості продукту. Один із ефективних інструментів для цього — Postman.

Postman — це набір інструментів для розробників, який дозволяє автоматизувати відправку запитів, тестувати API, та перевіряти функціональність веб-додатків. Його основна перевага полягає в можливості створення та виконання автоматизованих тестів для перевірки працездатності API. За допомогою Postman можна легко встановлювати параметри запитів, включаючи заголовки, параметри шляху та тіло запиту. Основною перевагою використання Postman є можливість створювати колекції тестів, що дозволяє групувати та організовувати їх для ефективного використання та редагування.[27]

Postman підтримує автоматизоване тестування різних видів запитів, включаючи GET, POST, PUT та DELETE (рис 2.3). Це дозволяє розробникам швидко перевіряти функціональність їхніх API та впевнено впроваджувати зміни без ризику порушення роботи системи.

Покрокова документація та інтуїтивний інтерфейс Postman роблять його ідеальним інструментом для тестування та валідації API. Такий підхід дозволяє зекономити час розробникам та забезпечити стабільність роботи програмного забезпечення.

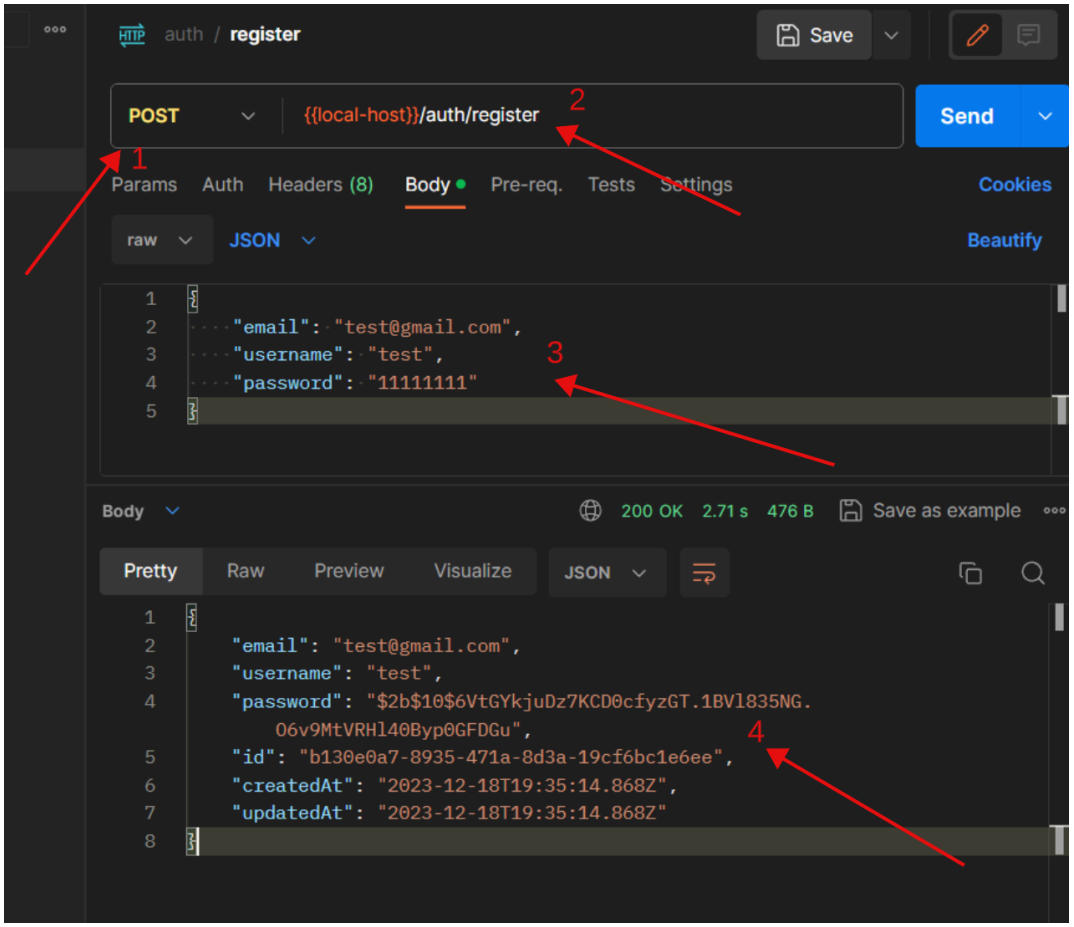


Рисунок 2.3 - Інтерфейс Postman

де 1 - HTTP метод;

2 - URL запиту;

3 - Body запиту

4 - Відповідь серверу

Основні HTTP методи включають:

1. GET: Використовується для отримання інформації з сервера. Запитує вказану ресурс і повертає його клієнту.

2. POST: Використовується для надсилання даних на сервер для обробки або зберігання. Зазвичай використовується для відправлення даних форми на сервер.

3. PUT: Використовується для оновлення інформації на сервері або створення нового ресурсу, якщо він не існує.

4. DELETE: Використовується для видалення вказаного ресурсу на сервері.

5. PATCH: Використовується для часткового оновлення ресурсу на сервері. Відправляє тільки ті дані, які потрібно змінити.

6. OPTIONS: Використовується для отримання інформації про можливості сервера або параметри з'єднання для конкретного ресурсу.

HTTP методи визначають, як клієнт і сервер взаємодіють між собою, дозволяючи виконувати різні дії та операції з веб-ресурсами.[28]

Тіло запиту (request body) - це частина HTTP-запиту, яка містить дані, що відправляються на сервер. Зазвичай це відбувається у вигляді тексту, але формат даних може варіюватися. Один з найпоширеніших форматів для представлення даних в тілі запиту є JSON (JavaScript Object Notation).

JSON - це легкочитабельний формат обміну даними, який використовується для передачі структурованих даних між сервером і клієнтом. Він використовується для представлення об'єктів та списків у вигляді тексту, що легко інтерпретується як мовою програмування, так і людиною. При використанні методів POST, PUT, PATCH або інших, які передбачають надсилання даних на сервер, відправка цих даних відбувається у тілі запиту. Тіло запиту може містити будь-яку структуровану інформацію, таку як об'єкти, масиви, числа, рядки тощо, у форматі JSON.

Відповідь серверу - це частина HTTP-запиту, яка передається від сервера до клієнта як відповідь на запит, зроблений клієнтом. Вона містить інформацію, яку сервер надсилає назад, включаючи статус виконання запиту, заголовки та, в більшості випадків, тіло відповіді, яке містить конкретні дані або ресурс.

Відповідь серверу зазвичай складається з трьох основних компонентів:

- Статус коду (Status Code): Це числовий код, який вказує на статус виконання запиту. Наприклад, код 200 означає успішний запит, 404 - не знайдено, 500 - внутрішня помилка сервера тощо.
- Заголовки (Headers): Це метадані, які містять різноманітну інформацію про відповідь, таку як тип вмісту, дата відправки, тип кешування тощо.
- Тіло відповіді (Response Body): Це фактична інформація, яка повертається сервером у відповідь на запит. Формат цього тіла може бути різноманітним і залежить від конкретного типу запиту і призначення сервера.

JWT (JSON Web Token) - це стандарт для передачі у претензіях між двома сторонами за допомогою об'єкта JSON. Він складається з заголовка, корисної навантаження та підпису. JWT може бути підписаним за допомогою секретного ключа або використовувати пару ключів для підпису/перевірки. JWT використовується для передачі автентифікаційної інформації між сторонами в компактному та самостійному способі. Це дозволяє сторонам перевірити автентичність переданих даних.

JWT зазвичай складається з трьох частин:

1. Заголовок (Header): Вказує тип та алгоритм підпису, використаного для створення токену.
2. Корисна навантаження (Payload): Містить дані.
3. Підпис (Signature): Це результат підпису, який генерується на основі заголовка та корисного навантаження за допомогою секретного ключа.

В контексті автентифікації та авторизації, JWT може бути використаний для передачі інформації про користувача, ролі та інші претензії в безпечний спосіб.

У Postman, при використанні вкладки Auth (рис. 2.4), JWT токен може бути доданий як метод автентифікації для використання у запитах. Користувач може вказати свій JWT токен та його параметри (наприклад, тип алгоритму підпису) для валідації та передачі у заголовках запиту. Це дозволяє автоматизувати процес автентифікації під час тестування API через Postman.

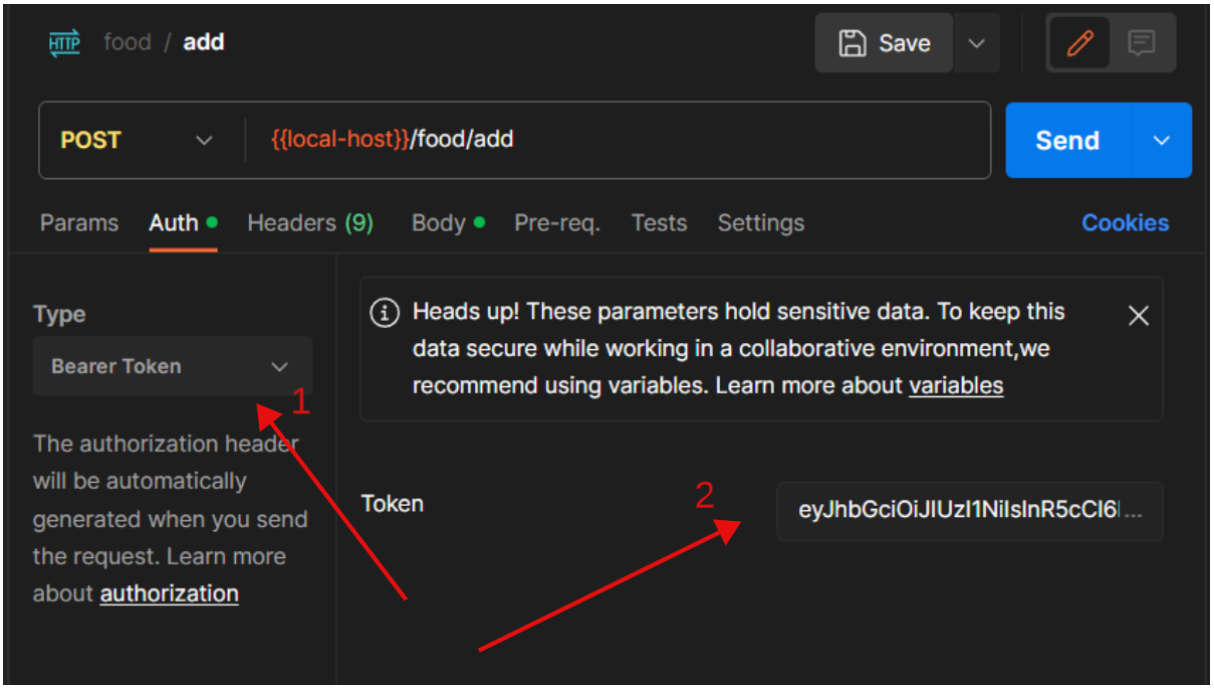


Рисунок 2.4 - Вкладка Auth у Postman

де 1 - Тип токену;

2 - Місце для токену.

2.2 Технічні виклики та рішення

Розробка цифрового продукту завжди супроводжується рядом технічних викликів, які вимагають ретельного вивчення та вибору ефективних рішень. У цьому розділі ми розглянемо ключові аспекти технічної реалізації нашого проекту та вирішення, які були здійснені для подолання викликів.

Масштабованість. Одним із головних викликів було забезпечення ефективної масштабованості системи з ростом обсягів даних та користувацького трафіку. Перед нами стояло завдання забезпечити стабільну роботу додатку навіть при інтенсивному збільшенні об'ємів інформації та кількості користувачів.

Для досягнення цієї мети ми використовуємо сучасні техніки масштабування, такі як [вказати технічне рішення для масштабованості]. Це включає в себе використання хмарних технологій для автоматичного масштабування ресурсів в залежності від потреб системи. Також ми

використовуємо оптимізацію запитів до бази даних, щоб забезпечити швидкий доступ до інформації при будь-якому обсязі.

Ці технічні підходи дозволяють нам ефективно вирішувати проблеми, пов'язані з ростом обсягів даних та забезпечувати надійність роботи додатку під час інтенсивного користувацького трафіку.

Безпека. Аспекти безпеки є визначальними для стабільності та довіри до нашого додатку. Забезпечення аутентифікації та авторизації, а також захист від потенційних атак, є невід'ємною частиною нашого підходу до розробки.

Ми використовуємо сучасні методи аутентифікації, такі як JWT (JSON Web Token), щоб забезпечити безпечний доступ користувачів до системи. Кожен запит на сервер вимагає правильного токена для перевірки ідентичності та визначення рівня доступу. Додаток захищений від атак типу SQL-ін'єкцій та інших потенційних загроз. Ми використовуємо параметризовані запити до бази даних, щоб уникнути вразливостей, пов'язаних із введенням користувачів.

Ці заходи забезпечують високий рівень безпеки та надійності нашого додатку, роблячи його стійким до потенційних загроз та забезпечуючи конфіденційність користувачів.

Інтеграції. Додаток успішно взаємодіє з іншими системами та сервісами для забезпечення оптимального функціонування. Вирішення питань інтеграцій та забезпечення сумісності з іншими платформами відіграє ключову роль у роботі нашого продукту.

Додаток використовує стандартизовані API для обміну даними з іншими системами. Це дозволяє нам ефективно інтегруватися з зовнішніми ресурсами та отримувати актуальну інформацію для подальшого використання в нашому додатку.

Забезпечення сумісності з різними платформами важливо для користувачів. Враховуються особливості різних операційних систем та браузерів, щоб забезпечити однакову якість використання додатку незалежно від вибору платформи.

Цей підхід до інтеграцій дозволяє підтримувати актуальність та функціональність продукту в умовах постійних змін технологічного середовища та вимог користувачів.

Швидкодія. Ефективна швидкодія додатку визначається важливою характеристикою, спрямованою на задоволення користувачів та оптимізацію його функціоналу. Для досягнення цієї мети було використано ряд стратегій та технологічних рішень.

Технічний стек, що включає TypeScript та Node.js, дозволяє використовувати сучасні підходи до програмування, сприяючи швидкому виконанню коду та оптимальній роботі серверної частини. Активно використовуються асинхронні патерни та паралельні обчислення для ефективної обробки користувацьких запитів та підвищення продуктивності.

TypeORM дозволяє ефективно взаємодіяти з базою даних PostgreSQL, використовуючи оптимізовані методи доступу до даних. Ретельна оптимізація запитів та масштабована обробка допомагають зменшити час відповіді на запити.

Використання стратегії оптимізації передачі даних спрямовані на зменшення часу завантаження сторінок та покращення загального досвіду взаємодії з додатком. Ці підходи допомагають підтримувати відмічену швидкість та стабільність.

Тестування. Приділення особливої уваги стратегії та інструментам тестування є важливою частиною нашого підходу до розробки. Ретельно розроблена система тестів дозволяє визначати та виправляти помилки ще на ранніх етапах розробки, забезпечуючи високу якість продукту.

В процесі розробки використовуються різні види тестувань, такі як модульне та інтеграційне. Модульні тести дозволяють перевіряти коректність роботи окремих компонентів, інтеграційні тести - взаємодію різних частин системи. Postman використовується для автоматизації тестування API, дозволяючи проводити швидке та ефективне тестування функціоналу додатку. Цей підхід

спрощує виявлення та виправлення помилок, а також допомагає забезпечити стабільність та надійність системи в умовах різного навантаження.

3 ОПИС ПРОГРАМНОГО ПРОДУКТУ

3.1 Вхідні та вихідні дані

Вхідні дані є основою будь-якої інформаційної системи, і вони мають вирішальне значення для правильного функціонування та надійності системи управління онлайн-магазину. Вхідні дані складають основу інформаційного обміну між клієнтами та системою управління магазину. Інформація про клієнтів, замовлення, товари та гарантії є ключовими компонентами, які визначають ефективність та якість обслуговування. З правильно зібраними та систематизованими вхідними даними магазин може пропонувати персоналізований підхід до кожного клієнта, оперативно виконувати замовлення та забезпечувати надійну підтримку гарантій.

Особлива увага приділяється питанням точності введених даних та їх конфіденційності. Забезпечення правильності та безпеки вхідних даних має стратегічне значення для побудови довіри клієнтів та запобігання можливим проблемам, пов'язаним із втратою чи неправильним використанням інформації.

На кожного клієнта вносять такі дані:

- Унікальний ідентифікаційний номер
- Юзернейм
- Емейл
- Пароль
- Дата створення
- Дата оновлення

Кожний заказ має такі поля:

- Унікальний ідентифікаційний номер
- Юзер айди
- Статус
- Дата створення

- Дата оновлення

Кожна корзина має такі поля:

- Унікальний ідентифікаційний номер
- Номер заказу
- Номер продукту
- Кількість
- Дата створення
- Дата оновлення

Кожний продукт складається з:

- Унікальний ідентифікаційний номер
- Назви
- Кількості
- Ціни
- Номеру корзини
- Дата створення
- Дата оновлення

3.2 Проектування бази даних

Проектування бази даних — це процес створення оптимальної та добре структурованої системи для зберігання та управління інформацією. Уявімо це як архітектурний чертеж для нашого онлайн-магазину, де кожна складова врахована для забезпечення ефективної роботи.

Проектування бази даних є ключовим етапом у розробці будь-якої інформаційної системи. Воно дозволяє структурувати та організувати інформацію, забезпечуючи легкий доступ та уникнення непотрібного дублювання даних. Навіщо це важливо?

1. Ефективність роботи: Проектування бази даних допомагає забезпечити оптимальний доступ до інформації. Це важливо для швидкого виконання запитів, обробки замовлень та інших операцій, які забезпечують плавну роботу магазину.

2. Уникнення дублювання: Визначення ефективних взаємозв'язків між сутностями дозволяє уникнути дублювання інформації. Наприклад, дані про клієнтів можуть бути зв'язані з їхніми замовленнями, що полегшує оновлення та управління даними.

3. Забезпечення безпеки та цілісності: Проектування бази даних включає в себе встановлення правильних обмежень та політик доступу для захисту інформації від несанкціонованого доступу та збереження цілісності даних.

4. Масштабованість: Добре спроектована база даних легко масштабується з ростом обсягу даних та розширенням функціоналу магазину.

5. Гнучкість та зручність розширення: Правильне проектування робить систему гнучкою, дозволяючи легко вносити зміни та розширювати функціонал.

6. Моделювання взаємозв'язків: Визначення сутностей та їхніх взаємозв'язків моделює внутрішню логіку магазину, дозволяючи ефективно організувати дані.

Система управління базами даних PostgreSQL відзначається своєю реляційною структурою, де дані організовані у вигляді таблиць. Зв'язки між цими таблицями відіграють визначальну роль у забезпеченні логічності та ефективності бази даних. Зв'язки в реляційних базах даних визначають спосіб, яким різні таблиці пов'язані між собою. Вони визначають відношення між записами різних таблиць та гарантують цілісність та зв'язність даних. Зв'язки між таблицями визначають:

- Цілісність Даних: Зв'язки дозволяють створювати залежності між даними в таблицях, забезпечуючи їх цілісність та уникнення аномалій при модифікації чи видаленні інформації.
- Оптимізація Простору та Ресурсів: Зв'язки сприяють оптимізації використання простору в базі даних, оскільки вони дозволяють уникати дублювання інформації, зберігаючи її в окремих таблицях.

- Забезпечення Логічності: Вони визначають, як дані взаємодіють, що робить структуру бази даних логічною та зрозумілою для розробників та аналітиків.
- Запобігання Втраті Даних: Зв'язки допомагають уникати втрати даних або виникнення конфліктів, забезпечуючи правильність та консистентність інформації.

У реляційній базі даних є 3 види зв’язків: Один до Одного (One-to-One), Один до Багатьох (One-to-Many), Багато до Багатьох (Many-to-Many).

Один до Одного (One-to-One). Зв'язок "Один до Одного" в базах даних PostgreSQL визначає відношення між записами двох таблиць так, що кожен запис у першій таблиці співставлений з одним, і тільки одним, записом у другій таблиці, і навпаки. Цей вид зв'язку застосовується в ситуаціях, коли існує потреба використовувати відокремлену таблицю для частини інформації, яка стосується головної таблиці. Характеристики Один до Одного Зв'язку:

- Унікальність: Кожен запис у першій таблиці має лише один відповідник у другій, і навпаки. Це забезпечує унікальність співставлення.
- Ефективність Простору: Зв'язок "Один до Одного" сприяє ефективному використанню простору в базі даних, оскільки інформація розподілена між двома таблицями.
- Підтримка Нормалізації:Цей вид зв'язку допомагає в підтримці нормалізованої структури бази даних, оскільки важливі дані розділені для уникнення дублювання інформації.

Один до Багатьох (One-to-Many). Зв'язок "Один до Багатьох" в базах даних PostgreSQL визначає відношення між двома таблицями так, що кожен запис у першій таблиці може мати багато відповідників у другій таблиці, але кожен запис у другій таблиці може бути пов'язаний лише з одним записом у першій таблиці. Цей вид зв'язку використовується, коли один об'єкт у першій таблиці пов'язаний з кількома об'єктами у другій таблиці. Характеристики Один до Багатьох Зв'язку:

- Можливість Багато До Одного: Кожен запис у другій таблиці може бути пов'язаний тільки з одним записом у першій таблиці, але кожен запис у першій таблиці може мати багато відповідників у другій таблиці.
- Забезпечення Зв'язку:Зв'язок забезпечується через використання зовнішнього ключа в другій таблиці, який вказує на унікальний ідентифікатор в першій таблиці.
- Ефективне Управління Даними:Цей вид зв'язку дозволяє ефективно управляти об'ємами даних, оскільки інформація розділена між двома таблицями.

Багато до Багатьох (Many-to-Many). Зв'язок "Багато до Багатьох" - це вид зв'язку між таблицями в базі даних PostgreSQL, де кожен запис у першій таблиці може бути пов'язаний з багатьма записами у другій таблиці, і навпаки. Цей вид зв'язку використовує проміжну таблицю, щоб утримувати ключі обох таблиць, які взаємодіють. Характеристики Багато до Багатьох Зв'язку:

- Таблиця-Сполучник: Для реалізації зв'язку Багато до Багатьох створюється додаткова (сполучникова) таблиця, яка містить зовнішні ключі обох з'єднуваних таблиць.
- Багатосторонній Зв'язок: Кожен запис у першій таблиці може мати багато відповідників у другій таблиці, і навпаки.
- Можливість Розширення: Цей вид зв'язку дозволяє легко розширювати і змінювати взаємодію між записами обох таблиць.

Під час розробки програмного продукту було створено базу даних, що включає чотири основні таблиці. (рис. 3.1):

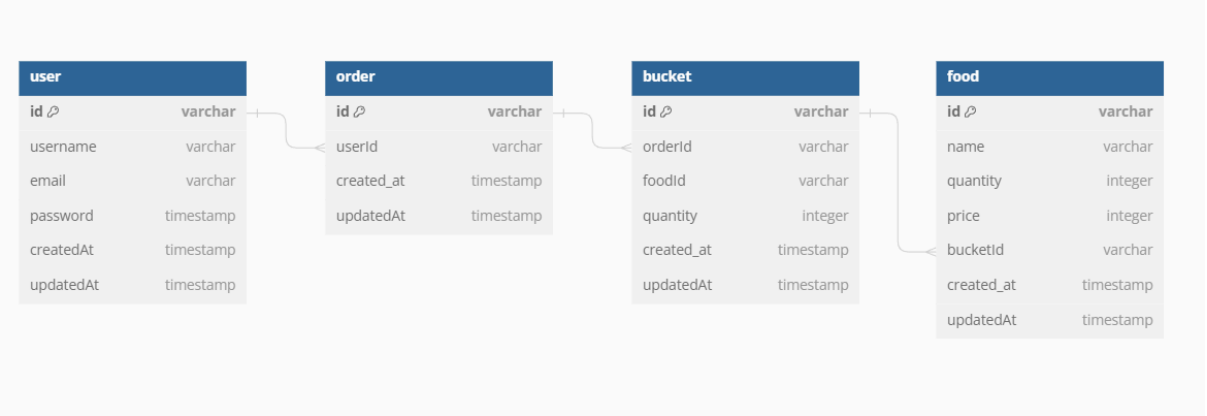


Рисунок . 3.1 - Схема бази даних

User – містить інформацію про користувача, його email та пароль;

Order – містить інформацію про замовлення;

Bucket – містить інформацію про склад корзини;

Food – містить інформацію про товар та його ціну, та наявність на складі.

У проекті ретельно вивчено потреби та особливості додатку, встановивши зв'язки один до одного та один до багатьох (рис 3.2-3.5) для оптимального управління та використання інформації:

- Таблиця Order зв'язана з таблицею User зв’язком Один до Багатьох.
- Таблиця Bucket зв'язана з таблицею Order зв’язком Один до Одного.
- Таблиця Food зв'язана з таблицею Food зв’язком Один до Одного.

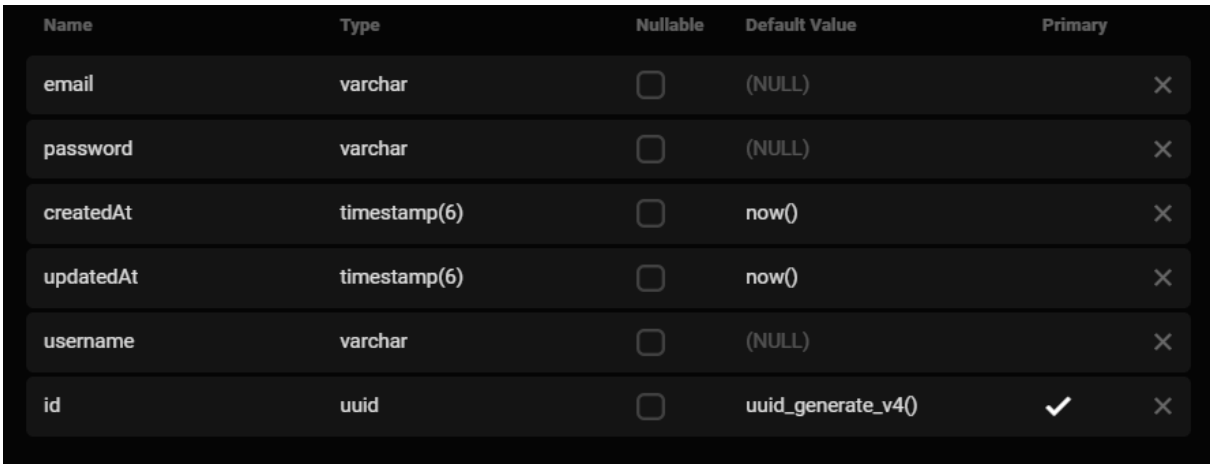
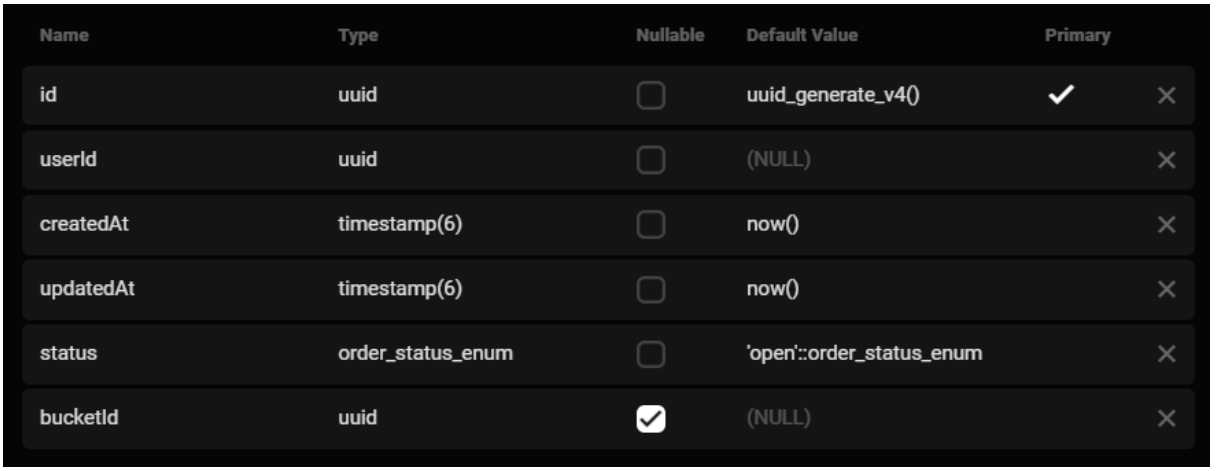
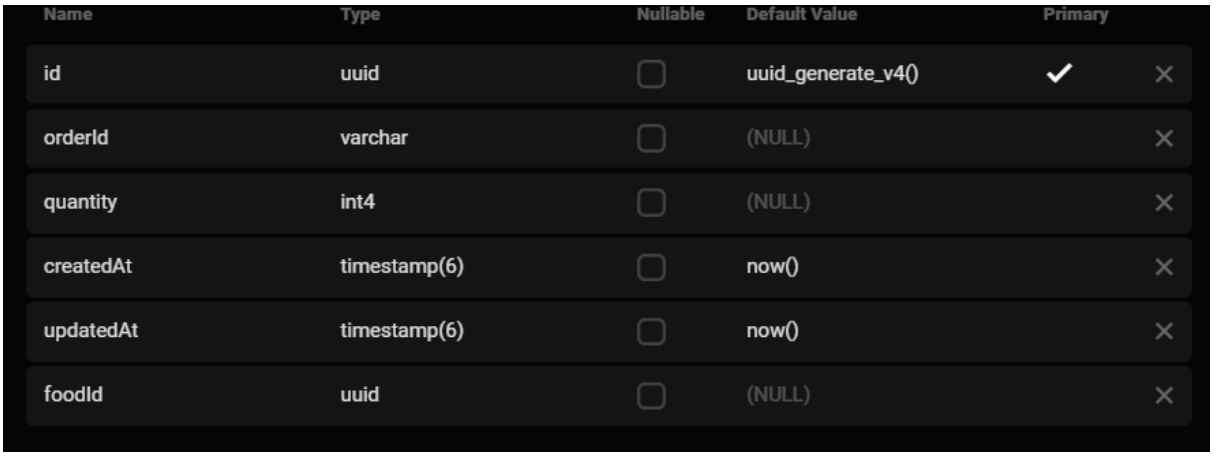


Рисунок 3.2 - Структура таблиці User



Name	Type	Nullable	Default Value	Primary
id	uuid	☐	uuid_generate_v4()	✓
userId	uuid	☐	(NULL)	
createdAt	timestamp(6)	☐	now()	
updatedAt	timestamp(6)	☐	now()	
status	order_status_enum	☐	'open'::order_status_enum	
bucketId	uuid	☑	(NULL)	

Рисунок 3.3 - Структура таблиці Order



Name	Type	Nullable	Default Value	Primary
id	uuid	☐	uuid_generate_v4()	✓
orderId	varchar	☐	(NULL)	
quantity	int4	☐	(NULL)	
createdAt	timestamp(6)	☐	now()	
updatedAt	timestamp(6)	☐	now()	
foodId	uuid	☐	(NULL)	

Рисунок 3.4 - Структура таблиці Bucket

Name	Type	Nullable	Default Value	Primary
id	uuid	☐	uuid_generate_v4()	✓
name	varchar	☐	(NULL)	
quantity	int4	☐	(NULL)	
createdAt	timestamp(6)	☐	now()	
updatedAt	timestamp(6)	☐	now()	
price	int4	☐	(NULL)	

Рисунок 3.5 - Структура таблиці Food

3.3 Опис роботи та тестування додатку

У цьому розділі детально розглядається функціональність та структура розробленого додатку, зокрема проведена комплексна перевірка роботи його бекенд-компоненту. Для здійснення перевірки та валідації різноманітних функцій, які забезпечують взаємодію з сервером, використовувався Postman.

Для початку нам потрібно зареєструвати нового користувача. Для цього відправимо POST запит на адресу auth/register, де передамо в тіло запиту email, username та password (рис 3.6).

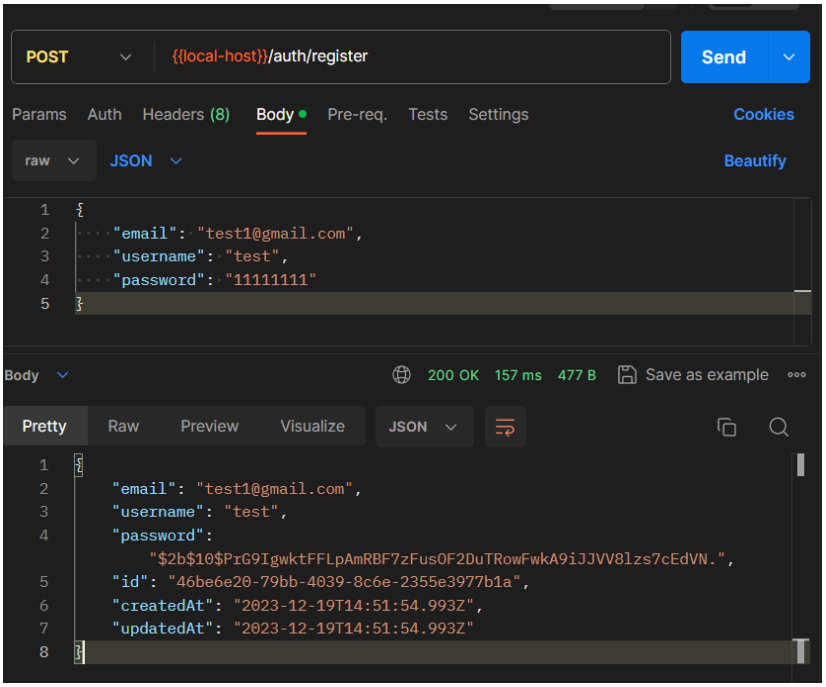


Рисунок 3.6 - POST запит auth/register

Як бачимо з відповіді серверу створився юзер з унікальнім айді, та полями які були передані в тіло запиту. Також, для захисту пароля в базі даних пароль хешується.

Тепер користувач може залогінитьcя в додаток. Для цього відправимо POST запит на адресу auth/login, де передамо в тіло запиту email та password. Спочатку передамо не існуючі дані (рис 3.7)

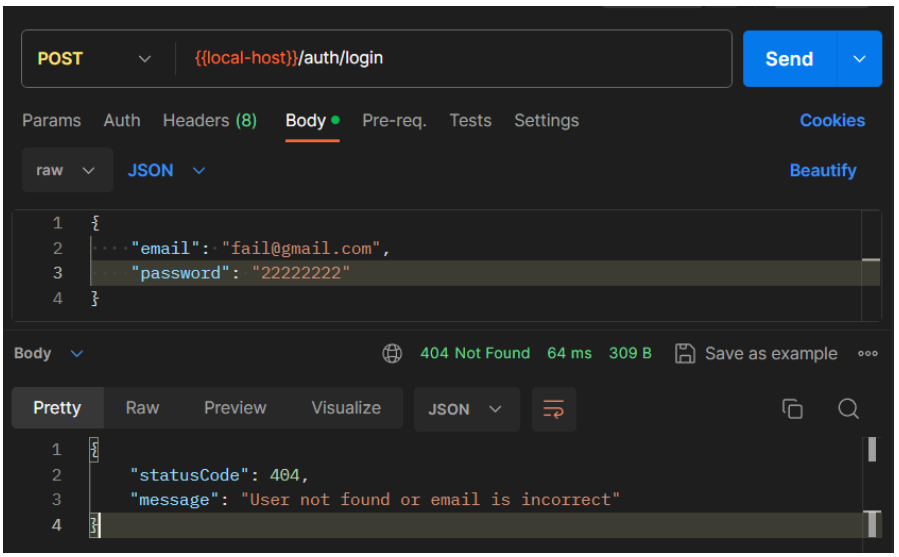


Рисунок 3.7 - Некоректний POST запит auth/login

Як бачимо, сервер не знайшов в базі даних юзера з таким емейлом і паролем, отже авторизація працює правильно. Тепер відправимо POST запит на адресу auth/login з правильними даними (рис 3.8).

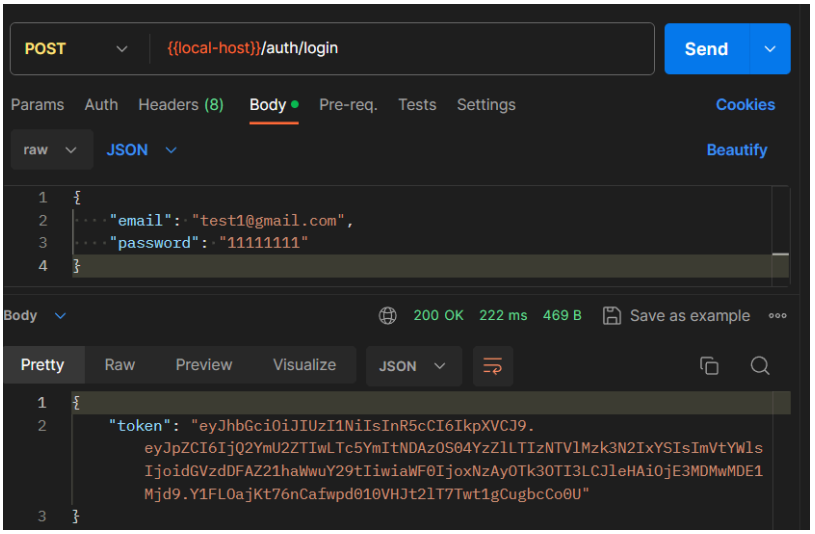


Рисунок 3.8 - Коректний POST запит auth/login

Тепер в відповіді з сервера можна побачити, що вернувся JWT токен, який далі можна використовувати для аутентифікації.

Далі адміністратор магазину може додати товари на склад. Для цього відправимо POST запит на адресу food/add, де передамо в тіло запиту name, quantity та price (рис 3.9).

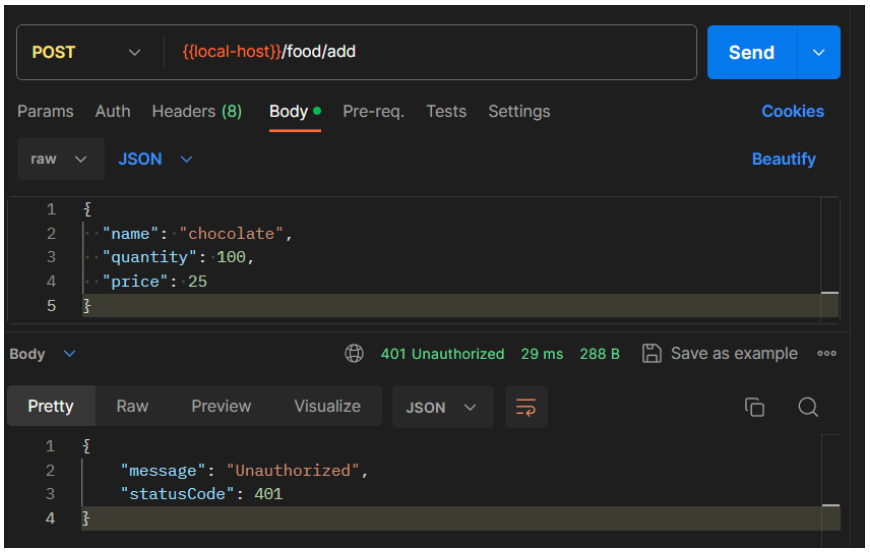


Рисунок 3.9 - Помилка у POST запиту food/add

Можна побачити, що у відповіді з сервера вернулася HTTP помилка зі статусом 401 - Unauthorized. Ця помилка виникла тому що в вкладці Auth, не було передано JWT токен. Отже аутентифікація також працює правильно. Давайте тепер вставимо JWT токен, та додамо кілька товарів (рис 3.10).

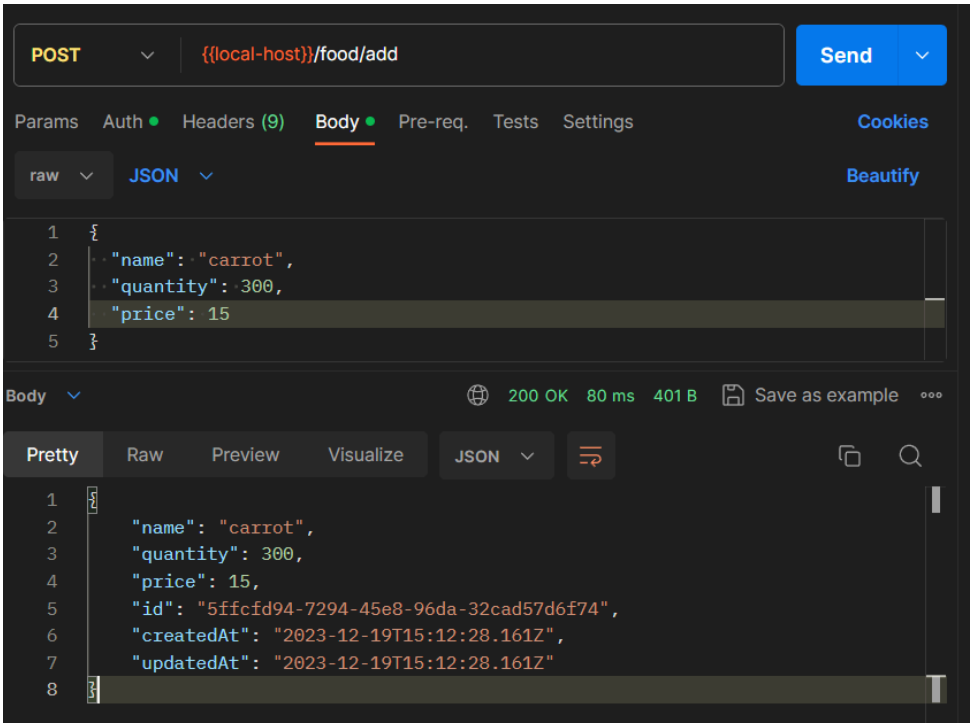


Рисунок 3.10 - POST запит food/add

У відповіді з серверу можна побачити, що створилась їжа carrot з унікальним айді, ціною та кількістю. Юзеру потрібно побачити, які товари вже є в наявності в магизині. Для цього потрібно відправити GET запит на адресу food/list (рис 3.11). В тіло запиту не потрібно передавати якісь параметри.

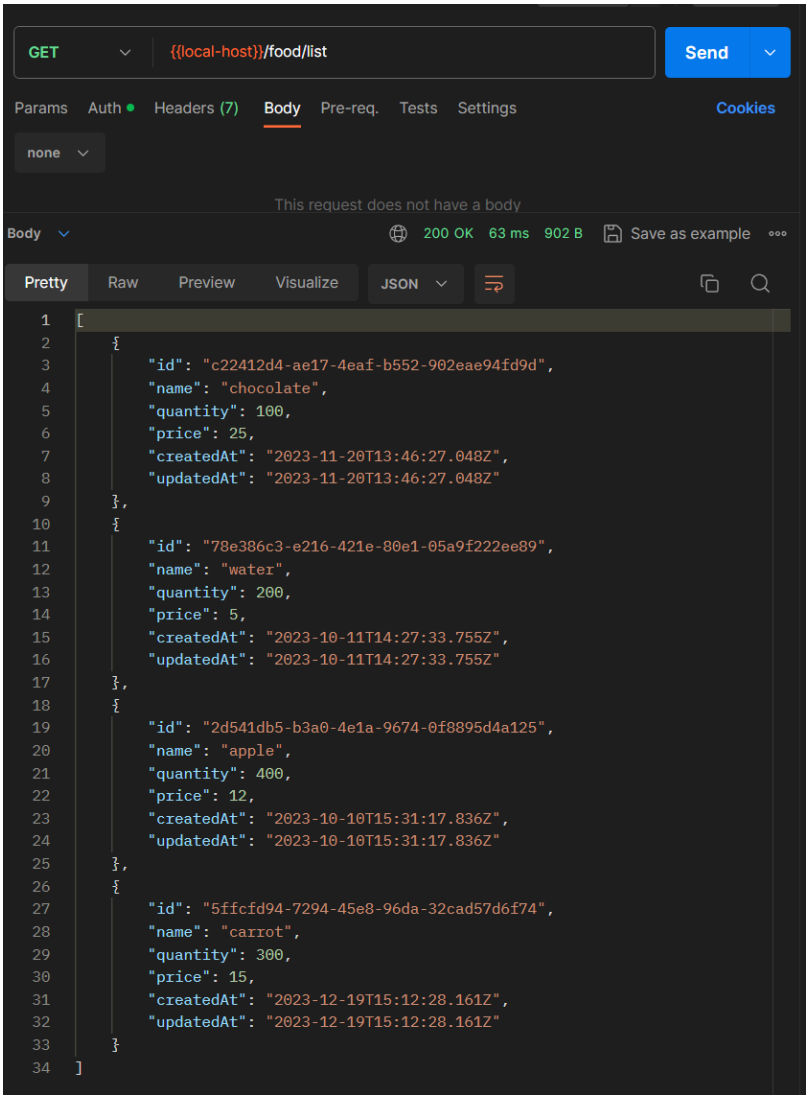


Рисунок 3.11 - GET запит food/list

Можна побачити, що ми додали 4 товари на склад: chocolate, water, apple та carrot. У всіх різні унікальні айді, ціна та кількість на складі.

Тепер юзер може також увійти в додаток, і спочатку фронтенду потрібно створити онлайн-замовлення, в якому юзер буде додавати товари з сайту. Для цього потрібно відправити POST запит на адресу order/create (рис 3.12). В тіло запиту потрібно передати айді юзера, якому створиться замовлення.

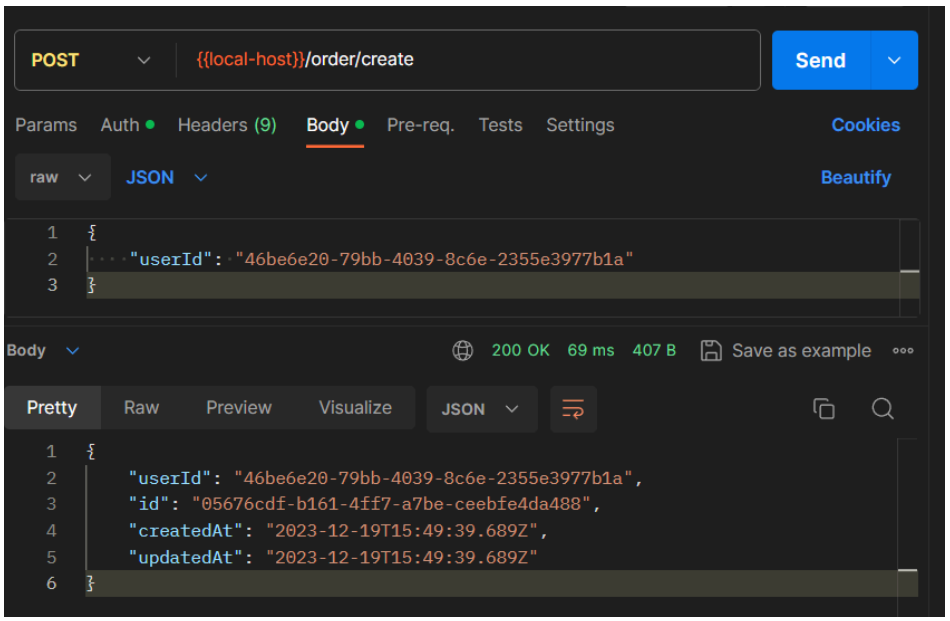


Рисунок 3.12 - POST запит order/create

Ми виконуємо багато різних кроків та запитів з метою підвищення безпеки нашого додатку. Ми активно взаємодіємо з сервером, виконуючи HTTP-запити різних методів. Кожен з цих методів використовується для конкретного завдання та спрямований на покращення безпеки системи.

Проведення різноманітних запитів дозволяє нам перевірити вразливості та недоліки в системі, забезпечуючи високий рівень безпеки. Ми також використовуємо JWT токени для забезпечення автентифікації та авторизації, що є додатковим заходом для захисту наших ресурсів від несанкціонованого доступу.

Цей виважений підхід до тестування та забезпечення безпеки гарантує, що наш додаток залишається захищеним від потенційних загроз та відповідає найвищим стандартам безпеки в індустрії програмного забезпечення.

То ж, вже тепер, знаючи айді замовлення, юзер може додати якісь товари у свою онлайн-корзину. Для цього потрібно відправити POST запит на адресу order/add. В тіло запиту потрібно передати айді замовленя, айді товари та кількість товару. Давайте, наприклад, додамо 2 бутилки води та 1 шоколадку (рис 3.13).

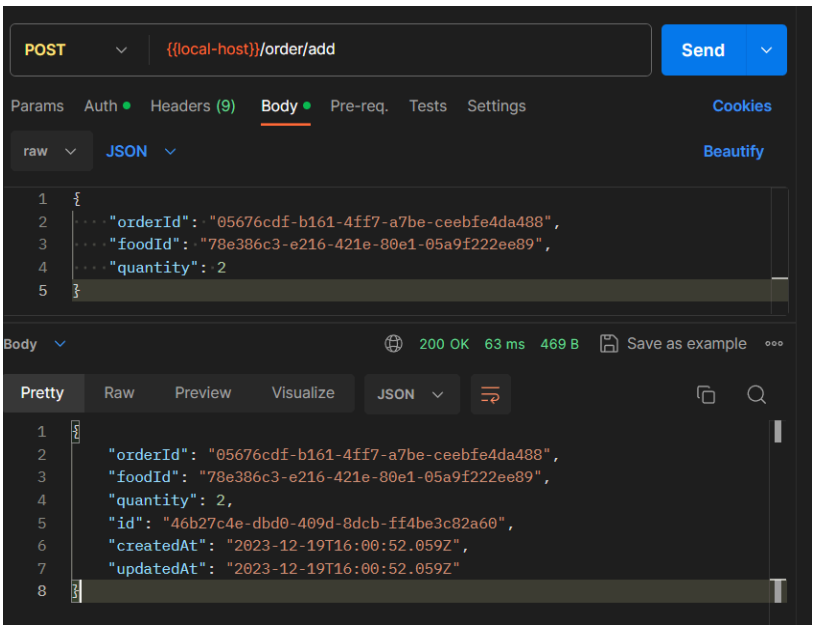


Рисунок 3.13 - POST запит order/add

Також юзер може побачити, яке замовлення він зібрав зараз. Для цього потрібно відправити GET запит на адресу /order. В тіло запиту потрібно передати айді замовлення (рис 3.14).

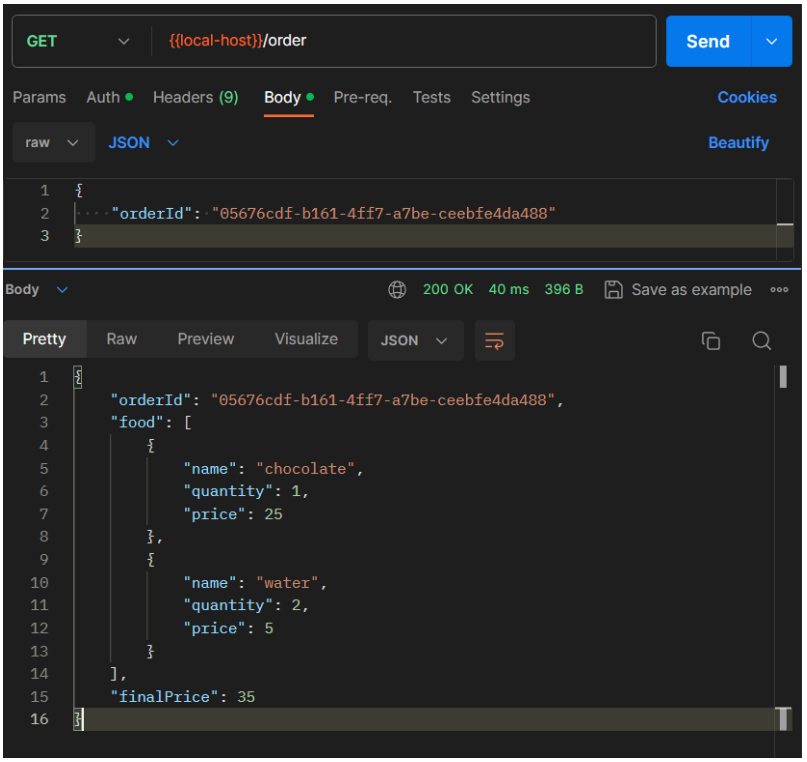


Рисунок 3.14 - GET запит /order

Можна побачити, що у відповіді серверу є окремо номер замовлення, кількість та назва товару, та фінальна ціна. Таким чином покупець може продовжувати наповнювати свою корзину, додавати все більше товарів або змінювати поточні.

Та давайте представимо, що товар на складі закінчився, або адміністратор вирішив більше не продавати цей товар. То ж потрібно ще мати можливість видалити товар з бази даних. Для цього потрібно відправити DELETE запит на адресу /food. В тіло запиту потрібно передати айді товару (рис 3.15).

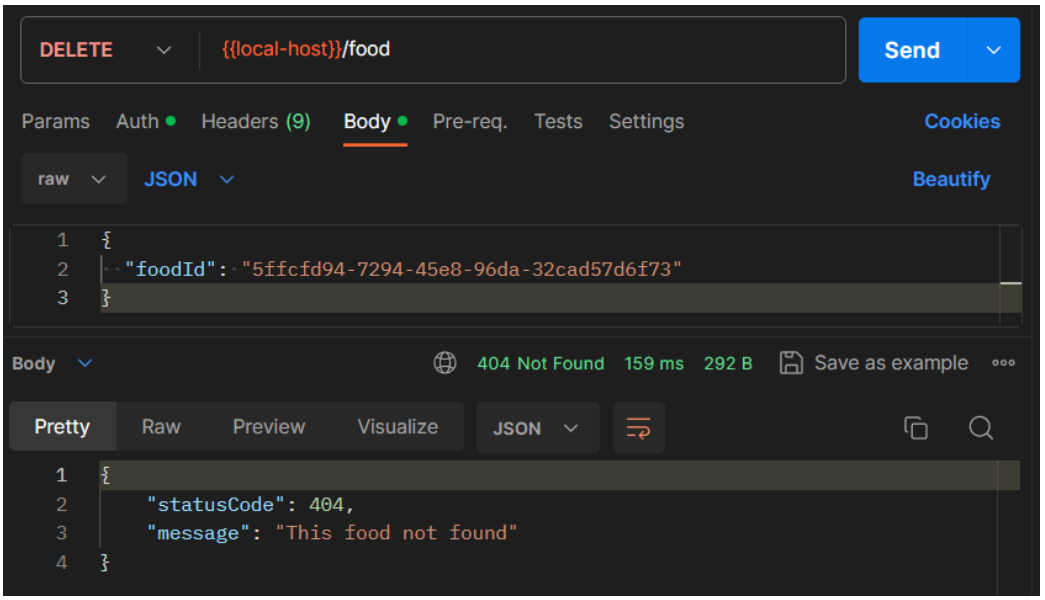


Рисунок 3.15 - Некоректний DELETE запит /food

Можна побачити, що адміністратор або ввів некоректний айді товару, чи такого товару вже не існує. То ж у відповіді з сервера вернулася HTTP помилка зі статусом 404 - Not found. Помилка 404, відома як "Not Found" або "Не знайдено," використовується для інформування користувача про те, що сервер не може знайти запитаний ресурс. Це стандартний HTTP-статус, який повідомляє клієнта, що запитаний URL не існує на сервері.

У контексті безпеки та взаємодії з Postman, отримання помилки 404 може бути корисним випробуванням на вразливості. Коли ми намагаємося здійснити

запит до неіснуючого ресурсу, і сервер повертає помилку 404, це може свідчити про те, що сервер правильно реагує на невірний або неіснуючий URL.

Таке тестування дозволяє нам перевірити, чи правильно оброблюється відсутність ресурсів та чи не допускається неправильний доступ до інформації чи функціональності, що може підвищити рівень безпеки нашого додатку.

Зараз база даних виглядає так (рис 3.16).

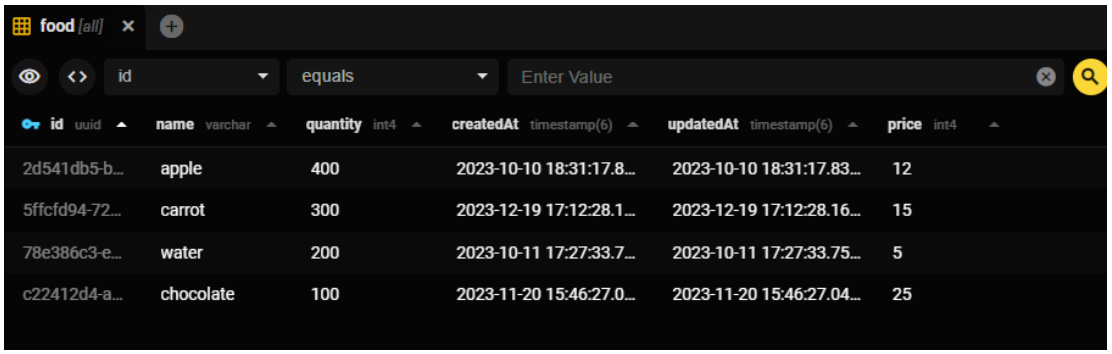


Рисунок 3.16 - Таблиця Food

Та якщо ввести кореткний айді товару, наприклад яблока, воно буде видалено з бази даних. Для цього ще раз потрібно відправити DELETE запит на адресу /food з правильним айді товару (рис 3.17).

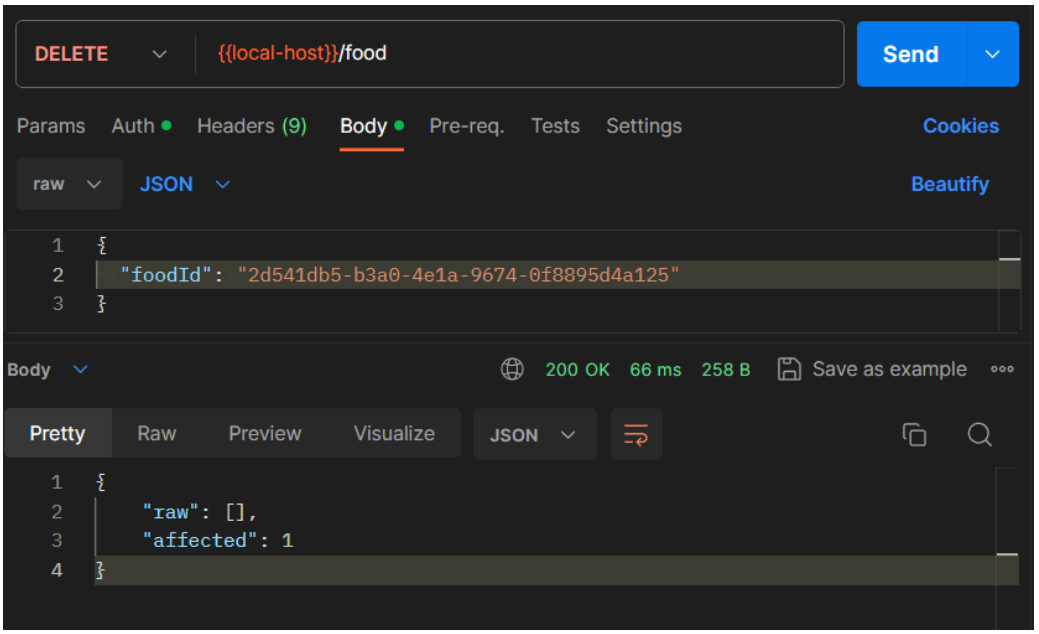


Рисунок 3.17 - DELETE запит /food

Можна побачити, що у відповіді з серверу пришло поле affected. Якщо affected дорівнює 1, це означає успішне видалення одного запису. Такі відповіді можна використовувати для перевірки стану і результатів операції видалення. Поле raw - це покажчик того, що під час видалення не було повернено жодних додаткових сирих даних.

Тепер база даних виглядає так (рис 3.18).

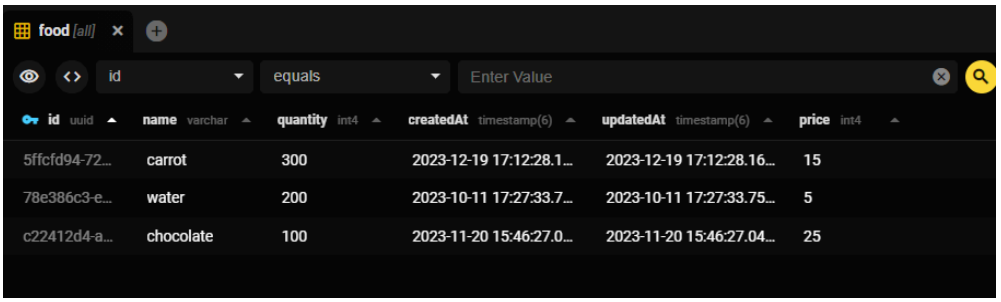


Рисунок 3.18 - Таблиця Food

Можна побачити, що запис яблока вже відсутній в базі, і тепер юзер не зможе додати його у свою корзину. Та якщо адміністратор забажає заново додати цей товар на склад, він може знову використати запит на додавання товару.

Після того, як користувач обрав всі товари, та хоче оплатити своє замовлення, потрібно відправити POST запит на адресу order/pay. В тіло запиту потрібно передати айді замовлення (рис 3.19).

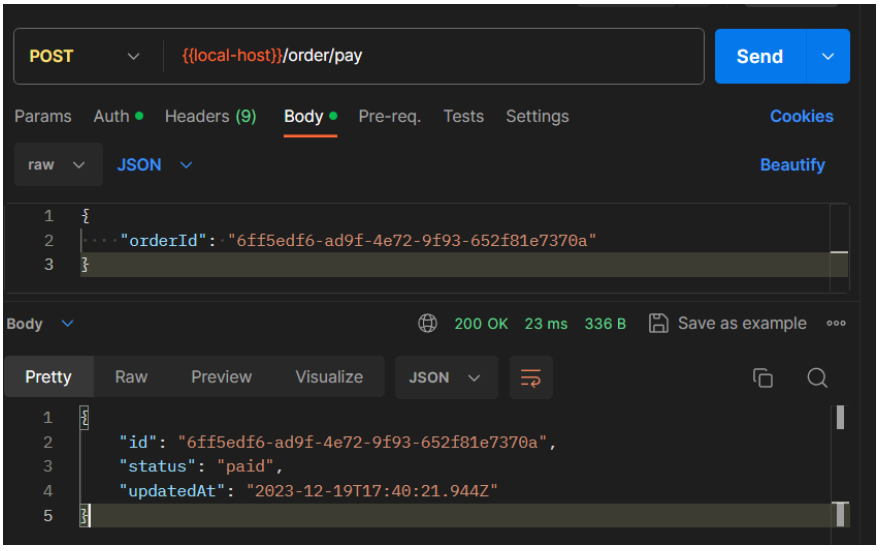


Рисунок 3.19 - Успішний POST запит order/pay

Можна побачити, що статус замовлення змінився на paid, тобто оплачений. З метою забезпечення безпеки операції оплати, система перевіряє замовлення із базою даних. Якщо таке замовлення вже оплачене, система відхиляє дубльований запит та повідомляє користувача про неможливість подвійної оплати. Це забезпечує надійність та цілісність операцій, а також захищає користувача від неправильного списання коштів (рис 3.20).

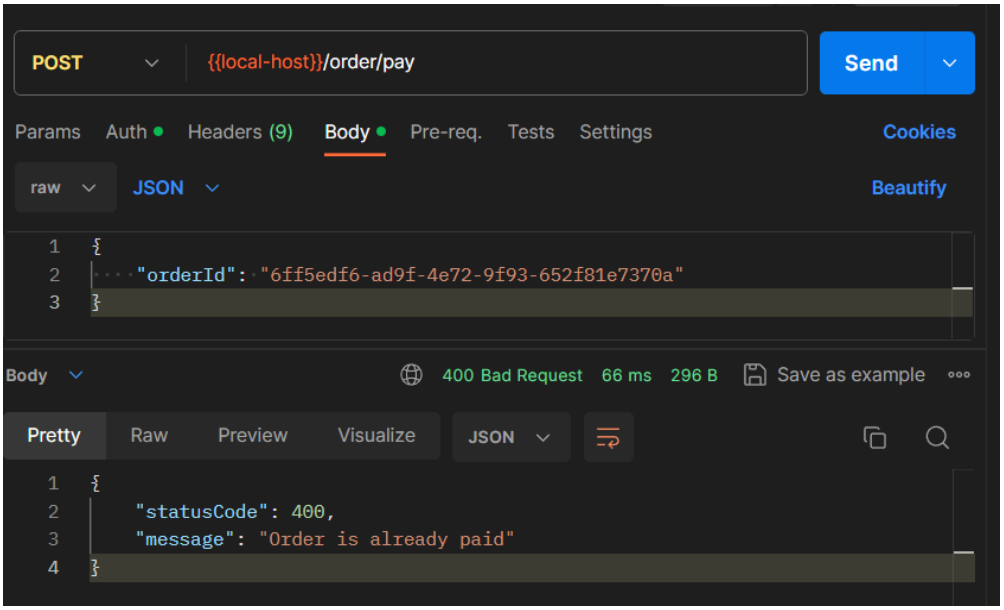


Рисунок 3.20 - Дубльований POST запит order/pay

Бачимо, що виникає помилка, що замовлення вже оплачене. Цей захищає нас від дублювання оплати ти проблем для користувача. Дублювання оплати є небажаним явищем з точки зору ефективного та правильного функціонування системи оплати. Це може призвести до невірних фінансових розрахунків, порушень в обліку та непорозумінь з користувачем. Крім того, подвійна оплата може викликати негативний досвід у співтоваристві користувачів та негативно позначитися на репутації бізнесу. Тому системи оплати повинні бути належним чином захищені та оптимізовані, щоб уникнути подібних непорозумінь та забезпечити стабільну та надійну операційну діяльність.

Також користувач може побачити історію своїх замовлень на сайті. Для цього потрібно відправити GET запит на адресу order/history (рис 3.21).

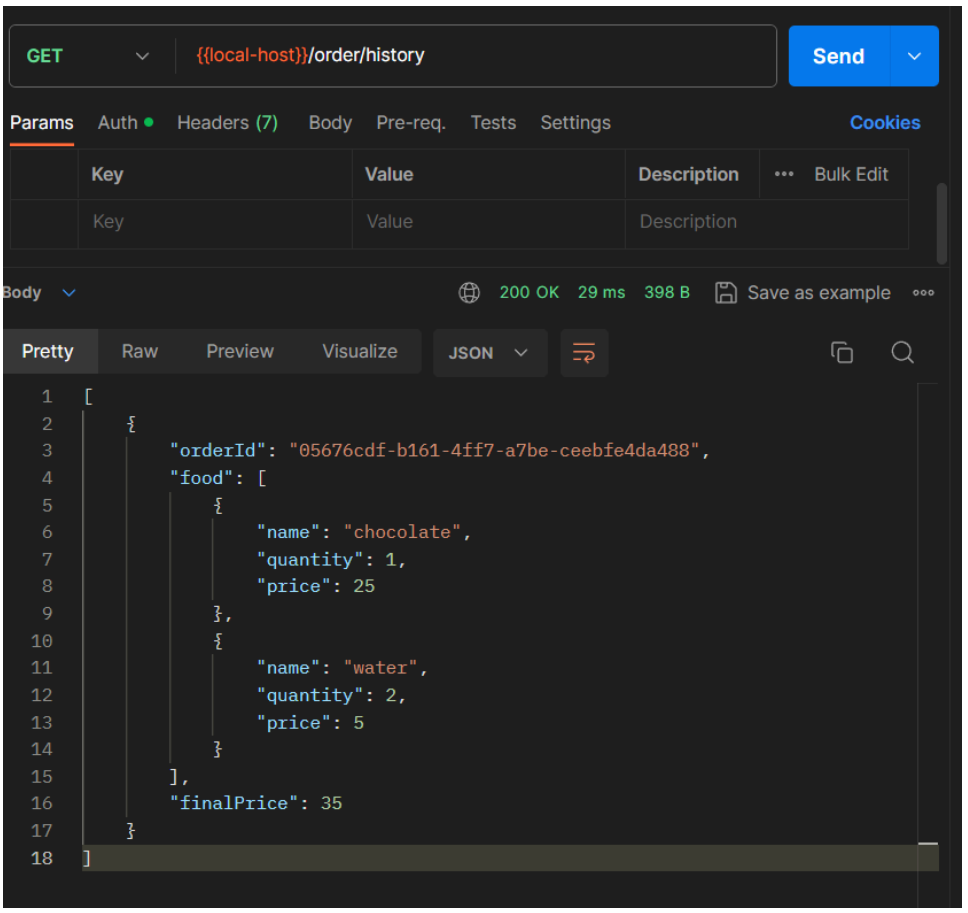


Рисунок 3.21 - GET запит order/history

Можна побачити, що в відповіді з серверу приходить масив замовлень, де є номер замовлення, придбані товари, та фінальна ціна. Наразі у користувача тільки одне сплачене замовлення, тож й у відповіді тільки воно і прийшло.

Додаток успішно пройшов перевірку та тестування. Всі важливі компоненти, такі як авторизація, робота з базою даних, обробка та відправка запитів були ретельно протестовані. Всі HTTP методи виконують свої завдання, а взаємодія із сервером відбувається швидко та надійно.

Було проведено тестування кожного етапу функціоналу, щоб переконатися в його правильній роботі. Успішне проходження усіх тестів свідчить про стабільність та надійність розробленого додатку. Такий підхід до тестування

дозволяє впевнено стверджувати, що додаток готовий до ефективної експлуатації та задоволення потреб користувачів.

ВИСНОВКИ

У ході дослідження та розробки даного проекту було проведено комплексний аналіз сучасних цифрових сервісів для онлайн-магазинів. Визначено ключові компоненти та технології, що впливають на функціональність та продуктивність магазинів. Обґрунтовано вибір мови програмування TypeScript у контексті Node.js, використання середовища розробки JetBrains WebStorm, бази даних PostgreSQL та ORM TypeORM.

Ретельно проаналізовано вхідні та вихідні дані, що визначають структуру додатку та взаємодію його компонентів. Проведено оцінку ефективності розроблених функціональностей та їх вплив на користувацький досвід. Проектування бази даних з використанням зв'язків один до одного та один до багатьох сприяло створенню структурованої та оптимізованої системи управління даними.

Розглянуті переваги використання Postman для автоматизації тестування та взаємодії з сервером. Проаналізовано важливість безпеки, ефективності маркетингу, лояльності клієнтів, технологічної інфраструктури та гнучкості в контексті сучасних онлайн-магазинів.

Основна мета дипломного проекту - створення надійного та ефективного онлайн-магазину - була досягнута завдяки виваженому вибору технологій та уважному підходу до всіх етапів розробки. Даний проект не лише відображає сучасні тенденції у сфері електронної комерції, але й визначає шляхи подальшого розвитку та удосконалення подібних інтернет-платформ.

Загальний висновок проекту відображає його ключові аспекти та вклад у сферу електронної комерції. Проект реалізовано з використанням передових технологій, що гарантує високу продуктивність та надійність онлайн-магазину. Розглянуті аспекти безпеки, ефективності маркетингу, лояльності клієнтів, технологічної інфраструктури та гнучкості дозволяють зрозуміти глибину дослідження та значущість розробленого рішення.

Створений додаток взаємодіє з базою даних, використовує сучасні підходи до проектування та забезпечує безпечність та надійність усіх транзакцій. Postman в ролі інструмента для автоматизації тестування дозволяє забезпечити якість коду та ефективність серверної частини додатку.

ПЕРЕЛІК ПОСИЛАНЬ

1. What is an API? - Application Programming Interface Explained - AWS. Amazon Web Services, Inc. URL: https://aws.amazon.com/what-is/api/?nc1=h_ls (date of access: 27.12.2023).
2. Gillis A. S. What is REST API (RESTful API)?. App Architecture. URL: https://www.techtarget.com/searchapparchitecture/definition/RESTful-API (date of access: 27.12.2023).
3. Richardson L. RESTful Web APIs: Services for a Changing World. O'Reilly, 2013. 406 p.
4. What Is SEO - Search Engine Optimization?. Search Engine Land. URL: https://searchengineland.com/guide/what-is-seo (date of access: 27.12.2023).
5. What is your carbon footprint?. The Nature Conservancy. URL: https://www.nature.org/en-us/get-involved/how-to-help/carbon-footprint-calculator/ (date of access: 27.12.2023).
6. Global shipping « World Ocean Review. World Ocean Review. URL: https://worldoceanreview.com/en/wor-1/transport/global-shipping/ (date of access: 27.12.2023).
7. 10 basic rules to making an irresistible website | Business Hacks Blog. Odoo S.A. URL: https://www.odoo.com/uk_UA/blog/business-hacks-1/10-basic-rules-to-making-an-irresistible-website-201 (date of access: 27.12.2023).
8. 13 2. M. The Business Book. Dorling Kindersley Ltd, 2003.
9. Contributor T. What is Web Application (Web Apps) and its Benefits. Software Quality. URL: https://www.techtarget.com/searchsoftwarequality/definition/Web-application-Web-app (date of access: 27.12.2023).
10. Purewal S. Learning Web App Development: Build Quickly with Proven JavaScript Techniques. O'Reilly, 2014. 306 p.
11. Progressive web apps | MDN. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps (date of access: 27.12.2023).
12. Firtman M. High Performance Mobile Web: Best Practices for Optimizing Mobile Web Apps. O'Reilly Media, Incorporated, 2016.
13. AR vs VR: What's The Difference?. Splunk-Blogs. URL: https://www.splunk.com/en_us/blog/learn/ar-vr.html. (date of access: 27.12.2023).
14. Ater T. Building Progressive Web Apps: Bringing the Power of Native to the Browser. O'Reilly Media, 2017. 288 p.

15. Watrall E. Head first Web design. Beijing : O'Reilly, 2009. 463 p.

16. Bloomenthal A. E-commerce Defined: Types, History, and Examples. Investopedia. URL: https://www.investopedia.com/terms/e/ecommerce.asp (date of access: 27.12.2023).

17. Website Development: In-Depth Guide For Beginners. Search Engine Journal. URL: https://www.searchenginejournal.com/website-development/480436/ (date of access: 27.12.2023).

18. Smith S. Internet of Risky Things: Trusting the Devices That Surround Us. O'Reilly Media, Incorporated, 2017.

19. Privacy vs. Security: Exploring the Differences & Relationship | Okta. Employee and Customer Identity Solutions | Okta. URL: https://www.okta.com/identity-101/privacy-vs-security/ (date of access: 27.12.2023).

20. Effective Marketing. In Easy Steps, 2011.

21. Самое важное о системах управления контентом и их типах. Oracle | Cloud Applications and Cloud Platform. URL: https://www.oracle.com/cis/content-management/what-is-cms/ (дата звернення: 27.12.2023).

22. Goldberg J. Learning TypeScript: Enhance Your Web Development Skills Using Type-Safe JavaScript. O'Reilly Media, Incorporated, 2022.

23. Introduction to Node.js | Node.js. Node.js. URL: https://nodejs.org/en/learn/getting-started/introduction-to-nodejs (date of access: 27.12.2023).

24. Obe R. O., Hsu L. S. PostgreSQL: Up and Running: A Practical Guide to the Advanced Open Source Database. O'Reilly Media, 2017. 314 p.

25. What are ACID Transactions?. Databricks. URL: https://www.databricks.com/glossary/acid-transactions (date of access: 27.12.2023).

26. TypeORM - Amazing ORM for TypeScript and JavaScript. URL: https://typeorm.io/ (date of access: 27.12.2023).

27. What is Postman? Postman API Platform. Postman API Platform. URL: https://www.postman.com/product/what-is-postman/ (date of access: 27.12.2023).

28. HTTP Methods GET vs POST. W3Schools Online Web Tutorials. URL: https://www.w3schools.com/tags/ref_httpmethods.asp (date of access: 27.12.2023).

ДОДАТОК А. КОДИ ДОДАТКУ

```
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';

async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  app.useGlobalPipes();
  await app.listen(3000);
}
bootstrap();

import { Module } from '@nestjs/common';
import { AppController } from './app.controller';
import { AppService } from './app.service';
import { AuthModule } from './auth/auth.module';
import { ConfigModule, ConfigService } from '@nestjs/config';
import { TypeOrmModule, TypeOrmModuleOptions } from '@nestjs/typeorm';
import { UserModule } from './user/user.module';
import { FoodModule } from './food/food.module';
import { OrderModule } from './order/order.module';
import { BucketModule } from './bucket/bucket.module';

@Module({
  imports: [
    AuthModule,
    ConfigModule.forRoot({ isGlobal: true }),
    TypeOrmModule.forRootAsync({
      imports: [ConfigModule],
```

```
      useFactory: (configService: ConfigService) =>
        ({
          type: 'postgres',
          host: configService.get('DB_HOST'),
          port: configService.get('DB_PORT'),
          username: configService.get('DB_USERNAME'),
          password: configService.get('DB_PASSWORD'),
          database: configService.get('DB_NAME'),
          synchronize: true,
          entities: [__dirname + '/**/*.entity{.js, .ts}'],
        }) as TypeOrmModuleOptions,
      inject: [ConfigService],
    }),
    UserModule,
    FoodModule,
    OrderModule,
    BucketModule,
  ],
  controllers: [AppController],
  providers: [AppService],
})
export class AppModule {}

import { IsEmail, IsNotEmpty, IsString } from 'class-validator';

export class UserCreds {
  @IsEmail()
  @IsNotEmpty()
  email: string;
```

```
  @IsNotEmpty()
  @IsString()
  password: string;
}

export class RegisterDto extends UserCreds {
  @IsNotEmpty()
  @IsString()
  username: string;
}

export class LoginDto extends UserCreds {}

import {
  Body,
  Controller,
  HttpStatus,
  Post,
  Response,
} from '@nestjs/common';
import { UserService } from '../user/user.service';
import { LoginDto, RegisterDto } from './dto/auth.dto';
import { AuthService } from './auth.service';

@Controller('auth')
export class AuthController {
  constructor(
    private readonly userService: UserService,
```

```
    private readonly authService: AuthService,
  ) {}

  @Post('register')
  async register(@Response() res: any, @Body() registerDto: RegisterDto) {
    return res
      .status(HttpStatus.OK)
      .json(await this.userService.createUser(registerDto));
  }

  @Post('login')
  async login(@Response() res: any, @Body() loginDto: LoginDto) {
    return res
      .status(HttpStatus.OK)
      .json(await this.authService.login(loginDto));
  }
}
import {
  CanActivate,
  ExecutionContext,
  Injectable,
  UnauthorizedException,
} from '@nestjs/common';
import { JwtService } from '@nestjs/jwt';
import { Request } from 'express';

import * as dotenv from 'dotenv';
dotenv.config();
```

```
@Injectable()
export class AuthGuard implements CanActivate {
  constructor(private jwtService: JwtService) {}

  async canActivate(context: ExecutionContext): Promise<boolean> {
    const request = context.switchToHttp().getRequest();
    const token = this.extractTokenFromHeader(request);
    if (!token) {
      throw new UnauthorizedException();
    }
    try {
      const payload = await this.jwtService.verifyAsync(token, {
        secret: process.env.JWT_SECRET_KEY,
      });

      request['user'] = payload;
    } catch {
      throw new UnauthorizedException();
    }
    return true;
  }

  private extractTokenFromHeader(request: Request): string | undefined {
    const [type, token] = request.headers.authorization?.split(' ') ?? [];
    return type === 'Bearer' ? token : undefined;
  }
}

import { Module } from '@nestjs/common';
```

```
import { AuthService } from './auth.service';
import { AuthController } from './auth.controller';
import { UserModule } from '../user/user.module';
import { JwtModule } from '@nestjs/jwt';

import * as dotenv from 'dotenv';
dotenv.config();

@Module({
  imports: [
    UserModule,
    JwtModule.register({
      global: true,
      secret: process.env.JWT_SECRET_KEY,
      signOptions: { expiresIn: '3600s' },
    }),
  ],
  providers: [AuthService],
  controllers: [AuthController],
})
export class AuthModule {}

import { HttpException, HttpStatus, Injectable } from '@nestjs/common';
import { UserService } from '../user/user.service';
import { JwtService } from '@nestjs/jwt';
import { LoginDto } from './dto/auth.dto';

@Injectable()
export class AuthService {
```

```
constructor(
  private readonly userService: UserService,
  private readonly jwtService: JwtService,
) {}

async login(loginDto: LoginDto): Promise<{ token: string }> {
  const user = await this.userService.getUserByEmail(loginDto.email);

  if (!user) {
    throw new HttpException(
      'User not found or email is incorrect',
      HttpStatus.NOT_FOUND,
    );
  }

  if (
    await this.userService.compareHash(
      loginDto.password,
      user.password as string,
    )
  ) {
    return { token: await this.signIn(user.id, user.email) };
  }
}

async signIn(userId: string, email: string): Promise<string> {
  const payload = {
    id: userId,
    email,
```

```
    };

    return this.jwtService.sign(payload);
  }
}

import {
  Column,
  CreateDateColumn,
  Entity,
  JoinColumn,
  OneToOne,
  PrimaryGeneratedColumn,
  UpdateDateColumn,
} from 'typeorm';
import { Order } from '../../order/entity/order.entity';
import { Food } from '../../food/entity/food.entity';

@Entity()
export class Bucket {
  @PrimaryGeneratedColumn('uuid')
  id: string;

  @Column()
  orderId: string;

  @OneToOne(() => Order, (order) => order.id)
  order: Order;
```

```
  @Column()
  foodId: string;

  @OneToOne(() => Food, (food) => food.id)
  @JoinColumn()
  food: Food;

  @Column()
  quantity: number;

  @CreateDateColumn()
  createdAt: Date;

  @UpdateDateColumn()
  updatedAt: Date;
}

import { forwardRef, Module } from '@nestjs/common';
import { BucketController } from './bucket.controller';
import { BucketService } from './bucket.service';
import { TypeOrmModule } from '@nestjs/typeorm';
import { Bucket } from './entity/bucket.entity';
import { OrderModule } from '../order/order.module';

@Module({
                imports: [forwardRef(() => OrderModule),
TypeOrmModule.forFeature([Bucket])],
  controllers: [BucketController],
  providers: [BucketService],
```

```
  exports: [BucketService],
})
export class BucketModule {}

import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { Repository } from 'typeorm';
import { Bucket } from './entity/bucket.entity';
import { AddToOrderDto } from '../order/dto/order.dto';
import { OrderService } from '../order/order.service';

@Injectable()
export class BucketService {
  constructor(
    @InjectRepository(Bucket)
    private readonly bucketRepository: Repository<Bucket>,
    private readonly orderService: OrderService,
  ) {}

  async addToOrder(data: AddToOrderDto) {
    return this.bucketRepository.save(data);
  }

  async getBucket(orderId: string) {
    const orders = await this.bucketRepository.find({
      where: {
        orderId,
      },
      relations: ['food'],
```

```
  });

  const food = orders.map((order) => ({
    name: order.food.name,
    quantity: order.quantity,
    price: order.food.price,
  }));

  const finalPrice = food.reduce(
    (sum, item) => sum + item.quantity * item.price,
    0,
  );

  return {
    orderId,
    food,
    finalPrice,
  };
}

async getBucketHistory(userId: string) {
  const orderIds = await this.orderService.findAllOrdersByUSerId(userId);

  const orders = await this.bucketRepository.find({
    where: {
      orderId: orderIds[0],
    },
    relations: ['food'],
  });
```

```
    const food = orders.map((order) => ({
      name: order.food.name,
      quantity: order.quantity,
      price: order.food.price,
    }));

    const finalPrice = food.reduce(
      (sum, item) => sum + item.quantity * item.price,
      0,
    );
    return [{ orderId: orderIds[0], food, finalPrice }];
  }
}

import { IsInt, Min, IsNotEmpty, IsString } from 'class-validator';

export class AddFoodDto {
  @IsString()
  @IsNotEmpty()
  name: string;

  @IsNotEmpty()
  @IsInt({})
  @Min(1)
  quantity: number;

  @IsNotEmpty()
  @IsInt({})
```

```
  @Min(1)
  price;
}

export class DeleteFoodFto {
  @IsString()
  @IsNotEmpty()
  foodId: string;
}

import {
  Column,
  CreateDateColumn,
  Entity,
  OneToOne,
  PrimaryGeneratedColumn,
  UpdateDateColumn,
} from 'typeorm';
import { Bucket } from '../../bucket/entity/bucket.entity';

@Entity()
export class Food {
  @PrimaryGeneratedColumn('uuid')
  id: string;

  @Column()
  name: string;

  @Column()
```

```
  quantity: number;

  @Column()
  price: number;

  @OneToOne(() => Bucket, (bucket) => bucket.id)
  bucket: Bucket;

  @CreateDateColumn()
  createdAt: Date;

  @UpdateDateColumn()
  updatedAt: Date;
}

import {
  Body,
  Controller,
  Delete,
  Get,
  HttpStatus,
  Post,
  Response,
  UseGuards,
} from '@nestjs/common';
import { FoodService } from './food.service';
import { AddFoodDto, DeleteFoodFto } from './dto/food.dto';
import { AuthGuard } from '../auth/auth.guard';
```

```
@Controller('food')
export class FoodController {
  constructor(private readonly foodService: FoodService) {}

  @Get('list')
  @UseGuards(AuthGuard)
  async foodList(@Response() res: any) {
    return res.status(HttpStatus.OK).json(await this.foodService.getFoodList());
  }

  @Post('add')
  @UseGuards(AuthGuard)
  async addFood(@Response() res: any, @Body() data: AddFoodDto) {
    return res
      .status(HttpStatus.OK)
      .json(await this.foodService.addNewFood(data));
  }

  @Delete('')
  async deleteFood(@Response() res: any, @Body() data: DeleteFoodFto) {
    return res
      .status(HttpStatus.OK)
      .json(await this.foodService.deleteFood(data.foodId));
  }
}

import { Module } from '@nestjs/common';
import { FoodService } from './food.service';
import { FoodController } from './food.controller';
```

```
import { TypeOrmModule } from '@nestjs/typeorm';
import { Food } from './entity/food.entity';

@Module({
  imports: [TypeOrmModule.forFeature([Food])],
  providers: [FoodService],
  controllers: [FoodController],
  exports: [FoodService],
})
export class FoodModule {}

import { HttpException, HttpStatus, Injectable } from '@nestjs/common';
import { AddFoodDto } from './dto/food.dto';
import { InjectRepository } from '@nestjs/typeorm';
import { Repository } from 'typeorm';
import { Food } from './entity/food.entity';

@Injectable()
export class FoodService {
  constructor(
                @InjectRepository(Food) private readonly foodRepository:
Repository<Food>,
  ) {}

  async getFoodList() {
    return this.foodRepository.find();
  }

  async addNewFood(data: AddFoodDto) {
```

```
  const ifFoodExist = await this.foodRepository.findOne({
    where: {
      name: data.name.trim(),
    },
  });

  if (ifFoodExist) {
    throw new HttpException(
      'This food is already in the store',
      HttpStatus.BAD_REQUEST,
    );
  }

  return this.foodRepository.save({
    name: data.name.trim(),
    quantity: data.quantity,
    price: data.price,
  });
}

async deleteFood(foodId: string) {
  const food = await this.foodRepository.findOne({
    where: { id: foodId },
  });

  if (!food) {
    throw new HttpException('This food not found', HttpStatus.NOT_FOUND);
  }
```

```
    return this.foodRepository.delete({
      id: foodId,
    });
  }
}

import { IsNotEmpty, IsString, IsNumber } from 'class-validator';

export class OrderDto {
  @IsString()
  @IsNotEmpty()
  userId: string;
}

export class AddToOrderDto {
  @IsString()
  @IsNotEmpty()
  orderId: string;

  @IsString()
  @IsNotEmpty()
  foodId: string;

  @IsNumber()
  @IsNotEmpty()
  quantity: number;
}

export class GetOrderDto {
```

```
  @IsString()
  @IsNotEmpty()
  orderId: string;
}

import {
  Column,
  CreateDateColumn,
  Entity,
  JoinColumn,
  ManyToOne,
  OneToOne,
  PrimaryGeneratedColumn,
  UpdateDateColumn,
} from 'typeorm';
import { User } from '../../user/entities/user.entity';
import { Bucket } from '../../bucket/entity/bucket.entity';

export enum EStatus {
  PAID = 'paid',
  OPEN = 'open',
}

@Entity()
export class Order {
  @PrimaryGeneratedColumn('uuid')
  id: string;

  @Column()
```

```
  userId: string;

  @ManyToOne(() => User, (user) => user.orders)
  user: User;

  @OneToOne(() => Bucket, (bucket) => bucket.id)
  @JoinColumn()
  bucket: Bucket;

  @Column({
    type: 'enum',
    enum: EStatus,
    default: EStatus.OPEN,
  })
  status: EStatus;

  @CreateDateColumn()
  createdAt: Date;

  @UpdateDateColumn()
  updatedAt: Date;
}

import { User } from '../../user/interface/user.interface';

export interface Order {
  id?: string;
  userId?: string;
  user?: User;
```

```
  createdAt?: Date;
  updatedAt?: Date;
}

import {
  Body,
  Controller,
  Get,
  HttpStatus,
  Post,
  Response,
  Request,
  UseGuards,
} from '@nestjs/common';
import { OrderService } from './order.service';
import { AddToOrderDto, GetOrderDto, OrderDto } from './dto/order.dto';
import { AuthGuard } from '../auth/auth.guard';
import { BucketService } from '../bucket/bucket.service';

@Controller('order')
export class OrderController {
  constructor(
    private readonly orderService: OrderService,
    private readonly bucketService: BucketService,
  ) {}

  @Post('create')
  @UseGuards(AuthGuard)
  async createOrder(@Response() res: any, @Body() data: OrderDto) {
```

```
  return res
    .status(HttpStatus.OK)
    .json(await this.orderService.createNewOrder(data));
}

@Post('add')
@UseGuards(AuthGuard)
async add(@Response() res: any, @Body() data: AddToOrderDto) {
  return res
    .status(HttpStatus.OK)
    .json(await this.bucketService.addToOrder(data));
}

@Get('')
@UseGuards(AuthGuard)
async getOrder(@Response() res: any, @Body() data: GetOrderDto) {
  return res
    .status(HttpStatus.OK)
    .json(await this.bucketService.getBucket(data.orderId));
}

@Get('history')
@UseGuards(AuthGuard)
async getOrderHistory(@Response() res: any, @Request() req: any) {
  return res
    .status(HttpStatus.OK)
    .json(await this.bucketService.getBucketHistory(req.user.id));
}
```

```
  @Post('pay')
  @UseGuards(AuthGuard)
  async pay(@Response() res: any, @Body() data: GetOrderDto) {
    return res
      .status(HttpStatus.OK)
      .json(await this.orderService.pay(data.orderId));
  }
}

import { forwardRef, Module } from '@nestjs/common';
import { OrderService } from './order.service';
import { OrderController } from './order.controller';
import { TypeOrmModule } from '@nestjs/typeorm';
import { Order } from './entity/order.entity';
import { BucketModule } from '../bucket/bucket.module';

@Module({
                imports: [forwardRef(() => BucketModule),
TypeOrmModule.forFeature([Order])],
  providers: [OrderService],
  controllers: [OrderController],
  exports: [OrderService],
})
export class OrderModule {}

import { HttpException, HttpStatus, Injectable } from '@nestjs/common';
import { OrderDto } from './dto/order.dto';
import { InjectRepository } from '@nestjs/typeorm';
import { In, Repository } from 'typeorm';
```

```
import { EStatus, Order } from './entity/order.entity';

@Injectable()
export class OrderService {
  constructor(
    @InjectRepository(Order)
    private readonly orderRepository: Repository<Order>,
  ) {}

  async createNewOrder(data: OrderDto) {
    return this.orderRepository.save({
      userId: data.userId,
    });
  }

  async findAllOrdersByUSerId(userId: string) {
    const orders = await this.orderRepository.find({
      where: {
        userId,
      },
      select: { id: true },
    });

    const orderIds = orders.map((order) => order.id);
    return orderIds;
  }

  async findHistory(orderIds: string[]) {
    return this.orderRepository.find({
```

```
      where: {
        id: In(orderIds),
        status: EStatus.PAID,
      },
      relations: ['bucket'],
    });
  }

  async pay(orderId: string) {
    const order = await this.orderRepository.findOne({
      where: {
        id: orderId,
      },
    });

    if (!order) {
      throw new HttpException(
        'Order with this id is incorrect',
        HttpStatus.NOT_FOUND,
      );
    }

    if (order.status === EStatus.PAID) {
                    throw new HttpException('Order is already paid', HttpStatus.BAD_REQUEST);
    }

    return this.orderRepository.save({
      id: orderId,
```

```
      status: EStatus.PAID,
    });
  }
}

import {
  Column,
  CreateDateColumn,
  Entity,
  OneToMany,
  PrimaryGeneratedColumn,
  UpdateDateColumn,
} from 'typeorm';
import { Order } from '../../order/entity/order.entity';

@Entity()
export class User {
  @PrimaryGeneratedColumn('uuid')
  id: string;

  @Column()
  username: string;

  @Column()
  email: string;

  @Column()
  password: string;
```

```
  @OneToMany(() => Order, (order) => order.user)
  orders: Order[];

  @CreateDateColumn()
  createdAt: Date;

  @UpdateDateColumn()
  updatedAt: Date;
}

import { Order } from '../../order/interface/order.interface';

export interface User {
  id?: string;
  username?: string;
  email?: string;
  password?: string;
  orders?: Order[];
  createdAt?: Date;
  updatedAt?: Date;
}

import { Module } from '@nestjs/common';
import { UserService } from './user.service';
import { UserController } from './user.controller';
import { TypeOrmModule } from '@nestjs/typeorm';
import { User } from './entities/user.entity';

@Module({
```

```
  imports: [TypeOrmModule.forFeature([User])],
  providers: [UserService],
  controllers: [UserController],
  exports: [UserService],
})
export class UserModule {}

import { HttpException, HttpStatus, Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { User } from './entities/user.entity';
import { Repository } from 'typeorm';
import { RegisterDto } from '../auth/dto/auth.dto';

import * as bcrypt from 'bcrypt';

@Injectable()
export class UserService {
  private salt = 10;

  constructor(
    @InjectRepository(User) private readonly userRepository: Repository<User>,
  ) {}

  async createUser(data: RegisterDto) {
    const ifUserExist = await this.userRepository.findOne({
      where: {
        email: data.email.trim(),
      },
    });
```

```
  if (ifUserExist) {
    throw new HttpException(
      'User with this name already exists.',
      HttpStatus.BAD_REQUEST,
    );
  }

  return this.userRepository.save({
    email: data.email.trim(),
    username: data.username.trim(),
    password: await this.getHash(data.password),
  });
}

async getUserByEmail(email: string): Promise<User> {
  return await this.userRepository.findOne({
    where: { email: email.trim() },
  });
}

async getUserById(userId: string): Promise<User> {
  return await this.userRepository.findOne({
    where: { id: userId },
  });
}

async getHash(password: string | undefined): Promise<string> {
  return bcrypt.hash(password, this.salt);
```

```
  }

  async compareHash(
    password: string | undefined,
    hash: string | undefined,
  ): Promise<boolean> {
    return bcrypt.compare(password, hash);
  }
}
```

ДОДАТОК Б. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

Державний університет телекомунікацій
Навчально-науковий інститут інформаційних технологій
Кафедра Комп'ютерних наук

Дипломна робота
на тему: «Проектування та розробка повноцінного **REST API** для створення онлайн-магазину»

Виконав: студент 6 курсу, група КНДМ-62
Бондаренко Кирило Ігорович

Характеристики дипломної роботи

Наукове завдання – Подолання проблем у процесі проектування та розробки повноцінного REST API для створення онлайн-магазину з фокусом на високий рівень безпеки та захисту даних користувачів.

Мета роботи – Дослідження та порівняння ефективності різних стратегій та методів захисту в контексті проектування REST API для електронного магазину. Розробка та впровадження найоптимальніших рішень щодо безпеки та конфіденційності.

Об'єкт дослідження – Процес створення та функціонування додатку.

Предмет дослідження – Проектування та розробка безпечного та надійного REST API для ефективного управління онлайн-магазином, з основним акцентом на захист особистих даних.

Актуальні проблеми в галузі

Безпека та конфіденційність. Безпека та конфіденційність є одними з найбільш критичних аспектів в електронній комерції та цифрових сервісах. Забезпечення високого рівня безпеки є обов'язковим завданням для будь-якої онлайн-платформи.

Ефективність маркетингу. Ефективність маркетингу та реклами в електронній комерції та цифрових сервісах визначається низкою факторів, які взаємодіють для привертання уваги аудиторії та забезпечення конверсії.

Лояльність та утримання клієнтів. Лояльність та утримання клієнтів стали критичними аспектами в електронній комерції, де конкуренція велика, а можливості перейти до іншого бренду завжди на вагу. Утримання клієнтів вимагає не лише залучення нових, але й створення позитивного досвіду для існуючих.

Огляд цифрових сервісів

Веб-сайт. Це візитівка бізнесу, де покупці можуть не лише переглядати асортимент товарів, але й легко та зручно здійснювати покупки через онлайн-платформу.

Веб-додаток. В еру мобільності та постійного руху веб-додатки стають невід'ємною частиною сучасного електронного бізнесу. Їхня роль полягає в наданні додаткового рівня зручності та доступності для клієнтів, незалежно від їхнього місця перебування.

Мобільний додаток. У світі, де швидкість та мобільність грають ключову роль, мобільні додатки стають невід'ємною частиною сучасного електронного бізнесу. Вони виступають як важливий інструмент, забезпечуючи клієнтам доступ до магазину зі своїх смартфонів та планшетів.

СУБД **PostgreSql. ACID**

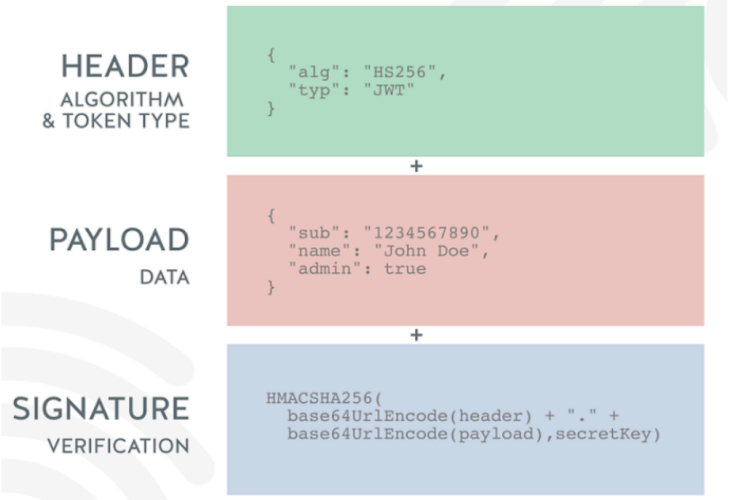
Атомарність (Atomicity): Якщо яка-небудь частина транзакції не може бути виконана, вся транзакція відміняється, і база даних залишається у стані, якщо нічого не трапилося.

Узгодженість (Consistency): Цей принцип забезпечує, що транзакція переводить базу даних з одного стабільного стану в інший. Тобто, якщо база даних була у коректному стані перед транзакцією, вона залишається коректною після транзакції.

Ізоляція (Isolation): Принцип ізоляції гарантує, що виконання однієї транзакції не впливає на інші транзакції, які відбуваються одночасно.

Тривалість (Durability): Цей принцип гарантує, що зміни, внесені в базу даних після успішного завершення транзакції, залишаються незмінними і не втрачаються.

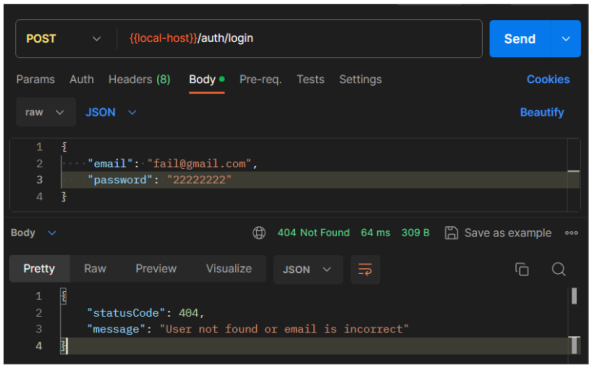
Основа первинного захисту - **JWT TOKEN**

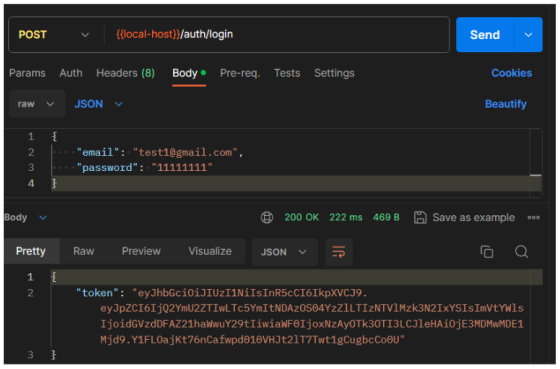


Вибір протоколу

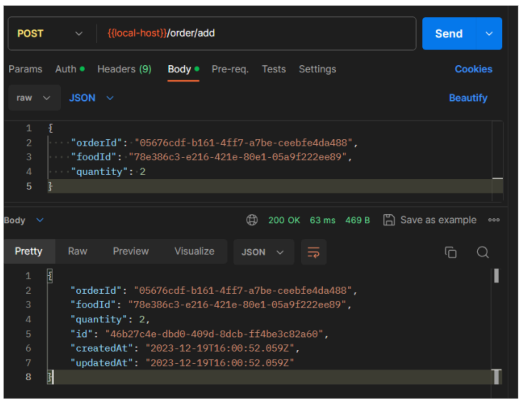
Характеристика	HTTP	WebSockets	FTP
Простота	Простий у використанні	Дещо складніше налаштування, повнодуплексне взаємодія	Складний для налаштування
Універсальність	Широко використовується для передачі веб-контенту	Часто використовується в реальному часі	Менше універсальний
Сумісність з браузерами	Практично всі сучасні браузери повністю підтримують	Підтримується більшістю, але потребує підтримки на рівні клієнта і сервера	Часткова вбудована підтримка
Швидкість передачі даних	Середня	Висока, низька затримка	Середня

Авторизація

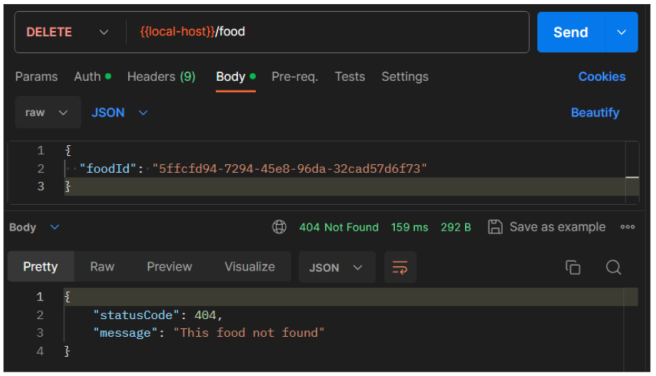




Приклади тестування додатку



Запит на додавання товару



Запит на видалення товару

Висновки

Швидкість використання: Додаток відзначається високою швидкістю реакції та ефективністю у виконанні завдань. Швидка завантаження та плавна робота інтерфейсу забезпечують зручне користування.

Надійність: Додаток проявляє високу стабільність у різних умовах та на різних пристроях.

Захист даних: Високий рівень захисту особистих даних користувачів гарантує конфіденційність інформації. Використання сучасних технологій шифрування та механізмів безпеки забезпечує надійний захист від несанкціонованого доступу.

Ефективність: Додаток відповідає високим стандартам продуктивності та виконує свої функції ефективно. Використання оптимізованих алгоритмів та оптимальних ресурсів дозволяє досягати високої продуктивності.