МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
МАРІУПОЛЬСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ЕКОНОМІКО-ПРАВОВИЙ ФАКУЛЬТЕТ
КАФЕДРА СИСТЕМНОГО АНАЛІЗУ
ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

До захисту допустити:
В.о. зав. кафедри

Ганна МАРТИНЮК

«04» червня 2025 р.

«РОЗРОБКА ВЕБ-ДОДАТКУ ІНТЕРНЕТ-МАГАЗИНУ ЗООТОВАРІВ»

Кваліфікаційна робота
здобувача вищої освіти першого (бакалаврського) рівня вищої освіти освітньо-професійної програми «Комп'ютерні науки»
Конькової Аліни Ігорівни
Науковий керівник:
Стахова Анжеліка Петрівна,
кандидат технічних наук, доцент,
доцент кафедри системного аналізу та інформаційних технологій
Рецензент:
Нечипорук Олена Петрівна,
доктор технічних наук, професор, завідувач кафедри інтелектуальних кібернетичних систем Державного університету «Київського авіаційного університету»

Кваліфікаційна робота захищена
з оцінкою добре 75 (С)
Секретар ЕК

«11» червня 2025 р.

Київ – 2025

ЗМІСТ

ВСТУП

У сучасних умовах динамічного розвитку цифрових технологій особливого значення набуває створення веб-застосунків, які забезпечують ефективного взаємодію між бізнесом та кінцевим споживачем. Однією з найбільш популярних форм електронної комерції є інтернет-магазини, які дають можливість користувачам здійснювати покупки онлайн, економлячи час та ресурси. Ринок зоотоварів також активно розвивається, і все більше компаній прагнуть реалізовувати свою продукцію за допомогою інтернет-технологій. У зв'язку з цим постає потреба у створенні сучасного, функціонального та зручного веб-додатку, який би дозволяв здійснювати повноцінну торгівлю товарами для тварин в онлайн-середовищі.

Мета дослідження: створення функціонального та зручного веб-додатку для інтернет-магазину зоотоварів, що дозволяє користувачам переглядати, обирати, замовляти товари, а адміністраторам – керувати асортиментом.

Об'єкт дослідження: процес розробки веб-застосунків у сфері електронної комерції. У межах дослідження розглядається повний життєвий цикл розробки програмного забезпечення – від постановки задачі та вибору технологій до тестування та впровадження.

Предмет дослідження: функціональні, технічні та візуальні рішення, які застосовуються під час створення веб-додатку саме для інтернет-магазину зоотоварів. Особлива увага приділяється вибору архітектури, технологічного стеку, побудові інтерфейсу користувача, організації бази даних, реалізації адміністративного функціоналу, а також забезпеченню зручності, безпеки та масштабованості розробленого застосунку.

РОЗДІЛ 1. ХАРАКТЕРИСТИКА ТЕМИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1.1. Загальна характеристика теми

У сучасних умовах цифрової трансформації суспільства та бізнесу особливої актуальності набувають інформаційні системи, які забезпечують ефективну взаємодію між підприємствами та кінцевим споживачами.

Одним з ключових інструментів реалізації таких систем є веб-додатки, які широко застосовуються у сфері електронної комерції. Вони дозволяють автоматизувати бізнес-процеси, розширити канали продажів, забезпечити зручність та доступність сервісів для користувачів.

Сфера торгівлі зоотоварами є прикладом динамічно зростаючої ніші ринку, яка вимагає сучасних цифрових рішень. Онлайн-магазини для продажу товарів для домашніх тварин дають змогу власникам тварин зручно замовляти корми, аксесуари, медичні засоби та інші товари, не виходячи з дому. Водночас для компаній така система забезпечує швидкий доступ до цільової аудиторії, ефективну логістику та аналітику.

Кваліфікаційна робота присвячена розробці веб-додатку для інтернет-магазину зоотоварів. Такий додаток повинен реалізовувати ключові функції: реєстрація та авторизація користувачів, перегляд асортименту, фільтрація товарів, додавання до кошика, оформлення замовлення, перегляд історії покупок, адміністрування товарів та користувачів.

Для реалізації розробки обрано сучасний технологічний стек:

1) React – бібліотека для побудови динамічного інтерфейсу користувача;

2) Node.js з Express.js – серверна платформа для створення REST API;

3) PostgreSQL – реляційна база даних для зберігання інформації про товари, користувачів і замовлення.

Обрана тема дозволяє комплексно продемонструвати практичні навички повного циклу веб-розробки – від аналізу вимог до створення, тестування та застосування готового продукту.

1.2. Обґрунтування вибору теми

Актуальність теми зумовлена кількома чинниками. По-перше, стрімкий розвиток електронної комерції потребує нових, адаптивних рішень, які відповідають сучасним вимогам безпеки, функціональності та зручності. По-друге, інтернет-магазини поступово витісняють традиційні офлайн-магазини завдяки простоті користування та доступності 24/7. По-третє, ринок товарів для тварин демонструє стабільне зростання, а отже, потребує ефективних інструментів для онлайн-продажу.

Обрана тема дозволяє:

1) застосувати здобуті теоретичні знання в реальному практичному проекті;

2) вивчити сучасні технології веб-програмування;

3) розвинути навички системного аналізу, архітектурного проектування, UI/UX-дизайну;

4) створити продукт, який може бути впроваджений як стартап або частина великого IT-рішення.

Оскільки створення інтернет-магазинів – це реальна задача з чітко визначеними критеріями якості, ефективності й безпеки, її реалізація в рамках кваліфікаційної роботи є цілком доцільною та обґрунтованою.

1.3. Об’єкт та предмет дослідження

Об’єктом дослідження виступає процес розробки веб-застосунків для електронної комерції, зокрема інтернет-магазинів, які включають клієнтську, серверну та адміністративну частини.

Предмет дослідження – це конкретні технічні, функціональні та візуальні рішення, що реалізуються при створенні веб-додатку інтернет-магазину зоотоварів. До них належать:

1) архітектура додатку (фронтенд-бекенд база даних);
2) структура бази даних, побудова зв’язків між сутностями;
3) REST API для взаємодії між клієнтом і сервером;
4) механізм авторизації користувачів;
5) UI/UX рішень для зручної навігації;
6) безпека обробка персональних даних;
7) обробка помилок, перевірка сесій, логіка замовлень.

1.4. Аналітичні методи та технології

Під час системного аналізу треба використовувати такі аналітичні методи та технології для досягнення мети дослідження:

1) SWOT-аналіз дозволяє виявити сильні сторони (використання сучасних технологій, популярна ніша), слабкі сторони (високий рівень конкуренції), можливості (потенціал інтеграції з платіжними сервісами, розширення на мобільні платформи) та загрози (безпекові ризики, складність підтримки).

2) Benchmarking — аналіз функціональності, дизайну та структури таких платформ, як ZooPlus, PetShop, Amazon Pets. Особливу увагу приділяє системі категорій, механізму фільтрації, карткам товарів, оформленню замовлення та взаємодії з кошиком.

3) Use Case діаграми — опис основних сценаріїв для трьох ролей: гість, зареєстрований користувач і адміністратор. Це дає змогу сформувати список базових вимог до системи [1].

4) ER-моделювання — допомагає створити логічну модель бази даних із сутностями: користувач, товар, категорія, замовлення, платіж, кошик. Кожна сутність описується атрибутами, типами зв’язків (один-до-багатьох, багато-до-багатьох тощо) [2].

5) MoSCoW-метод — дозволяє визначити критичний функціонал Must Have: реєстрація, кошик, оформлення замовлення [3].

Ці підходи дозволяють ефективно планувати розробку, виявляти помилки на ранніх етапах і приймати обґрунтовані рішення щодо архітектурних змін.

1.5. Програмне забезпечення

У ході розробки використовується такі інструменти та технології:

1) Visual Studio Code — багатофункціональне середовище розробки з великою кількістю розширень для React, Node.js, ESLint тощо.

2) React.js — JavaScript-бібліотека для побудови компонентного, реактивного інтерфейсу користувача. Застосовується компонентний підхід, використовуються хуки (useState, useEffect), маршрутизація (react-router-dom).

3) Node.js + Express.js — використовується для розробки серверної частини: маршрутизації, обробки запитів, авторизації, захисту API.

4) PostgreSQL — потужна реляційна СКБД, що підтримує транзакції, зв’язки між таблицями, індекси, типи даних [4].

5) Postman — для тестування API (GET/POST/PUT/DELETE-запити), перевірки автентифікації, авторизації, обробки помилок [5].

6) draw.io / diagrams.net — для побудови UML-діаграм (Use Case, Activity, Sequence) і логічної ER-моделі бази даних.

7) Figma — для створення інтерактивних прототипів інтерфейсу користувача, перевірки зручності навігації, адаптивності.

1.6. Оптимізація процесів і прийняття рішень

Щоб забезпечити ефективність розробки, високу якість продукту і оперативне реагування на зміни, треба застосовувати наступні методології:

1) Agile-методологія

- Робота організована за принципами Scrum: короткі ітерації (спринти) по 1-2 тижні.
- Кожен спринт закінчується переглядом досягнутих результатів.
- Принцип MVP (мінімально життєздатний продукт) дозволяє швидко тестувати основний функціонал на реальних користувачах.

2) Code review та модульне тестування

- Кожна зміна проходить перевірку іншими розробниками, що дозволяє підтримувати єдиний стиль коду та вчасно виявляти баги.
- Модульні тести для серверної логіки допомагають перевірити роботу критичних API-запитів.

3) Безперервна інтеграція (CI)

- Кожен коміт у GitHub автоматично запускає процеси збірки, перевірки форматування коду та запуску тестів.
- Наявність CI дозволяє швидко виявляти проблеми на етапі розробки та підтримувати стабільність основної гілки проекту.

4) Прототипування

- Прототипи сторінок будуть змодельовані у Figma ще до початку верстки.
- Це дозволяє:
  - Виявити потенційні помилки в логіці навігації;
  - Спростити адаптивність інтерфейсу для мобільних та десктопних пристроїв.

Оптимізація процесів розробки забезпечує високу якість коду, гнучкість у впровадженні змін і мінімізацію часу на виправлення помилок.

Висновки до розділу 1

У першому розділі було розглянуто загальні підходи, обґрунтування та аналітичні засади розробки веб-додатку для інтернет-магазину зоотоварів.

Проведений огляд теми дозволив визначити актуальність розробки веб-додатків у сфері електронної комерції, зокрема для динамічно зростаючого ринку товарів для домашніх тварин. Було обґрунтовано вибір теми як практично значущої й перспективної для впровадження сучасних IT-рішень.

Встановлено об’єктом дослідження процес розробки веб-застосунків, а предметом – технічні, функціональні та візуальні рішення, необхідні для створення повноцінного інтернет-магазину. Завдяки системному аналізу визначено вимоги до архітектури системи, структури бази даних, механізмів авторизації, взаємодії клієнтської та серверної частин.

Розглянуто основні аналітичні методи, такі як SWOT-аналіз, Benchmaking, Use Case-діаграми, ER-моделювання та MoSCoW-метод, що дозволяє побудувати чітку модель майбутнього додатку й мінімізувати ризики.

Вибір програмного забезпечення та інструментів, таких як React.js, Node.js, PostgreSQL, Visual Studio Code, Postman, Figma, обґрунтовано їх відповідністю вимогам проекту.

Оптимізація процесів розробки за допомогою застосування Agile-методології, впровадження практик code review, модульного тестування, безперервної інтеграції (CI) та прототипування дозволяє підвищити якість продукту, забезпечити гнучкість розробки та оперативне реагування на зміні.

Таким чином, результати першого розділу сформували надійну теоретичну основу для реалізації наступних етапів кваліфікаційної роботи.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ІНТЕРНЕТ-МАГАЗИНУ

2.1. Цілі та потреби інформаційної системи інтернет-магазину

Основні завдання і вимоги до інформаційної системи інтернет-магазину охоплюють низку аспектів, які заслуговують на детальний розгляд.

Продаж товарів: головною функцією системи є організація онлайн-продажу продукції. Вона має гарантувати оперативність і безпеку оформлення замовлень, проведення оплати та доставки товарів.

Управління асортиментом: інформаційна система повинна забезпечувати зручне та ефективне адміністрування асортименту, контроль за наявністю товарів і встановлення конкурентних цін.

Організація замовлень: система має дозволяти безперешкодне керування процесом замовлень – від їх створення до фінальної доставки клієнту.

Аналітика та звітність: необхідною є можливість збору та аналізу даних щодо продажів, замовлень і поведінки користувачів, що допоможе вдосконалити бізнес-процеси і сформувати стратегію розвитку магазину.

Комунікація з клієнтами: інформаційна система повинна створювати можливості для ефективної взаємодії із покупцями через електронну пошту, телефон, онлайн-чат та інші засоби зв’язку.

Безпека даних: дуже важливо, щоб система гарантувала захист персональних даних користувачів, операцій з платежами та іншої конфіденційної інформації.

2.2. Складання Use Case діаграм

Метою цього підрозділу є визначити функціональність системи та спрогнозувати можливі сценарії взаємодій між акторами. Цей етап допомагає підвищити якість та ефективність інтернет-магазину, забезпечуючи відповідність потребам користувачів [6]. Інтернет-магазин

включає двох дійових осіб, які взаємодіють з нею. На рисунку 2.1 представлено діаграму варіантів використання для актора «Клієнт».

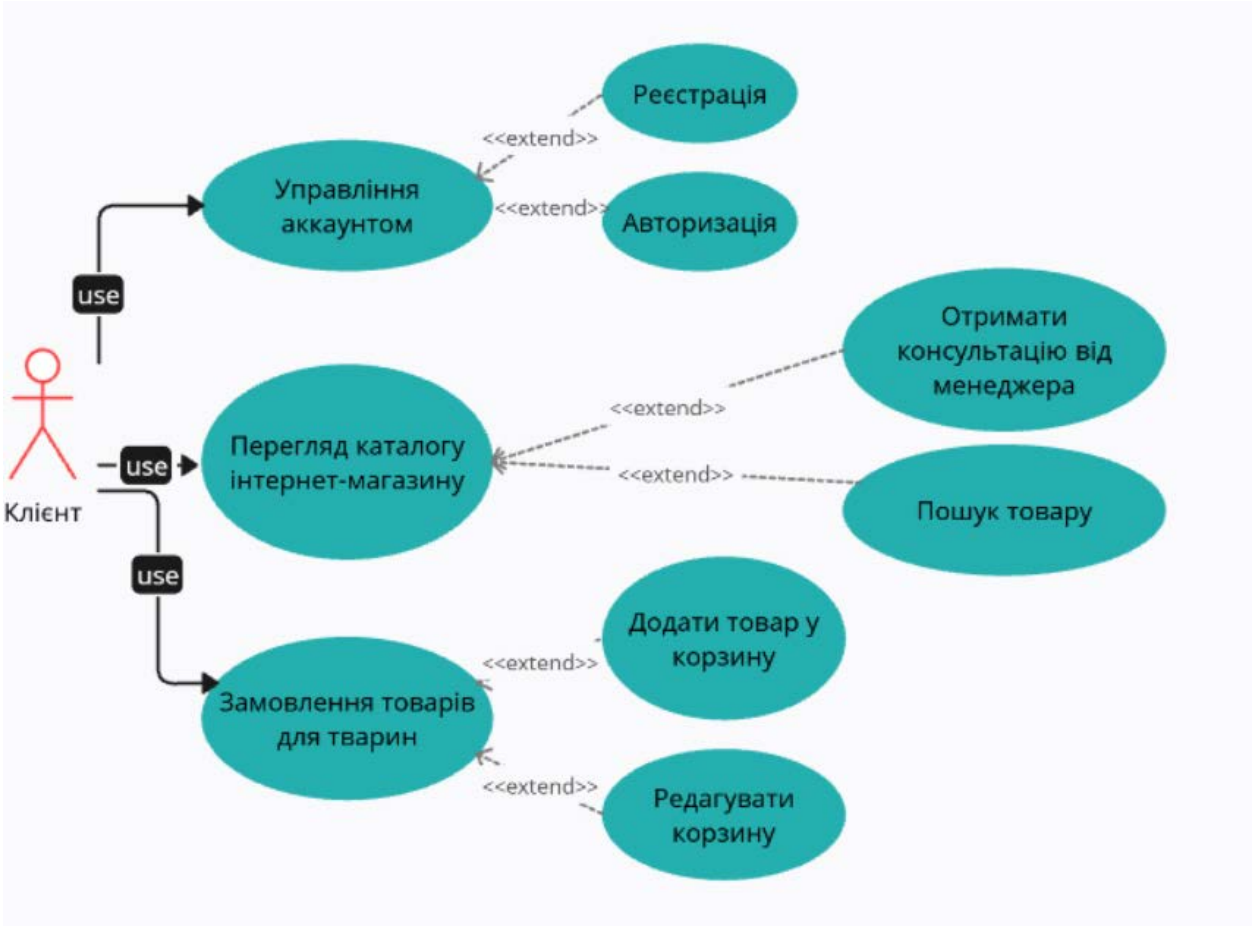

Рисунок 2.1 – Діаграма варіантів використання для актора «Клієнт»

Інтернет-магазин повинен реалізовувати вищезазначений функціонал для актора «Клієнт». Це включає можливість реєстрації та авторизації, управління аккаунтом, перегляд каталогу товарів інтернет-магазину, отримання консультацій від менеджера, пошук товарів, додавання товарів у кошик, редагувати корзину та замовлення товарів для тварин.

Короткий опис варіантів використання для актору «Клієнт» наведено в таблиці 2.1.

Таблиця 2.1 – Варіанти використання для актора « Клієнт»

Актор	**Найменування**	**Пояснення**

Клієнт	Управління аккаунтом	Дозволяє користувачам авторизуватися або зареєструватися, щоб отримати додаткові функції.
	Перегляд каталогу інтернет-магазину	Клієнт може переглядати, шукати товари, а також отримувати консультацію від менеджера.
	Замовлення товарів для тварин	Клієнт може додавати товари до кошику, редагувати її вміст та здійснювати замовлення.

На рисунку 2.2 представлено діаграму використання для актора «Менеджера» веб-додатку інтернет-магазину зоотоварів.

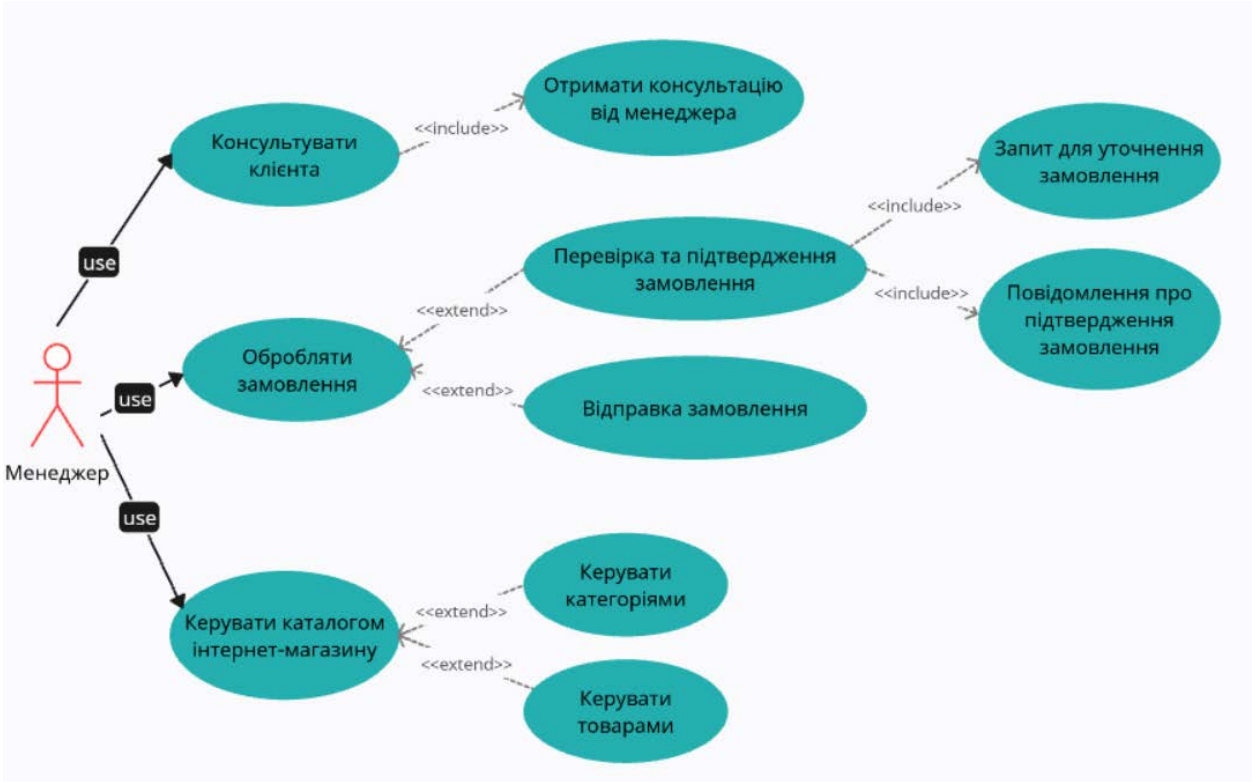

Рисунок 2.2 – Діаграма варіантів використання для актора «Менеджер»

Інтернет-магазин повинен реалізовувати вищезазначений функціонал для актора «Менеджер». Короткий опис варіантів використання для актору «Менеджер» наведено в таблиці 2.2.

Таблиця 2.2 – Варіанти використання для актора « Менеджер»

Актор	Найменування	Пояснення
Менеджер	Консультувати клієнта	Менеджер надає клієнту вичерпні відповіді на запитання щодо товару, його характеристик, використання, ціни, терміну придатності тощо.
	Обробляти замовлення	До цього варіанту використання входить перевірка та підтвердження замовлення, відправка замовлення.
	Керувати каталогом інтернет-магазину	Менеджер має можливості керувати категоріями та товарами для інтернет-магазину.

2.3. Проєктування дизайну у Figma

Процес проєктування інтерфейсу користувача (UI) є ключовим етапом розробки веб-додатку. Для створення прототипів та макетів було обрано онлайн-інструмент Figma завдяки його зручності, можливості командної роботи та широкого функціоналу.

Figma – це онлайн-інструмент для дизайну інтерфейсів, який дозволяє створювати макети, прототипи та документацію. Він є одним з найпопулярніших інструментів для дизайну, особливо для командної роботи, тому що дозволяє взаємодіяти та спільно працювати над проєктами [7].

Розглянемо основні можливості Figma:

Створення макетів та прототипів: Figma надає потужні інструменти для створення візуальних концепцій інтерфейсів. За допомогою засобів малювання та верстки можна розробляти детальні макети сторінок, а також створювати інтерактивні прототипи для демонстрації роботи функцій і переходів між сторінками.

Командна співпраця: платформа дозволяє організувати спільну роботу над проєктами. Є можливість додавати колег до проєкту, що дозволяє залишати коментарі, давати відгуки та вносити зміни в режимі реального часу, що значно полегшує командну взаємодію.

Робота з бібліотеками та шаблонами: Figma містить готові бібліотеки елементів дизайну та шаблони, які допомагають скоротити час розробки. Крім того, можна створювати власні бібліотеки, що дозволяє зберігати та повторно використовувати найбільш потрібні компоненти.

Експорт: інструмент дозволяє експортувати створені макети і прототипи в різних форматах (PNG, SVG, JPG, PDF тощо), що забезпечує зручність подальшої роботи з дизайном.

Хмарне збереження та синхронізація: усі дані зберігаються в хмарі, що дає змогу отримувати доступ до проєктів з будь-якого пристрою в будь-який час. Автоматичне збереження та синхронізація змін у режимі реального часу гарантують безпеку роботи й мінімізують ризик втрати даних.

Інтеграція: Figma інтегрується з багатьма іншими інструментами, такими як Jira, Trello, Slack та інші, що сприяє більш ефективному обміну інформацією і скорочення часу, необхідного для реалізації проєктів.

Таким чином, Figma є надзвичайно потужним інструментом для створення дизайну інтерфейсів. Завдяки широким можливостям, підтримці командної роботи та інтеграціям із сторонніми сервісами, вона стала популярним вибором серед дизайнерів і розробників для реалізації складних проєктів(див. рис. 2.3).

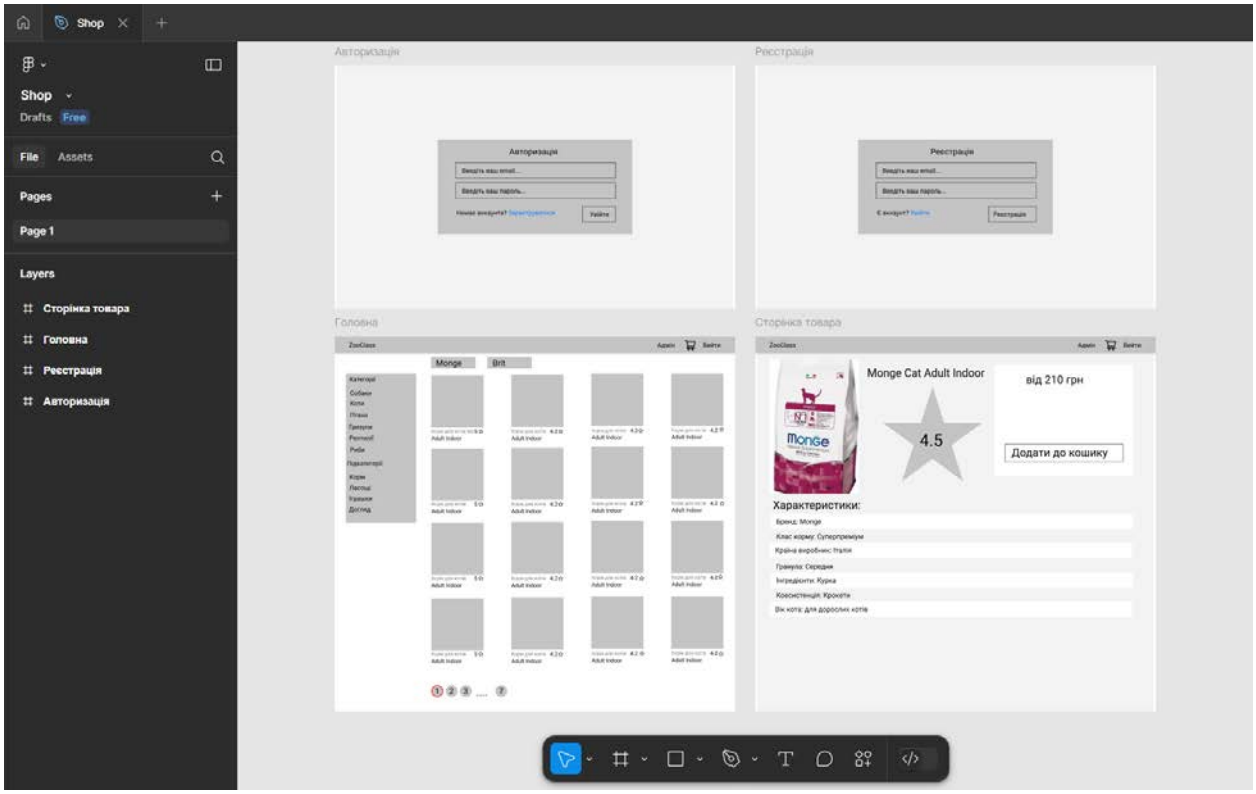

Рисунок 2.3 – Дизайн інтернет-магазину у Figma

У межах проєкту було створено кілька макетів основних сторінок веб-додатку інтернет-магазину зоотоварів, а саме:

Сторінка авторизації: містить поля для введення електронної пошти та пароля, кнопку «Увійти», а також посилання на реєстрацію нового користувача.

Сторінка реєстрації: аналогічна за структурою до авторизації, але з кнопкою «Реєстрація» і посиланням на вхід у разі наявності облікового запису.

Головна сторінка магазину: реалізовано навігаційне меню з категоріями товарів (наприклад, «Коти», «Собаки», «Птахи» тощо), та з брендами продукції (Monge, Brit тощо), блок із товарами у вигляді карток з назвою, рейтингом, фото, а також верхню панель із кнопками входу та реєстрації.

Сторінка товару: демонструє окремий товар із фото, назвою, ціною, рейтингом, кнопкою «Додати до кошику», та характеристиками: бренд, клас корму, країна-виробник, інгредієнти, форма, для кого призначено тощо.

Усі ці елементи створено у Figma вручну з використанням базових інструментів: прямокутників, текстових блоків, зображень, сітки та компонентів. Дизайн має мінімалістичний стиль для демонстрації основного функціоналу без надмірного графічного навантаження.

2.4. Створення діаграми бази даних

Діаграма бази даних (також відома як ER-діаграма, від англ. Entity-Relationship Diagram) – це графічне зображення структури бази даних, яка показує сутності (таблиці), їх атрибути (поля) та зв’язки між ними. ER-діаграма допомагає проєктувальникам, розробникам і аналітикам візуалізувати логічну структуру системи щодо створення самої бази даних.

Проєктування бази даних є одним з ключових етапів інформаційної системи. Важливою складовою цього процесу є побудова діаграми бази даних, яка відображає логічну структуру збереження та обробки даних. Створення діаграми бази даних передбачає послідовне виконання кількох етапів, кожен з яких має специфічне значення в контексті формування ефективної та оптимізованої структури сховища даних.

Визначення вимог до бази даних: на початковому етапі здійснюється збір і аналіз вимог до майбутньої системи з точки зору її інформаційного забезпечення. Цей процес передбачає тісну взаємодію з кінцевими користувачами або замовником, а також вивчення предметної області. Необхідно з’ясувати, які об’єкти підлягають збереженню, які атрибути їм притаманні, та як вони взаємодіють між собою. Результатом даного етапу є формалізований перелік вимог до даних і взаємозв’язків між ними, що стане основою для побудови моделі даних.

Побудова концептуальної моделі: концептуальна модель (модель «сутність-зв’язок», або ER-модель) відображає загальну логіку структури даних у незалежній від реалізації формі. Вона фокусується на сутностях, які існують у предметній області, їх атрибутах та взаємозв’язках. На цьому етапі визначаються типи зв’язків (один до одного, один до багатьох, багато

до багатьох), їх кардинальність, а також обмеження цілісності. Модель зазвичай візуалізується у вигляді ER-діаграми, яка є інструментом формалізації логіки системи. Вона дозволяє розробникам, аналітикам і зацікавленим сторонам отримати уявлення про логічну структуру бази даних.

Формування логічної моделі: логічна модель є деталізованим продовженням концептуальної, яка наближає структуру даних до реалізації. Вона створюється з урахуванням обмежень конкретної системи керування базами даних (СКБД), однак ще не включає аспектів фізичного зберігання. У логічній моделі визначається:

- Типи даних для кожного атрибута;
- Первинні та зовнішні ключі;
- Унікальність та обов’язковість полів;
- Норми нормалізації для усунення надлишковості та забезпечення цілісності даних.

Таким чином, логічна модель забезпечує точний опис схеми таблиць, яка буде реалізована у базі даних.

Розробка фізичної моделі: фізична модель відображає конкретні технічні аспекти реалізації бази даних на основі логічної структури. На цьому етапі враховуються особливості вибраної СКБД, а також параметри, що впливають на продуктивність і масштабованість системи.

До основних задач побудови фізичної моделі належать:

- Вибір точних типів даних з урахуванням обсягу та частоти використання інформації;
- Створення індексів для прискорення пошуку та фільтрації даних;
- Визначення ключів, зв’язків і обмежень між таблицями;
- Оптимізація структури таблиць з метою економії ресурсів зберігання.

Після завершення цього етапу створюється структурована схема даних, готова до реалізації.

Реалізація бази даних: наступним кроком є безпосереднє створення бази даних та відповідних таблиць на основі фізичної моделі. Реалізація може відбуватися як шляхом написання SQL-скриптів, так і за допомогою інструментів візуального проєктування [8]. На цьому етапі:

- Створюються всі об’єкти бази даних: таблиці, обмеження, зв’язки;
- Визначаються початкові значення або тестові дані;
- Перевіряється коректність реалізації моделі згідно з вимогами.

Налагодження та оптимізація: останній етап передбачає перевірку працездатності бази даних, її налагодження та оптимізацію для досягнення максимальної ефективності при виконанні запитів. Зокрема, проводяться такі дії:

- Аналіз продуктивності запитів до таблиць;
- Створення додаткових індексів;
- Оптимізація структури таблиць, якщо це необхідно;
- Налаштування прав доступу до даних;
- Забезпечення захисту цілісності інформації.

Налагодження бази даних має критичне значення в умовах роботи з великими обсягами інформації або високим навантаженням на систему.

На рисунку 2.4 зображена діаграма бази даних до веб-додатку інтернет-магазину зоотоварів.

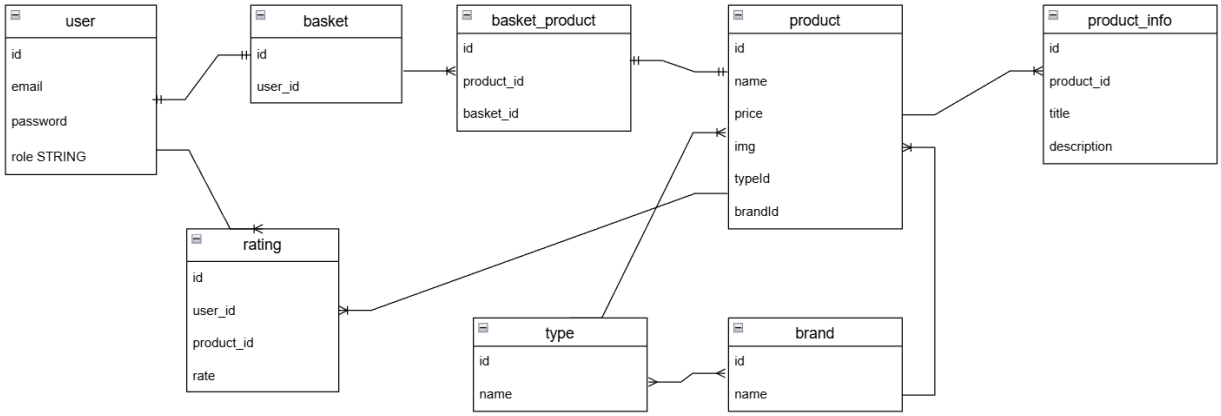

Рисунок 2.4 – Структура бази даних

У діаграмі бази даних можуть бути різні типи зв’язків між таблицями.

Один до одного (One-to-One): кожен запис в одній таблиці має відповідний запис в іншій таблиці і навпаки. Такий тип зв’язку зазвичай використовується, коли інформація, яка відноситься до одного запису, міститься у двох окремих таблицях, для полегшення роботи з даними.

Один до багатьох (One-to-Many): кожен запис в одній таблиці може мати багато відповідних записів в іншій таблиці, але кожен запис в іншій таблиці має лише один відповідний запис у першій таблиці. Такий тип зв’язку використовується, коли один запис може мати багато пов’язаних даних, наприклад, замовлення в інтернет-магазині має багато товарів.

Багато до багатьох (Many-to-Many): кожен запис в одній таблиці може мати багато відповідних записів в іншій таблиці, і навпаки. Такий тип зв’язку використовується, коли кожен запис може мати багато пов’язаних даних, і кожен запис у другій таблиці також може мати багато пов’язаних даних. Для забезпечення такого типу зв’язку необхідно використовувати додаткову таблицю-зв’язок, яка містить ключі обох таблиць.

Висновки до розділу 2

У другому розділі було детально розглянуто процес проєктування веб-додатку інтернет-магазину зоотоварів, що є ключовим етапом у створенні повноцінної інформаційної системи. Було визначено основні цілі та вимого

до системи, які охоплюють функціональність онлайн-продажів, управління асортиментом, обробку замовлень, комунікацію з клієнтами та забезпечення безпеки даних.

Для візуалізації взаємодії користувачів із системою були розроблені діаграми варіантів використання (Use Case), які охоплюють функціонал як для клієнта, так і для менеджера. Це дозволяє чітко структурувати сценарії використання та краще зрозуміти ролі користувачів у системі.

Особливу увагу було приділено розробці інтерфейсу користувача у Figma. Завдяки цьому інструменту було створено візуальні макети, що сприяють зручності користування системою та відповідають сучасним вимогам до UI/UX дизайну.

Також було виконано поетапне проєктування бази даних: від збору вимог і створення концептуальної ER-моделі до логічної та фізичної реалізації. Було розглянуто основні типи зв’язків між таблицями (один до одного, один до багатьох, багато до багатьох), що забезпечує цілісність і ефективність збереження даних.

Таким чином, проведене проєктування заклало міцну основу для подальшої реалізації веб-додатку, забезпечуючи відповідність функціоналу технічним і бізнес-вимогам.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОЄКТУ

3.1. Реалізація серверної частини

Процес реалізації серверної частини веб-додатку інтернет-магазину зоотоварів із використанням середовища Node.js передбачає виконання низки послідовних етапів, кожен з яких забезпечує необхідні умови для стабільного та масштабованого функціонування системи [9]. Розглянемо основні кроки розробки більш детально.

Інсталяція Node.js: початковим етапом є встановлення середовища виконання Node.js, яке дозволяє запускати JavaScript-код на серверній стороні. Node.js поширюється у вигляді інсталяційних пакетів для різних операційних систем (Windows, Linux, macOS), які можна завантажити з офіційного ресурсу розробника. Коректне встановлення Node.js включає також менеджер пакетів npm (Node Package Manager), який забезпечує керування залежностями та пакетами, необхідними для розробки додатку.

Вибір фреймворку: для спрощення побудови серверної частини веб-додатку на Node.js доцільно скористатися одним із доступних фреймворків. Найпоширенішим є Express.js – мінімалістичний і гнучкий фреймворк, який забезпечує зручне управління маршрутизацією, обробкою HTTP-запитів та middleware-функціональністю. Альтернативами можуть виступати Sails.js, Koa.js або Meteor.js. Вибір фреймворка здійснюється з урахуванням технічних вимог проєкту, рівня складності, очікуваного навантаження та структури даних.

Налаштування бази даних: інтернет-магазин передбачає зберігання значного обсягу структурованої інформації, такої як дані про товари, користувачів, замовлення тощо. Для цього необхідно інтегрувати базу даних із серверною частиною. Найбільш поширеними базами даних є PostgreSQL, MySQL, MongoDB. Після встановлення обраної бази даних здійснюється налаштування з’єднання за допомогою відповідних драйверів

або ORM-бібліотек, що забезпечують зручну роботу з базою даних на рівні об’єктів.

Розробка API: ключовим етапом є створення API, яке реалізує взаємодію між клієнтською (фронтенд) та серверною (бекенд) частинами додатку. API має надавати функціональність для реалізації базових операцій: отримання списку товарів, фільтрація та сортування, додавання елементів до кошика, реєстрація та авторизація користувачів та інше.

Ініціалізація проєкту: перед безпосередньою розробкою необхідно здійснити ініціалізацію Node.js-проєкту. Цей процес включає в себе створення файлів та каталогів, необхідних для проєкту, а також налаштування його середовища для розробки та випуску.

3.1.1. Ініціалізація проєкту

Основна мета ініціалізації проєкту Node.js полягає в налаштуванні його залежностей та структури проєкту. Залежності – це пакети, які потрібні для функціонування вашого проєкту, такі як бібліотеки, фреймворки та інші засоби, які ви плануєте використовувати в проєкті. Встановлення та налаштування цих залежностей допоможе забезпечити правильне функціонування проєкту.

Крім того, ініціалізація проєкту Node.js також дозволяє налаштувати вихідну структуру проєкту (див. рис. 3.1). Це може включати створення каталогів та файлів, необхідних для проєкту, таких як файли з конфігурацією, файли з даними, файли з логами та інші. Керування структурою проєкту допоможе зробити код більш організованим та легко зрозумілим для інших розробників, які працюватимуть з проєктом в майбутньому.

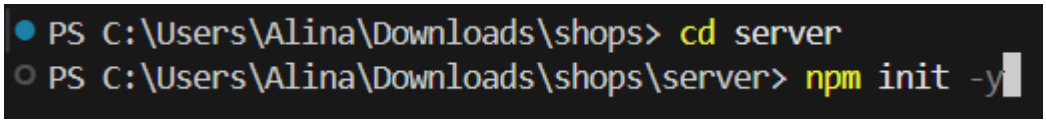

Рисунок 3.1 – Ініціалізація проєкту

Крім того, ініціалізація проєкту Node.js дозволяє налаштувати різні параметри проєкту, такі як середовище виконання, версії пакетів, сторонні

бібліотеки та інші. Це допоможе забезпечити максимальну продуктивність та ефективність вашого проєкту на Node.js.

3.2. База даних

PGAdmin є безкоштовним, відкритим програмним забезпеченням з графічним інтерфейсом користувача, призначеним для адміністрування та управління системою керування базами даних PostgreSQL [10]. Цей інструмент широко використовується адміністраторами баз даних, розробниками та аналітиками завдяки інтуїтивному інтерфейсу, гнучкому функціоналу та підтримці широкого спектра операцій з базами даних.

PGAdmin забезпечує взаємодію з PostgreSQL за допомогою візуального середовища, що значно спрощує виконання рутинних операцій порівняно з використанням командного рядка. До основних функціональних можливостей PGAdmin належать наступні:

Створення та управління об'єктами бази даних: інструмент дозволяє створювати нові бази даних, таблиці, схеми, індекси, представлення, тригери та інші об’єкти, характерні для PostgreSQL. Користувачі можуть здійснювати повне керування цими об’єктами: змінювати структуру таблиць, видаляти елементи або створювати зв’язки між ними за допомогою зовнішніх ключів.

Перегляд, редагування та маніпуляція даними: PGAdmin надає зручні засоби для перегляду вмісту таблиць у табличному форматі. Крім того, підтримується можливість прямого редагування даних, включаючи оновлення, вставку нових записів та видалення існуючих. Зміни відображаються у реальному часі, що є особливо корисним для налагодження бази даних під час розробки.

Виконання SQL-запитів: PGAdmin містить вбудований редактор SQL, що дозволяє виконувати запити будь-якої складності: від простих запитів SELECT до складних транзакцій з використанням умов, агрегатів, підзапитів та процедур. Результати виконання запитів відображаються у

зручному форматі, з можливістю експорту у різні формати (CSV, Excel, JSON тощо).

Аналіз та моніторинг баз даних: програма підтримує функції аналізу стану баз даних, зокрема моніторинг активних підключень, статистики використання ресурсів, обсягу даних у таблицях тощо. Крім того, PGAdmin дозволяє генерувати візуалізовані звіти, які можуть бути використані для оптимізації структури бази даних та діагностики проблем продуктивності.

Управління користувачами та безпекою: однією з важливих функцій є адміністрування доступу до баз даних. PGAdmin дозволяє створювати нові облікові записи користувачів, призначати їм ролі та визначати права доступу на рівні таблиць, стовпців або окремих операцій (читання, запис, виконання). Це дозволяє реалізувати багаторівневу модель контролю доступу, відповідно до принципів безпеки інформаційних систем.

Завдяки розширеним можливостям і гнучкості, PGAdmin є потужним інструментом для ефективного управління екземплярами PostgreSQL. Його використання сприяє прискоренню процесів розробки, тестування та обслуговування баз даних, а також покращує загальну продуктивність команд розробників і адміністраторів (див. рис. 3.2).

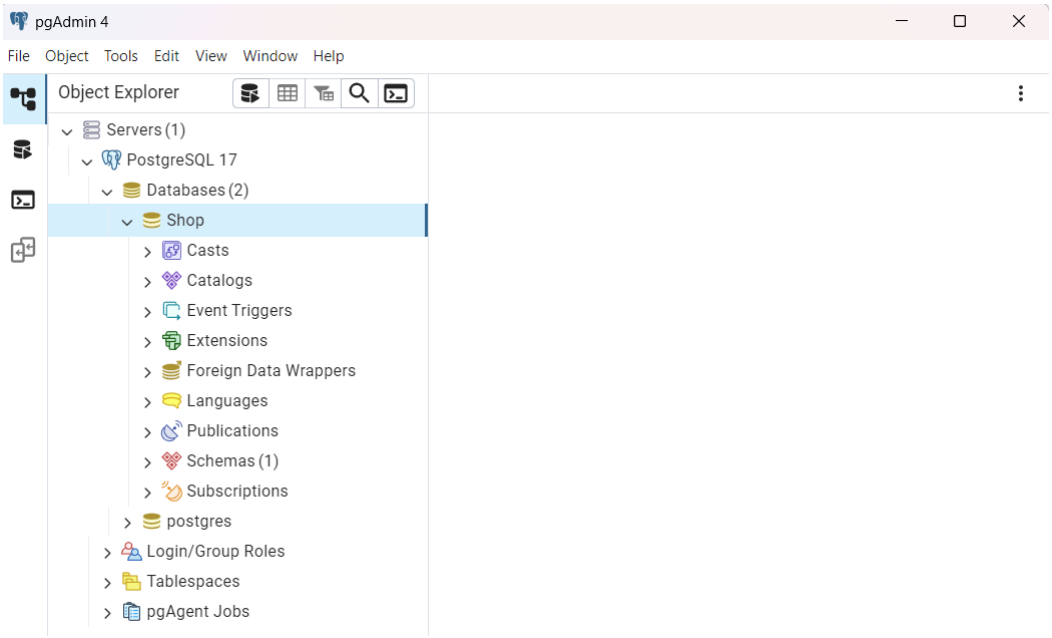

Рисунок 3.2 – Інтерфейс програми PGAdmin 4

3.2.1. Підключення бази даних

Процес підключення бази даних до серверного середовища є критично важливим етапом у забезпеченні функціонування веб-додатку або іншого програмного забезпечення, яке оперує зберіганням і обробкою даних. Коректне налаштування з'єднання з базою даних забезпечує цілісність, доступність і безпеку обміну даними між сервером та системою керування базами даних (СКБД). Нижче наведено основні етапи даного процесу.

Крок перший: встановлення та конфігурування СКБД на сервері. Перед початком роботи з базою даних необхідно встановити відповідне програмне забезпечення для управління даними. У випадку використання PostgreSQL як СКБД, слід виконати інсталяцію програмного пакету PostgreSQL на сервері, після чого налаштувати конфігураційні файли для дозволу зовнішніх підключень.

Крок другий: створення бази даних та облікового запису користувача. Після успішної інсталяції необхідно створити нову базу даних, яка використовуватиметься веб-додатком. Також слід створити окремий обліковий запис користувача з необхідними правами доступу.

Крок третій: налаштування параметрів підключення у серверному коді. Для взаємодії з базою даних із серверної частини веб-додатку, що реалізована на Node.js, зазвичай використовується бібліотека pg (node-postgres). Параметри підключення до СКБД включають: хост, порт, ім'я бази даних, ім'я користувача та пароль.

Крок четвертий: тестування з'єднання з базою даних. Після налаштування підключення слід перевірити його працездатність шляхом виконання простого SQL-запиту. У разі успішного підключення буде отримано відповідь від сервера бази даних із необхідними результатами. Якщо з'єднання не вдалося, потрібно перевірити конфігурацію доступу, правильність облікових даних і доступність СКБД.

Під час налаштування підключення слід приділити особливу увагу захисту конфіденційної інформації. Параметри з'єднання повинні бути

зашифровані або винесені в змінні середовища. Доступ до бази даних має бути обмежений за IP-адресами, а SQL-запити повинні проходити валідацію для запобігання SQL-ін’єкціям.

Для роботи з базою даних створимо файл db.js, в якому реалізуємо підключення до бази даних (див. рис. 3.3).

```
const {Sequelize} = require('sequelize')

module.exports = new Sequelize(
    process.env.DB_NAME, // Назва БД
    process.env.DB_USER, // Користувач
    process.env.DB_PASSWORD, // Пароль
    {
        dialect: 'postgres',
        host: process.env.DB_HOST,
        port: process.env.DB_PORT
    }
)
```

Рисунок 3.3 – Код файлу db.js

Для створення моделей бази даних (БД) у JavaScript можна використовувати різні бібліотеки та фреймворки, такі як Sequelize, Mongoose, Knex тощо. Розглянемо процес створення моделей у Sequelize.

Sequelize — це популярна ORM (Object-Relational Mapping) бібліотека для Node.js, яка забезпечує зручний інтерфейс для взаємодії з реляційними базами даних, такими як PostgreSQL, MySQL, MariaDB, SQLite та Microsoft SQL Server. Вона дозволяє працювати з базою даних на рівні об'єктів, а не через сирі SQL-запити, що значно спрощує розробку, підвищує читабельність коду та сприяє кращій підтримуваності проєкту.

Для створення моделей потрібно спочатку встановити Sequelize та драйвер для конкретної СКБД. Потім потрібно створити об’єкт Sequelize (див. рис. 3.4), вказавши параметри підключення до БД.

```
const Sequelize = require('sequelize');
const sequelize = new Sequelize('database', 'username', 'password', {
  host: 'localhost',
  dialect: 'postgres'
});
```

Рисунок 3.4 – Приклад створення об’єкту Sequelize

3.3. Реєстрація користувачів

Система реєстрації користувачів є ключовим функціональним компонентом інформаційної системи інтернет-магазину, що забезпечує персоналізований доступ користувачів до функцій ресурсу, зокрема – можливість здійснювати покупки, зберігати історію замовлень, управляти профілем тощо. Дана система повинна гарантувати як зручність для користувача, так і безпеку збереження персональних даних. Її реалізація включає декілька обов’язкових етапів і компонентів.

Форма реєстрації: на першому етапі користувач взаємодіє з інтерфейсом реєстрації, заповнюючи форму, яка містить обов’язкові поля, необхідні для створення облікового запису. Як правило, це такі поля, як ім’я, прізвище, електронна адреса, пароль, а в деяких випадках — номер телефону чи адреса доставки. Ці дані є основою для подальшої ідентифікації користувача в системі.

Валідація введених даних: наступним кроком є валідація вхідних даних. Вона виконується як на стороні клієнта (для підвищення зручності), так і на стороні сервера (з метою безпеки). Валідація включає перевірку правильності структури електронної адреси, відповідність пароля вимогам щодо мінімальної довжини, наявності великих/малих літер, цифр і спеціальних символів. Також може виконуватись перевірка унікальності електронної адреси в базі даних.

Обробка та збереження даних: після успішного проходження валідації, система обробляє отриману інформацію та зберігає її у базі даних. При цьому надзвичайно важливо забезпечити захист конфіденційних даних. Зокрема, паролі користувачів не повинні зберігатися у відкритому вигляді — замість цього використовуються криптографічні хеш-функції, які забезпечують надійне шифрування. Дані зберігаються в таблиці користувачів, яка є частиною реляційної моделі бази даних.

Авторизація користувача: після завершення реєстрації користувач отримує можливість входу до свого облікового запису за допомогою вказаної електронної адреси та пароля. Під час авторизації система порівнює введені користувачем облікові дані з записами в базі даних. У разі збігу користувач отримує маркер сесії (token), наприклад, на основі технології JWT (JSON Web Token), який використовується для автентифікації при наступних запитах.

Відновлення доступу до облікового запису: функціонал «забули пароль» передбачає можливість відновлення доступу до облікового запису у разі втрати пароля. Для цього користувач вводить електронну адресу, прив’язану до облікового запису, після чого система надсилає на неї листа з посиланням на форму зміни пароля. Це посилання, як правило, має обмежений термін дії з міркувань безпеки.

Підтвердження електронної адреси: для підвищення рівня безпеки та захисту від автоматизованих реєстрацій (ботів), часто реалізується механізм підтвердження електронної адреси. Після первинної реєстрації користувач отримує електронного листа з унікальним посиланням, перехід за яким активує його обліковий запис. Це дозволяє переконатися у дійсності введеної адреси електронної пошти та зменшити ризик зловживань.

3.3.1. JSON Web Token

JSON Web Token (JWT) — це компактний, безпечний і автономний спосіб передачі інформації між сторонами у вигляді JSON-об'єкта [11]. JWT часто використовується для аутентифікації і авторизації в сучасних веб-додатках і мікросервісних архітектурах.

JWT складається з трьох частин (див. рис. 3.5): заголовка, заяви та підпису. Заголовок та заява містять у собі закодовану інформацію про користувача та деякі дані про токен, які можуть бути використані для перевірки та підтвердження його автентичності. Підпис забезпечує захист від підробки.

Реєстрація через JWT токен передбачає наступні кроки:

- користувач вводить свої особисті дані для реєстрації;
- інформація про користувача зберігається у базі даних, а пароль – в захешованому вигляді;
- при вході в систему користувач вводить свій логін та пароль;
- сервер перевіряє правильність введеного логіна та пароля;
- якщо дані коректні, сервер генерує JWT токен та надсилає його на клієнтську сторону;
- клієнт зберігає отриманий JWT токен у локальному сховищі;
- при кожному наступному запиті до сервера, клієнт додає отриманий токен до заголовку запиту;
- сервер перевіряє підпис токену та достовірність інформації, що міститься в заяві;
- якщо перевірка успішна, сервер повертає запитані дані.

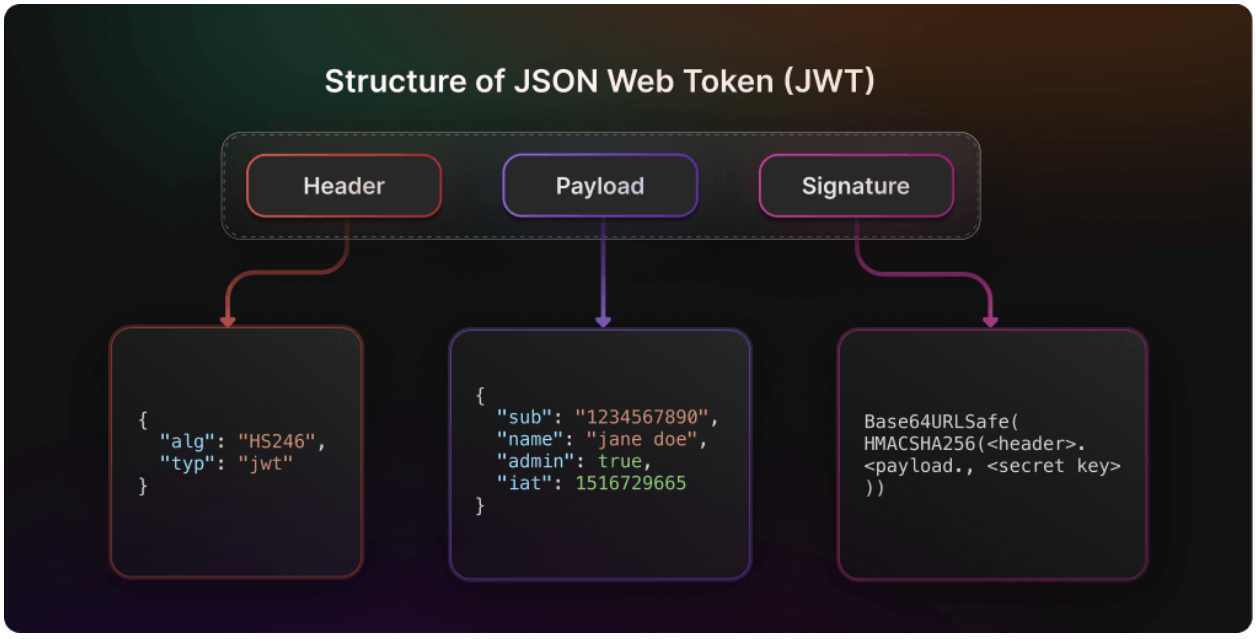

Рисунок 3.5 – Структура JWT

Застосування JWT (JSON Web Token) у процесі автентифікації дозволяє відмовитися від збереження стану сесії на стороні сервера. Це забезпечує низку переваг, зокрема спрощення побудови масштабованих та розподілених систем, зменшення навантаження на серверні ресурси, а також

підвищення загального рівня безпеки обробки облікових даних користувачів.

Для реалізації функціоналу автентифікації у середовищі Node.js необхідно встановити низку зовнішніх бібліотек. Зокрема, використовуються модулі «jsonwebtoken» і «bcrypt», які можна додати до проєкту за допомогою наступної команди: «npm install jsonwebtoken bcrypt».

«Jsomwebtoken» - бібліотека, яка надає інструменти для створення, підписування та верифікації JWT. Вона використовується для генерації токена під час входу користувача та його подальшої перевірки під час звернення до захищених маршрутів API.

«bcrypt» - модуль, призначений для безпечного хешування паролів. Він реалізує криптографічно стійкий алгоритм хешування «bcrypt», що дозволяє захистити паролі користувачів навіть у випадку компрометації бази даних.

Хешування паролів є обов'язковим етапом при збереженні облікових даних у базі даних. На відміну від шифрування, хеш-функція є односторонньою – її результат не може бути зворотно перетворений у вихідне значення. Це означає, що навіть у випадку витоку бази даних зловмисник не зможе відновити реальні паролі користувачів (див. рис. 3.6).

```
async registration(req, res, next) {
    const {email, password, role} = req.body
    if (!email || !password) {
        return next(ApiError.badRequest('Некоректний email або password'))
    }
    const candidate = await User.findOne({where: {email}})
    if (candidate) {
        return next(ApiError.badRequest('Користувач з таким email вже існує'))
    }
    const hashPassword = await bcrypt.hash(password, 5)
    const user = await User.create({email, role, password: hashPassword})
    const basket = await Basket.create({userId: user.id})
    const token = generateJwt(user.id, user.email, user.role)
    return res.json({token})
}
```

Рисунок 3.6 – Код системи реєстрації

3.4. Реалізація клієнтської частини

Реалізація клієнтської частини веб-додатку за допомогою JavaScript та React включає в себе наступні кроки.

Створення проєкту React за допомогою create-react-app або аналогічного інструменту (див. рис. 3.7) [12].

```
PS C:\Users\Alina\Downloads\shops\client> npm create-react-app .
```

Рисунок 3.7 – Створення проєкту React

Цей інструмент автоматично налаштує необхідні залежності та структуру проєкту (див. рис. 3.8).

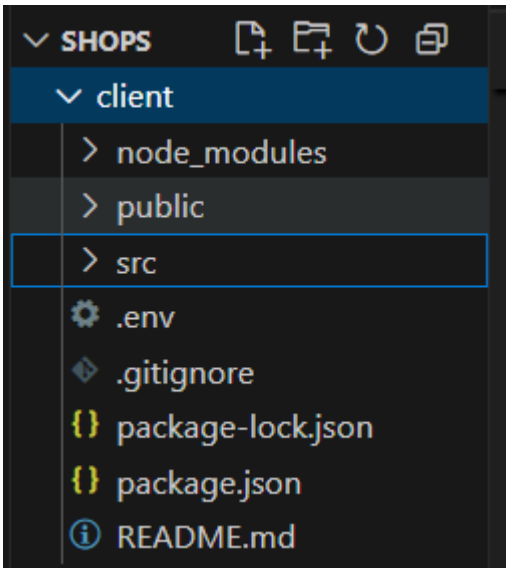

Рисунок 3.8 – Проєкт після виконання команди на рис. 3.7

Розробка компонентів, які відображатимуться на сторінці. Компоненти в React – це незалежні модулі, які містять логіку та представлення частини інтерфейсу користувача. Компоненти можна вкладати один в одного, утворюючи ієрархію компонентів.

Визначення стилів для компонентів. Для стилізації компонентів можна використовувати CSS, SASS або інші препроцесори CSS. Також можна використовувати бібліотеки стилів, такі як Bootstrap або Material UI.

Реалізація взаємодії між компонентами та зв’язок з серверною частиною додатку. Для зв’язку з сервером можна використовувати AJAX запити, Fetch API або бібліотеки, такі як Axios або jQuery.

Тестування та налагодження коду. Для тестування можна використовувати різні інструменти, такі як Jest, Enzyme або React Testing Library. Для налагодження коду можна використовувати DevTools у браузері.

Підготовка для публікації. Після завершення розробки клієнтської частини додатку необхідно зібрати та оптимізувати код для публікації. Для цього можна використовувати інструменти, такі як Webpack або Parcel [13].

3.4.1. Інсталяція залежностей

Команда «npm install axios react-router-dom mobx mobx-react-lite» встановлює три пакети:

1) axios — це популярна JavaScript-бібліотека, яка надає інтерфейс для здійснення HTTP-запитів до серверної частини додатку. Вона підтримує всі основні методи HTTP (GET, POST, PUT, DELETE тощо) та дозволяє обробляти відповіді сервера, керувати заголовками запитів, передавати параметри та обробляти помилки.

2) react-router-dom — це бібліотека маршрутизації, яка дозволяє реалізувати навігацію між різними компонентами або сторінками у React-додатку. Вона підтримує маршрутизацію на основі URL-адрес, дає змогу динамічно змінювати вміст сторінки без повного перезавантаження, а також дозволяє використовувати історію навігації браузера.

3) mobx та mobx-react-lite: ці дві бібліотеки дозволяють легко реалізувати патерн State Management в React додатках (mobx забезпечує зручний спосіб для створення та управління станом додатка, а mobx-react-lite робить його легко доступним для використання в React компонентах).

3.4.2. Створення сторінки авторизації та реєстрації

Створення клієнтської частини сторінки реєстрації на сайті можна зробити за допомогою React та деяких додаткових бібліотек. Перед початком роботи необхідно встановити бібліотеки за допомогою команди «npm install».

Основним елементом сторінки реєстрації є форма, яка складається з декількох полів (див. рис. 3.9), наприклад, поле введення імені, електронної пошти, пароля та кнопки «Зареєструватися». Така сама форма для авторизації (див рис. 3.10).

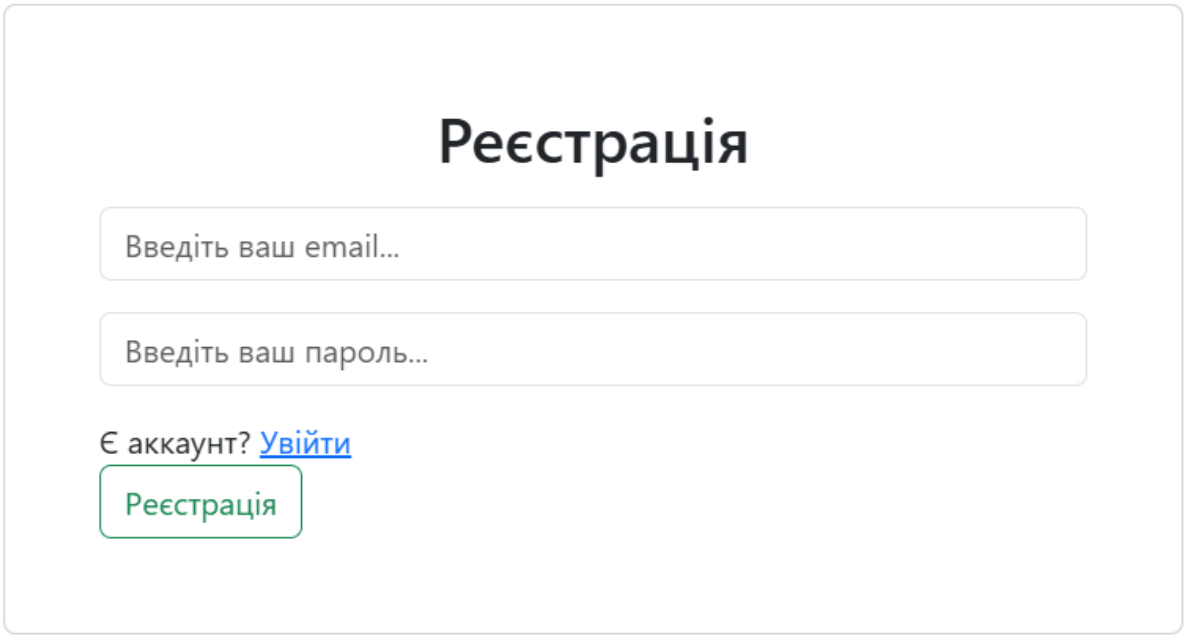

Рисунок 3.9 – Сторінка реєстрації

Щоб зробити цю форму, необхідно створити компонент React зі станом, який зберігає дані, введені користувачем в форму.

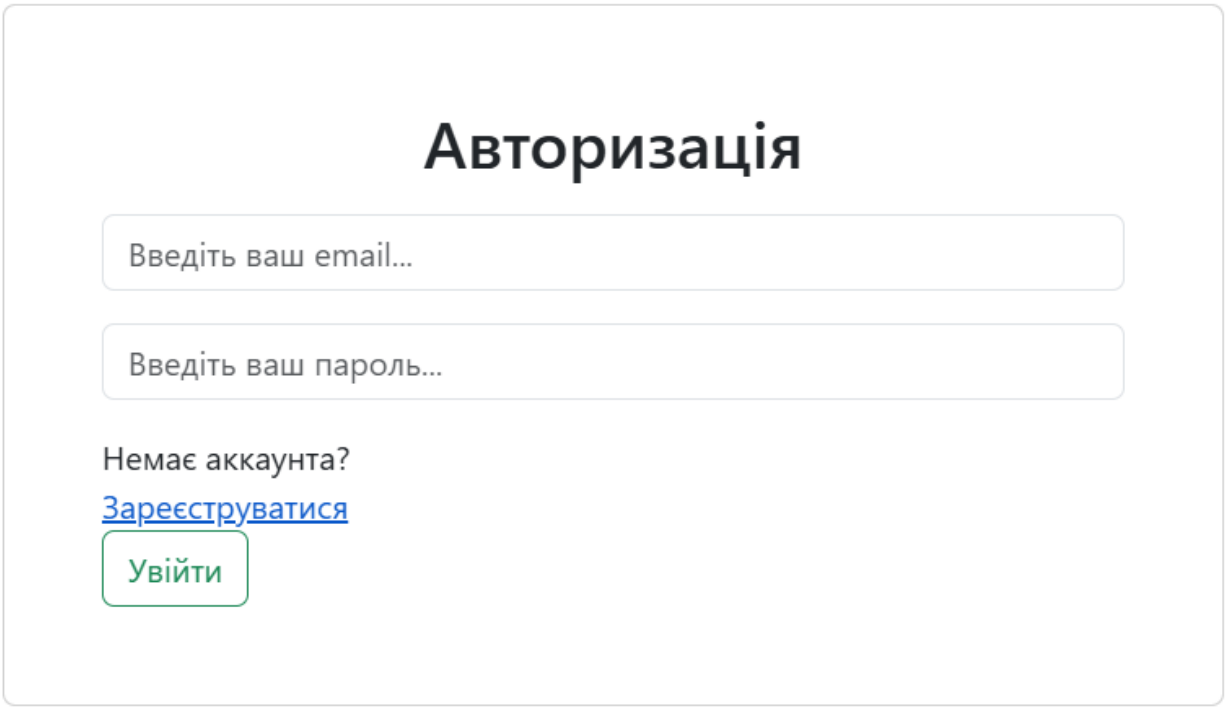

Рисунок 3.10 – Сторінка авторизації

Для зручності обробки форми можна використати бібліотеку axios, яка дозволяє відправляти запити на сервер. При натисканні на кнопку «Зареєструватися», потрібно створити обробник подій, який відправить дані з форми на сервер за допомогою axios.post().

При успішній реєстрації користувача на сервері буде створений запис у базі даних з введеними даними. Для збереження даних, які ввів користувач, можна використати сторонній стейт-менеджер, наприклад, MobX. Цей стейт-менеджер дозволяє зручно управляти даними, які зберігаються в програмі, та забезпечує їх автоматичне оновлення при зміні стану.

3.4.3. Створення переходу між сторінками

Для створення переходу між сторінками веб-сайту на фронтенд можна використовувати бібліотеку React Router.

Перш за все, потрібно встановити React Router за допомогою команди npm install react-router-dom. Після встановлення можна створювати компоненти сторінок та визначати, як вони будуть відображатись при переході на них (див. рис. 3.11).

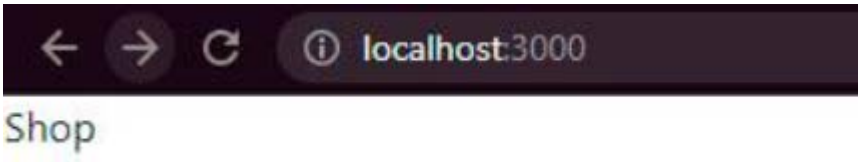

Рисунок 3.11 – Перехід між сторінками

Для цього необхідно імпортувати необхідні компоненти з React Router, такі як BrowserRouter для обгортання всього додатку та Route для визначення шляху до компоненту сторінки (див. рис. 3.12).

```
import React, {useContext} from 'react';
import { Routes, Route, Navigate } from 'react-router-dom';
import {authRoutes, publicRoutes} from "../routes";
import {SHOP_ROUTE} from "../utils/consts";
import {Context} from "../index";
import {observer} from "mobx-react-lite";

const AppRouter = observer(() => {
    const {user} = useContext(Context)

    console.log(user)
    return (
        <Routes>
            {user.isAuth && authRoutes.map(({ path, Component }) =>
                <Route key={path} path={path} element={<Component/>} exact />
            )}
            {publicRoutes.map(({ path, Component }) =>
                <Route key={path} path={path} element={<Component/>} exact />
            )}
            <Route path="*" element={<Navigate to={SHOP_ROUTE} />}/>
        </Routes>
    );
});

export default AppRouter;
```

Рисунок 3.12 – Реалізація переходу між сторінками

3.5. Тестування проєкту

Процес тестування інформаційної системи є невід'ємною складовою життєвого циклу розробки програмного забезпечення. Його основна мета полягає у виявленні функціональних та нефункціональних дефектів програмного забезпечення до моменту його впровадження в експлуатацію, з метою забезпечення відповідності системи технічному завданню, вимогам користувачів та стандартам якості.

Тестування інформаційної системи включає в себе створення тестових сценаріїв та даних для перевірки різних аспектів системи, таких як функціональність, надійність, продуктивність, безпеку, сумісність та інші. Ці сценарії виконуються з метою виявлення помилок та дефектів в програмному забезпеченні, що дозволяє їх виправити до того, як інформаційна система буде введена в експлуатацію.

Тестування інформаційної системи може проводитися на різних етапах її розробки та впровадження, включаючи модульне тестування, інтеграційне тестування та системне тестування. При цьому використовуються різні методи та підходи до тестування, такі як ручне тестування, автоматизоване тестування, тестування методом чорного ящика та тестування методом білого ящика.

Після запуску у браузері відкривається localhost:3000, на якому ми можемо побачити головну сторінку нашого сайту (див. рис. 3.13).

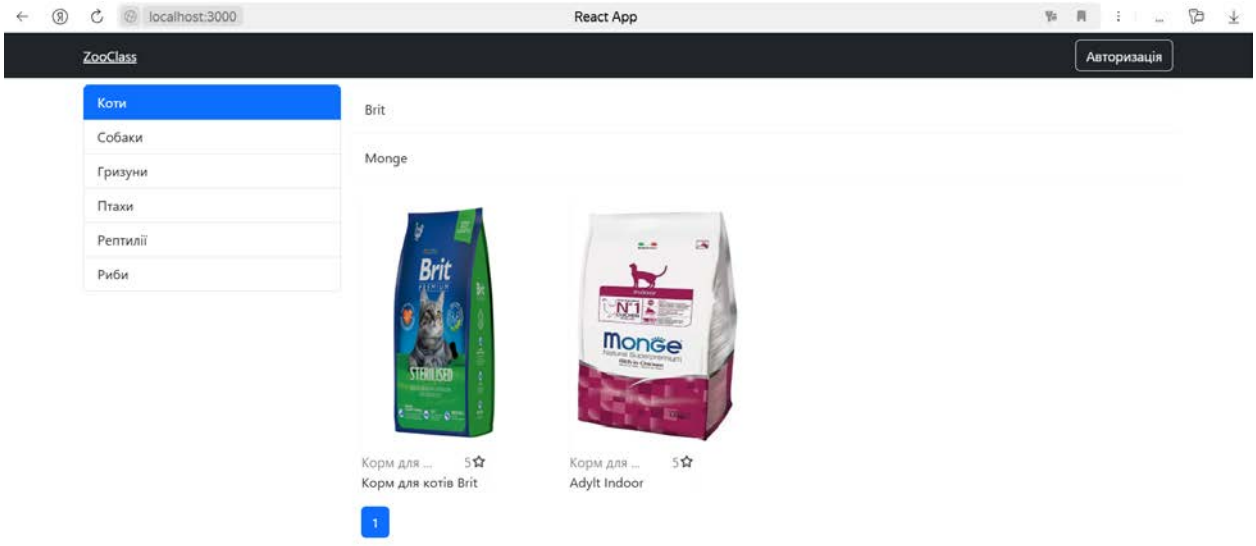

Рисунок 3.13 – Головна сторінка інтернет-магазину

На головній сторінці ми бачимо список товарів які є в нашому магазині. Як приклад змінимо деякі товари (див. рис. 3.14).

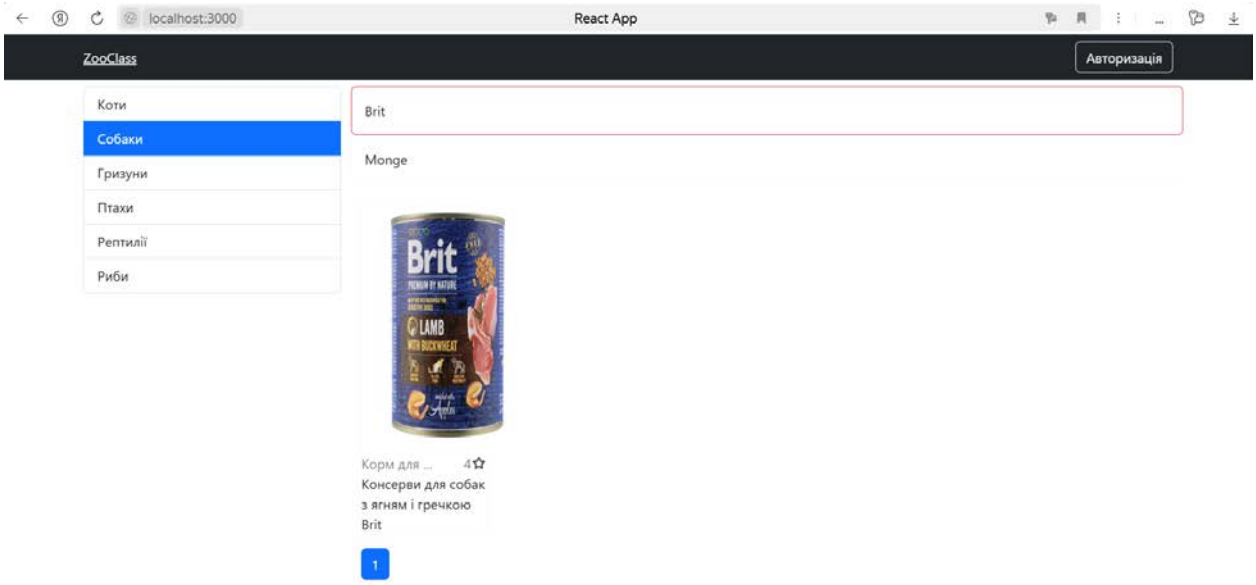

Рисунок 3.14 – Сторінка після зміни товарів

Після додавання нових товарів, натиснемо на товар який нас цікавить, та перейдемо на сторінки з ним (див. рис. 3.15).

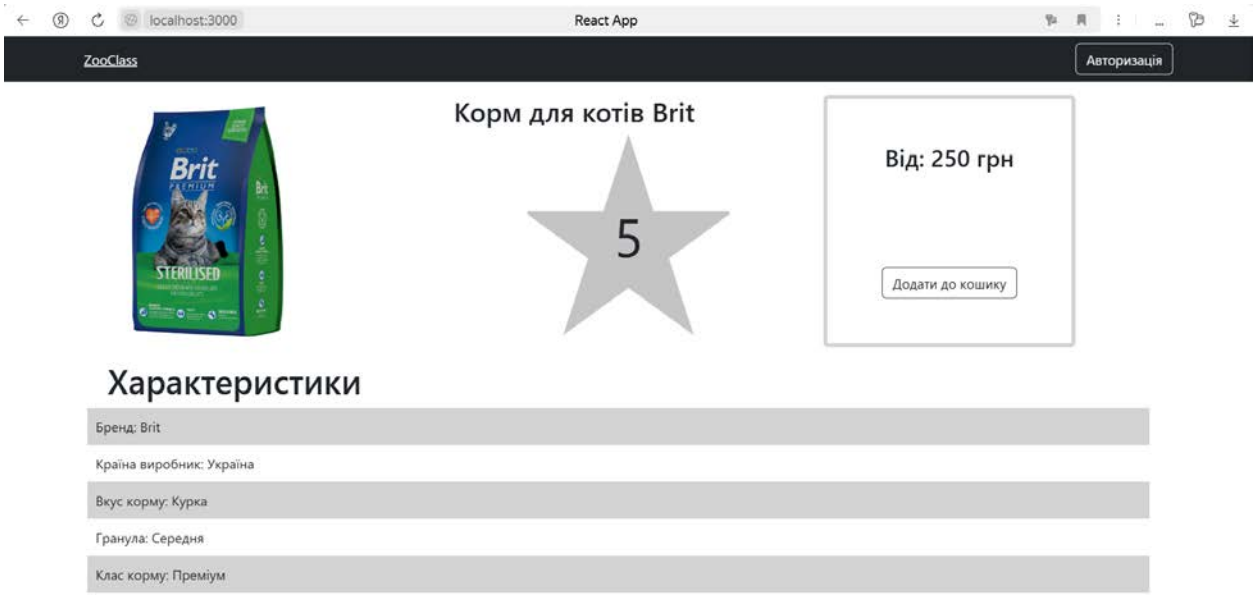

Рисунок 3.15 – Сторінка окремого товару

У кожного товару є своя назва, рейтинг, ціна та деякі характеристики стосовно цього товару, це все ми можемо подивитися на сторінці товару.

Також кожний товар має свій унікальний ідентифікатор (id), завдяки якому виконується перехід на сторінку певного товару.

Після цього натиснемо на кнопку «Реєстрація», та пройдемо етап реєстрації на сайті (див. рис. 3.16).

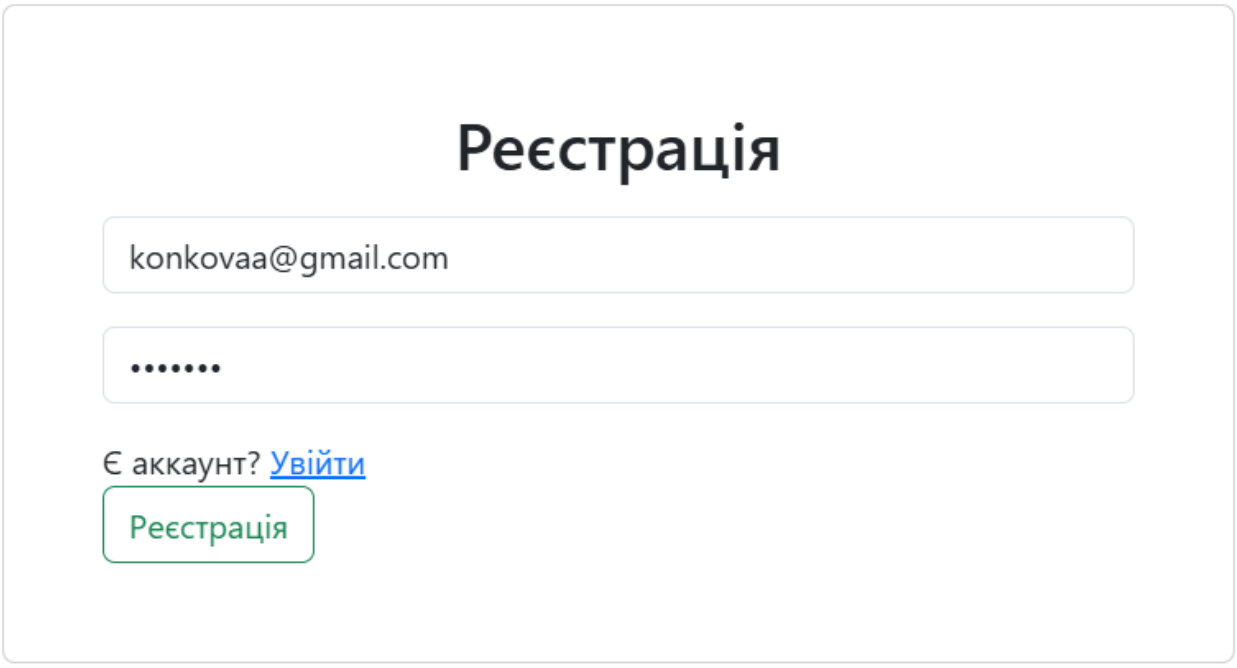

Рисунок 3.16 – Реєстрація на сайті

Після введення своїх даних у поле реєстрації, за допомогою програми Postman, ми можемо побачити JWT-token (див. рис. 3.17) який буде використовуватися подалі для авторизації на нашому сайті.


```
{
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
        eyJpZCI6OCwiZW1haWwiOiJrb25rb3ZhYUBnbWFpbC5jb20iLCJyb2xlIjoiQURNSU4iLCJpYXQiOjE3NDYyMDc1NjUsImV
        4cCI6MTc0NjI5Mzk2NX0.Yp77LFm9OcT8DoqnC_rBSQU-i4Ldq2PEAZPt6iqbBZc"
}
```


Рисунок 3.17 – Отримання JWT токену після реєстрації

Після авторизації перейдемо на головну сторінку (див. рис. 3.18).

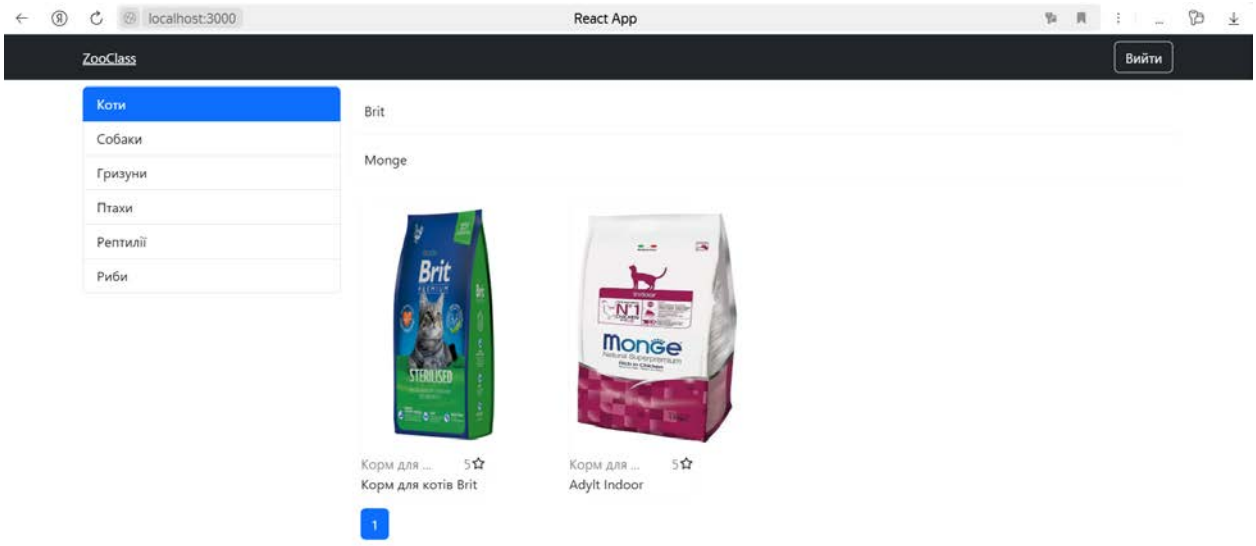

Рисунок 3.18 – Сторінка інтернет-магазину після авторизації

3.5.1. Тестування API-запитів та виявлення помилок

Для тестування серверної логіки використовувався Postman. Основні перевірені запити:

- POST /api/user/registration – тестувалася реєстрація нового користувача.
  - Успішний кейс: при валідних даних повертається статус «200 OK» та JWT-токен.
  - Помилка: при повторній реєстрації з тією ж email-адресою поверталася помилка «409 Conflict». Було виправлено логіку перевірки унікальності користувача в контролері.
- POST /api/user/login – тестувалася авторизація. Перевірялося:
  - Неправильний email – відповідь «401 Unauthorized».
  - Неправильний пароль – відповідь «401 Unauthorized».
  - Правильні дані – статус «200 OK», отримання JWT-токена.
- GET /api/product – отримання списку товарів.
  - При запиті без токена – «403 Forbidden» (перевірка безпеки).

- При запиті з валідним токеном – повертається масив товарів.

- GET /api/product/:id – отримання даних окремого товару.
  - Перевірялося конкретне відображення інформації по ID.
  - Виправлення: некоректне оброблення неіснуючого ID (поверталося «500 Internal Server Error»). Додано перевірку на наявність товару, реалізовано повернення «404 Not Found».

Короткий опис виявлених помилок та їх виправлення наведено в таблиці 3.1.

Таблиця 3.1 – Виявлені помилки та їх виправлення

Помилка	**Причина**	**Виправлення**
Відсутній JWT токен після реєстрації	Не було налаштовано повернення токена у відповіді	Додано генерацію та повернення токена у «registrationController.js».
Некоректне оновлення списку товарів після додавання	Стан не оновлювався у React	Виправлено логіку «useEffect» для оновлення після запиту.
Неможливість перейти на сторінку товару	Неправильне формування маршруту у React Router	Виправлено шлях з /product/:id на правильний шлях у «App.js».
Некоректна валідація форм	Усі поля приймались порожніми	Додано перевірку на клієнті та сервері.

Висновки до розділу 3

У третьому розділі було детально розглянуто процес реалізації та тестування веб-додатку інтернет-магазину зоотоварів. Розробка серверної частини здійснювалась із використанням середовища Node.js та фреймворку Express.js, що забезпечило ефективну організацію маршрутизації, обробки запитів та взаємодії з базою даних PostgreSQL через ORM-бібліотеку Sequelize. Було виконано повну ініціалізацію проєкту, налаштовано структуру директорій, підключено базу даних, реалізовано API та модулі реєстрації користувачів із забезпеченням необхідних заходів безпеки.

Реалізація серверної частини на базі Node.js із використанням фреймворку Express.js дозволила побудувати гнучку, масштабовану та структуровану архітектуру додатку. Ініціалізація проєкту, налаштування залежностей та створення API забезпечили основу для ефективної взаємодії між клієнтом і сервером. Особливу увагу було приділено правильній організації коду, що є запорукою зрозумілості та підтримуваності системи в майбутньому.

У межах реалізації клієнтської частини застосовано бібліотеку React, що дозволило побудувати динамічний інтерфейс з високою швидкістю реакції на дії користувача. Реалізовано базові функціональні компоненти: сторінку реєстрації та авторизації, головну сторінку з каталогом товарів. Компоненти взаємодіють з бекендом за допомогою HTTP-запитів до API.

Налаштування системи управління базами даних на прикладі PostgreSQL із застосуванням PGAdmin продемонструвало можливості гнучкого адміністрування, створення об'єктів бази, виконання запитів і контролю доступу. Важливим кроком стало підключення бази даних до серверного середовища та впровадження ORM-інструменту Sequelize, що спростило обробку даних на рівні об'єктів та прискорило розробку.

Реалізація механізму реєстрації користувачів забезпечила персоналізований доступ до функцій інтернет-магазину. Було впроваджено перевірку введених даних, безпечне зберігання облікової інформації,

автентифікацію та додаткові функції, зокрема підтвердження електронної адреси. Це значно підвищило рівень безпеки та зручність користування системою.

Таким чином, у ході реалізації третього розділу було створено повнофункціональний веб-додаток, який відповідає вимогам до сучасного інтернет-магазину: із зручною архітектурою, надійною базою даних, безпечною системою реєстрації користувачів та дружнім користувацьким інтерфейсом.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було досягнуто основної мети – розроблено веб-додаток інтернет-магазину зоотоварів, який забезпечує повний цикл взаємодії користувача з платформою.

У першому розділі проведено аналітичний огляд предметної області, визначено основні вимоги до сучасних інтернет-магазинів та досліджено наявні рішення. Було обґрунтовано доцільність створення власного веб-додатку з урахуванням специфіки сфери зоотоварів.

У другому розділі спроєктовано архітектуру додатку, визначено функціональні та нефункціональні вимоги, створено модель бази даних, описано ключові сценарії взаємодії користувача із системою та побудовано діаграми, що відображають логіку роботи проєкту.

У третьому розділі реалізовано серверну та клієнтську частини веб-додатку. Для бекенду використано Node.js та Express.js, а для фронтенду – React. Базу даних побудовано на основі PostgreSQL із застосуванням ORM-бібліотеки Sequelize. Реалізовано основні функціональні модулі: реєстрація/авторизація користувачів, перегляд каталогу товарів, додавання товарів до кошика. Проведено модульне та інтеграційне тестування, яке підтвердило коректність роботи системи.

Результати роботи продемонстрували, що запропоноване рішення є ефективним, масштабованим та зручним для кінцевих користувачів. Додаток відповідає сучасним вимогам до вебсервісів і може бути в подальшому розширений додатковими модулями — такими як оплата онлайн, відгуки, система знижок, особистий кабінет користувача тощо.

Таким чином, поставлені завдання були успішно виконані, а розроблений програмний продукт може використовуватись як основа для повноцінного комерційного інтернет-магазину.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What is Use Case Diagram? Visual-paradigm. URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/ (дата звернення: 20.04.2025).
2. Модель «сутність — зв’язок». Вікіпедія. URL: https://uk.wikipedia.org/wiki/Модель_«сутність_—_зв%27язок» (дата звернення: 20.04.2025).
3. MoSCoW method. Wikipedia. URL: https://en.wikipedia.org/wiki/MoSCoW_method (дата звернення: 20.04.2025).
4. PostgreSQL 17.5 Documentation. postgresql. URL: https://www.postgresql.org/docs/current/ (дата звернення: 21.04.2025).
5. Postman Learning Center. learning.postman. URL: https://learning.postman.com/ (дата звернення: 21.04.2025).
6. Онлайн ресурс Creately. Creately. URL: https://creately.com (дата звернення: 25.04.2025).
7. Онлайн ресурс Figma. Figma. URL: https://www.figma.com (дата звернення: 25.04.2025).
8. UML для бізнес-моделювання. Evergreens. URL: https://evergreens.com.ua/ua/articles/uml-diagrams.html (дата звернення: 26.04.2025).
9. Налаштування Node.js проекта. 8host. URL: https://www.8host.com/blog/nastrojka-proekta-node-js-pri-pomoshhi-typescript/ (дата звернення: 27.04.2025).
10. PGAdmin4guide.Serverspace.URL: https://serverspace.io/support/help/how-to-install-and-configure-pgadmin-4-in-server-mode-on-ubuntu-22-04/ (дата звернення: 29.04.2025).
11. JSON Web Token Introduction. jwt. URL: https://jwt.io/introduction (дата звернення: 30.04.2025).

12. Quick Start – React. react.dev. URL: https://react.dev/learn (дата звернення: 03.05.2025).
13. Адаптивність веб сайту. Webtune. URL: https://webtune.com.ua/statti/internet-marketing/yak-pereviryty-adaptyvnist-za-dopomogoyu-brauzera/ (дата звернення 03.05.2025).

ДОДАТОК А
Файл package.json

```
{
  "name": "client",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/dom": "^10.4.0",
    "@testing-library/jest-dom": "^6.6.3",
    "@testing-library/react": "^16.3.0",
    "@testing-library/user-event": "^13.5.0",
    "axios": "^1.8.4",
    "bootstrap": "^5.3.5",
    "jwt-decode": "^4.0.0",
    "mobx": "^6.13.7",
    "mobx-react-lite": "^4.1.0",
    "react": "^19.1.0",
    "react-bootstrap": "^2.10.9",
    "react-dom": "^19.1.0",
    "react-router-dom": "^7.5.1",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
```

```
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

ДОДАТОК Б
Код клієнтської частини авторизації на сайті

```
import React, {useContext, useState} from 'react';
import {Container, Form} from "react-bootstrap";
import Card from "react-bootstrap/Card";
import Button from "react-bootstrap/Button";
import Row from "react-bootstrap/Row";
import Col from 'react-bootstrap/Col';
import {NavLink, useLocation, useNavigate} from "react-router-dom";
import {LOGIN_ROUTE, REGISTRATION_ROUTE, SHOP_ROUTE} from "../utils/consts";
import {login, registration} from "../http/userAPI";
import {observer} from "mobx-react-lite";
import {Context} from "../index";

const Auth = observer(() => {
    const {user} = useContext(Context)
    const location = useLocation()
    const navigate = useNavigate();
    const isLogin = location.pathname === LOGIN_ROUTE
    const [email, setEmail] = useState('')
    const [password, setPassword] = useState('')

    const click = async () => {
        try {
            let data;
            if (isLogin) {
                data = await login(email, password);
```

```
            } else {
                data = await registration(email, password);
            }
            user.setUser(user)
            user.setIsAuth(true)
            navigate(SHOP_ROUTE)
        } catch (e) {
            alert(e.response.data.message)
        }
    }
    return (
        <Container
            className="d-flex justify-content-center align-items-center"
            style={{height: window.innerHeight - 54}}
        >
            <Card style={{width: 600}} className="p-5">
                <h2 className="m-auto">{isLogin ? 'Авторизація' : "Реєстрація"}</h2>
                <Form className="d-flex flex-column">
                    <Form.Control
                        className="mt-3"
                        placeholder="Введіть ваш email..."
                        value={email}
                        onChange={e => setEmail(e.target.value)}
                    />
                    <Form.Control
                        className="mt-3"
                        placeholder="Введіть ваш пароль..."
                        value={password}
                        onChange={e => setPassword(e.target.value)}
```

```
                type="password"
            />
            <Row className="d-flex justify-content-between mt-3 pl-3 pr-3">
                {isLogin ?
                    <div>
                        Немає аккаунта? <Col><NavLink to={REGISTRATION_ROUTE}>Зареєструватися</NavLink></Col>
                    </div>
                    :
                    <div>
                        Є аккаунт? <NavLink to={LOGIN_ROUTE}>Увійти</NavLink>
                    </div>
                }
                <Col><Button
                    variant={"outline-success"}
                    onClick={click}
                >
                    {isLogin ? 'Увійти' : 'Реєстрація'}
                </Button></Col>
            </Row>
        </Form>
    </Card>
</Container>
);
});
export default Auth;
```

ДОДАТОК В
Код окремої сторінки товару в інтернет-магазині

```
import React, {useEffect, useState} from 'react';
import bigStar from '../assets/bigStar.png'
import {Button, Card, Col, Container, Image, Row} from "react-bootstrap";
import {useParams} from 'react-router-dom'
import {fetchOneProduct} from "../http/productAPI";

const ProductPage = () => {
    const [product, setProduct] = useState({info: []})
    const {id} = useParams()
    useEffect(() => {
        fetchOneProduct(id).then(data => setProduct(data))
    }, [id])

    return (
        <Container className="mt-3">
            <Row>
                <Col md={4}>
                    <Image                    width={300}                    height={300}
src={process.env.REACT_APP_API_URL + product.img}/>
                </Col>
                <Col md={4}>
                    <Row className="d-flex flex-column align-items-center">
                        <h2>{product.name}</h2>
                        <div
                            className="d-flex align-items-center justify-content-center"
                            style={{background: `url(${bigStar}) no-repeat center center`,
width:240, height: 240, backgroundSize: 'cover', fontSize:64}}
```

```
                >
                    {product.rating}
                </div>
            </Row>
        </Col>
        <Col md={4}>
            <Card
                className="d-flex flex-column align-items-center justify-content-around"
                style={{width: 300, height: 300, fontSize: 32, border: '5px solid lightgray'}}
            >
                <h3>Від: {product.price} грн</h3>
                <Button variant={"outline-dark"}>Додати до кошику</Button>
            </Card>
        </Col>
    </Row>
    <Row className="d-flex flex-column m-3">
        <h1>Характеристики</h1>
        {product.info.map((info, index) =>
            <Row key={info.id} style={{background: index % 2 === 0 ? 'lightgray' : 'transparent', padding: 10}}>
                {info.title}: {info.description}
            </Row>
        )}
    </Row>
  </Container>
 );
};
export default ProductPage;
```

ДОДАТОК Г
Код у якому реалізовані моделі бази даних

```
const sequelize = require('../db')
const {DataTypes} = require('sequelize')

const User = sequelize.define('user', {
   id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true},
   email: {type: DataTypes.STRING, unique: true,},
   password: {type: DataTypes.STRING},
   role: {type: DataTypes.STRING, defaultValue: "USER"},
})

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

const BasketProduct = sequelize.define('basket_product', {
   id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true},
})

const Product = sequelize.define('product', {
   id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true},
   name: {type: DataTypes.STRING, unique: true, allowNull: false},
   price: {type: DataTypes.INTEGER, allowNull: false},
   rating: {type: DataTypes.INTEGER, defaultValue: 0},
   img: {type: DataTypes.STRING, allowNull: false},
})

const Type = sequelize.define('type', {
   id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true},
```

```
    name: {type: DataTypes.STRING, unique: true, allowNull: false},
})

const Rating = sequelize.define('rating', {
    id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true},
    rate: {type: DataTypes.INTEGER, allowNull: false},
})

const Brand = sequelize.define('brand', {
    id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true},
    name: {type: DataTypes.STRING, unique: true, allowNull: false},
})

const ProductInfo = sequelize.define('product_info', {
    id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true},
    title: {type: DataTypes.STRING, allowNull: false},
    description: {type: DataTypes.STRING, allowNull: false},
})

const TypeBrand = sequelize.define('type_brand', {
    id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true},
})
User.hasOne(Basket)
Basket.belongsTo(User)

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

Basket.hasMany(BasketProduct)
BasketProduct.belongsTo(Basket)
```

```
Type.hasMany(Product)
Product.belongsTo(Type)

Brand.hasMany(Product)
Product.belongsTo(Brand)

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

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

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

Type.belongsToMany(Brand, {through: TypeBrand })
Brand.belongsToMany(Type, {through: TypeBrand })

module.exports = {
    User,
    Basket,
    BasketProduct,
    Product,
    Type,
    Brand,
    Rating,
    TypeBrand,
    ProductInfo
}
```

ДОДАТОК Д

Код у якому реалізовано серверну частину реєстрації та авторизації

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

const generateJwt = (id, email, role) => {
    return jwt.sign(
        {id, email, role},
        process.env.SECRET_KEY,
        {expiresIn: '24h'}
    )
}

class UserController {
    async registration(req, res, next) {
        const {email, password, role} = req.body
        if (!email || !password) {
            return next(ApiError.badRequest('Некоректний email або password'))
        }
        const candidate = await User.findOne({where: {email}})
        if (candidate) {
            return next(ApiError.badRequest('Користувач з таким email вже існує'))
        }
        const hashPassword = await bcrypt.hash(password, 5)
        const user = await User.create({email, role, password: hashPassword})
        const basket = await Basket.create({userId: user.id})
        const token = generateJwt(user.id, user.email, user.role)
        return res.json({token})
```

```
    }

    async login(req, res, next) {
        const {email, password} = req.body
        const user = await User.findOne({where: {email}})
        if (!user) {
            return next(ApiError.internal('Користувача не знайдено'))
        }
        let comparePassword = bcrypt.compareSync(password, user.password)
        if (!comparePassword) {
            return next(ApiError.internal('Вказано невірний пароль'))
        }
        const token = generateJwt(user.id, user.email, user.role)
        return res.json({token})
    }

    async check(req, res, next) {
        const token = generateJwt(req.user.id, req.user.email, req.user.role)
        return res.json({token})
    }
}

module.exports = new UserController()
```