ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ

Навчально-науковий інститут денної освіти

Форма навчання денна

Кафедра комп’ютерних наук та інформаційних технологій

Допускається до захисту

Завідувач кафедри

_____ Олена ОЛЬХОВСЬКА
(підпис)

«___»_____202 4р.

КВАЛІФІКАЦІЙНА РОБОТА на тему

РОЗРОБКА СЕРВЕРНОЇ ЧАСТИНИ ДЛЯ ІНТЕРНЕТ-МАГАЗИНУ «ECOBAZAR»

зі спеціальності 122 Комп’ютерні науки освітня програма «Комп’ютерні науки» ступеня бакалавра

Виконавець роботи Каряковцев Вадим Олександрович

_____«____»_____ 2024 р.
(підпис)

Науковий керівник к. ф.-м. н. Ольховська Олена Володимирівна

_____«____»_____ 2024 р.
(підпис)

Рецензент

_____

ПОЛТАВА 2024

Завідувач кафедри _____Олена ОЛЬХОВСЬКА
«____»_____ 202_ р.

ЗАВДАННЯ ТА КАЛЕНДАРНИЙ ГРАФІК
ВИКОНАННЯ КВАЛІФАКАЦІЙНОЇ РОБОТИ

на тему «Програмне забезпечення для відслідковування академічних заборгованостей студентів академічної групи»
зі спеціальності 122 «Комп'ютерні науки»
освітня програма «Комп'ютерні науки»
ступеня бакалавр
Прізвище, ім'я, по батькові Каряковцев Вадим Олександрович
Затверджена наказом ректора № 177-Н від «10» вересня 2024 р.
Термін подання студентом роботи «___» _____ 20__ р.
Вихідні дані до кваліфікаційної роботи: публікації та лекційний матеріал з теми, навчальні відеоматеріали, технічна документація.
Зміст пояснювальної записки (перелік питань, які потрібно розробити)
ВСТУП
1. ПОСТАНОВКА ЗАДАЧІ
2. ІНФОРМАЦІЙНА ЧАСТИНА
2.1. Огляд типів веб-сайтів та переваг інтернет магазинів
2.2. Основні етапи створення інтернет-магазину
2.3. Інструменти та технології, використані для реалізації серверної частини
3. ТЕОРЕТИЧНА ЧАСТИНА
3.1. Проектування бази даних для інтернет-магазину
3.2. Особливості побудови реляційних баз даних для веб-застосунків
3.3. Порівняння реляційного підходу з документо-орієнтованим
4. ПРАКТИЧНА ЧАСТИНА
4.1. Налаштування середовища розробки
4.2. Створення моделей даних
4.3. Встановлення зв'язків між моделями
4.4. Реалізація API
4.5. Тестування та перевірка функціональності
ВИСНОВКИ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ
Перелік графічного матеріалу: Необхідні ілюстрації.
Консультанти розділів кваліфікаційної роботи

Розділ	ПІП, посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Постановка задачі			
Інформаційний огляд			
Теоретична частина			

Практична частина			

Календарний графік виконання кваліфікаційної роботи

Зміст роботи	Термін виконання	Фактичне виконання
1. Вступ		
2. Вивчення методичних рекомендацій та стандартів та звіт керівнику		
3. Постановка задачі		
4. Інформаційний огляд джерел бібліотек та інтернету		
5. Теоретична частина		
6. Практична частина		
7. Закінчення оформлення		
8. Доповідь студента на кафедрі		
9. Доробка (за необхідністю), рецензування		

Дата видачі завдання «____»_____ 202_ р.
Здобувач вищої освіти Каряковцев Вадим Олександрович
Науковий керівник к. ф.-м. н. Олена ОЛЬХОВСЬКА
Результати захисту кваліфікаційної роботи
Кваліфікаційна робота оцінена на

_____

(балів, оцінка за національною шкалою, оцінка за ECTS)

Протокол засідання ЕК № ____ від «____» _____ 202_ р.
Секретар ЕК_____ _____

(підпис) (ініціали та прізвище)

Затверджую
Зав. кафедрою _____
к.ф.-м.н. Олена ОЛЬХОВСЬКА
«____» _____ 202_ р.

Погоджено
Науковий керівник _____
к.ф.-м.н. Олена ОЛЬХОВСЬКА
«____» _____ 202_ р.

План

кваліфікаційної роботи на тему
«розробка серверної частини для інтернет-магазину “ecobazar”»
зі спеціальності 122 Комп’ютерні науки
освітня програма 122 Комп’ютерні науки
ступеня бакалавра
Каряковцев Вадим Олександрович
Прізвище, ім'я, по батькові

ВСТУП
1. ПОСТАНОВКА ЗАДАЧІ
2. ІНФОРМАЦІЙНА ЧАСТИНА
2.1. Огляд типів веб-сайтів та переваг інтернет магазинів
2.2. Основні етапи створення інтернет-магазину
2.3. Інструменти та технології, використані для реалізації серверної частини
3. ТЕОРЕТИЧНА ЧАСТИНА
3.1. Проектування бази даних для інтернет-магазину
3.2. Особливості побудови реляційних баз даних для веб-застосунків
3.3. Порівняння реляційного підходу з документо-орієнтованим
4. ПРАКТИЧНА ЧАСТИНА
4.1. Налаштування середовища розробки
4.2. Створення моделей даних
4.3. Встановлення зв'язків між моделями
4.4. Реалізація API
4.5. Тестування та перевірка функціональності
ВИСНОВКИ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

Здобувач вищої освіти _____ Каряковцев Вадим

«____» _____ 202_ р.

РЕФЕРАТ

Записка: 60 с., 30 рис., 3 таблиці, 19 джерел.
Об’єкт розробки — серверна частина інтернет-магазину “Ecobazar”
Мета роботи — створення функціональної, надійної та масштабованої серверної частини для інтернет-магазину продуктів харчування “Ecobazar”, що забезпечує обробку користувацьких запитів, збереження даних і безпечну взаємодію з клієнтською частиною.

- Методи реалізації — програмну реалізацію було виконано із застосуванням:
- Node.js — як середовище для виконання JavaScript-коду на сервері;
- Express.js — як веб-фреймворку для побудови RESTful API;
- PostgreSQL — реляційної СУБД для зберігання даних;
- Sequelize ORM — для зручної взаємодії з базою даних через об’єктно-реляційне представлення;
- JWT (jsonwebtoken) — для реалізації системи авторизації та безпечної аутентифікації;
- Postman — для тестування HTTP-запитів та перевірки коректності логіки;
- Git + GitHub — для контролю версій і командної роботи;
- Render — як хмарного хостингу для деплою backend-серверу та бази даних.

У результаті тестування були перевірені всі основні маршрути API: авторизація, управління товарами, кошиком, категоріями, взаємодія з базою даних. Виявлені помилки були оперативно виправлені. Система демонструє стабільність, відповідність вимогам і готовність до розширення.

Розробка даної серверної частини є актуальною, оскільки онлайн-торгівля потребує надійних і захищених інфраструктурних рішень. “Ecobazar” — це приклад сучасного веб-додатку, що дає змогу реалізувати повноцінний сервіс купівлі продуктів через Інтернет з можливістю швидкого масштабування під реальні бізнес-потреби.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

Умовні позначення, символи, скорочення, терміни	Пояснення умовних позначень, скорочень, символів
API	Application Programming Interface — інтерфейс прикладного програмування, набір готових методів для взаємодії між клієнтом і сервером
Backend	Серверна частина застосунку, яка відповідає за логіку, обробку даних, зв’язок із базою даних
CRUD	Create, Read, Update, Delete — базові операції з ресурсами в базі даних
DB / БД	Database — база даних, у якій зберігається інформація про товари, користувачів, замовлення
Express.js	Легковажний фреймворк для Node.js, що спрощує створення веб-додатків і API

HTTP	HyperText Transfer Protocol — протокол передавання гіпертексту в Інтернеті
Node.js	Середовище виконання JavaScript на сервері, побудоване на рушії V8
ORM	Object-Relational Mapping — підхід до роботи з базами даних через об’єкти моделей
PostgreSQL	Потужна об'єктно-реляційна система керування базами даних з відкритим кодом
REST	Representational State Transfer — архітектурний стиль взаємодії з веб-сервісами
Sequelize	ORM-бібліотека для Node.js, що дозволяє працювати з реляційними БД через JavaScript
UUID	Universally Unique Identifier — унікальний ідентифікатор, який використовується, зокрема, для назв файлів

JSON	JavaScript Object Notation — формат зберігання та обміну структурованими даними
Postman	Інструмент для тестування API через надсилання HTTP-запитів
Middleware	Проміжне програмне забезпечення, яке обробляє запити між клієнтом і сервером
bcrypt	Бібліотека для хешування паролів з метою забезпечення безпеки
.env	Файл конфігурації, в якому зберігаються змінні середовища, наприклад, дані підключення до БД
Render	Хмарна платформа для хостингу веб-додатків і баз даних
MVC	Model-View-Controller — шаблон проєктування для структурування програмного коду

ВСТУП

У сучасному цифровому середовищі електронна комерція займає провідні позиції серед галузей, що стрімко розвиваються. Однією з її важливих складових є інтернет-магазини, які дозволяють користувачам здійснювати покупки швидко, зручно та без потреби відвідування фізичних торгових точок. Особливої актуальності набувають продуктові онлайн-магазини, адже вони забезпечують базові життєві потреби людей у режимі 24/7.

Предметом розробки даної кваліфікаційної роботи є **створення серверної частини інтернет-магазину "Ecobazar"**, що спеціалізується на онлайн-продажі продуктів харчування. Основна мета — забезпечити стабільне функціонування, безпеку та гнучкість у взаємодії клієнтів із веб-застосунком, реалізувавши повноцінну backend-логіку за допомогою сучасного технологічного стеку.

Серверна частина веб-додатка (backend) є ядром системи, відповідальним за прийом і обробку запитів, взаємодію з базою даних, автентифікацію користувачів, збереження замовлень, формування кошиків, обробку платежів тощо. В рамках даного проєкту обрана технологія **Node.js** як високопродуктивне середовище виконання JavaScript на сервері, а **Express.js** використовується як гнучкий фреймворк для розробки REST API. У якості системи управління базами даних застосовується **PostgreSQL** — реляційна СУБД з підтримкою ACID-транзакцій, складних типів даних, ефективної індексації та широких можливостей масштабування. Для інтеграції з базою даних обрано **ORM-бібліотеку Sequelize**, яка забезпечує зручну роботу з моделями та зв'язками між сутностями на рівні коду.

На відміну від традиційного магазину, інтернет-платформа реалізує всі операції через API-запити. Наприклад, при додаванні товару до кошика або під час реєстрації нового користувача, серверна частина обробляє запит, оновлює відповідні таблиці в базі даних, і формує відповідь для клієнтської частини.

Таким чином, **успішна реалізація backend-логіки є критично важливою** для загальної працездатності системи.

У рамках проєкту також враховано аспекти **безпеки (включаючи JWT-автентифікацію, хешування паролів через bcrypt), масштабованості (через чітку модульну структуру), та інтеграції (шляхом RESTful API для клієнтської частини)**. Крім того, база даних розгорнута на хмарному сервісі **Render**, що забезпечує доступність і стабільність системи за межами локального середовища.

Загальна **мета кваліфікаційної роботи** — реалізувати функціональну, ефективну та захищену серверну частину веб-застосунку “Ecobazar”, яка дозволяє виконувати наступні задачі:

реєстрація та авторизація користувачів;

керування товарами, категоріями та знижками;

створення та збереження кошика користувача;

обробка замовлень та реалізація логіки рейтингу;

повноцінна взаємодія з клієнтською частиною через REST API.

Інтернет-магазин “Ecobazar” — це цифрове рішення, орієнтоване на простоту використання, швидку обробку запитів і гнучке масштабування, що робить його придатним для реального впровадження в комерційне середовище.

Структура дипломної роботи побудована логічно і включає такі розділи:

- Вступ;
- Постановка задачі;
- Теоретична частина;
- Практична частина;
- Висновки;
- Додатки.

Кожен із них містить як теоретичне обґрунтування вибору технологій, так і практичні етапи реалізації, що у сукупності забезпечує повне розкриття теми та досягнення поставлених цілей.

РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ

Основною метою цієї кваліфікаційної роботи є розробка серверної частини веб-додатку **"Ecobazar"** — інтернет-магазину, що надає користувачам можливість здійснювати зручні онлайн-замовлення продуктів харчування. Серверна частина відіграє ключову роль у функціонуванні будь-якого сучасного веб-додатку: вона відповідає за логіку обробки запитів, збереження, обробку та передачу даних, а також забезпечує повноцінну взаємодію з клієнтським інтерфейсом через механізми REST API.

Для досягнення поставленої мети було визначено низку взаємопов’язаних завдань, реалізація яких забезпечує повний цикл створення backend-функціоналу:

- **Аналіз існуючих веб-архітектур і систем електронної комерції.** Було проведено дослідження сучасних підходів до побудови інтернет-магазинів, типових архітектур серверної частини, інструментів та технологій, які найчастіше застосовуються у сфері e-commerce. Особливу увагу приділено практикам розробки масштабованих і безпечних рішень.
- **Розробка структури серверної частини.** Було спроєктовано окремі модулі для роботи з користувачами, товарами, кошиком покупок, замовленнями та оплатою. Ці модулі взаємодіють між собою за допомогою HTTP-запитів через REST API. Визначено логіку ендпоінтів, методи запитів (GET, POST, PUT, DELETE) та формат відповіді у форматі JSON.
- **Проєктування та реалізація бази даних.** Розроблено структуру бази даних на основі реляційної СУБД **PostgreSQL**, з дотриманням принципів нормалізації. Було створено таблиці для зберігання інформації про користувачів, товари, категорії, оцінки, кошики та пов’язані дані. Для роботи з базою використано ORM-бібліотеку **Sequelize**, яка дозволяє описувати структуру таблиць у вигляді моделей.

- **Реалізація API з використанням сучасного технологічного стеку.** Було обрано ефективний та популярний стек для розробки серверної частини: **Node.js** у поєднанні з **Express.js** для маршрутизації, а також **PostgreSQL** для надійного зберігання даних. Всі API-запити реалізовані відповідно до стандартів REST. Розробка проводилася з урахуванням розділення логіки по контролерах, маршрутах та middleware.
- **Забезпечення безпеки та валідації даних.** Було впроваджено систему авторизації та автентифікації користувачів за допомогою токенів JWT. Реалізовано перевірку вхідних даних, захист від типових атак (SQL-ін’єкції, XSS), а також передбачено обробку винятків та відповідей сервера згідно зі стандартами HTTP.
- **Тестування серверної логіки.** Для перевірки працездатності та коректності реалізованого API використовувався інструмент **Postman**, який дозволяє емулювати запити до серверу та перевірити відповідь. Було протестовано ключові сценарії: реєстрацію та вхід користувачів, додавання товарів у кошик, обробку помилок та правильність отриманих відповідей.

Усі вищезазначені завдання було детально реалізовано та описано в наступних розділах цієї дипломної роботи. Результатом є повноцінна серверна частина інтернет-магазину “Ecobazar”, яка може бути розгорнута на хмарному хостингу **Render**, підтримує інтеграцію з клієнтським інтерфейсом і готова до масштабування відповідно до потреб майбутніх користувачів.

РОЗДІЛ 2. ІНФОРМАЦІЙНА ЧАСТИНА

2.1 Огляд типів веб-сайтів та переваг інтернет-магазинів

У цифрову епоху вебсайти стали невід’ємною частиною бізнесу, освіти, медіа та розваг. Сайт — це набір вебсторінок, що зберігаються на певному доменному імені та доступні через браузер. Залежно від мети створення, існує кілька основних типів сайтів.

Найпростішим варіантом є сайт-візитка, що містить базову інформацію про компанію, перелік послуг і контактні дані. Такий сайт є візуально лаконічним та ідеально підходить для малого бізнесу або особистого бренду. Ще одним популярним форматом є лендінг (Landing Page) — односторінковий сайт, метою якого є просування конкретного продукту або послуги та мотивація користувача до цільової дії (наприклад, оформлення заявки).

Більш функціональними є корпоративні сайти, які представляють компанію в Інтернеті, мають кілька розділів (про компанію, новини, послуги, вакансії тощо) і служать для побудови іміджу. Окремо варто згадати блоги — ресурси, що орієнтовані на регулярні публікації текстів, фотографій чи відео на певну тематику, як правило, з можливістю коментування. Також поширеними є новинні портали, освітні сайти, соціальні мережі та розважальні платформи.

Серед усіх типів вебресурсів особливе місце посідають **інтернет-магазини** — повноцінні онлайн-платформи для продажу товарів чи послуг. Вони дають змогу користувачам не лише ознайомлюватися з асортиментом продукції, а й додавати товари до кошика, оформлювати замовлення, здійснювати оплату та відстежувати доставку. Однією з головних переваг інтернет-магазинів є цілодобова доступність, що дає змогу клієнтам купувати товари у зручний для них час незалежно від локації. Крім того, завдяки інструментам аналітики,

власники магазинів можуть детально вивчати поведінку користувачів, оптимізуючи маркетинг і покращуючи взаємодію з клієнтами.

Також інтернет-магазини дозволяють масштабувати бізнес — додавати нові товари, автоматизувати облік, адаптувати цінову політику та ефективно інтегрувати різні сервіси (наприклад, платіжні шлюзи чи CRM-системи).

2.2 Основні етапи створення інтернет-магазину

Створення інтернет-магазину — це багатоступеневий процес, що поєднує стратегічне планування, технічну реалізацію та UX/UI-дизайн. Кожен етап є важливим і впливає на кінцевий результат.

Першим кроком виступає **формування стратегії**, що передбачає постановку бізнес-цілей, вивчення цільової аудиторії та конкурентного середовища. На цьому етапі визначається, які продукти будуть продаватися, як здійснюватиметься доставка, оплата, обслуговування клієнтів.

Далі розробляється технічне завдання, де описуються функціональні та нефункціональні вимоги: які модулі має включати система (кошик, каталог, авторизація), які зовнішні сервіси будуть підключені (платіжні системи, аналітика тощо), терміни розробки та бюджет.

Проектування структури сайту охоплює створення логічної архітектури вебресурсу — розробку карти сторінок, побудову маршрутизації та взаємозв’язків між модулями. Після цього створюється прототип (або скелет) сайту, в якому визначаються розміщення основних елементів інтерфейсу: меню, фільтрів, кнопок тощо.

Завершальними етапами є розробка **графічного дизайну**, програмування (frontend і backend), інтеграція бази даних, написання API, налаштування серверного середовища. Наприкінці виконується повне **тестування**: функціональне, навантажувальне, безпекове та UI/UX, щоб переконатися в працездатності системи на різних пристроях і у різних браузерах.

2.3 Інструменти та технології, використані для реалізації серверної частини

Серверна частина інтернет-магазину “Ecobazar” реалізована з використанням сучасного технологічного стеку, орієнтованого на швидкість, модульність і зручність підтримки.

Ключовою платформою для розробки backend-частини став **Node.js** — серверне середовище виконання JavaScript, яке дозволяє обробляти тисячі паралельних запитів завдяки неблокуючій моделі введення/виводу. Разом із **Express.js** — потужним веб-фреймворком — створюється гнучка система маршрутів, middleware, а також RESTful API, що використовується для комунікації з клієнтською частиною.

Для зберігання та обробки даних використано реляційну СУБД **PostgreSQL**, яка підтримує складні SQL-запити, транзакції, індексацію, а також дозволяє зберігати дані у форматах JSON, масивах і навіть географічних координатах. У парі з нею використано ORM-бібліотеку **Sequelize**, яка спрощує роботу з моделями, зв’язками між таблицями та міграціями бази даних.

Розгортання backend-частини та самої бази даних було виконано на хмарному сервісі **Render**, що забезпечує стабільність, відмовостійкість і безперервний доступ до системи. Для налагодження і тестування API використовувався **Postman** — зручний інструмент для перевірки запитів, перевірки статусів відповідей, токенів, заголовків тощо.

Контроль версій і командну роботу реалізовано за допомогою **Git** і платформи **GitHub**, що дозволило ефективно зберігати історію змін, працювати з гілками та об'єднувати внески різних учасників команди (за потреби).

При виборі технологій було враховано такі фактори, як:
доступність (всі обрані інструменти мають безкоштовні версії або відкритий вихідний код),масштабованість (можливість обробляти збільшену кількість

запитів і користувачів),підтримка спільноти (документація, форуми, навчальні ресурси),безпека (наявність перевірених бібліотек і best practices).

Таким чином, використаний стек технологій забезпечує надійну основу для функціонування інтернет-магазину та дозволяє швидко масштабувати рішення за потреби.

2.3.1 Переваги PostgreSQL

Серед основних переваг використаної системи управління базами даних PostgreSQL варто виділити її високу надійність, що забезпечується підтримкою транзакцій, обмежень цілісності даних, а також широкими можливостями перевірки цілісності введених записів. PostgreSQL надає розробникам значну гнучкість у роботі завдяки підтримці складних типів даних, таких як JSON, масиви та географічні об’єкти, що дозволяє ефективно зберігати як структуровану, так і напівструктуровану інформацію. Ще однією ключовою характеристикою є масштабованість, яка досягається завдяки можливостям налаштування реплікації та оптимізації запитів до великих обсягів даних. Безпека бази даних підтримується системою ролей і прав доступу, а також можливістю використання шифрування для захисту конфіденційної інформації. Крім того, PostgreSQL чудово інтегрується з ORM-бібліотекою Sequelize, яка дозволяє працювати з базою даних на рівні моделей, спрощуючи розробку логіки взаємодії між сутностями у коді проєкту.

2.3.2 Переваги використання Node.js та Express.js

Платформа Node.js, обрана для реалізації серверної частини, також має низку важливих переваг. Насамперед, вона забезпечує асинхронну обробку запитів, що дозволяє обслуговувати одночасно тисячі підключень без значного навантаження на сервер. Важливою перевагою є уніфікація технологій, оскільки

і клієнтська, і серверна частина написані на JavaScript, що значно спрощує обмін даними та координацію між командами frontend і backend розробників. Розвинена екосистема Node.js, яка включає понад мільйон готових модулів у менеджері npm, дозволяє розробникам швидко впроваджувати нову функціональність без необхідності розробки її з нуля. Використання фреймворку Express.js надало можливість організувати структуру API у зручний та логічно розподілений спосіб, дозволяючи реалізувати основну серверну логіку без надмірного перевантаження коду та з високим рівнем гнучкості.

Цей стек технологій дозволив створити стабільну, масштабовану та безпечну серверну частину інтернет-магазину "Ecobazar", яка відповідає сучасним вимогам електронної комерції.

РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА

3.1 Проектування бази даних для інтернет-магазину

В цьому розділі буде розглянуто проектування структури бази даних для інтернет-магазину.

Список необхідних таблиць:

Таблиця 3.1 – Category містить інформацію про основні категорії товарів.

Поле	Тип	Опис
id	INT	Унікальний ідентифікатор каталогу
name	STRING	Назва групи товарів в каталозі

Таблиця 3.2 – Product містить інформацію про товари.

Поле	Тип	Опис
id	INT	Унікальний ідентифікатор підкаталогу
name	STRING	Назва товару
price	FLOAT	Звичайна ціна
salePrice	FLOAT	Акційна ціна
isSale	BOOL	Поле для позначення товару як акційного
description	TEXT	Опис товару

image	STRING	Фото товару (шлях до фото на сервері)

Таблиця 3.3 – BasketProduct містить інформацію про корзину.

Поле	Тип	Опис
id	INT	Унікальний ідентифікатор корзини
quantity	INT	Кількість товару

Таблиця 3.4 – User містить інформацію про користувача

Поле	Тип	Опис
id	INT	Унікальний ідентифікатор користувача
email	STRING	Email користувача
password	STRING	Пароль користувача
isActive	BOOL	Поле для позначення активності

База даних подана у вигляді блок-схеми (рис.3.1).

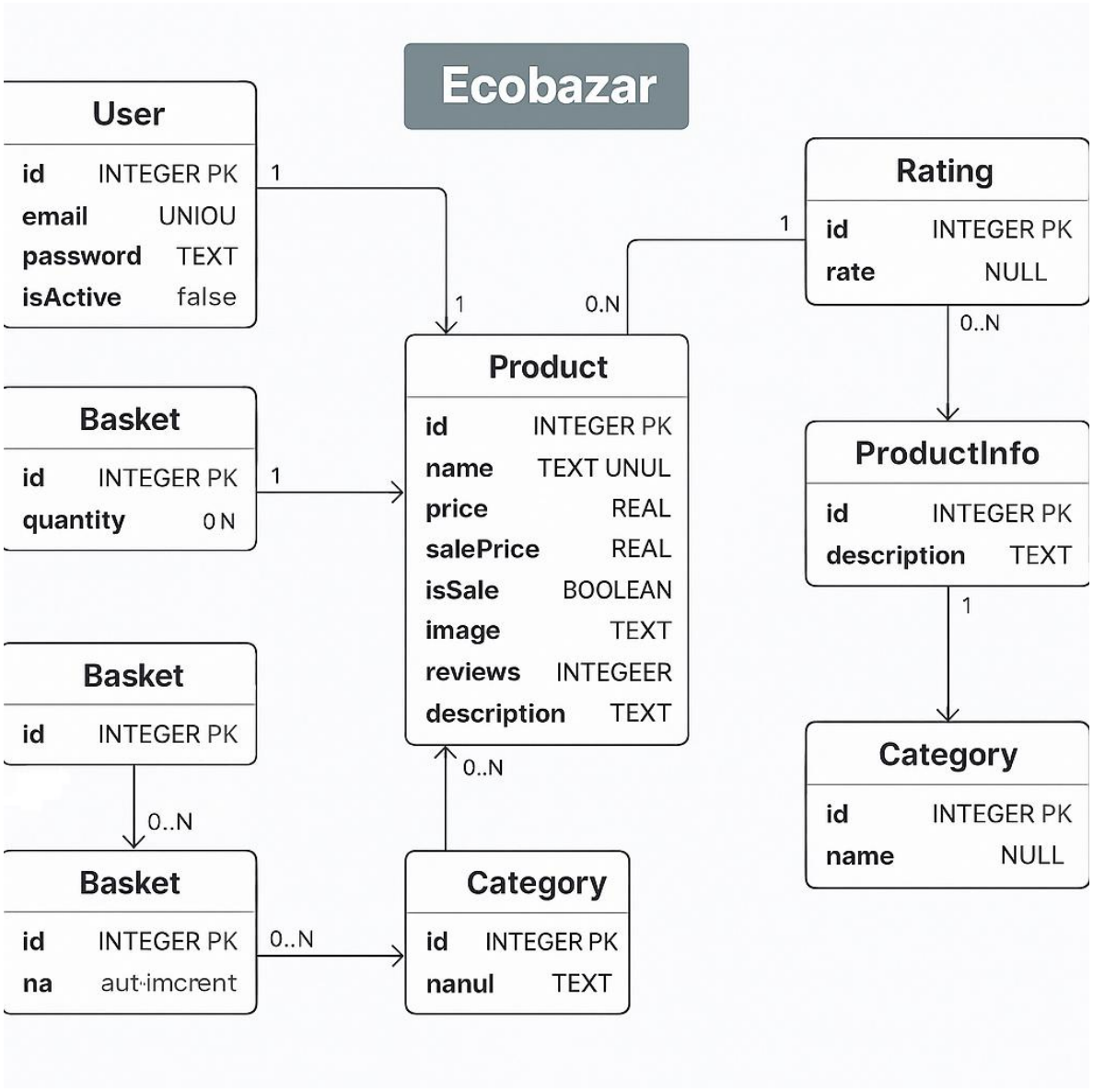


Рисунок 3.1 – Схема Баз даних

Зв’язки між таблицями:

- **User ↔ Basket**
  Кожен користувач (*User*) має один кошик (*Basket*), а кожен кошик належить одному користувачу. User.hasOne(Basket) / Basket.belongsTo(User)
- **User ↔ Rating**
  Один користувач може залишити багато оцінок (*Rating*), кожна оцінка належить конкретному користувачу. User.hasMany(Rating) / Rating.belongsTo(User)

- **Basket ↔ BasketProduct**
  Один кошик може містити багато товарів (*BasketProduct*), кожен з яких належить до конкретного кошика. Basket.hasMany(BasketProduct) / BasketProduct.belongsTo(Basket)
- **Category ↔ Product**
  Одна категорія (*Category*) може містити багато товарів (*Product*), кожен товар належить до певної категорії. Category.hasMany(Product) / Product.belongsTo(Category)
- **Product ↔ Rating**
  Один товар (*Product*) може мати багато оцінок (*Rating*), кожна оцінка стосується певного товару. Product.hasMany(Rating) / Rating.belongsTo(Product)
- **Product ↔ BasketProduct**
  Один товар (*Product*) може зберігатись у багатьох записах *BasketProduct*, тобто входити в різні кошики. Product.hasMany(BasketProduct) / BasketProduct.belongsTo(Product)
- **Product ↔ ProductInfo**
  Кожен товар (*Product*) має одну додаткову інформацію (*ProductInfo*), а кожна така інформація належить до одного товару. Product.hasOne(ProductInfo, {as: 'info'}) / ProductInfo.belongsTo(Product)

3.2 Особливості побудови реляційних баз даних для веб-застосунків

Сучасні веб-додатки, зокрема інтернет-магазини, потребують ефективної організації та зберігання даних, які постійно оновлюються та взаємодіють між собою. У цьому контексті реляційні бази даних (РБД) залишаються найбільш стабільним і перевіреним засобом зберігання структурованої інформації. Реляційна модель баз даних базується на таблицях, які пов’язані між собою

зовнішніми ключами. Така модель дозволяє реалізувати складні логічні зв’язки між сутностями та забезпечити високий рівень узгодженості даних.

Однією з ключових переваг реляційного підходу є можливість підтримки транзакцій, що особливо важливо для електронної комерції, де операції повинні бути або повністю завершеними, або повністю скасованими. Також слід відзначити роль нормалізації даних, яка дозволяє уникати надлишковості, забезпечити логічну цілісність і зменшити ймовірність помилок у збереженні та обробці інформації.

У випадку інтернет-магазину "Ecobazar" вибір на користь PostgreSQL дозволяє скористатися всіма перевагами реляційної архітектури: підтримка складних типів даних (JSON, масиви), сильна система типізації, наявність індексів, тригерів, зовнішніх ключів, а також безперешкодна інтеграція з ORM Sequelize. Також важливо відзначити, що база даних була задеплоєна на хмарній платформі Render, що дало змогу забезпечити цілодобовий доступ до неї, автоматичні бекапи та можливість горизонтального масштабування без зміни архітектури.

Таким чином, побудова РБД на основі PostgreSQL дозволила створити надійну, масштабовану та захищену основу для функціонування інтернет-магазину.

3.3 Порівняння реляційного підходу з документо-орієнтованим

У ході розробки серверної частини веб-додатку постає питання щодо вибору найбільш відповідної системи управління базами даних (СУБД). Залежно від структури і логіки зберігання даних, обирається або реляційна, або NoSQL (зокрема документо-орієнтована) модель.

Документо-орієнтовані СУБД, як-от MongoDB, ідеально підходять для проєктів зі слабкою схемою або її відсутністю. Вони зберігають дані у вигляді JSON-документів, що дозволяє зберігати гнучкі та динамічні структури. Проте,

при потребі реалізації складних зв’язків між сутностями (наприклад, користувачами, товарами, категоріями, замовленнями), реляційна модель виявляється значно зручнішою та ефективнішою.

Для інтернет-магазину "Ecobazar", в якому існує чітка структура взаємопов’язаних таблиць, реляційна модель бази даних є більш доцільною. PostgreSQL як представник РБД забезпечує надійність, транзакційність, а також можливість встановлення складних зв’язків. Це дає змогу розробнику не лише контролювати логіку даних, але й забезпечувати її цілісність на всіх етапах життєвого циклу програми.

Варто також враховувати, що реляційні СУБД краще масштабуються вертикально та мають розвинуту систему безпеки доступу до даних, що є критично важливим для e-commerce. У свою чергу, документо-орієнтовані бази часто потребують додаткової обробки логіки на рівні програми, що ускладнює супровід та розвиток великомасштабних систем.

Отже, хоча документо-орієнтовані СУБД мають свої переваги, саме реляційний підхід дозволив найбільш повно задовольнити потреби функціональної логіки інтернет-магазину "Ecobazar".

РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА

У цьому розділі буде детально розглянуто реалізацію серверної частини інтернет-магазину "Ecobazar", включаючи створення бази даних, опис моделей, налаштування зв’язків між сутностями, запуск локального середовища, а також розгортання REST API для взаємодії з фронтендом.

4.1 Налаштування середовища розробки

Розробка серверної частини веб-додатку "Ecobazar" розпочалася зі створення зручного та масштабованого середовища для роботи з сучасним JavaScript-стеком. Як основну платформу було обрано Node.js, що дає змогу виконувати JavaScript-код на сервері та забезпечує асинхронну обробку великої кількості запитів одночасно [2]. Завдяки цьому досягається висока продуктивність, що критично важливо для онлайн-магазину, де може одночасно працювати багато користувачів.

В якості фреймворку для створення HTTP-маршрутів та middleware обрано Express.js, який завдяки своїй гнучкості дозволив швидко реалізувати REST API. Для зберігання даних використовується PostgreSQL — надійна реляційна СУБД, яка підтримує транзакції, зовнішні ключі та індексацію, що робить її ідеальною для електронної комерції [6]. З'єднання з базою організовано через ORM-бібліотеку Sequelize, що дає змогу маніпулювати даними через об'єктні моделі, замість написання SQL-запитів вручну.

На початковому етапі було ініціалізовано проєкт за допомогою npm init, після чого встановлено ключові залежності: express, sequelize, pg, dotenv, cors, bcrypt, jsonwebtoken, nodemon (для автоматичного перезапуску сервера при зміні коду). Перелік усіх пакетів можна побачити у файлі package.json (Рисунок 4.1).

Рис. 4.1 – package.json

Далі було створено структуровану архітектуру проєкту. Було виділено окремі каталоги для моделей (models/), контролерів (controllers/), маршрутів (routes/), підключення до бази (db/) та проміжних обробників (middleware/). Такий підхід відповідає принципам розділення відповідальності (Separation of Concerns), що полегшує розширення і підтримку коду в майбутньому. Приклад структури проєкту наведено нижче (Рисунок 4.2).

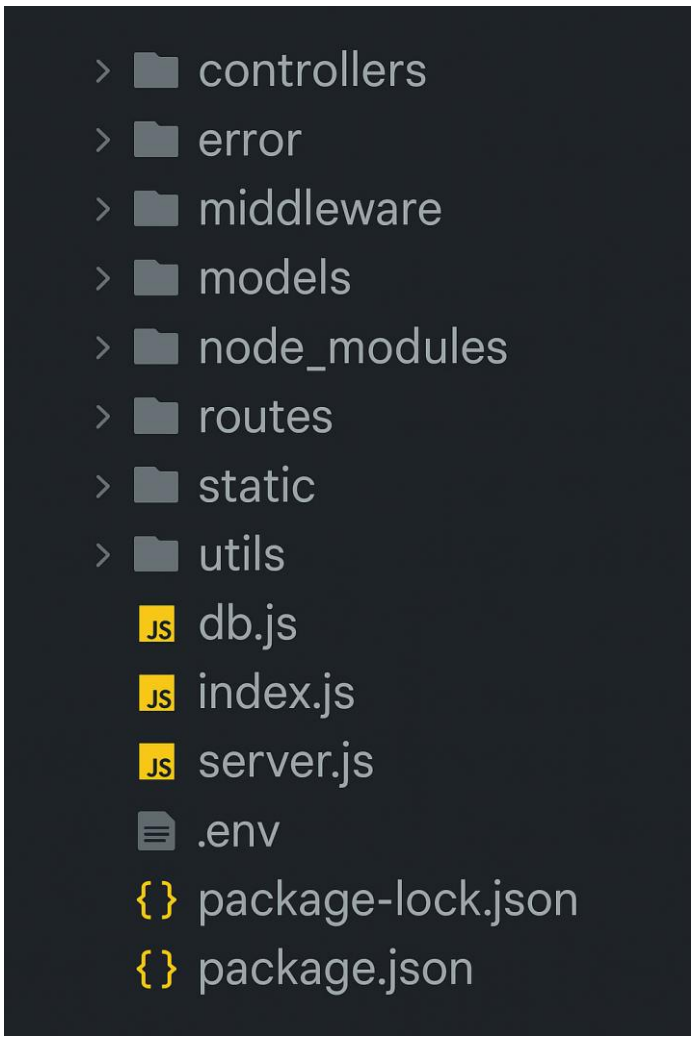


Рис 4.2 – Структура проекту

Параметри підключення до бази зберігаються у файлі .env, який містить змінні середовища, такі як DB_NAME, DB_USER, DB_PASSWORD, PORT, і підключається через бібліотеку dotenv. Це підвищує безпеку та гнучкість (Рисунок 4.3).

Рисунок 4.3 – .env (DB_USER та DB_PASSWORD не показано в цілях безпеки)

Підключення до бази даних здійснюється у файлі index.js, де ініціалізується Sequelize з урахуванням параметрів підключення. У цьому ж файлі запускається метод authenticate() для перевірки з’єднання (Рисунок 4.4).

```
index.js > ...
1   require('dotenv').config();
2   const express = require('express');
3   const cors = require('cors');
4   const sequelize = require('./db');
5   const models = require('./models');
6   const router = require('./routes/router');
7   const fileUpload = require('express-fileupload');
8   const errorHandler = require('./middleware/ErrorHandlingMiddleware');
9   const path = require('path');
10
11  const PORT = process.env.PORT || 5000;
12
13  const app = express(); // App
14  app.use(cors()); // Middleware
15  app.use(express.json()); // Middleware in JSON format
16  app.use(express.static(path.resolve(__dirname, 'static')));
17  app.use(fileUpload({}));
18  app.use('/api', router);
19  app.use('/', (req, res) => { res.status(200).json({message:"hello world!"})})
20  app.use(errorHandler); // Middleware that works with errors should be registered at the end
21
22  const start = async () => {
23      try {
24          await sequelize.authenticate();
25          await sequelize.sync({ alter: true });
26          app.listen(PORT, () => console.log('Server started on PORT ' + PORT));
27      } catch (e) {
28          console.log(e);
29      }
30  }
31  start();
```

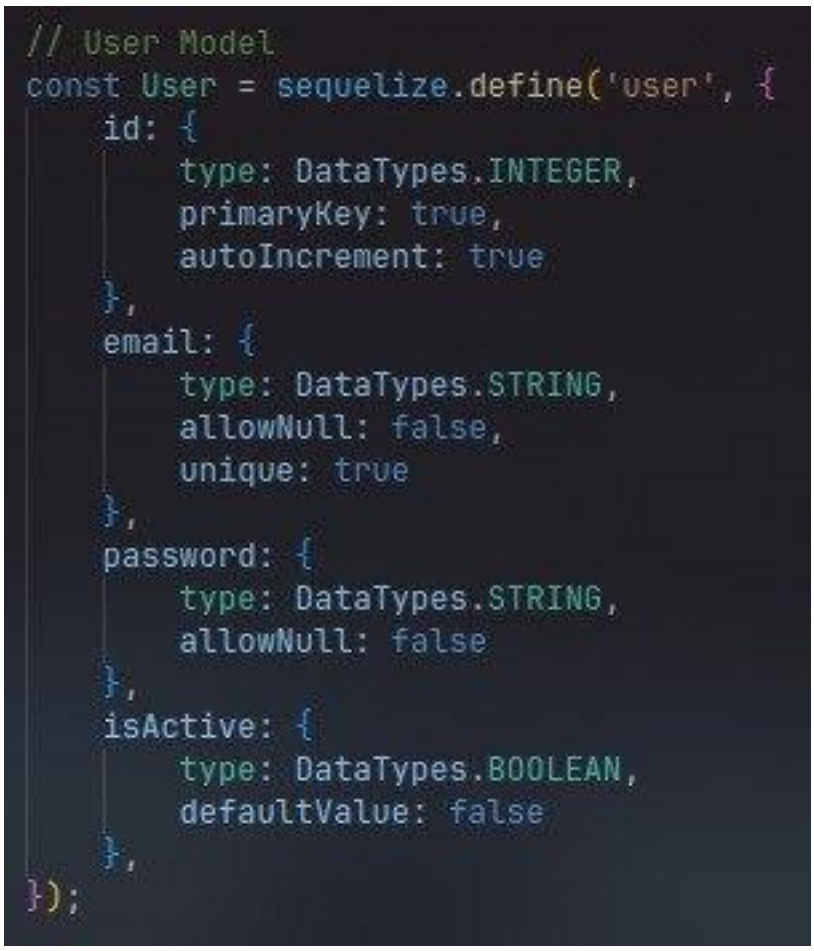
Рис. 4.4 – index.js

Таким чином, було сформовано повноцінне середовище для запуску серверної частини проєкту, яке легко розгортається локально та може бути адаптоване під хмарні сервіси на кшталт Render або Heroku.

4.2 Створення моделей даних

Другим етапом після налаштування середовища стало створення моделей даних, які відображають структуру об’єктів у базі та логіку бізнес-процесів інтернет-магазину. Моделі створювалися за допомогою ORM Sequelize, яка дозволяє описувати таблиці у вигляді класів з атрибутами, типами даних і правилами валідації.

Однією з перших моделей стала User, яка містить поля email, password та isActive. Для захисту паролів передбачено використання хешування через bcrypt у функції register. Усі паролі перед записом до бази проходять процес шифрування (Рисунок 4.5).



```
// User Model
const User = sequelize.define('user', {
    id: {
        type: DataTypes.INTEGER,
        primaryKey: true,
        autoIncrement: true
    },
    email: {
        type: DataTypes.STRING,
        allowNull: false,
        unique: true
    },
    password: {
        type: DataTypes.STRING,
        allowNull: false
    },
    isActive: {
        type: DataTypes.BOOLEAN,
        defaultValue: false
    },
});
```

Рис. 4.5 – Модель User

Далі була створена модель Product, яка зберігає інформацію про товари: назву, ціну, знижку, зображення, опис тощо. Це одна з ключових моделей, яка використовується у багатьох запитах, зокрема при формуванні списку товарів або вмісту кошика (Рисунок 4.6).

```
// Product Model
const Product = sequelize.define('product', {
    id: {
        type: DataTypes.INTEGER,
        primaryKey: true,
        autoIncrement: true
    },
    name: {
        type: DataTypes.STRING,
        unique: true,
        allowNull: false
    },
    price: {
        type: DataTypes.FLOAT,
        allowNull: false
    },
    salePrice: {
        type: DataTypes.FLOAT
    },
    isSale: {
        type: DataTypes.BOOLEAN,
        defaultValue: false
    },
    image: {
        type: DataTypes.STRING,
        allowNull: true
    },
        rating: {
        type: DataTypes.INTEGER,
        defaultValue: 0
    },
    reviews: {
        type: DataTypes.INTEGER,
        defaultValue: 0
    },
    description: {
        type: DataTypes.TEXT,
        allowNull: false
    }
});
```

Рис. 4.6 – Модель Product

Модель Category дозволяє групувати товари за категоріями — овочі, м’ясо, молочна продукція тощо. Вона містить поле name та має зв’язок «один до багатьох» із товарами (Рисунок 4.7).

```
// Category Model
const Category = sequelize.define('category', {
    id: {
        type: DataTypes.INTEGER,
        primaryKey: true,
        autoIncrement: true
    },
    name: {
        type: DataTypes.STRING,
        allowNull: false
    }
});
```

Рис. 4.7 – Модель Category

Також реалізовано моделі Basket, BasketProduct, Rating та ProductInfo, кожна з яких відповідає за окрему бізнес-логіку: додавання товарів у кошик, оцінювання товарів користувачами, зберігання додаткової інформації про продукт тощо. Структуру моделей було розроблено з урахуванням принципів нормалізації бази даних, щоб уникнути дублювання даних та забезпечити узгодженість записів (Рисунок 4.8, 4.9, 5.0).

```
// Basket Model
const Basket = sequelize.define('basket', {
    id: {
        type: DataTypes.INTEGER,
        primaryKey: true,
        autoIncrement: true
    }
});
```

Рис. 4.8 – Модель Basket

```
// BasketProduct Model
const BasketProduct = sequelize.define('basket_product', {
    id: {
        type: DataTypes.INTEGER,
        primaryKey: true,
        autoIncrement: true
    },
    quantity: {
        type: DataTypes.INTEGER,
        defaultValue: 0,
        validate: { min: 1, max: 100 }
    }
});
```

Рис. 4.9 – Модель BasketProduct

```
// ProductInfo Model
const ProductInfo = sequelize.define('product_info', {
    id: {
        type: DataTypes.INTEGER,
        primaryKey: true,
        autoIncrement: true
    },
    description: {
        type: DataTypes.STRING,
        allowNull: true
    },
    benefits: {
        type: DataTypes.ARRAY(DataTypes.STRING),
        allowNull: true,
        defaultValue: []
    }
});
```

Рис. 4.10 – Модель ProductInfo

Усі моделі пов’язані між собою відповідними асоціаціями, які описано в наступному підрозділі. Це дозволяє формувати складні запити, включати пов’язані об'єкти в один запит (include) та зручно працювати з транзакціями.

Загалом, створення моделей даних забезпечило гнучку, безпечну та масштабовану структуру даних, яка легко адаптується до змін у бізнес-логіці веб-додатку.

4.3 Встановлення зв’язків між моделями

```
// Links
User.hasOne(Basket);
Basket.belongsTo(User);

User.hasMany(Rating);
Rating.belongsTo(User);

User.hasOne(Basket);
Basket.belongsTo(User);

Basket.hasMany(BasketProduct);
BasketProduct.belongsTo(Basket);

Category.hasMany(Product);
Product.belongsTo(Category);

Product.hasMany(Rating);
Rating.belongsTo(Product);

Product.hasMany(BasketProduct);
BasketProduct.belongsTo(Product);

Product.hasOne(ProductInfo, {as: 'info'});
ProductInfo.belongsTo(Product);
```

Рисунок 4.11 – зв’язки між моделями таблиць

У межах проєктування серверної частини інтернет-магазину "Ecobazar" було реалізовано повний набір асоціацій між сутностями бази даних. Це дозволило забезпечити цілісність даних, автоматично створити необхідні зовнішні ключі та надалі ефективно будувати складні запити за допомогою ORM-бібліотеки Sequelize.

Перш за все, між моделями User та Basket встановлено зв’язок «один до одного», що відображає логіку — кожен користувач має лише один кошик, і кожен кошик належить конкретному користувачу. Це дає змогу прив’язувати замовлення до авторизованого клієнта та зберігати їхню історію.

Наступним кроком було визначення зв’язку між моделями Basket та BasketProduct. Оскільки один кошик може містити кілька різних товарів, реалізовано зв’язок «один до багатьох». Модель BasketProduct відіграє роль проміжної сутності, яка також містить додаткову інформацію про кількість кожного товару.

Модель Product має зв’язок з BasketProduct, що дозволяє кожному товару бути присутнім у кількох записах — наприклад, у кошиках різних користувачів. Таким чином, реалізується двосторонній зв’язок між товарами та їхнім представленням у кошиках.

Окрему увагу було приділено моделі Category, яка об’єднує всі товари у відповідні групи. Один запис у таблиці категорій пов’язаний з кількома товарами, які належать до цієї категорії, що дозволяє ефективно фільтрувати продукцію на сайті.

Додатково кожен товар може мати кілька оцінок, залишених користувачами. Для цього встановлено зв’язок між Product і Rating, а також між User і Rating, що забезпечує відслідковування, хто саме залишив певну оцінку конкретному товару.

Для зберігання додаткової специфікації товару — такої як технічні характеристики чи опис переваг — було створено модель ProductInfo. Вона має асоціацію «один до одного» з основною моделлю Product, яка реалізується через псевдонім (alias) info.

Ці зв’язки наведено у вигляді коду на рисунку вище (рис. 4.11), де показано структуру асоціацій між моделями у Sequelize. Така архітектура дає змогу ефективно реалізувати бізнес-логіку інтернет-магазину та гнучко обробляти взаємодію між користувачами, товарами і кошиками [3; 6].

4.4 Реалізація API

API інтернет-магазину "Ecobazar" було реалізовано на основі фреймворку Express.js [2], що дозволяє ефективно організувати серверні маршрути та обробку HTTP-запитів. Вся логіка маршрутизації була розділена за сутностями для зручності масштабування, підтримки та логічної структурованості проєкту. На основі REST-підходу реалізовані маршрути для користувачів, товарів, категорій, кошика та рейтингу. Роутери було підключено у головному маршрутизаторі, який об’єднує всі модулі в єдину структуру.

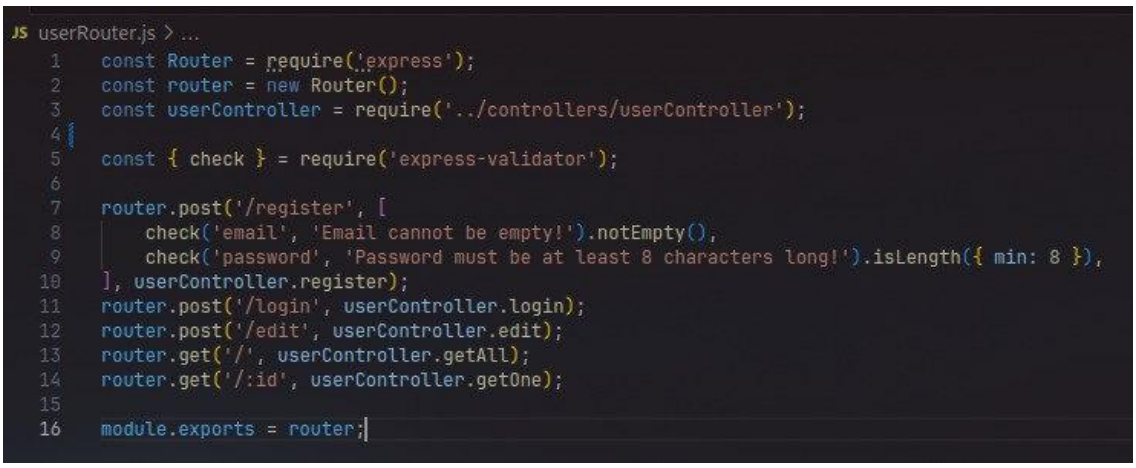
Маршрути було організовано наступним чином:

router.js — головний роутер (див. рис. 4.4.1), який імпортує та використовує всі окремі роутери:

```
JS router.js > ...
1   const Router = require('express');
2   const router = new Router();
3   const userRouter = require('./userRouter');
4   const productRouter = require('./productRouter');
5   const categoryController = require('./categoryRouter');
6   const basketRouter = require('./basketRouter');
7   const ratingRouter = require('./ratingRouter');
8
9   router.use('/user', userRouter);
10  router.use('/product', productRouter);
11  router.use('/category', categoryController);
12  router.use('/basket', basketRouter);
13  router.use('/rating', ratingRouter);
14
15  module.exports = router;
```

Рис. 4.12 – router.js - головний роутер

userRouter.js — відповідає за маршрути, пов’язані з реєстрацією, авторизацією, редагуванням та отриманням користувачів. Важливою частиною реалізації є використання express-validator для валідації вхідних даних, зокрема email та паролю (див. рис. 4.13):



```
JS userRouter.js > ...
1   const Router = require('express');
2   const router = new Router();
3   const userController = require('../controllers/userController');
4
5   const { check } = require('express-validator');
6
7   router.post('/register', [
8       check('email', 'Email cannot be empty!').notEmpty(),
9       check('password', 'Password must be at least 8 characters long!').isLength({ min: 8 }),
10  ], userController.register);
11  router.post('/login', userController.login);
12  router.post('/edit', userController.edit);
13  router.get('/', userController.getAll);
14  router.get('/:id', userController.getOne);
15
16  module.exports = router;
```

Рис. 4.13 – userRouter.js

Серверна логіка для роботи з користувачами зосереджена у контролері UserController(Рисунок 4.14), який відповідає за створення нового користувача, його авторизацію, оновлення даних, отримання інформації про користувачів та їх видалення. Весь функціонал реалізовано з використанням асинхронних функцій, що забезпечує неблокуючу обробку запитів. Обробка помилок виконується через централізований клас ApiError, а валідація даних — за допомогою модуля express-validator.

Реєстрація нового користувача

Метод register відповідає за реєстрацію нового користувача. На першому етапі здійснюється перевірка вхідних даних за допомогою validationResult, яка формує масив повідомлень про помилки, якщо такі є. У разі некоректного запиту сервер повертає клієнту відповідь з кодом 400 та відповідним поясненням.

Після валідації перевіряється, чи існує вже користувач з переданим email — запит виконується до бази даних через метод findOne. Якщо користувач знайдений, сервер відповідає помилкою про дублювання акаунта.

У разі успішного проходження перевірок, пароль хешується за допомогою модуля bcrypt, після чого створюється новий запис у таблиці User, а також автоматично створюється пов’язаний кошик (Basket) на основі зв’язку один-до-одного. В результаті клієнт отримує підтвердження про успішну реєстрацію та інформацію про створеного користувача.

Авторизація користувача

Метод login реалізує процес входу користувача до системи. Спочатку на основі email проводиться пошук користувача в базі. Якщо запис відсутній, повертається відповідь про помилку з кодом 500.

Якщо користувач знайдений, відбувається порівняння збереженого хешованого пароля із введеним за допомогою функції compareSync модуля bcrypt. У разі збігу, користувач отримує відповідь зі статусом 200. Поки що функція генерації токена авторизації (generateJwt) залишена закоментованою, однак у подальшому її можна легко інтегрувати для повноцінної системи авторизації через JWT.

Редагування даних користувача

Метод edit дозволяє оновлювати дані користувача, зокрема: email, пароль та статус активності. На основі переданого ідентифікатора здійснюється пошук запису в базі. У разі його відсутності сервер повертає код 404. Якщо користувач існує, значення полів оновлюються лише в тому випадку, якщо відповідні параметри були передані в запиті. Це реалізовано через перевірку оператором ??, який зберігає попереднє значення у разі відсутності нового.

Після оновлення інформації, виконується метод save, який зберігає зміни в базі даних, а клієнт отримує оновлений об’єкт користувача.

Видалення користувача

Функція delete дозволяє видалити користувача з бази даних. Ідентифікатор передається через параметри запиту. Якщо запис не знайдено або не вдалося виконати операцію видалення, система повертає повідомлення про помилку. У разі успішного видалення надсилається підтвердження з відповідним повідомленням.

Отримання всіх користувачів

Метод getAll надає змогу отримати список усіх користувачів, що зберігаються в базі. Запит реалізовано через findAll — стандартний метод ORM-бібліотеки Sequelize, який повертає масив записів. У відповідь повертаються всі дані у форматі JSON.

Отримання одного користувача за ідентифікатором

Функція getOne виконує пошук конкретного користувача за його ідентифікатором, що передається у параметрах запиту. Якщо користувача не знайдено, сервер повертає відповідь з помилкою типу notFound. У разі успіху клієнт отримує об’єкт користувача у форматі JSON.

Загальна характеристика

Весь функціонал контролера UserController реалізовано з дотриманням принципів розділення відповідальностей (Separation of Concerns), забезпечення обробки помилок та валідації. Це дозволяє гарантувати стабільність і безпечність роботи системи, а також полегшує майбутню модифікацію або розширення бізнес-логіки.

```
const ApiError = require('../error/ApiError');
const { User, Basket } = require('../models');

const bcrypt = require('bcrypt');
const { validationResult } = require('express-validator');


class UserController {
    async register(req, res, next) {
        try {
            const errors = validationResult(req);
            if (!errors.isEmpty()) {
                const errorMessages = errors.errors.map(error => error.msg);
                return next(ApiError.badRequest(errorMessages));
            }

            const { email, password } = req.body;
            if (!email || !password) {
                return next(ApiError.badRequest('Uncorrect email or password!'));
            }

            const candidate = await User.findOne({ where: { email } });
            if (candidate) {
                return next(ApiError.duplicate('User with such email already exists!'));
            }
            const hashPassword = await bcrypt.hash(password, 5);

            const user = await User.create({email, password: hashPassword})
            const basket = await Basket.create({ userId: user.id });
            return res.status(200).json({ message: 'User created!', user: user });
        } catch (e) {
            console.log(e);
            return next(ApiError.badRequest('Registration error'));
        }
    }

    async login(req, res, next) {
        try {
            const { email, password } = req.body;
            const user = await User.findOne({ where: { email } });
            if (!user) {
                return next(ApiError.internal('User not found!'));
            }

            let comparePassword = bcrypt.compareSync(password, user.password);
            if (!comparePassword) {
                return next(ApiError.badRequest('Incorrect password!'));
            }

            // const token = generateJwt(user.id, user.email, user.role);
            // return res.json({ token });
            return res.status(200).json({ message: 'Logged!' });
        } catch (e) {
            console.log(e);
            return next(ApiError.badRequest('Login error!'));
        }
    }

    async edit(req, res) {
        try {
            const { id, email, password, isActive } = req.body;

            const user = await User.findByPk(id);
            if (!user) {
                return res.status(404).json({ error: 'User not found.' });
            }

            user.email = email ?? user.email;
            user.password = password ?? user.password;
            user.isActive = isActive ?? user.isActive;
```

```
class UserController {
    async edit(req, res) {
            user.isActive = isActive ?? user.isActive;

            await user.save();

            return res.status(200).json(user);
        }catch (e) {
            return next(ApiError.internal('Failed to edit user.'));
        }
    }

    async delete(req, res, next) {
        try {
            const { id } = req.params;
            const deleted = await User.destroy({ where: { id } });
            if (!deleted) {
                return next(ApiError.badRequest('User not found'));
            }
            return res.status(200).json({ message: 'User deleted successfully!' });
        } catch (e) {
            return next(ApiError.internal('Failed to delete user.'));
        }
    }

    async getAll(req, res) {
        try {
            const users = await User.findAll();
            res.status(200).json(users);
        } catch (e) {
            console.log(e);
        }
    }

    async getOne(req, res) {
        try {
            const { id } = req.params;
            const user = await User.findByPk(id);
            if (!user) {
                return next(ApiError.notFound('User not found.'));
            }
            return res.status(200).json(user);
        }catch (e) {
            return next(ApiError.internal('Failed to fetch user.'));
        }
    }

}
module.exports = new UserController();
```

Рис. 4.14 – userController.js

productRouter.js — — реалізує маршрути для додавання, перегляду, редагування та видалення товарів. Кожен товар має унікальний ідентифікатор, що дозволяє здійснювати CRUD-операції (див. рис. 4.15):

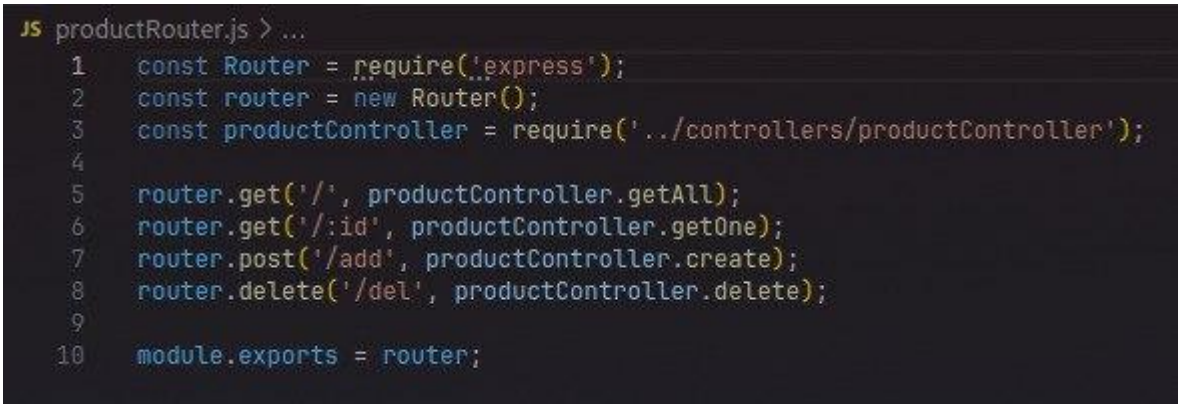


Рис. 4.15 – productRouter.js

Контролер ProductController(Рисунок 4.16) відповідає за створення, видалення, отримання одного або всіх товарів у системі. Він інтегрується з моделями Product, Category, ProductInfo та Rating, що дозволяє виконувати повний спектр CRUD-операцій над об'єктами товару. Логіка реалізована на базі асинхронних функцій, які забезпечують ефективну взаємодію з базою даних за допомогою Sequelize.

Створення нового товару

Метод create реалізує повноцінну логіку додавання нового товару до системи. У тілі запиту очікується передача основних характеристик товару — назви, ціни, акційної ціни, опису, статусу знижки та категорії, до якої належить товар.

Додатково метод підтримує завантаження зображень: якщо у req.files присутній файл, система перевіряє його MIME-тип, дозволяючи лише графічні формати. Після валідації створюється унікальна назва файлу за допомогою UUID та визначається шлях для збереження у локальну директорію static. Файл завантажується на диск методом mv().

Далі виконується створення самого об’єкта Product у базі даних. Якщо до запиту прикріплено додаткову інформацію (info) у вигляді JSON-масиву, вона парситься, перевіряється на валідність і зберігається окремими записами у

таблиці ProductInfo. Кожен об’єкт у цьому масиві повинен містити опис і, за бажанням, список переваг (benefits).

У разі успіху метод повертає статус 201 і створений товар у форматі JSON. Якщо трапляється помилка, вона логуються на сервері, і користувач отримує повідомлення з відповідним статусом.

Видалення товару

Метод delete реалізує видалення товару за його ідентифікатором. ID передається через параметри маршруту, після чого виконується метод destroy. Якщо товар не знайдено, користувач отримає відповідь з повідомленням про помилку. У разі успішного видалення повертається повідомлення з підтвердженням.

Отримання списку товарів

Метод getAll дозволяє отримати перелік усіх товарів з можливістю пагінації та фільтрації за категоріями. Якщо в запиті наявні параметри limit і page, вони використовуються для обчислення offset, який дозволяє виводити дані частинами.

Також метод підтримує фільтрацію за categoryId, якщо такий параметр передано. Запит виконується через метод findAndCountAll, який повертає масив знайдених товарів та їхню загальну кількість. Це особливо корисно при реалізації сторінкової навігації у клієнтському інтерфейсі.

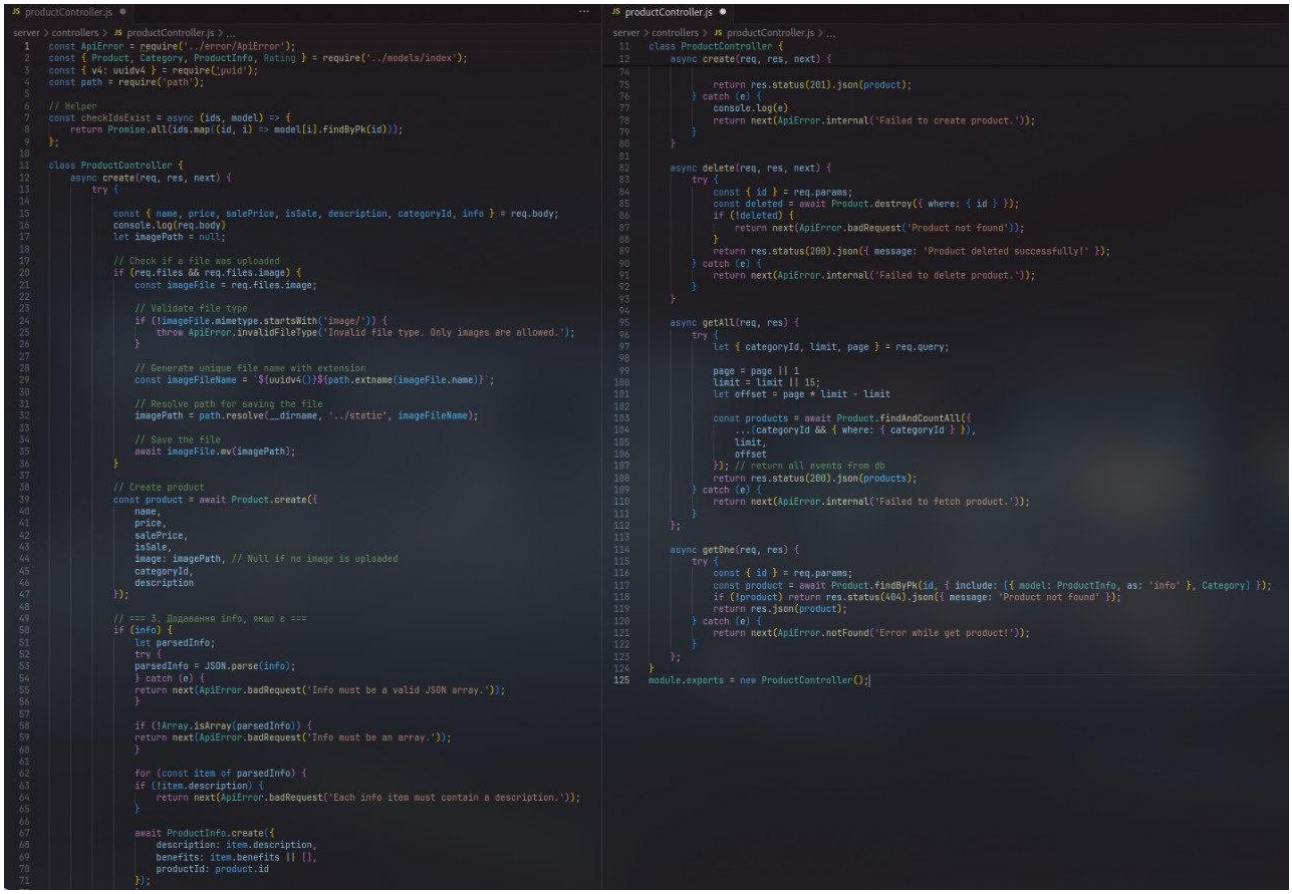
Отримання одного товару

Функція getOne виконує пошук конкретного товару за його ID. Разом із основними даними про товар завантажуються пов’язані записи з таблиці ProductInfo (через alias info) та відповідна категорія. Це досягається завдяки вбудованій підтримці include в Sequelize.

У разі відсутності товару повертається код 404 з відповідним повідомленням. Якщо ж запит виконується коректно, користувач отримує повну інформацію про товар у форматі JSON.

Загальна характеристика

Контролер ProductController побудований відповідно до принципів MVC та REST-архітектури. Він забезпечує повний цикл керування товарними позиціями в інтернет-магазині: від створення із зображенням і додатковими властивостями — до фільтрованого виводу й видалення. Його реалізація відповідає вимогам сучасного серверного програмування: обробка файлів, валідація JSON, обробка винятків, розмежування логіки за сутностями та підтримка масштабування.



```
const ApiError = require('../error/ApiError');
const { Product, Category, ProductInfo, Rating } = require('../models/index');
const { v4: uuidv4 } = require('uuid');
const path = require('path');

// Helper
const checkIdsExist = async (ids, model) => {
    return Promise.all(ids.map((id, i) => model[i].findByPk(id)));
};

class ProductController {
    async create(req, res, next) {
        try {

            const { name, price, salePrice, isSale, description, categoryId, info } = req.body;
            console.log(req.body)
            let imagePath = null;

            // Check if a file was uploaded
            if (req.files && req.files.image) {
                const imageFile = req.files.image;

                // Validate file type
                if (!imageFile.mimetype.startsWith('image/')) {
                    throw ApiError.invalidFileType('Invalid file type. Only images are allowed.');
                }

                // Generate unique file name with extension
                const imageFileName = `${uuidv4()}${path.extname(imageFile.name)}`;

                // Resolve path for saving the file
                imagePath = path.resolve(__dirname, '../static', imageFileName);

                // Save the file
                await imageFile.mv(imagePath);
            }

            // Create product
            const product = await Product.create({
                name,
                price,
                salePrice,
                isSale,
                image: imagePath, // Null if no image is uploaded
                categoryId,
                description
            });

            // === 3. Додавання info, якщо є ===
            if (info) {
                let parsedInfo;
                try {
                parsedInfo = JSON.parse(info);
                } catch (e) {
                return next(ApiError.badRequest('Info must be a valid JSON array.'));
                }

                if (!Array.isArray(parsedInfo)) {
                return next(ApiError.badRequest('Info must be an array.'));
                }

                for (const item of parsedInfo) {
                if (!item.description) {
                    return next(ApiError.badRequest('Each info item must contain a description.'));
                }

                await ProductInfo.create({
                    description: item.description,
                    benefits: item.benefits || [],
                    productId: product.id
                });
```

```
class ProductController {
    async create(req, res, next) {

            return res.status(201).json(product);
        } catch (e) {
            console.log(e)
            return next(ApiError.internal('Failed to create product.'));
        }
    }

    async delete(req, res, next) {
        try {
            const { id } = req.params;
            const deleted = await Product.destroy({ where: { id } });
            if (!deleted) {
                return next(ApiError.badRequest('Product not found'));
            }
            return res.status(200).json({ message: 'Product deleted successfully!' });
        } catch (e) {
            return next(ApiError.internal('Failed to delete product.'));
        }
    }

    async getAll(req, res) {
        try {
            let { categoryId, limit, page } = req.query;

            page = page || 1
            limit = limit || 15;
            let offset = page * limit - limit

            const products = await Product.findAndCountAll({
                ...(categoryId && { where: { categoryId } }),
                limit,
                offset
            }); // return all events from db
            return res.status(200).json(products);
        } catch (e) {
            return next(ApiError.internal('Failed to fetch product.'));
        }
    };

    async getOne(req, res) {
        try {
            const { id } = req.params;
            const product = await Product.findByPk(id, { include: [{ model: ProductInfo, as: 'info' }, Category] });
            if (!product) return res.status(404).json({ message: 'Product not found' });
            return res.json(product);
        } catch (e) {
            return next(ApiError.notFound('Error while get product!'));
        }
    };
}
module.exports = new ProductController();
```

Рис. 4.16 – productController.js

categoryRouter.js — обробляє запити, пов’язані з категоріями товарів. Сюди входять отримання всіх категорій, додавання та видалення (див. рис. 4.17):

Рис. 4.17 – categoryRouter.js

Контролер CategoryController(Рисунок 4.18) реалізує базову CRUD-логіку для керування категоріями товарів у системі інтернет-магазину. Категорії слугують логічним об’єднанням товарів за певними властивостями чи тематиками, що значно спрощує навігацію користувача сайтом та полегшує адміністрування бази даних.

Контролер включає три основні методи: створення нової категорії, видалення категорії за ідентифікатором та отримання повного списку категорій, що зберігаються в базі даних.

Метод створення нової категорії

Метод create приймає вхідні дані з req.body, зокрема назву категорії name. Перевірка наявності цього параметра є обов’язковою, оскільки без нього неможливо створити логічну групу товарів. Якщо поле name відсутнє, сервер повертає помилку 400 з відповідним повідомленням. У разі правильного запиту, метод створює новий запис у таблиці Category за допомогою методу create, після чого повертає щойно створений об’єкт категорії з кодом успіху 201.

Ця функціональність забезпечує гнучке розширення асортименту товарів шляхом створення нових категорій, не змінюючи структуру бази даних або логіку клієнтського інтерфейсу.

Метод видалення категорії

Метод delete очікує отримання ідентифікатора категорії (id) з параметрів запиту. Після отримання id, за допомогою методу destroy виконується спроба видалення категорії з бази. Якщо запис із таким ідентифікатором відсутній, клієнт отримає помилку з кодом 404, яка вказує на те, що вказана категорія не знайдена.

У разі успішного видалення сервер повертає повідомлення про успішну операцію з кодом 200. Цей метод важливий для адміністраторів магазину, які повинні мати змогу оперативно прибирати непотрібні або дубльовані категорії.

Метод отримання всіх категорій

Метод getAll виконує вибірку усіх записів з таблиці Category за допомогою методу findAll. Отриманий результат повертається у вигляді масиву об’єктів JSON. У разі помилки при запиті повертається повідомлення про неуспішну операцію.

Цей метод зазвичай використовується на стороні клієнта для динамічного відображення списку доступних категорій, зокрема у фільтрах, навігаційних меню та формі створення товарів. Він також може бути корисним для аналітики, коли потрібно зрозуміти структуру асортименту магазину.

Загальна оцінка

Контролер CategoryController реалізує мінімальний, але достатній набір функцій для управління категоріями товарів. Його логіка проста та ефективна, а використання структурованих помилок через об’єкт ApiError дозволяє стандартизувати обробку помилок у всій системі. Цей модуль відіграє ключову роль у побудові категоризованого інтерфейсу інтернет-магазину, що позитивно впливає на користувацький досвід і внутрішню організацію товарного каталогу.

```
JS categoryController.js > ...
const ApiError = require('../error/ApiError');
const { Category } = require('../models');

class CategoryController {
    async create(req, res, next) {
        try {
            const { name } = req.body;
            if (!name) return next(ApiError.badRequest('Missing required fields.'));
            const category = await Category.create({ name });
            return res.status(201).json(category);
        } catch (error) {
            return next(ApiError.badRequest('Failed to create product.'));
        }
    }

    async delete(req, res, next) {
        try {
            const { id } = req.params;
            const deleted = await Category.destroy({ where: { id } });
            if (!deleted) {
                return next(ApiError.notFound('Category not found.'));
            }
            return res.status(200).json({ message: 'Product deleted successfully.' });
        } catch (error) {
            return next(ApiError.internal('Failed to delete product.'));
        }
    }

    async getAll(req, res, next) {
        try {
            const categories = await Category.findAll();
            return res.status(200).json(categories);
        } catch (e) {
            return next(ApiError.badRequest('Failed to fetch products.'));
        }
    }
}
module.exports = new CategoryController();
```

Рис. 4.18 – categoryController.js

basketRouter.js — відповідає за додавання, перегляд і видалення товарів у кошику конкретного користувача. Система кошика тісно пов’язана з моделями Product і User (див. рис. 4.19):

```
JS basketRouter.js > ...
const Router = require('express');
const router = new Router();
const basketController = require('../controllers/basketController');

router.get('/:userId', basketController.getBasket);
router.post('/add', basketController.addItem);
router.delete('/remove', basketController.removeItem);

module.exports = router;
```

Рис. 4.19 – basketRouter.js

Контролер BasketController(Рисунок 4.20) відповідає за логіку керування кошиком користувача в інтернет-магазині. Його функціонал включає отримання всіх товарів у кошику, додавання нового товару до кошика, а також видалення певного товару з кошика. Усі методи взаємодіють з відповідними моделями Sequelize: Basket, BasketProduct та Product.

Отримання вмісту кошика користувача

Метод getBasket реалізує пошук поточного кошика для конкретного користувача за його ідентифікатором, що передається в параметрах запиту. За допомогою методу findOne виконується запит до таблиці Basket, де userId відповідає значенню з req.params.

Особливістю цього методу є вкладене включення пов’язаних сутностей: до знайденого кошика додається список усіх товарів (BasketProduct), кожен з яких у свою чергу містить повну інформацію про товар (Product). Такий підхід дозволяє отримати повний і структурований об'єкт із вмістом кошика, що значно спрощує роботу з цими даними на клієнтському боці.

Після виконання запиту результат повертається клієнту у форматі JSON.

Додавання товару до кошика

Метод addItem призначений для додавання нового товару до кошика. У тілі запиту передаються три параметри: basketId — ідентифікатор кошика, productId — ідентифікатор товару, який додається, та quantity — кількість одиниць товару.

На основі цих даних виконується створення нового запису у таблиці BasketProduct за допомогою методу create. Цей запис створює зв’язок між конкретним кошиком і товаром, із зазначенням кількості. У разі успішного додавання клієнт отримує відповідь зі статусом 201 та створеним об’єктом у форматі JSON.

Цей функціонал реалізує ключовий сценарій взаємодії користувача з інтернет-магазином — додавання товарів у кошик для подальшого замовлення.

Видалення товару з кошика

Метод removeItem реалізує логіку видалення конкретного товару з кошика. В тілі запиту очікується отримання двох параметрів: basketId і productId. Якщо хоча б один з них відсутній, сервер повертає відповідь з кодом 400 і повідомленням про помилку.

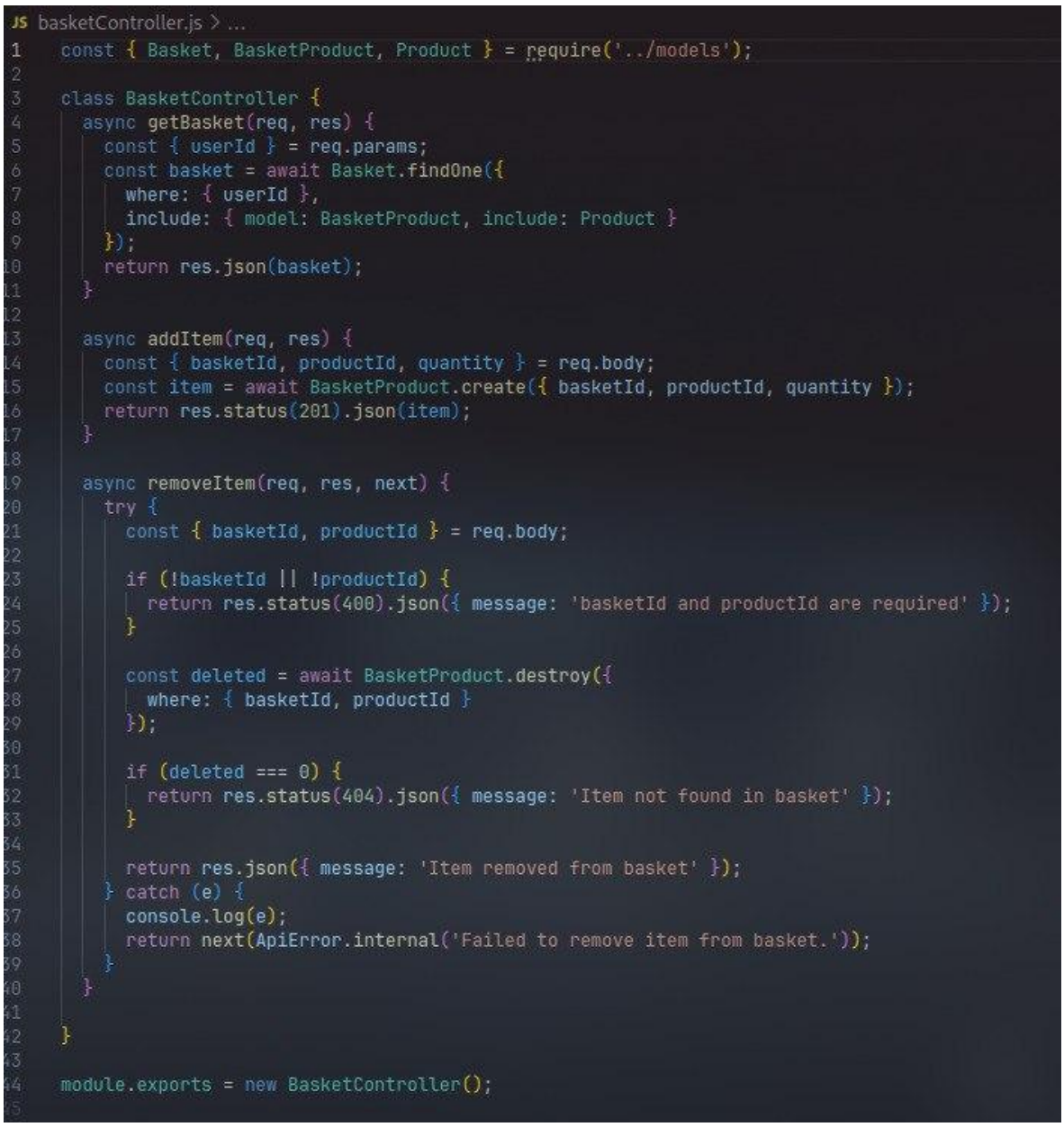
У разі, якщо обидва параметри передано, виконується спроба видалення запису з таблиці BasketProduct через метод destroy, де умова фільтрації — збіг basketId та productId.

Якщо запис не знайдено (deleted === 0), користувач отримає повідомлення про те, що товар у кошику відсутній. У випадку успішного видалення повертається підтвердження у вигляді JSON-об’єкта з відповідним повідомленням.

Для обробки можливих внутрішніх помилок використовується блок try-catch, який забезпечує логування помилок на сервері та відправку відповіді з повідомленням про внутрішню помилку сервера.

Загальна характеристика

Контролер BasketController — це базовий, але критично важливий модуль для функціонування e-commerce системи. Він забезпечує повноцінну логіку додавання, перегляду та видалення товарів із кошика користувача, інтегруючись із базовими моделями даних. Завдяки вкладеному включенню моделей, метод getBasket забезпечує комплексне отримання вмісту кошика з повною інформацією про товари, що зберігає зручність та ефективність на стороні клієнта.



```
basketController.js > ...
const { Basket, BasketProduct, Product } = require('../models');

class BasketController {
  async getBasket(req, res) {
    const { userId } = req.params;
    const basket = await Basket.findOne({
      where: { userId },
      include: { model: BasketProduct, include: Product }
    });
    return res.json(basket);
  }

  async addItem(req, res) {
    const { basketId, productId, quantity } = req.body;
    const item = await BasketProduct.create({ basketId, productId, quantity });
    return res.status(201).json(item);
  }

  async removeItem(req, res, next) {
    try {
      const { basketId, productId } = req.body;

      if (!basketId || !productId) {
        return res.status(400).json({ message: 'basketId and productId are required' });
      }

      const deleted = await BasketProduct.destroy({
        where: { basketId, productId }
      });

      if (deleted === 0) {
        return res.status(404).json({ message: 'Item not found in basket' });
      }

      return res.json({ message: 'Item removed from basket' });
    } catch (e) {
      console.log(e);
      return next(ApiError.internal('Failed to remove item from basket.'));
    }
  }

}

module.exports = new BasketController();
```

Рис 4.20 – basketController.js

Запити до API реалізовано за стандартами REST [5], що забезпечує зручність інтеграції з фронтендом.

Приклади запитів до контроллера productController.js:

1. GET /product — повертає всі товари;
2. GET /product/:id — повертає один конкретний товар;
3. POST /product — додає новий товар;
4. PUT /product/:id — оновлює товар;
5. DELETE /product/:id — видаляє товар.

Кожен роут викликає відповідну функцію з контролера, яка обробляє запит, взаємодіє з базою даних через Sequelize ORM і повертає результат клієнту у форматі JSON. Завдяки модульній структурі проєкту забезпечується зручна підтримка коду та легкість додавання нових функціональностей.

4.5 Тестування та перевірка функціональності

Для тестування працездатності реалізованих API-інтерфейсів було використано **інструмент Postman**, який є одним із найпоширеніших рішень для тестування RESTful-сервісів. Завдяки своїй гнучкості, Postman дозволяє здійснювати **надіслання HTTP-запитів різних типів (GET, POST, PUT, DELETE)** до локального або віддаленого сервера, формувати заголовки, додавати авторизаційні токени, передавати тіла запитів у форматі JSON та переглядати відповіді сервера у зручному форматі.

У процесі тестування були перевірені як **позитивні сценарії** (успішна обробка коректних запитів), так і **негативні сценарії** (валидаційні помилки, відсутність ресурсів, неправильні дані автентифікації тощо).

Це дозволило:

- виявити потенційні недоліки у логіці контролерів;
- перевірити відповідність статус-кодів HTTP стандартам;
- оцінити обробку виключних ситуацій (наприклад, запити з неіснуючим id або відсутніми параметрами);
- переконатися у коректності взаємодії з базою даних PostgreSQL через ORM Sequelize.

Тестування контролера користувача (UserController)

Було перевірено основні маршрути, які відповідають за реєстрацію, логін, редагування профілю, отримання користувачів та їх видалення.

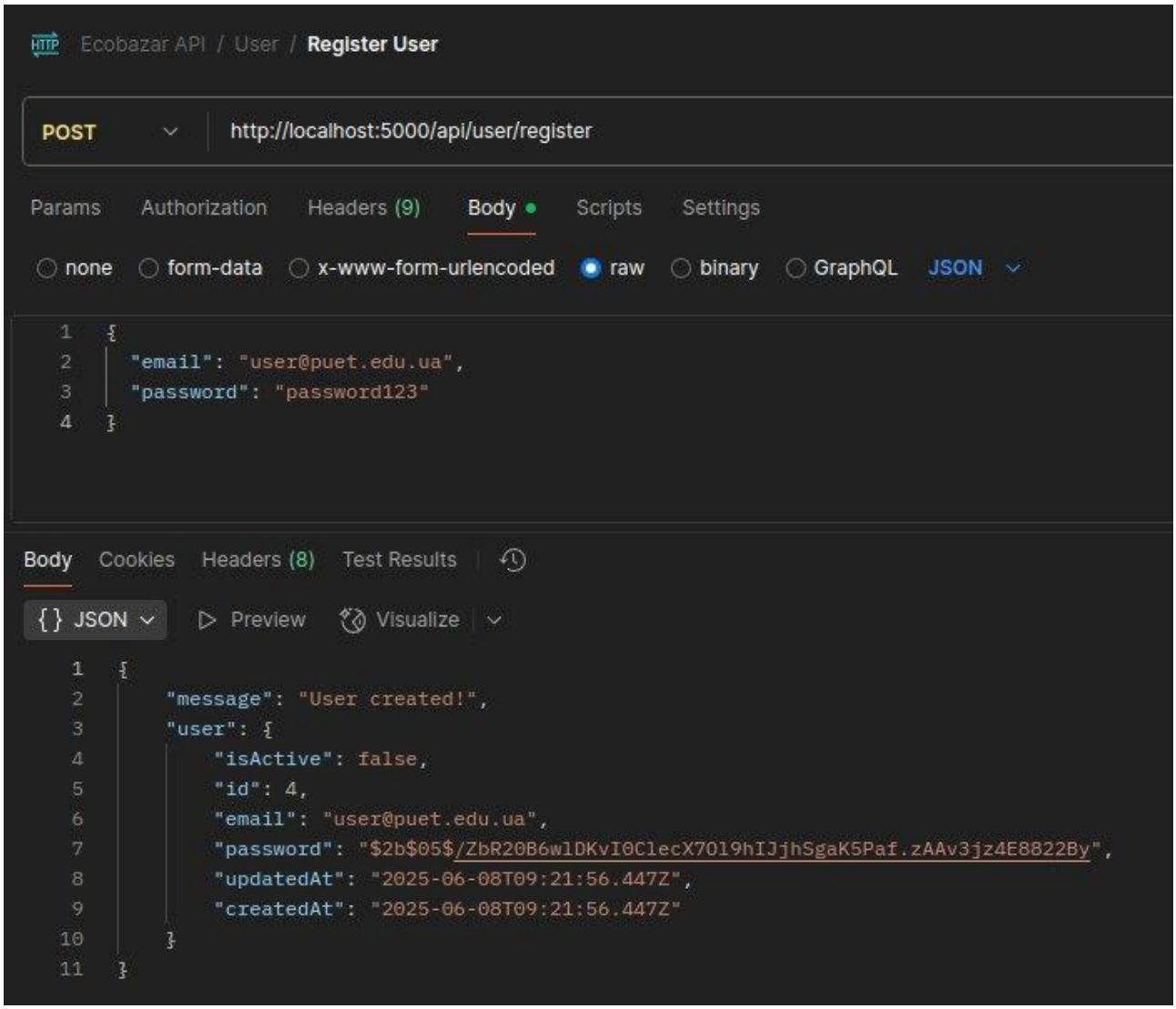


Рис. 4.21 – Postman. Запит на реєстрацію користувача

POST /api/user/register – запит із вказанням email і password для створення нового користувача. У випадку успіху повертається статус 200 та JSON з об’єктом нового користувача.

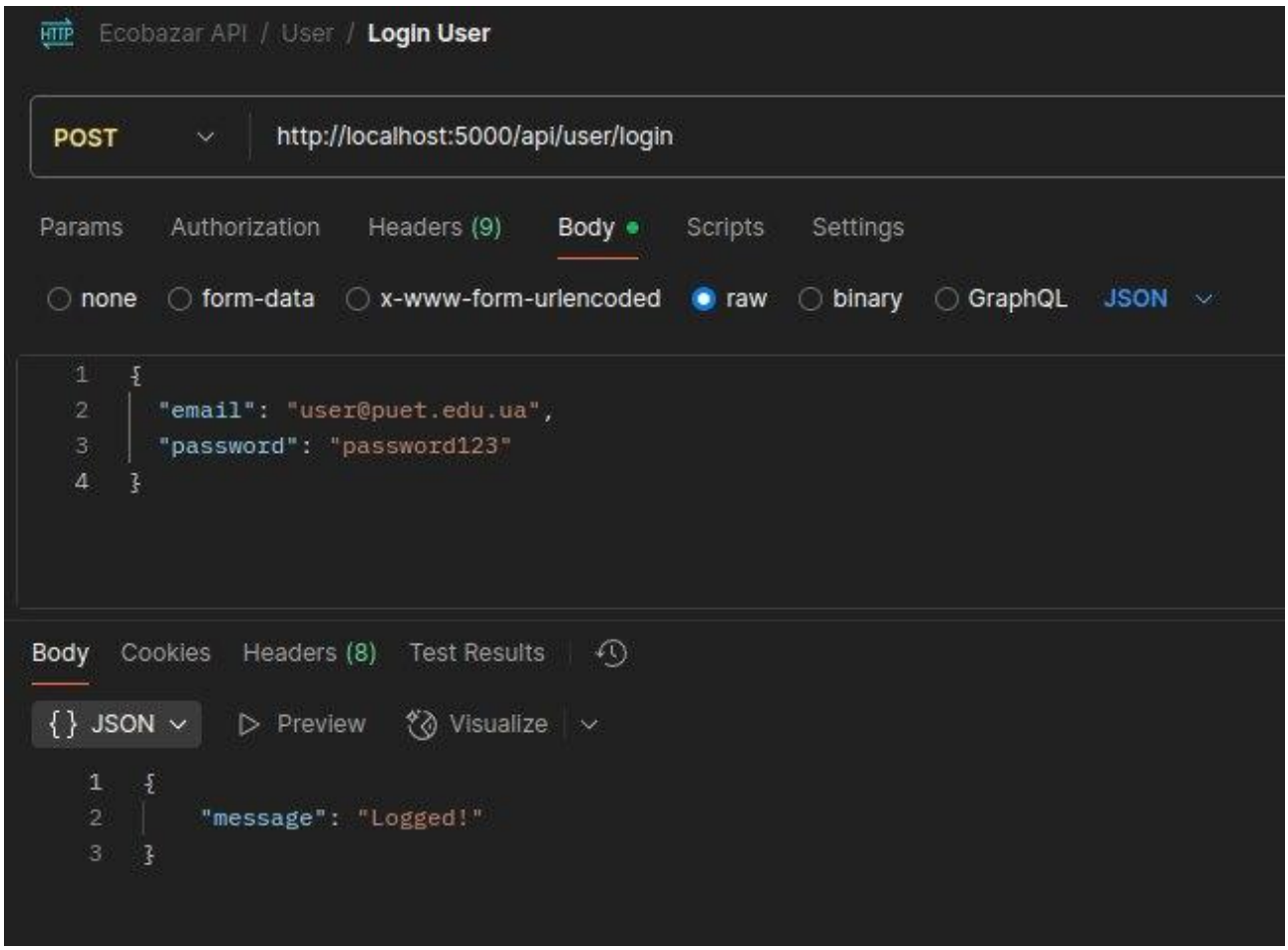


Рис. 4.22 – Postman. Запит на авторизацію користувача

POST /api/user/login – перевірка логіну з існуючими обліковими даними. В разі помилкового пароля або неіснуючого email система повертає відповідні повідомлення з кодами помилок.

Тестування контролера товарів (ProductController)

Перевірено обробку запитів, пов’язаних з додаванням нових товарів, їх видаленням та переглядом списку або конкретного товару.

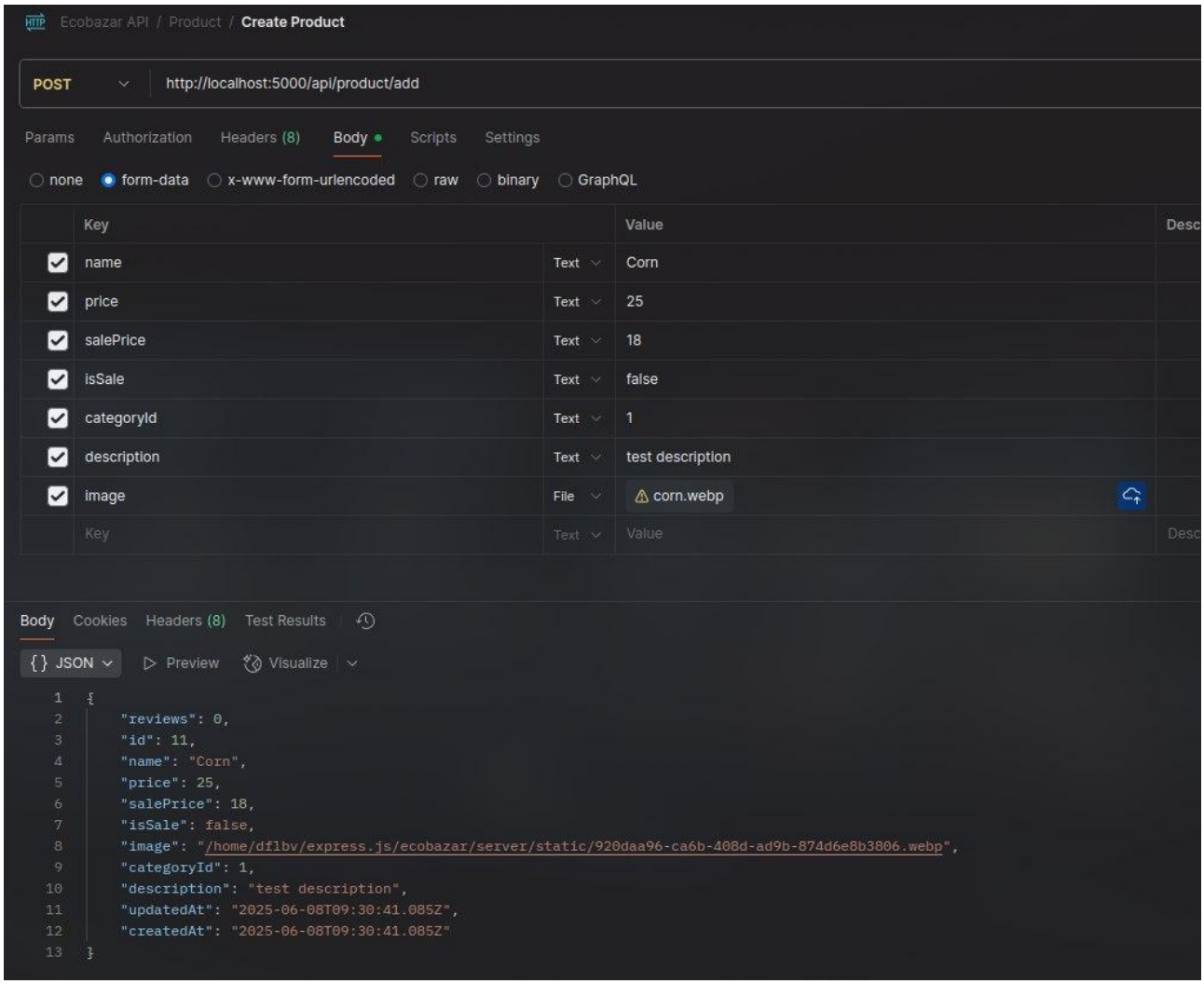


Рис. 4.23 – Postman. Запит для створення товару

POST /api/product – створення товару із зазначенням назви, ціни, категорії, опису та зображенням (multipart/form-data). У разі відсутності зображення поле image лишається порожнім.

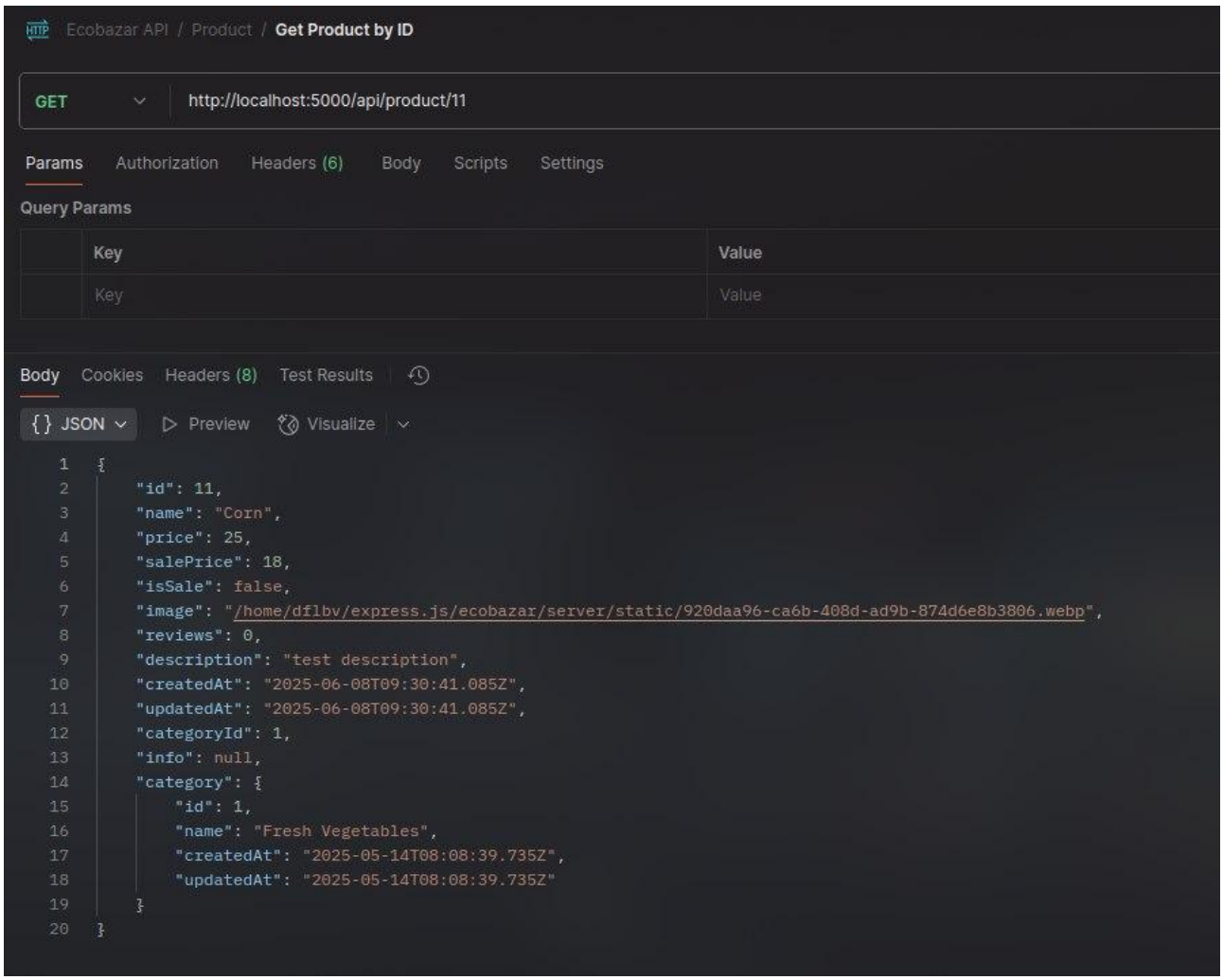


Рис. 4.24 – Postman. Запит для отримання товару по його ID

GET /api/product/:id – перевірка отримання конкретного товару за ID. У відповіді повертається також пов’язана додаткова інформація (ProductInfo) та категорія.

Тестування кошика (BasketController)

Окрема увага була приділена функціоналу кошика, який включає додавання товарів, видалення позицій та перегляд наповнення кошика.

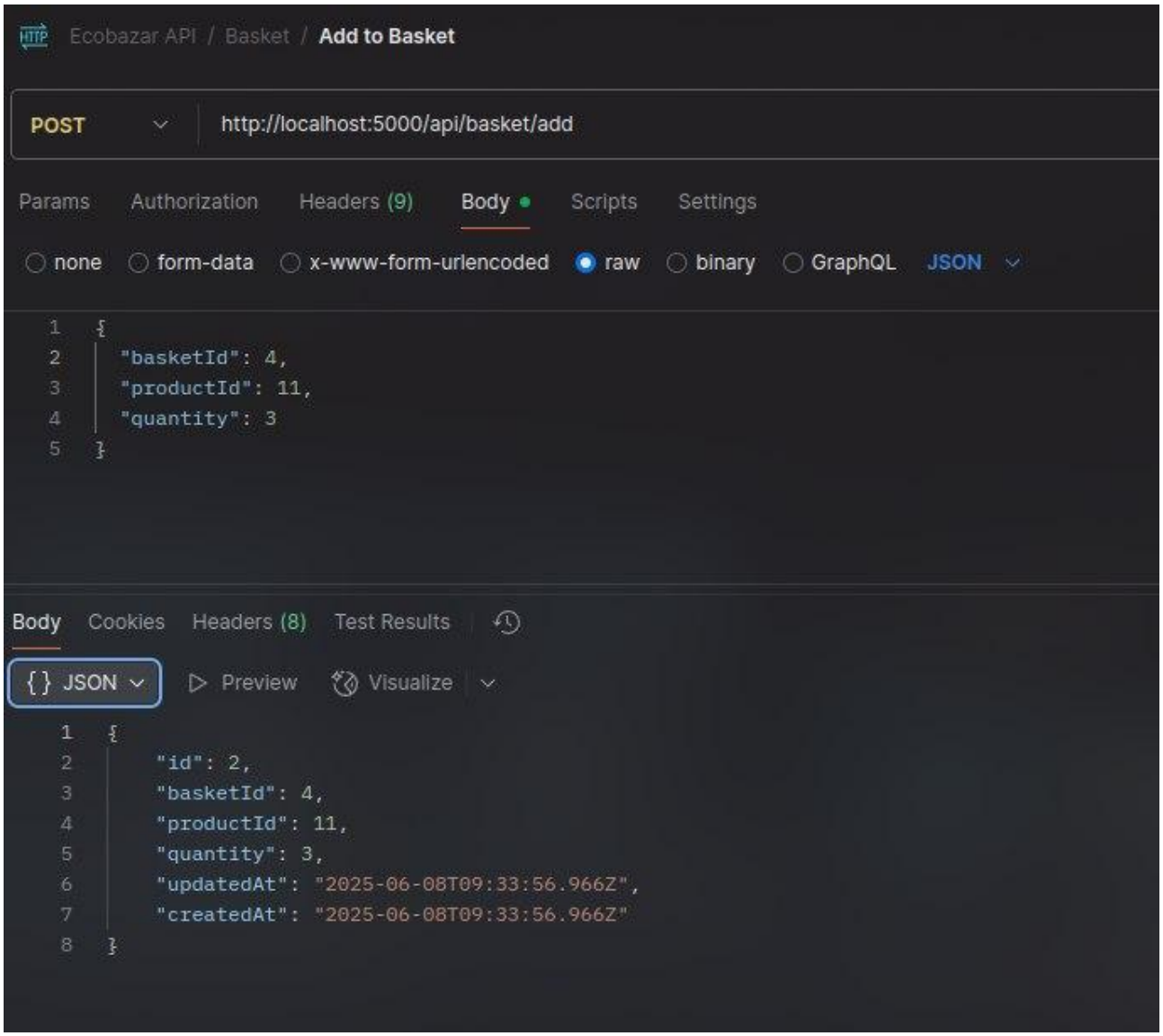


Рис. 4.25 – Postman. Запит для додавання товару у кошик

POST /api/basket/add – додавання товару до кошика користувача з вказанням basketId, productId та quantity. Успішна операція повертає статус 201 та об’єкт доданого товару.

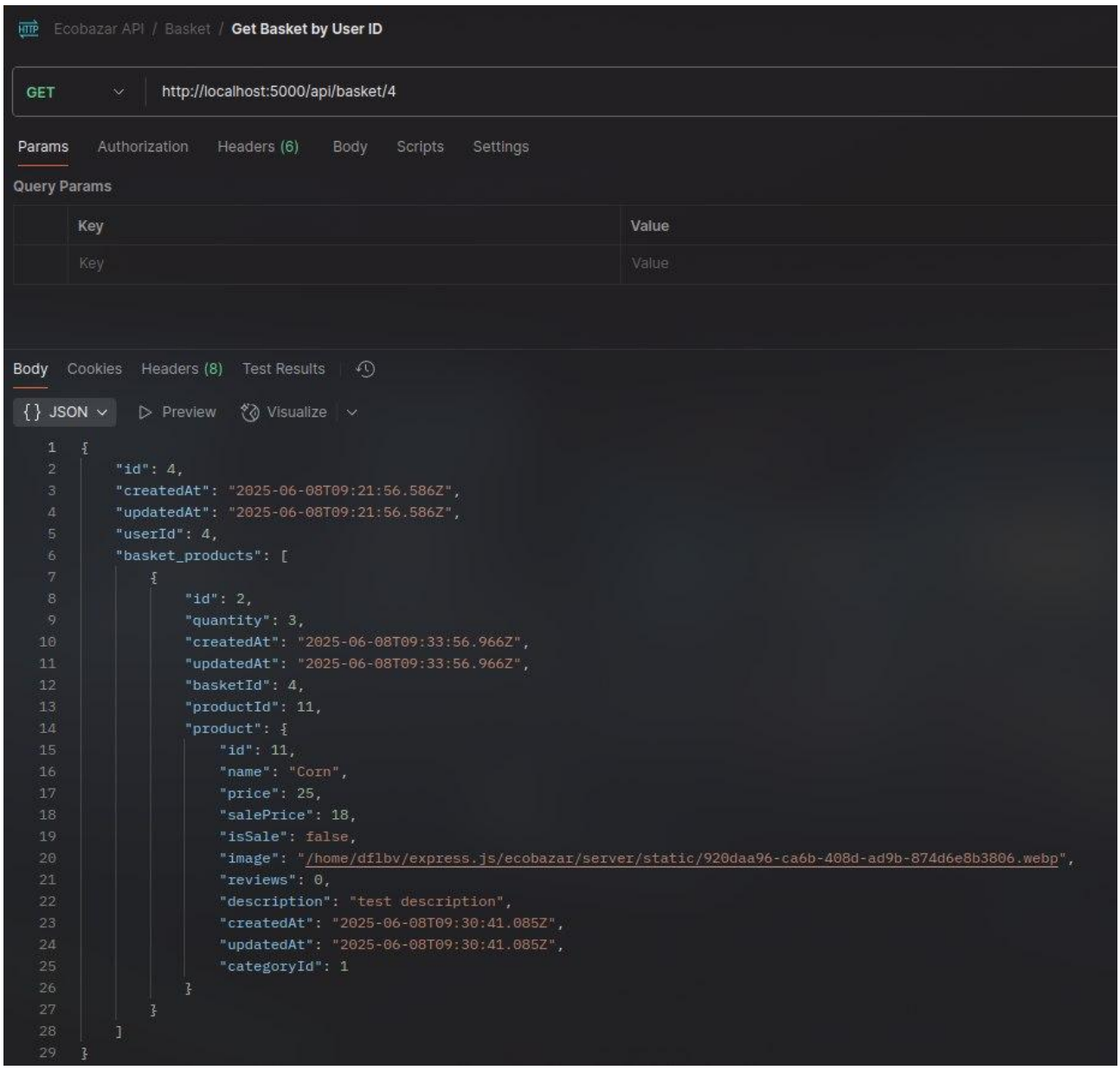


Рис. 4.26 – Postman. Запит для перегляду стану кошика

GET /api/basket/:userId – отримання поточного стану кошика певного користувача, включаючи всі товари та їх кількість.

Тестування категорій (CategoryController)

Функціонал категорій було перевірено шляхом створення, видалення та отримання повного списку.

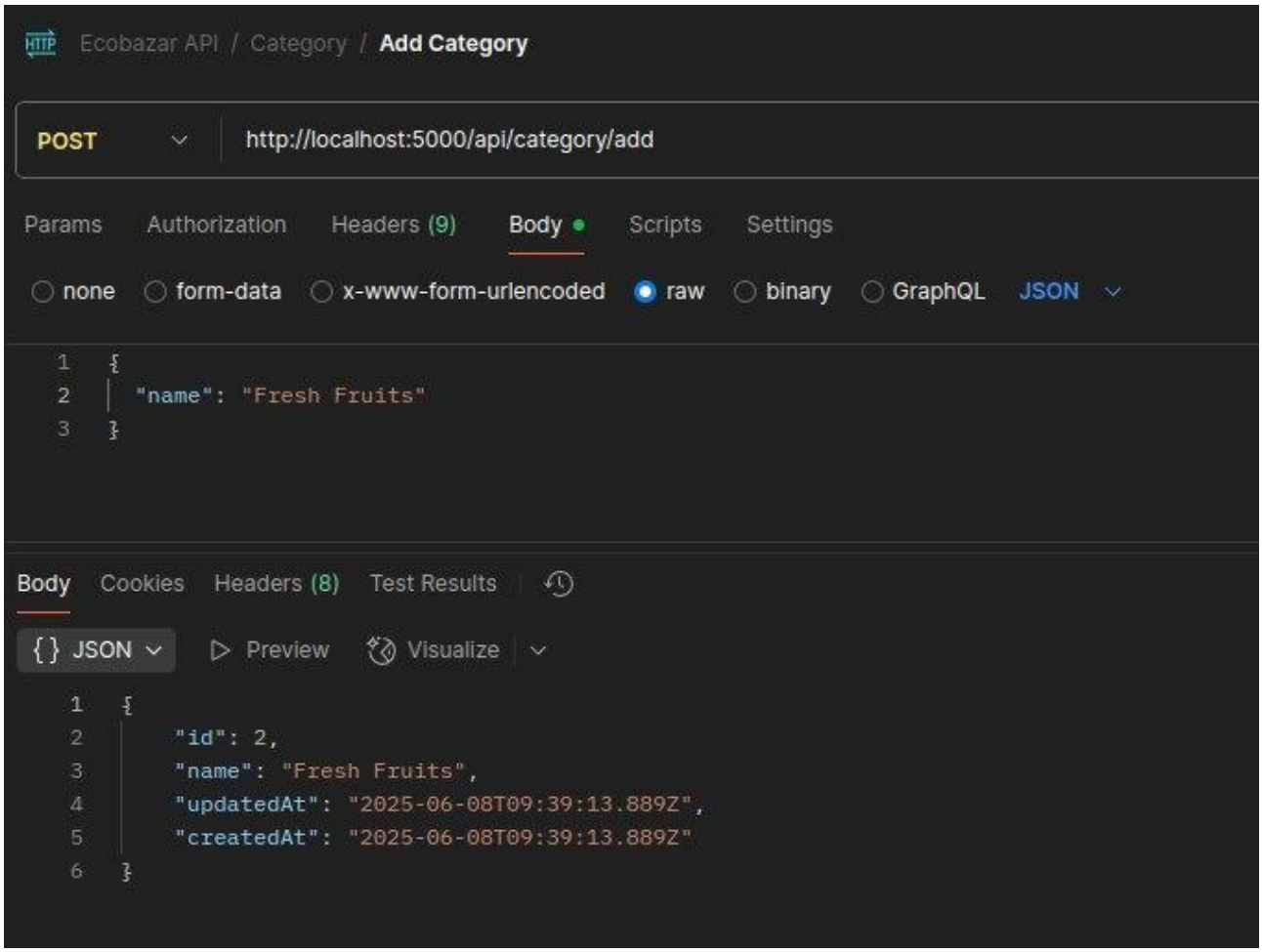


Рис. 4.27 – Postman. Запит для створення нової категорії

POST /api/category/add – створення нової категорії з вказанням лише одного параметра name. Успішна відповідь — статус 201.

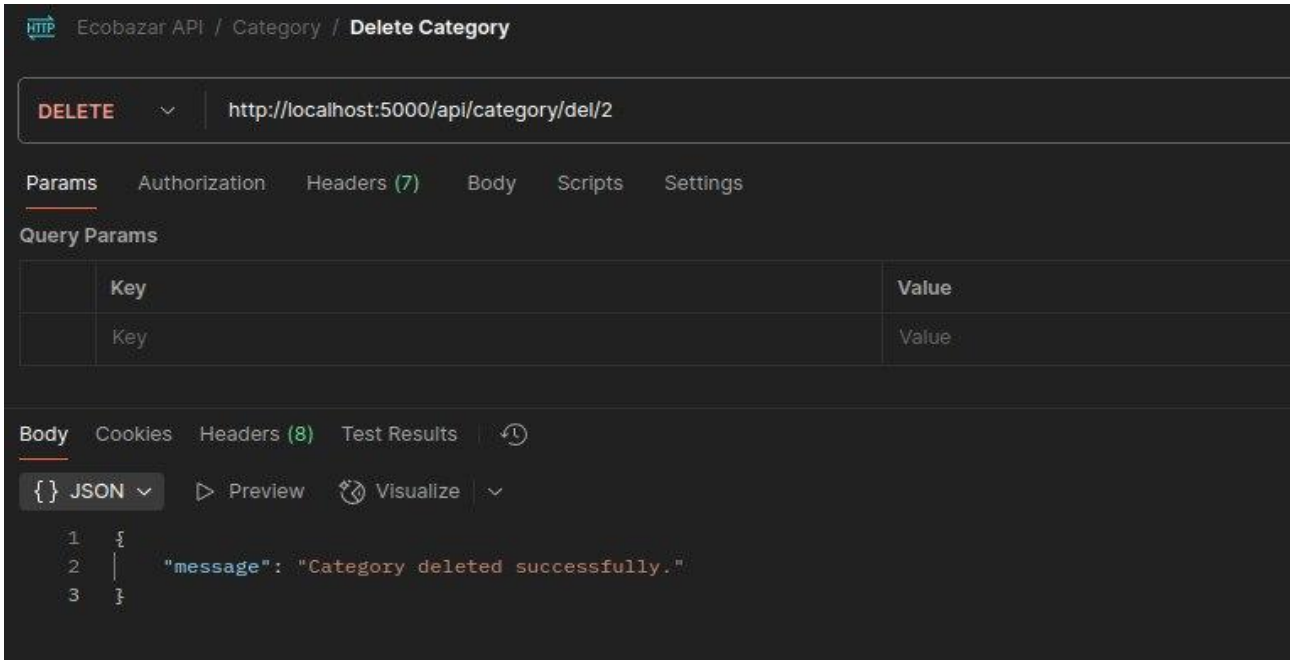


Рис. 4.28 – Postman. Запит для видалення категорії

DELETE /api/category/del/2 – видалення категорії за ID. Перевірялась правильність обробки запиту та відповідне повідомлення у разі відсутності такої категорії.

Крім позитивних сценаріїв, під час тестування було особливу увагу приділено **обробці невалідних запитів**, що є важливою частиною надійності будь-якого API. Усі маршрути мають вбудовану перевірку вхідних даних (валідацію) із використанням модуля express-validator, що дозволяє запобігти обробці некоректної інформації ще до взаємодії з бізнес-логікою.

Наприклад, під час **спроби авторизації користувача, якого не існує у системі**(Рисунок 4.29), сервер коректно повертає повідомлення про помилку із відповідним статус-кодом (HTTP 500 — внутрішня помилка сервера або 400 — помилка клієнта залежно від реалізації логіки обробки). Це демонструє готовність системи до роботи в умовах некоректних або зловмисних запитів і свідчить про дотримання принципів **захищеності та передбачуваної поведінки API**.

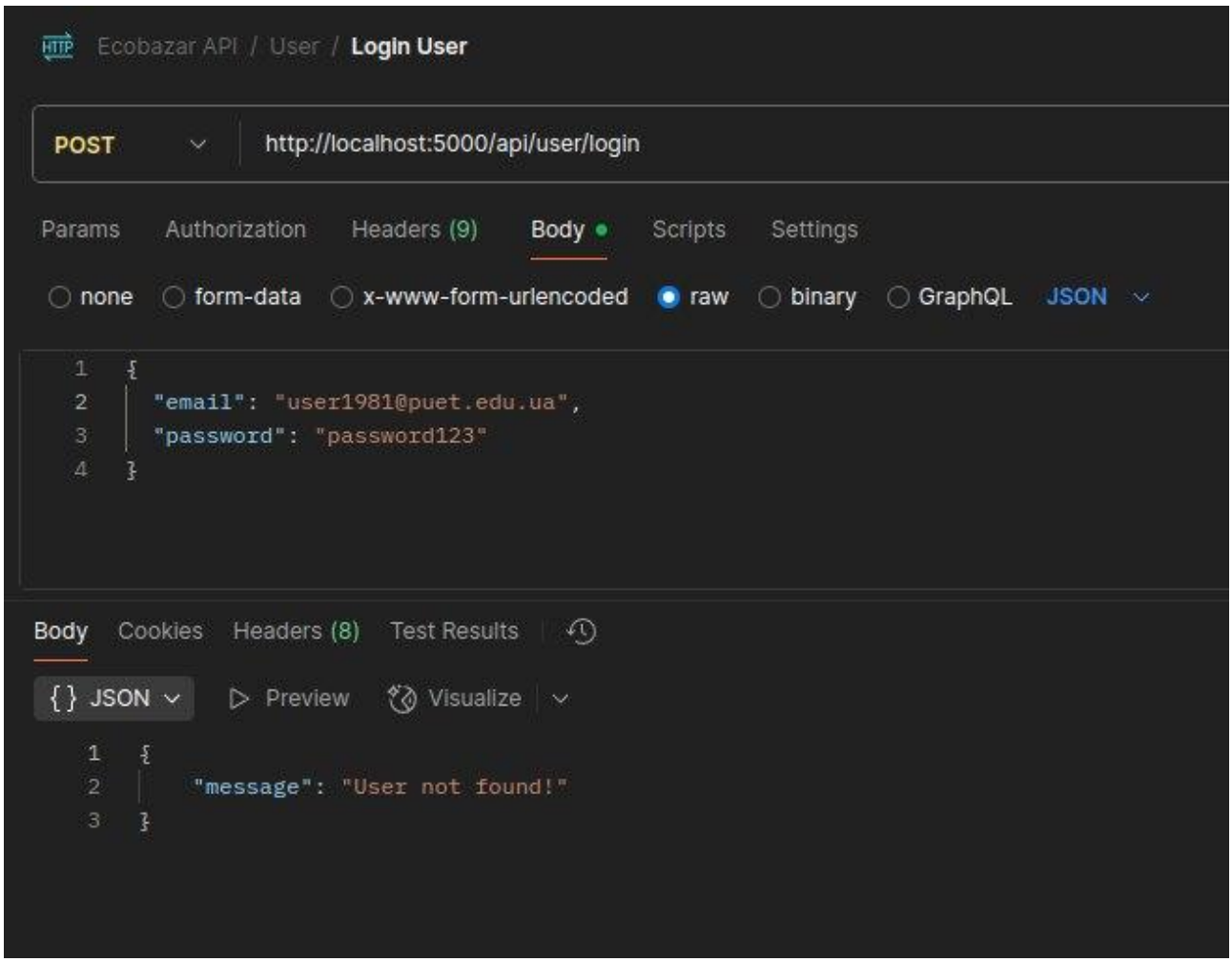


Рис. 4.29 – Postman. Помилка авторизації «User not found!»

Результатом тестування є стабільна, логічно цілісна серверна частина, яка повністю відповідає заданим вимогам. Всі основні маршрути, модулі та запити пройшли функціональні перевірки та демонструють правильну взаємодію з базою даних PostgreSQL, підтримку REST-запитів та обробку типових сценаріїв для інтернет-магазину.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було створено повноцінну серверну частину для інтернет-магазину "Ecobazar", яка забезпечує всі необхідні функціональні можливості для взаємодії з клієнтською частиною та базою даних. Розробка бекенду є критично важливим етапом у створенні сучасного вебсервісу, що гарантує стабільність, безпеку та масштабованість проєкту.

Під час реалізації було виконано такі основні завдання:

Аналіз технічних вимог – проведено вивчення типових рішень для реалізації інтернет-магазинів на Node.js та визначено доцільність використання Sequelize як ORM-бібліотеки.

Проєктування структури бази даних – створено логічну модель даних з урахуванням сутностей: користувач, товар, категорія, кошик, рейтинг, додаткова інформація, що дало змогу ефективно зберігати та обробляти інформацію.

Створення моделей Sequelize та зв’язків між ними – реалізовано повні реляційні зв’язки між таблицями відповідно до бізнес-логіки.

Розробка REST API – побудовано маршрути для авторизації, роботи з товарами, категоріями та кошиком. Забезпечено захист маршрутів та використання JWT для авторизації.

Тестування серверної частини – проведено перевірку роботи основних функцій за допомогою Postman, що дозволило виявити та усунути помилки на етапі розробки.

Для реалізації проєкту були використані такі технології: Node.js, Express.js, PostgreSQL, Sequelize, bcrypt, JWT, а також середовище розробки Visual Studio Code. Налаштування середовища та запуск локального серверу дозволили ефективно тестувати всі функції без потреби в окремому хостингу.

Отже, бекенд інтернет-магазину "Ecobazar" розроблено успішно, і він повністю відповідає вимогам до сучасних систем електронної комерції. Структура проєкту дозволяє легко масштабувати функціональність у

майбутньому та інтегрувати клієнтську частину без потреби переробки архітектури.

СПИСОК ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Дакетт Дж. HTML та CSS: Проектування та створення веб-сайтів. — Wiley, 2011. — 490 с.
2. Фланаган Д. JavaScript: Вичерпний посібник. — O'Reilly Media, 2021. — 746 с.
3. Уллман Л. PHP та MySQL для динамічних веб-сайтів. — Peachpit Press, 2018. — 704 с.
4. Мейєр Е., Вейл Е. CSS: Вичерпний посібник. — O'Reilly Media, 2021. — 1088 с.
5. Роббінс Дж. Вивчаємо веб-дизайн: Посібник для початківців з HTML, CSS та веб-графіки. — O'Reilly Media, 2018. — 810 с.
6. Ніксон Р. Вивчаємо PHP, MySQL та JavaScript. — O'Reilly Media, 2018. — 806 с.
7. Круг С. Не змушуй мене думати, переглянуте: Підхід до юзабіліті на основі здорового глузду. — New Riders, 2014. — 216 с.
8. Браун І. Веб-розробка за допомогою HTML, CSS та JavaScript. — O'Reilly Media, 2020. — 408 с.
9. Олсопп Дж. Окрема книга: Адаптивний веб-дизайн. — A Book Apart, 2011. — 183 с.
10. Беваква Н. Розробка додатків на JavaScript: A Build-First Approach. — O'Reilly Media, 2015. — 182 с.
11. Фрост Б. Атомарний дизайн. — BradFrost, 2016. — 256 с.
12. Хоу Ш. Вчимося програмувати HTML та CSS: Розробка та стилізація веб-сайтів. — Pearson, 2014. — 256 с.
13. Роббінс Дж. Веб-дизайн у двох словах. — O'Reilly Media, 2006. — 812 с.
14. Дакетт Дж. JavaScript та JQuery: Інтерактивна інтерфейсна веброзробка. — Wiley, 2014. — 640 с.

15. Гарретт Дж. Елементи користувацького досвіду: Орієнтований на користувача дизайн для Інтернету і не тільки. — New Riders, 2010. — 192 с.
16. МакФарланд Д. PHP та MySQL: Відсутній посібник. — O'Reilly Media, 2012. — 526 с.
17. Фріман А. Pro HTML5 з CSS, JavaScript та мультимедіа. — Apress, 2017. — 849 с.
18. Вейл Е. CSS: Відсутній посібник. — O'Reilly Media, 2020. — 898 с.
19. Ольховська О. В., Черненко О. О. Методичні рекомендації щодо оформлення пояснювальних записок до дипломної роботи. — ВНЗ Укоопспілки «Полтавський університет економіки і торгівлі», 2023. — 67 с.