Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра автоматизації та інтелектуальних інформаційних технологій

Бакалаврська кваліфікаційна робота на тему:

«РОЗРОБКА ЗАСТОСУНКУ ДЛЯ ОФОРМЛЕННЯ ОНЛАЙН-ЗАМОВЛЕНЬ ЗА ДОПОМОГОЮ REACT ТА NESTJS»

Виконав: студент IV курсу, групи 1ІСТ-21б
спеціальності 126 – Інформаційні системи та технології
(шифр і назва спеціальності)
Кривогубченко К.Д.
(прізвище та ініціали)

Керівник: к.т.н., доцент кафедри АІІТ
Богач І. В.
(прізвище та ініціали)
«11» 06 2025 р.

Рецензент: д.т.н., професор кафедри КН
Іванчук Я. В.
(прізвище та ініціали)
«11» 06 2025 р.

Допущено до захисту
завідувач кафедри АІІТ
«12» 06 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра автоматизації та інтелектуальних інформаційних технологій
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 – Інформаційні технології
Спеціальність 126 – Інформаційні системи та технології
Освітньо-професійна програма Інтелектуальні інформаційні системи

ЗАТВЕРДЖУЮ
завідувач кафедри АІІТ
д.т.н., проф. Бісікало О. В.
«12» 06 2025 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Кривогубченко Ксенії Денисівні
(прізвище, ім'я, по батькові)

1. Тема роботи «Розробка застосунку для оформлення онлайн-замовлень за допомогою React та NestJS»

керівник роботи Богач Ілона Віталіївна, к.т.н, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від «20» березня 2025 року №97

2. Термін подання студентом роботи 10.06.2025 р.

3. Вихідні дані до роботи:

- наявність пристрою із підключенням до мережі Інтернет для доступу до застосунку;
- встановлений Node.js;
- встановлений менеджер пакетів npm.

4. Зміст текстової частини (перелік питань, які потрібно розробити) Дослідження предметної області. Проєктування структури системи та огляд інструментів розробки. Розробка програмного забезпечення. Тестування застосунку.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень) Діаграма взаємодії користувача з додатком. Діаграма класів та сервісів додатку. Діаграма функціоналу збережених адрес.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Богач І. В., доц. каф. АІІТ		

7. Дата видачі завдання 20.03.2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Термін виконання		Примітки
		початок	закінчення	
1	Вибір, узгодження та затвердження теми БКР	03.10.2024	09.10.2024	Вик.
2	Аналіз предметної області та опрацювання літературних джерел	12.03.2025	29.03.2025	Вик.
3	Проєктування структури системи та огляд інструментів розробки	01.04.2025	26.04.2025	Вик.
4	Розробка програмного забезпечення	29.04.2025	10.05.2025	Вик.
5	Тестування додатку та виправлення помилок	13.05.2025	24.05.2025	Вик.
6	Оформлення пояснювальної записки та графічної частини	27.05.2025	31.05.2025	Вик.
7	Попередній захист, доопрацювання, та рецензування БКР	10.06.2025	15.06.2025	Вик.
8	Захист БКР ЕК	16.06.2025	23.06.2025	Вик.

Студент _____ Кривогубченко К.Д.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Богач І.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 73 сторінок формату А4 в тому числі і додатків, на яких є 36 рисунків, 2 таблиці, список використаних джерел містить 39 найменувань.

Робота присвячена розробці застосунку для оформлення замовлень за допомогою мови JavaScript та фреймворків React і NestJS. Метою роботи є вдосконалення та збільшення зручності процесу оформлення онлайн-покупок.

Ключові слова: React, NestJS, веб-застосунок, онлайн-покупки, електронна комерція.

ABSTRACT

The bachelor's thesis consists of 73 pages of A4 format, including appendices, on which there is 36 figures, 2 tables, the list of used sources contains 39 names.

This work is dedicated to the development of an application for placing orders using JavaScript and the React and NestJS frameworks. The goal of the work is to improve and enhance the convenience of the online shopping process.

Keywords: React, NestJS, web application, online shopping, e-commerce.

ЗМІСТ

ВСТУП

Актуальність роботи. Онлайн-покупки стали невід'ємною частиною сучасного життя завдяки своєму зручному формату, що дозволяє здійснювати покупки в будь-який час і з будь-якого місця. У світі, де темп життя постійно зростає, можливість придбати необхідні товари без необхідності виходити з дому стає все більш актуальною. Інтернет-магазини пропонують широкий асортимент товарів, що значно розширює вибір для споживачів і дозволяє знайти саме те, що потрібно, із можливістю миттєво порівнювати ціни та відгуки.

Згідно з даними Statista [1], у 2024 році глобальні онлайн-продажі склали понад 5,9 трильйона доларів, що є значним зростанням у порівнянні з попередніми роками. Прогнози на майбутнє також вражають: до 2027 року очікується, що обсяг онлайн-продажів досягне майже 8,1 трильйона доларів.

В Україні, згідно з дослідженням GfK [1], онлайн-продажі зросли на 20% за останній рік, а частка інтернет-покупок у загальному обсязі роздрібної торгівлі досягла 15%.

Покупці можуть легко здійснювати пошук товарів, переглядати їх опис та користуватися методами оплати, які зручні саме для них. Популярність онлайн-шопінгу також спричинена пандемією COVID-19, коли необхідність уникати фізичних контактів стала особливо важливою, адже такий спосіб здійснення покупок дозволяють зменшити ризики і отримати потрібні товари, не виходячи з дому.

Водночас, онлайн-шопінг є зручним не лише для покупців, а й для продавців, оскільки такий спосіб продажу відкриває можливості не лише на локальному, а й на міжнародному ринку. Споживачі можуть купувати продукцію з різних куточків світу, що робить процес покупки більш універсальним. До того ж, онлайн-платформи часто пропонують акції, знижки та розпродажі, що робить здійснення покупок через онлайн-платформи більш вигідним.

Однак на даний час існує проблема із зручністю оформлення онлайн-покупок, а саме нечіткості у процесі надання особистих даних, а також користувацькі помилки при введенні даних.

Метою роботи є удосконалення програмного забезпечення для створення онлайн замовлень за рахунок використання NestJS.

Для досягнення мети роботи необхідно **вирішити наступні задачі:**

1. Провести аналіз предметної області онлайн-платформ та сторінок для оформлення онлайн-замовлень.

2. Провести огляд та аналіз аналогів.

3. Розробити структуру додатку.

4. Розробити програмне забезпечення для сторінки оформлення онлайн-покупок та протестувати отриманий результат.

Об'єктом дослідження є процес створення онлайн-замовлення.

Предметом дослідження є засоби та методи розробки програмного забезпечення для створення онлайн-замовлень.

Методи дослідження. Аналіз існуючих рішень розробок веб-додатків для онлайн-замовлень, методи проєктування та розробки програмного забезпечення.

Науково-практичний результат роботи полягає в розробці удосконаленого програмного забезпечення для створення онлайн-замовлень, що на відміну від існуючих полегшить процес оформлення онлайн-покупок для користувачів та залучить більше покупців до продавців онлайн-платформи, що на відміну від існуючих пропонує покращений користувацький досвід та зниження користувацьких помилок при введенні даних.

В результаті роботи створено клієнтську та серверну частини для оформлення онлайн-замовлень, яку може застосувати будь-яка онлайн-платформа для залучення більшої кількості покупців. Застосунок допоможе знизити випадки користувацьких помилок при введенні даних і покращити досвід оформлення покупок, зробити його більш зручним порівняно із аналогами.

Апробація та публікація результатів дослідження

Результати роботи було представлено на LIV Всеукраїнській науково-технічній конференції підрозділів Вінницького національного технічного університету (2025) [2] та на Міжнародній науково-практичній Інтернет-конференції студентів, аспірантів та молодих науковців «МОЛОДЬ В НАУЦІ: ДОСЛІДЖЕННЯ, ПРОБЛЕМИ, ПЕРСПЕКТИВИ (МН-2025)» [3].

Впровадження результатів роботи

Результати роботи планується використати в розробках та управлінні проектами ТОВ «Файв Системз Девелопмент» (додаток Д).

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Дослідження актуальності онлайн замовлень

Інтернет-магазин — це сайт, який здійснює торгівлю за допомогою мережі Інтернет. Інтернет-магазини дозволяють користувачам онлайн, у браузері або через мобільний додаток, сформувати замовлення, вибрати спосіб оплати і доставки, оплатити замовлення.

Клієнтів залучають до інтернет-магазинів не тільки через високий рівень зручності, але і через широкий асортимент вибору, легкий доступ до конкурентних цін та допоміжної інформації. Бізнес-організації прагнуть пропонувати інтернет-магазини не тільки тому, що вони мають набагато меншу вартість, але також через те, що вони пропонують доступ до світового ринку, підвищують цінність для клієнтів та будують стійкі можливості.

Ритм сучасного світу змушує нас шукати зручніші та ефективніші способи здійснення покупок. Самообслуговування, онлайн-замовлення та персоналізовані знижки стали ключовими елементами нової епохи шопінгу, які не лише спрощують процес здійснення покупок, але й роблять його приємним і вигідним. Мобільні технології та інтернет повністю трансформували світ покупок. Онлайн-замовлення продуктів із доставкою додому або самовивозом — сервіс, який став незамінним для багатьох. Він об’єднує зручність, економію часу та простоту.

Онлайн-покупки дозволяють уникнути втомлюючих прогулянок між полицями магазину. Покупець може спокійно переглянути асортимент, вибрати потрібні товари і оформити замовлення з будь-якого місця — вдома, на роботі, чи навіть у дорозі. Це особливо зручно для людей, яким складно виділити час для походу в магазин.

До переваг такого формату належить, зокрема, можливість планування. Перед оформленням замовлення, ви маєте змогу ретельно обдумати свої покупки, порівняти ціни, перевірити, чи в списку є все необхідне. Такий підхід мінімізує імпульсивні витрати і допомагає використовувати бюджет більш раціонально.

Доставка продуктів також стала справжнім порятунком у кризових ситуаціях. Наприклад, під час пандемії COVID-19 багато людей змогли уникнути зайвих контактів і забезпечити себе всім необхідним завдяки онлайн-шопінгу. Ця функція залишається актуальною і сьогодні, забезпечуючи комфорт і безпеку. [4]

1.2 Приклади та аналоги застосунків для оформлення онлайн-замовлень

Застосунки для оформлення онлайн-замовлень є невід'ємною частиною онлайн-шопінгу та ефективної взаємодії з покупцями. Вони надають користувачам інтернет-магазину можливість зручно, легко та зрозуміло надати необхідні дані для оформлення замовлення та його оплати, а власникам інтернет-магазину – можливість отримувати та зберігати аналітику про покупців та замовлення, залучати більше покупців до свого бізнесу.

Такі застосунки зазвичай включають у себе область перегляду товарів, які користувач хоче придбати, форму для заповнення необхідних даних, відображення вартості для сплати замовлення та доступних способів оплати. Іноді також відображаються опції доставки, упаковки, застосування купонів та промо-кодів.

Розглянемо декілька існуючих аналогів застосунків для оформлення онлайн-замовлень.

1.2.1 Shopify

Shopify [5] - це потужна та універсальна платформа електронної комерції, розроблена, щоб допомогти компаніям будь-якого розміру створювати власні онлайн-магазини та керувати ними. Shopify надає комплексне рішення для продавців, пропонуючи інструменти, необхідні для створення, підтримки та масштабування онлайн-магазину без наявності глибоких технічних знань. Функціональні можливості Shopify охоплюють різні аспекти онлайн-торгівлі: налаштування вітрини, обробка платежів, керування запасами та аналіз ефективності бізнесу.

Однією з основних переваг Shopify є його зручний інтерфейс, приклад якого зображено на рисунку 1.1. Платформа пропонує ряд настроюваних тем, що дозволяє власникам магазинів легко створити професійний вебсайт без необхідності наймати веб-розробника. Однак для більш досвідчених користувачів, Shopify також надає параметри налаштування, які дозволяють більш детально коригувати дизайн і функціональність магазину.

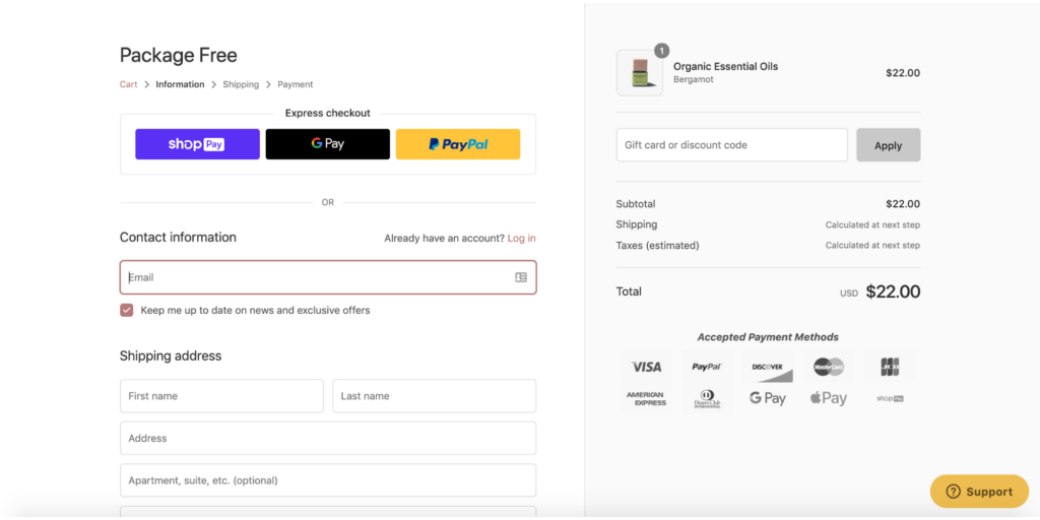

Рисунок 1.1 – Інтерфейс сторінки оформлення замовлення Shopify

Особливістю Shopify є бездоганна інтеграція з різними платіжними системами. Платформа підтримує кілька варіантів оплати, включаючи кредитні та дебетові картки, PayPal, Apple Pay і навіть криптовалюту, отже, підприємства можуть задовольнити широкий спектр уподобань клієнтів. Власна платіжна система Shopify – Shopify Payments, спрощує транзакції та усуває потребу в посередниках, полегшуючи керування платежами та відстеження фінансових даних. Платформа також забезпечує безпеку за допомогою шифрування SSL, яке допомагає захистити інформацію клієнтів і зміцнити довіру користувачів.

Управління запасами – ще один важливий аспект функціональності Shopify. Власники магазинів можуть легко відстежувати рівень запасів, керувати варіантами товару (наприклад, розміром або кольором) і налаштовувати автоматичні сповіщення про низький запас. Ця функція допомагає запобігти вичерпанню запасів і гарантує, що продукти завжди будуть доступні клієнтам, коли

вони переглядають сайт. Shopify також підтримує дропшипінг, що дозволяє компаніям продавати продукти безпосередньо від постачальників та без необхідності керувати запасами, що ідеально підходить для підприємців, які прагнуть мінімізувати накладні витрати.

Крім того, Shopify надає комплексний набір маркетингових інструментів, щоб допомогти компаніям залучити клієнтів і розширити свою присутність у Інтернеті. Ці інструменти включають функції SEO, які допомагають магазинам займати вищі позиції в результатах пошукової системи, та вбудовані опції для створення кодів знижок і проведення кампаній продажу. Платформа також підтримує інтеграцію соціальних медіа, що полегшує підключення інтернет-магазину до таких платформ, як Facebook, Instagram і Pinterest, де підприємства можуть безпосередньо продавати товари користувачам або рекламувати свої продукти за допомогою оголошень.

1.2.2 BigCommerce

BigCommerce [6] – це надійна платформа, призначена для компаній різного розміру, яка надає повний набір інструментів для створення, керування та розвитку онлайн-магазинів. Одними з головних переваг BigCommerce є гнучкість і масштабованість, що робить платформу придатною як для малих підприємств, які прагнуть здійснювати онлайн-торгівлю, так і для великих підприємств, які прагнуть розширити свої цифрові операції.

Інтерфейс BigCommerce (рис. 1.2) розроблений таким чином, щоб бути інтуїтивно зрозумілим навіть для тих, хто не має технічних знань. Він пропонує вибір професійно розроблених шаблонів, які можна легко налаштувати відповідно до бренду та стилю бізнесу. Ці шаблони також адаптовані до мобільних пристроїв, що гарантує, що магазин забезпечить зручні покупки незалежно від пристрою користувача. Для тих, хто хоче більше контролювати дизайн, BigCommerce також дозволяє детальне налаштування за допомогою HTML, CSS і JavaScript, пропонуючи більшу гнучкість для розробників і досвідчених користувачів.

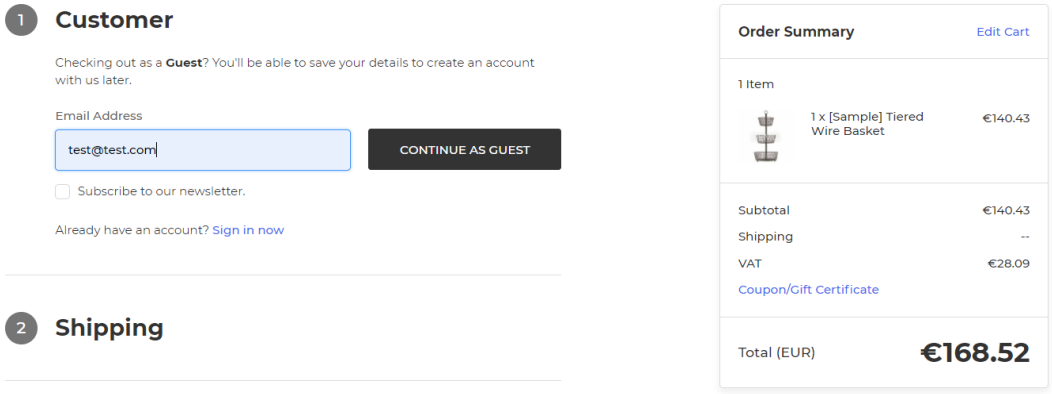

Рисунок 1.2 – Інтерфейс сторінки оформлення замовлення BigCommerce

Управління продуктами є фундаментальною частиною функціональності BigCommerce. Платформа пропонує потужні інструменти для організації каталогів продуктів і керування ними, включно з опціями для створення варіантів продуктів, категоризації товарів і керування рівнями запасів. Власники магазинів можуть налаштувати автоматичні сповіщення про низький рівень запасів, що гарантує, що популярні товари ніколи не закінчаться несподівано. Крім того, BigCommerce підтримує масовий імпорт і експорт продуктів, що особливо корисно для підприємств із великими запасами.

В основному, платформа розроблена для обробки великих обсягів трафіку та широких каталогів продуктів, що робить її вдалим варіантом для великих компаній, або компаній, які планують розширюватися. BigCommerce також пропонує ряд функцій для полегшення багатоканальних продажів, дозволяючи компаніям розміщувати товари на інших ринках, таких як eBay, Amazon і Google Shopping. Ця функція допомагає охопити нові сегменти клієнтів без потреби в створення окремої платформи чи вітрини.

Підтримка клієнтів – ще одна ключова перевага BigCommerce. Платформа пропонує цілодобову підтримку через кілька каналів, наприклад, текстовий чат, телефон та електронну пошту. Існує велика кількість доступних ресурсів, включаючи обширну базу знань, відеоуроки та спільноту користувачів, щоб допомогти власникам бізнесу вирішувати проблеми та оптимізувати свої магазини.

Для компаній, яким потрібна додаткова підтримка, BigCommerce надає послуги преміум-класу, зокрема спеціалізовані менеджери облікових записів для клієнтів корпоративного рівня.

1.2.3 Square

Square [7] – це онлайн-платформа, розроблена для бізнесів, які функціонують переважно офлайн. Як і попередні аналоги, Square дозволяє торговцям створювати повністю настроюваний веб-сайт із вбудованими інструментами для керування запасами, а також обробки платежів і замовлень (рис. 1.3).

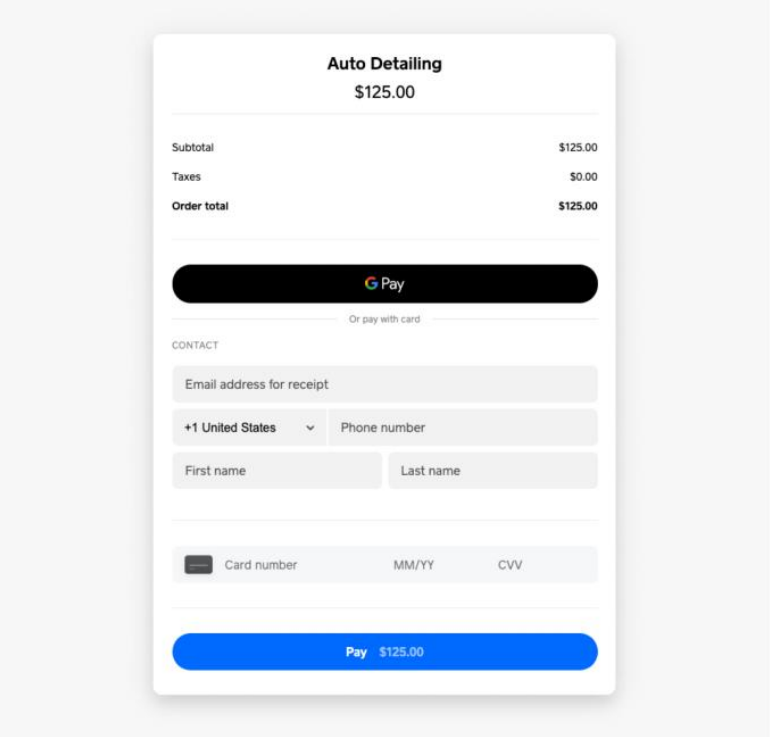

Рисунок 1.3 – Інтерфейс сторінки оформлення замовлення Square

Square – чудова система для малих підприємств, ресторанів та інших роздрібних магазинів, які зацікавлені в електронній комерції. Square підтримує онлайн-продажі через Square Online, що полегшує додавання функцій онлайн-замовлень на веб-сайт продавця. Ця служба автоматично синхронізує замовлення та інвентар із системою POS (Point of Sale).

Square Point of Sale – безкоштовна програма, яка не потребує щомісячної плати та витрат на запуск. Square надає безкоштовний доступ до стандартних функцій і є чудовим варіантом, якщо значна частина продажів бізнесу відбувається в реальному світі, а не онлайн.

Square робить прийом платежів безперешкодним і надає широкі можливості налаштування, що дозволяє підприємствам підтримувати численні варіанти оплати, програми лояльності, подарункові картки, виставлення рахунків, онлайн-продажі тощо. POS апаратне забезпечення є елегантним і привабливим, що робить його ідеальним для прийому мобільних платежів за допомогою чіп-карт і безконтактних платежів, таких як мобільні платежі NFC.

Однією з головних переваг Square є простота використання. Платформа також надає широкий спектр варіантів налаштування, що дозволяє компаніям адаптувати свій онлайн-магазин відповідно до цілей брендингу та клієнтського досвіду. Крім того, Square пропонує надійні інструменти аналітики, які дозволяють продавцям відстежувати ефективність продажів і поведінку клієнтів. Повна інтеграція з іншими продуктами Square спрощує управління бізнесом, а прозора структура ціноутворення робить платформу доступним рішенням для малог та середнього бізнесу.

1.3 Оцінка та порівняння існуючих аналогів

Під час аналізу аналогів додатків для оформлення замовлень, таких як Shopify, BigCommerce та Square, було виявлено деякі переваги та недоліки, а також специфіку використання для кожного випадку.

Shopify відомий своєю простотою використання, масштабованістю та розгалуженою екосистемою програм. Ця платформа пропонує настроюваний досвід та підходить для підприємств, які хочуть створити унікальний онлайн-магазин із певними функціями. Shopify включає велику кількість тем, як безкоштовних, так і платних, і забезпечує інтеграції із широким списком сторонніх програм, що робить цю платформу адаптованою для широкого спектру типів

бізнесу. Її ціна може бути вищою порівняно з іншими платформами, особливо якщо бізнес потребує програми та теми преміум-класу, але вона підтримує підприємства будь-якого розміру та є особливо зручною для проведення великих операцій, які прагнуть швидкого масштабування. Shopify також пропонує надійні інструменти для маркетингу, продажу, і підтримки клієнтів, що робить його універсальною платформою для підприємців і великих компаній.

BigCommerce зосереджується на наданні комплексних, багатофункціональних інструментів для компаній, які прагнуть розвиватися та масштабуватися. Ця платформа відома своїми сильними вбудованими функціями, зокрема управлінням продуктами, інструментами SEO та багатоканальними продажами (наприклад, інтеграція з Amazon, eBay і соціальними мережами). BigCommerce пропонує надійний набір інструментів для B2B і великомасштабних операцій без комісій за транзакції. Це особливо корисно для підприємств, які мають складні каталоги продуктів і комплексні налаштування, та не хочуть бути залежними від стороніх програми. Однак BigCommerce є більш комплексним порівняно із Shopify і може бути складним для розуміння для користувачів без технічних заннь. Також, ціна платформи залежить від доходу компанії, що може бути важливим фактором для підприємств, що мають більші обсяги продажів.

Square, у свою чергу, більше підходить для малих та середніх підприємств, особливо для тих, хто використовує систему POS для офлайн-продажів. Платформа електронної комерції Square є простішою та доступнішою, що робить її привабливим вибором для компаній, які хочуть розпочати роботу в Інтернеті із простим налаштуванням. Платформа легко інтегрується з POS Square, тож компанії, які вже використовують Square для фізичних магазинів, можуть легко перейти до продажів онлайн. Хоча Square пропонує менше розширених функцій порівняно з Shopify і BigCommerce, він забезпечує хороший баланс простоти використання, доступності та надійної базової функціональності електронної комерції. Це особливо корисно для компаній, які не потребують масштабних налаштувань або сторонніх інтеграцій, але все одно хочуть мати функціональний онлайн-магазин.

Отже, кожен із розглянутих аналогів має власні переваги та недоліки, а також специфічність у використанні. Розробка власного додатку для оформлення онлайн-замовлень може дати можливість врахувати і втілити кращі аспекти розглянутих прикладів та попрацювати над недоліками. Реалізація додатку прагне сприяти покращенню ефективності функціонування та якості обслуговування покупців.

2 ПРОЕКТУВАННЯ СТРУКТУРИ СИСТЕМИ ТА ОГЛЯД ІНСТРУМЕНТІВ РОЗРОБКИ

2.1 Огляд інструментів для веб-розробки

Для розробки застосунку для оформлення онлайн-замовлень, доцільно обрати інструменти та технології, що підтримують веб-розробку. Веб-розробка включає:

- клієнтську сторону (інтерфейс та логіка веб-додатку);
- серверну сторону (сервіс, до якого звертається веб-додаток);
- базу даних.

Інтерфейс часто створюється за допомогою HTML, CSS і JavaScript, де HTML забезпечує основну структуру сторінки, CSS керує стилем і швидкістю реагування, а JavaScript забезпечує інтерактивність, наприклад оновлення загальної ціни, коли елементи додаються або видаляються з візка. Для оптимізації процесу створення динамічних односторінкових додатків часто використовують такі фреймворки, як React, Angular або Vue.js, завдяки чому процес оформлення замовлення стає більш інтуїтивно зрозумілим і швидким. [8] Крім того, для плавної та надійної роботи додатку зазвичай впроваджують перевірку правильності форми за допомогою JavaScript або jQuery Validation, щоб гарантувати, що користувачі правильно вводять дані перед надсиланням. Це мінімізує помилки під час процесу оформлення замовлення та покращує взаємодію з користувачем.

Щоб забезпечити зручну взаємодію користувача із додатком, важливо розглянути адаптивний дизайн, щоб гарантувати, що сторінка правильно виглядає і працює на різних пристроях, незалежно від того, чи це смартфон, планшет або настільний комп’ютер. Такі інструменти, як Bootstrap або Foundation, можуть допомогти створити адаптивні макети із розробленими раніше компонентами.

Для керування бізнес-логікою каси, наприклад, розрахунок податків, знижок і доставки, застосовуються технології зі сторони серверу. Зазвичай запити з клієнтської сторони обробляються внутрішніми фреймворками або платформами,

як Node.js із Express, Ruby on Rails або Django для Python. Ці інструменти дозволяють створювати інтерфейси, які взаємодіють із веб-додатком, щоб оновлювати та отримувати необхідні дані, такі як інформація про клієнта, наявність інвентарю або деталі обробки платежів.

Для зберігання даних, використовуються бази даних, такі як MySQL, PostgreSQL або бази даних NoSQL, такі як MongoDB. [9] Ці бази даних працюють із логікою на стороні сервера, щоб гарантувати, що процес оформлення замовлення відстежує запаси, керує сеансами користувачів і записує транзакції покупки.

2.2 Обгрунтування вибору інструментів для веб-розробки

Найпопулярнішим вибором для сучасної веб-розробки є зокрема такі мови програмування, як JavaScript, PHP, Ruby.

Для розробки було обрано мову JavaScript, адже в порівнянні із PHP та Ruby, він має зручну структуру, легкий у вивченні та підтримує багато фреймворків, які дозволяють розширити функціонал та покращити його ефективність. З розвитком JavaScript з'явилося багато фреймворків для розробки веб-додатків, одним із найбільш популярних є React. [10] Цей фреймворк надає зручний підхід до створення багатофункціональних інтерфейсів, що легко масштабуються.

React конкурує з багатьма іншими популярними JavaScript фреймворками, такими як Angular, Vue.js, Svelte. [11] Кожен із них має свої сильні сторони, але React вигідно відрізняється у деяких аспектах, а саме у простоті та гнучкості у порівнянні з Angular та кращій у продуктивності в порівнянні з Vue.js.

Порівняння фреймворків для розробки клієнтської частини представлене у таблиці 2.1

Таблиця 2.1 – Порівняння фреймворків

	Продуктивність	Гнучкість	Інструменти тестування	Популярність
React	Висока	Висока (можна інтегрувати з іншими бібліотеками)	Jest, Enzyme, React Testing Library	Найбільша серед сучасних фреймворків
Vue.js	Висока	Висока	Vue Test Utils, Jest	Висока
Svelte	Висока	Менше гнучкості, але більше автоматизації	Svelte Testing Library, Jest	Зростаюча
Angular	Висока	Середня	Jasmine, Karma, Protractor	Висока

Отже, за характеристиками та вимогами веб-додатку, більше за все для розробки підходить React. React широко використовується для створення динамічних та інтерактивних сторінок, цей фреймворк зосереджений на продуктивності та зручності обслуговування.

Для розробки серверної частини було обрано фреймворк NestJS. [12] NestJS також має суттєві переваги порівняно з іншими фреймворками [13], а саме:

1. Масштабованість та впорядкованість. NestJS підтримує модульну архітектуру, що дозволяє розробникам легко організовувати програми в модулі, компоненти, служби та контролери.

2. Вбудована підтримка мікросервісів. У NestJS є зручний пакет, що обробляє найбільшу кількість комунікацій служб і підтримує низку транспортних протоколів, таких як gRPC.

3. Попередня обробка даних та безпека. NestJS пропонує засоби, що дозволяють розробникам перехоплювати вхідні запити та застосовувати логіку попередньої обробки до того, як вони досягнуть контролерів.

4. Підтримка Typescript. На відміну від простого Node,js, NestJS має вбудовану підтримку TypeScript, що робить розробку додатків простіше та усуває можливі проблеми завдяки типуванню даних.

5. Інтеграція із Swagger API. Swagger API дозволяє переглянути доступні ресурси та структуру запитів і відповідей у вигляді зручного графічного інтерфейсу прямо в браузері.

Бази даних поділяють на реляційні (SQL) та нереляційні (NoSQL). Для розробки було обрано нереляційну систему управління базами даних, адже такі СУБД відрізняються швидкістю та продуктивністю. Фізичні об’єкти у NoSQL зазвичай можна зберігати прямо у тому вигляді, у якому з ними потім працює додаток. [14]

Для управління даними було обрано СУБД MongoDB, яка дозволяє зберігати дані у вигляді JSON-подібних документів, що є знайомим виглядом для веб-розробників JavaScript. Зручність такої СУБД також полягає в тому, що на відміну від SQL, ми можемо вміщувати об’єкти або масиви в самому документі, легко редагувати структуру документів та їх залежності.

2.3 Проектування інтерфейсу

Проектування інтерфейсу користувача — це важливий етап у створенні програмного забезпечення, який визначає, як кінцевий користувач буде взаємодіяти з додатком або веб-сайтом. Цей процес включає в себе розробку елементів інтерфейсу, наприклад, кнопки, поля введення, меню, шрифти та кольори, а також планування того, як ці елементи будуть організовані на екрані для забезпечення зручності та ефективності користування додатком. Проектування інтерфейсу користувача є важливою складовою процесу розробки, оскільки без зручного та привабливого інтерфейсу будь-який додаток не може бути успішним. Ключовим є баланс між естетикою та функціональністю, а також постійне вдосконалення на основі відгуків користувачів.

До основних принципи проектування інтерфейсу користувача входить:

- Зрозумілість та інтуїтивність. Інтерфейс повинен бути зрозумілим і легким для навігації. Користувачі повинні без проблем розуміти, як виконувати дії, навіть якщо вони не мають досвіду в роботі з конкретним додатком. Ключові елементи інтерфейсу, такі як кнопки, іконки та меню, повинні бути інтуїтивно зрозумілими.

- Простота та мінімалізм. Чистий і простий дизайн допомагає користувачам швидко орієнтуватися в елементах додатку. Складні елементи та зайва інформація можуть відволікати від основної мети і створювати відчуття перевантаження, тоді як мінімалізм допомагає зберігати фокус на найважливіших функціях.

- Послідовність. Всі елементи інтерфейсу повинні виглядати і поводитися однаково на всіх сторінках або екранних блоках додатка. Це стосується кольорів, шрифтів, іконок і стилів кнопок. Консистентний дизайн сприяє зручності користування і допомагає уникнути плутанини.

- Адаптивність та респонсивність. Інтерфейс повинен бути адаптованим до різних пристроїв і екранів, наприклад: мобільні телефони, планшети та комп’ютерні монітори. Респонсивний дизайн забезпечує комфортне використання додатку на будь-якому пристрої та коректне відображення елементів залежно від розміру екрану.

- Залучення користувача. Дизайн повинен залучати користувача до взаємодії із застосунком. Це можна досягти через елементи, які сприяють інтересу користувача, як анімації, цікаві візуальні ефекти, переходи тощо.

До інструментів для проектування інтерфейсів користувача належать:

- Figma [15] — один з найбільш популярних інструментів для створення прототипів і дизайну інтерфейсів, дозволяє працювати в команді в реальному часі.

- Adobe XD [16] — потужний інструмент для створення інтерактивних прототипів та макетів, з можливістю інтеграції з іншими продуктами Adobe.

- Sketch [17] — один із лідерів у дизайні UI для MacOS, що активно використовується для створення інтерфейсів мобільних і веб-додатків.

- InVision [18] — платформа для створення інтерактивних прототипів і організації командної роботи над дизайном.

Зразок зручного дизайну, розробленого за допомогою Figma, можна побачити на рисунку 2.1. Для створення макета використовувалися сучасні підходи до UX/UI, зокрема адаптивна сітка, чітка типографіка та контрастні кольори для важливих елементів інтерфейсу (кнопок, полів вводу).

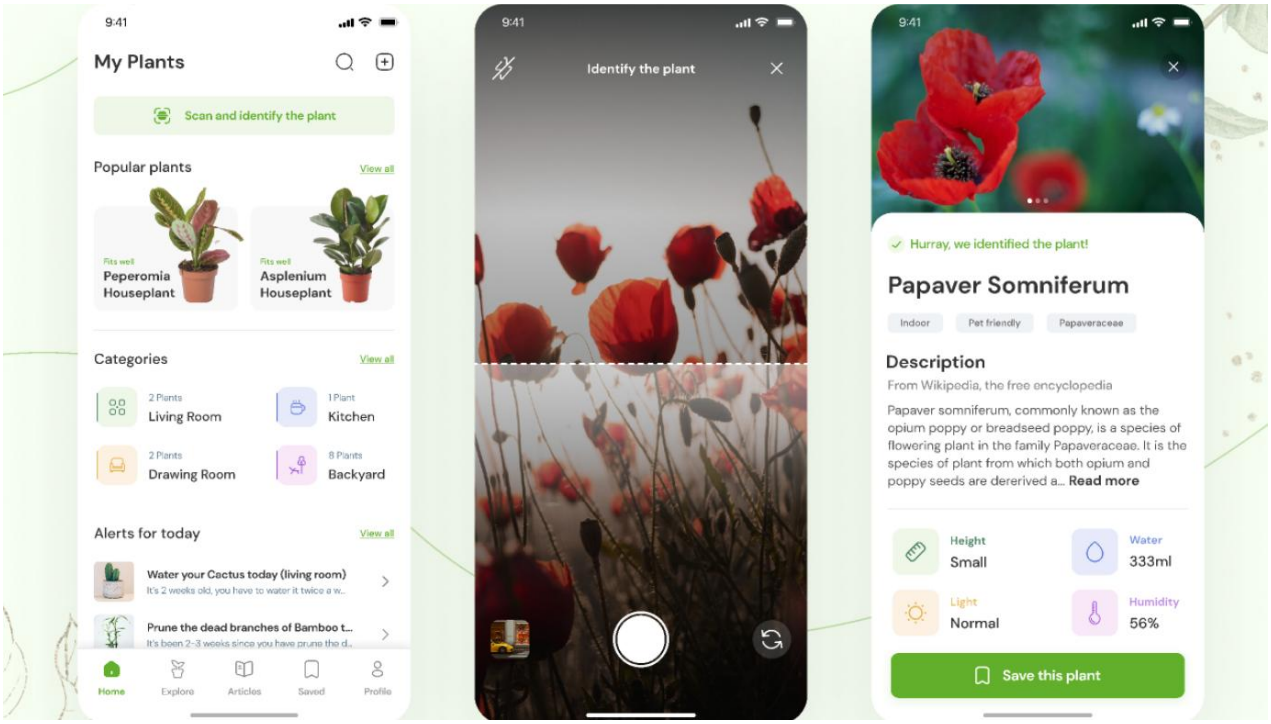

Рисунок 2.1 – Приклад користувацького інтерфейсу, створеного за допомогою інструменту Figma

На даному макеті видно, як відображаються стани елементів при наведенні курсора, активні та неактивні кнопки, а також підказки при помилках введення даних. Це забезпечує зрозумілу навігацію та зручність роботи з формами.

Спроектуємо користувацький інтерфейс для нашого додатку. Додаток матиме дві форми для заповнення – доставка та оплата, форми містимуть поля для заповнення, а також відображатимуть помилки при неправильному заповненні даних та індикатори успіху при правильному заповненні даних. Розроблений дизайн можна побачити на рисунку 2.2.

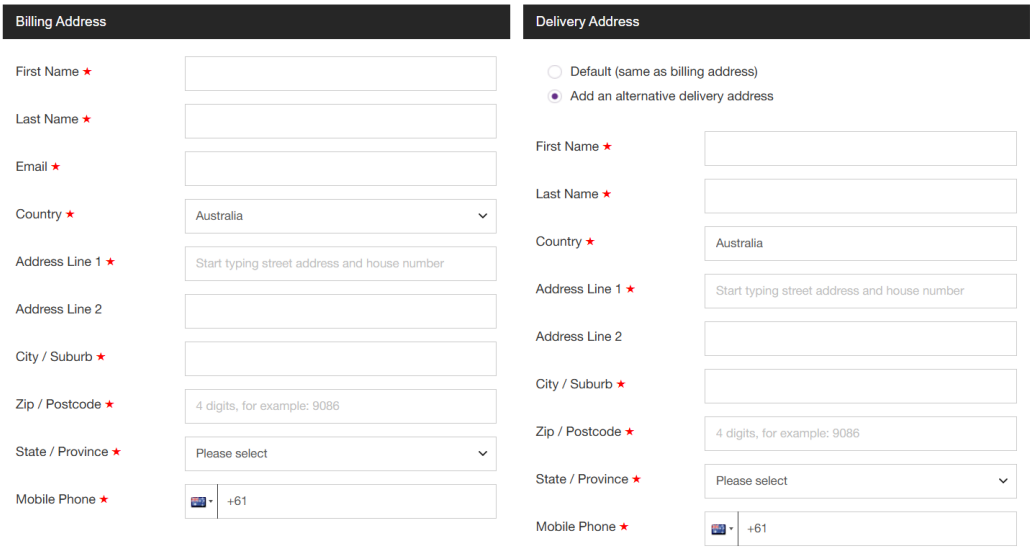

Рисунок 2.2 – Дизайн користувацького інтерфейсу

Форми міститимуть поля для заповнення, а також відображатимуть помилки при неправильному заповненні даних та індикатори успіху при правильному заповненні даних.

Наприклад, якщо користувач введе некоректний формат номера телефону, відповідне поле відразу підсвічується червоним, а під ним з’являється текст “Введіть номер у форматі +380XXXXXXXXX”. Після виправлення помилки поле підсвічується зеленим, що сигналізує про правильність введених даних.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка застосунку для оформлення онлайн-замовлень є комплексною задачею, яка вимагає ретельного вибору інструментів розробки. Для розробки ми обрали NestJS і React, адже в комбінації вони дозволяють створювати потужні, масштабовані та ефективні веб-додатки з високою продуктивністю, безпекою та простотою в підтримці. Поєднання цих технологій забезпечує легкість інтеграції клієнтської та серверної частини, високу продуктивність та підтримку більшості сучасних стандартів та практик розробки програмного забезпечення.

3.1 Розробка архітектури застосунку

Перед початком роботи, слід розробити архітектуру клієнтської та серверної сторін. Створення діаграм є важливим етапом у процесі розробки програмного забезпечення, адже вони дозволяють чітко й зрозуміло визначити архітектуру та взаємодію різних компонентів системи. Діаграми структури є необхідними за наступних причин:

- Чітке уявлення про архітектуру. Діаграми дозволяють розробникам, дизайнерам та іншим зацікавленим сторонам (наприклад, бізнес-аналітикам чи менеджерам) отримати візуальне уявлення про те, як система буде організована. Це допомагає зрозуміти, як різні частини додатку взаємодіють між собою.

- Планування та розподіл задач. Діаграми допомагають поділити систему на окремі компоненти, що дозволяє ефективно розподілити роботу та визначити пріоритети на кожному етапі. Вони дозволяють також оцінити складність окремих частин додатку.

- Модульність і масштабованість. Зрозуміла структура дозволяє вчасно виявляти модулі, які можуть бути перероблені або розширені в майбутньому. Це особливо важливо для забезпечення масштабованості додатку, що дає можливість згодом адаптувати систему до нових вимог без значних змін в архітектурі.

- Виявлення проблем на ранніх етапах. Рання розробка діаграм структури дозволяє виявити потенційні проблеми, недоліки в дизайні або невідповідність між компонентами, перш ніж ці проблеми призведуть до серйозних помилок у коді чи розробки на пізніших етапах.

- Полегшення тестування. Розуміння структури програми допомагає при написанні тестів, оскільки чітко видно, які компоненти необхідно тестувати і як вони повинні взаємодіяти.

- Документація та підтримка проєкту. Діаграми є важливою частиною технічної документації. Вони допомагають розробникам швидко зрозуміти, як працює система, що важливо для підтримки та розвитку проєкту.

Створені діаграми наведено на рисунках 3.1, 3.2.

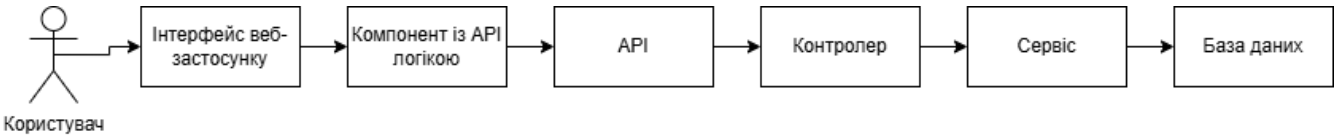

Рисунок 3.1 – Діаграма взаємодії компонентів системи між собою

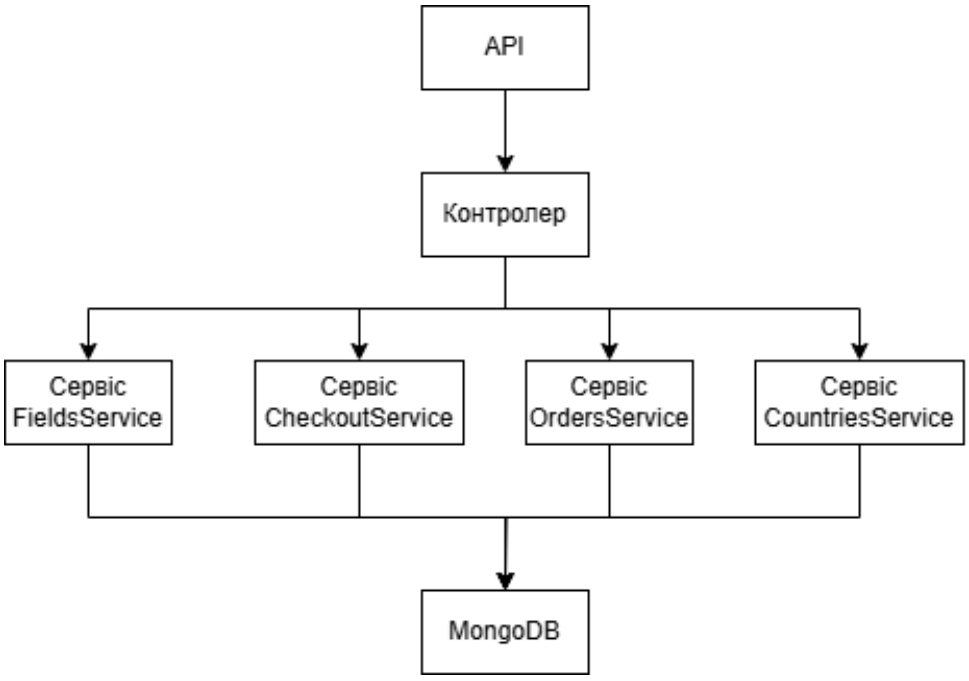

Рисунок 3.2 – Діаграма взаємодії компонентів серверної сторони між собою

3.2 Підготовка середовища для розробки

Перед початком розробки, важливо налаштувати середовище розробки, що включає: налаштування інтегрованого середовища розробки, вибір інструментів розробки та додаткових бібліотек.

Для розробки було обрано Visual Studio Code [19], адже це один із найбільш популярних та зручних інтегрованих середовищ розробки в даний час. Його перевагами, порівняно із, наприклад, близьким за популярністю Visual Studio, є менша потреба в ресурсах, більша легкість у розумінні та користуванні. Також Visual Studio Code підтримує безліч плагінів для покращення процесу розробки.

Як менеджер пакетів було обрано npm, як для клієнтської так і серверної частини, адже обидві частини застосунку підтримують цей інструмент. До інших переваг npm належить актуальність для невеликих проектів та проста структура команд.

Використання додаткових бібліотек є важливим аспектом розробки з наступних причин:

1. Пришвидшення та спрощення розробки. Завдяки використанню бібліотек, можна зекономити час, який пішов би на написання того ж функціоналу, його тестування та виправлення помилок.

2. Покращення функціональності. Популярними бібліотеками зазвичай користується велика кількість людей, а отже така бібліотека скоріше за все буде мати високу якість та невелику кількість або відсутність проблем із функціональністю.

3. Автоматична підтримка оновлень. Завдяки залежностям бібліотеки, із її оновленням завантажуються також оновлення її залежностей, що дає можливість не слідкувати за оновленнями самостійно.

Визначимо основний функціонал клієнтської та серверної сторін.

Клієнтська сторона:

- відображення сторінки оформлення замовлення;
- відображення форми для заповнення даних користувача та їх обробка;

- передача даних між компонентами, наявність певного сховища.

Серверна сторона:

- отримання та обробка даних про замовлення та користувача;
- взаємодія із базою даних;
- відображення наявних ресурсів у вигляді графічного інтерфейсу.

Для цього можна використати наступні бібліотеки:

Клієнтська сторона:

mui: Бібліотека з готовими для використання графічними компонентами.

styled-components: Бібліотека для стилізації компонентів.

zustand: Бібліотека для управління спільним сховищем додатку.

jest: Бібліотека для написання та виконання автоматизованих тестів.

typescript: Бібліотека для типізації.

Серверна сторона:

mongoose: Бібліотека для взаємодії з базою даних.

rxjs: Бібліотека для обробки асинхронних запитів.

axios: Бібліотека для створення та обробки http-запитів.

jest: Бібліотека для написання та виконання автоматизованих тестів.

typescript: Бібліотека для типізації.

swagger: Бібліотека для відображення доступних ресурсів у вигляді графічного інтерфейсу.

3.2.1 Клієнтська сторона

Створимо новий веб-застосунок на основі React і Typescript за допомогою утиліти create-react-app. Введемо наступну команду:

npx create-react-app checkout-client --template typescript

Створилася наступна структура папок (рисунок 3.3):

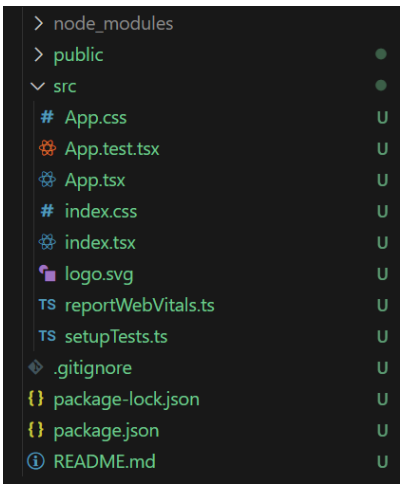

Рисунок 3.3 – Початкова структура проекту

Розглянемо створену структуру детальніше.

node_modules: Містить модулі встановлених залежностей, необхідних для запуску та роботи застосунку.

public: Містить файли властивостей сторінок застосунку, наприклад, іконка у веб-браузері.

src: Містить безпосередньо файли компонентів застосунку.

Встановимо необхідні залежності, наведені раніше. Застосуємо наступну команду:

npm i @mui/material styled-components zustand jest

3.2.2 Серверна сторона

Створимо новий серверний застосунок на основі NestJS. Для цього використаємо наступні команди:

npm i -g @nestjs/cli

nest new checkout-server

Створилася наступна структура папок (рисунок 3.4):

Рисунок 3.4 – Початкова структура проекту

Розглянемо створену структуру детальніше.

node_modules: Містить модулі встановлених залежностей, необхідних для запуску та роботи застосунку.

src: Містить безпосередньо файли сервісів та контролерів застосунку.

test: Містить файли автоматизованих тестів.

Встановимо необхідні залежності, наведені раніше. Застосуємо наступну команду:

npm i @nestjs/axios @nestjs/swagger @nestjs/mongoose mongoose rxjs jest

3.2.3 Система управління базами даних

Підготуємо також СУБД. Для розробки було обрано MongoDB. Для налаштування СУБД необхідно:

1. Перейти на сайт MongoDB [20] та завантажити останню версію СУБД (рис. 3.5)

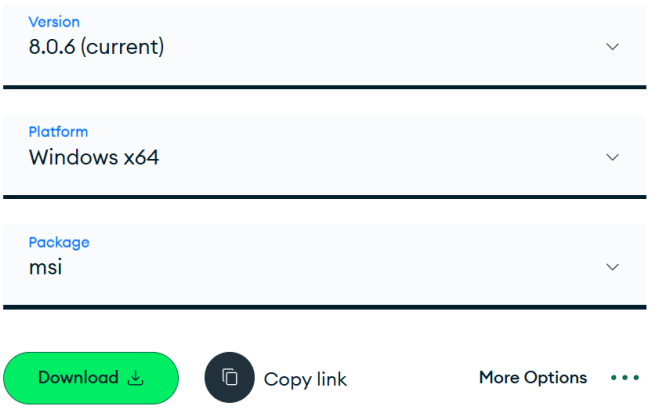

Рисунок 3.5 – Сторінка завантаження MongoDB

2. Відкрити завантажений файл та пройти усі кроки завантаження (рис. 3.6)

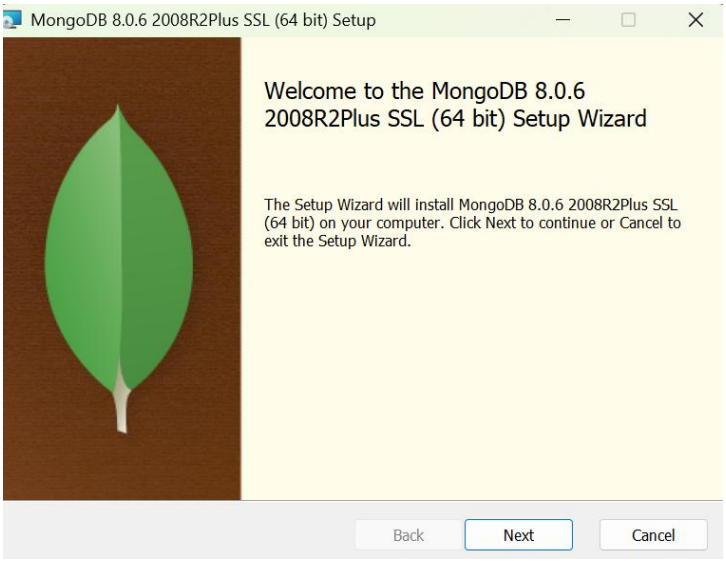

Рисунок 3.6 – Встановлювач MongoDB

3. Підключитися до серверу баз даних (рис. 3.7)

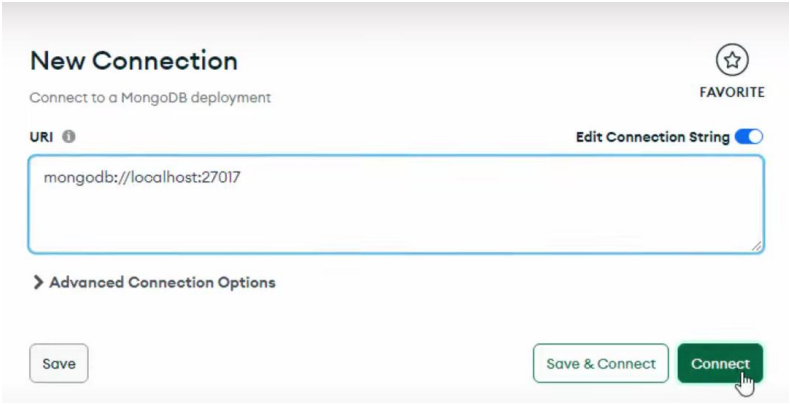

Рисунок 3.7 – Підключення до локального серверу баз даних

3.3 Практична реалізація застосунку

Отже, середовища для розробки підготовлені, тепер можна розпочати практичну реалізацію частин застосунку.

3.3.1 Сервер

Серверна сторона постачає дані про замовлення, доступні поля для заповнення, дані про користувача (якщо такі дані наявні), доступні дані для відповідних полей (наприклад, доступні для вибору країни або міста, регіони), правила заповнення полей.

Серверна сторона також отримує дані, які надав користувач, структурує їх, та передає для подальшої оброки.

Згідно зі створеними раніше діаграмами, ми маємо створити наступні компоненти серверної частини:

1. Контролер
2. Сервіси, що обробляють дані при отриманні та перед відправкою
3. Сервіси, що взаємодіють із базою даних

Створимо основний контролер. Контролер міститиме наступні ресурси (рис. 3.8):

GET checkout/countries – ресурс, який повертає всі доступні країни

POST checkout/countries/populate – ресурс, що популює країни в базі даних із наявних даних про країни

GET checkout/:id – ресурс, що повертає дані про сторінку (обрані товари, поля, правила заповнення полей тощо)

POST checkout/orders/ – ресурс, що створює замовлення при правильності заповнених даних

GET checkout/orders/:id – ресурс, що повертає деталі про замовлення після його створення

```
export class CheckoutController {

  @Get("/countries")
  getCountries() {
    return this.countriesService.get();
  }

  @Post("/countries/populate")
  populateCountries() {
    return this.countriesService.populate();
  }

  @Get(":id")
  async getData(@Param("id") id: string) {
    const checkoutData = await this.checkoutService.getData(id);
    const fields = await this.fieldsService.get();

    return { ...checkoutData, fields };
  }
```

Рисунок 3.8 – Частина контролеру

Опишемо моделі необхідних об’єктів та зареєструємо їх для використання (рис. 3.9):

```
@Module({
  imports: [
    HttpModule,
    MongooseModule.forFeature([
      { name: Field.name, schema: FieldSchema },
      { name: Order.name, schema: OrderSchema },
      { name: Country.name, schema: CountrySchema },
    ]),
  ],
```

Рисунок 3.9 – Зареєстровані моделі

Для моделі поля необхідно описати тип поля, щоб клієнтська частина знала, який саме елемент слід відображати для певного поля. Опишемо доступні типи полів (рис. 3.10):

```
export enum FieldType {
  Input = 1,
  Dropdown,
  DropdownWithSearch,
  Autocomplete,
  Phone,
}
```

Рисунок 3.10 – Доступні типи полів

Розробимо методи сервісів (рис. 3.11-3.14):

```
@Injectable()
export class CheckoutService {
  constructor(private readonly httpService: HttpService) {}

  async getData(checkoutId: string): Promise<CheckoutData> {
    const response: AxiosResponse<CheckoutData> = await firstValueFrom(
      this.httpService.get(`${EXTERNAL_CHECKOUT_DATA_API_URL}/${checkoutId}`),
    );

    return response.data;
  }
}
```

Рисунок 3.11 – Методи сервісу CheckoutService

```
@Injectable()
export class CountriesService {
  constructor(
    @InjectModel(Country.name) private countryModel: Model<Country>,
  ) {}

  get() {
    return this.countryModel.find();
  }

  async populate() {
    const existingCountries = await this.get();
    if (existingCountries.length === 0) {
      this.countryModel.insertMany(countriesData);
    }
  }
}
```

Рисунок 3.12 – Методи сервісу CountriesService

```
@Injectable()
export class FieldsService {
  constructor(@InjectModel(Field.name) private fieldModel: Model<Field>) {}

  get() {
    return this.fieldModel.find();
  }
}
```

Рисунок 3.13 – Методи сервісу FieldsService

```
@Injectable()
export class OrdersService {
  constructor(@InjectModel(Order.name) private orderModel: Model<Order>) {}

  create(createOrderDto: CreateOrderDto) {
    const newOrder = new this.orderModel(createOrderDto);
    return newOrder.save();
  }

  getById(id: string) {
    return this.orderModel.findById(id);
  }
}
```

Рисунок 3.14 – Методи сервісу OrdersService

Заповнимо таблиці MongoDB необхідними даними (рис. 3.15):

Рисунок 3.15 – Таблиця fields

3.3.2 Веб-застосунок

Веб-застосунок має обробляти наступний функціонал:

1. Користувач обирає товари з каталогу сайту, обирає країну для доставки, мову сайту тощо. Із корзини з обраними товарами користувач може перейти на сторінку оформлення замовлення та його оплати.

2. У випадку, якщо ця країна підтримується додатком або сам продавець вирішив, що ця країна має підтримуватися, дані про покупку, а також про мову і країну доставки, надсилається у запиті на сервер, щоб отримати дані для відображення сторінки оформлення.

3. На самій сторінці оформлення покупки користувач має ввести дані для відправки та оплати замовлення. Продавець також може налаштувати, які дані відображати як першорядні, а веб-додаток отримає та відобразить ці дані. Користувач може змінити мову сторінки також на цьому етапі.

4. Користувач має заповнити необхідні поля згідно обраної країни. Зазвичай, для усіх країн будуть присутні наступні поля:

- ім'я;

- прізвище;

- емейл-адреса;

- країна;

- місто;

- телефон.

5. Користувач може обрати однакові дані для оплати та відправлення, або ж внести нові, це відбувається за допомогою вибору відповідних кнопок для другорядної форми.

6. Якщо усі дані заповнено корректно, користувач може перейти до вибору типу доставки, а також типу оплати (кредитна картка, PayPal, Google Pay тощо). При коректності заповнення інформації, оплата проходить успішно, а дані замовлення надсилаються для подальшого опрацювання продавцю.

Розглянемо схему роботи сторінки, доступні поля для заповнення та наявний функціонал:

1. Звичайні текстові поля. Більшість полей будуть звичайними текстовими полями, наприклад, ім'я, прізвище, емейл-адреса.

2. Списки. Деякі поля, наприклад, країна, іноді - місто, штат, регіон міста, будуть випадними списками, з яких користувач може обрати потрібне значення.

3. Текстове поле з підказкою. Продавець може налаштувати поля, які пропонуватимуть підказку при введені користувачем тексту. Наприклад, поле адреси або поштовий індекс. При обранні однієї з опцій підказок, відповідні поля будуть заповнені відповідно до даних підказки (наприклад, ввевши "New York" і обравши опцію "New York, USA", поля адреси, штату, міста і поштового індексу будуть автоматично заповнені відповідно до даних підказки). Користувач також може обрати опцію ввести адресу самостійно, за натиском відповідної кнопки.

4. Список із пошуком. Функціонал схожий до звичайного списку, але такий список пропонує користувачу функціонал пошуку по доступним опціям, що полегшує заповнення форми.

5. Поле телефону. Це поле дозволяє користувачам ввести лише цифри. Також це поле має функціонал, завдяки якому користувач може знайти і обрати код необхідної країни.

Форма має також деякий допоміжний, особливий функціонал. Наприклад, якщо користувач має збережені раніше адреси для доставки, ми можемо відобразити їх у вигляді невеликого блоку із інформацією. У разі наявності декількох таких адрес, ми пропонуємо користувачеві вибрати одну із них за допомогою випадного списку. Якщо ж адреса заповнена некоректно, ми одразу показуємо користувачеві поля, які потребують редагування.

Отже, розіб’ємо функціонал на компоненти:

1. Компонент контейнеру форми. Він міститиме всі елементи, які стосуються поданої форми, а саме поле назви форми, поля для заповнення, збережені адреси, кнопки зміни форми.

2. Компонент поля. Клієнтська сторона отримуватиме дані про поля, які необхідно відобразити та відображатиме їх відповідно до типу та атрибутів поля для користувача.

3. Компонент кнопок зміни форми. Якщо покупець бажає, він може не заповнювати другорядну форму, а просто обрати опцію «такі ж дані, як і у першорядній формі». При цьому ми приховуватимо другорядну форму та встановлюватимо для другорядної форми ті ж дані, які користувач заповнив у першорядній формі.

4. Компонент відображення та зміни збереженої адреси. Якщо користувач зареєстрований у інтернет-магазині та має збережені адреси, ми відображатимо їх із можливістю переглянути дані для впевнення у їх правильності та можливістю зміни, якщо певні дані не відповідають дійсності.

Розробимо ці компоненти (рис. 3.16-3.18):

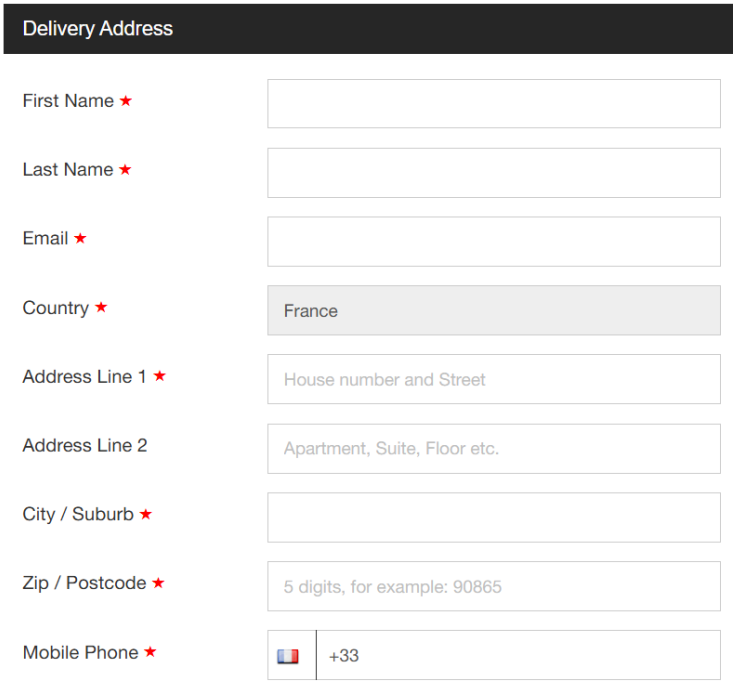

Рисунок 3.16 – Компоненти контейнеру форми та полів

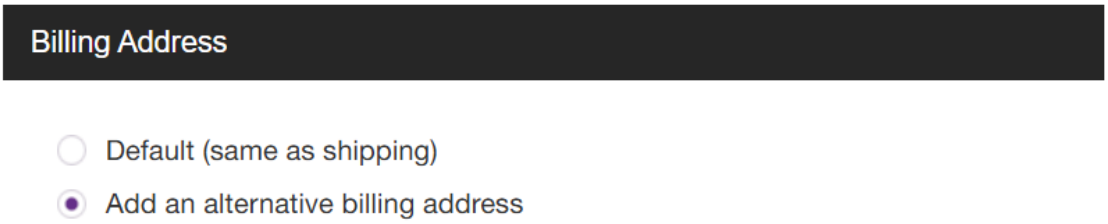

Рисунок 3.17 – Компонент зміни типу другорядної форми

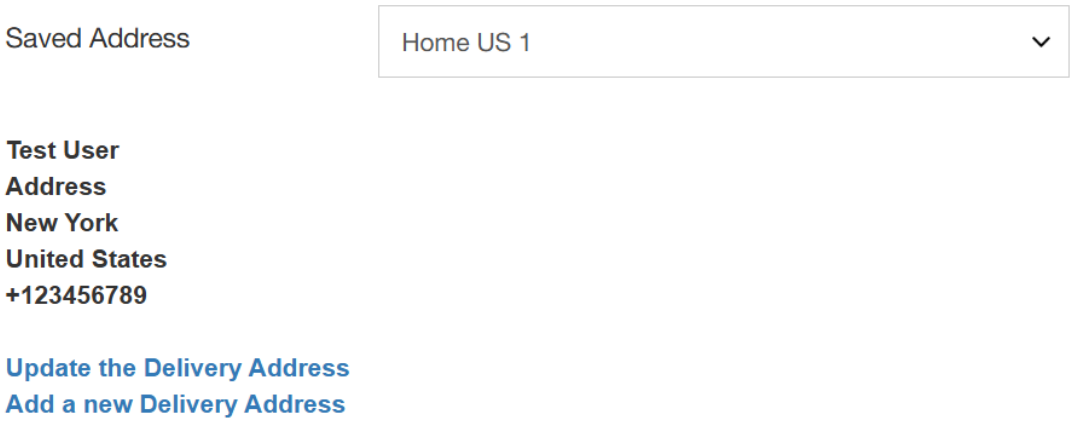

Рисунок 3.18 – Компонент відображення та зміни збереженої адреси

3.3.3 Взаємодія клієнтського застосунку із сервером

Взаємодія веб-застосунку із сервером відбуватиметься за допомогою бібліотеки Axios та REST API.

Axios [21] – це бібліотека для роботи з HTTP-запитами, яка має багато зручностей у порівнянні зі звичайним fetch, що вбудований у браузер. Основні переваги Axios:

1. Простіший у використанні. Axios автоматично перетворює відповіді з сервера у зручний формат (наприклад, JSON), тоді як у fetch це потрібно робити вручну.

2. Краще обробляє помилки. Якщо сервер повертає помилку (наприклад, сторінку не знайдено або внутрішня помилка), Axios одразу показує це як помилку. Fetch при цьому вважає, що запит вдалий, і треба додатково перевіряти статус відповіді.

3. Зручніше працювати з заголовками. Axios дозволяє легко додавати додаткові налаштування, наприклад, токени для авторизації або інші заголовки.

4. Є підтримка таймаутів. У Axios можна задати обмеження часу, скільки чекати на відповідь. Якщо таймаут спливе, запит зупиниться. У fetch це зробити складніше.

5. Можна налаштувати окремі «екземпляри». Це зручно gри частій роботі з одним API – можна один раз вказати всі налаштування і не повторювати їх щоразу.

REST — абревіатура від REpresentational State Transfer (передача репрезентативного стану). RESTful API — це архітектура інтерфейсу прикладних програм (API), що використовує HTTP-запити для доступу до ресурсів та їхнього використання. Такими HTTP-запитами є GET, PUT, POST і DELETE. Вони дають змогу, відповідно, читати, змінювати, створювати й видаляти ресурси. [22]

До переваг використання RESTful API належать:

- Масштабованість. Завдяки відокремленню клієнта від сервера групи розробників можуть масштабувати продукт без особливих проблем.

- Гнучкість і переносимість. Оскільки API-інтерфейси в стилі REST вимагають правильного надсилання даних у запитах, можна виконувати міграцію з одного сервера на інший. Також можна будь-коли вносити зміни до бази даних.

- Незалежність. Завдяки відокремленню клієнта від сервера протокол спрощує незалежну розробку в межах проєкту. Також API REST адаптуються до синтаксису й платформи, що дає змогу одночасно тестувати кілька середовищ під час розробки.

- Легкість. API-інтерфейси REST легкі та швидкі, оскільки використовують стандарт HTTP, що підтримує декілька форматів, включно з JSON, XML і HTML. Це робить REST ідеальним для мобільних застосунків, пристроїв Інтернету речей та багатьох інших застосувань.

Проблеми використання RESTful API:

- Узгодженість. Узгодженість API має вирішальне значення. У великій базі коду, створеній багатьма розробниками, може бути складно досягти узгодженості.

- Керування версіями RESTful API. Щоб запобігти проблемам сумісності, часто створюють версії API. Однак старі кінцеві точки залишаються активними, що спричиняє збільшення робочого навантаження, оскільки підтримується декілька API.

- Аутентифікація RESTful API**.** Аутентифікація API залежить від контексту використання. Може бути використано понад 20 різних підходів до авторизації, що ускладнює завдання зробити перший виклик API.

- Безпека RESTful API. Попри простоту доступу, все одно можуть виникнути проблеми безпеки. Наприклад, клієнт надсилатиме тисячі запитів щосекунди, що призведе до збою сервера.

- Множинні запити й непотрібні дані. Відповідь може містити більше даних, ніж потрібно, або вимагати додаткових запитів для доступу до всіх даних.

4 ТЕСТУВАННЯ ДОДАТКУ

Тестування додатків — це ключовий етап у розробці програмного забезпечення, який часто визначає успіх чи провал продукту. Тестування є важливим етапом за наступних причин:

- Виявлення помилок на ранньому етапі. Тестування дозволяє знайти баги та недоліки ще до того, як додаток потрапить до користувача. Це значно дешевше і простіше, ніж виправляти проблеми після релізу.

- Покращення якості продукту. Завдяки тестуванню розробники можуть переконатися, що всі функції працюють згідно з вимогами. Це підвищує загальну стабільність і надійність додатку.

- Безпека. Особливо актуально для фінансових, медичних або будь-яких додатків, які працюють з персональними даними. Тестування допомагає виявити вразливості, які можуть бути використані зловмисниками.

- Підвищення довіри користувачів. Користувачі очікують, що додаток буде працювати правильно за будь-яких обставин. Якщо продукт часто зависає, вилітає або поводиться непередбачувано — користувачі швидко втрачають довіру.

- Забезпечення стабільності після оновлень. Автоматизоване тестування дозволяє швидко перевірити, чи не зламали нові зміни щось у вже працюючому функціоналі (регресійне тестування).

- Покращення продуктивності розробки. Коли розробники мають налагоджений процес тестування, це зменшує кількість неочікуваних проблем і дозволяє зосередитись на розробці нового функціоналу, а не на виправленні старих проблем.

4.1 Види тестування

У процесі створення додатків існує багато різних видів тестування, і кожен із них має свою мету. Щоб зробити якісний продукт, важливо перевіряти його з усіх боків — як він працює, як взаємодіє з іншими частинами системи, наскільки

швидко реагує, чи зручно ним користуватись тощо. Розглянемо наступні види тестування:

- Модульне тестування. Це коли тестують окремі частини коду — наприклад, окрему функцію чи компонент. Мета — переконатися, що кожен компонент сам по собі працює правильно.

- Інтеграційне тестування. Цей тип тестування має за мету перевірку, як різні частини програми працюють разом. Наприклад, чи правильно клієнтська частина обмінюється даними з сервером.

- Системне тестування. На цьому етапі тестують вже готову систему в цілому, як один великий механізм. Це допомагає побачити, чи відповідає додаток усім вимогам, які ставилися під час розробки.

- Регресійне тестування. Коли в коді щось змінюється або додається новий функціонал, потрібно перевірити, чи все інше залишилося працювати, як раніше. Це і є регресійне тестування.

- Функціональне тестування. Це перевірка того, чи виконує програма ті функції, які від неї очікуються. Наприклад, чи працює кнопка реєстрації, чи правильно обробляється форма зворотного зв’язку тощо.

- Нефункціональне тестування. Тут перевіряється не стільки "що" програма робить, а "як". Наприклад, наскільки швидко вона працює, як справляється з великим навантаженням, чи безпечно обробляє дані, чи зручно нею користуватись.

- Тестування інтерфейсу. Це перевірка того, як виглядає і поводиться додаток для користувача. Тут важливо, щоб усе було зрозуміло, красиво, й усі елементи були там, де користувач очікує їх побачити.

- Ручне і автоматизоване тестування. Тестування може бути ручним — коли тестувальник сам перевіряє все, або автоматизованим — коли використовуються спеціальні програми, які запускають тести автоматично. Зазвичай обирають комбінацію обох підходів.

- Smoke і Sanity тестування. Це швидкі перевірки. Smoke-тестування — це перевірка на «чи все взагалі запускається». Sanity — чи конкретна функція взагалі працює після зміни.

- Тестування прийняття. Це фінальна перевірка, яку зазвичай проводить замовник або кінцевий користувач. Вони дивляться, чи відповідає продукт їхнім очікуванням і потребам.

- Тестування збоїв і відновлення. Цей тип перевіряє, як додаток поводиться у критичних ситуаціях — наприклад, якщо зникне інтернет або впаде сервер. Також тестується, як програма відновлюється після таких збоїв.

Основні принципи тестування — це правила, які допомагають створювати якісне програмне забезпечення та ефективно перевіряти його. Вони лежать в основі будь-якої стратегії тестування і є важливими для того, щоб досягти максимального покриття і мінімізувати ризики виникнення помилок. Деякі з ключових принципів тестування:

- Тестування має бути розпочато якомога раніше. Тестування повинно розпочинатися на ранніх етапах розробки, ще до того, як код буде завершено. Це дозволяє виявляти помилки і проблеми на початкових етапах, коли їх можна виправити без значних витрат часу і ресурсів. Якщо тестування почати на пізнішому етапі, може бути складно знайти причини помилок, а виправлення може вимагати значних змін у коді.

- Тестування має проводитись з урахуванням вимог. Тестування повинно базуватися на вимогах до програмного продукту, які були визначені на етапі планування. Це дозволяє визначити, чи відповідає розроблений додаток усім очікуванням і специфікаціям замовника, а також чи працює програма так, як повинна. Всі функціональні і нефункціональні вимоги повинні бути перевірені через відповідні тести.

- Тестування не може гарантувати відсутність усіх помилок. Ідея про те, що тестування може повністю усунути всі помилки у програмному забезпеченні, є міфом. Навіть ретельно протестоване ПЗ може містити помилки, особливо у складних системах. Тестування дозволяє знизити кількість помилок до мінімуму, але не може повністю їх усунути. Важливо зрозуміти, що тестування — це не лише перевірка коду, а й постійний процес поліпшення і вдосконалення продукту.

- Тестування повинно бути систематичним і планомірним. Тестування не повинно бути випадковим або хаотичним. Воно має відбуватись за чітким планом, з визначеними цілями, типами тестів і критеріями успіху. Це дозволяє зменшити ймовірність пропуску важливих тестових випадків та досягти кращих результатів у виявленні помилок. Наявність тестової документації, зокрема тестових планів і випадків, допомагає здійснити тестування на систематичній основі.

- Тестування має бути незалежним**.** Незалежність тестувальників є важливим принципом, адже вони мають виявляти проблеми, яких розробники можуть не помітити. Тестувальник часто має зовнішній погляд на систему, що допомагає виявити баги, які можуть бути пропущені командою розробки. Це дозволяє зберігати об'єктивність і забезпечує більш ретельне тестування.

- Тестування повинно бути повторюваним. Тестування не є одноразовим процесом. Воно повинно проводитись на різних етапах розробки, а також після кожного оновлення або зміни коду. Регресійне тестування — це ключовий момент, оскільки зміни в одній частині додатку можуть вплинути на роботу іншої частини, і це потребує постійної перевірки.

- Тестування повинно охоплювати як позитивні, так і негативні сценарії. Позитивне тестування перевіряє, чи працює система згідно з вимогами в ідеальних умовах. Негативне тестування, у свою чергу, перевіряє, як система поводиться у ситуаціях, коли вводяться некоректні або неочікувані дані, чи система здатна правильно обробляти помилки.

- Тестування має бути незалежним від середовища. Тестування повинно бути незалежним від операційної системи, браузера, пристрою чи інших зовнішніх факторів, що можуть впливати на функціональність додатку. Це означає, що програма повинна однаково добре працювати в різних умовах, і тестування повинно перевіряти її в таких різноманітних середовищах.

- Тестування має бути орієнтованим на користувача. Програмне забезпечення створюється для людей, тому важливо проводити тестування з урахуванням досвіду кінцевого користувача. Тестування інтерфейсу та перевірка зручності

використання — це важливі елементи тестування, які допомагають зробити продукт не лише функціональним, але й зручним для людей.

4.2 Тестування запитів

Swagger — це інструмент, який допомагає розробникам створювати, описувати та тестувати API (інтерфейси програмування додатків). Swagger дозволяє:

- Описати, як саме працює API — які є запити, які дані треба передавати, і що повертається у відповідь.

- Тестувати API прямо в браузері — не треба писати код, достатньо натиснути кнопку.

- Зробити документацію зрозумілою — навіть якщо розробник не писав цей API, він може швидко зрозуміти, як ним користуватися.

Swagger зазвичай працює разом із файлом swagger.json або swagger.yaml, де описані всі запити API. У нашому випадку ми використаємо версію з графічним інтерфейсом — Swagger UI (рис.4.1).

Тестування запитів у Swagger означає перевірку того, як працює API прямо через веб-інтерфейс. Він дозволяє вводити дані, надсилати запити й одразу бачити відповіді від сервера. Це дозволяє впевнитися, що API працює правильно та повертає очікувані результати.

Необхідність тестувати API через Swagger UI полягає в наступному:

- Швидка перевірка функціональності Swagger дозволяє миттєво перевірити, чи працює конкретний ресурс. Це зручно для виявлення помилок на ранніх етапах, перевірки, чи правильно налаштовані параметри (наприклад, id, token, body) та розуміння, що саме повертає API — формат, структура, помилки.

- Прозорість для команди. Swagger — це одночасно і документація, і інструмент тестування. Це значить, що фронтенд-розробник може сам протестувати API без звернення до бекенду, а тестувальники можуть перевірити роботу запитів без складних інструментів.

- Простіше проводити дебагінг. Якщо запит не працює — в Swagger UI можна відразу побачити, яку відповідь віддає сервер (успіх чи помилка), що саме було надіслано (headers, тіло запиту), а також чи проблема в даних чи у логіці.

- Підготовка до автоматизованого тестування. Swagger підтримує OpenAPI-стандарт, тож опис API може використовуватися для генерації автотестів або мок-серверів. Але перед автоматикою завжди важливо зробити ручне тестування — переконатися, що все працює базово.

До переваг тестування за допомогою Swagger належать:

- інтерактивне тестування прямо в браузері;

- реальна перевірка всіх типів запитів;

- синхронізована документація та тестування;

- швидке виявлення помилок;

- Swagger UI можна розгорнути як окремий веб-сервіс — дати доступ клієнтам або партнерам;

- автоматизована генерація запитів;

- основа для автоматичного тестування.

До недоліків тестування за допомогою Swagger належать:

- потрібно вручну запускати кожен запит;

- відсутня перевірка логіки бізнес-процесів;

- залежність від правильності опису API;

- обмежена перевірка сценаріїв з авторизацією;

- не показує всі технічні деталі;

- є потреба певної конфігурації.

Тестування запиту за допомогою Swagger дає змогу впевнитися в тому, що API працює відповідно до очікувань. Розглянемо тестування запиту GET http://localhost:3000/fields/UA (рис. 4.2). Ми отримали успішний результат тестування, що дозволяє перевірити наступні параметри:

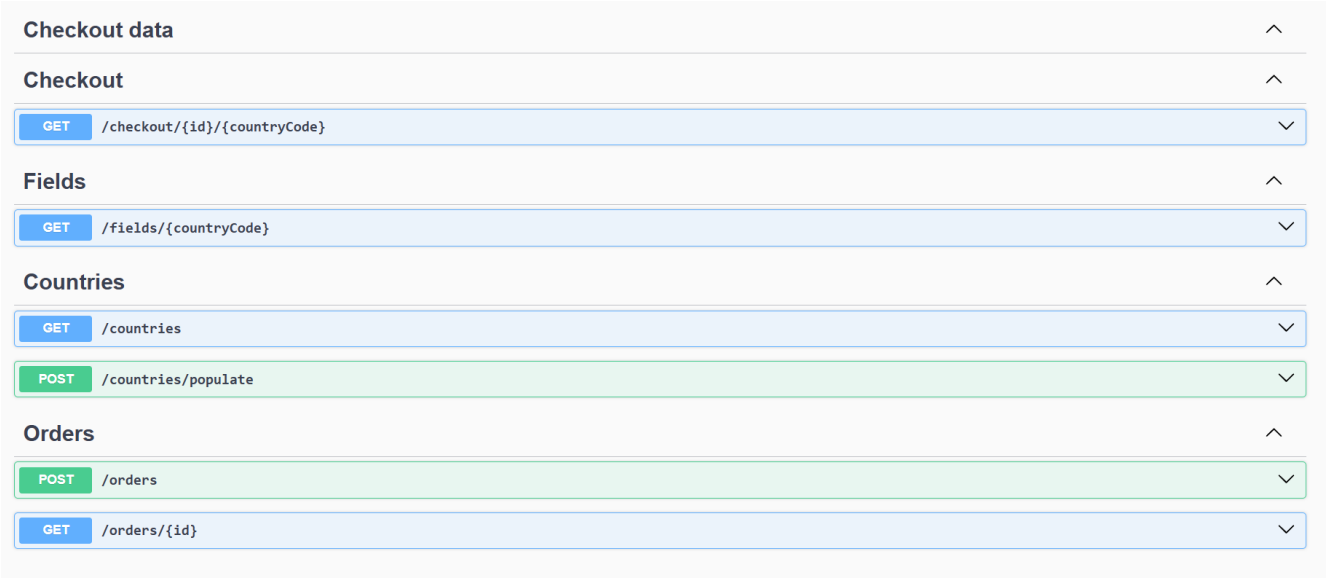

Рисунок 4.1 – Графічний інтерфейс Swagger UI

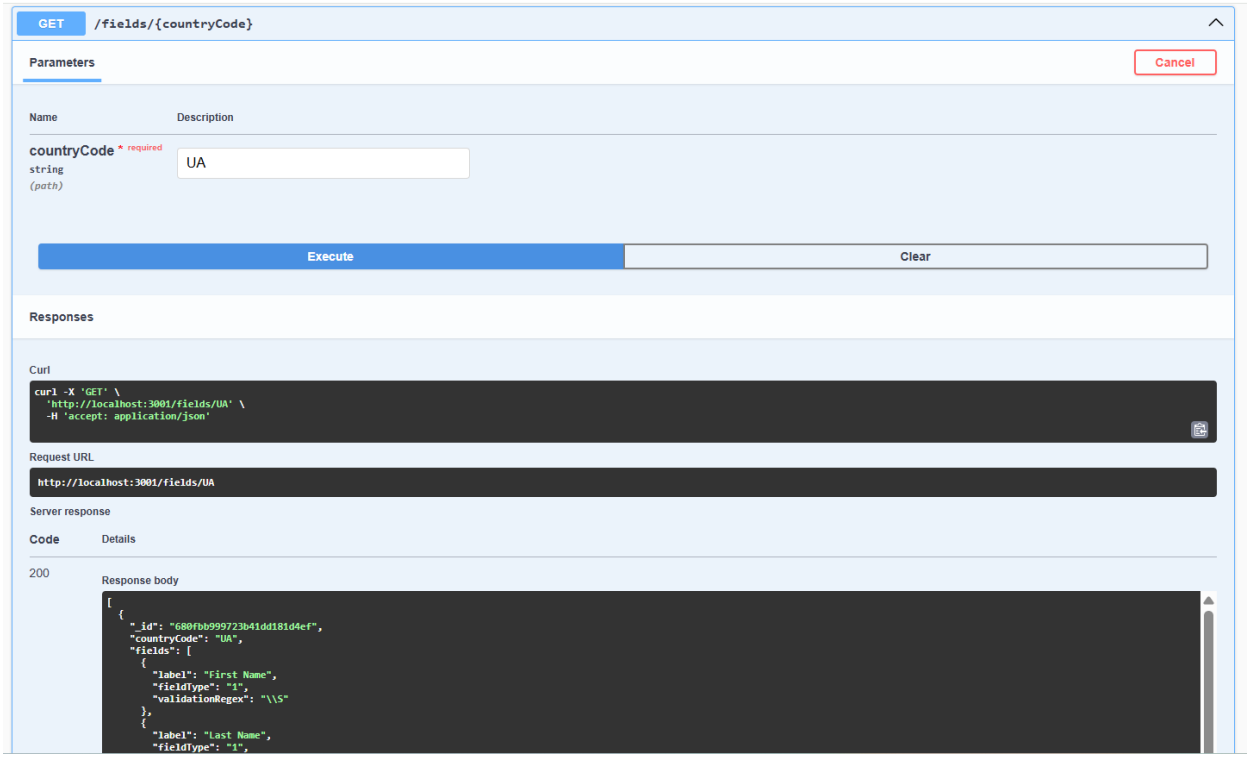

Рисунок 4.2 – Приклад виконання одного з запитів

1. Перевірка структури відповіді. Тестування дозволяє переконатися, що структура відповіді відповідає очікуваній. У даному випадку ми перевіряємо поля за кодом країни, які містять властивості label, fieldType, validationRegex.

2. Перевірка роботи сервісів та бази даних. Запит повертає інформацію про поля за кодом країни, що допомогає переконатися, що сервіси працюють коректно і мають доступ до бази даних.

3. Перевірка коректності повернення даних. У даному випадку відповідь містить коректні дані для коду країни UA. Якщо у запиті ми надішлемо некоректний код країни, відповідь міститиме пусте значення.

4.3 Тестування можливостей додатку та порівняння з аналогами

Перевірка можливостей додатку є ключовим етапом у розробці програмного забезпечення. Вона охоплює тестування функціональності, продуктивності та безпеки, щоб гарантувати коректну роботу додатку та відповідність вимогам користувачів. Було здійснено тестування наступних частин:

1. Перегляд обраних продуктів

Тестування функції перегляду обраних продуктів включає перевірку відображення основної інформації про продукт, а саме: назва, опис, розмір, колір, ціна тощо (рис. 4.3).

2. Перегляд та заповнення особистих даних

Тестування функції перегляду та заповнення особистих даних включає перевірку функціональності, зручності та рівня зворотного зв’язку полів для заповнення особистих даних, наприклад, ім’я замовника, електрона адреса, країна, мобільний телефон тощо (рис. 4.4).

Order Summary		Quantity	Price	Total
	Activa Sonar X Invicta Men's Watch - 45.5mm, Black (ACW423-002)	1	AED 105.00	AED 105.00
	Invicta Pro Diver Men's Watch - 48.8mm, Steel (0879)	1	AED 930.00	AED 930.00
NO IMAGE FOUND	Shipping Insurance	1	AED 16.00	AED 16.00
			Items total	AED 1051.00

Рисунок 4.3 – Інтерфейс секції перегляду обраних продуктів

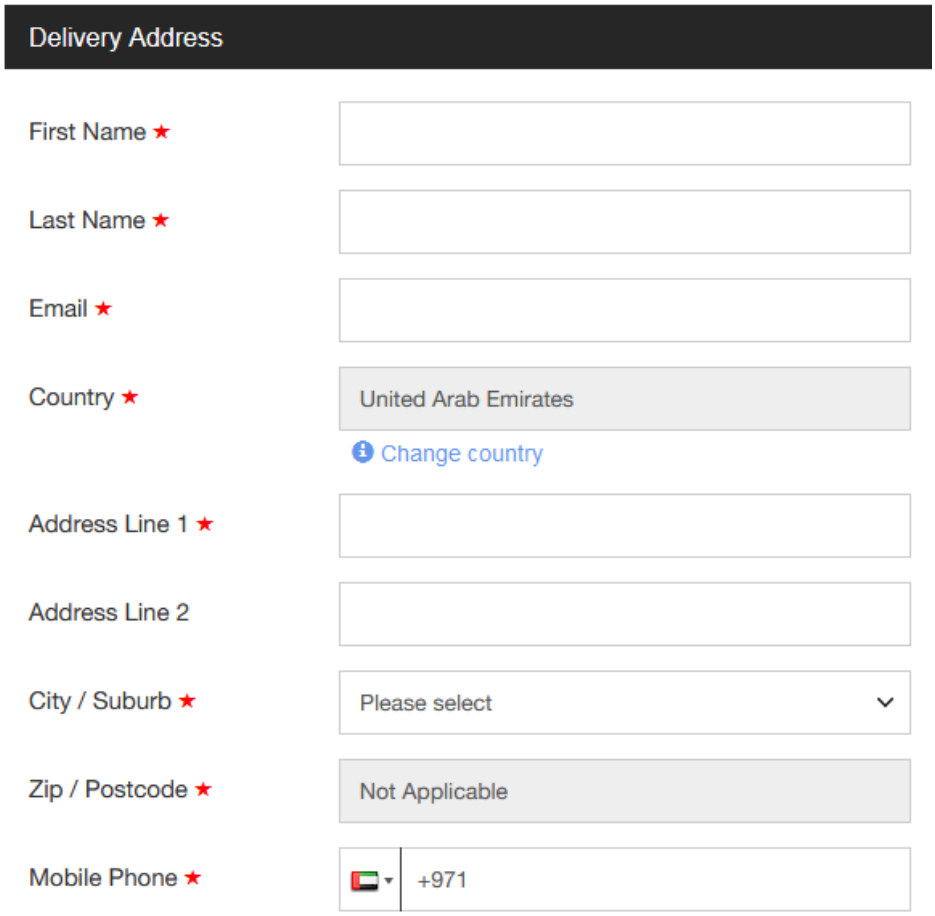

Рисунок 4.4 – Інтерфейс полів для заповнення особистих даних

Потребує перевірки також наступний функціонал: відображення повідомлень про коректність заповнення даних, оновлення полів при зміні країни, вибір країни в полі мобільного номеру.

Відображення повідомлень про коректність заповнення даних (рис. 4.5):

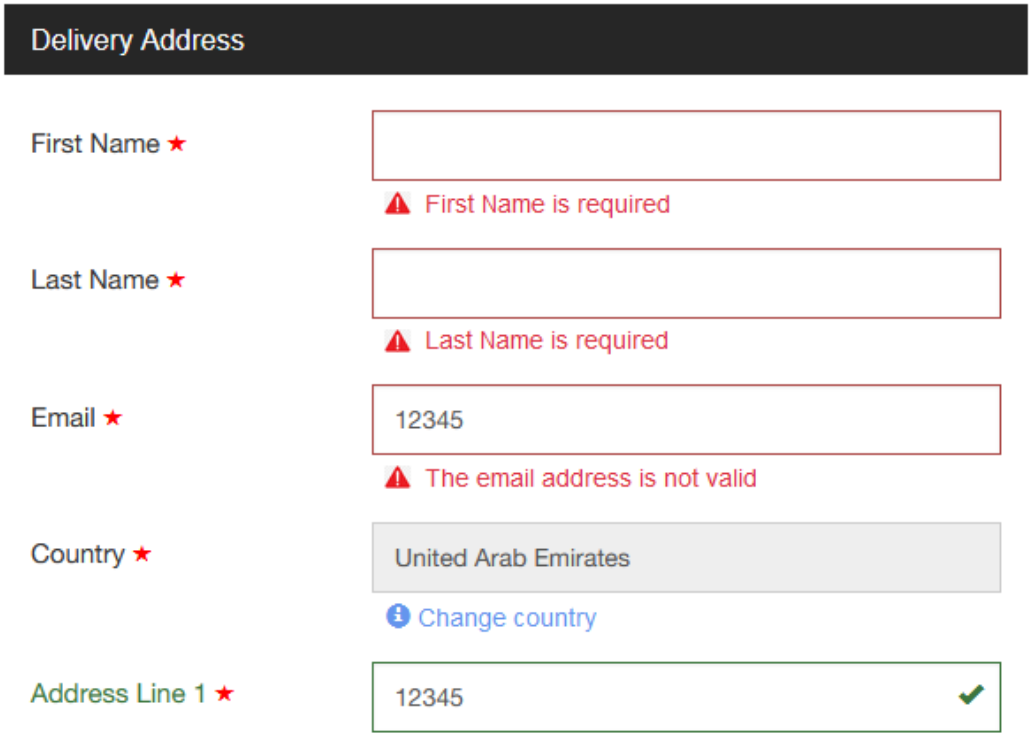

Рисунок 4.5 – Інтерфейс повідомлень про коректність заповнення даних

Оновлені поля, що завантажуються при зміні країни доставки (наприклад, нове поле «Штат» для Сполучених Штатів Америки) наведено на рисунку 4.6.

Вибір країни у полі вводу мобільного номеру наведено на рисунку 4.7.

Тестування функціоналу збережених адрес включає перевірку можливості перегляду даних збереженої адреси, відображення повідомлень про помилки або недостатнє заповнення даних, редагування даних адреси, додавання нової адреси.

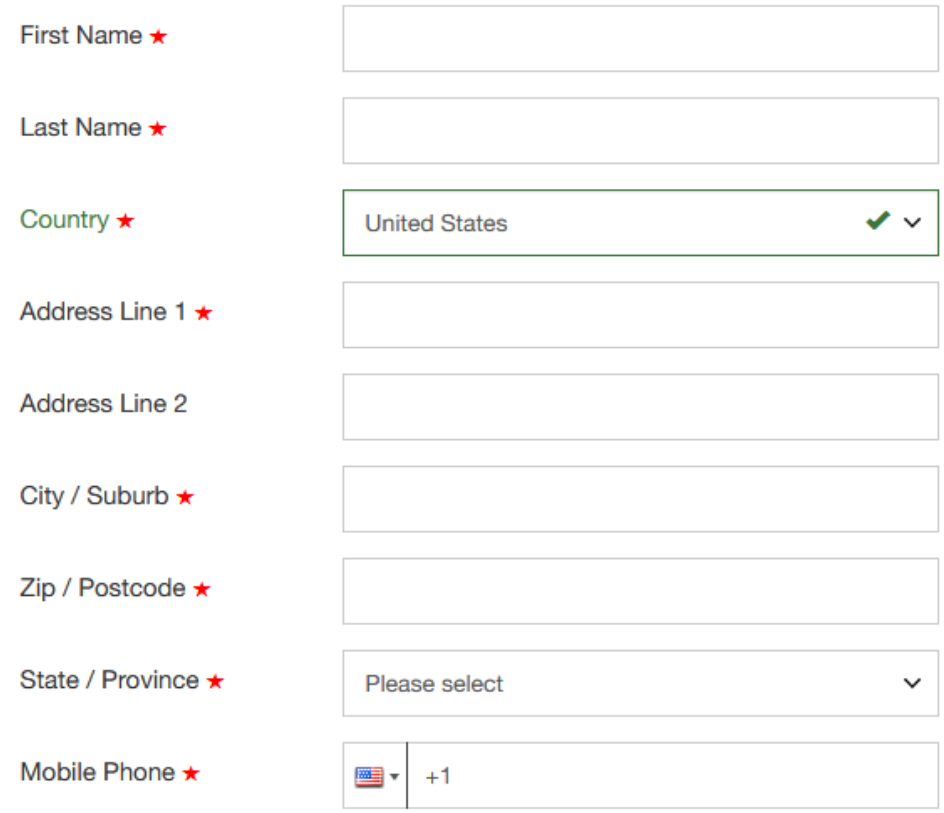

Рисунок 4.6 – Інтерфейс оновлених полів після вибору нової країни доставки

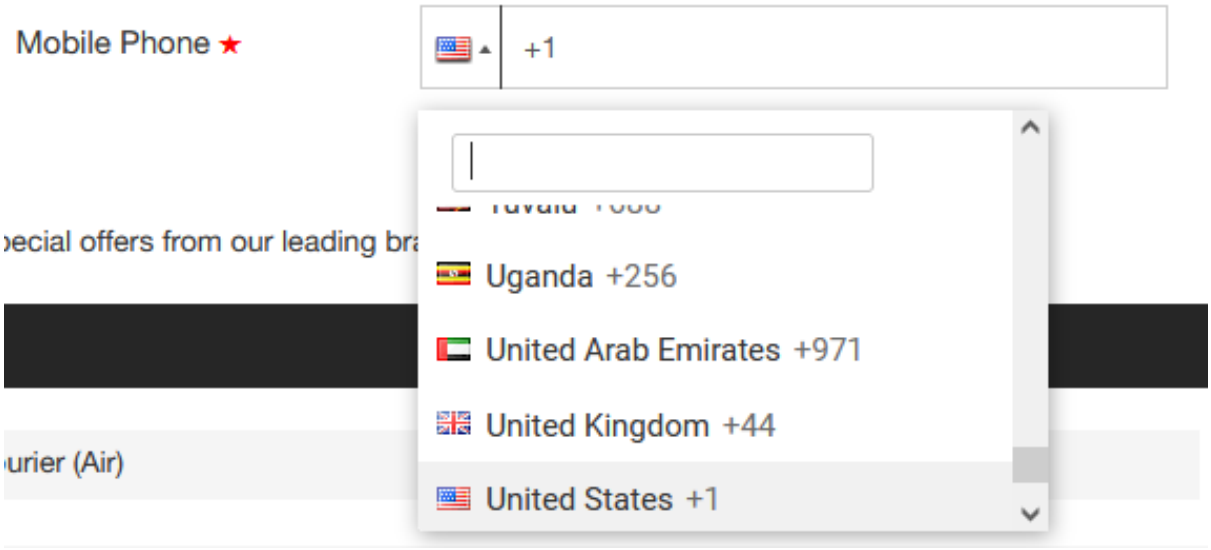

Рисунок 4.7 – Інтерфейс вибору країни у полі вводу мобільного номеру

Перегляд даних збереженої адреси (рис. 4.8):

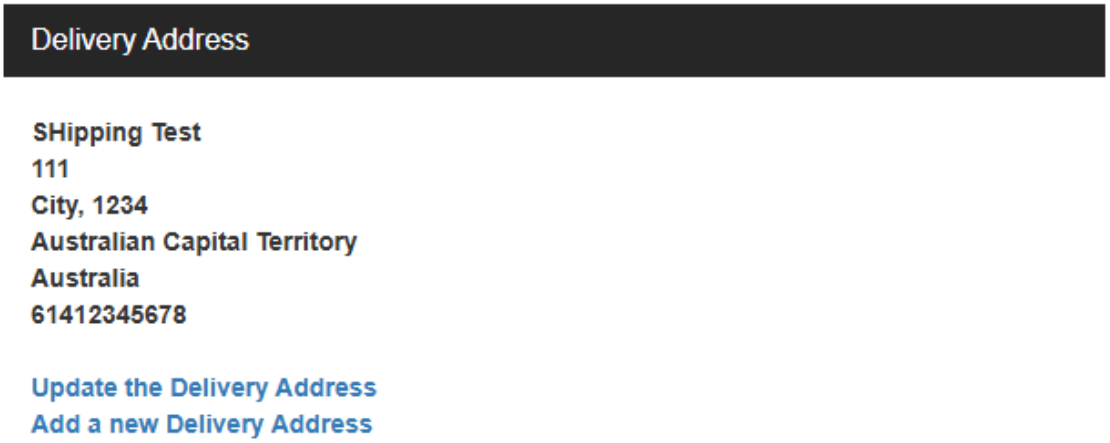

Рисунок 4.8 – Інтерфейс правильно заповненої збереженої адреси

Відображення помилок про неправильно заповнені дані в адресі наведено н рисунку 4.9.

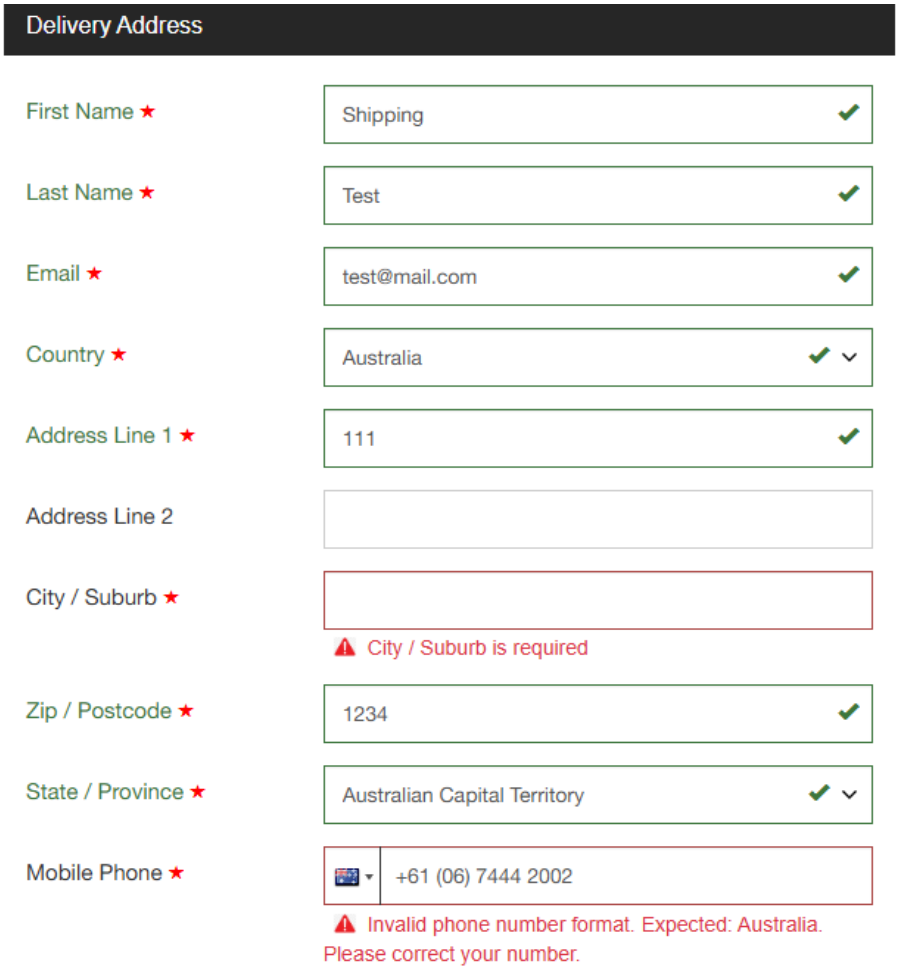

Рисунок 4.9 – Інтерфейс неправильно заповненої збереженої адреси

Редагування адреси (рис. 4.10):

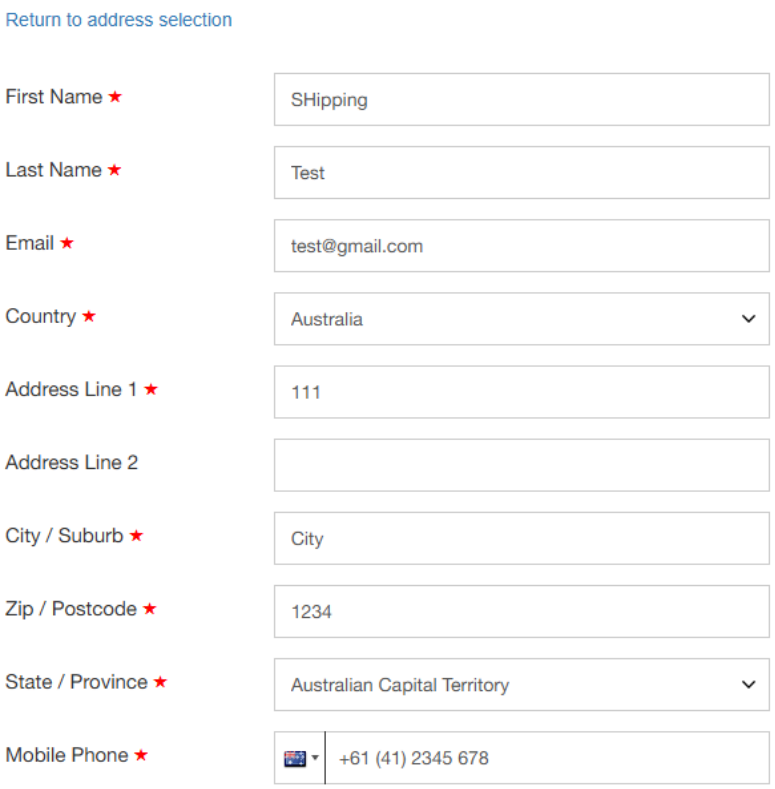

Рисунок 4.10 – Інтерфейс редагування збереженої адреси

Додавання нової адреси (рис. 4.11):

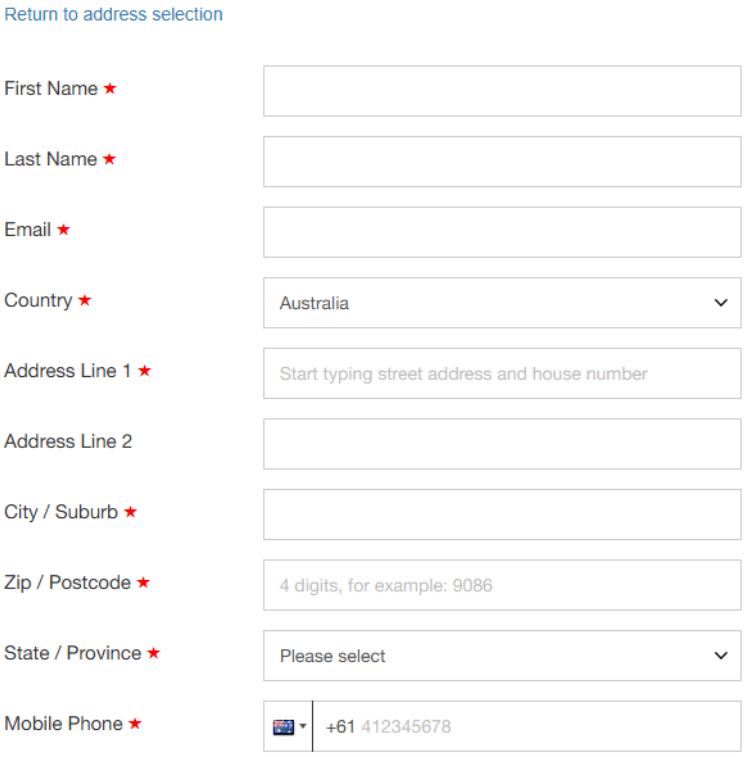

Рисунок 4.11 – Інтерфейс додавання нової адреси

Тестування також потребує налаштування зовнішнього вигляду власником або дизайнерами інтернет-магазину за допомогою можливості застосування власних CSS-стилів. Перевіримо інтерфейс деяких сторінок із налаштованими власними стилями (рис. 4.12, 4.13):

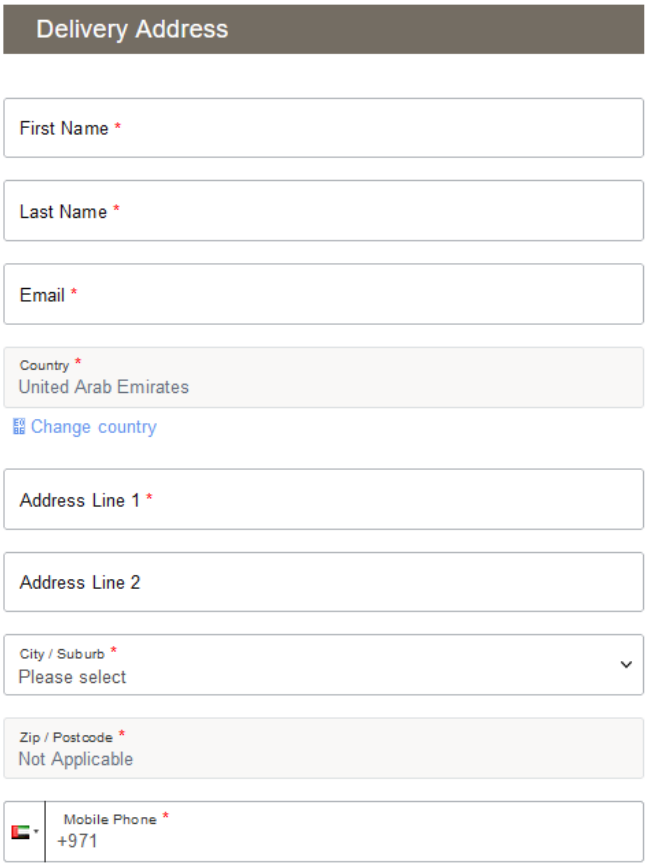

Рисунок 4.12 – Інтерфейс сторінки із налаштованими власними стилями

Щоб оцінити ефективність та конкурентоспроможність створеного додатку для оформлення онлайн-замовлень, важливо порівняти його з іншими популярними додатками для електронної комерції, які вже встигли зарекомендувати себе на ринку. Таке порівняння допоможе виявити сильні та слабкі сторони нашого рішення, а також визначити області для подальшого покращення та розвитку. У аналізі будемо розглядати основні функціональні можливості, які є важливими для успішного використання в оформленні онлайн-замовлення.

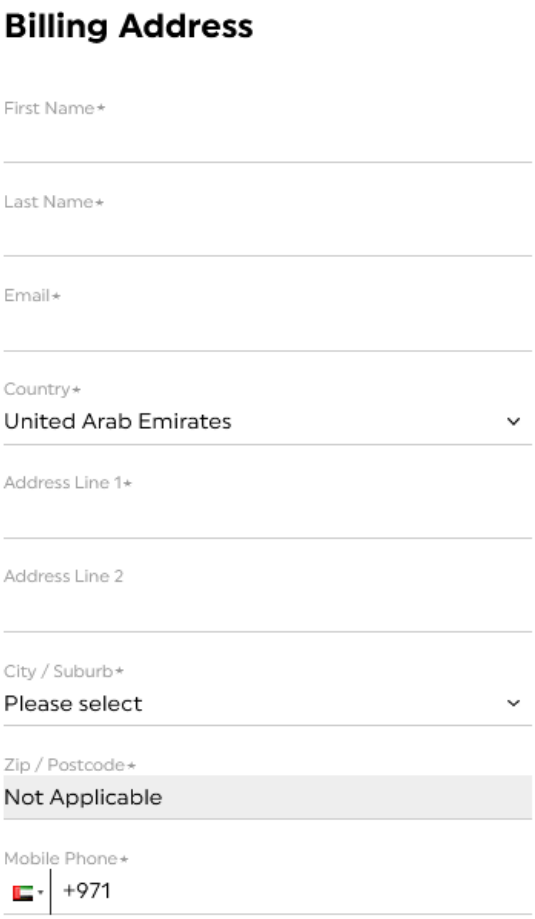

Рисунок 4.13 – Інтерфейс сторінки із налаштованими власними стилями

Основними критеріями для порівняння стали такі функції, як перегляд інформації про обрані продукти, внесення особистих даних, відображення індикаторів правильності заповнення даних, налаштування зовнішнього вигляду додатку, інтеграція з платіжними системами, доступність у країнах. Це дозволить отримати всебічний огляд можливостей додатку та порівняти його із такими відомими додатками для оформлення онлайн-замовлень, як Shopify та Square. Таблиця порівняння 4.1 надасть чітке і зрозуміле представлення функціональних можливостей кожного з цих додатків, допомагаючи визначити, наскільки розроблений додаток відповідає сучасним вимогам електронної комерції.

Таблиця 4.1 – Порівняльний аналіз додатків електронної комерції

	Розроблений додаток	**Shopify**	**Square**
Перегляд інформації про обрані продукти	Базові дані, відображення повної вартості із самого верху сторінки	Базові дані, для перегляду повної вартості необхідно пролистати до кінця сторінки	Базові дані, для перегляду повної вартості необхідно пролистати до кінця сторінки
Заповнення даних	Розширений функціонал (автозаповнення, вибір країни телефону)	Базовий функціонал	Базовий функціонал
Відображення індикаторів правильності заповнення даних	Відображається одразу, змістовні повідомлення	Відображається після кліку «Далі», базові повідомлення	Відображається після кліку «Далі», базові повідомлення
Налаштування зовнішнього вигляду додатку	Детальне налаштування, можливість налаштувати вигляд кожного елементу	Обмежене налаштування	Обмежене налаштування
Інтеграція з платіжними системами	Широкий перелік підтримуваних платіжних систем: Google Pay, PayPal, Klarna, Apple Pay, кредитна картка, банківський платіж тощо	Обмежений перелік підтримуваних платіжних систем	Обмежений перелік підтримуваних платіжних систем
Доступність у країнах	Доставка доступна майже в кожну країні, продавець може налаштувати список доступних країн	Доставка доступна майже в кожну країні, продавець може налаштувати список доступних країн	Повністю недоступний до перегляду в деяких країнах

Під час проведеного порівняльного аналізу вдалося виявити, що розроблений додаток має ряд переваг у порівнянні з конкурентами, такими як Shopify та Square. Зокрема, виявлено, що наш додаток має розширений функціонал для перегляду інформації про обрані продукти та налаштування зовнішнього вигляду, що

дозволяє забезпечити більшу інформативність, а також гнучкість і контроль над виглядом додатку та користувацьким досвідом його використання. Також розроблений додаток підтримує більше платіжних систем та доступний майже в усіх країнах, що надає користувачам покращений досвід оформлення та оплати онлайн-замовлень.

ВИСНОВКИ

У бакалаврській дипломній роботі було проведено дослідження та розроблено додаток для оформлення онлайн-замовлень із використанням технологій ReactJS і NestJS. Під час аналізу було встановлено, що ця тема є дуже актуальною, оскільки зміни в сучасному суспільстві роблять важливою роль в цифровій взаємодії продавців та покупців. Однак, розробка такого додатку стикається з рядом проблем, зокрема масштабованістю, продуктивністю, безпекою та користувацьким досвідом.

В процесі розробки було виявлено, що технології React і NestJS у поєднанні дозволяють легко створювати зручні, ефективні та високофункціональні веб-додатки. Використання REST API дає змогу ефективно взаємодіяти з сервером, обмінюючись необхідними даними та гарантуючи високу доступність застосунку.

Для подальшого вдосконалення додатку рекомендується впровадити додаткові інструменти, такі як аналітика та розширені анімації. Також доцільно розглянути застосування сучасних бібліотек із останніми версіями для забезпечення високої надійності та масштабованості. У порівнянні з існуючими аналогами, такий підхід дозволить підвищити швидкодію, безпеку та якість користувацького досвіду.

У цілому, мета роботи була досягнута в повному обсязі, що підтверджується практичними результатами дослідження та розробки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Yomo.in.ua. Популярність покупок через інтернет: тренди, статистика та переваги онлайн-шопінгу. URL: https://yomo.in.ua/shoping/populyarnist-pokupok-cherez-internet-trendy-statystyka-ta-perevagy-onlajn-shopingu/ (дата звернення: 24.04.2025).

2. Розробка веб-додатків на основі фреймворків React та NestJS / Кривогубченко К.Д., Богач І.В. // Матеріали LIV науково-технічної конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ-2025) Вінниця, ВНТУ, 2025. URL: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2025/paper/view/23640 (дата звернення 24.04.2025).

3. Розробка застосунку для оформлення онлайн-замовлень за допомогою React та NestJS / Кривогубченко К.Д., Богач І.В. // Матеріали Всеукраїнської науково-практичної інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)» 2025», Вінниця, 2025. URL: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/25425 (дата звернення 10.06.2025).

4. Наш Край – Сучасний шопінг: як самообслуговування, онлайн-замовлення та персоналізовані знижки змінюють наші покупки. URL: https://nashkraj.ua/uk/blog/suchasnyj-shoping-yak-samoobslugovuvannya-onlajn-zamovlennya-ta-personalizovani-znyzhky-zminyuyut-nashi-pokupky/ (дата звернення 07.05.2025).

5. Офіційний сайт Shopify. URL: https://www.shopify.com/ (дата звернення 07.05.2025).

6. Офіційний сайт BigCommerce. URL: https://www.bigcommerce.com/ (дата звернення 07.05.2025).

7. Square – Онлайн-магазин. URL: https://squareup.com/us/en/online-store (дата звернення 07.05.2025).

8. Mate academy – Vue, Angular чи React: що обрати новачку? URL: http://mate.academy/blog/front-end-and-js/vue-angular-or-react-what-to-choose-for-a-novice/ (дата звернення 07.05.2025).

9. HostIQ – База даних (Wiki). URL: https://hostiq.ua/wiki/ukr/database/ (дата звернення 07.05.2025).

10. React. – URL: https://react.dev/ (дата звернення: 06.03.2025).

11. Порівняння фреймворків JavaScript: Svelte проти React. URL: https://www.dreamhost.com/blog/uk/svelte-proti-react/ (дата звернення: 12.03.2025).

12. NestJS. Офіційна документація. URL: https://docs.nestjs.com/ (дата звернення: 12.03.2025).

13. Beighton, B. NestJS: The Pros and Con. URL: https://bradbeighton.medium.com/nestjs-the-pros-and-cons-aff714607b07 (дата звернення: 13.03.2025).

14. Alternative Science – SQL чи NoSQL: ось в чому питання. URL: https://alternativescience.net/programming/242-sql-chy-nosql-os-v-chomu-pytannya/ (дата звернення 07.05.2025).

15. Офіційний сайт Figma. URL: https://www.figma.com/ (дата звернення 07.05.2025).

16. Adobe XD – Початок роботи. URL: https://helpx.adobe.com/ua/xd/get-started.html (дата звернення 07.05.2025).

17. Офіційний сайт Sketch. URL: https://www.sketch.com/ (дата звернення 07.05.2025).

18. Startpack – InVision. URL: https://startpack.ru/application/invision (дата звернення 07.05.2025).

19. Visual Studio Code. URL: https://code.visualstudio.com/ (дата звернення: 26.05.2025).

20. MongoDB. Завантаження Community Kubernetes Operator. URL: https://www.mongodb.com/try/download/community-kubernetes-operator (дата звернення: 26.05.2025)

21. Axios. Документація та введення. URL: https://axios-http.com/ru/docs/intro (дата звернення: 26.05.2025).

22. Robot Dreams. Вступ до REST API: RESTful вебсервіси. URL: https://robotdreams.cc/uk/blog/466-vstup-do-rest-api-restful-vebservisi (дата звернення: 26.05.2025).

23. Srivastava A. Setting Up a Simple API with Nest.js and React.js: A Step-by-Step Guide. URL: https://medium.com/%40aviralsrivastava57/setting-up-a-simple-api-with-nest-js-and-reactjs-a-step-by-step-guide-javascript-30b868548df6 (дата звернення: 12.03.2025).

24. JavaScript Tutorial. URL: https://www.w3schools.com/js/ (дата звернення: 06.03.2025).

25. Wieruch, R. The Road to React: Your journey to master React.js in JavaScript. URL: https://www.roadtoreact.com/ (дата звернення: 10.04.2025).

26. Banks, A., Porcello, E. Learning React: Modern Patterns for Developing React Apps. URL: https://www.oreilly.com/library/view/learning-react-2nd/9781492051718/ (дата звернення: 10.04.2025).

27. Santana Roldán, C. React 18 Design Patterns and Best Practices: Design, build, and deploy production-ready web applications with React by leveraging industry-best practices. URL: https://www.packtpub.com/product/react-18-design-patterns-and-best-practices-fourth-edition/9781803233475 (дата звернення: 10.04.2025).

28. Linjanja, P. Scalable Application Development with NestJS: Leverage REST APIs, GraphQL, and microservices for efficient web development. URL: https://www.packtpub.com/product/scalable-application-development-with-nestjs/9781835468606 (дата звернення: 10.04.2025).

29. Krause, M. The Complete Developer: Master the Full Stack with TypeScript, React, Next.js, MongoDB, and Docker. URL: https://www.amazon.com/Complete-Developer-TypeScript-React-MongoDB-ebook/dp/B0BZZZ8QFP (дата звернення: 10.04.2025).

30. Sakhniuk, M. React and React Native: Build cross-platform JavaScript and TypeScript apps for web, desktop, and mobile. URL:

https://www.packtpub.com/product/react-and-react-native-third-edition/9781803237046 (дата звернення: 10.04.2025).

31. Schwarzmüller, M. React Key Concepts: Consolidate your knowledge of React’s core features. URL: https://www.amazon.com/React-Key-Concepts-Consolidate-knowledge/dp/1803245712 (дата звернення: 10.04.2025).

32. Bugl, D. Learn React Hooks: Build and refactor modern React.js applications using Hooks. URL: https://www.packtpub.com/product/learn-react-hooks/9781838828728 (дата звернення: 10.04.2025).

33. Minnick, C. Beginning ReactJS Foundations: Building User Interfaces with ReactJS. URL: https://www.wiley.com/en-us/Beginning+ReactJS+Foundations%3A+Building+User+Interfaces+with+ReactJS-p-9781119685548 (дата звернення: 10.04.2025).

34. Casciaro, M., Mammino, L. Node.js Design Patterns: Design and implement production-grade Node.js applications using proven patterns and techniques. URL: https://www.packtpub.com/product/node-js-design-patterns-third-edition/9781839214117 (дата звернення: 10.04.2025).

35. Bugl, D. React 18 Design Patterns and Best Practices: Design, build, and deploy production-ready web applications with React by leveraging industry best practices. URL: https://www.packtpub.com/product/react-18-design-patterns-and-best-practices-fourth-edition/9781803233475 (дата звернення: 10.04.2025).

36. Krause, M. The Complete Developer: Master the Full Stack with TypeScript, React, Next.js, MongoDB, and Docker. URL: https://www.amazon.com/Complete-Developer-TypeScript-React-MongoDB-ebook/dp/B0BZZZ8QFP (дата звернення: 10.04.2025).

37. Sakhniuk, M. React and React Native: Build cross-platform JavaScript and TypeScript apps for web, desktop, and mobile. URL: https://www.packtpub.com/product/react-and-react-native-third-edition/9781803237046 (дата звернення: 10.04.2025).

38. Schwarzmüller, M. React Key Concepts: Consolidate your knowledge of React’s core features. URL: https://www.amazon.com/React-Key-Concepts-Consolidate-knowledge/dp/1803245712 (дата звернення: 10.04.2025).

39. Bugl, D. Learn React Hooks: Build and refactor modern React.js applications using Hooks. URL: https://www.packtpub.com/product/learn-react-hooks/9781838828728 (дата звернення: 10.04.2025).

Додаток А

(обов’язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА ЗАСТОСУНКУ ДЛЯ ОФОРМЛЕННЯ ОНЛАЙН-ЗАМОВЛЕНЬ ЗА ДОПОМОГОЮ REACT ТА NESTJS

Діаграма взаємодії користувача з додатком

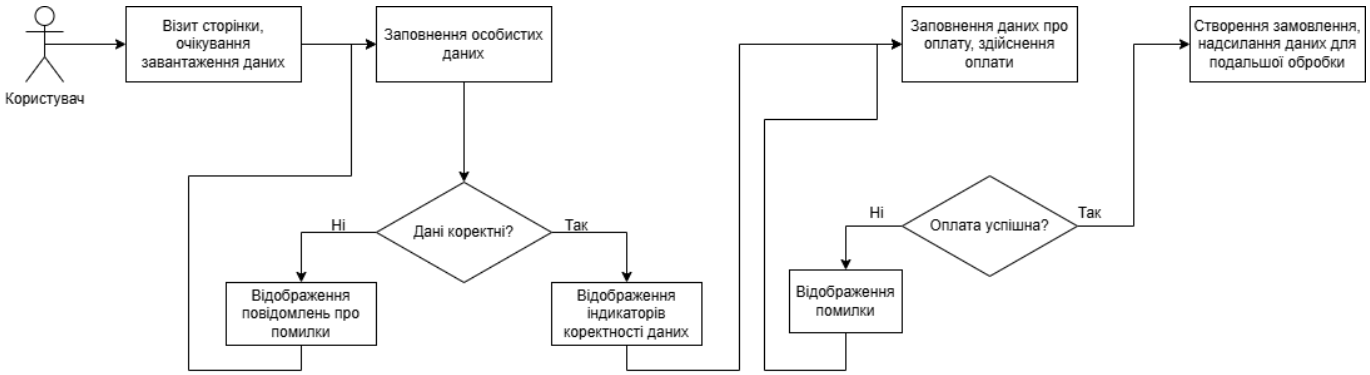

Рисунок А.1 - Діаграма взаємодії користувача з додатком

Діаграма класів та сервісів додатку

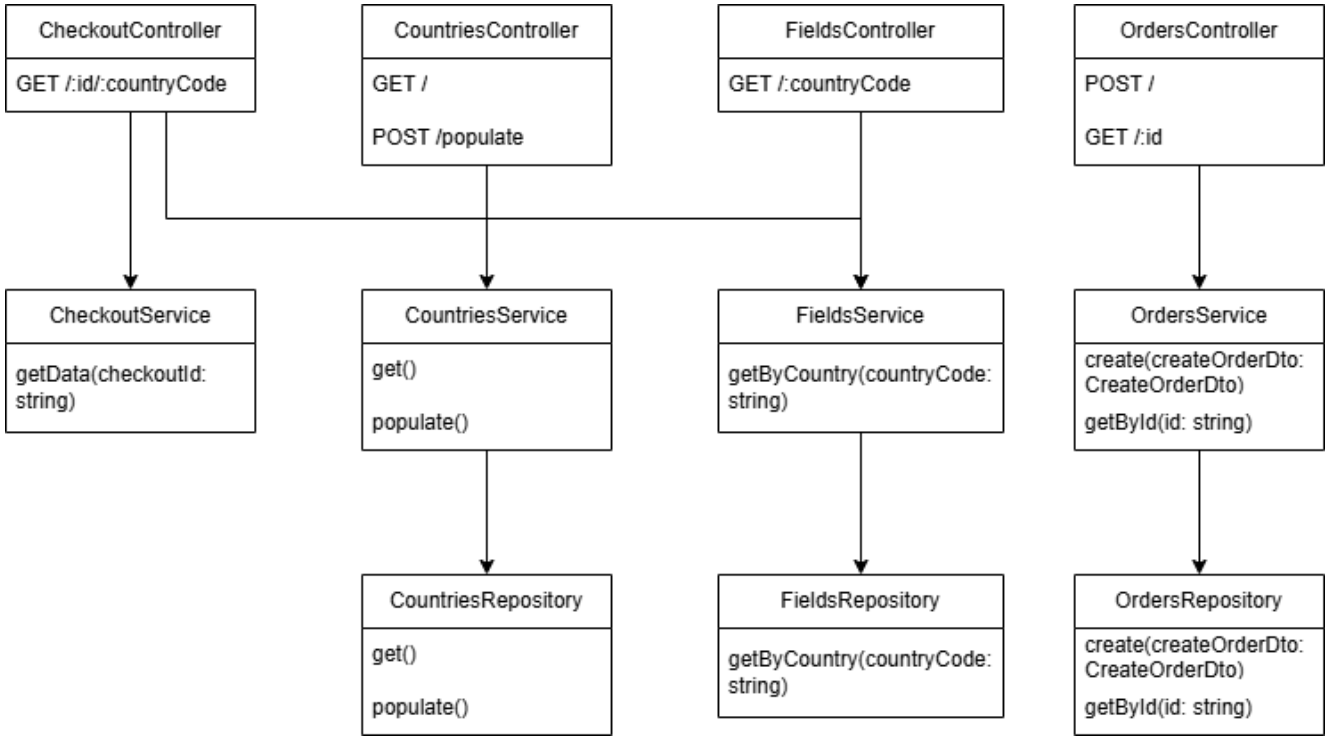

Рисунок А.2 - Діаграма класів та сервісів додатку

Діаграма функціоналу збережених адрес

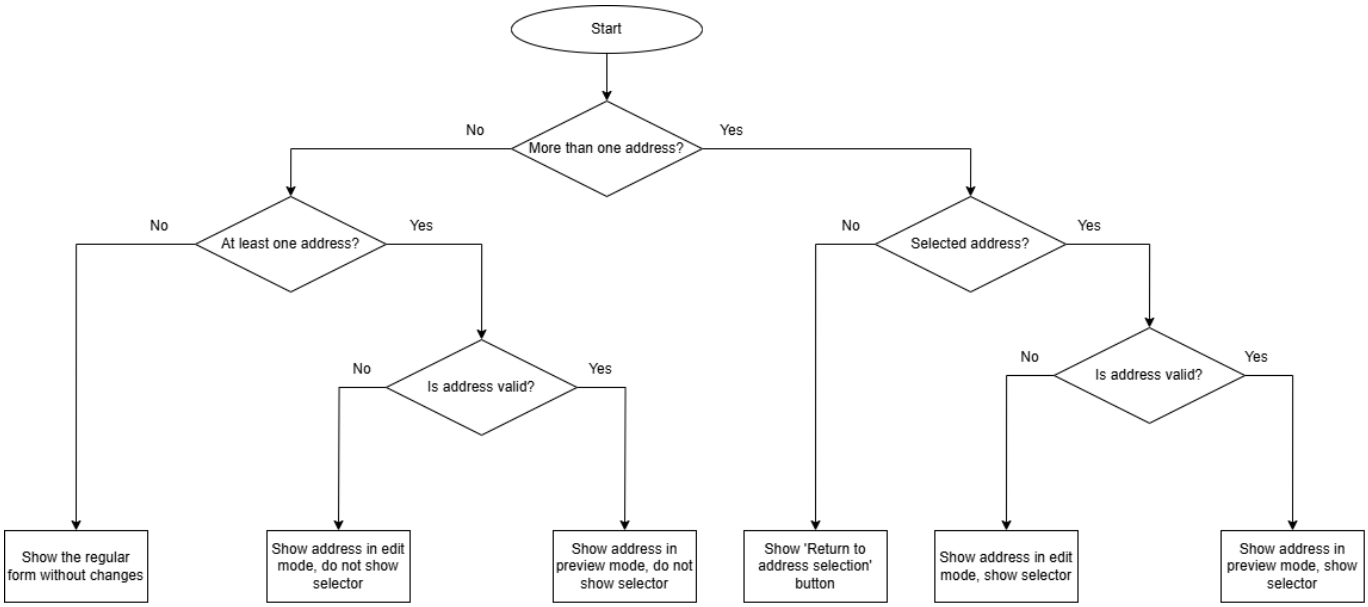

Рисунок А.3 - Діаграма функціоналу збережених адрес

Додаток Б (обов'язковий)

Лістинг основних функцій програми

main.ts

```
import { NestFactory } from "@nestjs/core";
import { AppModule } from "./app.module";
import { DocumentBuilder, SwaggerModule } from "@nestjs/swagger";

async function bootstrap() {
  const app = await NestFactory.create(AppModule);

  const config = new DocumentBuilder()
    .setTitle("Checkout app")
    .setDescription("The checkout app API description")
    .setVersion("1.0")
    .addTag("Checkout data")
    .build();
  const documentFactory = () => SwaggerModule.createDocument(app, config);
  SwaggerModule.setup("api", app, documentFactory);

  await app.listen(3001);
}
bootstrap();
```

checkout.controller.ts

```
import { Controller, Get, Param } from "@nestjs/common";
import { CheckoutService } from "./services/checkout.service";
import { FieldsService } from "./services/fields.service";

@Controller("checkout")
export class CheckoutController {
  constructor(
    private readonly checkoutService: CheckoutService,
    private readonly fieldsService: FieldsService,
```

```
  ) {}

  @Get("/:id/:countryCode")
  async getData(
    @Param("id") id: string,
    @Param("countryCode") countryCode: string,
  ) {
    const checkoutData = await this.checkoutService.getData(id);
    const fields = await this.fieldsService.getByCountry(countryCode);

    return { ...checkoutData, fields };
  }
}
```

countries.controller.ts

```
import { Controller, Get, Post } from "@nestjs/common";
import { CountriesService } from "./services/countries.service";

@Controller("countries")
export class CountriesController {
  constructor(private readonly countriesService: CountriesService) {}

  @Get()
  getCountries() {
    return this.countriesService.get();
  }

  @Post("/populate")
  populateCountries() {
    return this.countriesService.populate();
  }
}
```

fields.controller.ts

```
import { Controller, Get, Param } from "@nestjs/common";
```

```
import { FieldsService } from "./services/fields.service";

@Controller("fields")
export class FieldsController {
  constructor(private readonly fieldsService: FieldsService) {}

  @Get(":countryCode")
  async getFieldsForCountry(@Param("countryCode") countryCode: string) {
    return this.fieldsService.getByCountry(countryCode);
  }
}
```

orders.controller.ts

```
import { Controller, Get, Post, Body, Param } from "@nestjs/common";
import { OrdersService } from "./services/orders.service";
import { CreateOrderDto } from "./dto/create-order.dto";

@Controller("orders")
export class OrdersController {
  constructor(private readonly ordersService: OrdersService) {}

  @Post()
  create(@Body() createOrderDto: CreateOrderDto) {
    return this.ordersService.create(createOrderDto);
  }

  @Get(":id")
  getById(@Param("id") id: string) {
    return this.ordersService.getById(id);
  }
}
```

countries.service.ts

```
import { Injectable } from "@nestjs/common";
import { CountriesRepository } from "../repositories/countries.repository";
```

```
@Injectable()
export class CountriesService {
  constructor(private readonly countriesRepository: CountriesRepository) {}

  get() {
    return this.countriesRepository.get();
  }

  async populate() {
    return this.countriesRepository.populate();
  }
}
```

fields.service.ts

```
import { Injectable } from "@nestjs/common";
import { FieldsRepository } from "../repositories/fields.repository";

@Injectable()
export class FieldsService {
  constructor(private readonly fieldsRepository: FieldsRepository) {}

  getByCountry(countryCode: string) {
    return this.fieldsRepository.getByCountry(countryCode);
  }
}
```

orders.service.ts

```
import { Injectable } from "@nestjs/common";
import { CreateOrderDto } from "../dto/create-order.dto";
import { OrdersRepository } from "../repositories/orders.repository";

@Injectable()
export class OrdersService {
  constructor(private readonly ordersRepository: OrdersRepository) {}
```

```
  create(createOrderDto: CreateOrderDto) {
    return this.ordersRepository.create(createOrderDto);
  }

  getById(id: string) {
    return this.ordersRepository.getById(id);
  }
}
```

fields.repository.ts

```
import { Injectable } from "@nestjs/common";
import { CreateOrderDto } from "../dto/create-order.dto";
import { InjectModel } from "@nestjs/mongoose";
import { Model } from "mongoose";
import { Field } from "../schemas/field.schema";

@Injectable()
export class FieldsRepository {
  constructor(@InjectModel(Field.name) private fieldModel: Model<Field>) {}

  getByCountry(countryCode: string) {
    return this.fieldModel.find({ countryCode: countryCode.toUpperCase() });
  }
}
```

countries.repository.ts

```
import { Injectable } from "@nestjs/common";
import { InjectModel } from "@nestjs/mongoose";
import { Model } from "mongoose";
import { Country } from "../schemas/country.schema";
import { countriesData } from "../helpers/countriesData";

@Injectable()
export class CountriesRepository {
```

```
  constructor(
    @InjectModel(Country.name) private countryModel: Model<Country>,
  ) {}

  get() {
    return this.countryModel.find();
  }

  async populate() {
    const existingCountries = await this.get();
    if (existingCountries.length === 0) {
      this.countryModel.insertMany(countriesData);
    }
  }
}
```

orders.repository.ts

```
import { Injectable } from "@nestjs/common";
import { CreateOrderDto } from "../dto/create-order.dto";
import { InjectModel } from "@nestjs/mongoose";
import { Order } from "../schemas/order.schema";
import { Model } from "mongoose";

@Injectable()
export class OrdersRepository {
  constructor(@InjectModel(Order.name) private orderModel: Model<Order>) {}

  create(createOrderDto: CreateOrderDto) {
    const newOrder = new this.orderModel(createOrderDto);
    return newOrder.save();
  }

  getById(id: string) {
    return this.orderModel.findById(id);
  }
```

```
}
```

Додаток В (обов'язковий)

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: «Розробка застосунку для оформлення онлайн-замовлень за допмогою React та NestJS»

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра АІІТ, ФІІТА, 1ІСТ-21б
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 10.88 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

☑ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Олег БІСІКАЛО, зав. кафедри АІІТ _____ (підпис)
(прізвище, ініціали, посада)

Любов ЯНЧУК, секретар кафедри АІІТ _____ (підпис)
(прізвище, ініціали, посада)

Особа, відповідальна за перевірку _____ Костянтин ОВЧИННИКОВ
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Ілона БОГАЧ, к.т.н., доцент кафедри АІІТ
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Ксенія КРИВОГУБЧЕНКО
(підпис) (прізвище, ініціали)

FIVE
BUILDING THE FUTURE

21027, Україна, м.Вінниця, просп.Космонавтів 42,кв.3,
тел. +380673490106, www.fivesysdev.com

За місцем вимоги

Довідка

Видана Кривогубченко Ксенії Денисівні в тому, що результати, одержані нею в процесі виконання бакалаврської кваліфікаційної роботи «Розробка застосунку для оформлення онлайн-замовлень за допомогою React та NestJS», а саме алгоритми та програмну реалізацію, планується використати в розробках ТОВ «ФАЙВ Системз Девелопмент».

Директор _____ Терещенко С.М.
(підпис) (прізвище та ініціали)

Товариство з обмеженою відповідальністю «ФАЙВ СИСТЕМЗ ДЕВЕЛОПМЕНТ» 39859344 Україна м.Вінниця