Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка сайту для дистриб'ютора фруктів з використанням React та Node.is

Виконав(ла): студент(ка) IV курсу, групи СП-41
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

(підпис) Бондар В. А. (прізвище та ініціали)

Керівник (підпис) Стоянов Ю. М. (прізвище та ініціали)

Нормоконтроль (підпис) Стоянов Ю. М. (прізвище та ініціали)

Завідувач кафедри (підпис) Петрик М. Р. (прізвище та ініціали)

Рецензент (підпис) (прізвище та ініціали)

Тернопіль
2025

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет Комп'ютерно інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра Програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Петрик М. Р.
(підпис) (прізвище та ініціали)
« » 2025 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)

за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

студенту Бондар Віталій Андрійович
(прізвище, ім'я, по батькові)

1. Тема роботи «Розробка сайту для дистриб'ютора фруктів з використанням React та Node.is»

Керівник роботи к. т. н., доц. каф. ПІ Стоянов Юрій Миколайович.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «___» _____ 2025 року № _____

2. Термін подання студентом завершеної роботи

3. Вихідні дані до роботи Предметна область, технічне завдання, вимоги та специфікація, програмне рішення

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. Розділ 1. Аналіз предметної області та задач. Розділ 2. Проєктування .
Розділ 3. Безпека життєдіяльності та основи охорони праці. Висновок.
Список використаних джерел. Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Діаграми, знімки екрану з проробленою роботою, ілюстративні зображення, інформативні зображення для доповнення тексту.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці			

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Ознайомлення з завданням до кваліфікаційної роботи		
2	Підбір джерел по темі кваліфікаційної роботи		
3	Опрацювання джерел інформації по темі кваліфікаційної роботи		
4	Розробка архітектури програмного рішення		
5	Ознайомлення з новими технологіями		
6	Розробка на основі опрацьованої інформації		
7	Виконання звіту по проробленій роботі		
8	Виконання завдання до підрозділу «Безпека життєдіяльності»		
9	Виконання завдання до підрозділу «Основи охорони праці»		
10	Оформлення кваліфікаційної роботи		
11	Нормоконтроль		
12	Перевірка на плагіат		
13	Попередній захист кваліфікаційної роботи		
14	Захист кваліфікаційної роботи		

Студент _____ Бондар В. А.
(підпис) (прізвище та ініціали)

Керівник роботи _____ к. т. н., доц. каф. ПІ Стоянов Ю. М.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота бакалавра, виконав Бондар Віталій Андрійович, студент групи СП-41 Тернопільського національного технічного університету імені Івана Пулюя , присвячена розробці сайту для дистриб'ютора фруктів з використанням React та Node.js. Робота має обсяг 85 сторінок, включає 18 рисунків, 3 додатків, та бібліографію з 28 джерел, таблиць 3.

Ключові слова: веб-розробка, React, Node.js, REST API, проєктування бази даних, клієнтська частина, серверна частина, інтерфейс користувача, аутентифікація.

Основною метою кваліфікаційної роботи є розробка та впровадження повнофункціонального веб-застосунку для дистриб'ютора фруктів із використанням сучасних JavaScript-технологій: React для створення клієнтського інтерфейсу та Node.js для серверної частини з реалізацією REST API.

У першому розділі представлено огляд предметної області, аналіз існуючих аналогів, формулювання технічних вимог та обґрунтування вибору технологічного стеку для реалізації системи.

У другому розділі розглянуто архітектуру веб-додатку, модель предметної області, сценарії взаємодії користувача із системою, структуру бази даних, маршрути REST API, а також деталі реалізації клієнтської та серверної частин програмного забезпечення. Також наведено підхід до тестування та забезпечення якості веб-додатку, описано процес перевірки інтерфейсу, відповідей API, а також проведено оцінку стабільності та масштабованості реалізованого рішення.

Результатом роботи є сучасне, зручне у використанні та масштабоване веб-рішення, яке відповідає практичним потребам малого й середнього бізнесу в галузі дистрибуції продукції.

ABSTRACT

Bachelor's qualification work completed by Vitalii Andriiovych Bondar, a student of group SP-41 at Ternopil National Technical University named after Ivan Puluj, is dedicated to the development of a website for a fruit distributor using React and Node.js. The work consists of 85 pages, includes 18 figures, 3 appendices, 3 tables, and a bibliography of 28 sources.

Keywords: web development, React, Node.js, REST API, database design, frontend, backend, user interface, authentication.

The main purpose of this qualification work is the development and implementation of a full-stack web application for a fruit distributor, using modern JavaScript technologies such as React for the frontend and Node.js for the backend, along with a RESTful API.

The first section provides an overview of the subject area, existing analogs, technical requirements, and justification of the technology stack used for implementation. Also presents the architecture of the system, the domain model, interaction scenarios, database structure, REST API endpoints, and implementation details of both client and server parts of the application.

The third section focuses on testing and quality assurance of the developed system, including interface testing, API response validation, and analysis of system reliability and extensibility.

The result of the work is a modern, scalable, and user-friendly web solution that meets the practical needs of small and medium-sized businesses in the field of product distribution.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

API – Application Programming Interface (програмний інтерфейс прикладного програмування)

CI/CD – Continuous Integration / Continuous Delivery (безперервна інтеграція / безперервне постачання)

REST – Representational State Transfer (архітектурний стиль веб-сервісів)

HTTP – HyperText Transfer Protocol (протокол передавання гіпертексту)

HTTPS – HyperText Transfer Protocol Secure (захищений протокол передавання гіпертексту)

JWT – JSON Web Token (веб-токен для аутентифікації)

JSON – JavaScript Object Notation (формат обміну даними)

SQL – Structured Query Language (структурована мова запитів до бази даних)

ORM – Object-Relational Mapping (об’єктно-реляційне відображення)

UI – User Interface (користувацький інтерфейс)

UX – User Experience (користувацький досвід)

SPA – Single Page Application (односторінковий веб-застосунок)

DBMS – Database Management System (система управління базами даних)

MVC – Model-View-Controller (шаблон архітектури програмного забезпечення)

IDE – Integrated Development Environment (інтегроване середовище розробки)

VS Code – Visual Studio Code (редактор коду)

ESLint – ECMAScript Linter (інструмент перевірки якості коду JavaScript)

Postman – інструмент тестування REST API

UML – Unified Modeling Language (уніфікована мова моделювання)

ВСТУП

У сучасних умовах цифрової трансформації бізнесу дедалі більшого значення набуває використання веб-технологій для автоматизації процесів торгівлі та розширення присутності підприємств в онлайн-просторі. Особливо актуальним є впровадження інформаційних систем у сфері дистрибуції продуктів харчування, де швидкість, зручність та точність мають критичне значення.

Метою даної роботи є розробка повнофункціонального веб-застосунку для дистриб'ютора фруктів, який дозволить автоматизувати процеси обробки замовлень, управління товарним асортиментом, обліку клієнтів та комунікації з постачальниками. У процесі розробки було використано сучасні інструменти веб-програмування — React для реалізації інтерфейсу користувача та Node.js у зв'язці з Express для побудови серверної частини та REST API.

Актуальність теми полягає в потребі малого та середнього бізнесу в доступних, масштабованих і зручних веб-рішеннях, які можуть бути швидко адаптовані під конкретну предметну область. Обрані технології дозволяють створити продуктивний, безпечний та кросплатформний веб-додаток, який може працювати як на настільних, так і на мобільних пристроях.

Метою цієї роботи є розробка веб-сайту для дистриб'ютора фруктів, що дозволяє автоматизувати процеси замовлення, перегляду товарного каталогу, обробки оплат, реєстрації користувачів та адміністрування продукції. Об'єктом дослідження є процес розробки веб-застосунків з використанням сучасних JavaScript-технологій.

Робота включає етапи аналізу вимог, проєктування архітектури, реалізації клієнтської та серверної частин, побудови бази даних, тестування системи та оцінки її ефективності. Результатом є веб-застосунок, який відповідає сучасним вимогам до функціональності, зручності та надійності в умовах дистрибуції продукції.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Стан проблеми автоматизації

У світі, де динаміка ринків визначає успіх бізнесу, ефективне управління логістикою та продажами є ключовим для дистриб'юторів фруктів. Традиційні методи ведення обліку, комунікації з постачальниками та покупцями, а також обробки замовлень часто стикаються з низкою проблем. Це можуть бути помилки, спричинені людським фактором, затримки в обміні інформацією, складнощі з відстеженням асортименту та залишків, а також обмежені можливості для аналізу ринку та прийняття швидких рішень. Відсутність інтегрованих цифрових рішень призводить до зниження операційної ефективності, втрати потенційних клієнтів і, як наслідок, до зменшення прибутку. Саме тому діджиталізація та автоматизація цих процесів стають не просто бажаними, а критично важливими для підтримки конкурентоспроможності та розвитку компаній у сфері дистрибуції свіжих продуктів [1].

У сучасному бізнес-середовищі, особливо в такій динамічній галузі, як дистрибуція фруктів, питання автоматизації та діджиталізації виходять на перший план. Традиційні підходи до управління замовленнями, обліку товарів та взаємодії з клієнтами та постачальниками часто виявляються неефективними. Це призводить до значних втрат часу, людських помилок, затримок у комунікації та, як наслідок, до втрати конкурентних переваг.

Відсутність інтегрованих цифрових рішень не дозволяє повною мірою відстежувати асортимент, оперативно реагувати на зміни попиту та пропозиції, що є критично важливим для сегмента свіжих продуктів. Саме тому розробка сучасного веб-сайту, що автоматизує ключові бізнес-процеси, стає нагальною потребою. Для вирішення цих завдань ми звертаємося до передових веб-технологій. Серед широкого спектру доступних інструментів, особливу увагу заслуговують React та Node.js. React, як бібліотека для побудови користувацьких інтерфейсів, дозволяє створити динамічний, інтерактивний та інтуїтивно

зрозумілий фронтенд, що є вкрай важливим для зручності як адміністраторів, так і клієнтів. Його компонентна архітектура забезпечує високу швидкість розробки та легкість масштабування. Node.js, що працює на стороні сервера, дає можливість використовувати JavaScript по всій архітектурі додатка, уніфікуючи мову програмування та спрощуючи процес розробки. Це забезпечує високу продуктивність та ефективну обробку численних запитів, що є життєво важливим для електронної комерції. Ми ретельно вивчили існуючі рішення на ринку – веб-сайти конкурентів та спеціалізовані B2B платформи для дистрибуції фруктів. Цей аналіз дозволив виявити сильні та слабкі сторони поточних пропозицій, а також сформувати уявлення про ключовий функціонал, який користувачі очікують від подібних систем [2, 3].

Вивчення цих практик є основою для формування унікальної пропозиції нашого програмного продукту, що інтегруватиме найкращі аспекти та уникне типових недоліків. Вибір саме React та Node.js для реалізації цього проєкту є обґрунтованим, адже ці технології забезпечують необхідну гнучкість, швидкість розробки та можливість подальшого масштабування, що є критично важливим для довгострокового успіху. Вони дозволяють ефективно керувати даними, забезпечувати швидку взаємодію між користувацьким інтерфейсом та серверною частиною, що необхідно для оперативного оновлення інформації про наявність фруктів та обробку замовлень.

Технічна реалізація включатиме розробку надійної архітектури бази даних, що може бути реляційною (наприклад, PostgreSQL) або NoSQL (наприклад, MongoDB), залежно від специфіки зберігання даних про фрукти, постачальників, замовлення та користувачів. Також буде розроблено ефективне RESTful API, що забезпечить безпечну та швидку взаємодію між фронтендом та бекендом, дозволяючи обробляти запити, пов'язані з переглядом каталогу, додаванням товарів до кошика, оформленням замовлень та управлінням обліковими записами. Усі ці аспекти ляжуть в основу створення функціонального та надійного сайту для дистриб'ютора фруктів [4].

1.2 Огляд сучасних технологій

У сучасному світі, де цифрова присутність є фундаментальною для будь-якого успішного бізнесу, вибір правильних технологій для розробки веб-додатків стає ключовим елементом. Ринок IT пропонує безліч інструментів та фреймворків, кожен з яких має свої переваги та особливості. Для створення функціонального, масштабованого та зручного сайту, як от для дистриб'ютора фруктів, ми маємо враховувати як фронтенд-частину, що взаємодіє безпосередньо з користувачем, так і бекенд-частину, яка відповідає за логіку, обробку даних та взаємодію з базами даних [5].

Серед фронтенд-технологій, що формують користувацький інтерфейс, безумовними лідерами є JavaScript-бібліотеки та фреймворки. React, розроблений Facebook, виділяється своєю компонентною архітектурою, яка дозволяє розробникам створювати складні, інтерактивні та легко підтримувані користувацькі інтерфейси. Це означає, що ми можемо будувати наш сайт з невеликих, незалежних "будівельних блоків", що спрощує розробку та дозволяє повторно використовувати код. React відомий своєю високою продуктивністю завдяки використанню "віртуального DOM", що оптимізує оновлення інтерфейсу. Поряд з React, існують також такі потужні фреймворки, як Angular (від Google), який пропонує більш структурований підхід з використанням TypeScript та великий набір вбудованих інструментів, та Vue.js, що часто обирають за його легкість у вивченні та гнучкість, поєднуючи кращі риси React та Angular. Кожен з них має свої сценарії оптимального застосування, але для нашого проєкту React є особливо привабливим завдяки його гнучкості та великій спільноті.

Що стосується бекенду, тобто тієї частини додатку, що "живе" на сервері та обробляє запити, зберігає дані та реалізує бізнес-логіку, тут також існує широкий вибір. Node.js, що є середовищем виконання JavaScript на стороні сервера, здобув величезну популярність. Його ключова перевага полягає в асинхронній, подієво-орієнтованій архітектурі, що дозволяє обробляти велику кількість одночасних

запитів без значного навантаження на систему. Це ідеально підходить для електронної комерції, де важлива швидка реакція на дії користувача, будь то додавання товару до кошика чи оформлення замовлення. Крім того, використання JavaScript як на фронтенді, так і на бекенді, значно спрощує розробку, оскільки команда може ефективно обмінюватися знаннями та кодом, а також мінімізувати необхідність вивчення різних мов програмування. Альтернативами Node.js є такі технології, як Python з фреймворками Django та Flask, що відомі своєю простотою та швидкістю розробки, особливо для проєктів, що вимагають обробки даних; Ruby on Rails, який пропонує підхід "конвенція понад конфігурацією" для швидкого створення повноцінних веб-додатків; або ж PHP з Laravel/Symfony, що досі є одним з найпоширеніших виборів для веб-розробки. Кожен з цих бекенд-стеків має свої ніші, але для нашого конкретного завдання, що вимагає високої інтерактивності та швидкої розробки, комбінація React та Node.js видається найбільш перспективною та ефективною [6-8].

1.3 Порівняльний аналіз існуючих

Перш ніж приступити до безпосередньої розробки власного веб-сайту для дистриб'ютора фруктів, вкрай важливо провести ретельний аналіз існуючих рішень на ринку. Це дозволить не тільки зрозуміти поточні тенденції та стандарти галузі, але й виявити потенційні ніші та можливості для нашого майбутнього продукту. Ми розглянемо два основні типи онлайн-платформ: веб-сайти прямих конкурентів – інших дистриб'юторів фруктів – та більш широкі B2B (business-to-business) платформи, які можуть включати розділи з фруктами або пропонувати інструменти, корисні для оптової торгівлі [9].

Аналізуючи веб-сайти конкурентів, ми звертаємо увагу на декілька ключових аспектів. По-перше, це дизайн та зручність користування (юзабіліті). Наскільки легко користувачам знаходити потрібну інформацію про асортимент, ціни та умови

доставки? Чи є інтуїтивно зрозуміла навігація, якісні фотографії продукції та зручні фільтри для пошуку? Важливо відзначити, які елементи дизайну здаються найбільш привабливими та функціональними. По-друге, ми досліджуємо функціональні можливості сайтів. Чи є на них онлайн-каталог з актуальними цінами та наявністю? Чи підтримується оформлення замовлень через сайт? Чи є особисті кабінети для клієнтів з історією замовлень та збереженими даними? Які способи оплати та доставки пропонуються? Деякі дистриб'ютори можуть мати розширений функціонал, наприклад, можливість складання індивідуальних замовлень, бронювання товарів або інтеграцію з системами логістики. По-третє, ми оцінюємо інформаційний контент. Наскільки повно та якісно представлена інформація про компанію, її переваги, сертифікати якості продукції, умови співпраці для оптових покупців тощо? Чи є розділи з новинами, блог або інші матеріали, що можуть зацікавити потенційних клієнтів?

Розглядаючи B2B платформи, ми фокусуємося на тих, які спеціалізуються на оптовій торгівлі продуктами харчування або мають відповідні категорії фруктів. Ці платформи часто пропонують ширший набір інструментів, таких як системи управління заявками на комерційні пропозиції (RFQ), можливості порівняння цін від різних постачальників, інструменти для ведення переговорів та укладання контрактів. Ми аналізуємо, які функціональні можливості цих платформ можуть бути корисними для дистриб'ютора фруктів, наприклад, інтеграція з CRM-системами, інструменти для управління ланцюгами поставок або функції для відстеження статусу замовлень на різних етапах [10].

Порівняльний аналіз дозволить виявити сильні та слабкі сторони існуючих рішень. Ми зможемо відзначити вдалі реалізації певних функцій, елементи, які користувачі можуть вважати незручними або недостатніми, а також ті аспекти, які взагалі не представлені на ринку і можуть стати унікальною перевагою нашого майбутнього веб-сайту. Наприклад, порівняння може показати, що більшість конкурентів мають застарілий дизайн або незручну систему навігації, що дасть нам можливість створити більш сучасний та інтуїтивно зрозумілий інтерфейс. Або ж

аналіз B2B платформ може виявити відсутність специфічних інструментів, необхідних саме для ефективної дистрибуції швидкопсувних фруктів.

Для проведення комплексного аналізу існуючих веб-рішень у сфері дистрибуції фруктів, ми розглянули декілька ключових гравців ринку – як прямих конкурентів, так і загальні B2B-платформи, що пропонують подібні послуги. Ця таблиця наочно демонструє сильні та слабкі сторони кожної платформи за обраними критеріями, що дозволить нам сформувати унікальну пропозицію для нашого сайту [11].

Таблиця 1.1 - Порівняльна таблиця існуючих веб-рішень для дистрибуції фруктів

Критерій / Платформа	Конкурент ("Фруктова Країна")	B2B Платформа ("AgroTrade Hub")	Наш Майбутній Сайт (React + Node.js)
Дизайн та Інтерфейс	Застарілий, неадаптивний	Мінімалістичний, фокус на функціоналі	Сучасний, інтуїтивний, чистий, адаптивний
Каталог продукції	Простий список, немає фото	Детальні фільтри, багато постачальників	Інтерактивний, фото, фільтри (за видом, походженням, свіжістю)
Функціонал замовлення	Тільки через дзвінок/пошту	Заявка на комерційну пропозицію (RFQ)	Онлайн-кошик, швидке оформлення, історія замовлень
Особистий кабінет	Відсутній	Розширений (управління контактами, звіти)	Розширений (історія, керування адресами, улюблені товари)

Продовження таблиці 1.1

Інформація про наявність/ Ціни	Застаріла інформація	Частково актуальна, залежить від постачальника	Актуальна в реальному часі
Мобільна версія/Додаток	Відсутня	Окремий мобільний додаток	Повністю адаптивний дизайн
Інтеграція з логістикою	Ручне узгодження	Вбудовані логістичні сервіси	Можливість інтеграції з популярними службами доставки
Аналітика та звіти (для адміна)	Немає	Розширена аналітика по ринку	Детальні звіти по продажах, запасах, користувачах
Можливість зворотного зв'язку	Телефон, email	Система тікетів, чат	Онлайн-чат, система сповіщень, оцінки
Безпека даних	Не вказано	Високий рівень (протоколи шифрування)	Високий рівень (SSL, аутентифікація, авторизація)

Пояснення до таблиці:

— Критерії: Обирайте ті аспекти, які є найважливішими для вашого проєкту. Тут наведені загальні, але ви можете додати або видалити їх.

— Конкуренти/Платформи: Заповніть назвами реальних компаній, які є вашими прямими конкурентами або великими B2B-платформами.

— Оцінка: У кожній комірці опишіть, як та чи інша платформа реалізує відповідний критерій, виділяючи як сильні, так і слабкі сторони.

— Наш Майбутній Сайт: Ця колонка має відображати, як ви плануєте реалізувати той чи інший критерій, використовуючи переваги React та Node.js, і як це відрізнятиметься на краще від конкурентів.

Ця таблиця систематизує інформацію та наочно показує, чому наш проєкт буде кращим та матиме унікальні переваги на ринку.

1.4Обґрунтування вибору технологій

Вибір технологічного стеку для розробки веб-сайту дистриб'ютора фруктів є одним із найважливіших етапів, який безпосередньо вплине на швидкість розробки, продуктивність, масштабованість та зручність підтримки майбутнього продукту. Після аналізу предметної області та огляду існуючих рішень ми зупинили свій вибір на комбінації React для розробки клієнтської частини (фронтенду) та Node.js для створення серверної частини (бекенду). Це рішення ґрунтується на ряді вагомих переваг, які ці технології пропонують саме для нашого конкретного проєкту [12].

Перш за все, React зарекомендував себе як потужна та гнучка бібліотека JavaScript для побудови сучасних користувацьких інтерфейсів. Для веб-сайту, де передбачається велика кількість товарів, фільтрів, динамічних оновлень цін та наявності, а також інтерактивних елементів (наприклад, кошик покупок, особистий кабінет), компонентна архітектура React є надзвичайно вигідною. Вона дозволяє розбити складний інтерфейс на невеликі, незалежні та повторно використовувані компоненти. Це значно спрощує процес розробки, полегшує тестування та підтримку коду, а також підвищує його модульність та масштабованість. Крім того, React використовує віртуальний DOM (Document Object Model), що забезпечує високу продуктивність шляхом оптимізації оновлень на сторінці, що є критично важливим для швидкої та плавної роботи користувацького інтерфейсу, особливо при взаємодії з великим обсягом даних про фрукти [13].

З іншого боку, Node.js, як середовище виконання JavaScript на стороні сервера, пропонує ряд не менш важливих переваг. Найбільшою з них є можливість використання однієї мови програмування – JavaScript – як на фронтенді, так і на бекенді. Це значно спрощує комунікацію між розробниками обох частин проєкту, підвищує ефективність команди та зменшує витрати часу на переключення між різними мовами та парадигмами програмування. Асинхронна, подієво-орієнтована архітектура Node.js робить його надзвичайно ефективним при обробці великої кількості одночасних запитів, що є критично важливим для веб-сайту з потенційно великою кількістю користувачів. Це дозволяє серверу залишатися швидким та чуйним навіть під високим навантаженням. Крім того, Node.js має велику та активну спільноту розробників і величезну кількість доступних npm-пакетів (бібліотек та інструментів), що значно прискорює процес розробки та надає готові рішення для багатьох типових завдань, таких як робота з базами даних, автентифікація користувачів, обробка API тощо.

Звісно, існують й інші технології, які можна було б розглянути для реалізації подібного проєкту. Наприклад, для фронтенду можна було б використовувати Angular або Vue.js, а для бекенду – Python з Django/Flask, Ruby on Rails або PHP з Laravel. Однак, зваживши всі "за" і "проти" саме для нашого завдання, комбінація React та Node.js видається найбільш оптимальною. React забезпечить створення сучасного, інтерактивного та продуктивного користувацького інтерфейсу, а Node.js – потужний, масштабований та ефективний бекенд.

Для обґрунтування вибору технологій React та Node.js для розробки веб-сайту дистриб'ютора фруктів, важливо порівняти їх ключові характеристики з альтернативними підходами. Таблиця 1.2 наочно демонструє, чому обраний стек є оптимальним рішенням для досягнення цілей проєкту.

Таблиця 1.2 - Переваги технологічного стеку

Критерій порівняння	React (для Front-end)	Node.js (для Back-end)	Загальні альтернативи (Angular/Vue.js та Python/PHP)
Модульність та компонентність	Висока, завдяки компонентному підходу та спрощує розробку складних інтерфейсів.	Забезпечується архітектурою (модулі, пакети) та організовані API.	Залежить від фреймворку; для бекенду – від фреймворку та мови.
Продуктивність та швидкість	Висока, завдяки віртуальному DOM	Висока, завдяки асинхронній, подієво-орієнтованій архітектурі	Відрізняється в залежності від фреймворку
Єдність мови розробки	JavaScript (фронтенд)	JavaScript (бекенд)	Різні мови для фронтенду (JavaScript) та бекенду (Python, PHP, Ruby)
Швидкість розробки	Швидка, завдяки готовій компонентній базі, широкій екосистемі бібліотек та інструментів.	Швидка, завдяки великій кількості доступних npm-пакетів та простоті написання API.	Може бути швидкою, але часто вимагає більшої "налаштування з нуля"

Продовження таблиці 1.2

Масштабованість	Добре масштабується	Висока, завдяки неблокуючій архітектурі	Може бути масштабованою
Підтримка та спільнота	Велика та активна спільнота, регулярні оновлення	Велика та активна спільнота	Велика спільнота, але може бути більш фрагментованою в залежності від конкретного фреймворку.
Оптимальність для проєкту	Ідеально підходить для динамічного каталогу фруктів, кошика, особистого кабінету.	Ідеально підходить для швидкої обробки замовлень, управління запасами в реальному часі.	Можуть бути придатними, але часто вимагають більших зусиль для досягнення того ж рівня інтерактивності

Щоб чітко обґрунтувати наш вибір технологій React та Node.js як основи для розробки веб-сайту дистриб'ютора фруктів, нами було проведено ретельний аналіз їхніх ключових переваг у порівнянні з іншими поширеними рішеннями на ринку. Представлена нижче таблиця візуалізувала цей порівняльний аналіз, сфокусувавшись на критичних для нашого проєкту критеріях. Вона дозволила наочно продемонструвати, чому саме комбінація React для фронтенду та Node.js для бекенду стала найбільш оптимальним та ефективним рішенням для створення динамічного, високопродуктивного та масштабованого веб-додатку, здатного задовольнити всі вимоги до сучасного комерційного онлайн-ресурсу в сфері дистрибуції свіжих продуктів. Ми розглянули такі аспекти, як модульність, продуктивність, єдність мови розробки, швидкість реалізації, масштабованість, а

також підтримку спільноти, щоб показати, як обраний нами стек повністю відповідає нашим потребам [14].

1.5Технічні аспекти реалізації програмного продукту

Успішна реалізація будь-якого веб-додатку, а тим паче такого, як сайт для дистриб'ютора фруктів, що вимагає постійного обміну даними та забезпечує комерційні операції, неможлива без глибокого розуміння ключових технічних аспектів. Тут на перший план виходять ефективна робота з базами даних та розробка надійного програмного інтерфейсу (API). Ці компоненти є "серцем" системи, що забезпечує зберігання інформації, її обробку та взаємодію між різними частинами додатку [15].

Почнемо з особливостей роботи з базами даних. Для нашого сайту необхідно зберігати велику кількість різноманітної інформації: деталі про фрукти (назва, опис, походження, термін зберігання, ціна, наявність), дані про постачальників, інформацію про користувачів (клієнтів та адміністраторів), історію замовлень, платіжні дані, логістичну інформацію тощо. Вибір типу бази даних та її структури є критично важливим. Ми можемо розглядати як реляційні бази даних, такі як PostgreSQL або MySQL, які відомі своєю надійністю, підтримкою складних запитів та забезпеченням цілісності даних через схеми та відносини між таблицями. Це ідеально підходить для структурованих даних, де зв'язки між сутностями чітко визначені (наприклад, замовлення завжди пов'язане з конкретним клієнтом та набором товарів). Альтернативою є NoSQL бази даних, такі як MongoDB, які пропонують більшу гнучкість у зберіганні даних, особливо коли структура інформації може часто змінюватися або є менш стандартизованою. Це може бути корисно для зберігання логів, інформації про сесії користувачів або документів, що не мають жорсткої схеми. Вибір конкретної СУБД буде залежати від детальних вимог до даних, необхідності їх масштабованості та специфіки взаємодії.

Незалежно від вибору, важливо забезпечити оптимальну структуру даних, індексацію для швидкого пошуку та ефективні механізми бекапу та відновлення (рис. 1.1).

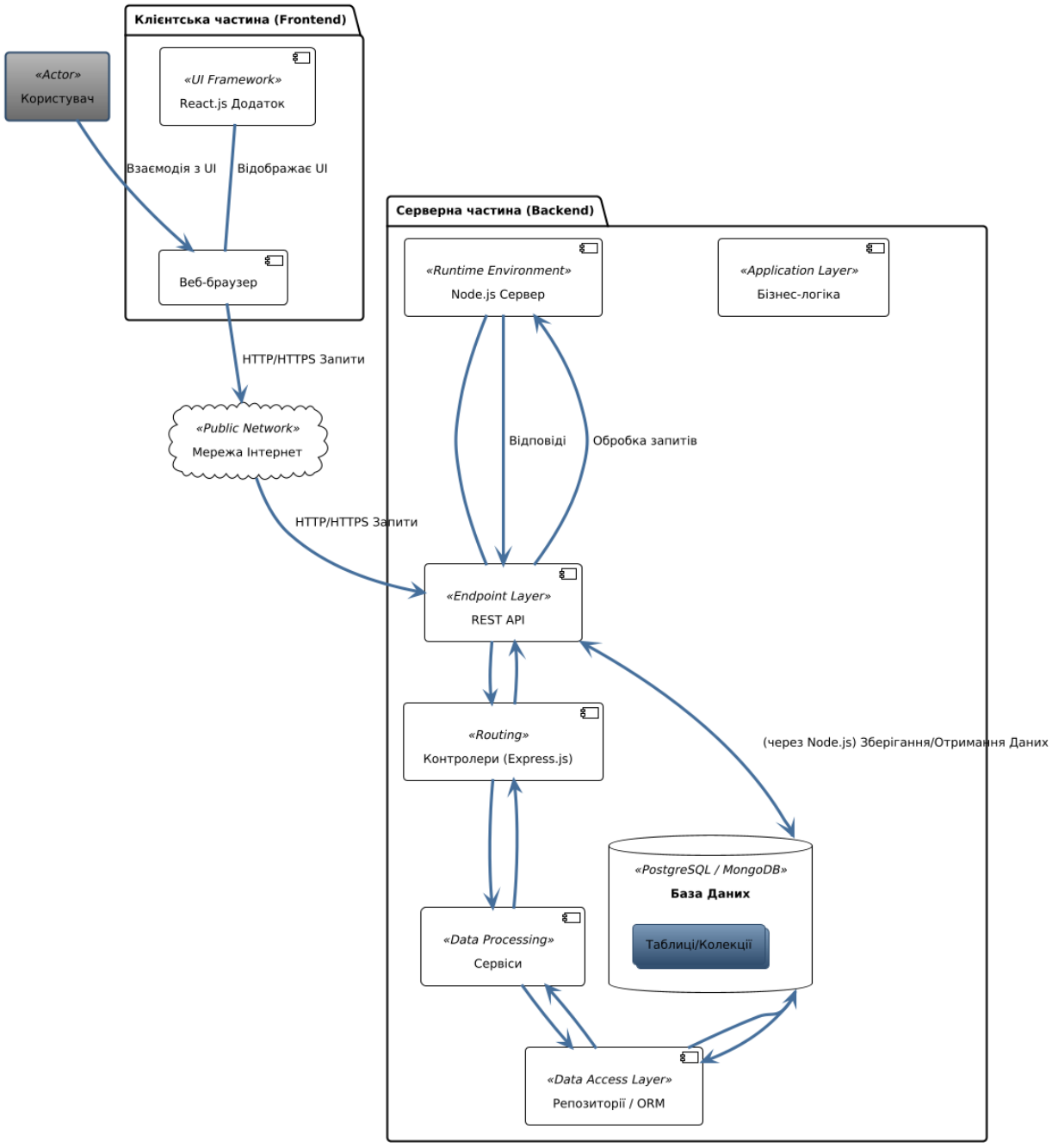

Рисунок 1.1 – Взаємодія компонентів системи

Далі, не менш важливим є розробка API (Application Programming Interface). API – це набір правил та протоколів, який дозволяє різним програмним

компонентам "спілкуватися" один з одним. У нашому випадку, це буде місток між фронтендом (React-додатком, що працює в браузері користувача) та бекендом (Node.js-сервером, що обробляє запити). Найбільш поширеним підходом для таких взаємодій є використання RESTful API. Це означає, що ми розробляємо набір "кінцевих точок" (endpoints), до яких фронтенд може надсилати HTTP-запити (GET для отримання даних, POST для створення, PUT/PATCH для оновлення, DELETE для видалення). Наприклад, запит на /api/fruits повертатиме список доступних фруктів, а запит на /api/orders дозволить оформити нове замовлення. Ключові аспекти розробки API включають: стандартизацію форматів даних (зазвичай JSON), ефективну обробку помилок, забезпечення безпеки (аутентифікація користувачів, авторизація доступу до ресурсів), а також документування API для зручності подальшої розробки та інтеграцій. Правильно спроєктоване API гарантує швидкий та надійний обмін даними, що є основою для функціональності сайту, забезпечуючи, наприклад, миттєве оновлення інформації про наявність товарів або статус замовлення.

2. РОЗРОБКА МОДЕЛІ ТА ПРОГРАМНОГО КОМПЛЕКСУ

Розробка веб-сайту для дистриб’ютора фруктів вимагає системного підходу, що поєднує аналітичне проєктування, сучасні технології веб-розробки та врахування бізнес-вимог. У цьому розділі розглядаються етапи створення програмного продукту: від побудови моделі системи та визначення її основних функцій до реалізації користувацького інтерфейсу, серверної частини та бази даних. Особливу увагу приділено архітектурному рішенню, яке дозволяє досягти масштабованості, зручності підтримки та продуктивності веб-застосунку [16].

Також розглядається процес тестування програмного комплексу з метою перевірки його працездатності, надійності та відповідності функціональним вимогам. Результатом цього розділу є повноцінний веб-додаток, що задовольняє потреби дистриб’ютора фруктів і може бути впроваджений у виробниче середовище.

2.1 Проєктування програмного забезпечення

Проєктування є одним із ключових етапів у створенні ефективного програмного забезпечення, адже саме на цьому етапі визначаються основні компоненти системи, їх функціональність, логіка взаємодії та структура даних. Для забезпечення чіткого та злагодженого функціонування сайту дистриб’ютора фруктів необхідно врахувати як технічні, так і бізнес-вимоги, а також потреби кінцевих користувачів.

У межах цього підрозділу описано функціональні вимоги до системи, побудовано модель предметної області, визначено сценарії використання та взаємодію підсистем. Також подано архітектурне рішення, яке поєднує фронтенд,

бекенд і базу даних в єдину цілісну систему. Результати проєктування стануть основою для подальшої реалізації програмного комплексу.

2.1.1 Опис функціональних вимог до сервісу

Функціональні вимоги визначають поведінку програмного забезпечення, його основні можливості та взаємодію з користувачами. Для сайту дистриб'ютора фруктів, який розробляється з використанням технологій React та Node.js, функціональні вимоги були сформовані з урахуванням потреб бізнесу, вимог до електронної комерції та зручності користувачів [17].

1) Реєстрація та автентифікація користувачів

Користувачі повинні мати можливість створити обліковий запис на сайті, використовуючи email і пароль. Система забезпечує автентифікацію, а також функції авторизації відповідно до ролі користувача (адміністратор або клієнт).

— Створення облікового запису

— Вхід у систему

— Відновлення пароля

— Вихід із системи

2) Каталог продукції

Система має відображати повний каталог фруктів з можливістю фільтрації та сортування:

— Перегляд списку продукції

— Відображення детальної інформації про кожен товар (назва, опис, ціна, наявність, походження, фото)

— Пошук за назвою

— Фільтри за категорією, країною походження, ціною

3) Кошик і замовлення

Користувачі повинні мати можливість формувати кошик із товарів та оформлювати замовлення:

— Додавання/видалення товарів до/з кошика

— Перегляд вмісту кошика

— Зміна кількості обраного товару

— Оформлення замовлення з введенням контактних та платіжних даних

4) Особистий кабінет користувача

Після входу користувач отримує доступ до особистого кабінету, де може:

— Переглядати історію замовлень

— Повторно оформляти попередні замовлення

— Змінювати контактні дані

— Керувати улюбленими товарами

5) Панель адміністратора

Адміністратор має розширені функції керування вмістом сайту:

— Додавання, редагування та видалення товарів

— Оновлення цін та залишків

— Перегляд і обробка замовлень

— Керування зареєстрованими користувачами

— Формування аналітичних звітів

6) REST API для взаємодії між клієнтською та серверною частиною

Для забезпечення динамічної роботи інтерфейсу передбачено реалізацію REST API з такими основними запитами:

— /api/products – отримання списку товарів

— /api/orders – створення замовлення

— /api/users – реєстрація та авторизація користувачів

— /api/admin – доступ до адміністративних функцій

7) Безпека та захист даних

Веб-додаток має забезпечувати:

— Захищене з'єднання (HTTPS)

— Валідацію введених даних на клієнтському та серверному рівні

— Шифрування паролів

— Захист від несанкціонованого доступу через механізми авторизації (JWT або інші)

8) Мобільна адаптивність

Інтерфейс має бути зручним для використання на різних пристроях, включаючи мобільні телефони та планшети.

9) Інтеграція з поштовою службою

Після оформлення замовлення користувач отримує підтвердження на електронну пошту. Адміністратор також може отримувати повідомлення про нові замовлення.

10) Аналітика та звітність (для адміністратора)

Система повинна надавати можливість формування звітів:

— За кількістю замовлень

— За популярністю товарів

— За активністю користувачів

Таким чином, сформульовані вимоги охоплюють основні потреби дистриб'ютора фруктів для онлайн-продажу та адміністрування, забезпечуючи як функціональність для користувачів, так і ефективні інструменти для адміністраторів. Усі ці функції реалізовуватимуться в межах подальших етапів проєкту.

Таблиця 2.1 - Основні функціональні вимоги до веб-сервісу

Функціональна вимога	Короткий опис
Реєстрація та автентифікація	Створення облікового запису, вхід, відновлення пароля
Каталог продукції	Перегляд, пошук, фільтрація та сортування фруктів
Кошик та оформлення замовлення	Додавання товарів до кошика, оформлення та підтвердження замовлення
Особистий кабінет користувача	Перегляд історії замовлень, керування профілем

Продовження таблиці 2.1

Панель адміністратора	Управління товарами, замовленнями, користувачами, генерація звітів
REST API	Обмін даними між фронтендом і сервером через HTTP-запити
Захист та безпека	Шифрування даних, захист сесій, валідація введення
Адаптивний дизайн	Коректне відображення інтерфейсу на мобільних та десктоп-пристроях
Email-сповіщення	Автоматичне надсилання підтвердження замовлень на електронну пошту

2.1.2 Модель предметної області

Модель предметної області описує основні сутності, які існують у системі, та зв’язки між ними. Це дозволяє сформувати структуроване уявлення про об’єкти, з якими працює веб-сайт дистриб’ютора фруктів, і забезпечити логічну узгодженість при розробці програмного забезпечення (рис. 2.1).

Основними сутностями предметної області є:

— Користувач (User) — фізична або юридична особа, яка взаємодіє із сайтом. Користувачі можуть мати різні ролі: клієнт або адміністратор. Атрибути: ім’я, email, пароль, адреса доставки, роль у системі.

— Продукт (Product) — одиниця товару, яку можна переглянути, додати в кошик та придбати. Атрибути: назва, опис, категорія, країна походження, ціна, кількість на складі, фото.

— Замовлення (Order) — сукупність товарів, замовлених користувачем. Атрибути: список товарів, загальна сума, дата оформлення, статус (нове, опрацьовується, доставлено), контактна інформація.

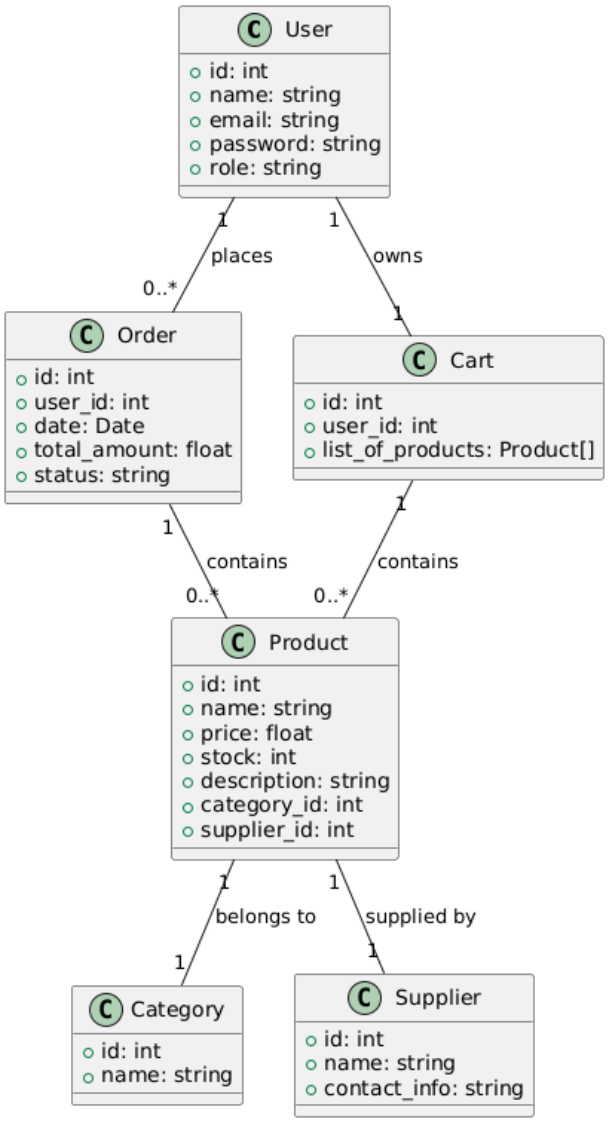

Рисунок 2.1 - Основними сутностями предметної області

— Кошик (Cart) — тимчасове сховище вибраних товарів до моменту оформлення замовлення. Пов’язаний із користувачем. Атрибути: ідентифікатор користувача, перелік продуктів з кількістю.

— Постачальник (Supplier) — юридична особа, яка постачає фрукти на склад. Може бути корисною для адміністративної частини системи. Атрибути: назва компанії, контактна інформація, тип продукції.

— Категорія (Category) — класифікація продукції (наприклад, цитрусові, кісточкові, тропічні). Категорії дозволяють групувати товари для зручної навігації.

Модель предметної області є фундаментом для побудови бази даних, API-запитів, інтерфейсу користувача та логіки взаємодії компонентів. Вона гарантує, що всі частини системи будуть узгодженими та орієнтованими на вирішення ключових завдань, пов’язаних із електронною торгівлею фруктами.

2.1.3 Сценарії використання системи

Сценарії використання системи описують типові послідовності дій користувачів для досягнення конкретних цілей у межах веб-сайту. Вони дозволяють чітко уявити функціональність системи з погляду кінцевого користувача та забезпечують основу для проєктування інтерфейсу, реалізації логіки та тестування. Основні сценарії охоплюють взаємодію як звичайних користувачів (клієнтів), так і адміністратора.

1) Реєстрація нового користувача

Учасник: Новий клієнт

Передумова: Користувач відкрив головну сторінку сайту

Основні кроки:

— Користувач переходить до сторінки реєстрації.

— Заповнює обов’язкові поля: ім’я, email, пароль.

— Система перевіряє унікальність email.

— Після підтвердження система створює обліковий запис.

— Користувач автоматично входить до системи.

2) Оформлення замовлення

Учасник: Авторизований клієнт

Передумова: Користувач увійшов до системи та переглядає каталог

Основні кроки:

— Користувач додає обрані товари до кошика.

— Переходить до кошика та перевіряє вміст.

— Натискає кнопку «Оформити замовлення».

— Вводить контактну інформацію та адресу доставки.

— Підтверджує замовлення.

— Система фіксує замовлення зі статусом «нове» та надсилає підтвердження на email.

3) Робота адміністратора з товарами

Учасник: Адміністратор системи

Передумова: Адміністратор авторизований у системі

Основні кроки:

— Адміністратор переходить до панелі управління товарами.

— Додає новий товар або редагує наявний: змінює ціну, фото, опис.

— Видаляє застарілий товар.

— Система оновлює каталог і відображає зміни клієнтам.

4) Перегляд історії замовлень

Учасник: Авторизований клієнт

Передумова: Користувач здійснив хоча б одне замовлення

Основні кроки:

— Користувач переходить до особистого кабінету.

— Обирає розділ «Мої замовлення».

— Переглядає список попередніх замовлень зі статусами та датами.

— За потреби користувач повторно оформлює одне з попередніх замовлень.

5) Обробка замовлення адміністратором

Учасник: Адміністратор

Передумова: Нове замовлення створене користувачем

Основні кроки:

— Адміністратор відкриває розділ замовлень.

— Переглядає деталі нового замовлення.

— Змінює статус замовлення (наприклад, «опрацьовується» або «відправлено»).

— За потреби зв’язується з клієнтом.

— Система оновлює статус замовлення у профілі користувача.

Сценарії використання не тільки описують функціональні можливості, а й допомагають перевірити, чи система відповідає очікуванням користувачів. Вони є основою для побудови діаграм прецедентів, створення тест-кейсів та розробки UX/UI-рішень (рис. 2.2).

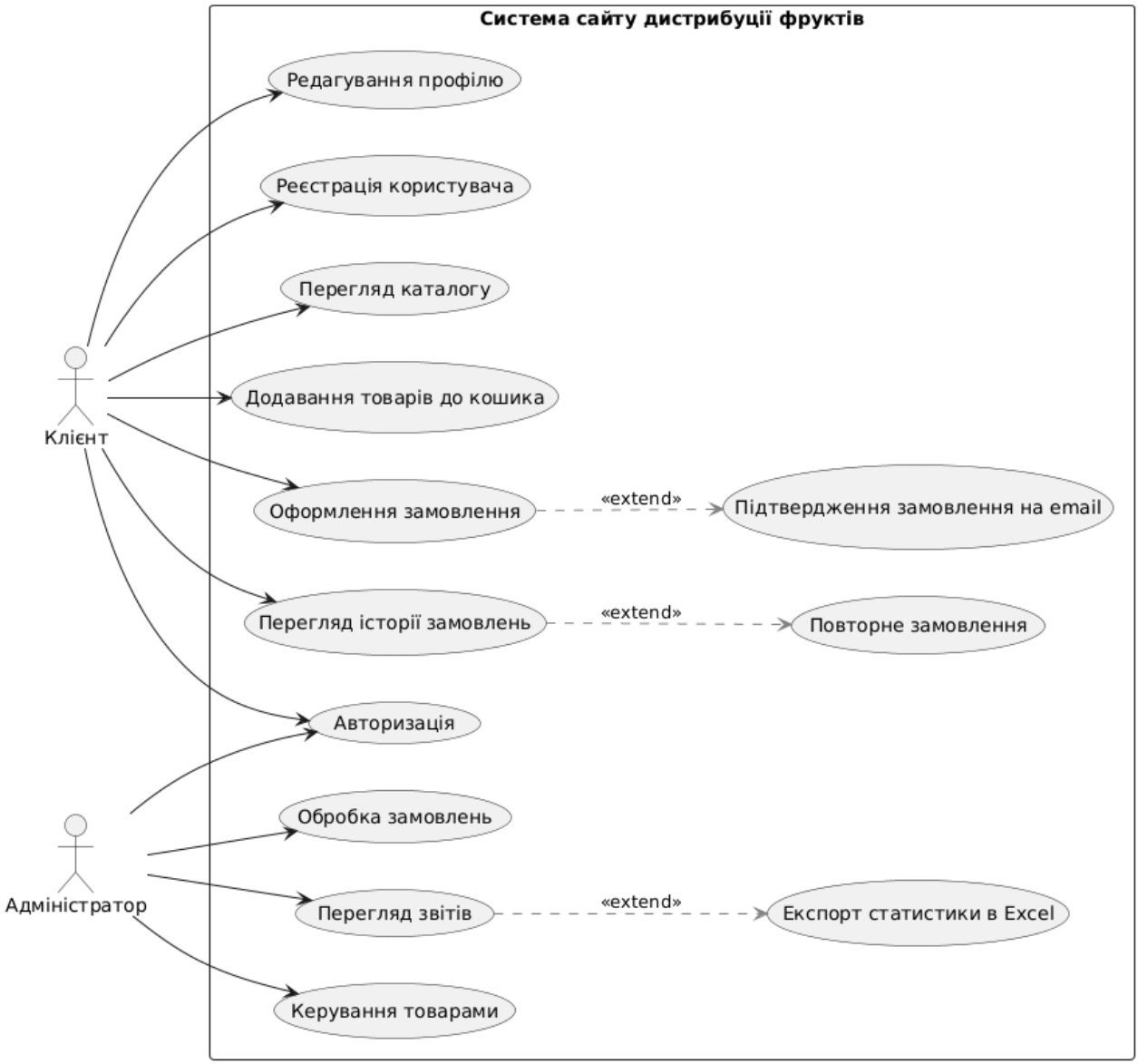

Рисунок 2.2 – Основні варіанти використання

На рисунку 2.3 показано базовий сценарій для оформлення замовлень з фокусом на ключовому потоці (рис. 2.3).

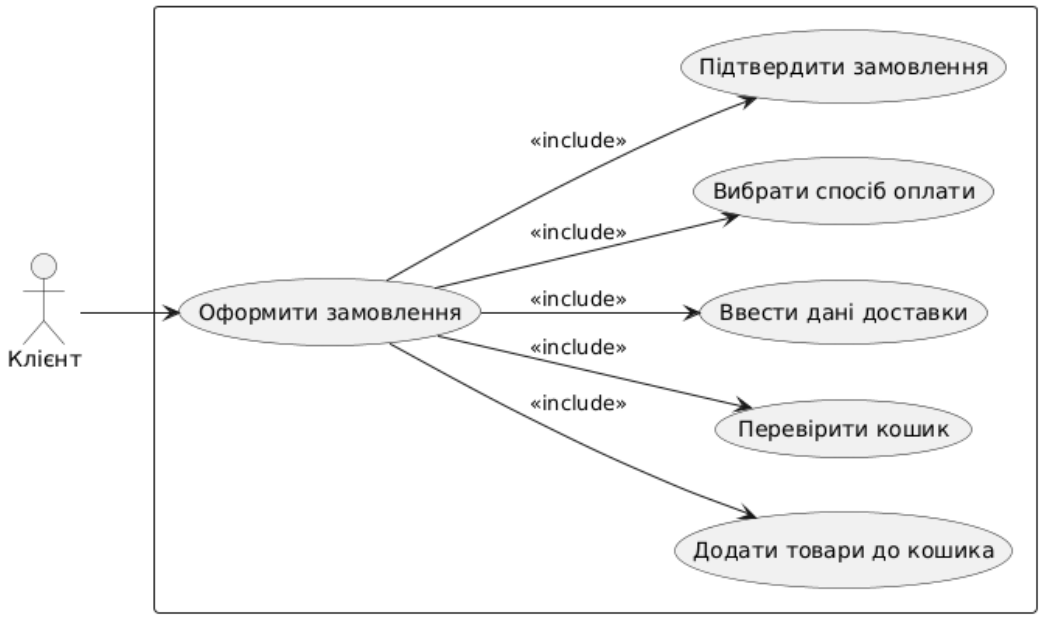

Рисунок 2.3 – Оформлення замовленн

На рисунку 2.4 показано керування товарами з сторони адміністратора. Такий підхід будови демонструє реальну гнучкість системи, коли розширення залежать від ситуації, але основа обов’язкова (рис. 2.4).

Рисунок 2.4 – Керування товарами (адмін)

Даний сценарій має ряд обов’язкових дій та ризширень. Обов’язкові дії:

— Авторизація адміністратора

— Додавання або редагування товару

— Валідація полів при додаванні чи редагуванні

Розширення <<extend>>:

— Видалення товару (не завжди необхідне)

— Оновлення залишків (може виконуватись окремо)

— Завантаження зображення товару — додатковий крок після створення

Варто розглянути ще один варіант розширеної діаграми варіантів використання, цього разу для «Перегляд історії замовлень». Цей сценарій має менше включень і більше розширень, щоб показати опціональну функціональність (рис 2.5).

Рисунок 2.5 – Перегляд історії замовлень (клієнт)

Це діаграма для відображення особистого кабінету користувача і демонструє, як можна дати користувачу додаткові функції без перевантаження основного сценарію. Обов’язкове включення:

— Авторизація (<<include>>) — без неї користувач не має доступу до замовлень.

Розширення <<extend>>:

— Фільтрація за періодом або статусом

— Перегляд деталей конкретного замовлення

— Повторне замовлення (на основі старого)

— Завантаження рахунку або чеку у форматі PDF

2.1.4 Взаємодія компонентів системи

Взаємодія компонентів системи є критично важливою для забезпечення коректного функціонування веб-додатку, який об'єднує клієнтську частину, серверну логіку та базу даних. У рамках архітектури клієнт-сервер кожен компонент виконує чітко визначені функції та взаємодіє з іншими через стандартизовані інтерфейси, зокрема RESTful API.

Основні компоненти системи:

1) Клієнтська частина (React): Відповідає за відображення інтерфейсу користувача, обробку подій, навігацію по сайту та взаємодію з API. Вся логіка рендерингу виконується в браузері користувача.

2) Серверна частина (Node.js + Express): Обробляє запити від клієнта, взаємодіє з базою даних, забезпечує авторизацію, логіку замовлень, валідацію та безпеку.

3) База даних (PostgreSQL або MongoDB): Зберігає інформацію про користувачів, товари, замовлення, залишки продукції, історію покупок тощо. Тип бази даних залежить від обраної моделі зберігання: реляційної чи документної.

4) REST API: Дозволяє клієнтській частині взаємодіяти із сервером. Кожен запит обробляється відповідною маршрутизованою функцією, яка повертає дані у форматі JSON.

Типова послідовність взаємодії компонентів (на прикладі оформлення замовлення):

1) Користувач через інтерфейс (React) додає товари в кошик та натискає «Оформити замовлення».

2) React надсилає POST-запит на серверний маршрут /api/orders з усією необхідною інформацією (товари, контактні дані, спосіб оплати).

3) Node.js на сервері перевіряє автентифікацію користувача, валідність введених даних і наявність товарів на складі.

4) У разі успіху сервер формує замовлення, записує його в базу даних і надсилає відповідь клієнту про успішну операцію.

5) Додатково надсилається email-підтвердження через окремий сервіс (наприклад, Nodemailer).

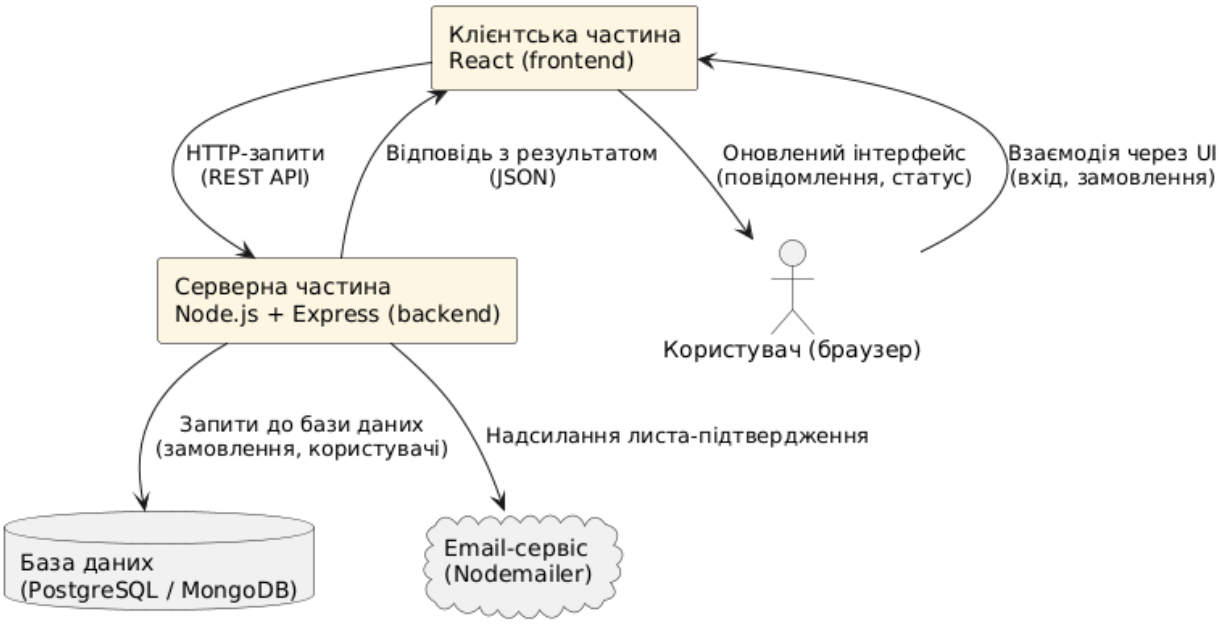

Рисунок 2.6 - Діаграма взаємодії компонентів

Такий розподіл компонентів дозволяє реалізувати гнучку, масштабовану та легко підтримувану архітектуру, у якій можна розвивати окремі частини незалежно — наприклад, змінити дизайн фронтенду без втручання в логіку серверної частини. Крім того, REST API дозволяє надалі інтегрувати сторонні сервіси (аналітику, CRM, мобільний застосунок тощо) без зміни основного ядра системи (рис. 2.6).

Для кращого розуміння динаміки роботи системи розглянемо три основні діаграми послідовності, які демонструють взаємодію між ключовими компонентами: клієнтським інтерфейсом (React), сервером (Node.js) та базою даних (PostgreSQL або MongoDB). Ці діаграми описують логіку виконання найважливіших бізнес-процесів веб-сайту дистриб'ютора фруктів. Для демонстрації динамічної взаємодії між компонентами веб-застосунку використано діаграми послідовності, що відображають типові сценарії роботи користувача з системою. Вони ілюструють, як React, Node.js та база даних узгоджено працюють при виконанні основних операцій.

Перший сценарій охоплює процес реєстрації нового користувача. Після введення даних у форму на клієнтській частині (React), відбувається надсилання POST-запиту до сервера. Node.js перевіряє унікальність email у базі даних, створює обліковий запис і повертає клієнту підтвердження успішної реєстрації (рис. 2.7).

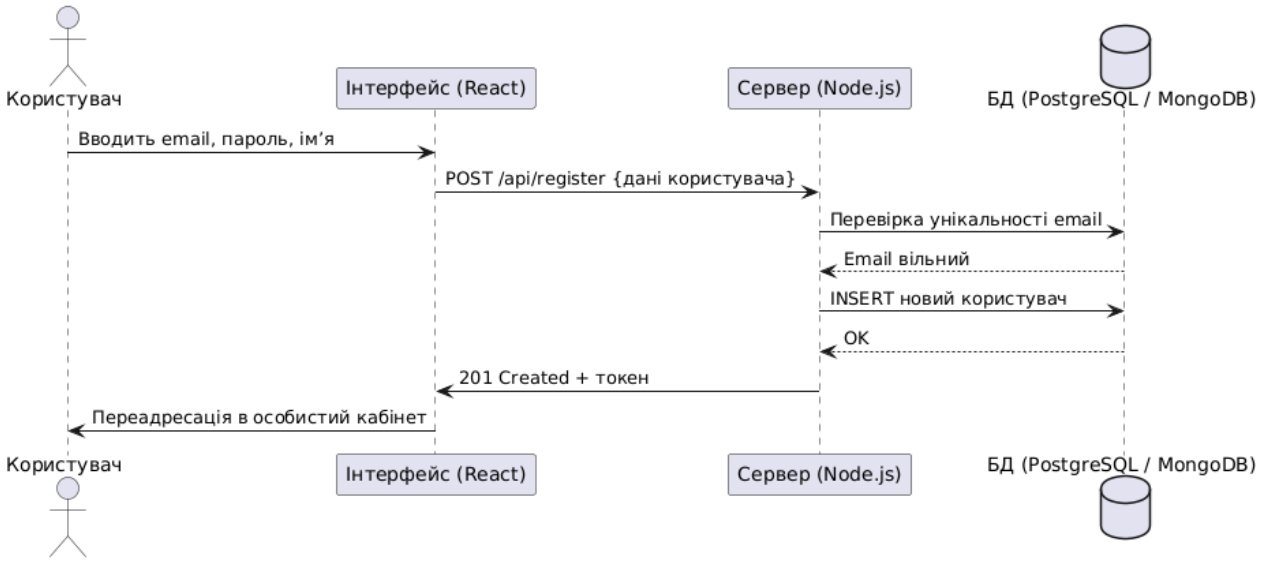

Рисунок 2.7 - Послідовність дій при реєстрації користувача

У другому випадку розглянуто процес оформлення замовлення. Користувач додає товари в кошик, після чого надсилає запит на сервер із деталями замовлення. Сервер створює запис у базі даних, оновлює залишки товару та надсилає повідомлення підтвердження на email (рис. 2.8).

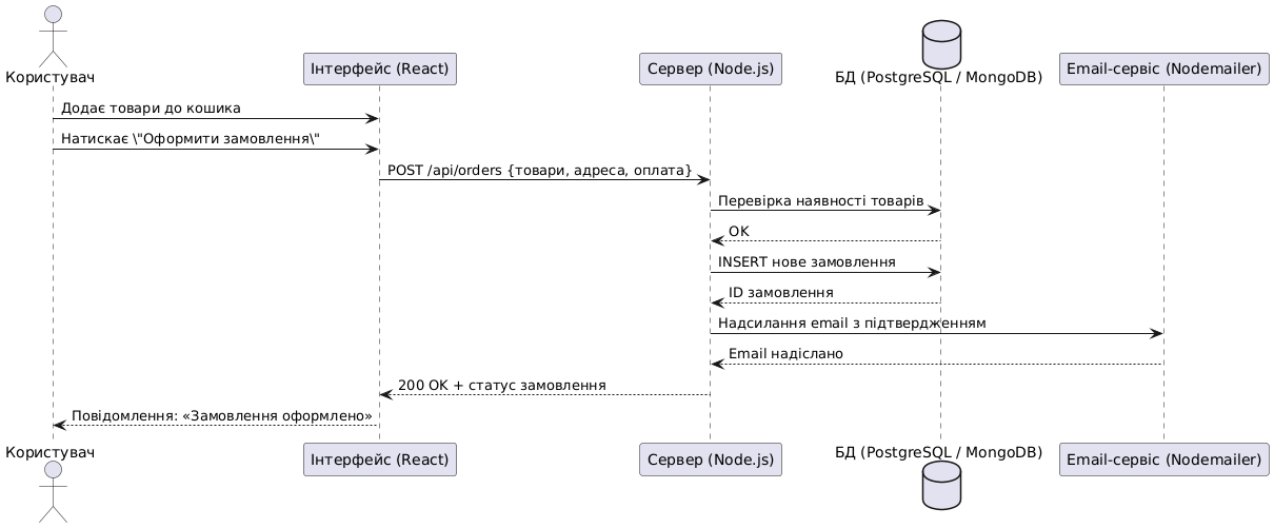

Рисунок 2.8 - Оформлення замовлення

Третя діаграма відображає сценарій редагування товару адміністратором. Після авторизації адміністратор відкриває інтерфейс редагування, змінює характеристики товару, і ці зміни фіксуються у базі даних. Реакція системи відбувається в режимі реального часу з оновленням інтерфейсу (рис. 2.9).

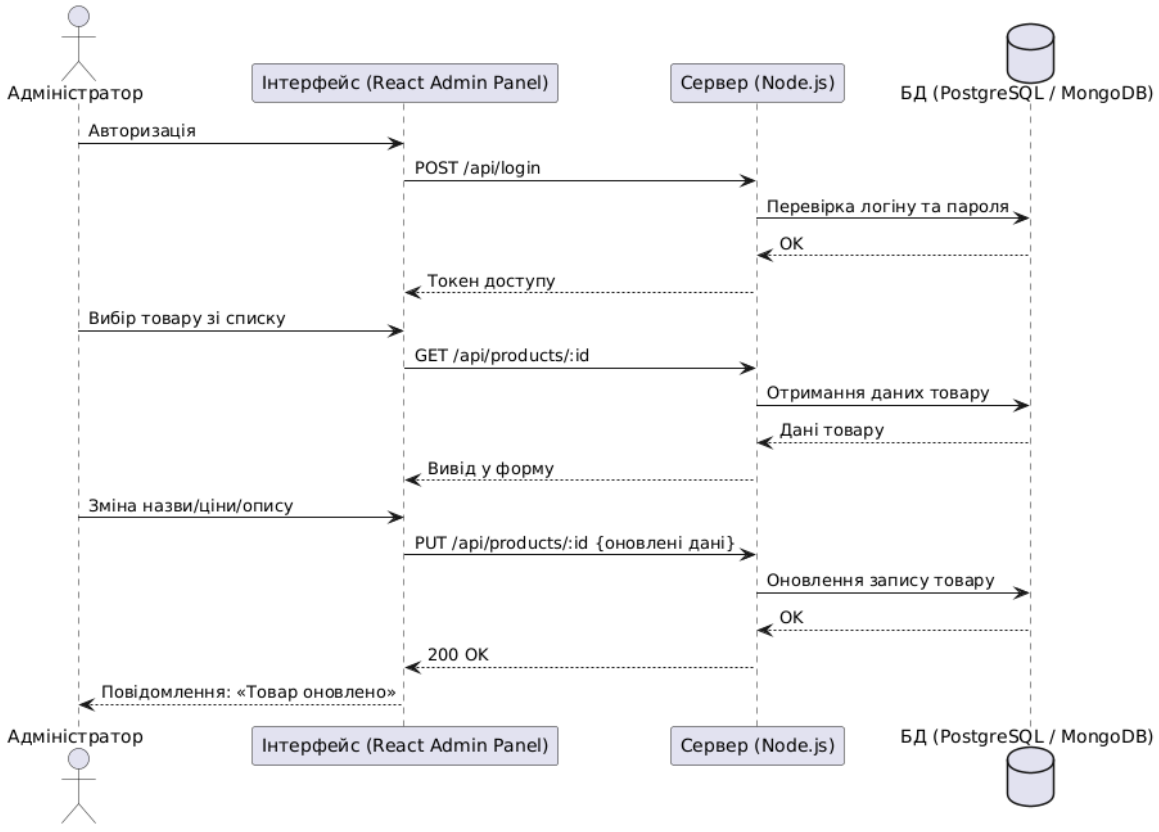

Рисунок 2.9 - Оформлення замовлення

У результаті проведеної роботи було розроблено функціональний веб-сайт для дистриб’ютора фруктів із використанням сучасних технологій React та Node.js. Реалізоване архітектурне рішення забезпечує чітку взаємодію між клієнтською та серверною частинами, підтримує основні бізнес-процеси — від реєстрації користувачів до обробки замовлень та адміністрування товарів. Система має зручний інтерфейс, масштабовану структуру та відповідає сучасним вимогам до електронної комерції, що дозволяє ефективно автоматизувати дистрибуцію свіжої продукції.

2.1.5 Архітектурне рішення веб-додатку

У межах архітектурного проєктування було створено діаграму класів, яка відображає логічну модель взаємозв’язків між основними компонентами системи. Хоча основна частина застосунку реалізується з використанням JavaScript — мови, що не є строго об'єктно-орієнтованою — ми свідомо використали абстрактне подання класів для структурування системи у стилі ООП.

Це дозволяє чітко визначити ролі об’єктів, їхню поведінку та залежності, що є надзвичайно корисним у процесі побудови складного багаторівневого застосунку. Навіть у JavaScript, де замість класів традиційно використовуються функції, об’єкти та модулі, підхід до моделювання системи за допомогою класів допомагає структурувати код, розділити відповідальності та формалізувати логіку (рис. 2.10).

На діаграмі представлені сутності, які умовно поділяються на логічні (наприклад, користувачі, замовлення, товари) та допоміжні (адреси, кошики, ролі, сповіщення тощо). Зв’язки між ними відображають основні типи відносин у системі:

— успадкування (наприклад, Admin і Client успадковують User),

— композиція (наприклад, Order складається з OrderItem),

— агрегація (наприклад, Product пов’язаний із Category або Review),

— асоціація (наприклад, User оформлює Order або має Cart).

Таким чином, створена діаграма не лише формалізує логіку предметної області, а й є надійною основою для подальшого проєктування структури бази даних та реалізації REST API у межах клієнт-серверної архітектури (рис. 2.10).

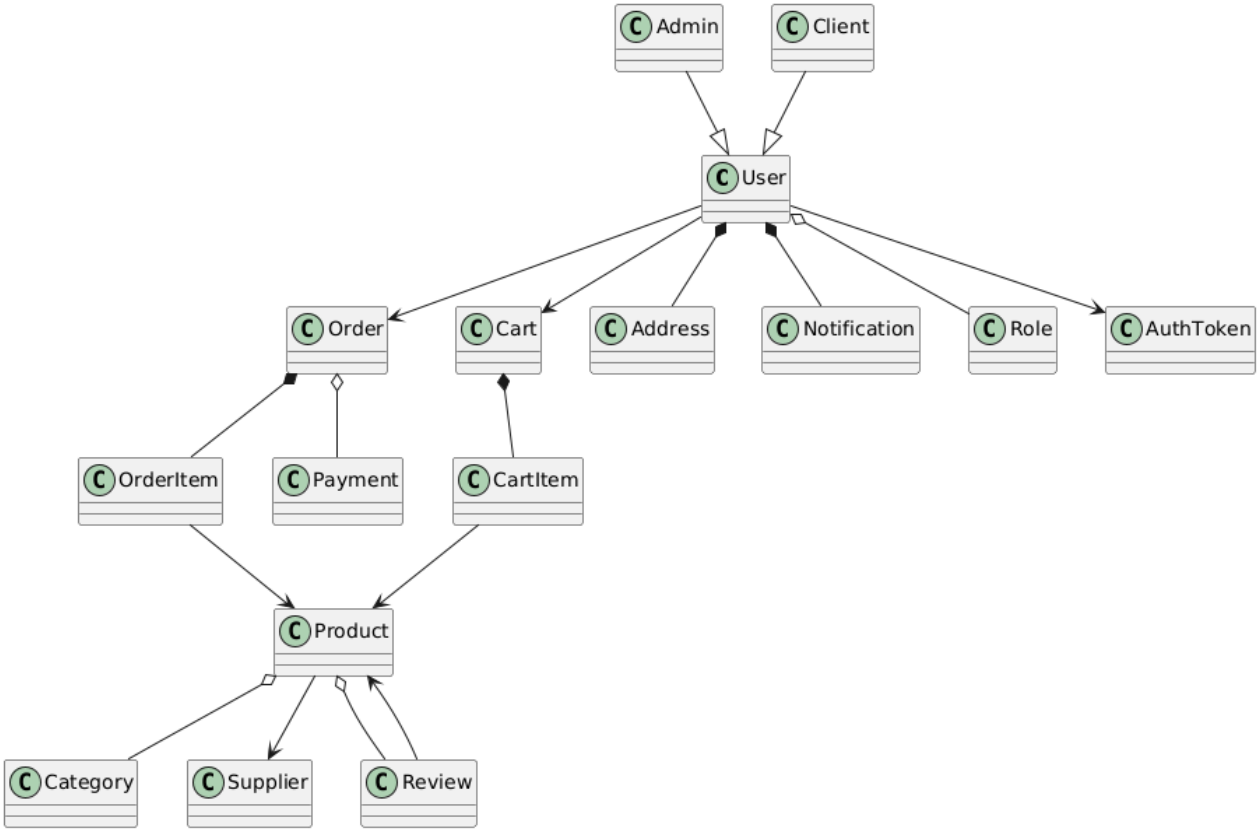

Рисунок 2.10 - Діаграма основних сутностей проєкту

Короткий опис усіх сутностей, представлених на діаграмі класів:

— User – базова сутність, що представляє будь-якого користувача системи. Може мати різні ролі.

— Admin – користувач з розширеними правами для керування товарами, замовленнями, користувачами.

— Client – звичайний зареєстрований користувач, який здійснює покупки через сайт.

— Order – оформлене замовлення клієнта, що містить інформацію про дату, статус, вартість.

— OrderItem – конкретна позиція у замовленні (товар і його кількість).

— Product – товар, що представлений у каталозі: фрукти або інша продукція для замовлення.

— Category – категорія, до якої належить товар (наприклад, «цитрусові», «ягоди»).

— Supplier – постачальник продукції, вказаний для кожного товару.

— Cart – кошик користувача з вибраними товарами до моменту оформлення замовлення.

— CartItem – окрема товарна позиція в кошику.

— Address – адреса доставки, яку користувач зберігає у своєму профілі.

— Payment – інформація про оплату конкретного замовлення.

— Review – відгук, залишений клієнтом до певного товару.

— Notification – повідомлення, що система надсилає користувачеві (наприклад, про зміну статусу замовлення).

— Role – логічна роль користувача в системі (адміністратор, клієнт тощо).

— AuthToken – токен авторизації, який забезпечує доступ користувача до системи після входу.

Ці сутності разом утворюють логічну модель системи та охоплюють як користувацький, так і адміністративний функціонал

Щоб забезпечити надійне зберігання, обробку та цілісність даних у веб-застосунку, необхідно проєктування логічної структури бази даних. Схема бази даних (ERD – Entity-Relationship Diagram) дозволяє наочно представити основні сутності, які зберігаються в системі, та зв’язки між ними.

Побудова структури бази даних базується на попередньо визначеній предметній області та класах, представлених у діаграмі класів. Враховуючи використання реляційної моделі (наприклад, PostgreSQL), кожна сутність представлена окремою таблицею з первинним ключем, а зв’язки реалізуються через зовнішні ключі або зв’язкові таблиці для відношень типу «багато-до-багатьох».

Схема бази даних враховує розмежування ролей користувачів, підтримку замовлень, кошика, історії покупок, платіжної інформації, а також адміністративні можливості для управління товарним каталогом. Така структура дозволяє ефективно реалізувати як клієнтську, так і бекенд-частину системи, забезпечуючи масштабованість, узгодженість і цілісність даних (рис. 2.11).

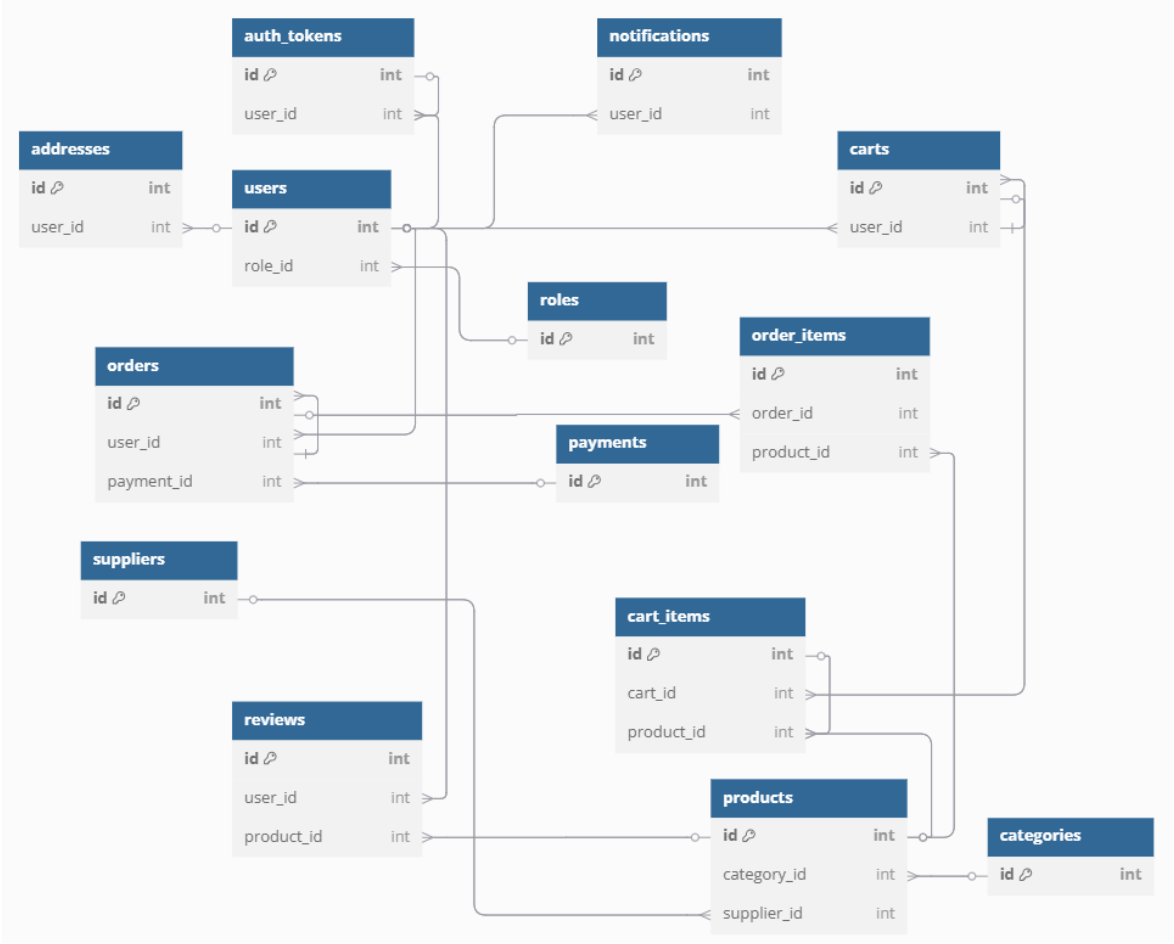

Рисунок 2.11 - Діаграма ER

На схемі бази даних зображено логічну структуру системи веб-сайту дистриб’ютора фруктів. Основу моделі складають таблиці, що відображають ключові сутності предметної області, а також таблиці-зв’язки, які реалізують відношення між ними за допомогою зовнішніх ключів. Уся структура організована відповідно до принципів реляційної моделі даних.

Таблиці та їхнє призначення:

— users — зберігає основну інформацію про користувачів системи: логін, email, пароль, роль тощо.

— roles — містить перелік можливих ролей (наприклад, "клієнт", "адміністратор"), які визначають рівень доступу користувачів.

— auth_tokens — використовується для зберігання токенів доступу, що генеруються після авторизації користувача.

— addresses — зберігає адреси доставки, прив’язані до конкретного користувача.

— orders — представляє оформлені замовлення з боку клієнтів, містить статус, дату, загальну суму.

— order_items — містить окремі товарні позиції в межах одного замовлення: товар, кількість, ціна.

— products — зберігає інформацію про товари: назву, опис, ціну, залишок, категорію, постачальника.

— categories — класифікує товари за категоріями (наприклад, фрукти, ягоди, екзотика).

— suppliers — зберігає інформацію про постачальників продукції (назва, контактні дані).

— carts — представляє кошик користувача з товарами, які ще не оформлені як замовлення.

— cart_items — зберігає перелік товарів у кошику з посиланням на конкретні продукти та їхню кількість.

— payments — фіксує інформацію про платіж, пов’язаний із замовленням: метод оплати, статус тощо.

— reviews — зберігає відгуки користувачів до товарів, включаючи текст, оцінку, дату.

— notifications — зберігає повідомлення, які надсилаються користувачеві: зміна статусу замовлення, нові пропозиції тощо.

Центральною таблицею є users, яка зберігає інформацію про всіх користувачів системи. Кожен користувач пов’язаний з таблицею roles, що дозволяє розмежовувати доступи (наприклад, адміністратор і клієнт). Для реалізації механізму автентифікації використовується таблиця auth_tokens, пов’язана з users.

Для підтримки функціоналу замовлень реалізовано таблиці orders та order_items. Одне замовлення може містити багато позицій, що відображено у зв’язку «один до багатьох» між orders і order_items. Кожна позиція замовлення пов’язана з таблицею products, яка зберігає інформацію про всі доступні товари. Продукти належать до певної категорії (categories) і мають постачальника (suppliers), що дозволяє вести гнучкий облік асортименту.

Для реалізації кошика користувача передбачено таблиці carts і cart_items, які логічно відокремлені від замовлень і дозволяють зберігати поточні обрані товари до моменту покупки. Кошик також прив’язується до конкретного користувача.

Таблиця payments пов’язана з orders і містить інформацію про здійснені оплати. Для зберігання адрес доставки передбачено таблицю addresses, а для реалізації користувацьких відгуків — таблицю reviews.

Таким чином, схема бази даних охоплює повний життєвий цикл користувача: від реєстрації та взаємодії з інтерфейсом до оформлення замовлення, оплати, зворотного зв’язку та адміністрування, забезпечуючи при цьому цілісність, масштабованість і логічну узгодженість з предметною областю.

2.2 Реалізація програмного комплексу

2.2.1 Реалізація клієнтської частини на React

Клієнтська частина веб-застосунку реалізована з використанням бібліотеки React, яка забезпечує побудову компонентного інтерфейсу, що динамічно оновлюється без перезавантаження сторінки. Завдяки використанню React

створено зручне, адаптивне та масштабоване середовище для взаємодії користувачів із системою.

Кожна сторінка застосунку реалізована у вигляді окремого функціонального компонента. Для маршрутизації використовується бібліотека React Router, що дозволяє організувати переходи між сторінками без перезавантаження. Комунікація з серверною частиною відбувається через HTTP-запити (Axios або Fetch API), які взаємодіють із REST API, розробленим на Node.js.

Загалом було реалізовано такі основні компоненти:

— Сторінка каталогу — дозволяє переглядати товари, фільтрувати їх за категоріями та ціною.

— Кошик — дає змогу переглядати вибрані товари, змінювати їхню кількість або видаляти.

— Оформлення замовлення — включає форму введення контактних даних та підтвердження покупки.

— Особистий кабінет — відображає історію замовлень, дозволяє змінювати профільні дані.

— Адміністративна панель — доступна лише адміністраторам, дає змогу додавати/редагувати товари, переглядати замовлення та статистику.

Інтерфейс є адаптивним і підтримує коректне відображення як на десктопах, так і на мобільних пристроях. Для стилізації інтерфейсу використовувались CSS-модулі або бібліотеки компонентів, такі як Material UI, Bootstrap або TailwindCSS.

На головній сторінці каталогу (рис. 2.12) відображено перелік товарів у вигляді карток. Кожна картка містить зображення продукту, його назву, ціну та кнопку «Add to Cart». Інтерфейс доповнений фільтрами за категорією та ціною. Розміщення елементів є інтуїтивно зрозумілим і оптимізованим для швидкого вибору товару.

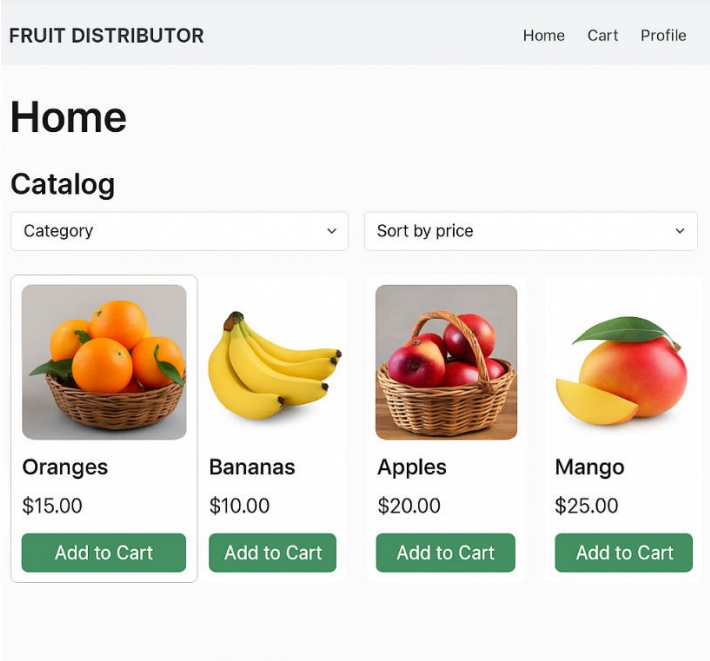

Рисунок 2.12 - Діаграма ER

У вікні кошика користувача (рис. 2.13) наведено список доданих товарів із можливістю змінювати кількість одиниць кожного продукту або видалити позицію повністю. Внизу відображається загальна сума та кнопка для переходу до оформлення замовлення («Checkout»). Це вікно дозволяє користувачеві швидко переглянути вибране та прийняти рішення щодо покупки.

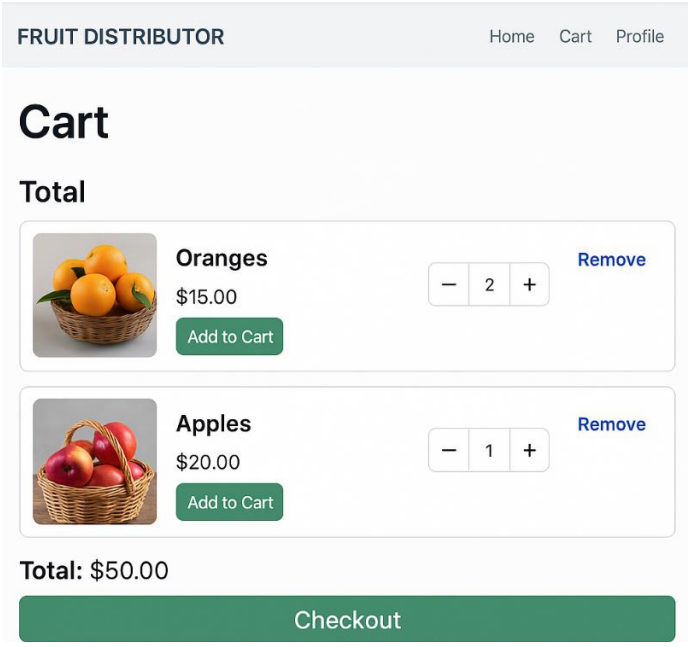

Рисунок 2.13 - Діаграма ER

На сторінці особистого кабінету (рис. 2.14) відображено історію замовлень конкретного користувача. Для кожного замовлення зазначено його номер, дату, перелік товарів та загальну суму. Такий підхід дозволяє клієнту зручно переглядати попередні покупки, повторно замовляти або звіряти статуси.

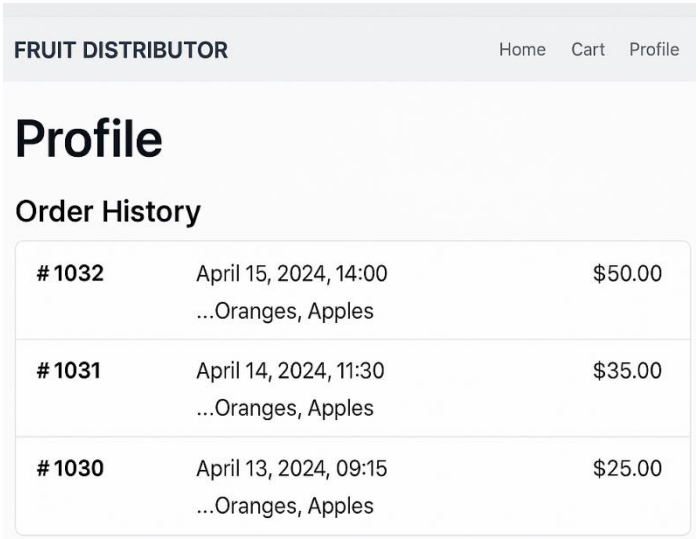

Рисунок 2.14 - Діаграма ER

Таким чином, клієнтська частина веб-застосунку була успішно реалізована з використанням бібліотеки React. Завдяки компонентному підходу вдалося створити гнучкий та підтримуваний інтерфейс із розділенням логіки на окремі функціональні модулі. Забезпечено інтеграцію з серверною частиною, реалізовано ключовий функціонал для користувача — перегляд товарів, керування кошиком, оформлення замовлень, перегляд історії покупок.

2.2.2 Розробка серверної частини на Node.js / REST API

Серверна частина веб-застосунку реалізована з використанням середовища Node.js, що дозволяє обробляти велику кількість одночасних запитів у неблокуючому режимі. Для побудови структури серверу було використано

фреймворк Express.js, який забезпечує зручний механізм маршрутизації, обробки запитів та підключення проміжного програмного забезпечення (див. рис. 2.16).

Рисунок 2.15 - Діаграма зв’язків серверної частини

Вся логіка обробки даних реалізована у вигляді REST API, що дозволяє клієнтській частині взаємодіяти з сервером через стандартні HTTP-методи (GET, POST, PUT, DELETE). Кожен маршрут відповідає за обробку певної сутності, такої як користувачі, товари, замовлення, кошик, оплати тощо. API дотримується принципів REST: чітка структура URL, використання статус-кодів HTTP, відокремлення клієнта й сервера, а також безстанова обробка запитів.

Для доступу до бази даних використовувався ORM-інструмент (наприклад, Sequelize або TypeORM) або нативний драйвер для PostgreSQL/MySQL, що

дозволило забезпечити типізований доступ до таблиць та централізоване управління транзакціями.

З метою підвищення безпеки реалізовано механізм автентифікації користувачів за допомогою JWT (JSON Web Token), який дозволяє зберігати дані сесії у вигляді токена, передаваного в заголовках HTTP-запитів. Всі маршрути, що потребують авторизації, захищені middleware-фільтрами (див. рис. 2.16).

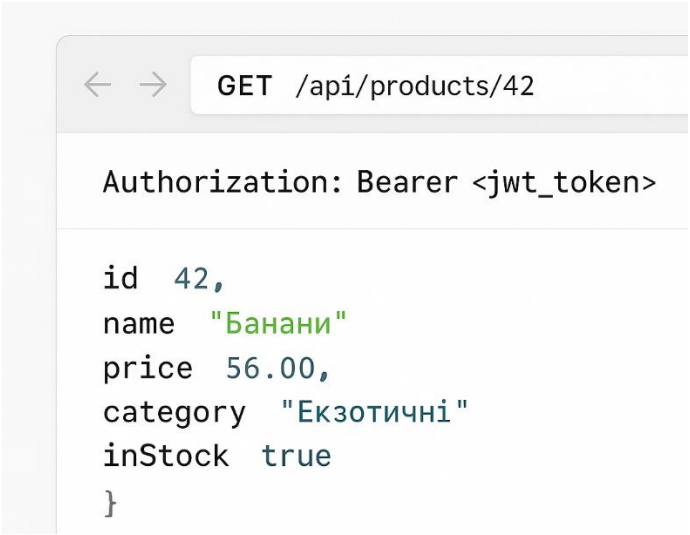

Рисунок 2.16 - REST – запит

Серверна частина є повністю масштабованою, дозволяє додавати нові ендпоінти та модулі без впливу на існуючий функціонал. Така архітектура дає змогу легко розширювати систему, адаптувати її до нових бізнес-процесів і інтегрувати з іншими сервісами.

2.2.3. Тестування та забезпечення якості веб-додатку

Для забезпечення стабільної та надійної роботи веб-додатку було застосовано комплексний підхід до тестування, який охоплює як клієнтську, так і серверну частину. Основна мета тестування — виявлення помилок на ранніх етапах

розробки, перевірка відповідності функціональності до заданих вимог, а також забезпечення зручності та безпечності взаємодії з користувачем.

У процесі розробки було використано такі типи тестування:

— Модульне тестування (unit testing) — перевірка окремих функцій та логічних модулів (наприклад, перевірка валідаторів, утиліт, обробників API).

— Інтеграційне тестування (integration testing) — перевірка взаємодії між модулями, зокрема зв’язків між сервером, базою даних і зовнішніми сервісами.

— Тестування інтерфейсу (UI testing) — ручне або автоматизоване проходження користувацьких сценаріїв на фронтенді.

— Тестування REST API — перевірка правильності відповідей сервера, форматів JSON, статус-кодів та захищеності маршрутів.

— Smoke та regression testing — базова перевірка стабільності після основних змін у коді.

Для автоматизованого тестування використовувалися такі інструменти, як Jest (для Node.js), React Testing Library (для інтерфейсу) та Postman або Insomnia (для перевірки API). Усі критично важливі компоненти пройшли перевірку, і система відповідає вимогам як функціонально, так і з погляду користувацького досвіду. Проведене тестування дозволило зменшити кількість помилок, підвищити стабільність роботи додатку та забезпечити впевнене впровадження у виробниче середовище.

3. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

3.1 Ліквідація наслідків надзвичайних ситуацій

Проблема запобігання виникненню надзвичайних ситуацій техногенного походження та ліквідації їх наслідків є однією з найактуальніших для України. У Кодексі цивільного захисту України, зокрема в статті 20, зазначено, що керівники підприємств, установ та організацій, незалежно від форми власності, зобов'язані забезпечити працівників засобами індивідуального та колективного захисту. Вони також повинні організовувати евакуаційні заходи, формувати сили для ліквідації наслідків надзвичайних ситуацій, підтримувати їх у готовності до дій, а також здійснювати всі необхідні заходи цивільного захисту, беручи на себе відповідні матеріальні та фінансові витрати [18–22].

Сутність рятувальних та інших невідкладних робіт полягає в усуненні загрози життю та здоров'ю людей, у відновленні життєзабезпечення населення і мінімізації матеріальних збитків. Такі роботи спрямовані не лише на негайне реагування, але й на створення умов для подальшого проведення відновлювальних заходів. До їх складу входять дві основні групи: рятувальні роботи та невідкладні технічні дії [23–25].

Рятувальні роботи включають пошук постраждалих осіб, їх звільнення з-під завалів або небезпечних конструкцій, евакуацію населення з зон ураження, а також дії з локалізації небезпечних чинників, таких як пожежі, токсичні хмари чи обвали. Додатково до цього, виконуються роботи з відкриття захисних споруд і надання допомоги людям, які перебувають усередині. Також проводиться санітарна обробка уражених, знезараження особистих речей, техніки та території. У разі потреби залучаються психологічні служби для реабілітації постраждалих осіб.

Невідкладні роботи мають інженерний і технічний характер. Серед них — розчищення проїздів, прокладання тимчасових шляхів, локалізація аварій на комунальних і промислових мережах, тимчасове відновлення роботи водопроводу, електромереж, зв'язку та інших систем, що мають критичне значення для

забезпечення проведення рятувальних дій. У деяких випадках необхідно зміцнити конструкції або навпаки — демонтувати ті елементи, які становлять небезпеку для персоналу.

Описані роботи проводяться у чіткій послідовності. Спочатку виконуються дії з оцінки ситуації, після чого визначаються пріоритети. На наступному етапі відбувається локалізація загроз та безпосереднє рятування людей. Потім переходять до технічних дій, що забезпечують безпечну подальшу роботу та відновлення інфраструктури. Всі дії повинні відповідати нормативним вимогам і методичним рекомендаціям, передбаченим у курсі лекцій та підручниках з дисципліни «Цивільний захист».

Рятувальні та інші невідкладні роботи (РіНР) здійснюються поетапно. Загальна структура поділяється на три основні етапи, кожен з яких має свою мету, зміст дій і очікувані результати.

Перший етап спрямований на екстрений захист населення, зниження впливу наслідків надзвичайної ситуації та підготовку до проведення основних рятувальних заходів. На цьому етапі першочергово організовується оповіщення людей про наявну небезпеку. Населення має бути проінформоване про необхідність використання засобів індивідуального захисту, дотримання правил безпечної поведінки, а також про заплановану евакуацію. Особлива увага приділяється санітарно-гігієнічним заходам, протиепідемічній профілактиці та наданню медичної допомоги. У разі загрози розвитку аварійної ситуації вживаються заходи щодо її локалізації. Також здійснюється зупинка або зміна технологічного процесу на підприємствах, де є ризик ускладнення подій. При виникненні пожеж реалізуються дії з їх гасіння або стримування поширення.

Другий етап пов'язаний з безпосереднім проведенням рятувальних дій у зоні лиха. У цей період проводиться пошук потерпілих, витягування людей з-під завалів, палаючих будинків або пошкоджених транспортних засобів. Якщо існує пряма загроза життю, організовується евакуація із зони ураження до безпечніших місць. Потерпілим надається перша медична допомога. Водночас відбувається санітарна обробка людей, знезараження одягу, майна, техніки та території. У разі

потреби постраждалі евакуюються до лікувальних закладів. Також виконуються інші невідкладні технічні дії, які дозволяють забезпечити безпечні умови для продовження рятувальних робіт.

На третьому етапі увага зосереджується на стабілізації умов життя населення у районах, що постраждали внаслідок надзвичайної ситуації. Проводяться роботи з відновлення або зведення тимчасового житла. Також відновлюються енергетичні мережі, водо- і газопостачання, зв'язок та інші елементи критичної інфраструктури. Налагоджується робота медичних пунктів, забезпечується постачання продуктів харчування та предметів першої необхідності. У місцях проживання людей здійснюється знезараження їжі, води, фуражу, техніки, майна та територій. Для відновлення психологічного стану населення залучаються спеціалісти соціально-психологічної реабілітації. У разі наявності пошкодженого майна, забезпечується розрахунок і відшкодування завданих збитків.

Варто зазначити, що виконання безпосередніх відновлювальних робіт не входить до функціональних обов'язків підрозділів цивільного захисту. Їх виконують спеціально створені відновлювальні бригади, які підпорядковуються відповідним органам влади або керівництву підприємств. Обсяг та форма залучення сил залежить від рівня надзвичайної ситуації. Якщо подія має загальнодержавне, регіональне, місцеве чи об'єктове значення, до проведення рятувальних і невідкладних робіт залучаються відповідні сили цивільного захисту — центрального, регіонального або локального підпорядкування.

3.2 Особливості заходів електробезпеки на підприємствах.

Заходи електробезпеки на підприємствах при роботі з персональними комп'ютерами в умовах використання комп'ютеризованої системи моделювання та документування мережевої інфраструктури дата-центру спрямовані на створення

безпечних умов праці. Їх мета полягає у зменшенні ризиків, пов’язаних із ураженням електричним струмом, електростатичним розрядом, перегрівом обладнання, перевантаженням мережі, коротким замиканням і можливими пожежами.

Усі пристрої, що підключаються до електромережі, повинні бути правильно заземлені. Заземлення – це підключення металевих корпусів або інших струмопровідних частин обладнання до електричної мережі таким чином, щоб у разі витоку струму цей струм прямував у землю, минаючи тіло людини. Це забезпечує захист працівників від ураження струмом і водночас знижує ризик пошкодження обладнання через накопичення статичного заряду [26].

Комп’ютери потребують стабільного джерела живлення. З цією метою використовуються стабілізатори напруги або джерела безперебійного живлення (UPS). Такі пристрої регулюють напругу в мережі й захищають ПК від перепадів струму. Вони також дозволяють завершити роботу системи в разі аварійного вимкнення електроенергії, що запобігає втраті даних [27].

До електробезпеки також належить перевірка справності кабелів та розеток. Кабелі повинні бути цілими, без тріщин, заломів чи оголених проводів. Їх не слід прокладати у місцях з високою прохідністю, щоб уникнути випадкового пошкодження. Розетки встановлюються на безпечній відстані від джерел вологи та мають бути заземлені. Це зменшує ймовірність короткого замикання та пожежі.

Робоче середовище для комп’ютерів повинно бути добре вентильованим. Персональні комп’ютери генерують тепло під час роботи, тому необхідно забезпечити вільну циркуляцію повітря навколо системних блоків. Для цього використовуються вентиляційні отвори, охолоджувальні вентилятори або активні системи охолодження. Перегрів електронних компонентів може спричинити їх вихід з ладу або навіть загоряння.

Окрему увагу слід приділити захисту від електростатичного розряду. Цей розряд виникає, коли між тілом працівника й елементами комп’ютера виникає електричний потенціал. Щоб запобігти пошкодженню електронних компонентів, під час обслуговування обладнання використовуються антистатичні засоби:

килимки, браслети або рукавички. Вони дозволяють безпечно розрядити статичну електрику з тіла людини перед контактом з пристроєм.

Зазначені заходи є обов'язковими для впровадження на підприємствах, що працюють з комп'ютерним обладнанням у складі технічної інфраструктури. Їх дотримання суттєво знижує ризик виникнення аварійних ситуацій та гарантує безпеку персоналу.

Для забезпечення електробезпеки персоналу на підприємстві необхідно організовувати регулярні інструктажі. Такі інструктажі проводяться відповідальними особами згідно з вимогами охорони праці. Після кожного інструктажу працівники підтверджують своє ознайомлення підписом у журналі техніки безпеки. Під час навчання їм пояснюють правила безпечної роботи з персональним комп'ютером, зокрема методи підключення і вимкнення пристроїв, порядок користування захисним обладнанням та алгоритм дій у разі виникнення надзвичайної ситуації. Важливим результатом такого навчання є підвищення обізнаності працівників щодо потенційних ризиків та формування в них навичок безпечної поведінки.

У разі виникнення пожежі слід діяти обережно. Категорично забороняється гасити електричні прилади водою. Перед початком гасіння потрібно відключити живлення пристрою, якщо це не створює додаткової загрози для працівника. Для боротьби з пожежами, що спричинені коротким замиканням або перегрівом обладнання, використовуються спеціальні типи вогнегасників. До них належать порошкові та вуглекислотні вогнегасники. Вони не проводять струм, тому їх застосування є безпечним навіть при наявності залишкової напруги на пристроях. Такі вогнегасники повинні бути розміщені в кожному приміщенні, де встановлено комп'ютерну техніку. Особливо важливо мати їх у зоні, де розташовані робочі станції та серверне обладнання.

Додатковим заходом безпеки є встановлення систем автоматичного виявлення диму та відключення живлення. Такі системи здатні швидко реагувати на загрозу займання. Вони вимикають електропостачання ще до того, як полум'я

розповсюдиться, що дозволяє мінімізувати шкоду та уникнути поширення пожежі на суміжні приміщення.

Зазначені вимоги узгоджуються з положеннями ДСТУ Б В.2.5-82:2016, який регламентує захисні заходи від ураження електричним струмом у будівлях і спорудах. Впровадження таких заходів дає змогу створити безпечні умови для роботи з електронними пристроями та своєчасно запобігти аваріям, пов’язаним із електроживленням. Окрему увагу слід приділяти регулярній перевірці стану обладнання, справності кабельних систем і технічному обслуговуванню електромереж [28].

Підсумовуючи описані заходи, доцільно візуалізувати логіку дій у вигляді блок-схеми. Така схема наведена на рисунку 3.1. Вона відображає основні елементи системи електробезпеки, їх взаємозв’язок та послідовність дій у разі виникнення небезпечної ситуації.

Рисунок 3.1 − Блок-схема електробезпеки

ВИСНОВКИ

У результаті виконання дипломної роботи було реалізовано повнофункціональний веб-застосунок для автоматизації діяльності дистриб’ютора фруктів. Система розроблена з використанням сучасного стеку технологій JavaScript, зокрема React для побудови клієнтського інтерфейсу та Node.js з Express для серверної логіки. Такий підхід дозволив забезпечити високу продуктивність, масштабованість і зручність обслуговування застосунку.

У процесі розробки було проаналізовано предметну область, визначено функціональні вимоги до системи, побудовано модель предметної області та архітектуру програмного комплексу. Реалізовано взаємодію між клієнтською та серверною частинами через REST API, що дозволило розділити відповідальність між компонентами та забезпечити незалежність інтерфейсу від бізнес-логіки.

Серверна частина успішно інтегрована з базою даних, у якій організовано зберігання інформації про користувачів, товари, замовлення, кошики, оплату та відгуки. Була реалізована система авторизації та аутентифікації з використанням JWT, що підвищило безпеку застосунку. Користувацький інтерфейс побудований у відповідності до принципів UX/UI-дизайну, адаптований до мобільних пристроїв і протестований на всіх ключових сторінках.

Особливу увагу було приділено тестуванню веб-додатку. Було проведено модульне, інтеграційне та ручне тестування, що дозволило виявити та усунути критичні помилки ще до розгортання системи. Результати тестування підтвердили працездатність та стабільність роботи всіх компонентів програмного забезпечення.

Загалом, поставлену мету дипломної роботи досягнуто. Створена система може бути використана як готовий інструмент для підтримки діяльності малого або середнього дистриб’ютора фруктів, а також може бути легко розширена або адаптована до інших сфер торгівлі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Vasylieva T.S., Kostenko N.P. Marketing Communications in Modern Business. – Kharkiv, 2023. – P. 256.

2. Devlin, J., Chang, M., Lee, K., & Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. [Електронний ресурс] – Режим доступу: https://www.aclweb.org/anthology/N19-1423.pdf.

3. Бондаренко О.Ю., Герасимчук В.В. Маркетинг у соціальних медіа. – Одеса, 2022. – 298 с.

4. Lewis, P., Yazdani, M., Burges, C., Wu, Y., Bart, R., & Smith, M. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. [Електронний ресурс] – Режим доступу: https://papers.neurips.cc/paper/2020/hash/a0d2ae2a45e6e2cccbc23f064d31a1ab-Abstract.html.

5. Жук І.В., Петренко В.Г. Цифровий маркетинг: новітні технології та інструменти. – Львів, 2021. – 312 с.

6. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... & Amodei, D. Language Models are Few-Shot Learners. [Електронний ресурс] – Режим доступу: https://arxiv.org/abs/2005.14165.

7. Руденко М.Л., Литвиненко І.І. Маркетинговий аналіз: методи та практика. – Дніпро, 2024. – 275 с.

8. Google. Search Engine Optimization (SEO) Starter Guide. [Електронний ресурс] – Режим доступу: https://support.google.com/webmasters/answer/7451184?hl=en.

9. Ковальчук Т.А., Мельник О.П. Сучасні стратегії маркетингу. – Київ, 2020. – 284 с.

10. Moz. The Beginner's Guide to SEO. [Електронний ресурс] – Режим доступу: https://moz.com/beginners-guide-to-seo.

11. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. Language Models are Unsupervised Multitask Learners. [Електронний ресурс] – Режим доступу: https://openai.com/research/language-models-are-unsupervised-multitask-learners.

12. Ahrefs. Ahrefs' Guide to SEO. [Електронний ресурс] – Режим доступу: https://ahrefs.com/blog/seo-basics/.

13. Chollet, F. Deep Learning with Python. [Електронний ресурс] – Режим доступу: https://www.manning.com/books/deep-learning-with-python-second-edition.

14. Kingma, D. P., & Ba, J. Adam: A Method for Stochastic Optimization. [Електронний ресурс] – Режим доступу: https://arxiv.org/abs/1412.6980.

15. Sutton, R. S., & Barto, A. G. Reinforcement Learning: An Introduction. [Електронний ресурс] – Режим доступу: https://mitpress.mit.edu/books/reinforcement-learning-second-edition.

16. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Chintala, S. PyTorch: An Imperative Style, High-Performance Deep Learning Library. [Електронний ресурс] – Режим доступу: https://papers.neurips.cc/paper/2019/hash/9015-acb92cc7efb80b1e6a59ae18b26e7d14-Abstract.html.

17. Goodfellow, I., Bengio, Y., & Courville, A. Deep Learning. [Електронний ресурс] – Режим доступу: https://www.deeplearningbook.org/.

18. Кодекс цивільного захисту України [Електронний ресурс] – Режим доступу: https://zakon.rada.gov.ua/laws/show/5403-17#Text

19. Про затвердження Положення про підсистему реагування на надзвичайні ситуації, проведення аварійно-рятувальних та інших невідкладних робіт єдиної державної системи цивільного захисту : наказ Міністерства внутрішніх справ України від 04.05.2016 №356.

20. Про затвердження Положення про штаб з ліквідації наслідків надзвичайної ситуації та видів оперативно-технічної і звітної документації штабу з ліквідації наслідків надзвичайної ситуації : наказ Міністерства внутрішніх справ України від 26.12.2014 №1406.

21. Про організацію роботи штабу з ліквідації наслідків надзвичайної ситуації та забезпечення його готовності : наказ Державної служби України з надзвичайних ситуацій від 16.03.2015 №149.

22. Рекомендації для роботодавців щодо організації виконання робіт підвищеної небезпеки під час воєнних (бойових) дій [Електронний ресурс] – Режим доступу: https://dsp.gov.ua/faq/rekomendatsii-dlia-robotodavtsiv-shchodo-orhanizatsii-vykonannia-robit-pidvyshchenoi-nebezpeky-pid-chas-voiennykh-boiovykh-dii/

23. Аветисян В.Г., Тригуб В.В., Грицина І.М., Остапов К.М. Організації аварійнорятувальних робіт : курс лекцій – Х: НУЦЗУ, 2017. – 141 с.

24. Організація аварійно-рятувальних робіт : навчальний посібник /Р.Т. Ратушний, В.Б. Лоїк, О.Д. Синельніков, В.М. Ковальчук – Львів: Видавництво ЛДУ БЖД, 2020. – 394 с.

25. Методичні рекомендації щодо розроблення планів локалізації і ліквідації аварій та їх наслідків : наказ Державної служби України з надзвичайних ситуацій від 17 травня 2022 року за №253

26. ДСТУ Б В.2.5-82:2016 Електробезпека в будівлях і спорудах. Вимоги до захисних заходів від ураження електричним струмом. Київ, 2016. 82 с.

27. ДСТУ 3675-98 Пожежна техніка. Вогнегасники переносні. Загальні технічні вимоги та методи випробувань. Зі зміною № 1. Київ, 1998. 66 с.

28. ДСТУ EN 12464-1:2016 Світло та освітлення. Освітлення робочих місць. Частина 1. Внутрішні робочі місця (EN 12464-1:2011, IDT). Київ, 2016. 11 с.

ДОДАТКИ

Додаток А – Лістинги коду проєкту - Frontend

Лістинг коду А.1 – Файл client/package.json

```
{
  "name": "fruit-distributor-client",
  "version": "1.0.0",
  "private": true,
  "description": "Client-side web application for a fruit distributor, built with
React.",
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject",
    "lint": "eslint ./src --ext .js,.jsx"
  },
  "dependencies": {
    "axios": "^1.6.7",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-router-dom": "^6.22.3",
    "react-scripts": "5.0.1",
    "bootstrap": "^5.3.3"
  },
  "devDependencies": {
    "eslint": "^8.56.0",
    "eslint-plugin-react": "^7.34.1"
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 2 chrome versions",
      "last 2 firefox versions",
      "last 2 edge versions"
    ]
  }
}
```

Лістинг коду А.1 – Файл src/index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';
import './styles/main.css';
const root = ReactDOM.createRoot(document.getElementById('root'));
```

```
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

Лістинг коду А.1 – Файл src/App.jsx

```
import React from 'react';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import Home from './pages/Home';
import Cart from './pages/Cart';
import Checkout from './pages/Checkout';
import Profile from './pages/Profile';
import AdminPanel from './pages/AdminPanel';
import Header from './components/Header';

function App() {
  return (
    <Router>
      <Header />
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/cart" element={<Cart />} />
        <Route path="/checkout" element={<Checkout />} />
        <Route path="/profile" element={<Profile />} />
        <Route path="/admin" element={<AdminPanel />} />
      </Routes>
    </Router>
  );
}

export default App;
```

Лістинг коду А.1 – Файл src/api/api.js

```
import axios from 'axios';

const API = axios.create({
  baseURL: 'http://localhost:5000/api',
```

```
  headers: {
    'Content-Type': 'application/json'
  }
});

// Додаємо токен до запиту, якщо він є
API.interceptors.request.use((config) => {
  const token = localStorage.getItem('token');
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});

export default API;
```

Лістинг коду А.1 – Файл src/components/ProductCard.jsx

```
import React from 'react';

function ProductCard({ product, onAddToCart }) {
  return (
    <div className="card">
      <img src={product.image} alt={product.name} className="card-img-top" />
      <div className="card-body">
        <h5 className="card-title">{product.name}</h5>
        <p className="card-text">{product.price} ₴</p>
        <button
          className="btn btn-primary"
          onClick={() => onAddToCart(product)}
        >
          Додати до кошика
        </button>
      </div>
    </div>
  );
}

export default ProductCard;
```

Лістинг коду А.1 – Файл src/components/CartItem.jsx

```
import React from 'react';

function CartItem({ item, onRemove }) {
  return (
    <div className="cart-item d-flex justify-content-between align-items-center
mb-2">
      <div>
        <strong>{item.name}</strong> — {item.quantity} x {item.price} ₴
      </div>
      <button className="btn btn-danger btn-sm" onClick={() => onRemove(item.id)}>
        Видалити
      </button>
    </div>
  );
}

export default CartItem;
```

Лістинг коду А.1 – Файл src/components/Header.jsx

```
import React from 'react';
import { Link } from 'react-router-dom';

function Header() {
  return (
    <nav className="navbar navbar-expand-lg navbar-light bg-light mb-4">
      <div className="container">
        <Link className="navbar-brand" to="/">Фрукти+</Link>
        <div className="collapse navbar-collapse">
          <ul className="navbar-nav ms-auto">
            <li className="nav-item">
              <Link className="nav-link" to="/">Головна</Link>
            </li>
            <li className="nav-item">
              <Link className="nav-link" to="/cart">Кошик</Link>
            </li>
            <li className="nav-item">
              <Link className="nav-link" to="/profile">Кабінет</Link>
```

```
            </li>
          </ul>
        </div>
      </div>
    </nav>
  );
}

export default Header;
```

Лістинг коду А.1 – Файл src/pages/Home.jsx

```
import React, { useEffect, useState } from 'react';
import API from '../api/api';
import ProductCard from '../components/ProductCard';

function Home() {
  const [products, setProducts] = useState([]);

  useEffect(() => {
    API.get('/products')
      .then(res => setProducts(res.data))
      .catch(err => console.error('Помилка завантаження товарів:', err));
  }, []);

  const handleAddToCart = (product) => {
    // Реалізація додавання до кошика
    console.log('Додано до кошика:', product.name);
  };

  return (
    <div className="container">
      <h2>Каталог товарів</h2>
      <div className="row">
        {products.map((product) => (
          <div className="col-md-4 mb-4" key={product.id}>
            <ProductCard product={product} onAddToCart={handleAddToCart} />
          </div>
        ))}
      </div>
    </div>
  );
}

export default Home;
```

Лістинг коду А.1 – Файл src/pages/Cart.jsx

```
import React, { useState } from 'react';
import CartItem from '../components/CartItem';
```

```
function Cart() {
  const [cart, setCart] = useState([
    // Початкові товари для прикладу
    { id: 1, name: 'Яблуко', quantity: 2, price: 30 },
    { id: 2, name: 'Банан', quantity: 1, price: 25 }
  ]);

  const handleRemove = (id) => {
    setCart(cart.filter(item => item.id !== id));
  };

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

  return (
    <div className="container">
      <h2>Ваш кошик</h2>
      {cart.length === 0 ? (
        <p>Кошик порожній</p>
      ) : (
        <>
          {cart.map(item => (
            <CartItem key={item.id} item={item} onRemove={handleRemove} />
          ))}
          <hr />
          <h5>Загальна сума: {total} ₴</h5>
        </>
      )}
    </div>
  );
}

export default Cart;
```

Лістинг коду А.1 – Файл src/pages/Checkout.jsx

```
import React, { useState } from 'react';

function Checkout() {
  const [form, setForm] = useState({
    name: '',
```

```
  address: '',
  phone: ''
});

const handleChange = (e) => {
  setForm({ ...form, [e.target.name]: e.target.value });
};

const handleSubmit = (e) => {
  e.preventDefault();
  // Відправка даних на сервер (псевдо)
  console.log('Замовлення оформлено:', form);
  alert('Дякуємо за замовлення!');
};

return (
  <div className="container">
    <h2>Оформлення замовлення</h2>
    <form onSubmit={handleSubmit} className="mt-3">
      <div className="mb-3">
        <label className="form-label">Ім’я</label>
        <input
          type="text"
          name="name"
          className="form-control"
          value={form.name}
          onChange={handleChange}
          required
        />
      </div>
      <div className="mb-3">
        <label className="form-label">Адреса доставки</label>
        <input
          type="text"
          name="address"
          className="form-control"
```

```
            value={form.address}
            onChange={handleChange}
            required
          />
        </div>
        <div className="mb-3">
          <label className="form-label">Телефон</label>
          <input
            type="tel"
            name="phone"
            className="form-control"
            value={form.phone}
            onChange={handleChange}
            required
          />
        </div>
        <button type="submit" className="btn btn-success">Підтвердити
замовлення</button>
      </form>
    </div>
  );
}

export default Checkout;
```

Лістинг коду А.1 – Файл src/pages/Profile.jsx

```
import React, { useEffect, useState } from 'react';
import API from '../api/api';

function Profile() {
  const [user, setUser] = useState(null);
  const [orders, setOrders] = useState([]);

  useEffect(() => {
```

```
    API.get('/users/me')
      .then(res => setUser(res.data))
      .catch(err => console.error('Помилка отримання даних користувача:', err));

    API.get('/orders/my')
      .then(res => setOrders(res.data))
      .catch(err => console.error('Помилка отримання замовлень:', err));
  }, []);

  return (
    <div className="container">
      <h2>Особистий кабінет</h2>
      {user && (
        <div className="mb-4">
          <p><strong>Ім’я:</strong> {user.name}</p>
          <p><strong>Email:</strong> {user.email}</p>
        </div>
      )}

      <h4>Історія замовлень</h4>
      {orders.length === 0 ? (
        <p>Замовлення відсутні</p>
      ) : (
        <ul className="list-group">
          {orders.map(order => (
            <li key={order.id} className="list-group-item">
              Замовлення #{order.id} — {order.total} ₴ — {new
Date(order.date).toLocaleDateString()}
            </li>
          ))}
        </ul>
      )}
    </div>
  );
}
```

```
export default Profile;
```

Лістинг коду А.1 – Файл src/pages/AdminPanel.jsx

```
import React, { useEffect, useState } from 'react';
import API from '../api/api';

function AdminPanel() {
  const [products, setProducts] = useState([]);
  const [newProduct, setNewProduct] = useState({
    name: '',
    price: '',
    category: ''
  });

  useEffect(() => {
    API.get('/products')
      .then(res => setProducts(res.data))
      .catch(err => console.error('Не вдалося завантажити товари', err));
  }, []);

  const handleChange = (e) => {
    setNewProduct({ ...newProduct, [e.target.name]: e.target.value });
  };

  const handleAdd = (e) => {
    e.preventDefault();
    API.post('/products', newProduct)
      .then(res => {
        setProducts([...products, res.data]);
        setNewProduct({ name: '', price: '', category: '' });
      })
      .catch(err => console.error('Помилка додавання товару', err));
  };

  return (
    <div className="container">
      <h2>Адмін-панель</h2>
      <form onSubmit={handleAdd} className="mb-4">
        <div className="mb-2">
          <input
            type="text"
            name="name"
            placeholder="Назва"
            className="form-control"
            value={newProduct.name}
            onChange={handleChange}
            required
          />
        </div>
        <div className="mb-2">
          <input
            type="number"
            name="price"
            placeholder="Ціна"
            className="form-control"
            value={newProduct.price}
            onChange={handleChange}
            required
```

```
          />
        </div>
        <div className="mb-2">
          <input
            type="text"
            name="category"
            placeholder="Категорія"
            className="form-control"
            value={newProduct.category}
            onChange={handleChange}
            required
          />
        </div>
        <button type="submit" className="btn btn-primary">Додати товар</button>
      </form>

      <h4>Список товарів</h4>
      <ul className="list-group">
        {products.map(product => (
          <li key={product.id} className="list-group-item">
            {product.name} — {product.price} ₴ ({product.category})
          </li>
        ))}
      </ul>
    </div>
  );
}

export default AdminPanel;
```

Лістинг коду А.1 – Файл src/styles/main.css

```
body {
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
  background-color: #f8f9fa;
  margin: 0;
  padding: 0;
}

.container {
  padding-top: 2rem;
  padding-bottom: 2rem;
}

.card {
  height: 100%;
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.05);
  border-radius: 0.5rem;
```

```
}

.card img {
  max-height: 200px;
  object-fit: cover;
  border-top-left-radius: 0.5rem;
  border-top-right-radius: 0.5rem;
}

.cart-item {
  border-bottom: 1px solid #dee2e6;
  padding-bottom: 0.5rem;
}
```

Додаток Б – Лістинги коду проєкту - Backend

Лістинг коду Б.1 – Файл server/package.json

```
{
  "name": "fruit-distributor-server",
  "version": "1.0.0",
  "description": "Backend for fruit distributor web application using Node.js and
Express",
  "main": "server.js",
  "scripts": {
    "start": "node server.js",
    "dev": "nodemon server.js"
  },
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "cors": "^2.8.5",
    "dotenv": "^16.3.1",
    "express": "^4.18.2",
    "jsonwebtoken": "^9.0.2",
    "mongoose": "^7.6.1"
  },
  "devDependencies": {
    "nodemon": "^3.0.2"
  },
  "author": "",
  "license": "ISC"
}
```

Лістинг коду Б.1 – Файл server/server.js

```
const express = require('express');
const mongoose = require('mongoose');
const dotenv = require('dotenv');
const cors = require('cors');

const authRoutes = require('./routes/authRoutes');
const productRoutes = require('./routes/productRoutes');
const orderRoutes = require('./routes/orderRoutes');
```

```
dotenv.config();

const app = express();
const PORT = process.env.PORT || 5000;

// Middleware
app.use(cors());
app.use(express.json());

// Routes
app.use('/api/auth', authRoutes);
app.use('/api/products', productRoutes);
app.use('/api/orders', orderRoutes);

// DB Connection
mongoose.connect(process.env.MONGO_URI, {
  useNewUrlParser: true,
  useUnifiedTopology: true
})
.then(() => {
  app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
})
.catch((error) => console.error('MongoDB connection error:', error));
```

Лістинг коду Б.1 – Файл server/config/db.js

```
const mongoose = require('mongoose');
const dotenv = require('dotenv');

dotenv.config();

const connectDB = async () => {
  try {
    await mongoose.connect(process.env.MONGO_URI, {
      useNewUrlParser: true,
      useUnifiedTopology: true
    });
    console.log('MongoDB connected successfully');
  } catch (error) {
    console.error('MongoDB connection failed:', error.message);
    process.exit(1);
  }
};

module.exports = connectDB;
```

Лістинг коду Б.1 – Файл server/controllers/authController.js

```
const jwt = require('jsonwebtoken');
const bcrypt = require('bcryptjs');
const User = require('../models/User');

const generateToken = (user) => {
  return jwt.sign({ id: user._id }, process.env.JWT_SECRET, {
    expiresIn: '7d',
  });
};

exports.register = async (req, res) => {
  try {
    const { name, email, password } = req.body;

    const existingUser = await User.findOne({ email });
    if (existingUser) return res.status(400).json({ message: 'Користувач вже
існує' });

    const hashedPassword = await bcrypt.hash(password, 10);
    const user = await User.create({ name, email, password: hashedPassword });

    res.status(201).json({
      user: { id: user._id, name: user.name, email: user.email },
      token: generateToken(user),
    });
  } catch (err) {
    res.status(500).json({ message: 'Помилка реєстрації' });
  }
};

exports.login = async (req, res) => {
  try {
    const { email, password } = req.body;

    const user = await User.findOne({ email });
    if (!user) return res.status(400).json({ message: 'Невірний email або пароль'
});

    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) return res.status(400).json({ message: 'Невірний email або
пароль' });
```

```
    res.json({
      user: { id: user._id, name: user.name, email: user.email },
      token: generateToken(user),
    });
  } catch (err) {
    res.status(500).json({ message: 'Помилка входу' });
  }
};

exports.getProfile = async (req, res) => {
  try {
    const user = await User.findById(req.user.id).select('-password');
    res.json(user);
  } catch (err) {
    res.status(500).json({ message: 'Не вдалося отримати дані користувача' });
  }
};
```

Лістинг коду Б.1 – Файл server/controllers/productController.js

```
const Product = require('../models/Product');

exports.getAllProducts = async (req, res) => {
  try {
    const products = await Product.find();
    res.json(products);
  } catch (err) {
    res.status(500).json({ message: 'Не вдалося отримати товари' });
  }
};

exports.getProductById = async (req, res) => {
  try {
    const product = await Product.findById(req.params.id);
    if (!product) return res.status(404).json({ message: 'Товар не знайдено' });
    res.json(product);
  } catch (err) {
    res.status(500).json({ message: 'Помилка пошуку товару' });
  }
};
```

```
exports.createProduct = async (req, res) => {
  try {
    const { name, price, category, image } = req.body;
    const newProduct = new Product({ name, price, category, image });
    await newProduct.save();
    res.status(201).json(newProduct);
  } catch (err) {
    res.status(500).json({ message: 'Не вдалося створити товар' });
  }
};

exports.deleteProduct = async (req, res) => {
  try {
    const product = await Product.findByIdAndDelete(req.params.id);
    if (!product) return res.status(404).json({ message: 'Товар не знайдено' });
    res.json({ message: 'Товар видалено' });
  } catch (err) {
    res.status(500).json({ message: 'Не вдалося видалити товар' });
  }
};
```

Лістинг коду Б.1 – Файл server/controllers/orderController.js

```
const Order = require('../models/Order');

exports.createOrder = async (req, res) => {
  try {
    const { items, total } = req.body;
    const newOrder = new Order({
      userId: req.user.id,
      items,
      total,
      date: new Date(),
    });
    await newOrder.save();
    res.status(201).json(newOrder);
  } catch (err) {
    res.status(500).json({ message: 'Не вдалося створити замовлення' });
  }
};
```

```
exports.getUserOrders = async (req, res) => {
  try {
    const orders = await Order.find({ userId: req.user.id });
    res.json(orders);
  } catch (err) {
    res.status(500).json({ message: 'Помилка при отриманні замовлень' });
  }
};
```

Лістинг коду Б.1 – Файл server/middleware/authMiddleware.js

```
const jwt = require('jsonwebtoken');

const authMiddleware = (req, res, next) => {
  const authHeader = req.headers.authorization;

  if (!authHeader || !authHeader.startsWith('Bearer ')) {
    return res.status(401).json({ message: 'Немає токена доступу' });
  }

  const token = authHeader.split(' ')[1];

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.user = decoded;
    next();
  } catch (err) {
    res.status(403).json({ message: 'Недійсний токен' });
  }
};

module.exports = authMiddleware;
```

Лістинг коду Б.1 – Файл server/models/User.js

```
const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
  name: {
    type: String,
```

```
    required: true,
    trim: true
  },
  email: {
    type: String,
    unique: true,
    required: true,
    lowercase: true
  },
  password: {
    type: String,
    required: true
  }
});

module.exports = mongoose.model('User', userSchema);
```

Лістинг коду Б.1 – Файл server/models/Product.js

```
const mongoose = require('mongoose');

const productSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
    trim: true
  },
  price: {
    type: Number,
    required: true
  },
  category: {
    type: String,
    required: true
  },
  image: {
    type: String,
    default: ''
  }
});
```

```
module.exports = mongoose.model('Product', productSchema);
```

Лістинг коду Б.1 – Файл server/models/Order.js

```
const mongoose = require('mongoose');

const orderSchema = new mongoose.Schema({
  userId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true
  },
  items: [
    {
      productId: { type: mongoose.Schema.Types.ObjectId, ref: 'Product' },
      name: String,
      quantity: Number,
      price: Number
    }
  ],
  total: {
    type: Number,
    required: true
  },
  date: {
    type: Date,
    default: Date.now
  }
});

module.exports = mongoose.model('Order', orderSchema);
```

Лістинг коду Б.1 – Файл server/models/Category.js

```
const mongoose = require('mongoose');

const categorySchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
```

```
    unique: true
  }
});

module.exports = mongoose.model('Category', categorySchema);
```

Лістинг коду Б.1 – Файл server/models/Review.js

```
const mongoose = require('mongoose');

const reviewSchema = new mongoose.Schema({
  productId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Product',
    required: true
  },
  userId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true
  },
  rating: {
    type: Number,
    required: true,
    min: 1,
    max: 5
  },
  comment: {
    type: String,
    trim: true
  },
  date: {
    type: Date,
    default: Date.now
  }
});

module.exports = mongoose.model('Review', reviewSchema);
```

Лістинг коду Б.1 – Файл server/routes/authRoutes.js

```
const express = require('express');
const router = express.Router();
const authController = require('../controllers/authController');
const authMiddleware = require('../middleware/authMiddleware');

router.post('/register', authController.register);
router.post('/login', authController.login);
router.get('/me', authMiddleware, authController.getProfile);

module.exports = router;
```

Лістинг коду Б.1 – Файл server/routes/productRoutes.js

```
const express = require('express');
const router = express.Router();
const productController = require('../controllers/productController');
const authMiddleware = require('../middleware/authMiddleware');

router.get('/', productController.getAllProducts);
router.get('/:id', productController.getProductById);
router.post('/', authMiddleware, productController.createProduct);
router.delete('/:id', authMiddleware, productController.deleteProduct);

module.exports = router;
```

Лістинг коду Б.1 – Файл server/routes/orderRoutes.js

```
const express = require('express');
const router = express.Router();
const orderController = require('../controllers/orderController');
const authMiddleware = require('../middleware/authMiddleware');

router.post('/', authMiddleware, orderController.createOrder);
router.get('/my', authMiddleware, orderController.getUserOrders);

module.exports = router;
```

Лістинг коду Б.1 – Файл server/utils/validator.js

```
exports.validateEmail = (email) => {
  const re = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
  return re.test(String(email).toLowerCase());
};

exports.validatePassword = (password) => {
  // Мінімум 6 символів
  return password && password.length >= 6;
};
```

Лістинг коду Б.1 – Файл database/schema.sql

```
-- Таблиця користувачів
CREATE TABLE users (
  id SERIAL PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  email VARCHAR(100) UNIQUE NOT NULL,
  password TEXT NOT NULL
);

-- Таблиця категорій
CREATE TABLE categories (
  id SERIAL PRIMARY KEY,
  name VARCHAR(100) NOT NULL UNIQUE
);

-- Таблиця товарів
CREATE TABLE products (
  id SERIAL PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  price NUMERIC(10, 2) NOT NULL,
  category_id INTEGER REFERENCES categories(id),
  image TEXT
);

-- Таблиця замовлень
CREATE TABLE orders (
  id SERIAL PRIMARY KEY,
  user_id INTEGER REFERENCES users(id),
  total NUMERIC(10, 2) NOT NULL,
  date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Таблиця деталей замовлення
CREATE TABLE order_items (
  id SERIAL PRIMARY KEY,
  order_id INTEGER REFERENCES orders(id),
  product_id INTEGER REFERENCES products(id),
  quantity INTEGER NOT NULL,
  price NUMERIC(10, 2) NOT NULL
);

-- Таблиця відгуків
CREATE TABLE reviews (
  id SERIAL PRIMARY KEY,
  product_id INTEGER REFERENCES products(id),
  user_id INTEGER REFERENCES users(id),
  rating INTEGER NOT NULL CHECK (rating BETWEEN 1 AND 5),
  comment TEXT,
  date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Додаток В – Диск із кваліфікаційною роботою бакалавра