Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Берг Катерини Андріївни*
(ПІБ)

академічної групи *122-19-2*
(шифр)

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

освітньої програми *Комп'ютерні науки*
(назва освітньої програми)

на тему: *Розробка інтернет-магазину з продажу засобів індивідуального захисту з використанням фреймворку Angular*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*доц. Спірінцев В.В.*			
розділів:				
спеціальний	*доц. Спірінцев В.В.*			
економічний	*проф. Вагонова О.Г.*			
Рецензент				
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2023

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » 2023 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *122-19-2* *Берг Катерини Андріївни*
(група) (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка інтернет-магазину з продажу засобів індивідуального захисту з використанням фреймворку Angular*

затверджена наказом ректора НТУ «ДП» від *16.05.2023* № *350-с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів проєктно-технологічної практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі. Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2023 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***27.05.2023 р.***

Завдання видав _____ *доц. Спірінцев В.В.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Берг К.А.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2023 р.*

Термін подання кваліфікаційної роботи до ЕК: *12.06.2023 р.*

РЕФЕРАТ

Пояснювальна записка: 87 с., 42 рис., 21 джерел.

Об'єкт розробки: інтернет-магазин з продажу засобів індивідуального захисту на базі фреймворку Angular.

Мета кваліфікаційної роботи: розробка інтернет-магазину з продажу засобів індивідуального захисту на базі фреймворку Angular.

У вступі розглянуто сучасний стан проблеми, зроблено аналіз предметної області, визначено мету кваліфікаційної роботи та галузь її застосування, обґрунтовано актуальність тематики та конкретизовано постановку завдання.

У першому розділі зроблено аналіз предметної галузі, визначено призначення розробки та актуальність мети, сформульовано постановку завдання, зазначено вимоги до програмних засобів, технологій та програмної реалізації.

У другому розділі була обрана платформа для розробки, виконано проектування веб-орієнтованої системи з подальшою її розробкою, описана структура та принцип роботи програмного продукту, алгоритми його функціонування, визначено вхідні та вихідні дані, а також виклик та завантаження програми, зроблено характеристику складу параметрів технічних засобів.

В економічному розділі зроблено визначення трудомісткості розробленого програмного забезпечення, розраховано вартість роботи та час на створення інформаційної системи.

Практичне значення даної кваліфікаційної роботи полягає у створені веб-орієнтованого додатку, що надає послуги у сфері продажу засобів індивідуального захисту з використанням сучасних веб-технологій, а також на базі фреймфорку Angular. Інтернет-магазин має зрозумілу та зручну для використання структуру. Правильно спроектований графічний інтерфейс користувача дозволяє отримати швидкий доступ до інформації.

Актуальність розробки веб-орієнтованої інформаційної системи, що надає послуги у сфері продажу засобів індивідуального захисту не викликає сумніву та визначається підвищенням продуктивності бізнесу, поліпшенням іміджу та зростанням популярності магазину, збільшенням кількості клієнтів та відповідно прибутку компанії.

Список ключових слів: ІНФОРМАЦІЙНА СИСТЕМА, ІНТЕРНЕТ-МАГАЗИН, БАЗА ДАНИХ, SQLITE, HTML, CSS, ANGULAR, TYPESCRIPT, C#, HTTPS, BOOTSTRAP, DOM, JSON.

ABSTRACT

Explanatory note: 87 pp., 42 fig., 21 sources.

The object of development: an online store selling personal protective equipment based on the Angular framework.

The purpose of the thesis: development of software for an online store selling personal protective equipment based on the Angular framework.

The introduction describes the current state of the problem and the analysis of the subject area, specifies the purpose of the qualification work and its scope, substantiates the relevance of the topic and clearly defines the task statement.

The first section analyzes the subject area, defines the purpose of the development and the relevance of the goal, formulates the task statement, and specifies the requirements for software tools, technologies, and software implementation.

In the second section, the platform for development was selected, the design of a web-based system was carried out with its subsequent development, the structure and principle of operation of the software product, algorithms for its functioning were described, input and output data were determined, as well as the program call and download, and the composition of the parameters of technical means was characterized.

In the economic section, the labor intensity of the developed software is determined, the cost of work and the time to create an information system are calculated.

The practical significance of this qualification work is the creation of a web-based application that provides services in the field of personal protective equipment sales using modern web technologies, as well as based on the Angular framework. The online store has a clear and easy-to-use structure. A properly designed graphical user interface allows you to get quick access to information.

The relevance of developing a web-based information system that provides services in the field of personal protective equipment sales is undeniable and is determined by increasing business productivity, improving the image and growing popularity of the store, increasing the number of customers and, accordingly, the company's profits.

List of keywords: INFORMATION SYSTEM, ONLINE STORE, DATABASE, SQLITE, HTML, CSS, ANGULAR, TYPESCRIPT, C#, HTTPS, BOOTSTRAP, DOM, JSON.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ОС - Операційна система

БД - База даних

СУБД - Система управління базами даних

ID - Ідентифікаційний номер

SQL - Structured Query Language

HTML - Hyper Text Markup Language

CSS - Cascading Style Sheets

SCSS - Sassy Cascading Style Sheets

API - Application Programming Interface

NPM - Node Package Manager

HTTPS - Hypertext Transfer Protocol Secure

HTTP - Hypertext Transfer Protocol

JSON - JavaScript Object Notation

DOM - Document Object Model

EF - Entity Framework

ВСТУП

На сьогоднішній день найпопулярнішим видом електронної віртуальної торгівлі є інтернет-магазин. Потужний розвиток інформаційних технологій зумовлює зростання та збільшення кількості даних інформаційних систем, що призводить до здорової конкуренції на ринку торгівлі. Інтернет-магазин визначається вичерпним та наочним каталогом товарів, що пропонує компанія, з широким описом та зазначенням ціни на даний товар. Це, у свою чергу, дозволяє клієнтам отримати сервіс високого рівня, сформувати загальне враження отримане при взаємодії з компанією та її співробітниками.

Актуальність розробки програмного забезпечення інтернет-магазину з продажу засобів індивідуального захисту не викликає сумніву та визначається великим попитом на подібні системи з метою підвищення продуктивності бізнесу, поліпшення іміджу та зростання популярності магазину, збільшення кількості клієнтів та відповідно прибутку компанії.

Об'єктом дослідження є веб-орієнтована інформаційна система з продажу засобів індивідуального захисту на базі фреймворку Angular.

Мета кваліфікаційної роботи: розробка інтернет магазину з продажу засобів індивідуального захисту на базі фреймворку Angular.

Наступні кроки є необхідними для вирішення поставленої задачі:

- визначення та аналіз основних рис електронної комерції;
- чітке та структуроване формування вимог щодо функціональних та технічних характеристик, засобів, інформаційного наповнення, способу реалізації та інформаційної безпеки програмного застосунку;
- створення макетів та розробка візуальної концепції сторінок;
- проектування архітектури додатку;
- вибір та налаштування програмного інструментарію;
- визначення структури та розробка веб-орієнтованої системи;
- здійснення розрахунку трудомісткості та витрат на розробку інформаційної системи.

Практичне значення даної кваліфікаційної роботи полягає у створенні веб-орієнтованого програмного додатку, що забезпечує надання послуг у сфері продажу засобів індивідуального захисту для фізичних та юридичних осіб з використанням сучасних веб-технологій, а також на базі фреймфорку Angular. Система управління базами даних SQLite була використана для роботи з базою даних. Інтернет-магазин має зрозумілу та зручну для використання структуру. Правильно спроектований графічний інтерфейс користувача дозволяє отримати швидкий доступ до інформації.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальні відомості з предметної галузі

Електронна комерція (електронна торгівля) - це діяльність з купівлі або продажу товарів в електронному вигляді на онлайн-сервісах або через Інтернет [1].

Електронна комерція бере свій початок у 1960-х роках, коли компанії використовували електронну систему під назвою «Електронний обмін даними» для полегшення передачі документів. Перша транзакція відбулася лише в 1994 році. Це був продаж компакт-диска між друзями через інтернет-магазин під назвою NetMarket. Відтоді галузь зазнала багато змін, що призвело до значної еволюції. Традиційні роздрібні торговці були змушені впроваджувати нові технології, щоб утриматися на плаву, оскільки такі компанії, як Alibaba, Amazon, eBay та Etsy, стали відомими. Ці компанії створили віртуальний ринок товарів і послуг, до якого споживачі можуть легко отримати доступ.

Електронна комерція зазвичай використовує Інтернет принаймні протягом частини життєвого циклу транзакції, хоча може також використовувати інші технології, наприклад, електронну пошту. Типові транзакції електронної комерції включають купівлю товарів (наприклад, книг на Amazon) або послуг (наприклад, завантаження музики у формі цифрової дистрибуції, такої як iTunes Store).

Існує три сфери електронної комерції: роздрібна торгівля в Інтернеті, електронні ринки та онлайн-аукціони. Цінність існування електронної комерції полягає в тому, щоб дозволити споживачам робити покупки в Інтернеті та оплачувати їх через Інтернет, заощаджуючи час і простір клієнтів і підприємств, значно підвищуючи ефективність транзакцій.

Залежно від товарів, послуг та організації компанії електронної комерції, існує п'ять основних категорій електронної комерції [2, 3]:

- Бізнес для споживача (B2C)
- Бізнес для бізнесу (B2B)
- Бізнес для уряду (B2G)
- Від споживача до споживача (C2C)
- Від споживача до бізнесу (C2B)
- Від споживача до уряду (C2G)

Технологічний стек електронної комерції - це набір цифрових інструментів, програмних додатків і платформ, які власник інтернет-магазину використовує для ведення свого бізнесу. Ці інструменти працюють разом для оптимізації повсякденних завдань - продажів, маркетингу, обслуговування клієнтів, виконання та повернення замовлень, обробки платежів та інших ключових сфер.

Щоб зрозуміти як працює стек технологій, розглянемо детальніше як працює веб-розробка. Технологічні стеки для веб-розробки електронної комерції можна розділити на два основні компоненти: фронтенд і бекенд [6].

Фронтенд - це зовнішні додатки та мови, які використовуються при розробці та дизайні вашого веб-сайту електронної комерції (вітрини інтернет-магазину). Основні з них включають в себе:

- каскадні таблиці стилів (CSS). Мова для додавання стилів до HTML-документів;
- мова розмітки гіпертексту (HTML). Стандартна мова розмітки, що використовується для розробки веб-сторінок;
- JavaScript (JS). Мова програмування, що використовується для додавання динамічних та інтерактивних елементів на веб-сайти (також використовується у бекенд-розробці).

Бекенд відповідає за функціональність. Це серверні програми та мови, які використовуються для підтримки вашого веб-сайту електронної комерції та програмних рішень, здатних взаємодіяти один з одним. Основні категорії функціональності бекенда включають:

– бази даних. Допомагають отримати доступ до інформації, що має відношення до електронної комерції, та керувати нею (наприклад DynamoDB, Firebase, MongoDB, MySQL та PostgreSQL);
– сховище. Система або стратегія для зберігання та управління даними (наприклад AWS S3 і Firebase Storage);
– веб-сервер. Комп'ютерна програма, яка приймає запити від користувачів (відвідувачів веб-сайту), робить їх аналіз, та розподіляє веб-сторінки відповідно до цих запитів.

1.2. Призначення розробки та галузь застосування

У представленій кваліфікаційній роботі розглядається розробка веб-орієнтованої інформаційної системи з продажу засобів індивідуального захисту, представленої у вигляді веб-сайту, який реалізує можливість швидкого доступу до каталогу товарів підприємства та формування замовлення.

Наступні вимоги були сформовані стосовно реалізації інтернет-магазину:

– швидка робота та доступ до даних;
– надійний захист даних;
– зручний та продуманий користувацький інтерфейс;
– подання структурованої та змістовної інформації.

На рис.1.1. зображено прототип головної сторінки за допомогою сервісу «Figma».

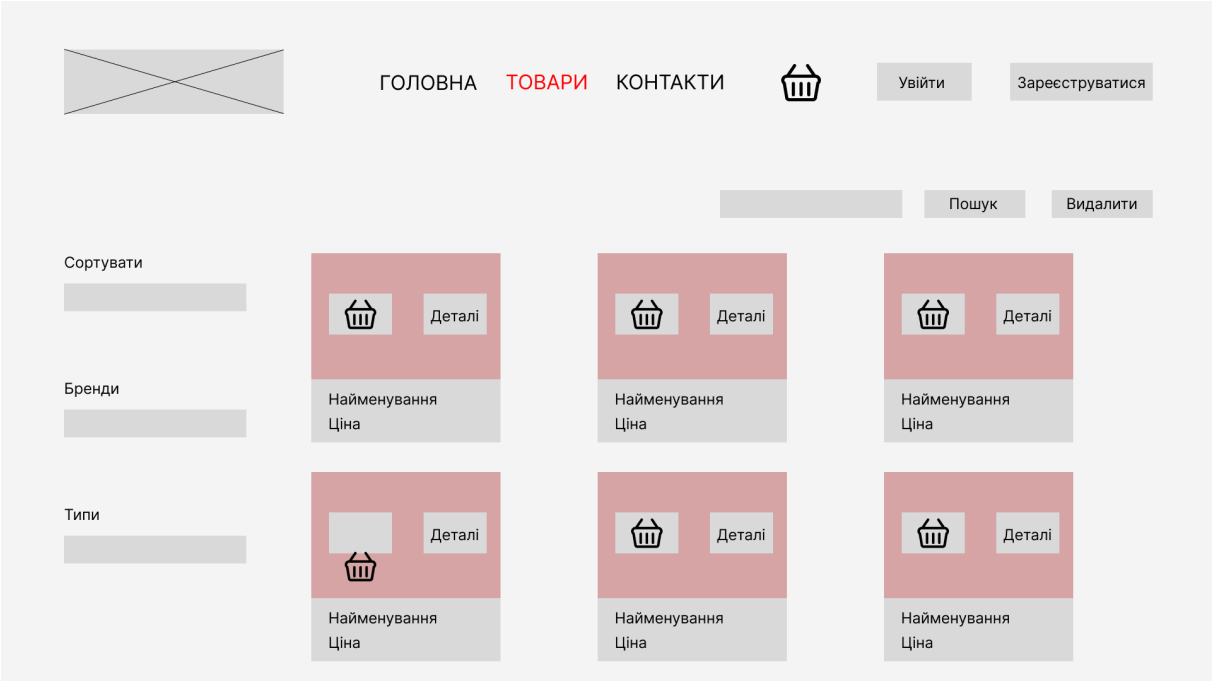

Рис. 1.1. Прототип сторінки товарів інформаційної системи

Аналізуючи вищезазначені вимоги, було зроблено висновок про те, що обов'язково буде реалізовано в проекті:

– головна сторінка (лаконічна презентація магазину для відвідувача);
– каталог товарів (основний структурний елемент сайту), що складається з категорій та підкатегорій, карточок товарів, пошукової строки та фільтру для можливості сортування;
– кошик товарів, у якому відображається детальна інформація стосовно обраних позицій з можливістю редагування та подальшої оплати замовлення.

Розроблена інформаційна система призначена для:

– можливості створювати покупки, виконувати пошук товарів та відображати їх за окремими категоріями;
– підвищення продажів та прибутку підприємства;
– впровадження сервісу високого рівня, за рахунок комфортного та зручного доступу користувача до пропозицій магазину.

1.3. Підстава для розробки

Відповідно до освітньої програми, згідно навчального плану та графіків навчального процесу, в кінці навчання студент виконує кваліфікаційну роботу (проект).

Тема роботи узгоджується з керівником проекту, випускаючою кафедрою, та затверджується наказом ректора.

Отже, підставами для розробки (виконання кваліфікаційної роботи) є:

- освітня програма спеціальності 122 «Комп’ютерні науки»;
- навчальний план та графік навчального процесу;
- наказ ректора Національного технічного університету «Дніпровська політехніка» № 350-с від 16.05.2023 р;
- завдання на дипломний проект на тему «Розробка інтернет-магазину з продажу засобів індивідуального захисту на базі фреймворку Angular».

1.4. Постановка завдання

У даній кваліфікаційній роботі розглядається створення інформаційної веб-орієнтованої системи з продажу засобів індивідуального захисту.

Мета створення інформаційної системи:

- надання послуг клієнтам, зокрема: отримання інформації про товарні позиції та формування замовлення;
- підвищення продуктивності бізнесу;
- сприяння залучення нових клієнтів та підтримання зв’язку з існуючими.

Тематикою веб-орієнтованої інформаційної системи є продаж товарів індивідуального захисту фізичним та юридичним особам.

Програмний додаток повинен відповідати наступним вимогам:

- зручний користувальницький інтерфейс;
- подання вичерпної та детальної інформації;
- безперервна та швидка робота у процесі відображення сторінок;

– зрозумілість та наочність.

Розробка веб-орієнтованої інформаційної системи передбачає виконання наступних етапів:

– визначення мети розробки та впровадження інформаційної системи, затвердження завдання на її розробку;
– формування основних вимог щодо розробленого програмного продукту;
– створення макетів та розробка візуальної концепції сторінок;
– проектування архітектури додатку;
– вибір та налаштування програмного інструментарію;
– визначення структури та розробка веб-орієнтованої системи;
– здійснення розрахунку трудомісткості та витрат на розробку програмного продукту.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

З метою досягнення поставленої мети в інформаційній системі, що розробляється, повинні бути реалізовані:

– інтуїтивно зрозумілий користувальницький інтерфейс;
– швидка та зручна навігація по сторінках;
– можливість авторизації та реєстрації користувачів;
– наявність швидкого доступу до кошику товарів;
– простота формування замовлення.

1.5.2. Вимоги до інформаційної безпеки

Під інформаційною безпекою розуміється захист конфіденційної інформації від несанкціонованих дій, включаючи перевірку, модифікацію, запис, а також порушення або знищення [10].

Мета інформаційної безпеки – забезпечити непорушність та конфіденційність критично важливих даних, таких як реквізити рахунків клієнтів, фінансові дані чи інтелектуальна власність.

Розгянемо найпоширеніші типи загроз веб-безпеки [11].

Фішинг – атака користувачів через електронну пошту, текстові повідомлення або сайти обміну повідомленнями в соціальних мережах.

Програми-вимагачі - це різновид шкідливого програмного забезпечення, в результаті якого зловмисник утримує дані або комп'ютер жертви в заручниках.

SQL-ін'єкція - це загроза веб-безпеці, при якій зловмисники використовують уразливості в коді програми.

DDoS-атака - це загроза веб-безпеці, яка полягає в тому, що зловмисники переповнюють сервери великими обсягами інтернет-трафіку, щоб порушити обслуговування і вивести веб-сайти з ладу. Великий обсяг фальшивого трафіку призводить до перевантаження цільової мережі або сервера, що

Для впровадження безпечної та безперервної роботи додатку, необхідними для виконання є наступні дії [4, 5]:

- Шифрування даних. Використання SSL-сертифікату дозволить перейти з HTTP на HTTPS, та буде слугувати сигналом довіри для клієнтів, що сайт є безпечним;
- відображення інформації про помилки, зроблені користувачем;
- валідація даних (перевірка правильності введення даних).

1.5.3. Вимоги до складу та параметрів технічних засобів

Для стабільної роботи веб-орієнтованого інформаційного додатку необхідними є певні вимоги до технічних засобів.

Клієнтська частина має наступні параметри:

- підтримувані протоколи передачі даних: HTTP / HTTPS;
- стабільний канал зв'язку від 10 Мб/сек;
- операційна система Windows 10 і вище, або MacOS;

– відеопам’ять: 4 Гб;
– оперативна пам'ять: не менше 2 ГБ вільного простору;
– наявність будь-якого браузера (наприклад Google Chrome).

Технічні характеристики, наведені вище, є рекомендованими для роботи та правильного функціонування розробленого програмного застосунку.

1.5.4. Вимоги до інформаційної та програмної сумісності

Серверна та клієнтська частина інтернет магазину, що розробляється має наступні вимоги стосовно встановлених програмних продуктів:

– модульна платформа .NET (версія 7.0);
– редактор вихідного коду Visual Studio Code;
– API платформа Postman для тестування створеної програми.

Веб-орієнтована інформаційна система має бути розроблена за допомогою мов програмування C#, TypeScript, HTML5, CSS3, Bootstrap5, з використанням фреймворку Angular за допомогою системи управління базами даних SQLite.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1. Функціональне призначення програми

Відповідно до завдання кваліфікаційної роботи було розроблено веб-орієнтовану інформаційну систему з продажу товарів індивідуального захисту з використанням фреймворку Angular.

Функціональне призначення даної програми включає в себе приймання та передачу запитів до бази даних, що у свою чергу генерує контент web-сторінок інтернет-магазину.

З точки зору клієнта, інформаційна система дозволяє:

- отримати швидкий та зручний доступ до каталогу товарів магазину за допомогою пошукової стрічки та фільтрації товарів за категоріями та виробниками;
- переглядати вичерпну інформацію по кожній позиції товару;
- формувати замовлення;
- мати можливість редагувати замовлення шляхом видалення та додавання товарів через особистий кошик;
- оплатити замовлення після його підтвердження.

З точки зору бізнесу, дана система сприяє:

- збільшенню клієнтури;
- підвищення прибутку;
- зростанню іміджу компанії.

Інтернет-магазин із зручним та інтуїтивно-зрозумілим користувальницьким інтерфейсом допоможе зростанню іміджу компанії та укріпить лояльність клієнтів, що, як наслідок, підвищить конкурентоспроможність та можливість зайняти передові позиції на ринку продажу засобів індивідуального захисту в Україні.

2.2. Опис застосованих математичних методів

Проектування та розробка даної веб-орієнтованої інформаційної системи не передбачає використання математичних методів.

2.3. Опис використаних технологій та мов програмування

Архітектура Angular-додатків базується на певних фундаментальних концепціях. Фреймворк складається з компонентів. Angular-компоненти організовані в NgModules, які у свою чергу з'єднують пов'язаний між собою код у функціональні набори. Додаток Angular складається з набору NgModules. Варто зазначити, що принаймні один (кореневий) модуль є необхідною складовою додатку, адже саме він забезпечує завантаження програми.

На рис.2.1 наведено дерево компонентів в Angular-додатку.

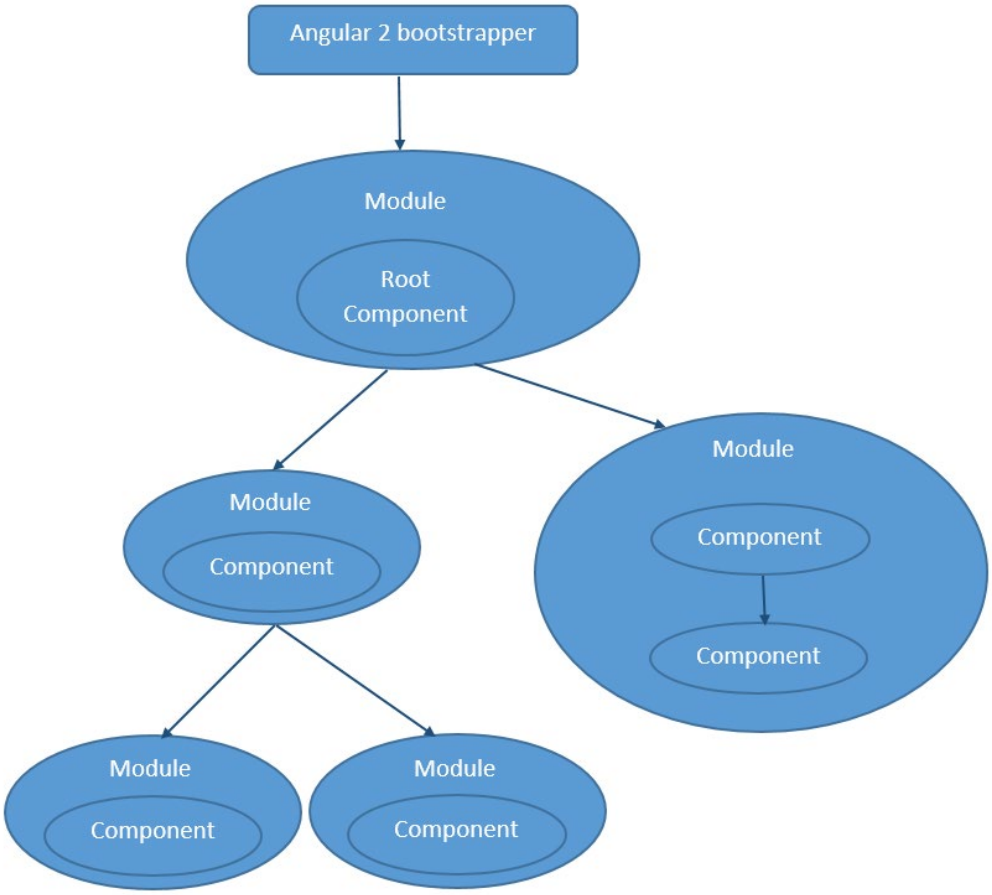

Рис. 2.1. Структура дерева компонентів в Angular-додатку

Відповідно до логіки даних та програми Angular має змогу вибирати та змінювати компоненти, які у свою чергу визначають представлення, у подальшому перетворювані на набори екранних елементів.

Сервіси, використовувані компонентами, надають функціонал, який безпосередньо не пов'язаний з представленнями. До компонентів наявна можливість додавати постачальників послуг у вигляді залежностей. Саме це спряє повторному використанню коду.

До класів, що використовують декоратори, можна віднести компоненти, модулі та сервіси. Декоратори мають на меті позначення типу та надання метаданих, які у подальшому вказують Angular на правильне використання зазначених вище класів.

Вигляд та відображення тієї чи іншої сторінки визначається шаблоном, який поєднує між собою HTML, директиви Angular та прив'язуючу розмітку. За допомогою цього Angular має можливість змінювати HTML перед відображенням.

Angular має чітко визначений сервіс Router, який використовується задля визначення шляхів навігації між поданнями [8].

Модулі

Angular NgModules відрізняються від модулів JavaScript (ES2015) і доповнюють їх. NgModule визначає контекст компіляції для набору компонентів. Компонент та відповідний йому код можуть бути пов'язані разом для формування функціональних одиниць за допомогою NgModule.

Кожен Angular-додаток має кореневий модуль, умовно названий AppModule. Це модуль відповідає за механізм завантаження, який запускає додаток.

Так само як JavaScript-модулі, NgModules мають змогу імпорту функціоналу з інших NgModules та експорту власного функціоналу з подальшим його використанням іншими модулями. Як приклад можна навести Router NgModule, який необхідно імпортувати задля використання служби маршрутизатора.

Проектування архітектури додатку Angular починається з наступних модулів, кожен з яких має власний набір структурних елементів:

- component, є відповідальним за візуальну частину сторінки. Містить HTML-шаблон та CSS-стилі;
- service, є постачальником даних для компонента, зазначеного вище;
- directive, перетворює певну частину DOM у відповідному вигляді.

Перераховані вище компоненти об’єднуються в кореневий модуль, що має назву AppModule.

Angular-додаток передбачає наявність лише одного кореневого модуля. Цей модуль, у свою чергу може використовувати інші модулі та їх функціонал. Ці модулі мають бути оголошені у властивості imports об’єкта декоратора @NgModule().

Компоненти

Angular-додаток повинен мати у своєму складі принаймні один компонент, як є кореневим та з’єднує усю ієрархію компонентів з DOM. Компоненти визначають класи, які містять логічну складову та дані. HTML-шаблон визначає вигляд даних компонентів.

@Component() є декоратором, що ідентифікує клас, надає метадані та шаблон, специфічні для цього компонента.

Декоратор @Component() ідентифікує клас, розташований безпосередньо під ним, як компонент, і надає шаблон і пов'язані з ним метадані, специфічні для компонента.

Додаток Angular складається із сукупності усіх компонентів.

Сервіси

Для даних або логіки, які не пов'язані з конкретним поданням, і які необхідно спільно використовувати між компонентами, створюється службовий клас. Визначенню класу сервісу безпосередньо передує декоратор @Injectable(). Декоратор надає метадані, які дозволяють іншим провайдерам вставляти у клас сервіси як залежності.

Ін'єкція залежностей дозволяє зберігати класи компонентів стрункими та ефективними. Вони не отримують дані з сервера, не перевіряють дані, введені користувачем, і не виводять дані безпосередньо в консоль; вони делегують ці завдання сервісам.

Маршрутизація

Angular сервіс Router дозволяє визначати навігацію та шляхи між сторінками (станами) додатку. Router втілює у собі звичайні навігаційні умови браузера, в результаті яких відбувається перехід на нову сторінку:

- в адресний рядок вводиться URL-адреса;
- натискається посилання на сторінці;
- використовується веб-переглядач з відповідним переходом назад та вперед.

Для визначення правил навігації обов’язковим є прив’язка шляхів навігації до своїх компонентів. Синтаксис, який використовує шлях, подібний до URL-адреси, який інтегрує програмні дані, подібно до інтеграції переглядів з даними програми. У подальшому застосування логіки програми дозволяє обирати, як саме інформація повинна бути представлена у відповідь на вхідні дані та правила доступу.

Розглянемо більше детальніше особливості фреймворку Angular [7].

DOM (об’єктна модель документа). На рис.2.2 наведено звичайний вигляд об’єктної моделі документа.

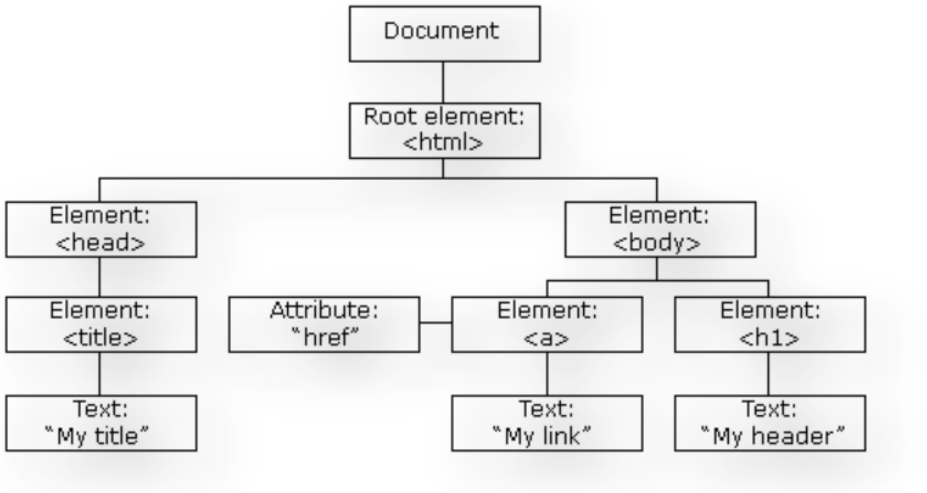

Рис. 2.2. Об’єктна модель документа

TypeScript виконує функцію визначення типів для JavaScript, який робить код більш зрозумілим. Компіляція коду TypeScript відбувається за допомогою JavaScript з подальшою роботою на будь-якій платформі. Для написання додатків на Angular не є обов’язковим використання TypeScript, але завдяки кращій синтаксичній структурі та покращенню сприйняття коду, TypeScript є рекомендованим саме для розроби Angular програм.

До переваг фреймворку Angular можна віднести наступні:

- спеціальні компоненти. Angular дозволяє користувачам створювати власні компоненти. Фреймворк також добре працює з веб-компонентами;
- прив'язка даних. Angular дозволяє користувачам легко переміщувати дані з коду JavaScript у представлення та реагувати на події користувача без необхідності писати код вручну;
- ін'єкція залежностей. Angular дозволяє користувачам писати модульні сервіси та впроваджувати їх там, де вони потрібні. Це покращує можливість тестування та повторного використання тих самих служб;
- тестування. Тести — це першокласні інструменти, і Angular було створено з нуля з урахуванням можливості тестування;

– сумісність з браузером. Angular є кросплатформним і сумісним з кількома браузерами. Програма Angular зазвичай може працювати у всіх браузерах (наприклад, Chrome, Firefox) і ОС, таких як Windows, macOS і Linux.

До .NET 3.5 для написання коду використовувалася ADO.NET (технологія, що надає доступ і управління даними, що зберігаються в базі даних або інших джерелах) з метою отримання або збереження даних програми з основної бази даних. Це зумовлювало необхідність з'єднуватися з базою даних, реалізацію DataSet для надсилання або отримання даних з БД. Цей процес передбачав схильність до утворення помилок, тому компанія Microsoft створила структуру, що має назву «Entity Framework». Автоматизація дій, які пов'язані з БД виконується саме за допомогою даної структури.

Entity Framework являється відкритою платформою для додатків на .NET. Дана платформа передбачає використання об'єктів доменних класів, при цьому не зосереджуючись на таблицях і стовпцях, де ці дані зазвичай зберігаються. Розробники мають можливість працювати з вищим рівнем абстракції, розробляти та впроваджувати програми, орієнтовані на дані з використанням меншої кількості коду [9].

На рис.2.3 показано розміщення Entity Framework у програмі.

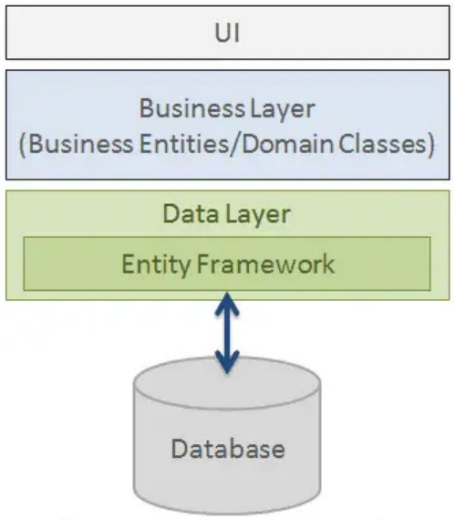

Рис. 2.3. Entity Framework

Як показано на рисунку вище, Entity Framework має місце поміж класами домену (бізнес-сутностями) та базою даних. Відбувається збереження даних, що містяться у властивостях бізнес-сутностей, а також отримання дані з БД з подальшим автоматичним перетворенням їх в об'єкти бізнес-сутностей.

Entity Framework має наступні можливості: кросплатформеність, запити, відстеження змін, збереження, моделювання, кешування, транзакції, паралелізм.

Для розробки додатку на базі фреймворку Angular було використано мови програмування C#, TypeScript.

C# - об'єктно-орієнтована мова програмування. C# має безпеку типів та передбачає створення безпечних і надійних додатків, які працюють в .NET [14].

TypeScript — це синтаксичний набір JavaScript, який додає статичний тип. Це означає, що TypeScript додає синтаксис поверх JavaScript, дозволяючи розробникам додавати типи. TypeScript має той самий базовий синтаксис, що й JavaScript.

JavaScript — це мова з вільною типізацією. Може бути важко зрозуміти, які типи даних передаються в JavaScript. У JavaScript параметри та змінні функції не мають жодної інформації. TypeScript дозволяє вказувати типи даних, які передаються в коді, і має можливість повідомляти про помилки, коли типи не збігаються. Наприклад, TypeScript повідомить про помилку під час передачі рядка у функцію, яка очікує число. У JavaScript така функція відсутня.

Для верстки сторінок сайту було використано HTML.

HTML є офіційною рекомендацією Консорціуму всесвітньої павутини (W3C) і, як правило, підтримується всіма основними веб-браузерами, включаючи десктопні та мобільні веб-браузери. HTML5 є останньою версією специфікації.

Для дизайну розробленої інформаційної системи був використаний фреймворк Bootstrap.

Bootstrap - це відкритий і безкоштовний HTML-, CSS- і JS-фреймворк, який використовують веб-розробники для швидкого верстання адаптивних дизайнів сайтів і веб-додатків [15].

За своєю суттю, Bootstrap - це набір файлів. Після їх під'єднання до сторінки для верстки стане доступною велика кількість готових компонентів і класів. Вони дають змогу швидко і якісно створювати адаптивний дизайн сайту.

2.4. Опис структури програми та алгоритмів її функціонування

Розглядаючи внутрішню частину розробленої програми, а саме бек-енд, інформаційна система для продажу засобів індивідуального захисту складається з трьох основних проектів, які зображені на рис.2.4.

Вигляд структури даного проекту в редакторі VS Code наведено на рис.2.5.

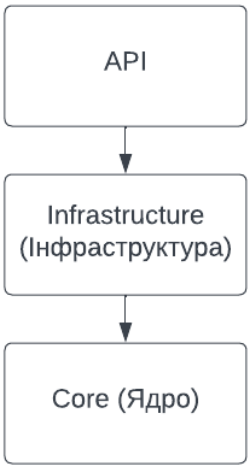

Рис. 2.4. Проекти програми

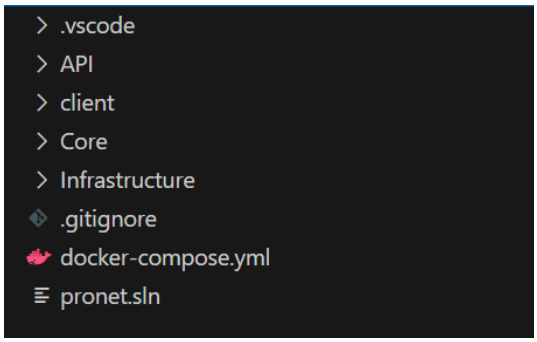

Рис. 2.5. Структура програми в VS Code

Як видно з наведеної вище діаграми, API залежить від Інфраструктури, а Інфраструктура залежить від Ядра. Це означає, що API має транзитивну залежність від компонентів всередині Ядра.

Розглянемо детальніше призначення кожного проекту веб-орієнтованого застосунку.

API отримує HTTP-запити та відповідає на них. Структурний вигляд API-частини у VS Code наведено на рис.2.6.

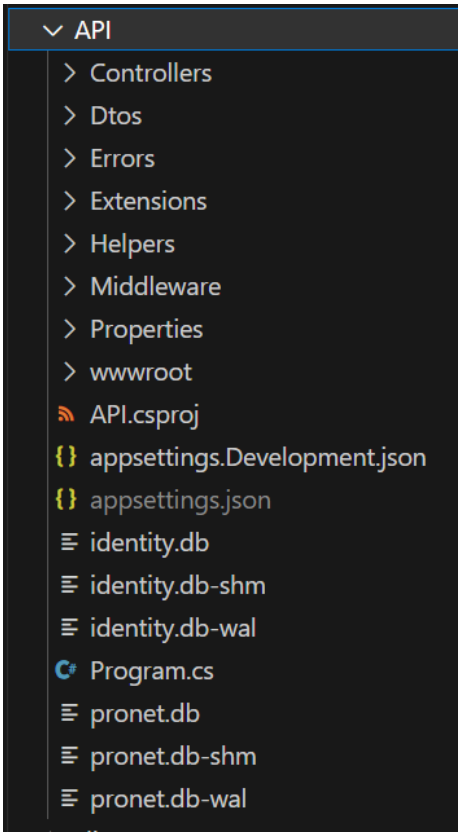

Рис. 2.6. Структура проекту API в VS Code

У проекті Інфраструктури налагоджена комунікація з базою даних, відбувається надсилання запитів до БД та отримання даних. Структурний вигляд Інфраструктури у VS Code наведено на рис.2.7.

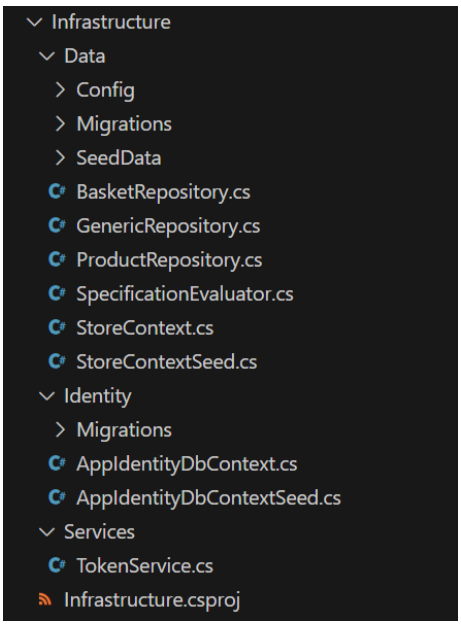

Рис. 2.7. Структура проекту Infrastructure в VS Code

Проект Ядра містить сутності, притаманні нашому конкретному проекту. Під сутностями мається на увазі окремі об’єкти – класи, що містять властивості, притаманні даному об’єкту. Як приклад можна навести сутність Продукт, так як розроблювана система призначена для продажу товарів. Зазначимо, що увесь проект побудовано навколо таких сутностей, які є незалежними об’єктами.

Структурний вигляд проекту Core у VS Code наведено на рис.2.8.

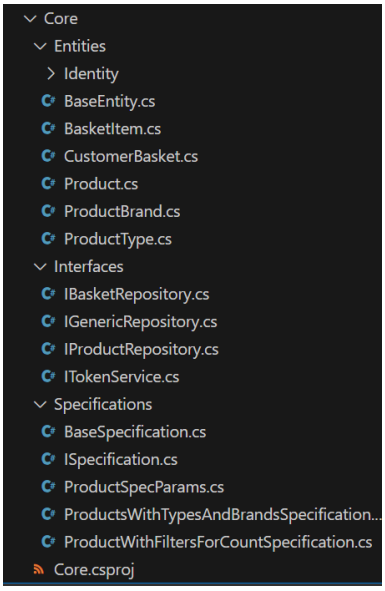

Рис. 2.8. Структура проекту Core в VS Code

Розглядаючи зовнішню частину – фронт-енд даної програми, структура програми можна представити у вигляді, наведеному на рис.2.10.

Користувальницький інтерфейс розробленого програмного продукту реалізовано у проекті Client. Даний проект, як і попередні, має компонентну структуру, тобто складові програми (кошик, процес оплати, сторінка товарів, головна сторінка, тощо) виступають у якості модулів.

Структурний вигляд проекту Client у VS Code наведено на рис.2.9.

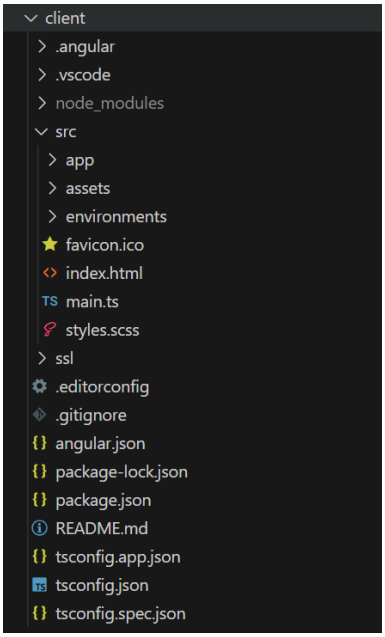

Рис. 2.9. Структура проекту Client в VS Code

Рис. 2.10. Структура програмного інтерфейсу

Як можна побачити з рисунку, структура інформаційної системи складається з наступних елементів:

- головна сторінка;
- каталог товарів з можливістю сортування, фільтрації за категоріями та виробниками продуктів та пошуком товарів;
- власна сторінка для кожного товару з детальною інформацією;
- сторінка з кошиком, у якому містяться обрані товари;
- сторінка оформлення замовлення.

Можливості користувача при взаємодії з інформаційною системою краще всього ілюструє діаграма сценаріїв використання, яку наведено на рис.2.11.

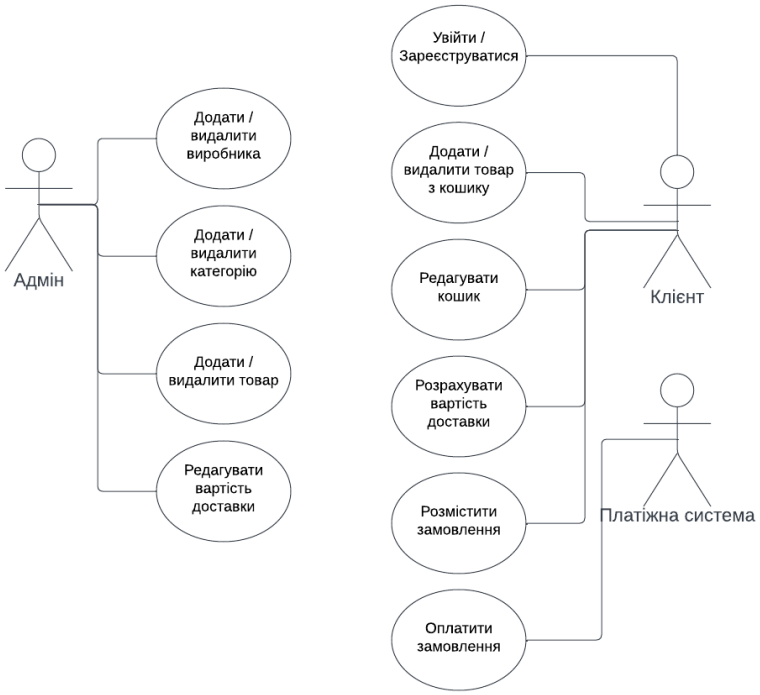

Рис. 2.11. Діаграма сценаріїв використання програми

Сценарій використання розробленої інформаційної системи з боку користувача наведено на рис.2.12, тобто наведено діаграму діяльності.

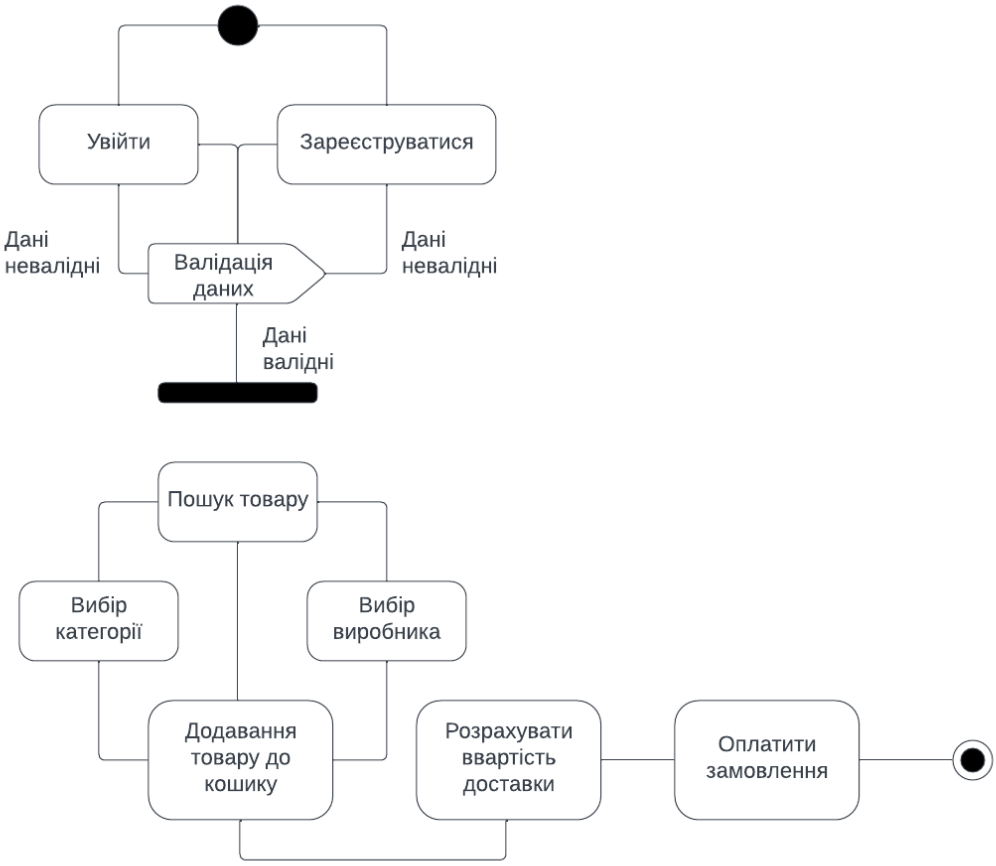

Рис. 2.12. Діаграма діяльності

2.5. Обґрунтування та організація вхідних та вихідних даних програми

Вхідні та вихідні дані, які отримує, обробляє та повертає розроблена інформаційна система, зберігаються у форматі JSON.

JSON (англ. JavaScript Object Notation) є текстовим форматом даних, який було засновано на JavaScript, але може бути використаний в будь-якій мові програмування.

Передані вхідні дані від клієнта до БД відображаються за допомогою резидентної системи керування базами даних Redis (рис.2.13).

Redis є резидентною системою керування базами даних класу. Redis має відкритий вихідний код та працює по типу "ключ - значення". Система використовується для реалізації баз даних, кешів, тощо.

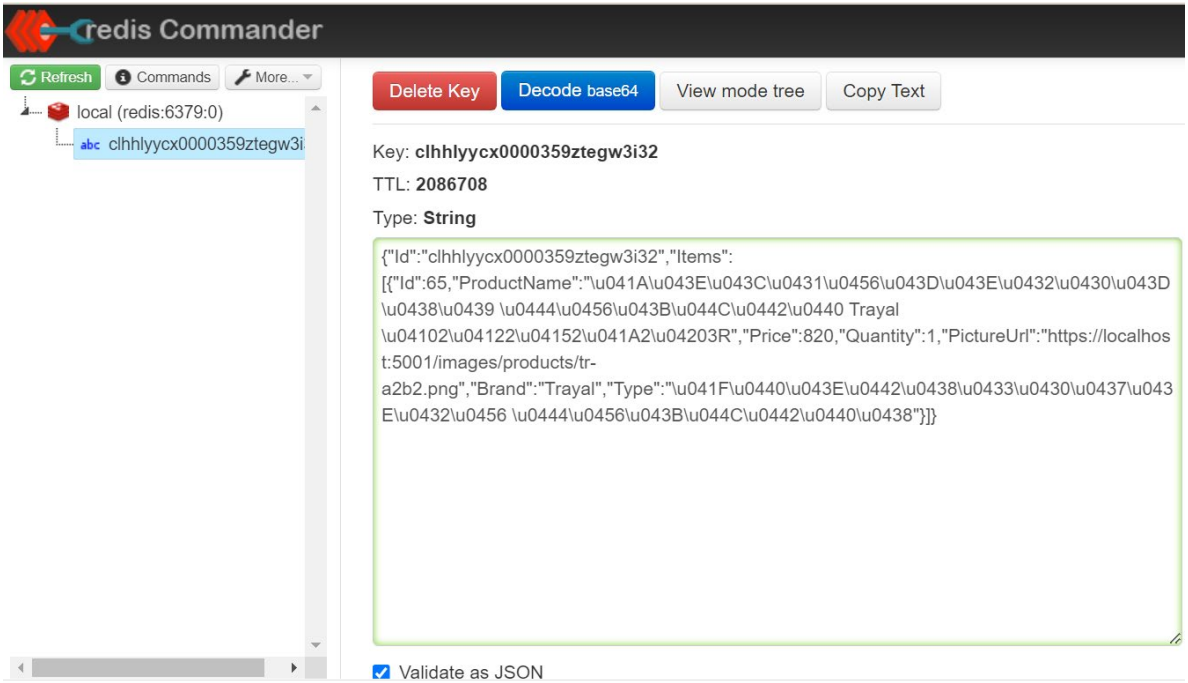

Рис. 2.13. Отримання кошику клієнта Redis

Для більш зручного представлення вхідних даних використовується API платформа Postman. Вигляд запиту наведено на рис. 2.14.

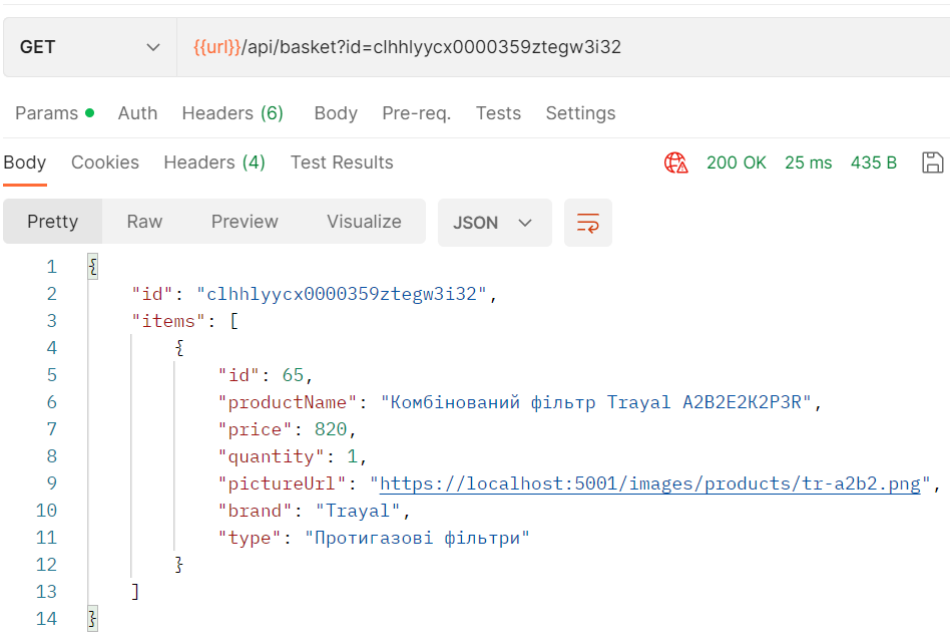

Рис. 2.14. Отримання кошику клієнта Postman

Для передачі вхідних даних використовується об’єкт CustomerBasket, у якому зберігається детальна інформація про створене клієнтом замовлення, в тому числі кількість та характеристики товарів, що визначаються окремим об’єктом BasketItem. Поля об’єкту CustomerBasket наведені у табл. 2.1. Поля об’єкту BasketItem наведені у табл. 2.2.

Таблиця 2.1

Об’єкт «CustomerBasket»

Назва поля	Тип даних
Id	string
Items	List<BasketItem>

Таблиця 2.2

Об’єкт «BasketItem»

Назва поля	Тип даних
Id	int
ProductName	string
Price	decimal
Quantity	string
PictureUrl	string
Brand	string
Type	string

Вихідні дані від інформаційної системи надаються так само у текстовому форматі JSON. Для передачі вихідних даних використовується об’єкт Product, поля якого наведені у табл. 2.3. Тип та бренд товару представлені у вигляді окремих сутностей – ProductType (табл. 2.4) та ProductBrand (табл. 2.5).

Для більш зручного представлення вихідних даних використовується API платформа Postman. Вигляд запиту наведено на рис. 2.15.

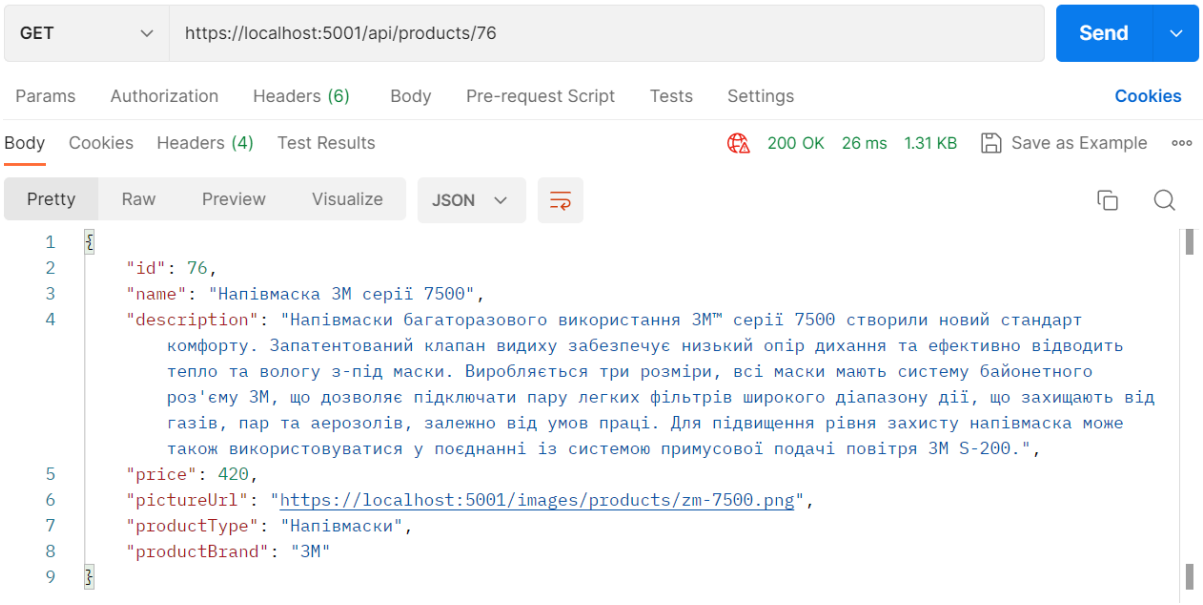

Рис. 2.15. Отримання вихідних даних Postman

Таблиця 2.3

Об’єкт «Product»

Назва поля	Тип даних
Id	int
Name	string
Description	string
Price	decimal
PictureUrl	string
ProductType	ProductType
ProductTypeId	int
ProductBrand	ProductBrand
ProductBrandId	int

Таблиця 2.4

Об’єкт «ProductType»

Назва поля	Тип даних
Id	int
Name	string

Таблиця 2.5

Об’єкт «ProductBrand»

Назва поля	Тип даних
Id	int
Name	string

Вихідні дані, отримані з бази даних у форматі JSON, відображаються на графічному інтерфейсі програми, тобто на front-end частині. Для візуального відображення даних використовуються технології TypeScript, HTML та SCSS.

2.6. Опис розробленого програмного продукту

Веб-орієнтована інформаційна система, що була розроблена в ході даної кваліфікаційної роботи має на меті впровадження зручного та швидкого доступу до товарних позицій магазину, перегляду інформації стосовно обраного товару, оформлення замовлення з подальшою можливістю його оплати.

Окрім цього програмний продукт виконує ряд важливих функцій з точки зору бізнесу, а саме: сприяє розширенню клієнтури та, у свою чергу, збільшенню виторгу компанії; підвищенню лояльності клієнтів, зростанню популярності та конкурентоспроможності компанії на ринку продажів засобів індивідуального захисту в Україні.

2.6.1. Використані технічні засоби

Для розробки, тестування та налагодження додатку була використана система з наступними технічними характеристиками:

- процесор: AMD Ryzen 7 4700U, 8 ядер, 8 потоків, частота 2ГГц;
- відеоадаптер: AMD Radeon Graphics 1600 мГц;
- оперативна пам’ять: 8 Гб, 3200 МГц, DDR4;

– накопичувач: 512 ГБ SSD.

Наведені вище характеристики є рекомендованими для стабільної та безперервної роботи розробленої інформаційної системи.

2.6.2. Використані програмні засоби

З метою розробки та тестування веб-орієнтованого додатку було використано наступне програмне забезпечення:

– Visual Studio Code з встановленим розширенням SQLite;
– Angular CLI;
– Node.js, NPM;
– Postman;
– Stripe.

Visual Studio Code, який також зазвичай називають VS Code —редактор вихідного коду для Windows, Linux та macOS. До основного функціоналу даного програмного продукту можна віднести підсвічування синтаксису, підтримку налагодження, інтелектуального завершення коду, рефакторинг коду та вбудований Git [16].

VS Code дозволяє встановити спеціальне розширення для дослідження та запитів до баз даних SQLite [17].

До переваг даного розширення можна віднести наступні:

– передача запитів до БД SQLite і перегляд результатів в таблиці;
– експорт результатів запитів у формати json, csv та html;
– список баз даних, таблиць і стовпців відображаються на бічній панелі;
– автодоповнення для ключових слів SQLite.

Node.js є асинхронним середовищем виконання JavaScript, яке кероване подіями та призначене для створення масштабованих мережевих додатків [18]. Node.js має відкритий вихідний код та не залежить від програмного забезпечення операційної системи. Написання JavaScript-коду в текстовому редакторі не

передбачає виконання жодних завдань до його запуску. Для цього і є необхідним середовище виконання Node.js.

NPM (Node Package Manager) - бібліотека та реєстр для програмних пакетів на JavaScript. NPM є онлайн-репозиторієм для публікації проектів Node.js з відкритим кодом. Також пакетний менеджер є утилітою командного рядка для взаємодії із згаданим репозиторієм, яка допомагає встановлювати та керувати версіями пакунків і залежностями [19].

Postman - платформа для створення та використання API. Postman спрощує кожен крок життєвого циклу API та оптимізує співпрацю, з метою прискорення розробки API [20].

До переваг платформи Postman можна віднести наступні:

- репозиторій API, що дозволяє легке збереження та співпрацю з усіма артефактами API на одній центральній платформі;
- набір інструментів, які допомагають прискорити життєвий цикл API - від проектування, тестування, документування та імітації до публікації та відкриття API.

Stripe - це платформа для обробки онлайн-платежів та обробки кредитних карток для бізнесу. Stripe дозволяє безпечно та ефективно обробляти кошти за допомогою кредитної картки або банку та переказувати ці кошти на рахунок продавця [21].

2.6.3. Виклик та завантаження програми

Клієнтська та серверна частина розробленого програмного продукту тісно пов’язані поміж собою.

Для запуску серверної частини необхідно виконати наступні дії в командному рядку:

- з кореневої директорії проекту перейти то директорії api, яка є відповідальною за серверну частину додатку;

– запустити команду «dotnet watch --no-hot-reload» (рис.2.16). Ця команда запускає сервер та вимикає функцію постійного перезавантаження з метою відстежування зміни у файлах. Зупинка та повторний виклик даної команди є необхідним лише при значних змінах в API-частині додатку (наприклад додавання нових модулів).

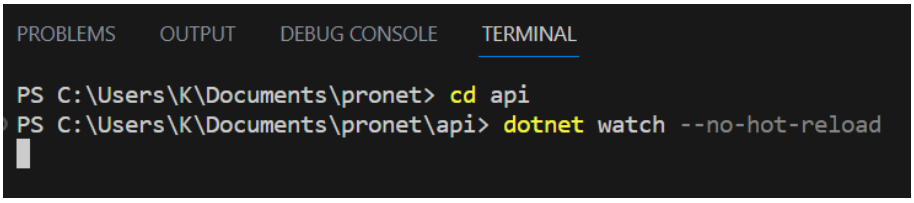

Рис. 2.16. Команда запуску серверної частини додатку

Після запуску серверної частини, термінал повідомляє про успішне з’єднання з базою даних, як відображено на рис.2.17.

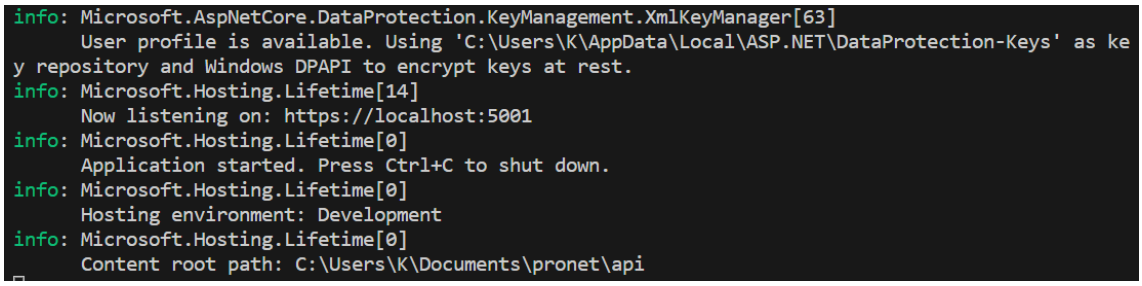

Рис. 2.17. Повідомлення про успішний запуск серверу

Для запуску клієнтської частини необхідними до виконання є наступні дії в командному рядку:

– з кореневої директорії проекту перейти то директорії client, яка є відповідальною за клієнтську частину додатку;
– запустити команду «ng serve» (рис.2.18). Ця команда створює і обслуговує клієнтську частину додатку. Одночасне відстеження змін у файлах програми дозволяє отримувати повідомлення про можливі помилки та несправності при роботі.

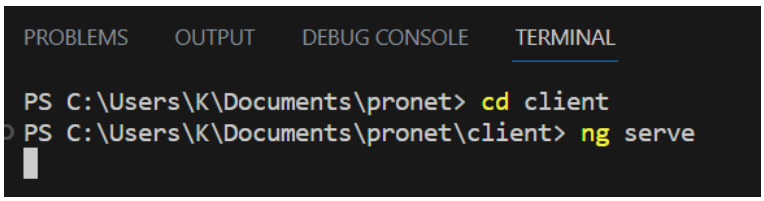

Рис. 2.18. Команда запуску клієнтської частини додатку

Наступним кроком, як можна побачити з рис.2.19, відбувається успішний запуск клієнтської частини додатку, який розміщено на порті localhost:4200.

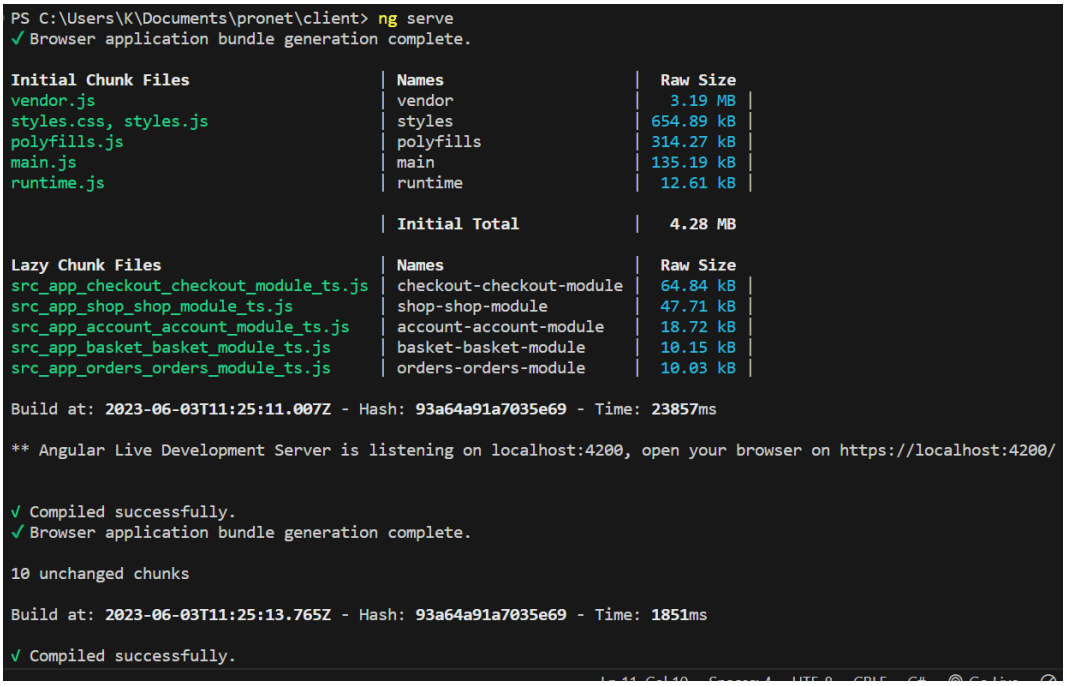

Рис. 2.19. Повідомлення про успішний запуск клієнта

Після запуску обох частин додатку, необхідно у будь-якому браузері ввести адресу https://localhost:4200/, після чого буде завантажено головну сторінку розробленого веб-додатку.

2.6.4. Опис інтерфейсу користувача

Після запуску програми користувач потрапляє на головну сторінку магазину (рис.2.20). Зверху, в так званій «шапці» сторінки міститься навігація на сторінку товарів, вхід та реєстрацію користувачів в системі. Основну частину

сторінки займає слайдер, що у подальшому може бути використаний для відображення корпоративної інформації, переліку товарів, тощо.

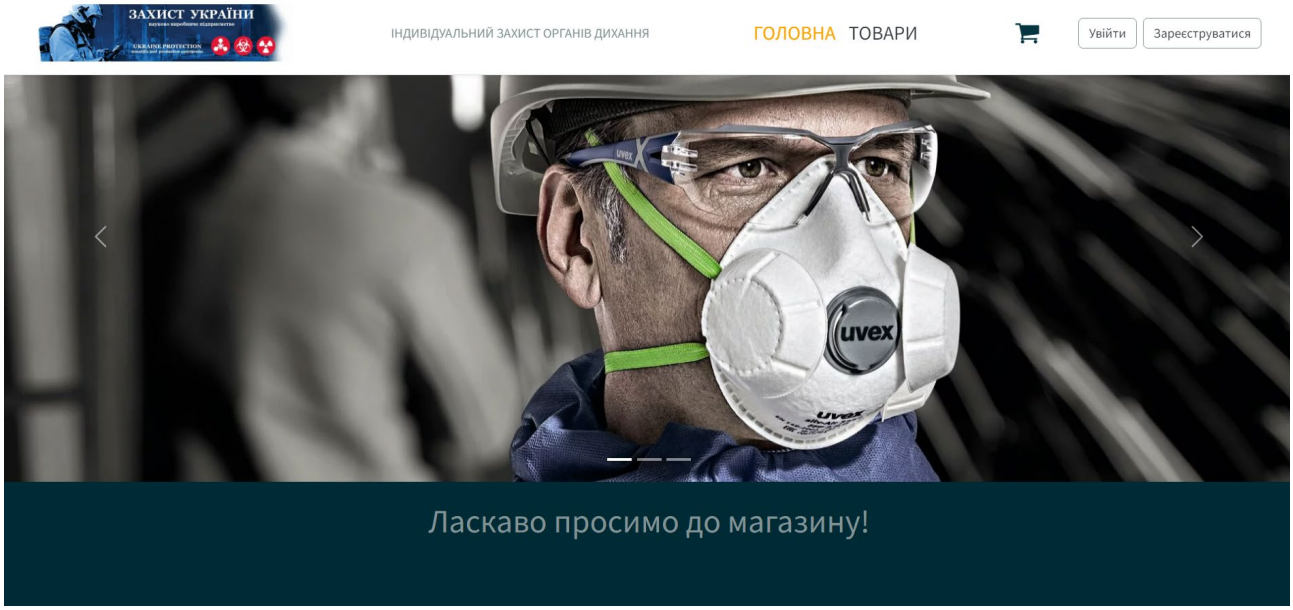

Рис. 2.20. Головна сторінка додатку

Наступним кроком користувач має декілька варіантів: увійти до системи, зареєструватися в системі, або перейти на сторінку товарів пропустивши попередні кроки.

Розглянемо детальніше процес авторизації. Як показано на рис.2.21, при спробі увійти до системи користувач має ввести у відповідні поля електронну адресу та пароль. Дані поля, як показано на рис.2.22, мають перевірку, або валідацію (мають бути обов’язково заповнені відповідним патерном). Кнопка входу залишається неактивною, допоки система не отримає валідні дані (рис.2.23).

Процес авторизації викликається так само при спробі неавторизованого користувача перейти до оплати замовлення після додавання товару до кошика.

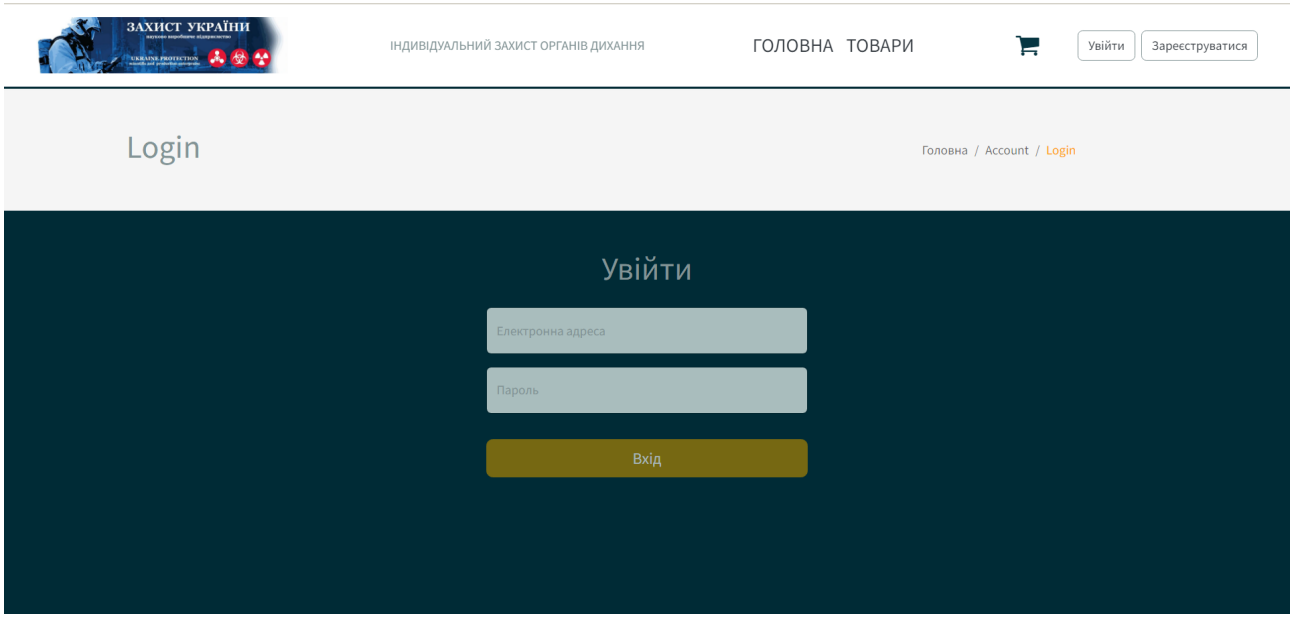

Рис. 2.21. Сторінка авторизації

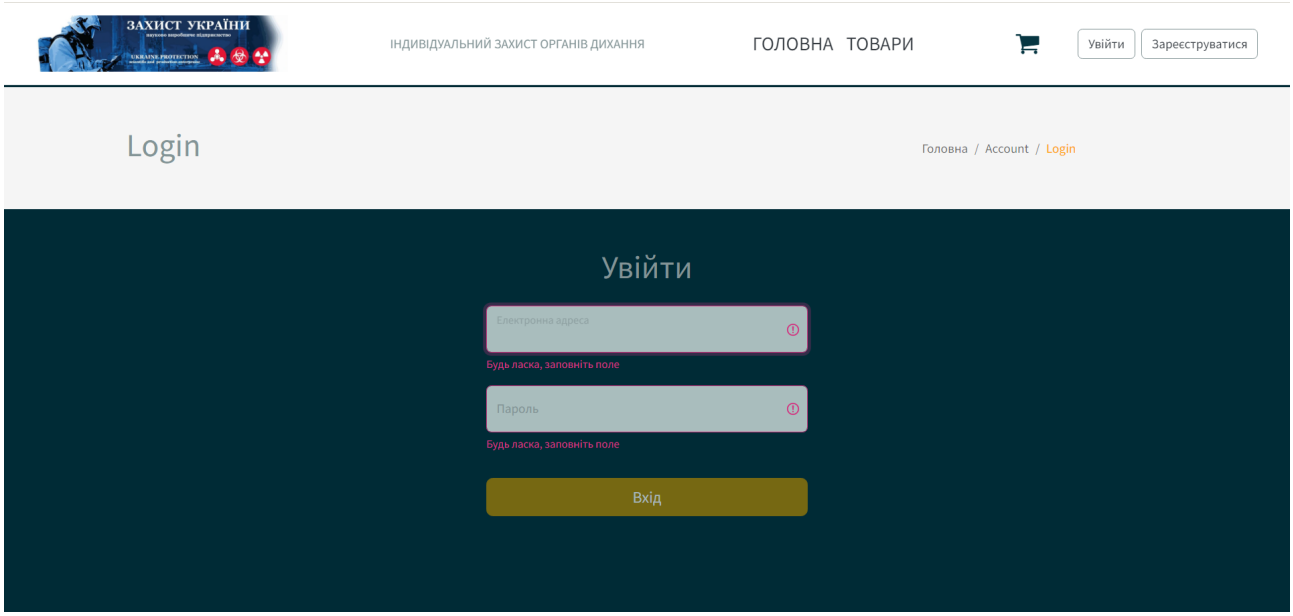

Рис. 2.22. Введення даних (кнопка неактивна)

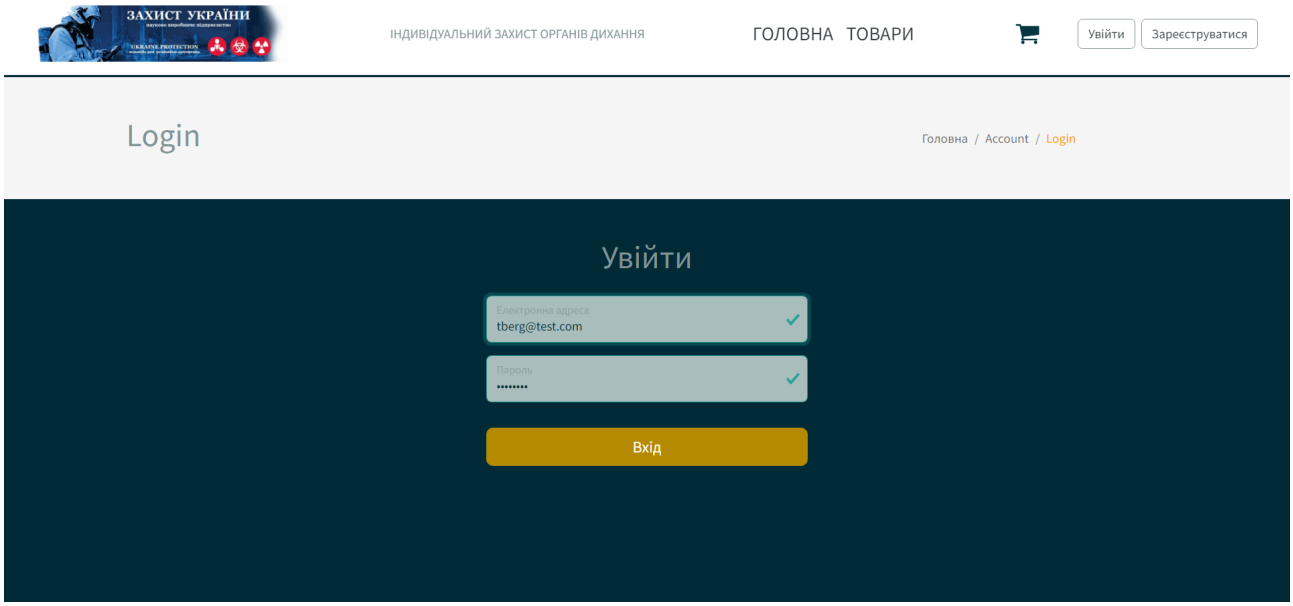

Рис. 2.23. Введення даних (кнопка активна)

Наступним розглянемо процес реєстрації нового користувача на сайті. На рис.2.24 зображена сторінка реєстрації. Як можна побачити з рисунку, користувач має визначитися з ім’ям, яке буде відображатися на сайті після авторизації, електронним адресом та паролем.

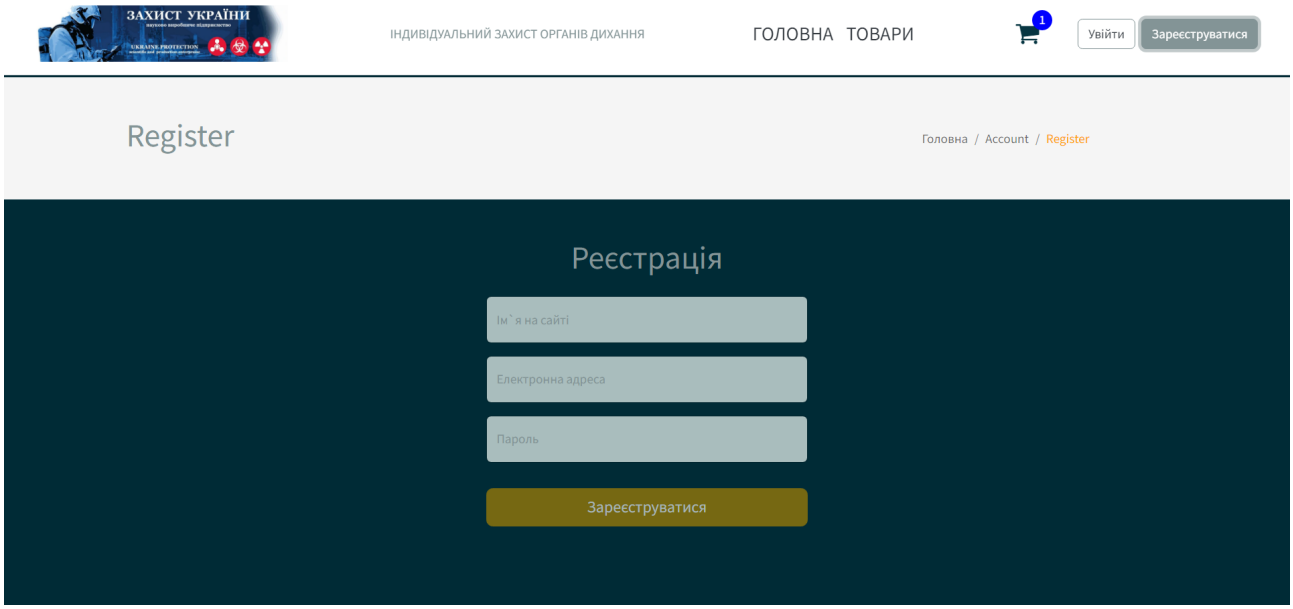

Рис. 2.24. Сторінка реєстрації

Поле ім’я приймає на вхід будь які числові, буквенні та спеціальні символи. На відміну від цього, поля адреси та пароля мають патерни, які слід враховувати при спробі реєстрації (рис.2.25). Після вдалої спроби, кнопка реєстрації набуває вигляду, як показано на рис.2.26.

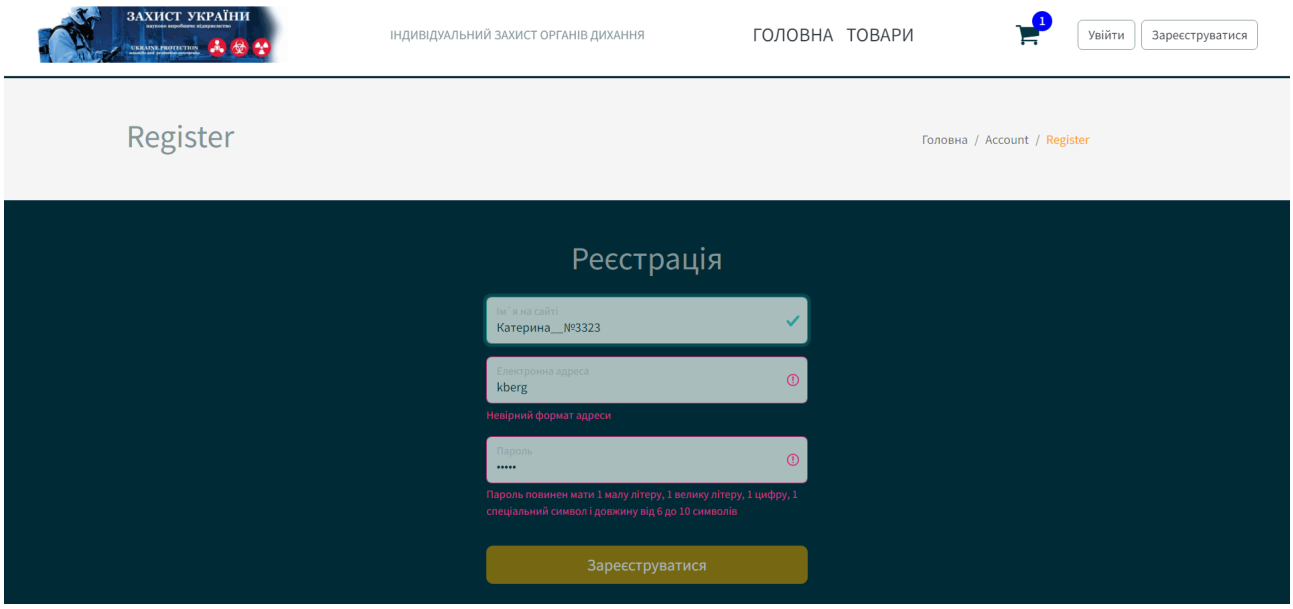

Рис. 2.25. Сторінка реєстрації (дані невірні)

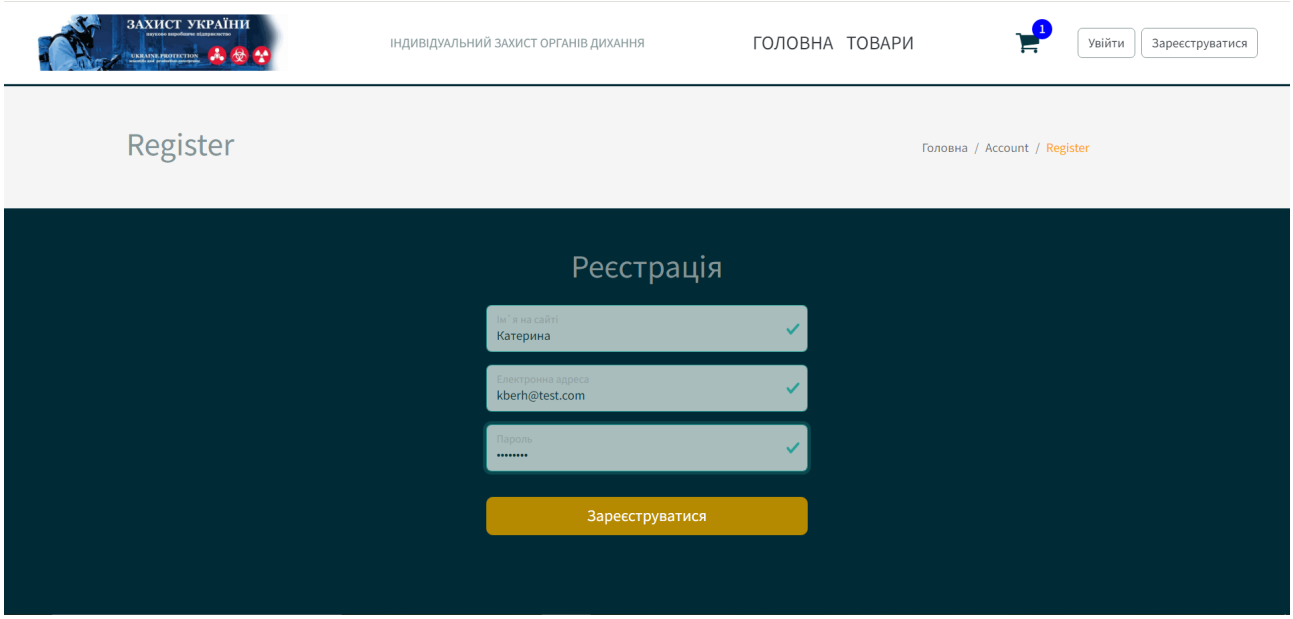

Рис. 2.26. Сторінка реєстрації (дані вірні)

Далі перейдемо до сторінки каталогу магазину. З рис.2.27 можна побачити, що структура даної сторінки включає в себе наступні компоненти: сортування (за алфавітом, зростанням та спаданням ціни), фільтрація товарів за брендами та типами (рис.2.28) та стрічки пошуку товарів (рис.2.29). При наведенні на картинку товарної позиції з’являється можливість одразу додати товар до кошику, або перейти на сторінку даного товару, де відображається детальна інформація.

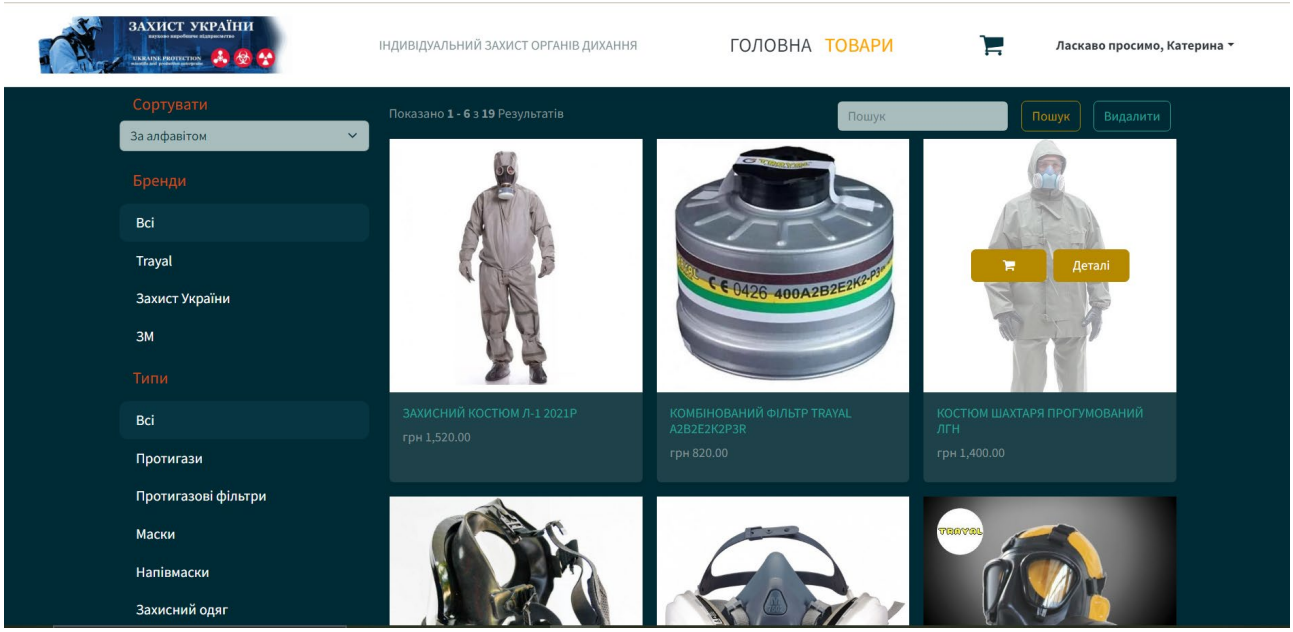

Рис. 2.27. Каталог товарів

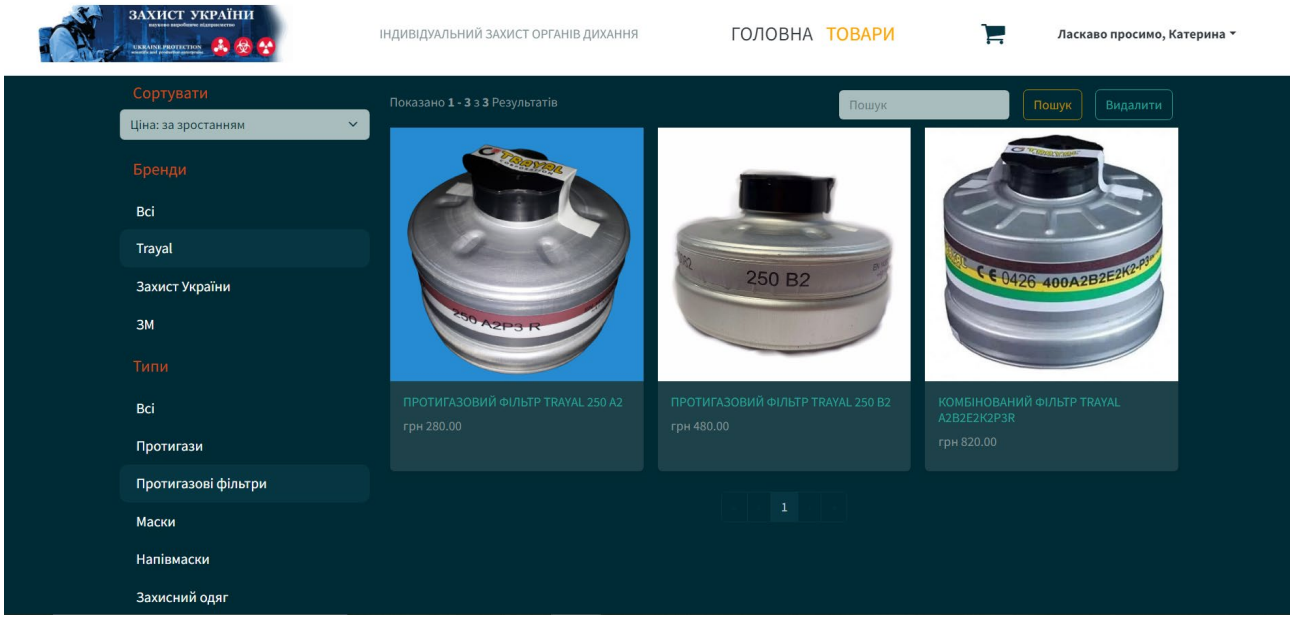

Рис. 2.28. Каталог товарів. Сортування та фільтрація

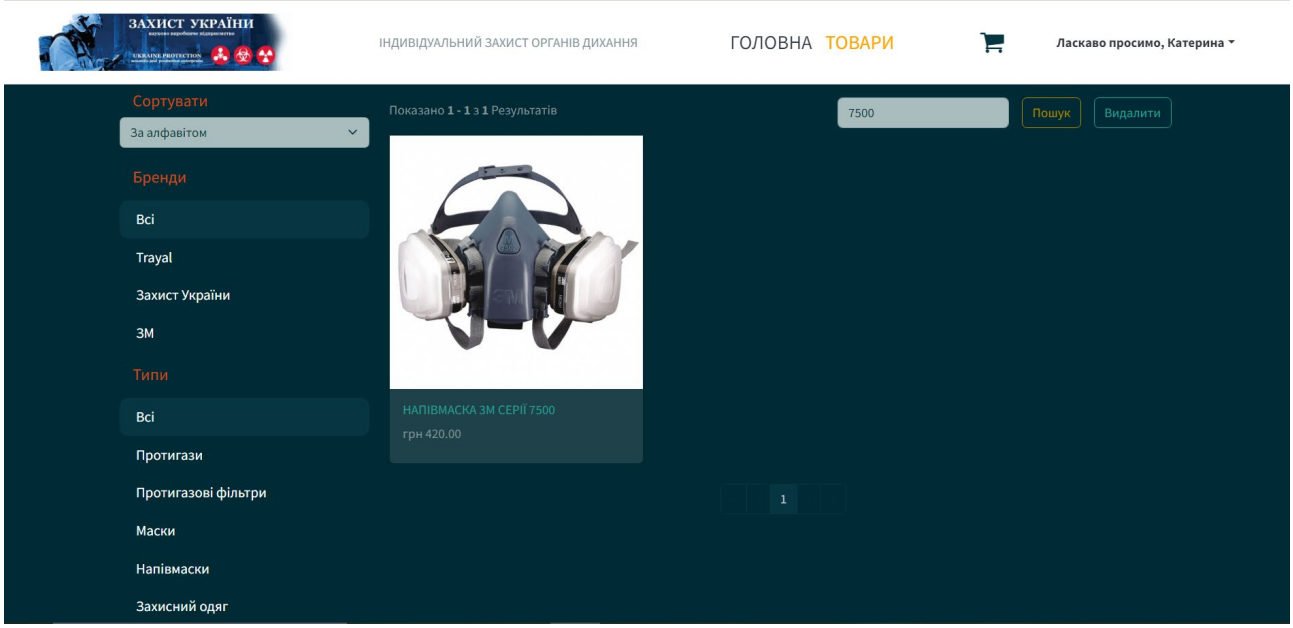

Рис. 2.29. Каталог товарів. Пошук

Розглянемо більш детально сторінку окремого товару. На рис.2.30 представлено вигляд даної сторінки. Основною функцією є зміна кількості товарів перед додаванням його до кошику. Після того, як позицію було додано до кошику, сторінка відображає відповідне повідомлення, яке вказує, яка саме кількість даного товару вже є присутньою у кошику клієнта (рис.2.31).

Рис. 2.30. Сторінка товару

Рис. 2.31. Сторінка товару. Повідомлення про наявність товару у кошику

Наступним перейдемо до кошику покупця. Припустимо, що клієнт додав до кошику два товари, як показано на рис.2.32. З рисунку можна побачити, що до перегляду у таблиці доступна наступна інформація: найменування товару, ціна за одну позицію, кількість, загальна ціна з урахуванням кількості, можливість видалення товару. Під інформаційною таблицею розміщується загальна інформація щодо замовлення з урахуванням доставки. Типи та вартість доставки

можна переглянути та обрати у процесі оплати замовлення, який ми розглянемо далі.

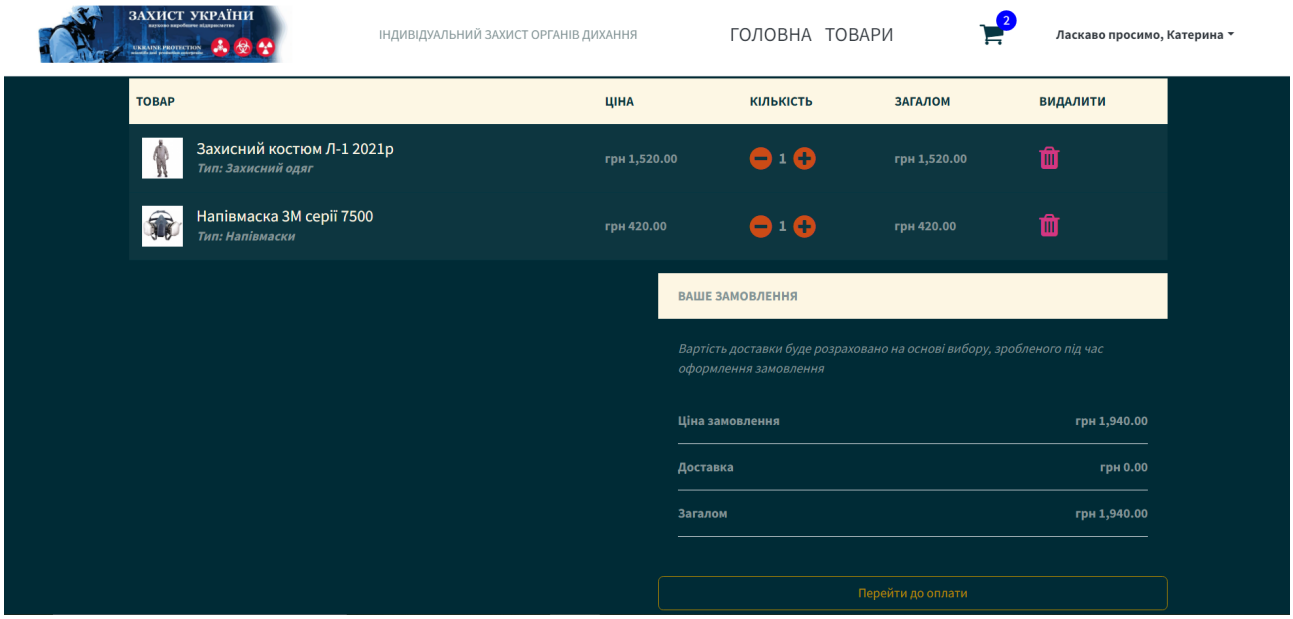

Рис. 2.32. Сторінка кошику

Сторінка оплати має вигляд як показано на рис.2.33. Процес складається з чотирьох послідовних кроків, перехід до жодного з яких є неможливим допоки не буде заповнено попередній. Така функція була впроваджена задля зручного та зрозумілого шляху користувача через процес оплати товару.

Першим кроком є заповнення адреси доставки. Перевірка даних включає в себе обов’язкове заповнення усіх полів. Після цього клієнт має можливість зберегти адресу як основну для запобігання повторного введення.

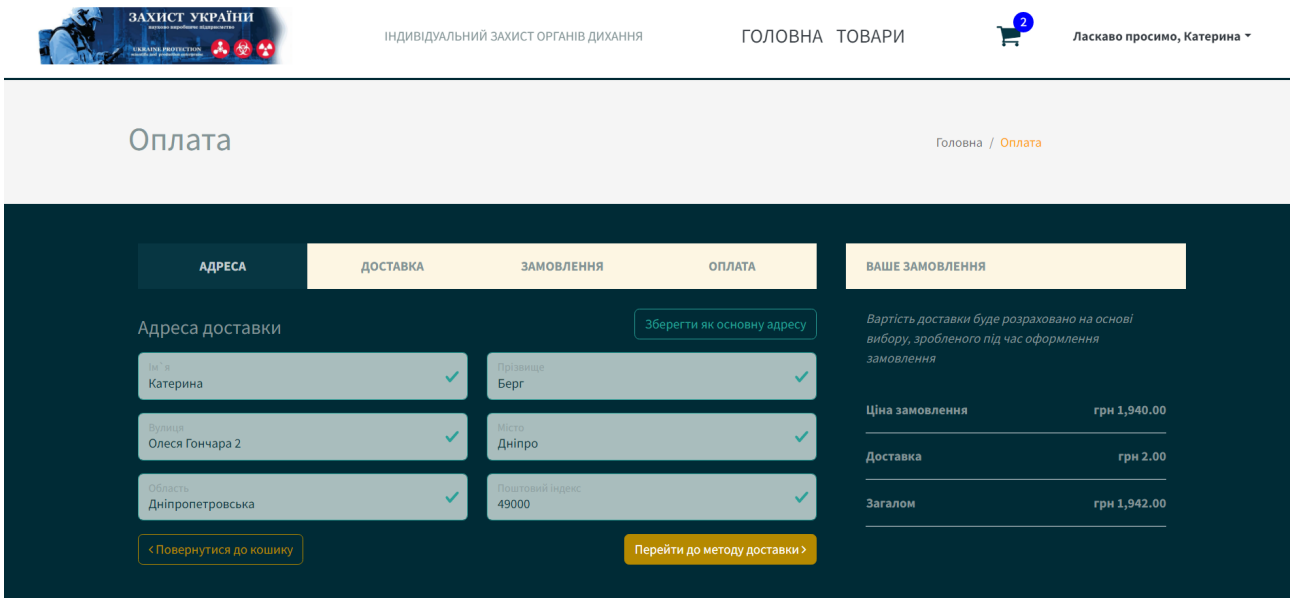

Рис. 2.33. Сторінка оплати (введення адреси)

Після введення адреси користувач потрапляє на сторінку доставки (рис.2.34). На даній сторінці покупець обирає найбільш вигідний для себе варіант доставки (термін та вартість доставки зазначені), після чого обрана доставка автоматично запам’ятовується для наступних покупок, а клієнт має можливість перейти до наступного кроку оформлення замовлення.

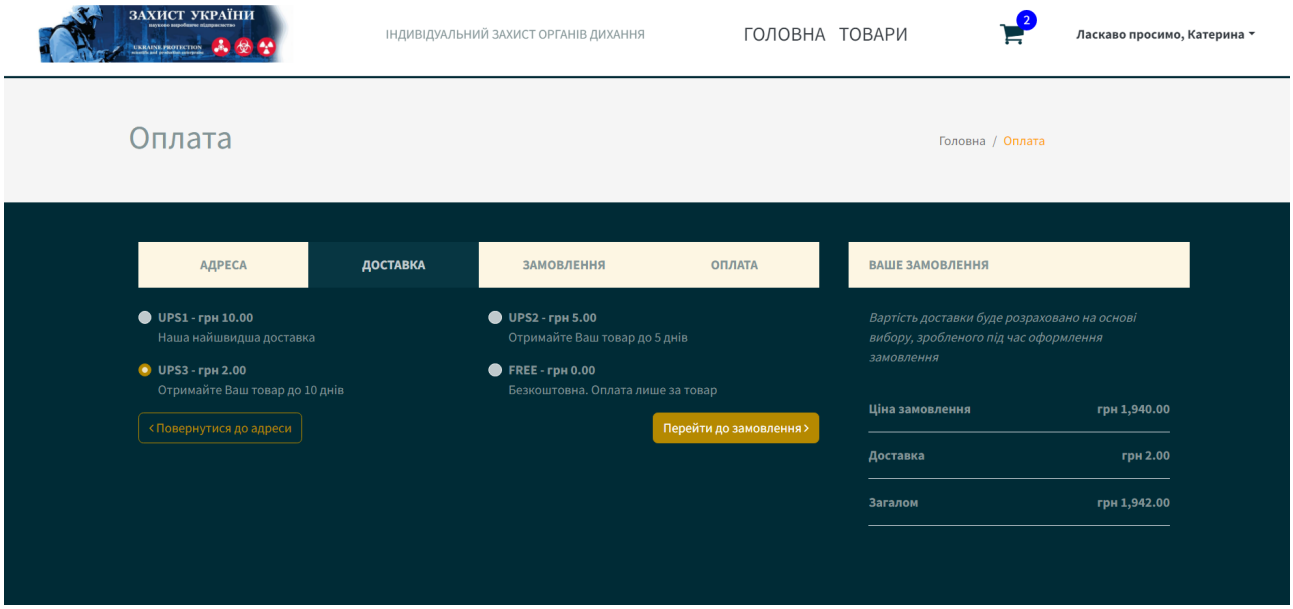

Рис. 2.34. Сторінка оплати (вибір доставки)

На наступній сторінці – сторінці перегляду замовлення (рис.2.35), клієнт має можливість переконатися у відповідності обраних позицій, додатково перевірити інформацію, побачити розраховану загальну суму с урахуванням обраного методу доставки, та перейти до вибору способу оплати.

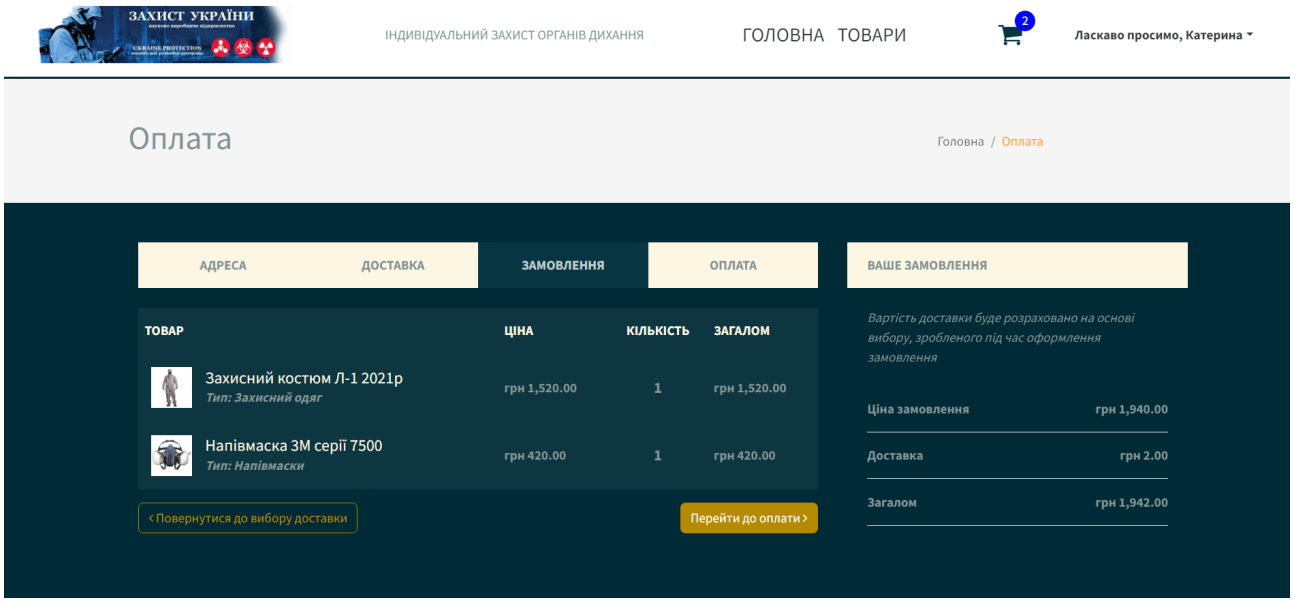

Рис. 2.35. Сторінка оплати (перегляд замовлення)

Останнім кроком оформлення замовлення є сторінка введення платіжних даних, як показано на рис.2.36. Для процесу оплати була налаштована платіжна інфраструктура Stripe, в якій у свою чергу створено так званий вхідний веб-хук (webhook). Веб-хуки використовується з метою отримання оновлень в режимі реального часу та є особливо корисними для асинхронних подій.

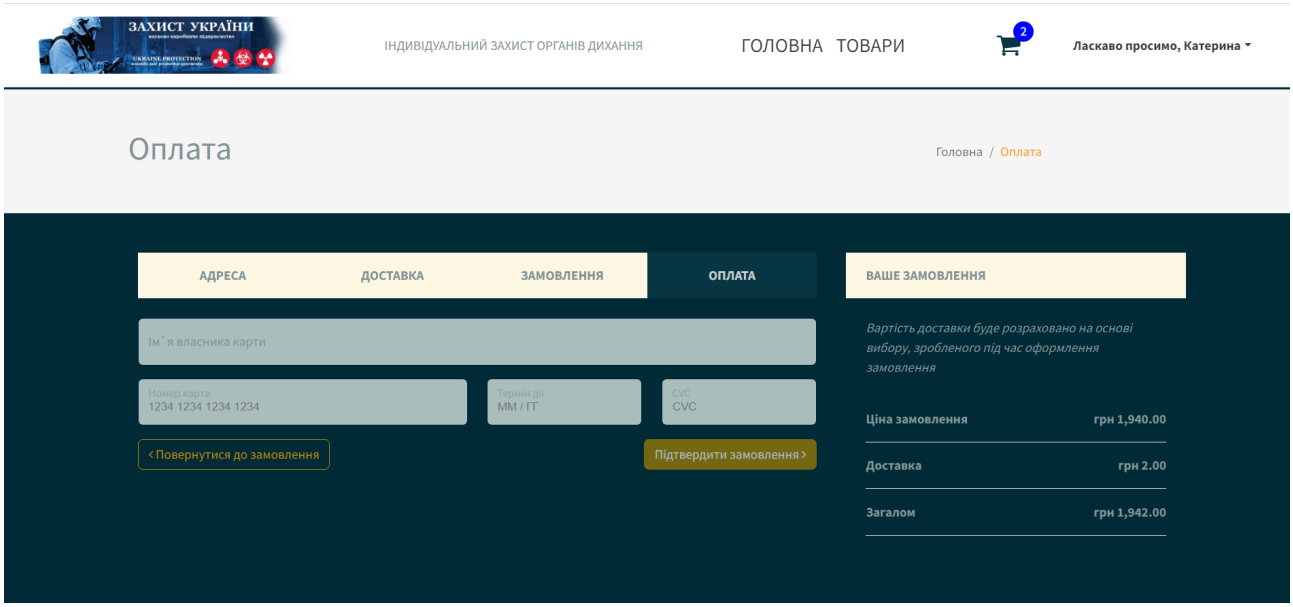

Рис. 2.36. Сторінка оплати (оплата замовлення)

Для тестування будемо використовувати номер картки з потребою в аутентифікації (підтвердження). Це означає, що після введення даних (рис.2.37) з’являється відповідна процедура прийняття або відхилення платежу (рис.2.38).

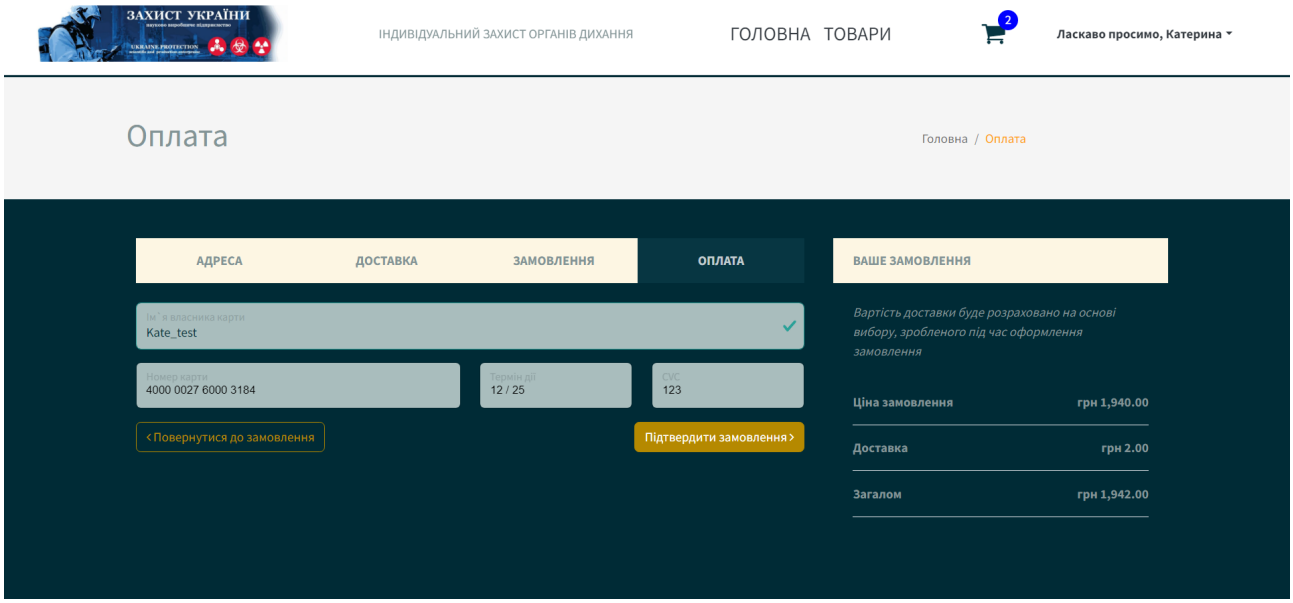

Рис. 2.37. Оплата замовлення (введення даних)

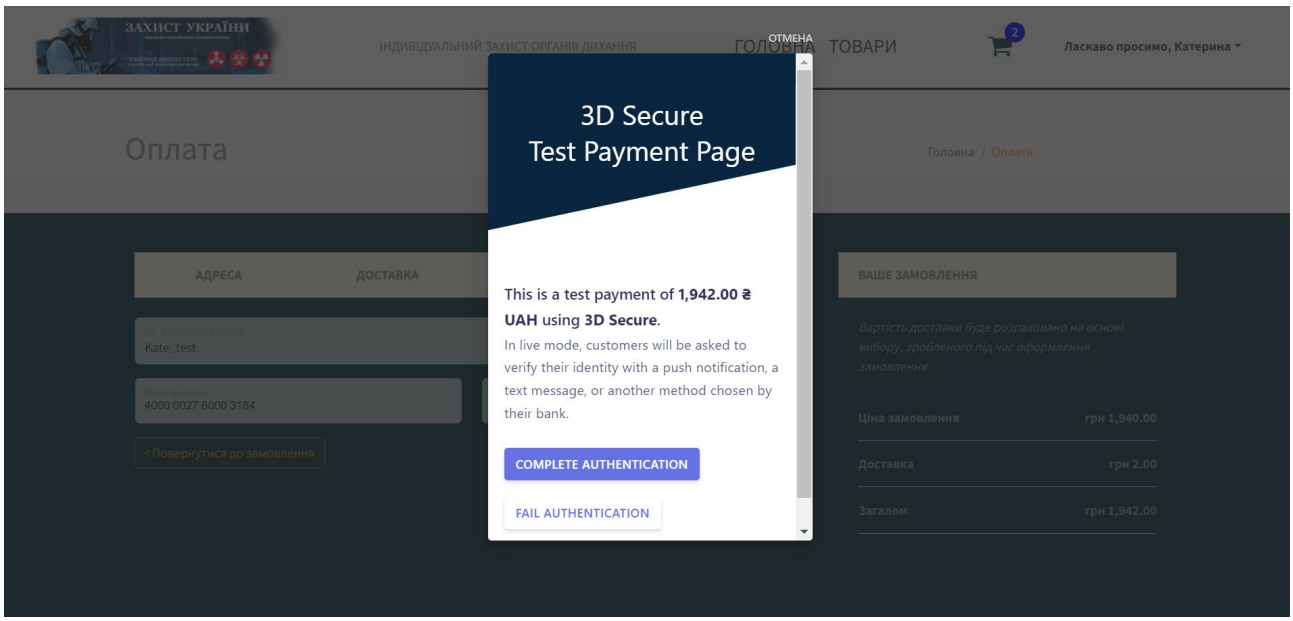

Рис. 2.38. Оплата замовлення (аутентифікація)

Для першого випадку розглянемо відхилення платежу. Після вибору даної опції користувач все ще залишається у кошику з метою повторної спроби оплати, але на сторінці замовлень, як можна побачити з рис.2.39 наявною є оновлена інформація стосовно замовлення, яка вказує на неуспішну оплату.

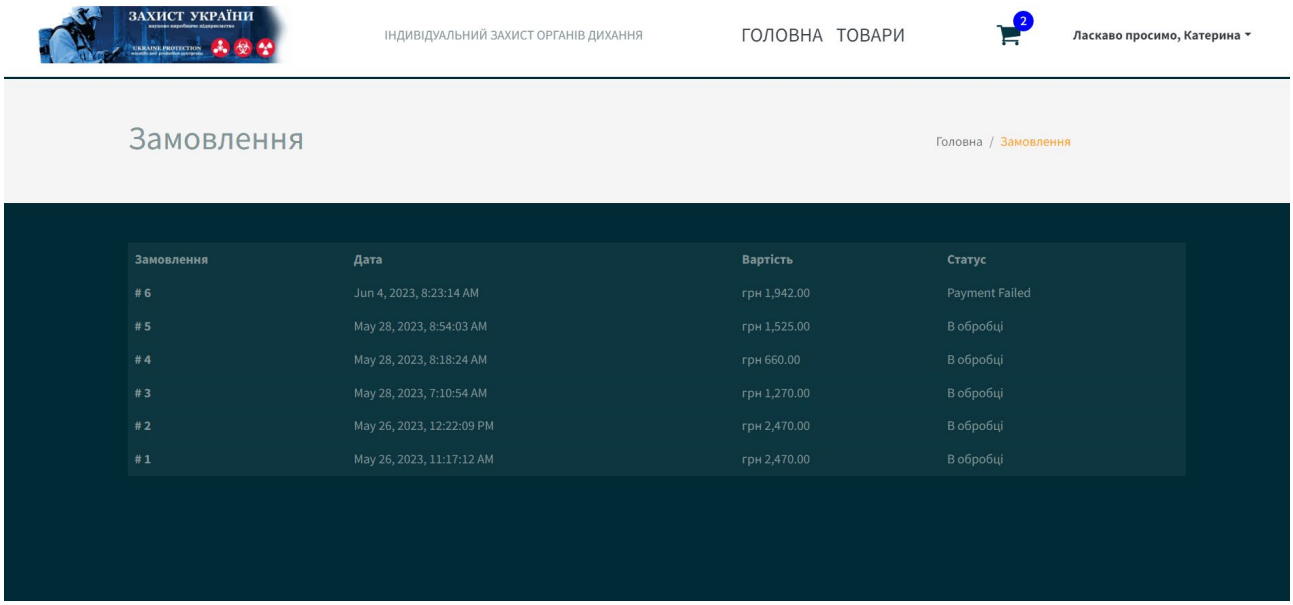

Рис. 2.39. Оплата замовлення (оплата неуспішна)

Другий випадок – підтвердження платежу. Після відповідної дії замовлення переходить до статусу «оплата отримана», завантажується сторінка підтвердження оплати, як показано на рис.2.40. Оновлену інформацію про успішний платіж також можна побачити на сторінці замовлень.

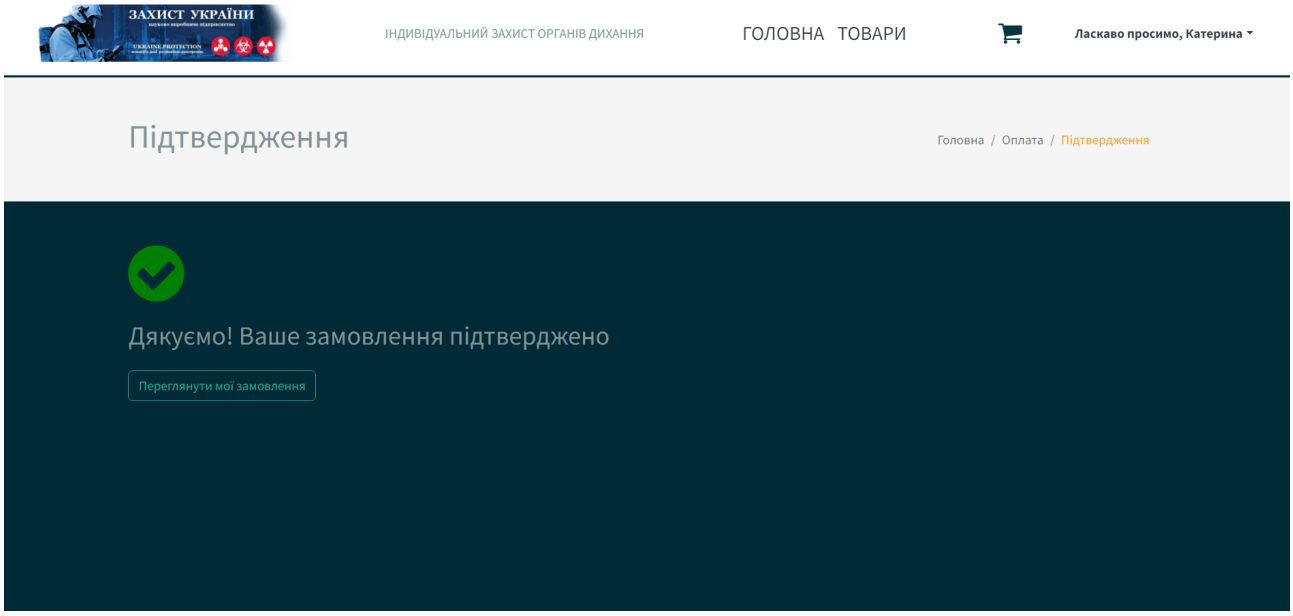

Рис. 2.40. Оплата замовлення (оплата успішна)

Для переконання в тому, що оплата була отримана, можна звернутися безпосередньо до розділу «Payments» в особистому кабінеті Stripe, де відображено інформацію стосовно статусу отриманих платіжок (рис.2.41).

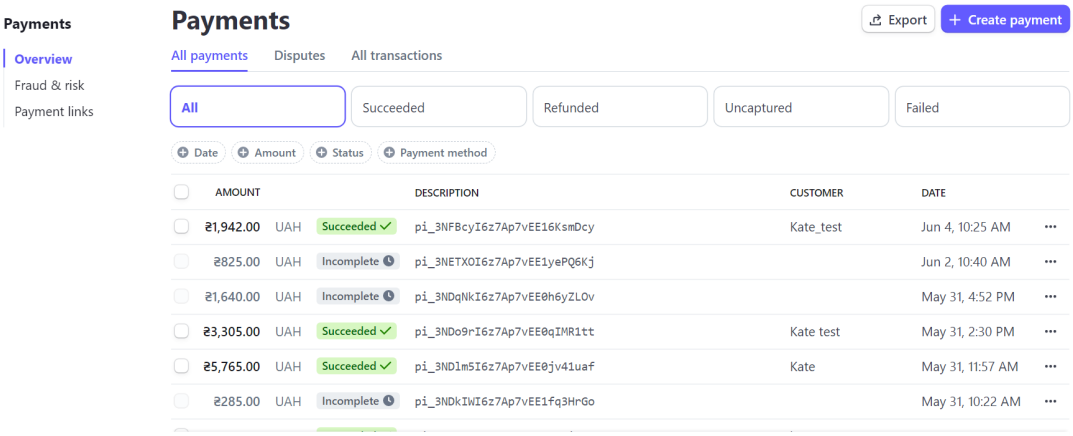

Рис. 2.41. Відображення статусу на сторінці Stripe

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Вхідні дані:

- передбачуване число операторів – 1500;
- коефіцієнт корекції програми в ході її розробки – 0,05;
- коефіцієнт складності програми – 1,3;
- годинна заробітна плата програміста – 187,9 грн/год;
- коефіцієнт збільшення витрат праці в наслідок недостатнього опису задачі – 1,4;
- коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1;
- вартість машино-години ЕОМ – 28,7 грн/год;
- кількість розробників – 1.

Заробітна плата за годину Junior TypeScript Software Engineer була вирахувана згідно «Української спільноти програмістів (DOU)» [12]. Станом на 2022 рік зарплата Junior TypeScript Engineer вар'юється від 613$ до 1200$. Вирахувавши середню заробітну плату програміста маємо плату 906$ у місяць. При курсі валют НБУ на кінець грудня 2022 року один американський долар дорівнює 36,5 грн, тому середня зарплата в гривнях дорівнює 33 069 грн. При стандартному графіку (176 годин/місяць) зарплата за годину буде становити близько 187,9 грн.

Виходячи з того, що для розробки даної системи необхідною є значна потужність ПК для безперервної роботи серверу, зберігання та підтримки великої кількості даних на сервері, вдалим рішенням буде оренда ноутбуку.

Вартість оренди потужного ноутбуку на місяць становить 4590 грн [13]. При стандартному графіку (160 годин/місяць) вартість машино-години ЕОМ за годину роботи буде становити 28,7 грн.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{\text{отл}} + t_{\text{д}}, \qquad (3.1)$$

де t_o - витрати праці на підготовку й опис поставленої задачі (приймається 50 людино-годин);

t_u - витрати праці на дослідження алгоритму рішення задачі;

t_a - витрати праці на розробку блок-схеми алгоритму;

t_n - витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ - витрати праці на налагодження програми на ЕОМ;

$t_д$ - витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у програмному забезпеченні, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q * C * (1 + p), \qquad (3.2)$$

де q - передбачуване число операторів (1500);

C - коефіцієнт складності програми (1,3);

p - коефіцієнт корекції програми в ході її розробки (0,05).

За формулою (3.2) умовне число операторів в програмі становить:

$$Q = 1500 * 1{,}3 * (1 + 0{,}05) = 2047$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q*B}{(75..85)*k}, \quad (3.3)$$

де B - коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі (1,4);

k - коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності. При стажі роботи від 2 до 3 років він складає 1.

Збільшення витрат праці внаслідок недостатнього опису завдання будемо приймати не більше 50% (B = 1,3).

З урахуванням коефіцієнта кваліфікації k = 1, за формулою (3.3) отримуємо витрати праці на вивчення опису завдання:

$$t_u = \frac{2047*1,3}{85*1} = 31,3, \text{ людино-годин},$$

Витрати праці на розробку алгоритму рішення задачі визначаються за формулою:

$$t_a = \frac{Q}{(20...25)*k}, \quad (3.4)$$

де Q – умовне число операторів програми (2047);

k – коефіцієнт кваліфікації програміста (1).

Згідно формулі (3.4), отримаємо:

$$t_a = \frac{2047}{25*1} = 81,9, \text{ людино-годин},$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20...25)*k}, \quad (3.5)$$

Маючи формулу (3.5) витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{2047}{25*1} = 81{,}9, \text{ людино-годин},$$

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання:

$$t_{\text{отл}} = \frac{Q}{(4..5)*k}, \quad (3.6)$$

Витрати праці на налагодження програми на ЕОМ за умови автономного налагодження одного завдання за формулою (3.6):

$$t_{\text{отл}} = \frac{2047}{5*1} = 409, \text{ людино-годин},$$

– за умови комплексного налагодження завдання:

$$t_{\text{отл}}^{k} = 1{,}5 * t_{\text{отл}}, \quad (3.7)$$

Витрати праці на налагодження програми на ЕОМ за умови комплексного налагодження завдання за формулою (3.7):

$$t_{\text{отл}}^{k} = 1{,}5 * 409 = 613{,}5, \text{ людино-годин},$$

Витрати праці на підготовку документації визначаються за формулою:

$$t_{\text{д}} = t_{\text{др}} + t_{\text{до}}, \tag{3.8}$$

де $t_{\text{др}}$ - трудомісткість підготовки матеріалів і рукопису:

$$t_{\text{др}} = \frac{Q}{(15..20)*k}, \tag{3.9}$$

$t_{\text{до}}$ - трудомісткість редагування, печатки й оформлення документації:

$$t_{\text{до}} = 0{,}75 * t_{\text{др}}, \tag{3.10}$$

Маючи формули (3.8), (3.9) та (3.10) обчислюємо витрати праці на документацію:

$$t_{\text{др}} = \frac{2047}{20*1} = 102{,}35, \text{ людино-годин},$$
$$t_{\text{до}} = 0{,}75 * 102{,}35 = 76{,}76, \text{ людино-годин},$$
$$t_{\text{д}} = 102{,}35 + 76{,}76 = 179{,}11, \text{ людино-годин},$$

Повертаючись до формули (3.1), отримаємо повну оцінку трудомісткості розробки програмного забезпечення:

$$t = 50 + 31{,}3 + 81{,}9 + 81{,}9 + 409 + 179{,}11 = 833{,}2, \text{ людино-годин}.$$

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ КПО включають витрати на заробітну плату виконавця програми $З_{ЗП}$ і витрат машинного часу, необхідного на налагодження програми на ЕОМ:

$$К_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн.} \quad (3.11)$$

Заробітна плата виконавців визначається за формулою:

$$З_{ЗП} = t * С_{ПР}, \text{ грн,} \quad (3.12)$$

де t - загальна трудомісткість, людино-годин (833,2);

$С_{ПР}$ - середня годинна заробітна плата програміста, грн/година.

З урахуванням того, що середня годинна зарплата програміста становить 187,9 грн / год, за формулою (3.12) отримуємо:

$$З_{ЗП} = 833{,}2 * 187{,}9 = 156\,558, \text{ грн,}$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ, визначається за формулою:

$$З_{МВ} = t_{отл} * С_{Мч}, \text{ грн,} \quad (3.13)$$

де $t_{отл}$ - трудомісткість налагодження програми на ЕОМ, год (409 год);

$С_{Мч}$ - вартість машино-години ЕОМ, грн/год (28,7 грн/год).

Підставивши в формулу (3.13) відповідні значення, обчислюємо вартість необхідного для налагодження машинного часу:

$$З_{МВ} = 409 * 28{,}7 = 11\,738, \text{ грн,}$$

Звідси, за формулою (3.11), витрати на створення програмного продукту:

$$К_{ПО} = 156\,558 + 11\,738 = 168\,296, \text{ грн},$$

Очікуваний період створення програмного застосунку:

$$T = \frac{t}{B_k * F_p}, \text{ міс}, \quad (3.14)$$

де B_k - число виконавців (1);

F_p - місячний фонд робочого часу (при 40 годинному робочому тижні F_p =176 годин).

За формулою (3.14) очікуваний період створення програмного забезпечення:

$$T = \frac{833,2}{1*176} \approx 4,73 \text{ міс.}$$

Висновки: програмний продукт з продажу засобів індивідуального захисту був розроблений для отримання швидкого та зручного доступу споживачів до вітрини товарів, підвищенню рейтингу магазину та конкурентоспроможності компанії за рахунок впровадження сервісу високого рівня.

Розробка даного програмного забезпечення становить 168 296 грн без додаткових витрат.

Згідно отриманих даних при розрахунках, очікуваний час розробки становить 833,2 години, або 4,73 місяці, з урахуванням стандартного робочого графіку. Результат, що отримано, залежить від незначного коефіцієнта кваліфікації розробника, та у свою чергу включає час на виконання досліджень та розробку концепції для втілення поставленої задачі, надалі програмування за створеною концепцією, тестування програмного продукту та впровадження документації.

ВИСНОВКИ

У даній кваліфікаційній роботі був розроблений веб-орієнтований інформаційний додаток для продажу засобів індивідуального захисту. Система реалізує можливість швидкого доступу до каталогу товарів компанії та формування замовлення. Розроблений продукт призначено для можливості створювати замовлення, виконувати пошук товарів та відображати їх за окремими категоріями. Програмований веб-орієнтований додаток сприяє підвищенню продажів та прибутку підприємства та впровадження сервісу високого рівня за рахунок комфортного та зручного доступу користувача до пропозицій магазину.

Процес виконання кваліфікаційної роботи передбачав виконання наступних задач:

- визначено мету розробки та впровадження інформаційної системи, затверджено завдання на її розробку;
- сформовано основні вимоги щодо розробленого програмного продукту;
- створено макети та розроблено візуальну концепцію сторінок;
- спроектовано архітектуру додатку;
- обрано та налаштовано програмний інструментарій;
- визначено структуру та розроблено веб-орієнтовану систему;
- здійснено розрахунок трудомісткості та витрат на розробку програмного продукту.

З метою виконання поставлених задач були використанні технології та мови програмування, такі як фреймворк Angular, мова програмування TypeScript, мова розмітки HTML. Для стилізації були використані інструменти SCSS, Bootstrap.

В розробленій інформаційній системі, з метою досягнення поставленої мети, був реалізований наступний функціонал:

- швидка та зручна навігація по сторінках;
- можливість авторизації та реєстрації користувачів;
- інтуїтивно зрозумілий користувальницький інтерфейс;

– наявність швидкого доступу до кошику товарів;

– простота формування замовлення.

Варто зазначити, що в розробленій інформаційній системі присутня можливість тестової оплати замовлення за допомогою платіжної інфраструктури Stripe. Дана платформа для обробки онлайн-платежів була налаштована з метою отримання оновлень в режимі реального часу.

В «Економічному розділі» визначено трудомісткість розробки програмного забезпечення (833,2 людино-годин), підраховані витрати на створення програмного забезпечення (168 296 грн.) і гаданий період розробки (4,73 місяці).

Актуальність поставленої мети полягала у створенні веб-орієнтованого інформаційного додатку, що забезпечує надання послуг у сфері продажу засобів індивідуального захисту для фізичних та юридичних осіб з використанням сучасних веб-технологій, на базі фреймфорку Angular.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. E-commerce [Електронний ресурс] – Режим доступу: https://www.britannica.com/technology/e-commerce
2. What is ecommerce? Launch and grow an online sales channel [Електронний ресурс] – Режим доступу: https://sell.amazon.com/learn/what-is-ecommerce
3. Ecommerce Defined: Types, History, and Examples [Електронний ресурс] – Режим доступу: https://www.investopedia.com/terms/e/ecommerce.asp
4. Ecommerce Security: Importance, Issues & Protection Measures [Електронний ресурс] – Режим доступу: https://www.getastra.com/blog/knowledge-base/ecommerce-security/
5. What You Need to Know About Securing Your Ecommerce Site Against Cyber Threats [Електронний ресурс] – Режим доступу: https://www.bigcommerce.com/articles/ecommerce/ecommerce-website-security/
6. Ecommerce Tech Stack: What You Need to Know [Електронний ресурс] – Режим доступу: https://www.shipbob.com/blog/ecommerce-tech-stack/
7. What is Angular: Architecture, Features, and Advantages [Електронний ресурс] – Режим доступу: https://www.simplilearn.com/tutorials/angular-tutorial/what-is-angular
8. Introduction to Angular concepts [Електронний ресурс] – Режим доступу: https://angular.io/guide/architecture
9. What is Entity Framework? [Електронний ресурс] – Режим доступу: https://www.entityframeworktutorial.net/what-is-entityframework.aspx#:~:text=Entity%20Framework%20is%20an%20open,where%20this%20data%20is%20stored.
10. Information Security: The Ultimate Guide [Електронний ресурс] – Режим доступу: https://www.imperva.com/learn/data-security/information-security-infosec/#:~:text=Information%20security%20protects%20sensitive%20information,financial%20data%20or%20intellectual%20property.

11.7 Common Web Security Threats for an Enterprise [Електронний ресурс] – Режим доступу: https://www.fortinet.com/resources/cyberglossary/web-security-threats
12. Junior TypeScript Software Engineer. Зарплата станом на грудень, 2022 року [Електронний ресурс] – Режим доступу: https://jobs.dou.ua/salaries/?period=2022-12&position=Junior%20SE&technology=TypeScript&experience=0-2
13. Оренда ноутбука для особистого користування [Електронний ресурс] – Режим доступу: https://shop.vortex.dp.ua/ua/cp65737-arenda-noutbukov.html
14. A tour of the C# language [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/
15. What is a Bootstrap and how does it work? [Електронний ресурс] – Режим доступу: https://www.techtarget.com/whatis/definition/bootstrap
16. Visual Studio Code [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Visual_Studio_Code#:~:text=Visual%20Studio%20Code%2C%20also%20commonly,code%20refactoring%2C%20and%20embedded%20Git.
17. SQLite [Електронний ресурс] – Режим доступу: https://marketplace.visualstudio.com/items?itemName=alexcvzz.vscode-sqlite
18. What Exactly is Node.js? Explained for Beginners [Електронний ресурс] – Режим доступу: https://www.freecodecamp.org/news/what-is-node-js/
19. What is NPM? A Beginner's Guide [Електронний ресурс] – Режим доступу: https://careerfoundry.com/en/blog/web-development/what-is-npm/#what-is-npm
20. What is Postman? [Електронний ресурс] – Режим доступу: https://www.postman.com/product/what-is-postman/
21. What Is Stripe and How Does it Work? [Електронний ресурс] – Режим доступу: https://www.freshbooks.com/hub/payments/what-is-stripe

ДОДАТОК А

КОД ПРОГАМИ

app.component.ts

```
import { Component, OnInit } from '@angular/core';
import { BasketService } from './basket/basket.service';
import { AccountService } from './account/account.service';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent implements OnInit {
  title = 'Pronet';

  // injection of HTTP client
  constructor(private basketService: BasketService, private accountService:
AccountService) {}

  ngOnInit(): void {
    this.loadBasket();
    this.loadCurrentUser();
  }

  loadBasket() {
    const basketId = localStorage.getItem('basket_id');
    if (basketId) this.basketService.getBasket(basketId);
  }

  loadCurrentUser() {
    const token = localStorage.getItem('token');
    this.accountService.loadCurrentUser(token).subscribe();
  }
}
```

app.module.ts

```
import { NgModule } from ‘@angular/core’;
import { BrowserModule } from ‘@angular/platform-browser’;

import { AppRoutingModule } from ‘./app-routing.module’;
```

```
import { AppComponent } from ‘./app.component’;
import { BrowserAnimationsModule } from ‘@angular/platform-browser/animations’;
import { HTTP_INTERCEPTORS, HttpClientModule } from ‘@angular/common/http’;
import { CoreModule } from ‘./core/core.module’;
import { HomeModule } from ‘./home/home.module’;
import { ErrorInterceptor } from ‘./core/interceptors/error.interceptor’;
import { LoadingInterceptor } from ‘./core/interceptors/loading.interceptor’;
import { JwtInterceptor } from ‘./core/interceptors/jwt.interceptor’;
import { OrderDetailedComponent } from ‘./order-detailed/order-detailed.component’;

@NgModule({
  // when we create a new component
  // it should be registered and declared inside NgModule component
  declarations: [
    AppComponent,
    OrderDetailedComponent
  ],
  // any other modules should be imported
  imports: [
    BrowserModule,
    AppRoutingModule,
    BrowserAnimationsModule,
    HttpClientModule,
    CoreModule,
    HomeModule
  ],
  // provider provides services
  providers: [
    {provide: HTTP_INTERCEPTORS, useClass: ErrorInterceptor, multi: true},
    {provide: HTTP_INTERCEPTORS, useClass: LoadingInterceptor, multi: true},
    {provide: HTTP_INTERCEPTORS, useClass: JwtInterceptor, multi: true}
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

app.component.html

```
<ngx-spinner type=»timer»>
    <h3>Завантаження…</h3>
</ngx-spinner>

<app-nav-bar></app-nav-bar>
<app-section-header></app-section-header>

<router-outlet></router-outlet>
```

app-routing.module.ts

```
import { NgModule } from ‘@angular/core’;
import { RouterModule, Routes } from ‘@angular/router’;
import { HomeComponent } from ‘./home/home.component’;
```

```
import { TestErrorComponent } from ‘./core/test-error/test-error.component’;
import { NotFoundComponent } from ‘./core/not-found/not-found.component’;
import { ServerErrorComponent } from ‘./core/server-error/server-error.component’;
import { AuthGuard } from ‘./core/guards/auth.guard’;

const routes: Routes = [
  {path: ‘’, component: HomeComponent, data: {breadcrumb: ‘Головна’}},
  {path: ‘test-error’, component: TestErrorComponent},
  {path: ‘not-found’, component: NotFoundComponent},
  {path: ‘server-error’, component: ServerErrorComponent},

  // this route going to be lazily loaded when we click on the shop path
  {path: ‘товари’, loadChildren: () => import(‘./shop/shop.module’).then(m =>
m.ShopModule)},
  {path: ‘кошик’, loadChildren: () => import(‘./basket/basket.module’).then(m =>
m.BasketModule)},
  {
    path: ‘оплата’,
    canActivate: [AuthGuard],
    loadChildren: () => import(‘./checkout/checkout.module’).then(m =>
m.CheckoutModule)
  },
  {
    path: ‘замовлення’,
    canActivate: [AuthGuard],
    loadChildren: () => import(‘./orders/orders.module’).then(m => m.OrdersModule)
  },
  {path: ‘account’, loadChildren: () => import(‘./account/account.module’).then(m =>
m.AccountModule)},
  {path: ‘**’, redirectTo: ‘’, pathMatch: ‘full’},
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

login.component.html

```
<div class=»d-flex justify-content-center mt-5»>
    <div class=»col-3»>
```

```
        <form [formGroup]=»loginForm» (ngSubmit)=»onSubmit()»>
            <div class=»text-center mb-4»>
                <h1 class=»mb-3»>Увійти</h1>
            </div>
            <app-text-input [formControl]=»loginForm.controls[‘email’]»
[label]=»’Електронна адреса’»></app-text-input>
            <app-text-input [formControl]=»loginForm.controls[‘password’]»
[label]=»’Пароль’» [type]=»’password’»></app-text-input>
            <div class=»d-grid»>
                <button [disabled]=»loginForm.invalid» class=»btn btn-lg btn-primary
mt-3» type=»submit»>Вхід</button>
            </div>
        </form>
    </div>
</div>
```

account-routing.module.ts

```
import { NgModule } from '@angular/core';
import { LoginComponent } from './login/login.component';
import { RegisterComponent } from './register/register.component';
import { RouterModule, Routes } from '@angular/router';

const routes: Routes = [
  {path: 'login', component: LoginComponent},
  {path: 'register', component: RegisterComponent},
]

@NgModule({
  declarations: [],
  imports: [
    RouterModule.forChild(routes)
  ],
  exports: [RouterModule]
})
export class AccountRoutingModule { }
```

account.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { LoginComponent } from './login/login.component';
import { RegisterComponent } from './register/register.component';
```

```
import { AccountRoutingModule } from './account-routing.module';
import { SharedModule } from '../shared/shared.module';



@NgModule({
  declarations: [
    LoginComponent,
    RegisterComponent
  ],
  imports: [
    CommonModule,
    AccountRoutingModule,
    SharedModule
  ]
})
export class AccountModule { }
```

basket.component.html

```
<div class="container mt-5">
    <div *ngIf="(basketService.basketSource$ | async) === null">
        <p>Немає товарів у кошику.</p>
    </div>

    <ng-container *ngIf="(basketService.basketSource$ | async) as basket">
        <div class="container">
            <div class="row">
                <app-basket-summary
                    (addItem)="incrementQuantity($event)"
                    (removeItem)="removeItem($event)"
                ></app-basket-summary>
            </div>

            <div class="row">
                <div class="col-6 offset-6">
                    <app-order-totals></app-order-totals>
                    <div class="d-grid">
                        <a routerLink="/оплата" class="btn btn-outline-primary py-
2 mb-5">
                            Перейти до оплати
                        </a>
```

```
                        </div>
                    </div>
                </div>
            </div>
        </ng-container>
</div>
```

basket.component.ts

```
import { Component } from '@angular/core';
import { BasketService } from './basket.service';
import { BasketItem } from '../shared/models/basket';

@Component({
  selector: 'app-basket',
  templateUrl: './basket.component.html',
  styleUrls: ['./basket.component.scss']
})
export class BasketComponent {

  constructor(public basketService: BasketService) {}

  incrementQuantity(item: BasketItem) {
    this.basketService.addItemToBasket(item);
  }

  removeItem(event: {id: number, quantity: number}) {
    this.basketService.removeItemFromBasket(event.id, event.quantity);
  }
}
```

product-item.component.html

```
<div class="card h-100 shadow-sm" *ngIf="product">

    <div class="image position-relative" style="cursor: pointer;">
        <img src="{{product.pictureUrl}}" alt="{{product.name}}" class="img-fluid
bg-info">
        <div class="d-flex align-items-center justify-content-center hover-
overlay">
            <button (click)="addItemToBasket()" class="btn btn-primary fa fa-
shopping-cart me-2"></button>
```

```
                <button    routerLink="/товари/{{product.id}}"    class="btn    btn-
        primary">Деталі</button>
                </div>
            </div>
            <div class="card-body d-flex flex-column">
             <a href="" class="text-decoration-none">
                <h6 class="text-uppercase">
                    {{product.name}}
                </h6>
             </a>
             <span class="mb-2">{{product.price | currency:'грн '}}</span>
            </div>
</div>
```

shop.component.html

```
        <div class="container">
            <div class="row" *ngIf="types.length > 0 && brands.length > 0">
                <section class="col-3">
                    <h5 class="text-warning ms-3">Сортувати</h5>
                    <select class="form-select mb-4" (change)="onSortSelected($event)">
                        <option *ngFor="let sort of sortOptions"
                            [selected]="shopParams.sort === sort.value"
                            [value]="sort.value">
                            {{sort.name}}
                        </option>
                    </select>

                    <h5 class="text-warning ms-3">Бренди</h5>
                    <ul class="list-group my-3">
                        <!-- [class.active] sets the class with the active attributes if
                        the condition matches -->
                        <li class="list-group-item"
                            *ngFor="let brand of brands"
                            [class.active]="brand.id === shopParams.brandId"
                            [value]="brand.id"
                            (click)="onBrandSelected(brand.id)"
                        >{{brand.name}}</li>
                    </ul>
                    <h5 class="text-warning ms-3">Типи</h5>
                    <ul class="list-group my-3">
                        <li class="list-group-item"
```

```
                        *ngFor="let type of types"
                        [class.active]="type.id === shopParams.typeId"
                        [value]="type.id"
                        (click)="onTypeSelected(type.id)"
                    >{{type.name}}</li>
                </ul>
            </section>
            <section class="col-9">
                <div class="d-flex justify-content-between align-items-center pb-2">
                    <app-paging-header
                        [totalCount]="totalCount"
                        [pageNumber]="shopParams.pageNumber"
                        [pageSize]="shopParams.pageSize"
                    ></app-paging-header>
                    <div class="d-flex mt-2">
                        <input         (keyup.enter)="onSearch()"         type="text"
placeholder="Пошук" class="form-control me-2" #search>
                        <button  (click)="onSearch()"  class="btn  btn-outline-primary
mx-2">Пошук</button>
                        <button (click)="onReset()" class="btn btn-outline-success mx-
2">Видалити</button>
                    </div>
                </div>
                <div class="row row-cols-3 g-3 mb-4">
                    <div class="col" *ngFor="let product of products">
                        <!-- [] - receiving something -->
                        <app-product-item [product]="product"></app-product-item>
                    </div>
                </div>

                <div class="d-flex justify-content-center" *ngIf="totalCount > 0">
                    <app-pager
                        [totalCount]="totalCount"
                        [pageSize]="shopParams.pageSize"
                        [pageNumber]="shopParams.pageNumber"
                        (pageChanged)="onPageChanged($event)"
                    ></app-pager>
                </div>
            </section>
        </div>
</div>
```

shop.component.ts

```
import { Component, ElementRef, OnInit, ViewChild } from '@angular/core';
import { Product } from '../shared/models/product';
import { ShopService } from './shop.service';
import { Brand } from '../shared/models/brand';
import { Type } from '../shared/models/type';
import { ShopParams } from '../shared/models/shopParams';

@Component({
  selector: 'app-shop',
  templateUrl: './shop.component.html',
  styleUrls: ['./shop.component.scss']
})
export class ShopComponent implements OnInit {
  @ViewChild('search') searchTerm?: ElementRef;
  products: Product[] = [];
  brands: Brand[] = [];
  types: Type[] = [];
  shopParams: ShopParams;
  sortOptions = [
    {name: 'За алфавітом', value: 'name'},
    {name: 'Ціна: за зростанням', value: 'priceAsc'},
    {name: 'Ціна: за спаданням', value: 'priceDesc'},
  ];
  totalCount = 0;

  constructor(private shopService: ShopService) {
    this.shopParams = shopService.getShopParams();
  }

  ngOnInit(): void {
    this.getProducts();
    this.getBrands();
    this.getTypes();
  }

  getProducts() {
    // we should subscribe to an Observable
    this.shopService.getProducts().subscribe({
      next: response => {
```

```
      this.products = response.data;
      this.totalCount = response.count;
    },
    error: error => console.log(error)
  })
}

getBrands() {
  this.shopService.getBrands().subscribe({
    // ... - takes the array of brands that are inside of response
    // and speads them
    next: response => this.brands = [{id: 0, name: 'Bci'}, ...response],
    error: error => console.log(error)
  })
}

getTypes() {
  this.shopService.getTypes().subscribe({
    next: response => this.types = [{id: 0, name: 'Bci'}, ...response],
    error: error => console.log(error)
  })
}

onBrandSelected(brandId: number) {
  const params = this.shopService.getShopParams();
  params.brandId = brandId;
  params.pageNumber = 1;
  this.shopService.setShopParams(params);
  this.shopParams = params;
  this.getProducts();
}

onTypeSelected(typeId: number) {
  const params = this.shopService.getShopParams();
  params.typeId = typeId;
  params.pageNumber = 1;
  this.shopService.setShopParams(params);
  this.shopParams = params;
  this.getProducts();
}
```

```
  onSortSelected(event: any) {
    const params = this.shopService.getShopParams();
    params.sort = event.target.value;
    this.shopService.setShopParams(params);
    this.shopParams = params;
    this.getProducts();
  }

  onPageChanged(event: any) {
    const params = this.shopService.getShopParams();
    if (params.pageNumber !== event) {
      params.pageNumber = event;
      this.shopService.setShopParams(params);
      this.shopParams = params;
      this.getProducts();
    }
  }

  onSearch() {
    const params = this.shopService.getShopParams();
    params.search = this.searchTerm?.nativeElement.value;
    params.pageNumber = 1;
    this.shopService.setShopParams(params);
    this.shopParams = params;
    this.getProducts();
  }

  onReset() {
    if (this.searchTerm) this.searchTerm.nativeElement.value = '';
    this.shopParams = new ShopParams();
    this.shopService.setShopParams(this.shopParams);
    this.getProducts();
  }
}
```

shop.service.ts

```
import { HttpClient, HttpParams } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Pagination } from '../shared/models/pagination';
import { Product } from '../shared/models/product';
import { Brand } from '../shared/models/brand';
```

```
import { Type } from '../shared/models/type';
import { ShopParams } from '../shared/models/shopParams';
import { Observable, map, of } from 'rxjs';
import { environment } from 'src/environments/environment';

// services will be used to centralize our HTTP requests
@Injectable({
  providedIn: 'root'
})
export class ShopService {
  baseUrl = environment.apiUrl;
  products: Product[] = [];
  brands: Brand[] = [];
  types: Type[] = [];
  pagination?: Pagination<Product[]>;
  shopParams = new ShopParams();
  productCache = new Map<string, Pagination<Product[]>>();

  constructor(private http: HttpClient) { }

  getProducts(useCache = true): Observable<Pagination<Product[]>> {

    if (!useCache) this.productCache = new Map();

    if (this.productCache.size > 0 && useCache) {
      if (this.productCache.has(Object.values(this.shopParams).join('-'))) {
        this.pagination                                                    =
this.productCache.get(Object.values(this.shopParams).join('-'));
        if (this.pagination) return of(this.pagination);
      }
    }


    let params = new HttpParams();

    if(this.shopParams.brandId    >    0)    params    =    params.append('brandId',
this.shopParams.brandId);
    if(this.shopParams.typeId)       params       =       params.append('typeId',
this.shopParams.typeId);
    params = params.append('sort', this.shopParams.sort);
    params = params.append('PageIndex', this.shopParams.pageNumber);
```

```
    params = params.append('pageSize', this.shopParams.pageSize);
    if     (this.shopParams.search)     params     =     params.append('search',
this.shopParams.search);

    // sending out query string parameters
    return    this.http.get<Pagination<Product[]>>(this.baseUrl    +    'products',
{params}).pipe(
      map(response => {
        this.productCache.set(Object.values(this.shopParams).join('-'),
response);
        this.pagination = response;
        return response;
      })
    )
  }

  setShopParams(params: ShopParams) {
    this.shopParams = params;
  }

  getShopParams() {
    return this.shopParams;
  }

  getProduct(id: number) {
    const product = [...this.productCache.values()]
      .reduce((acc, paginatedResult) => {
        return {...acc, ...paginatedResult.data.find(x => x.id === id)}
      }, {} as Product)

    if (Object.keys(product).length !== 0) return of(product);

    return this.http.get<Product>(this.baseUrl + 'products/' + id);
  }

  getBrands() {
    if (this.brands.length > 0) return of(this.brands);

    return this.http.get<Brand[]>(this.baseUrl + 'products/brands').pipe(
      map(brands => this.brands = brands)
    );
```

```
  }

  getTypes() {
    if(this.types.length > 0) return of(this.types);

    return this.http.get<Type[]>(this.baseUrl + 'products/types').pipe(
      map(types => this.types = types)
    );
  }
}
```

shop.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { ShopComponent } from './shop.component';
import { ProductItemComponent } from './product-item/product-item.component';
import { SharedModule } from '../shared/shared.module';
import { ProductDetailsComponent } from './product-details/product-details.component';
import { RouterModule } from '@angular/router';
import { ShopRoutingModule } from './shop-routing.module';



@NgModule({
  declarations: [
    ShopComponent,
    ProductItemComponent,
    ProductDetailsComponent
  ],
  imports: [
    CommonModule,
    SharedModule,
    ShopRoutingModule
  ]
})
export class ShopModule { }
```

checkout.component.html

```
<div class="container mt-5">
```

```
        <div class="row">
            <div class="col-8">
                <app-stepper #appStepper>
                    <cdk-step                                          [label]="'Адреса'"
[completed]="checkoutForm.get('addressForm')?.valid">
                        <app-checkout-address    [checkoutForm]="checkoutForm"></app-
checkout-address>
                    </cdk-step>
                    <cdk-step                                        [label]="'Доставка'"
[completed]="checkoutForm.get('deliveryForm')?.valid">
                        <app-checkout-delivery   [checkoutForm]="checkoutForm"></app-
checkout-delivery>
                    </cdk-step>
                    <cdk-step [label]="'Замовлення'">
                        <app-checkout-review         [appStepper]="appStepper"></app-
checkout-review>
                    </cdk-step>
                    <cdk-step                                          [label]="'Оплата'"
[completed]="checkoutForm.get('paymentForm')?.valid">
                        <app-checkout-payment    [checkoutForm]="checkoutForm"></app-
checkout-payment>
                    </cdk-step>
                </app-stepper>
            </div>
            <div class="col-4">
                <app-order-totals></app-order-totals>
            </div>
        </div>
</div>
```

checkout.component.ts

```
import { Component, OnInit } from '@angular/core';
import { FormBuilder, Validators } from '@angular/forms';
import { AccountService } from '../account/account.service';
import { BasketService } from '../basket/basket.service';

@Component({
  selector: 'app-checkout',
  templateUrl: './checkout.component.html',
  styleUrls: ['./checkout.component.scss']
})
```

```
export class CheckoutComponent implements OnInit{

  constructor(private fb: FormBuilder, private accountService: AccountService,
    private basketService: BasketService) {}

  ngOnInit(): void {
    this.getAddressFormValues();
    this.getDeliveryMethodValue();
  }

  checkoutForm = this.fb.group({
    addressForm: this.fb.group({
      firstName: ['', Validators.required],
      lastName: ['', Validators.required],
      street: ['', Validators.required],
      city: ['', Validators.required],
      state: ['', Validators.required],
      zipcode: ['', Validators.required]
    }),
    deliveryForm: this.fb.group({
      deliveryMethod: ['', Validators.required]
    }),
    paymentForm: this.fb.group({
      nameOnCard: ['', Validators.required]
    })
  })

  getAddressFormValues() {
    this.accountService.getUserAddress().subscribe({
      next: address => {
        address && this.checkoutForm.get('addressForm')?.patchValue(address)
      }
    })
  }

  getDeliveryMethodValue() {
    const basket = this.basketService.getCurrentBasketValue();
    if (basket && basket.deliveryMethodId) {
      this.checkoutForm.get('deliveryForm')?.get('deliveryMethod')
        ?.patchValue(basket.deliveryMethodId.toString());
    }
```

```
  }
}
```

checkout.service.ts

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { environment } from 'src/environments/environment';
import { DeliveryMethod } from '../shared/models/deliveryMethod';
import { map } from 'rxjs';
import { Order, OrderToCreate } from '../shared/models/order';

@Injectable({
  providedIn: 'root'
})
export class CheckoutService {
  baseUrl = environment.apiUrl;

  constructor(private http: HttpClient) { }

  createOrder(order: OrderToCreate) {
    return this.http.post<Order>(this.baseUrl + 'orders', order);
  }

  getDeliveryMethods() {
    return this.http.get<DeliveryMethod[]>(this.baseUrl + 'orders/deliveryMethods').pipe(
      map(dm => {
        return dm.sort((a, b) => b.price - a.price)
      })
    )
  }
}
```

home.component.html

```
<carousel [isAnimated]="true">
    <slide>
        <img src="assets/images/slide4.png" alt="slide" style="display: block; width: 100%;">
    </slide>
    <slide>
```

```
            <img  src="assets/images/slide6.png"  alt="slide"  style="display:  block;
        width: 100%;">
            </slide>
            <slide>
                <img  src="assets/images/slide7.jpg"  alt="slide"  style="display:  block;
        width: 100%;">
            </slide>
        </carousel>
        <section class="featured">
            <div class="d-flex justify-content-center pt-4 mt-10">
                <h1>Ласкаво просимо до магазину!</h1>
            </div>
</section>
```

order-detailed.component.html

```
        <div class="container mt-5">
            <div class="row" *ngIf="order">
                <div class="col-8">
                    <div>
                        <div class="table-responsive">
                            <table class="table">
                                <thead>
                                    <tr>
                                        <th scope="col" class="border-0 bg-light">
                                            <div class="p-2 text-uppercase">Товар</div>
                                        </th>
                                        <th scope="col" class="border-0 bg-light">
                                            <div class="py-2 text-uppercase">Ціна</div>
                                        </th>
                                        <th scope="col" class="border-0 bg-light">
                                            <div             class="py-2            text-
        uppercase">Кількість</div>
                                        </th>
                                        <th scope="col" class="border-0 bg-light">
                                            <div             class="py-2            text-
        uppercase">Загалом</div>
                                        </th>
                                    </tr>
                                </thead>
                                <tbody>
                                    <tr *ngFor="let item of order.orderItems">
```

```
                                <th scope="row">
                                    <div class="p-2">
                                        <img            src="{{item.pictureUrl}}"
alt="{{item.productName}}" class="img-fluid"
                                             style="max-height: 50px">
                                        <div  class="ml-3  d-inline-block  align-
middle">
                                            <h5 class="mb-0">
                                                {{item.productName}}
                                            </h5>
                                        </div>
                                    </div>
                                </th>
                                <td  class="align-middle"><strong>{{item.price  |
currency: 'грн '}}</strong></td>
                                <td class="align-middle">
                                    <span       class="font-weight-bold       px-
2">{{item.quantity}}</span>
                                </td>
                                <td  class="align-middle"><strong>{{item.price  *
item.quantity | currency: 'грн '}}</strong></td>
                            </tr>
                        </tbody>
                    </table>
                </div>
            </div>
        </div>
        <div class="col-4">
            <div class="bg-light px-4 py-3 text-uppercase font-weight-bold">Ваше
замовлення</div>
            <div class="p-4">
                <ul class="list-unstyled mb-1">
                    <li class="d-flex justify-content-between py-3 border-bottom">
                        <strong class="text-muted">Ціна замовлення</strong>
                        <strong>{{order.subtotal | currency: 'грн '}}</strong>
                    </li>
                    <li class="d-flex justify-content-between py-3 border-bottom">
                        <strong class="text-muted">Доставка</strong>
                        <strong>{{order.shippingPrice     |     currency:     'грн
'}}</strong>
                    </li>
```

```
                        <li class="d-flex justify-content-between py-3 border-bottom">
                            <strong class="text-muted">Загалом</strong>
                            <strong>{{order.total | currency: 'грн '}}</strong>
                        </li>
                    </ul>
                </div>
            </div>
        </div>
</div>
```

order-detailed.component.ts

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute, Router } from '@angular/router';
import { Order } from 'src/app/shared/models/order';
import { BreadcrumbService } from 'xng-breadcrumb';
import { OrdersService } from '../orders/orders.service';

@Component({
  selector: 'app-order-detailed',
  templateUrl: './order-detailed.component.html',
  styleUrls: ['./order-detailed.component.scss']
})
export class OrderDetailedComponent implements OnInit {
  order?: Order;
  constructor(private orderService: OrdersService, private route: ActivatedRoute,
    private bcService: BreadcrumbService) {
      this.bcService.set('@OrderDetailed', ' ');
  }

  ngOnInit(): void {
    const id = this.route.snapshot.paramMap.get('id');
    id && this.orderService.getOrderDetailed(+id).subscribe({
      next: order => {
        this.order = order;
        this.bcService.set('@OrderDetailed',    `Замовлення#    ${order.id}    -
${order.status}`);
      }
    })
  }
}
```

orders.component.html

```
<div class="container mt-5">
    <div class="row">
        <div class="col-12">
            <table class="table table-hover" style="cursor: pointer;">
                <thead class="thead-light">
                    <tr>
                        <th>Замовлення</th>
                        <th>Дата</th>
                        <th>Вартість</th>
                        <th>Статус</th>
                    </tr>
                </thead>
                <tbody>
                    <tr *ngFor="let order of orders"
routerLink="/замовлення/{{order.id}}">
                        <th># {{order.id}}</th>
                        <td>{{order.orderDate | date: 'medium'}}</td>
                        <td>{{order.total | currency: 'грн '}}</td>
                        <td>{{order.status}}</td>
                    </tr>
                </tbody>
            </table>
        </div>
    </div>
</div>
```

orders.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Order } from '../shared/models/order';
import { OrdersService } from './orders.service';

@Component({
  selector: 'app-orders',
  templateUrl: './orders.component.html',
  styleUrls: ['./orders.component.scss']
})
export class OrdersComponent implements OnInit {
  orders: Order[] = [];

  constructor(private orderService: OrdersService) { }
```

```
    ngOnInit(): void {
      this.getOrders();
    }

    getOrders() {
      this.orderService.getOrdersForUser().subscribe({
        next: orders => this.orders = orders
      })
    }
}
```

ДОДАТОК Б

ВІДГУК КЕРІВНИКА ЕКОНОМІЧНОГО РОЗДІЛУ

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	**Опис**
Пояснювальні документи	
Кваліфікаційна_робота_Берг.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word
Кваліфікаційна_робота_Берг.pdf	Пояснювальна записка до кваліфікаційної роботи. Формат PDF
Програма	
Program.rar	Архів. Містить коди програми.
Презентація	
Презентація_Берг.pptx	Презентація кваліфікаційної роботи