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 *Бондаренко Діани Олександрівни*
(ПІБ)

академічної групи *121-18-2*
(шифр)

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

освітньої програми *Інженерія програмного забезпечення*
(назва освітньої програми)

на тему: *Розробка веб-орієнтованого програмного додатку пошуку продуктових товарів та послуг з використанням фреймворків Node.js, React.js*

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

Дніпро
2022

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

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

_____ І.М. Удовик
(підпис) (прізвище, ініціали)

« » _____ 2022 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *121-18-2* *Бондаренко Д.О.*
(група) (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка веб-орієнтованого програмного додатку пошуку продуктових товарів та послуг з використанням фреймворків Node.js, React.js*

затверджена наказом ректора НТУ «ДП» від *18 травня 2022 р.* № *268-с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів проєктно-технологічної та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі. Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2022 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***27.05.2022 р.***

Завдання видав _____ *доц. Гуліна І.Г.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Бондаренко Д.О.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2022 р.*

Термін подання кваліфікаційної роботи до ЕК: *13.06.2022 р.*

РЕФЕРАТ

Пояснювальна записка: 84 с., 30 рис., 6 табл., 3 дод., 30 джерел.

Об'єкт розробки: веб-орієнтований програмний додаток пошуку продуктових товарів та послуг.

Мета кваліфікаційної роботи: розробка веб-орієнтованого програмного додатку пошуку продуктових товарів та послуг з використанням фреймворків Node.js, React.js.

У вступі здійснюється аналіз проблеми, уточняється мета кваліфікаційної роботи та галузь її застосування, розглядається актуальність теми та конкретизується постановка завдання.

У першому розділі роботи проводиться аналіз предметної галузі, визначається актуальність завдання та призначення розробки, розробляється постановка завдання, задані вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі описуються використані технології та мови програмування, наводиться опис використаної архітектури і шаблонів проєктування, алгоритмів і структур функціонування додатку, визначаються вхідні і вихідні дані, наводяться характеристики складу параметрів технічних засобів, описується виклик та завантаження застосунку, зображується інтерфейс користувача програми.

В економічному розділі визначається трудомісткість розробленої інформаційної підсистеми, виконується підрахунок вартості розробки програмного продукту та розраховується час на його створення.

Практичне значення роботи полягає у створенні програмного додатку, що забезпечує користувачам сайту доступ до каталогу товарів бази магазинів, пошуку бажаних товарів за ключовими словами, можливість додавання товарів до корзини та редагування її вмісту. Також він надає можливість вибору магазину серед списку доступних, враховуючи наявність та ціни товарів там, а також місцезнаходження цих магазинів з використанням карти, та створення замовлення у обраному магазині.

Актуальність сервісу обумовлюється великим попитом на онлайн послуги, що надають можливість зекономити час та кошти, оптимізують та спрощують буденні турботи.

Список ключових слів: ПРОГРАМНИЙ ДОДАТОК, ПРОГРАМА, САЙТ, СЕРВІС, ВЕБ-ДОДАТОК, FRONT-END, BACK-END, БРАУЗЕР.

ABSTRACT

Explanatory note: 84 p., 30 fig., 3 apps, 30 sources.

Object of development: a web-based application for food products and services searching using Node.js and React.js framework.

The purpose of the the qualification work: development of a web-based application for food products and services searching using Node.js and React.js framework.

The purpose of the qualification work: development of a software solution providing search features for grocery products and services by using Node.js and React.js technologies.

The introduction contains an analysis of the problem and clarifies the purpose of the qualification work, its business domain and relevance. Furthermore, the setup of the task is being concretize.

In the first section of the work the analysis of the subject area is carried out, the urgency of the task and the purpose of development are determined, and the task statement is developed. The requirements for software implementation, technologies and software are set.

The second section describes programming languages and technologies that were used to implement the solution. The system's architecture, including design patterns being used, as well as algorithms and functional structures are laid out. Input and output data is being determined. Characteristics of the technical tools are presented, together with call and load procedures for the application. The user interface is being displayed as well.

The economic section determines the complexity of the developed information subsystem, calculates the cost of software development and calculates the time for its creation.

The practical significance of the work is in creating a software application providing access to the catalog of products available in the set of shops for the website users. Moreover, allowing users to search for goods by keywords, adding goods to a cart and editing the cart plays a critical practical role. Besides the main features, the solution allows users to filter and sort search results by availability, location and price. As a final step, users can send an order for the selected goods to the appropriate shops.

The relevance of the service is based on the high demand for the online platforms that help users efficiently and effectively use their time and money, therefore, optimizing and facilitating their day-to-day tasks.

Keywords: SOFTWARE APPLICATION, PROGRAM, WEBSITE, WEB SERVICE, WEB APPLICATION, FRONT-END, BACK-END, INTERNET BROWSER.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

HTML - Hypertext Markup Language;
CSS - Cascading Style Sheets;
БД - база даних;
ПК - персональний комп’ютер;
API - Application Programming Interface;
DOM - Document Object Model;
IT - інформаційні технології;
FE - Front-end;
BE - Back-end;
JS - Java Script;
MVC - Model – View – Controller;
SPA - Single Page Application;
JSX - JavaScript XML;
ODM - Object Data Modelling;
JSON - JavaScript Object Notation;
ПЗ - Програмне забезпечення;
ЕОМ - Електронна обчислювальна машина.

ВСТУП

Тематика даної кваліфікаційної роботи присвячена розробці веб-орієнтованого програмного додатку пошуку продуктових товарів та послуг з використанням фреймворків Node.js, React.js.

На сьогодні великий відсоток торгівлі відбувається онлайн, так як будь-хто, маючи доступ до мережі Інтернет, має змогу здійснити покупку з будь-якого куточку світу.

Інтернет-магазини стають все більш популярними, конкуренція росте і стає важко розробляти щось нове для користувачів. Одними з найважливіших факторів успіху веб-додатку є його візуальна складова та швидкість завантаження сторінок. Користувачам також подобаються інтуїтивно зрозумілі та зручні інтерфейси і якісні медіа-матеріали. В ідеалі сайт має надавати потенційному покупцю максимально повну необхідну інформацію, щоб повноцінно відтворити відчуття перебування в звичайному магазині. Покупець має мати можливість передивитися фото товару, вагу, склад та іншу детальну інформацію.

Результатом виконання даної роботи є програмний додаток, що значно спрощує рутинний процес вибору та придбання продуктів та економить час користувачів.

Завдання даної кваліфікаційної роботи та об'єкт її діяльності безпосередньо пов'язані зі спеціальністю «Інженерія програмного забезпечення» та відповідає узагальненій тематиці кваліфікаційних робіт і переліку зазначених виробничих функцій, типових задач діяльності, умінню та компетенціям.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальні відомості з предметної галузі

Зараз існує безліч сайтів для магазинів, різних компаній, державних органів і т.д. Веб-додатки умовно можна поділити односторінкові та багатосторінкові сайти.

Односторінкові сайти найчастіше використовуються для простого відображення інформації, наприклад портфоліо, сайт-візитка, реклама, афіша, тощо. Багатосторінковими сайтами є банківські додатки, інтернет-магазини, державні сервіси – тобто ті, що містять багато інформації, яку необхідно структурувати і розміщувати на окремих сторінках. Також різниця полягає в тому, що найчастіше односторінкові сайти вантажать усю інформацію одразу і не мають інтеграції з окремою базою даних. А багатосторінкові сайти натомість довантажують потрібні дані відповідно до взаємодії користувача з інтерфейсом, тобто за допомогою відправки та обробки запитів взаємодіють з back-end’ом і відповідно з базою даних.

Зазвичай статичні сайти створюються за допомогою HTML з використанням стилів, фреймворків з шаблонами компонентів. Такі сайти мають свої переваги та недоліки.

Переваги систем на базі HTML:

- сайт буде працювати на будь-якому сервері хостингу, навіть на локальному;
- дані сайту завантажуються разом з усіма елементами і ми можемо бачити усе і одразу;
- для розробки додатку не потрібні спеціальні програми, писати код можна навіть у WordPad;

- легко змінити зовнішній вигляд будь-якої конкретної сторінки чи додати нову, не чіпаючи інші частини сайту;
- невелике число використовуваних програмних компонентів утруднює поломку такої системи.

Такий спосіб створення сайтів у наш час вважається застарілим та використовується рідше, бо цей підхід має певні недоліки:

- складно внести зміни в структуру і зовнішній вигляд сайту, бо нема розподілення на «компоненти» і якщо ми змінюємо стиль шапки сайту у одному місці, треба змінювати це на всіх сторінках, що містить додаток;
- неможливо використовувати такий функціонал, як коментарі та відгуки користувачів, голосування, форум, чат і т.д.;
- щоб змінити якісь дані контенту треба робити правки в коді, де в HTML тегах зберігаються абзаци.

Щоб компенсувати недоліки чистого HTML до сайту підключають різні мови програмування, що дають можливість зробити сайт динамічним та додають інтеграцію з базами даних, створення анімацій і т.д.

Динамічна сторінка, на відміну від статичної, являє собою комбінацію статичних даних та даних, що зберігаються на сервері, які поєднуються, і тільки після цього показуються користувачу. Динамічним називається будь-який сайт, на якому є хоча б одна динамічна сторінка.

Коли користувач відкриває сторінку, відповідна інформація витягається з бази даних, вставляється в шаблон, утворюючи нову сторінку сайту, і відображається належним чином у браузері. Таким чином, якщо потрібно оновити вміст сайту, слід просто змінити дані у базі даних чи відредагувати з адміністративної панелі, якщо така є.

З цієї властивості безпосередньо випливає одна з найвагоміших переваг – спрощення модифікації і оновлення сторінок на сайті. Інформація повинна бути свіжою, інакше відвідувачі швидко втратять інтерес до сайту.

Першим недоліком динамічних сайтів є необхідність використання додаткових програмних засобів для побудови динамічних сторінок. Але зараз

існує достатня кількість фреймворків та бібліотек, що значно полегшують процес розробки. Другим недоліком є складність великих структурних змін сайту. В даному випадку все зав'язано на програмному забезпеченні, що об'єднує шматочки дизайну і даних в один повноцінний сайт. Але у будь-якої програми є свої обмеження, і, якщо необхідно отримати щось, що програмно не передбачено, потрібно змінювати саму структуру проєкту.

У випадку виконання даної кваліфікаційної роботи першим кроком у розробці сайту є вибір способу зберігання контенту (інформаційного наповнення), що відображається на Web-сторінці. В даному випадку для цього використовується NoSQL база даних MongoDB. MongoDB – це крос-платформна, потужна, гнучка, документно-орієнтована база даних, що легко масштабується. База даних зберігає в собі колекції. Кожна колекція складається з документів, які в свою чергу складаються з полів. Поля є парами ключ-значення. Колекції також можуть бути індексовані, що покращує швидкість вибірки та сортування. Тож після вибору варіанту з баз даних відбувається проєктування бази даних, необхідної для цього проєкту.

Наступним кроком стає розробка додатку для взаємодії з базою даних і сторінками сайту. В даному випадку для цих цілей був обраний фреймворк Node.js.

Node.js – це середовище виконання JavaScript, побудоване на механізмі JavaScript V8 Chrome. Node.js використовує керовану подіями модель, що не блокує введення-виведення, що робить її легкою та ефективною. Пакетна екосистема Node.js, npm, є найбільшою екосистемою бібліотек з відкритим кодом у світі.

Node.js написаний не на JavaScript (він написаний на C ++), але він використовує мову JavaScript як інтерпретаційну мову для обробки запиту/відповіді на стороні сервера. Іншими словами, Node.js запускає автономні програми JavaScript. Перевага полягає в тому, що програмісти можуть використовувати свої поточні, хоч і на стороні клієнта, знання програмування і починати кодування з Node.js набагато легше.

Після цього створюється front-end частина за допомогою бібліотеки React JS.

React – це декларативна, ефективна та гнучка JavaScript-бібліотека для створення користувальницьких інтерфейсів. Вона дозволяє збирати складний UI з маленьких ізольованих шматочків коду, що називають «компонентами».

1.2. Призначення розробки та галузь застосування

Як об'єкт розробки веб-орієнтованого програмного додатку пошуку продуктових товарів та послуг розглядається веб-додаток, в якому користувач може знайти бажані продуктові товари та замовити їх з доставкою або опцією самовивозу. Сервіс має містити структурований каталог продуктів, функцію пошуку товару за ключовими словами, сортування за ціною, пошук наявності обраних товарів у магазинах, оформлення замовлення з зручнішою опцією доставки. Сервіс має надавати повну інформацію про товар, бути легким у користуванні, щоб користувач відчував себе як у справжньому магазині, але при цьому економив час.

Розроблений додаток призначений для:

- відображення контенту програмного додатку за різними категоріями або результатами пошуку;
- забезпечення відвідувачам сайту простоти і комфортності доступу до каталогу товарів та створення замовлення;
- зручного пошуку та замовлення продуктових товарів у найближчому магазині або магазині з найбільшим асортиментом / найвигіднішими цінами;
- підвищення продажу продуктів магазинів за рахунок швидкості роботи сервісу.

1.3. Підстава для розробки

Підставами для розробки «виконання кваліфікаційної роботи» є:

- освітня програма 121 «Інженерія програмного забезпечення»;

- навчальний план та графік навчального процесу;

- наказ ректора Національного технічного університету «Дніпровська політехніка» №268-с від 18.05.2022;

- завдання на кваліфікаційну роботу на тему «Розробка веб-орієнтованого програмного додатку пошуку продуктових товарів та послуг з використанням фреймворків Node.js, React.js».

1.4. Постановка завдання

Завданням цієї кваліфікаційної роботи є розробка веб-орієнтованого програмного додатку пошуку продуктових товарів та послуг з використанням фреймворків Node.js, React. Програмне забезпечення призначене для надання універсального інструменту для відображення контенту бази магазинів.

Програма повинна реалізувати наступні функції:

- формування web-сторінок з використанням контенту з серверної частини застосунку;
- оформлення замовлень з даного магазину.

Для досягнення поставленої мети необхідно:

- вивчити предметну галузь розв'язуваної задачі;
- створити алгоритм для реалізації поставленого завдання;
- створити базу даних і клієнтську та серверну програми, що працюють з нею.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Для досягнення поставлених цілей програмне забезпечення, що розробляється, має підтримувати можливість виконання наступних дій:

- зчитування вхідних даних з пристроїв, баз даних, за адресними посиланнями;

- можливість надання віддаленого доступу до застосунку через веб-браузер на комп'ютері користувача;
- зберігання даних системи в нереляційній базі даних.

Для виконання перерахованих вище функцій у застосунку повинні бути реалізовані:

- можливість користуватися програмою через веб-браузер;
- наявність простої конфігурації та документації до неї для швидкого введення застосунку в експлуатацію;

1.5.2. Вимоги до інформаційної безпеки

Для уникнення некоректної роботи програмного додатку необхідно реалізувати:

- семантичний та синтаксичний контроль вхідних даних;
- обробку виняткових ситуацій;
- виведення повідомлень про помилки для користувача;
- виведення детальних повідомлень про помилки для розробників;
- можливість повторного введення даних;
- можливість безперервної роботи протягом не менше 168 годин (7 діб);
- платформну незалежність.

1.5.3. Вимоги до складу та параметрів технічних засобів

Для коректного функціонування програмного додатку необхідно, щоб комп’ютер, на якому буде працювати програмний додаток, відповідав наступним вимогам:

- процесор класу Intel Core i3 з частотою не менше 3.6 ГГц та чотирма ядрами;
- не менше 16 GB оперативної пам’яті;
- не менше ніж 256 GB вільного місця на жорсткому диску;
- доступ до мережі Internet;
- клавіатура;

- маніпулятор «миша».

Наведені технічні характеристики є рекомендованими. При наявності технічних засобів не нижче зазначених, розроблений програмний додаток буде функціонувати відповідно до вимог щодо швидкості обробки даних і безпеки, надійності, висунутих замовником.

1.5.4. Вимоги до інформаційної та програмної сумісності

Для нормального функціонування програми необхідно, щоб програмне забезпечення персонального комп'ютера, на якому буде функціонувати веб-орієнтований програмний додаток, відповідало наступним вимогам:

- ✓ операційна система Windows (7+) / Linux / MacOs;
- ✓ веб-браузер Google Chrome / Opera / Safari/ Firefox / Microsoft Edge.

Застосунок має бути реалізовано на мові програмування JavaScript з використанням бібліотеки ReactJS, фреймворку Node.js та бази даних MongoDB.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1. Функціональне призначення програми

За завданням кваліфікаційної роботи було реалізовано розробку веб-орієнтованого програмного додатку пошуку продуктових товарів та послуг з використанням фреймворків Node.js, React.js.

Призначення розробленої системи:

- систематизувати дані мереж магазинів;
- підвищити обсяг продажу товарів мереж магазинів;
- забезпечити просте користування додатком для пересічного користувача;
- надати можливість користувачам легко обирати та знаходити товари за різними категоріями та ключовими словами;

Розроблена програма реалізує наступні функції:

- відображення контенту, що містить в собі інформацію з бази даних;
- зручний вибір магазинів за наявністю товарів, цінами та розташуванням;
- перегляд магазинів за допомогою функціоналу Google Maps API;
- оформлення замовлень з обраного магазину через програмний додаток.

Для досягнення поставленої задачі розроблений програмний додаток підтримує виконання таких операцій:

- пошуку товарів за ключовим словом;
- пошуку товарів за категоріями;
- зміни кількості бажаних товарів у «кошику»;
- перегляду наявності товарів та їх цін в різних точках мереж магазинів та відображення їх на карті;
- формування замовлення в обраному магазині;
- вибору способу оплати та доставки.

2.2. Опис застосованих математичних методів

Оскільки особливості предметної галузі розв'язуваної задачі не передбачають застосування математичних методів, при розробці системи відображення та управління контенту програмного додатку математичні методи не використовувалися.

2.3. Опис використаної архітектури та шаблонів проєктування

У React використовується так званий компонентний підхід, який і був використаний для front-end додатку. Компоненти потрібні для перевикористання коду, успадковування, компонування. Компонент – це своєрідна «цеглинка», з якої будується інтерфейс. Концепція компонентного підходу у FE відкриває чудові можливості для повторного використання вже написаного коду.

У React використовується компонентний підхід для створення додатків, згідно з яким додаток складається з повторно використовуваних компонентів, код кожного з яких розташовується в окремих файлах.

React відповідає за візуальне відображення даних (так званий View шар). А взаємодії з даними з BE в даному випадку допомагає Flux архітектура.

Flux архітектура має 4 складові (схема наведена на рис. 2.1.):

- Диспетчер (Dispatcher);
- Сховище (Store);
- Представлення (View (React компоненти));
- Дія (Action).

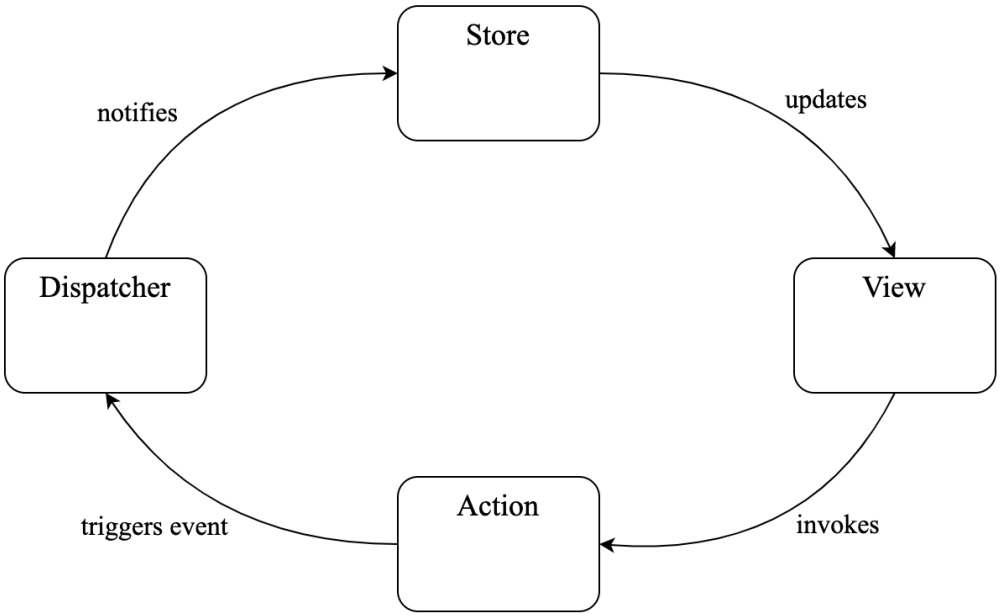

Рис. 2.1. Схема Flux архітектури

Для отримання даних з back-end’у, компонент має викликати згаданий вище “Action”, що в свою чергу викличе відповідну сагу, яка виконає всі сторонні ефекти (такі як запити на BE, асинхронні дії, взаємодії з local storage, session storage, виклики інших “Action” тощо). Saga – це бібліотека, яка створена для того, щоб спростити і покращити виконання сторонніх ефектів, спростити їх тестування і краще опрацьовувати можливі помилки.

Роботу саги можна уявити так, що сага – це як окремий потік в додатку, який відповідає за всі його сторонні ефекти. Її робота базується на концепції ES6 під назвою функції-генератори.

Типовий action у додатку виглядає так:

```
export const getProductsBySearchValue = asyncActionsCreator(
  constants.GET_PRODUCTS_BY_SEARCH_VALUE
);
```

Типова сага у додатку виглядає так:

```
export function* getProductsBySearchValueSaga({
  payload,
}: GetProductsSagaParams) {
```

```
  try {
    const response: ProductsResponse = yield call(
      ProductsService.getProductsBySearchValue,
      payload
    );
    const products = mapProductsData(response.data);
    yield put(actions.getProductsBySearchValue.success(products));
  } catch (error) {
    yield put(actions.getProductsBySearchValue.error(error));
  }
}
```

Типовий state має такий вигляд:

```
const initialState: ProductsPageState = {
  productsPage: {
    data: null,
    loading: false,
    error: null,
  },
};
```

Типовий reducer виглядає наступним чином:

```
      const reducer = createReducer(initialState, {
        [actions.getProductsBySearchValue.REQUEST]: (state) => {
          state.productsPage.loading = true;
        },
        [actions.getProductsBySearchValue.ERROR]: (state, action) => {
          state.productsPage.error = action.payload;
          state.productsPage.loading = false;
        },
        [actions.getProductsBySearchValue.SUCCESS]: (state, action) => {
          state.productsPage.data = action.payload;
```

```
          state.productsPage.loading = false;
        },
      });
```

Типовий сервіс має такий вигляд:

```
class ShopService {
  static getProducts = async (url) => {
    try {
      const response = await axios.get(url, {
        headers: {
          "Content-Type": "application/json; charset=UTF-8",
        },
      });
      return response;
    } catch (err) {
      throw err;
    }
  };
  static getCategories = async () => {
    try {
      const response = await axios.get('/api-v1/categories/', {
        headers: {
          "Content-Type": "application/json; charset=UTF-8",
        },
      });
      return response;
    } catch (err) {
      throw err;
    }
  }
  static getTypes = async () => {
```

```
    try {
      const response = await axios.get('/api-v1/types/', {
        headers: {
          "Content-Type": "application/json; charset=UTF-8",
        },
      });
      return response;
    } catch (err) {
      throw err;
    }
  }
}
```

Структура BE додатку базується на MVC архітектурі у вигляді трьох рівнів. Кожен з рівнів відповідає за свою дію на тому, чи іншому рівні. В додатку є три рівні (схему наведено на рис.2.2.):

- рівень роутингу (View);
- рівень контролера (Controller);
- рівень моделі (Model).

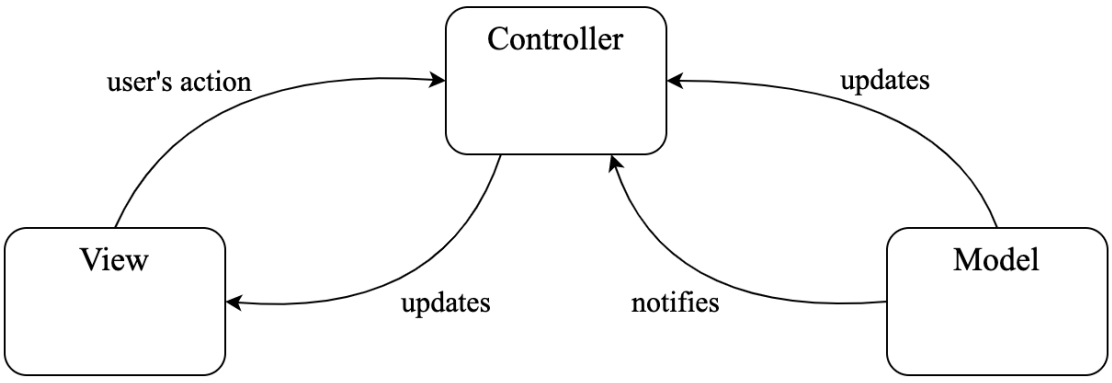

Рис. 2.2. Схема взаємодії компонентів веб-додатку в моделі MVC

Роутинг відповідає за розподілення дій на контролер. Контролер у свою чергу відповідає за виклик моделі з заданими параметрами виклику та відправку даних на клієнт. Після виклику модель формує запит до бази даних та за допомогою ORM здійснює його та отримує запитані дані. Після цього дані у зворотному напрямку передаються до контролера. Дана архітектура допомагає розділити відповідальність на кожному з рівнів та уникнути проблем з програмним застосунком.

При створенні масштабного додатку рекомендуються такі папки (файлова структура наведена на рис. 2.3.):

- /models – містить всі моделі (в даному випадку mongoose Schemas);
- /routes – містить всі доступні експрес-маршрути;
- /controllers – містить всі доступні функції-обробники для кожного маршруту;
- /utils – містить додаткові методи, що спрощують написання функціоналу.

В свою чергу база даних розробляється для того, щоб процес систематизації відбувався швидше відповідно покращуючи результативність програми. База даних є ключовим складником в системі управління програмним додатком, так як в ній будуть зберігатися всі дані, на яких базується робота програми та її складових.

Також дуже важливо, щоб у кожного поля, що буде вноситися в базу, був унікальний ідентифікатор, який не повторюватиметься. За даний функціонал в додатку відповідають MongoDB та mongoose, що автоматично генерують унікальні ідентифікатори.

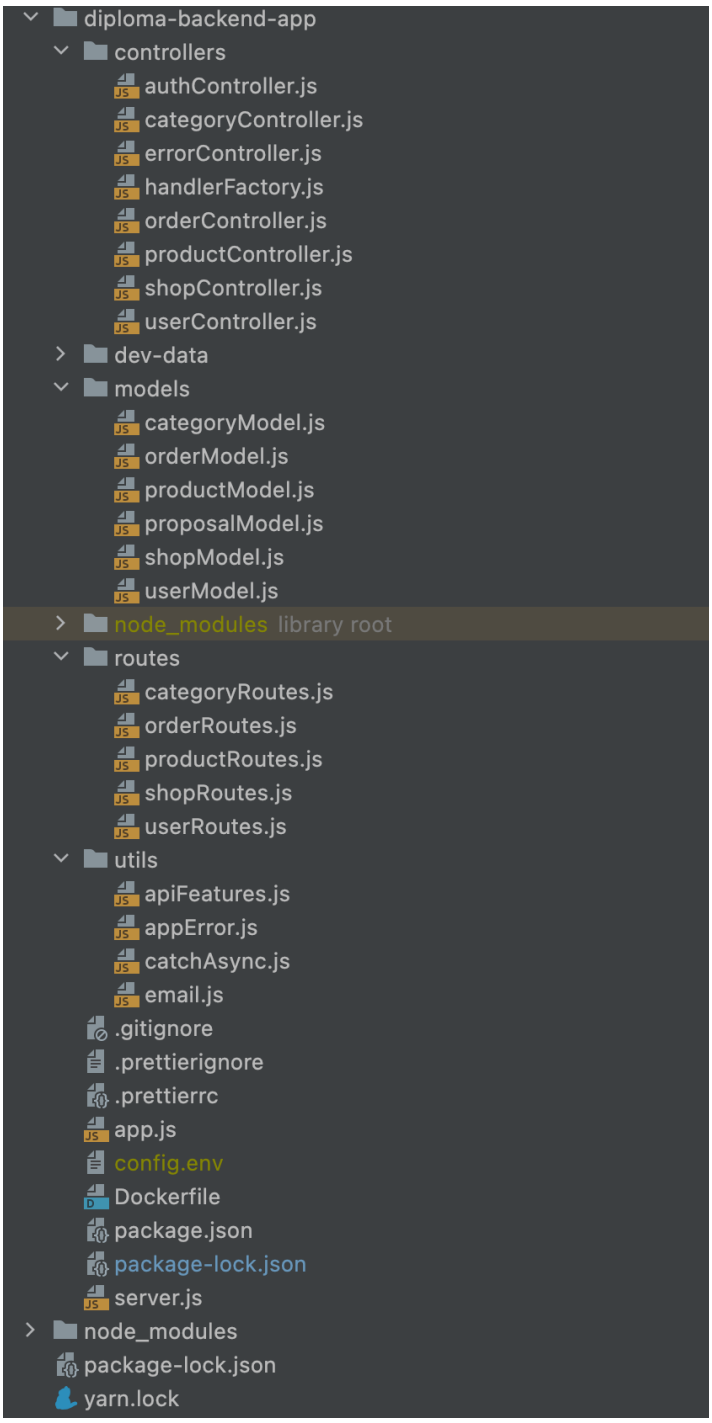

Рис. 2.3. Файлова структура BE додатку

Для даної концепції бази даних, можна навести такий перелік сутностей:

1. Користувачі – містить інформацію про користувачів системи: електронна пошта, роль (користувач / менеджер магазину / адміністратор), номер телефону, ім`я, прізвище;

2. Магазини – містить інформацію про магазини: назва магазину, адреса, координати, розклад роботи, ідентифікатор власника магазину;

3. Пропозиції магазинів – містить інформацію про товари в наявності для кожного магазину: ціна, кількість, ідентифікатор продукту, ідентифікатор магазину;

4. Товари – містить інформацію про товари: назва, ідентифікатор категорії, вага, пакування, виробник, зображення та пропозиції магазинів;

5. Категорії – містить інформацію про наявні категорії: назва категорії, ідентифікатор «батьківської» категорії;

6. Замовлення – містить інформацію про активні замовлення: номер телефону замовника, ідентифікатор магазину, час замовлення, метод оплати, метод доставки та список замовлених продуктів;

Зв`язки між розписаними вище сутностями будуть такими:

1. Між атрибутами Продукт та Категорія буде зв`язок 1:M, так як продукт може мати одну категорію, а категорія може мати багато продуктів

2. Між атрибутами Користувач і Замовлення буде зв`язок 1:M, так як користувач може мати безліч замовлень, а замовлення може мати тільки одного користувача

3. Між атрибутами Користувач і Магазин буде зв`язок 1:M, так як користувач може мати безліч магазинів, а магазин може мати тільки одного користувача

4. Між атрибутами Замовлення та Товар буде зв`язок M:M, так як замовлення може містити багато товарів, та товари можуть бути в багатьох замовленнях

5. Між атрибутами Пропозиція і Товар буде зв`язок 1:M, так як товар може містити безліч пропозицій, а пропозиція може мати тільки один товар

6. Між атрибутами Магазин і Пропозиція буде зв`язок 1:M, так як магазин може мати безліч пропозицій, а пропозиція може мати тільки один магазин

2.4. Опис використаних технологій та мов програмування

Front-end частина веб-додатку реалізована за допомогою TypeScript, React, Redux, redux-saga, Google Maps API.

TypeScript – це мова програмування, що позиціонує себе як засіб розробки веб-застосунків, що розширює можливості JavaScript. TypeScript є зворотно сумісним з JavaScript. Тому, після компіляції програму на TypeScript можна виконувати в будь-якому сучасному браузері або використовувати разом з Node.js.

Переваги над JavaScript:

- підтримання підключення модулів;
- можливість явного визначення типів;

Ці відмінності підвищують швидкість розробки, рефакторинг і повторне використання коду, та надають більше інформації в порівнянні з базовою бібліотекою для React/JavaScript додатків – prop-types.

Основний принцип TypeScript – будь-який код на JavaScript сумісний з TypeScript, що означає що в додатках на TypeScript можна використовувати стандартні бібліотеки JS. Також TypeScript дуже легкий у впровадженні його у вже існуючих додатках написаних на JS, так як дозволяє впроваджувати його поступово «файл за файлом» не порушуючи роботу додатку.

Тому можна зробити висновок, що TypeScript в основному відрізняється від JS додатковою типізацією.

JS/TS також використовують для створення мобільних додатків, в back-end розробці та в десктопних програмах. Але в основному їх використовують для веб-додатків: від landing page (односторінкові сайти) до складних інформаційних систем.

JS/TS мають такі беззаперечні переваги:

- підтримка усіма сучасними браузерами;
- легка інтеграція з верстkою сторінок (HTML + CSS) і серверною частиною;

- швидкість роботи та продуктивність;
- велика кількість бібліотек та фреймворків;
- легкість освоєння.

Завдяки великому ком'юніті у front-end галузі, розвитку набули окремі фреймворки. Фреймворки створені для економії часу та вартості проєктів, дають можливість концентруватися на логіці, а не витрачати час на створення нових архітектурних рішень.

На даний момент найпопулярніші JS фреймворки та бібліотеки (наведено на рис.2.4.):

- React;
- Angular JS;
- Vue.js;
- Node.js.

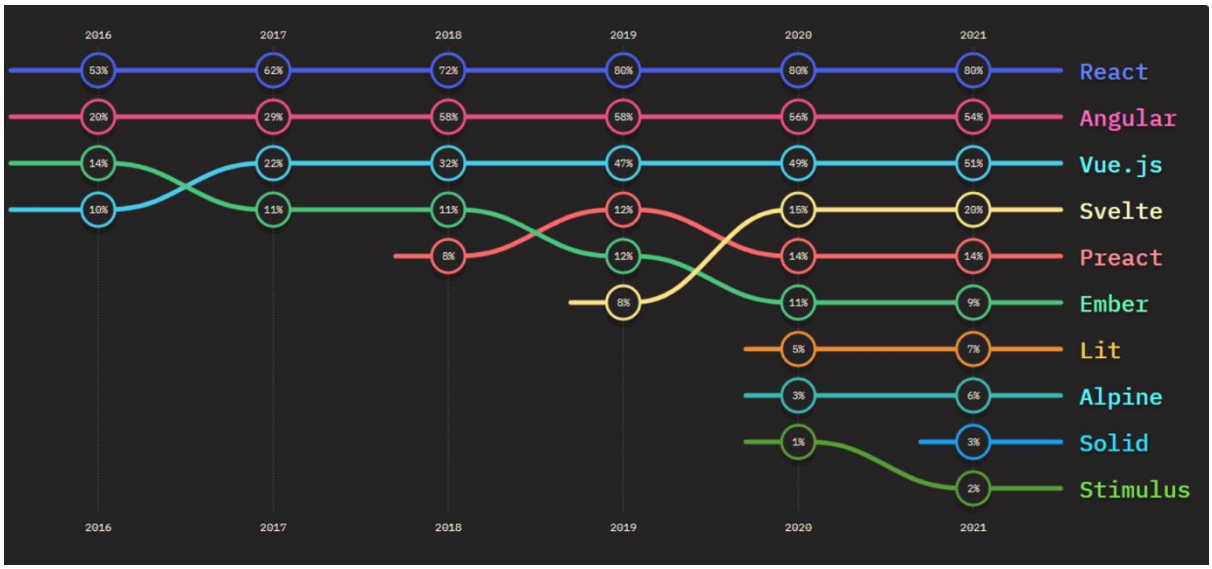

Рис. 2.4. Звіт з популярності FE фреймворків

Дивлячись на звіт The State of JS 2021: Front-end frameworks and libraries можемо побачити React, Angular та Vue.js серед лідерів.

React – відкрита JS бібліотека для створення інтерфейсів користувача створена розробниками Facebook, щоб вирішити питання підтримки коду. Бібліотеку вперше почали використовувати в 2011 році, а в 2013 році бібліотека стала доступною для використання для всіх бажаючих.

React дозволяє розробникам створювати веб-додатки, що можуть змінювати відображення без перезавантаження сторінки (SPA). SPA (Single Page Application) – це веб-додаток або сайт, що використовує єдиний HTML документ як обгортку для всіх веб сторінок і взаємодіє з користувачем через динамічно підвантажувані HTML, CSS, JS. Завдяки SPA додаток швидко реагує на дії користувача (використання фільтрів, заповнення форм, додавання товарів у кошик, тощо). Зазвичай React використовують з бібліотеками для управління станом додатку і роутингом – redux і react-router.

React додаток побудований на компонентному підході, коли кожен елемент додатку є компонентою. До особливостей React належить використання JSX синтаксису. Приклад JSX можна побачити нижче (рис. 2.5.):

```
const App = () => {
 const number = 10;
 return (
  <div>
   <p>Number: {number}</p>
  </div>
 );
};
```

Рис. 2.5. Приклад JSX

JSX дозволяє писати JS код вбудовуючи його одразу в HTML, що спрощує розробку.

Також до особливостей відноситься використання Virtual DOM. Virtual DOM – це копія DOM дерева. Замість того щоб взаємодіяти з DOM напряму, ми працюємо з його копією. Ми можемо вносити зміни до Virtual DOM, а після цього застосовувати зміни до реального DOM. При цьому відбувається порівняння DOM дерева з його віртуальною копією, визначається різниця і запускається заміна того, що було змінено.

Redux-saga – це бібліотека, що спрощує взаємодію з side ефектами додатків (асинхронні виклики, взаємодії з local storage та session storage, виклики інших action’s, тощо). Saga – це middleware для redux, що використовує концепцію ES6

під назвою генератори, для того щоб зробити асинхронні виклики легшими для розуміння, написання та тестування.

Google Maps API – безкоштовний картографічний сервіс від Google, що являє собою географічну карту та дозволяє легко інтегрувати її до свого додатку та конфігурувати її під свої потреби.

Back-end частина додатку реалізована за допомогою Node.js, Express, mongoose, MongoDB.

Node.js – open-source платформа для створення високопродуктивних мережевих додатків, написаних на JavaScript. Node.js представляє осередок виконання коду на JavaScript, що побудований на основі движка V8, що в свою чергу дозволяє транслювати виклики на JS у машинний код. Node.js попередньо призначений для створення серверних додатків на JS, хоча і існують проекти з написання десктопних програм і коду для мікроконтролерів, але все ж таки ми маємо на увазі Node.js, як платформу для створення веб-додатків.

Однією з особливостей цієї технології є підхід до роботи з вводом-виводом і відсутності багато-потоковості.

Npm (Node Package Manage) – це пакетний менеджер для Node.js, що є важливою частиною екосистеми.

Для написання простого сервера потрібно мати хоча б два зовнішніх пакетів – це фреймворк Express і модуль body-parser. Та в даному випадку ми також використовуємо mongoose ODM (Object Data Modelling) для взаємодії з MongoDB.

Express – це швидкий та мінімалістичний фреймворк для веб-додатків побудованих на Node.js. В розпорядженні Express є безліч допоміжних методів та проміжних обробників, що спрощує і пришвидшує написання надійних API і дозволяє зфокусуватися на логіці, а не на налаштовуванні сервера.

Mongoose являє собою спеціальну ODM бібліотеку для роботи з MongoDB, що дозволяє порівнювати об’єкти класів і документи з бази даних.

2.5. Опис структури системи та алгоритмів її функціонування

Під час проєктування був проведений аналіз структури вхідних та вихідних даних, призначення веб-орієнтованого програмного додатку, цільових користувачів та принципів роботи їх проєктних рішень.

Структура додатку – це опис руху користувача між сторінками інтерфейсу. Чим простіше і зрозуміліше розроблений цей рух – тим більше буде зацікавленість клієнта у даному додатку. Також додає зручності можливість з будь-якого місця повернутися на попередню, домашню сторінку додатку та на сторінку з кошиком.

Загальна структура програмного додатку з точки зору користувача (рис. 2.6.) має наступний вигляд:

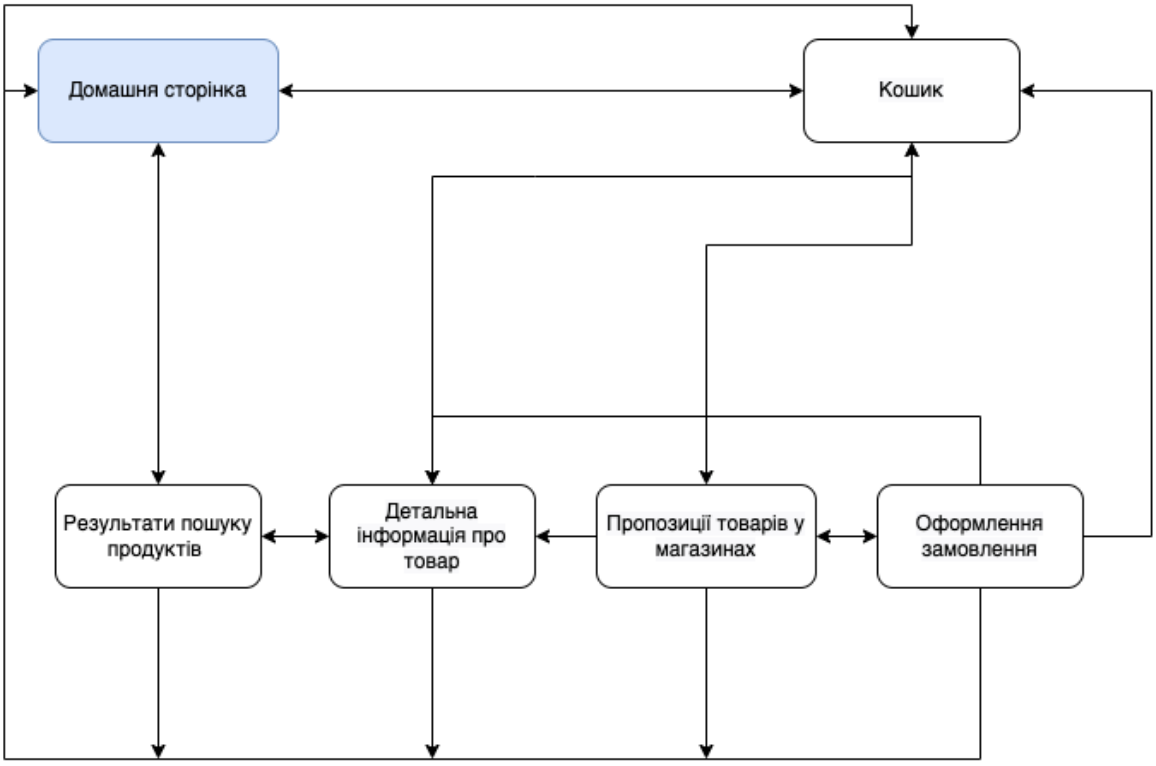

Рис. 2.6. Структура програмного додатку

Структура програмного додатку складається з таких частин:

- головна сторінка з компонентом для пошуку товарів, інформацією про сервіс та кроками для оформлення замовлення.

- каталог товарів з категоріями продуктів;
- окрема сторінка для кожного продукту з детальною інформацією про нього;
- сторінка з кошиком обраних товарів;
- сторінка з пропозиціями магазинів на запит певних товарів та картою магазинів;
- сторінка оформлення замовлення.

Для створення комфортних умов для здійснення покупок клієнтами та збільшення конверсії сервісу можна дотримуватися таких правил: привабливий дизайн, потрібний і зрозумілий функціонал та деякі маркетингові фішки.

Основна увага користувача має припадати на пошук та каталог товарів. На цій частині робиться акцент в інтерфейсі. На сторінці каталогу користувач може бачити список категорій, які можна обирати. Також зручності додають пошук за назвою або фірмою товару і сортування товарів.

Шапка (header) та підвал (footer) веб-додатка – це два елементи дизайну, що дублюються на всіх сторінках. Шапка сайту постійно знаходиться перед очима користувача, тому в ній містяться важливі елементи. Там знаходяться логотип додатку, компонент для пошуку за ключовими словами і кнопка для вибору товарів за категоріями. Також у верхньому меню можна знайти кнопку кошика, яка перенаправить користувача на окрему сторінку кошика з товарами. Коли користувач додає товари до кошика, цифра кількості товарів змінюється на іконці кошика.

Другий наскрізний елемент дизайну – футер – клієнти бачать в останню чергу, але є досить важливим. Переглянувши сайт, у потенційного покупця має виникнути відчуття, що в інтернет-магазині можна безпечно і легко придбати потрібну продукцію. Тому в футері розташовується інформація, що сприяє здійсненню покупки.

Також досить важливою є грамотно утворена файлова система програмного додатку.

Також для проєкту використовується система версій Git. Використовуючи Git, рекомендовано робити коміти для кожної логічної нової частину коду, щоб була можливість легко повертатися до попередніх змін.

Файлова система front-end додатку містить папку components з компонентами, що можуть з легкістю повторно використовуватися у додатку, папку containers з окремою папкою файлів для кожної сторінки додатку. А також папку global, що містить медіа-файли, файли з константами, допоміжними функціями тощо (рис.2.7.).

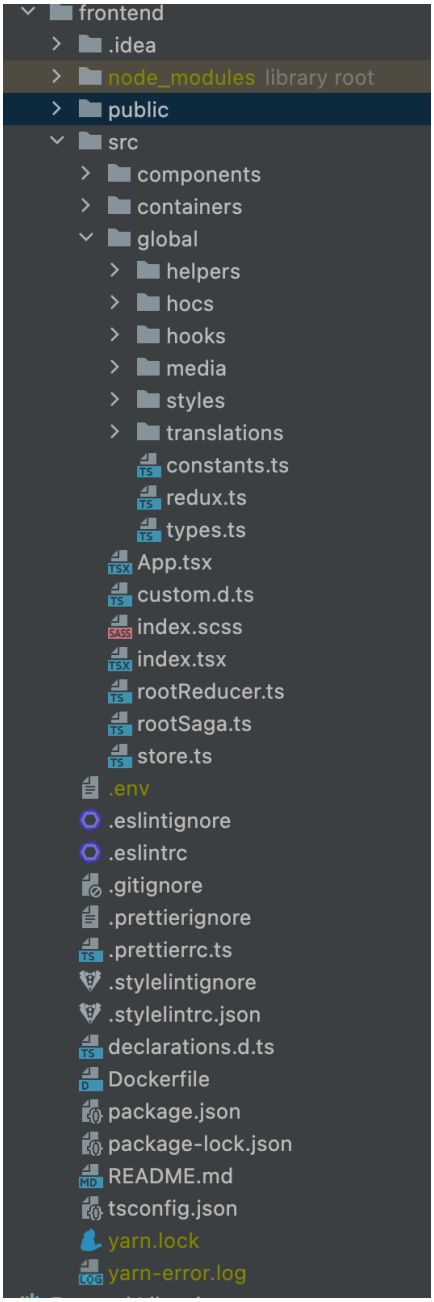

Рис. 2.7. Структура файлової системи FE додатку

2.6. Обґрунтування та організація вхідних та вихідних даних програми

Програмний додаток отримує вхідні дані шляхом надсилання запиту на BE додаток, що в свою чергу отримує дані з бази даних.

Вхідними даними для back-end частини додатку є сформовані запити за допомогою mongoose Schemas, що приходять з клієнтської частини додатку, коли користувач хоче отримати ту чи іншу інформацію про магазин. Вихідними даними є JSON файл з потрібною інформацією, що передається інтерфейсу для відображення.

Обов`язковими даними, які будуть зберігатися в базі є:

- список товарів
- список категорій товарів
- список користувачів системи
- список магазинів
- список пропозицій від магазинів
- список замовлень

На даному етапі розробки БД можна скласти такі схеми сутностей в яких прописані назви атрибутів та їх типи даних.

Таблиця 2.1.

Схема «Product»

Назва	Тип даних	Обмеження
name	String	min_length = 5, max_length = 80, required, unique
category_id	mongoose.Schema.ObjectId	required
weight	String	required
packing	String	required
manufacturer	String	required
image	String	-
proposals	Array	-

Таблиця 2.2.

Схема «Category»

Назва	Тип даних	Обмеження
name	String	required, unique, min_length = 3,
parent_id	mongoose.Schema.ObjectId	required

Таблиця 2.3.

Схема «Order»

Назва	Тип даних	Обмеження
phone	String	required
shop_id	mongoose.Schema.ObjectId	required
created_at	Date	-
payment_method	String	-
delivery_method	String	-
products	Array	-

Таблиця 2.4.

Схема «Proposal»

Назва	Тип даних	Обмеження
product_id	mongoose.Schema.ObjectId	required
shop_id	mongoose.Schema.ObjectId	required
price	Number	required
count	Number	-

Таблиця 2.5.

Схема «Shop»

Назва	Тип даних	Обмеження
name	String	required
address	String	required
coordinates	[Number]	-
schedule	String	required
owner_id	mongoose.Schema.ObjectId	required

Таблиця 2.6.

Схема «User»

Назва	Тип даних	Обмеження
email	String	required, unique
role	String	user / shop
phone	String	unique
name	String	required
surname	String	required
password	String	required
passwordConfirm	String	required
passwordChangedAt	Date	-
passwordResetToken	String	-
passwordResetExpires	Date	-

Даний back-end додаток обробляє достатню кількість запитів. Запити для продуктів зображено на рис. 2.8., 2.9.

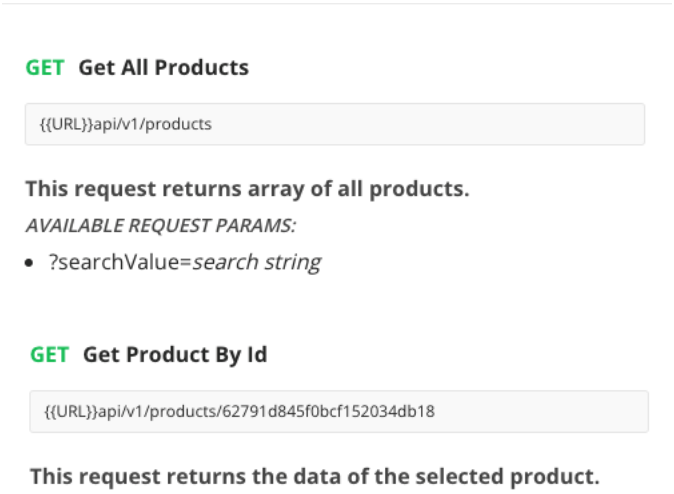

Рис. 2.8. Запити для отримання інформації про продукти

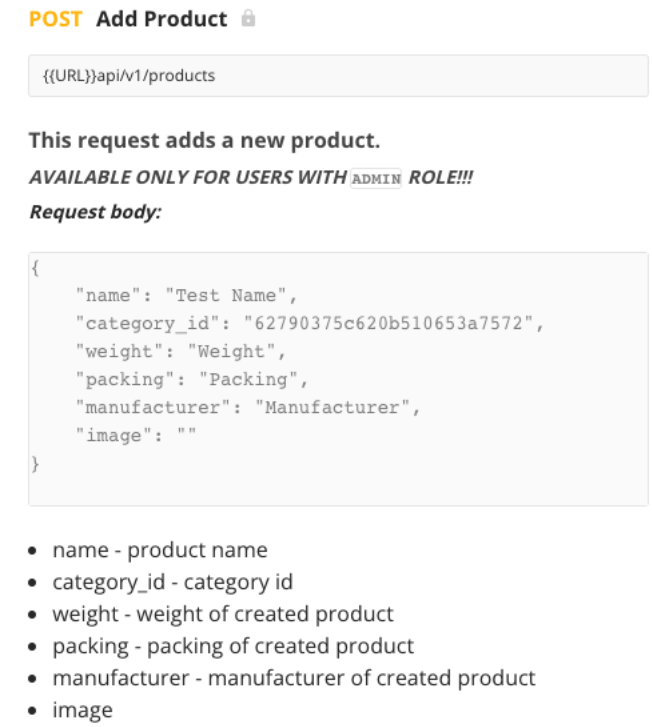

Рис. 2.9. Запит для внесення інформації про продукт

Запити для категорій товарів зображено на рис. 2.10.

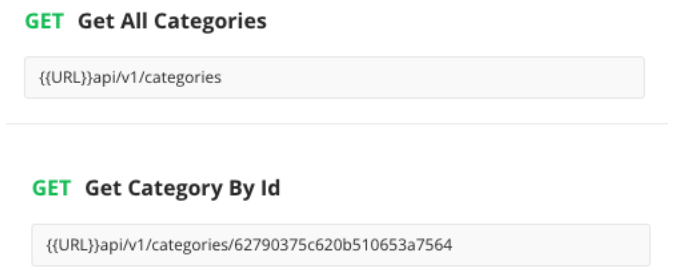

Рис. 2.10. Запити для отримання інформації про категорії товарів

Запити для магазинів зображено на рис. 2.11. - 2.13.

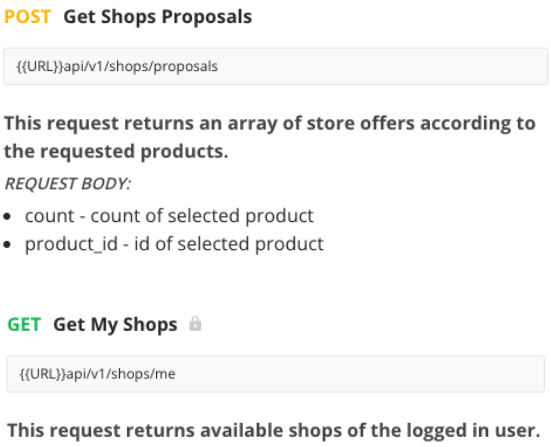

Рис. 2.11, 2.12. Запити для отримання інформації про магазини

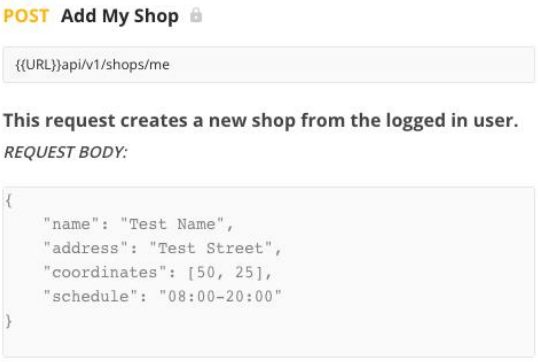

Рис. 2.13. Запит для додавання магазину

Запити для замовлень зображено на рис. 2.14.

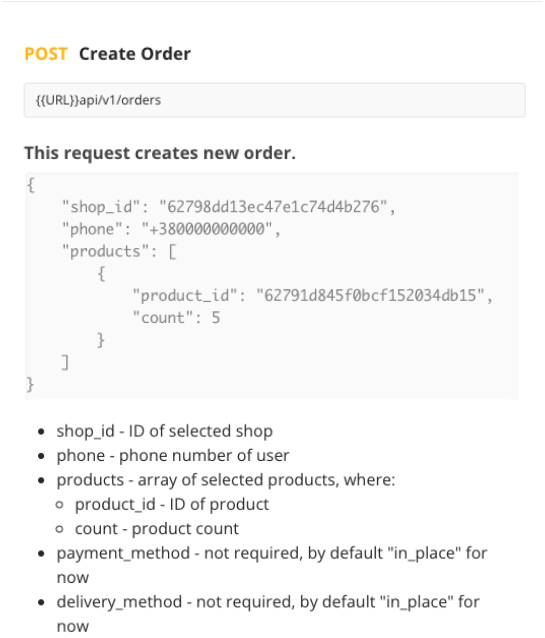

Рис. 2.14. Запит для створення замовлення

Запити для реєстрації та авторизації зображено на рис. 2.15.

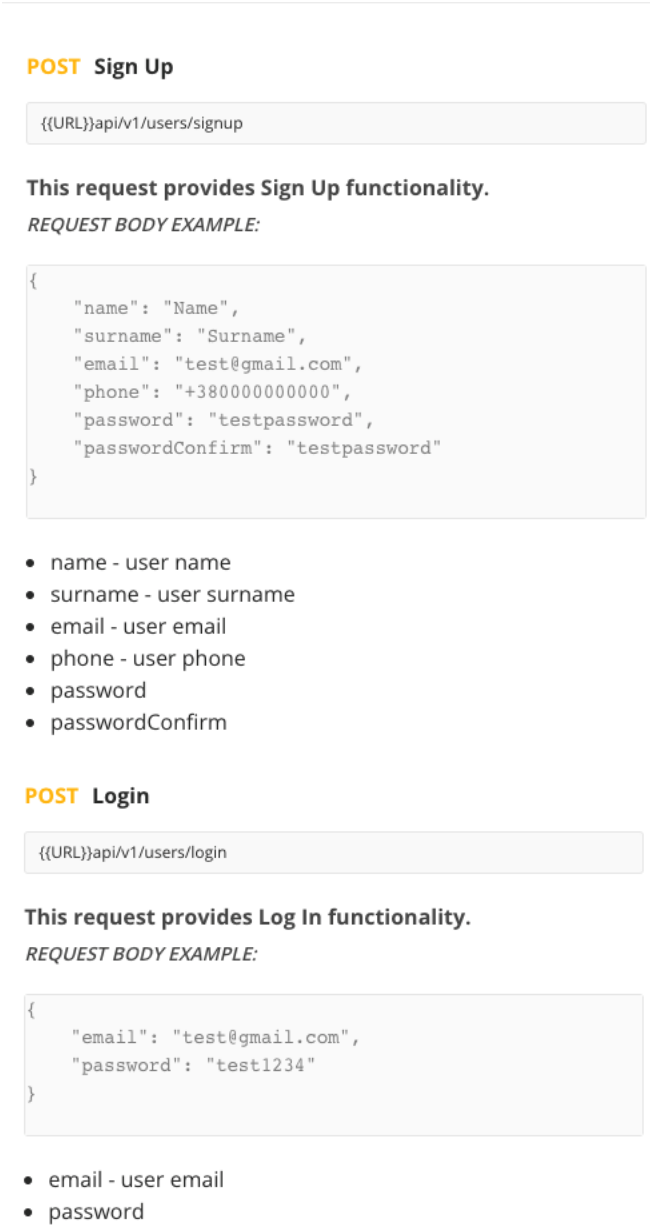

Рис. 2.15. Запити для реєстрації та авторизації

Запити для операцій з паролем зображено на рис. 2.16.

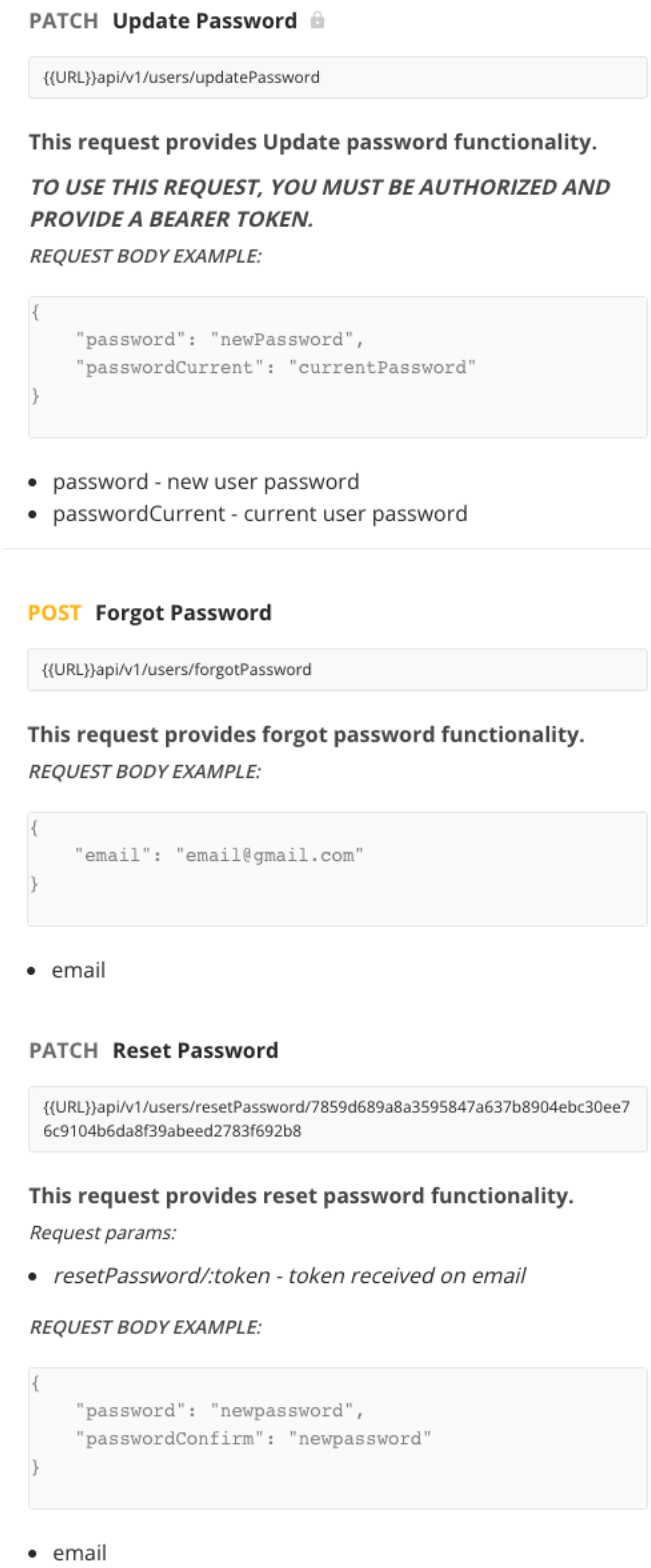

Рис. 2.16. Запити для операцій з паролем

Запити для операцій з паролем зображено на рис. 2.17.

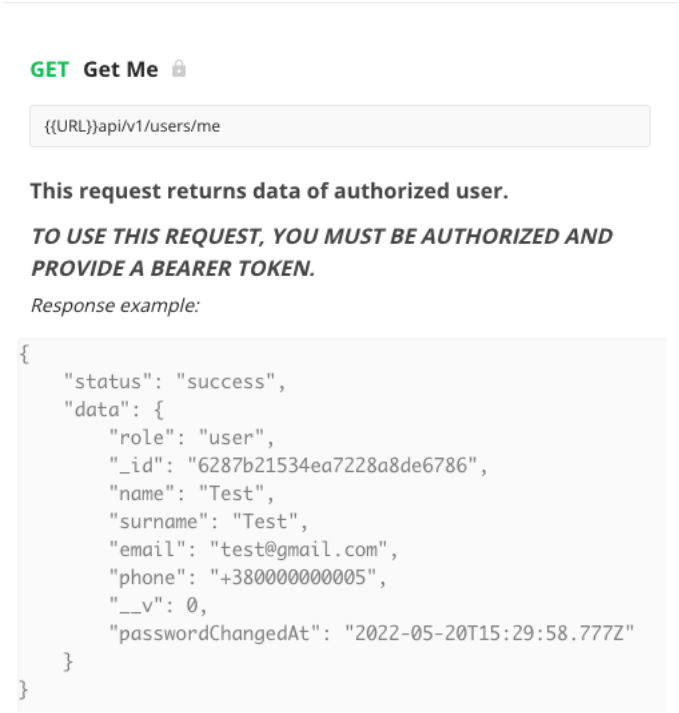

Рис. 2.17. Запити для отримання інформації про продукти

Найбільшими правами володіє адміністратор. На відміну від менеджера магазину, який може додавати тільки пропозиції для наявних товарів, адміністратор може додавати товари загалом.

Менеджери магазину можуть переглядати інформацію про свої магазини та наявний асортимент, додавати нові магазини і пропозиції товарів, переглядати наявні замовлення та керувати ними.

2.7. Опис роботи розробленої системи

2.7.1. Використані технічні засоби

Більшу частину подібних веб-орієнтованих додатків розміщують на сторонніх серверах, відповідно вимоги до складу та параметрів технічних засобів є не дуже актуальними.

Для користувачів веб-додатку програма не має вимог до складу та параметрів технічних засобів та може завантажуватись на ноутбуках, ПК,

смартфонах та планшетах різноманітних видів та конфігурацій під управлінням різних операційних систем. Для користування додатком користувачу необхідний тільки веб-браузер з підтримкою JavaScript.

2.7.2. Використані програмні засоби

Front-end додаток було реалізовано за допомогою мови програмування TypeScript з використанням бібліотеки React.js, a back-end додаток за допомогою JavaScript з використанням Node.js.

В якості середовища розробки було використано WebStorm від компанії JetBrains. Також використовувалися довільні веб-браузери з підтримкою JavaScript.

В якості бази даних була використана MongoDB – гнучка, документно-орієнтована база даних, що легко масштабується. А для роботи з базою даних використовувалися сервіси платформи MongoDB Cloud.

Для користувача необхідними програмними та технічними засобами є ноутбук, ПК, смартфон, планшет або інший девайс з доступом до мережі Інтернет та браузером, що має пдтримку JavaScript.

2.7.3. Виклик та завантаження програми

Для виклику та завантаження необхідно виконати наступні дії:

- необхідно орендувати або придбати серверне обладнання та доменне ім’я;
- встановити на серверне обладнання серверне забезпечення;
- налаштувати серверне програмне забезпечення;
- за допомогою FTP клієнта приєднатися до сервера та перемістити у папку src усі скомпільовані файли програми;
- підключити доменне ім’я та сертифікати до серверу;
- ввімкнути сервер.

2.7.4. Опис інтерфейсу користувача

Відкриваючи додаток, користувач бачить домашню сторінку, що містить заголовок, поле введення для пошуку товарів за ключовими словами (назвою / виробником), секцію з інформацією про сервіс та кроками для здійснення замовлення.

Інтерфейс є інтуїтивно зрозумілим і основні сторінки та елементи зображено на рис.2.18. - 2.30.

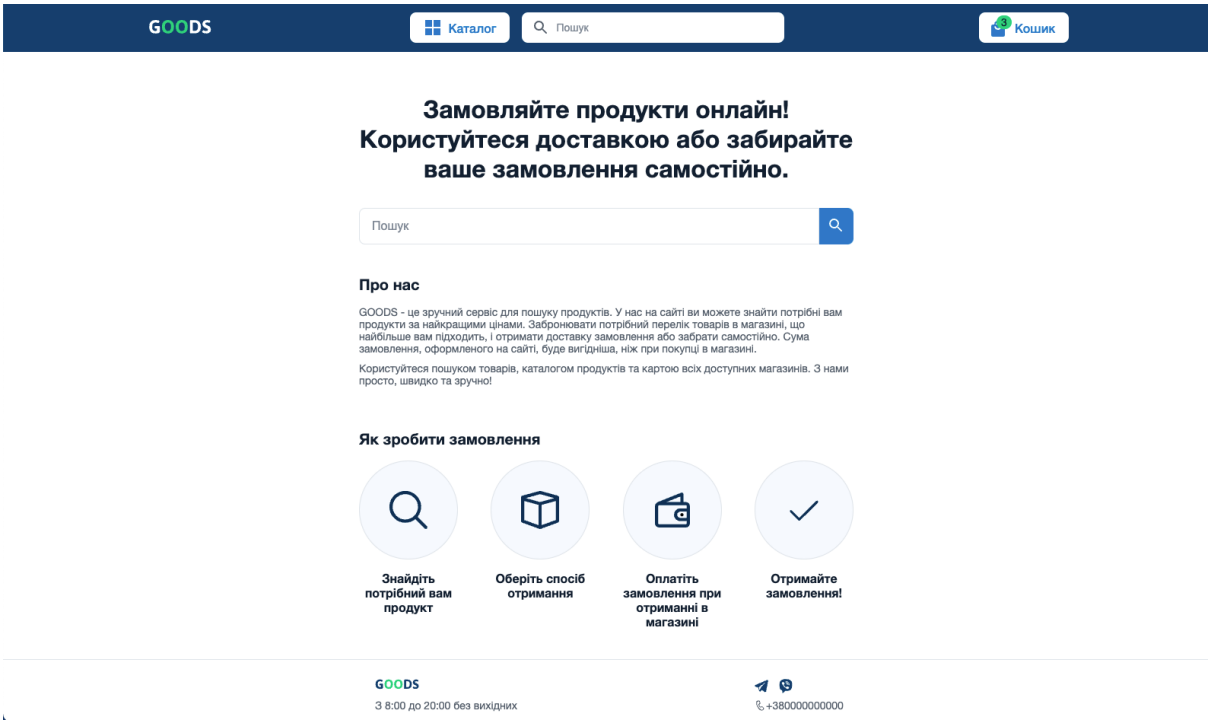

Рис. 2.18. Домашня сторінка

Натиснувши іконку на полі введення, здійснюється пошук товарів, результати зображуються на окремій сторінці. На цій сторінці можна додати товари, що є в наявності, в кошик або ж видалити їх. А також можна переглянути більш детальну інформацію про товар на окремій сторінці, натиснувши на зображення або назву товару.

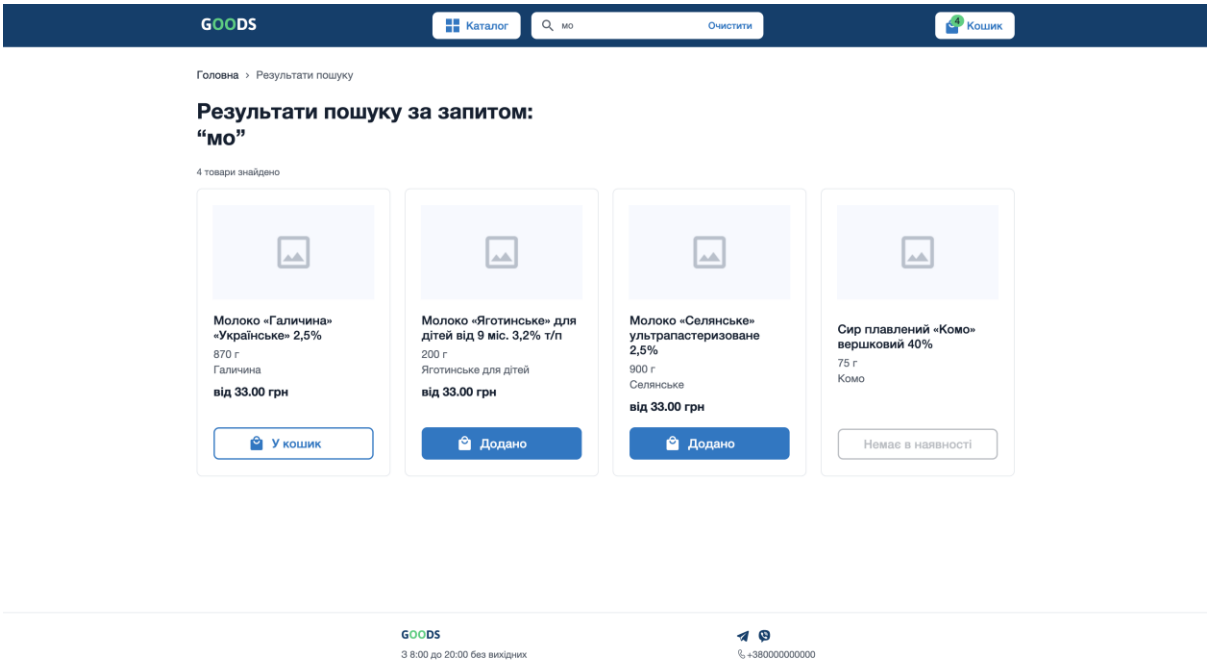

Рис. 2.19. Сторінка з результатами пошуку товарів за назвою

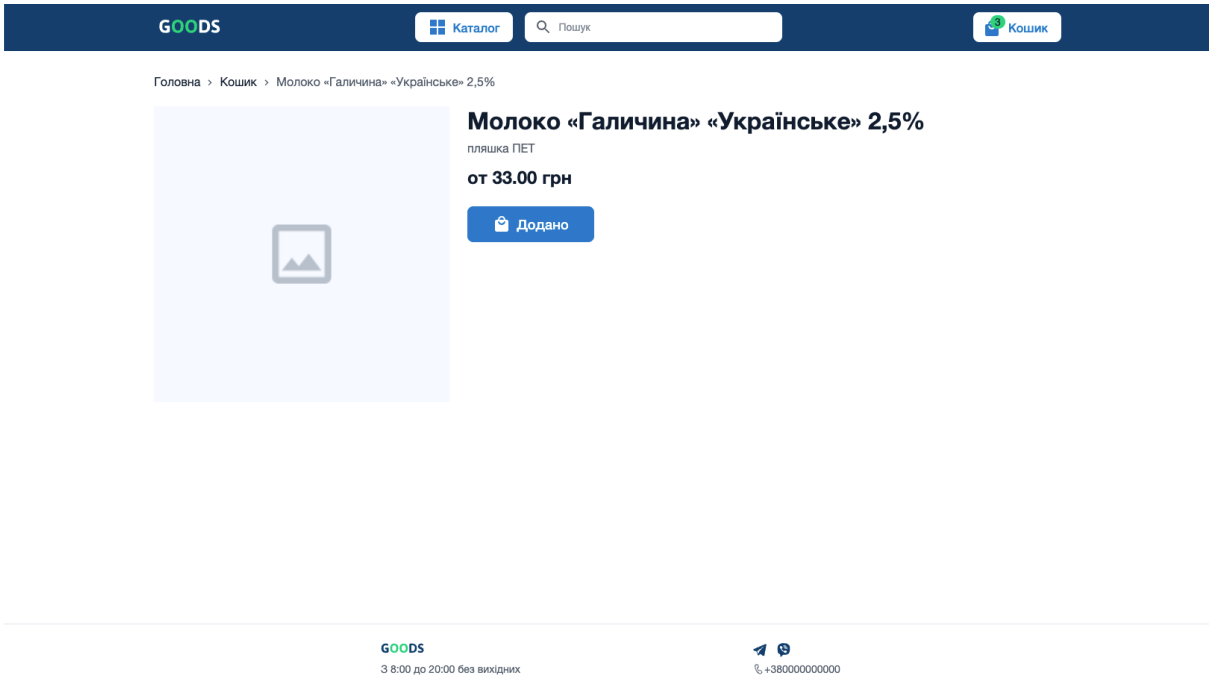

Рис. 2.20. Сторінка з деталями про товар

Також відвідувач сайту може здійснити вибір товарів у потрібній йому категорії, відкривши модальне вікно з каталогом з будь-якої сторінки додатку за допомогою кнопки в шапці.

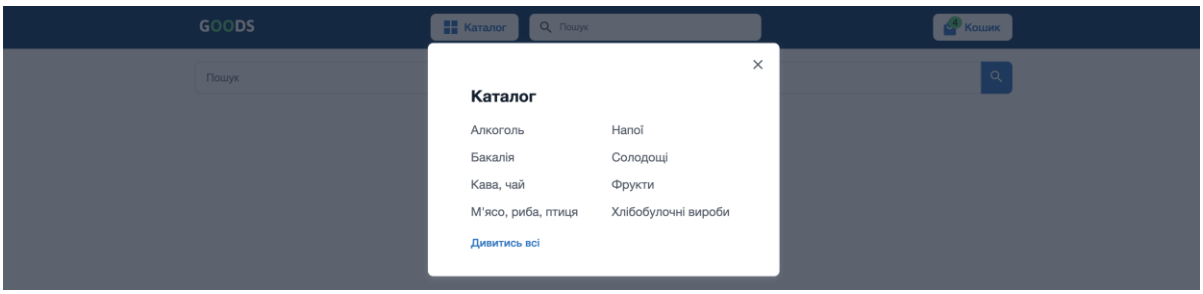

Рис. 2.21. Каталог товарів

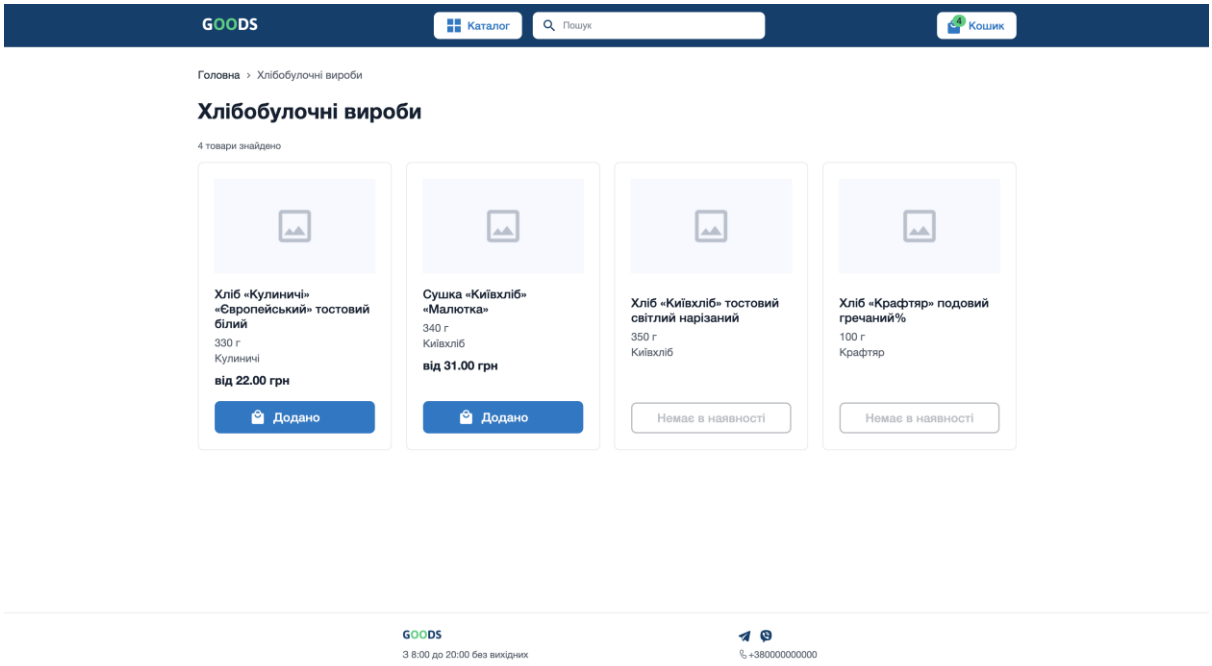

Рис. 2.22. Сторінка з товарами в обраній категорії

Після того, як користувач обрав потрібні товари та додав їх у кошик, він може з будь-якого місця додатку за допомогою кнопки в шапці потрапити на сторінку з вмістом кошика. На даній сторінці надається можливість переглянути обрані товари, змінити їхню кількість або видалити їх з кошику. Потім, натиснувши кнопку «Знайти в магазинах» користувача буде перенаправлено на сторінку з результатами наявності бажаних товарів у магазинах.

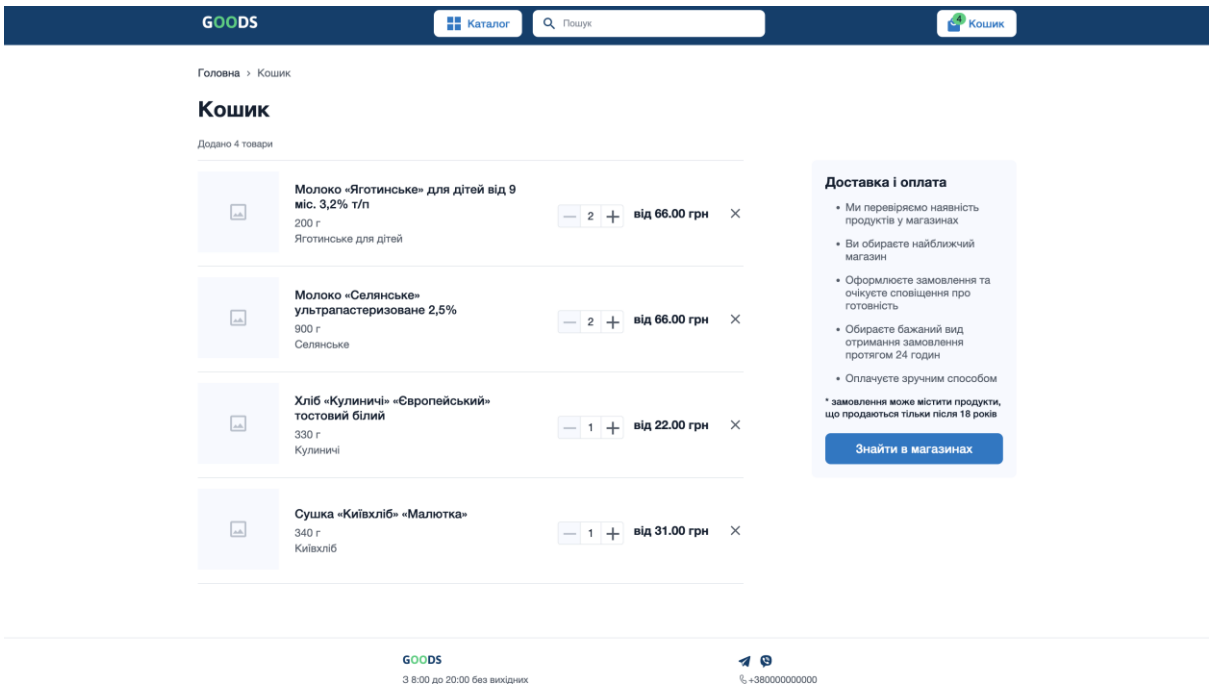

Рис. 2.23. Сторінка з кошиком обраних товарів

На сторінці з пропозиціями магазинів відображаються дані про магазини, наявність в них товарів і загальну вартість наявних товарів із списку обраних. А також справа можна побачити карту, що містить мітки магазинів, при натискання на які відкривається «віконце» з детальнішою інформацією, і мітку місцезнаходження користувача (за умови, що він дав дозвіл на його використання). На цій сторінці користувач може обрати магазин, в якому він хоче зробити замовлення, врахувавши наявність товарів, їхні ціни або відстань до магазину.

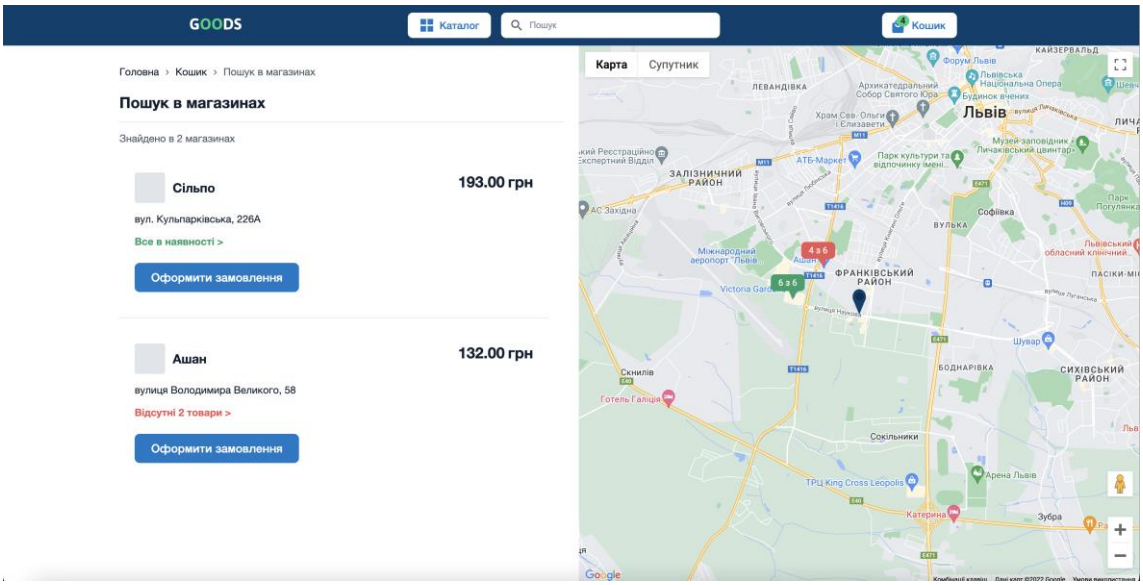

Рис. 2.24. Сторінка з наявністю обраних товарів у магазинах

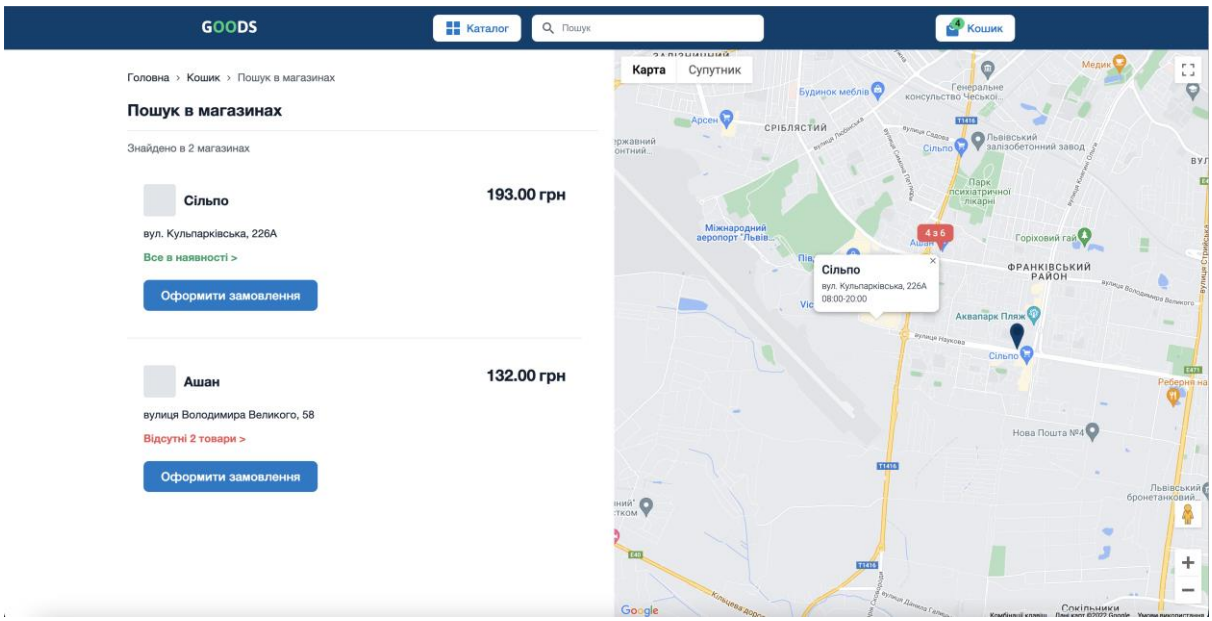

Рис. 2.25. Віконце з інформацією про обраний магазин на сторінці з наявністю обраних товарів у магазинах з

При виборі бажаного магазину відкривається модальне вікно, в якому можна переглянути інформацію про товари і оформити замовлення з товарами у наявності в даному магазині.

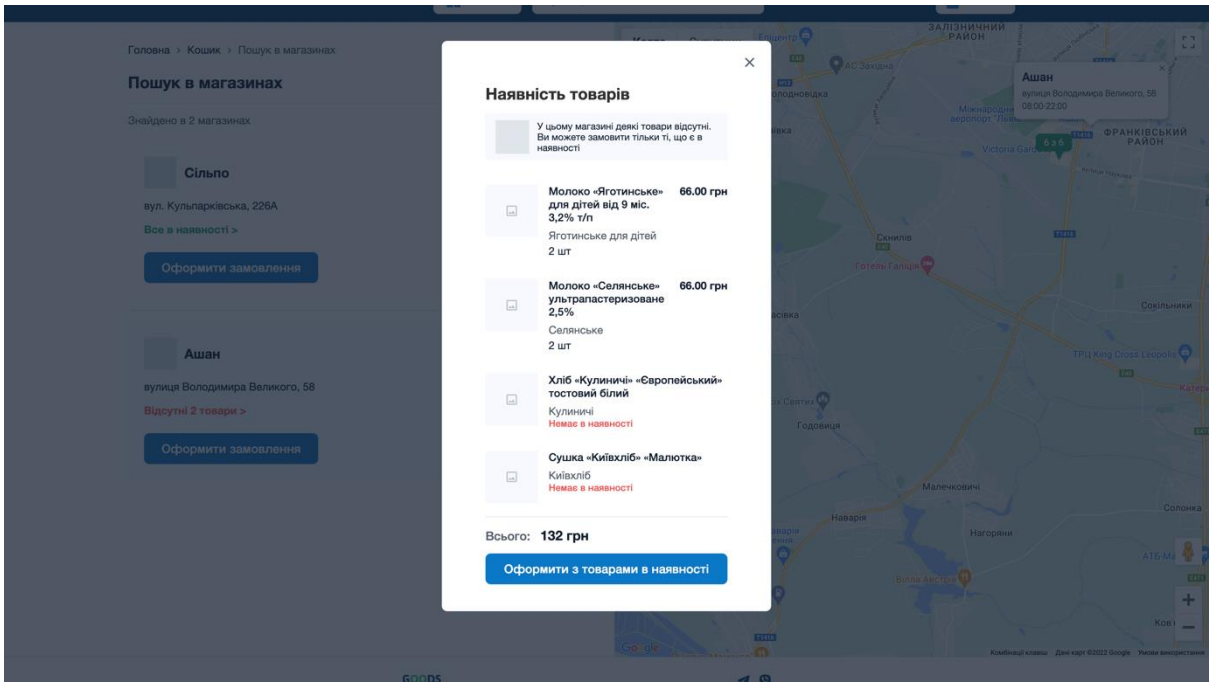

Рис. 2.26. Модальне вікно на сторінці з наявністю обраних товарів у магазинах

Якщо користувачу підходить пропозиція товарів у обраному магазині, він може перейти до оформлення замовлення. Сторінка для оформлення замовлення містить форму для даних користувача та блок з інформацією про складові замовлення. Тут потрібно заповнити усі необхідні і опціональні поля, обрати тип доставки, погодитися з умовами.

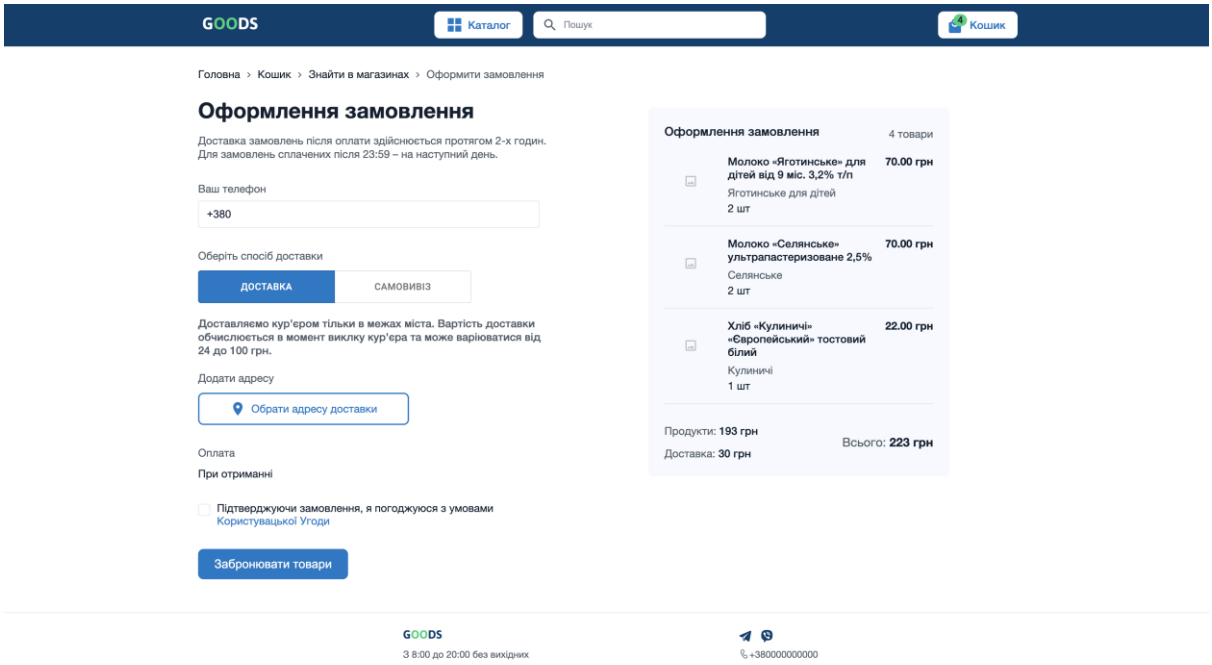

Рис. 2.27. Сторінка з формою для оформлення замовлення

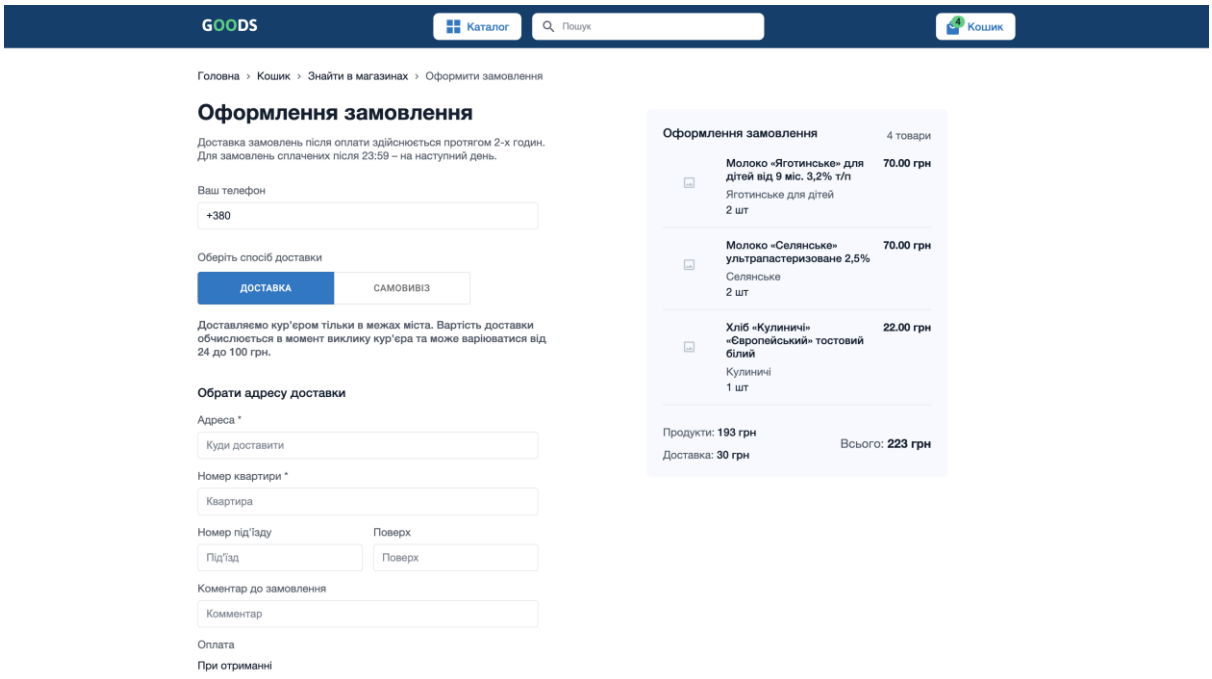

Рис. 2.28. Сторінка з формою для оформлення замовлення з полями для введення адреси доставки

При натисканні на кнопку відбувається валідація введених і обраних значень, і за умов помилок відповідні поля підсвічуються червоним кольором і виводяться повідомлення про помилки.

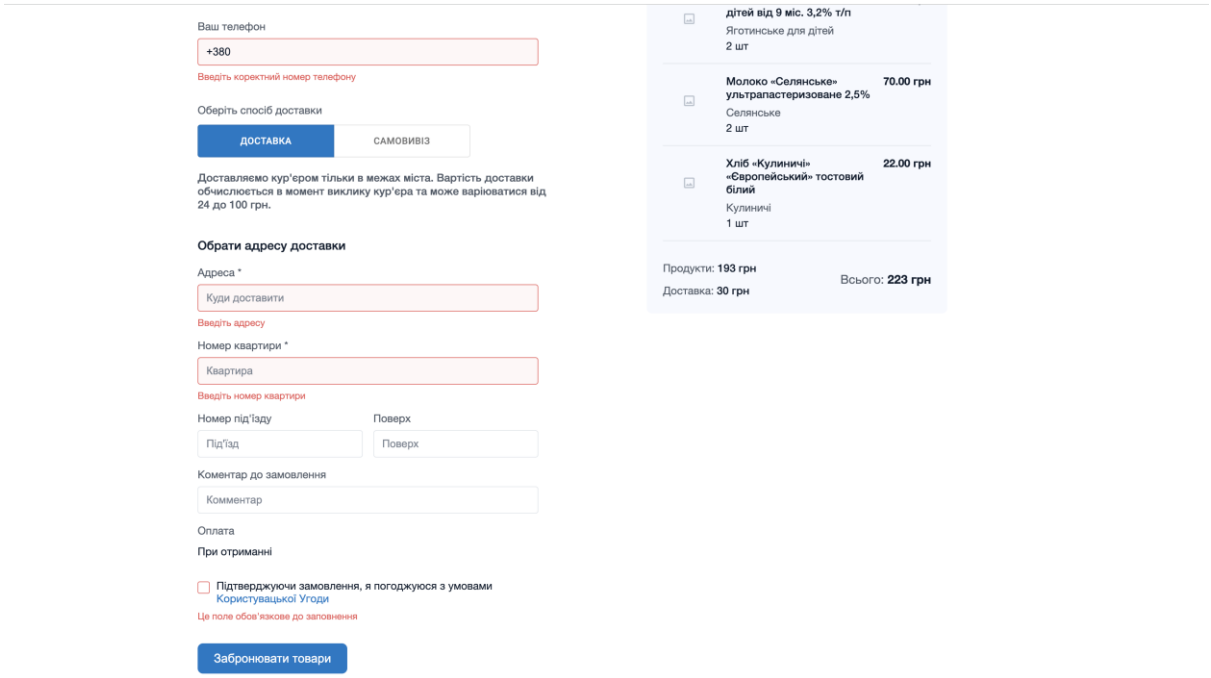

Рис. 2.29. Валідація даних форми для оформлення замовлення

Після успішної валідації відбувається створення замовлення. В разі успішного його створення відображається модальне вікно з відповідним повідомленням та очищується кошик з товарами. Так само у випадку помилки при створенні замовлення відкривається модальне вікно з повідомленням про помилку.

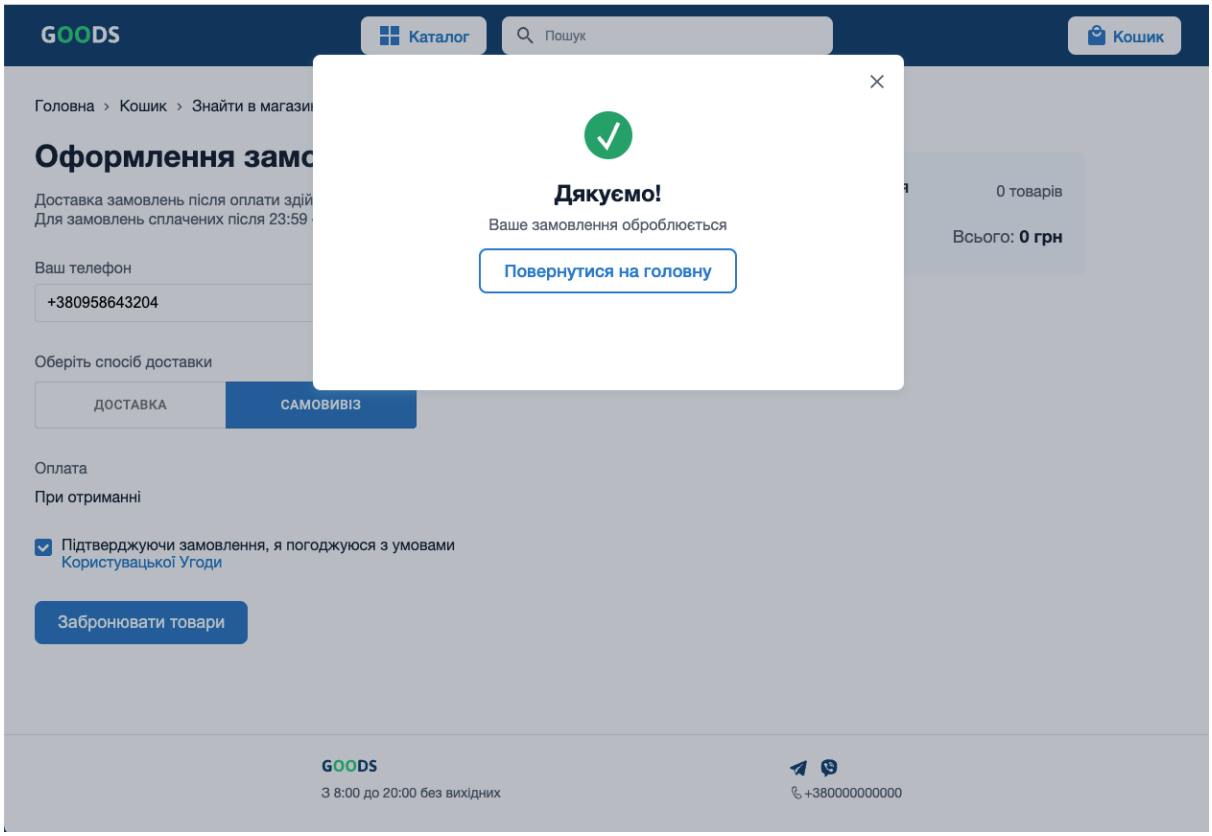

Рис. 2.30. Модальне вікно після успішного оформлення замовлення

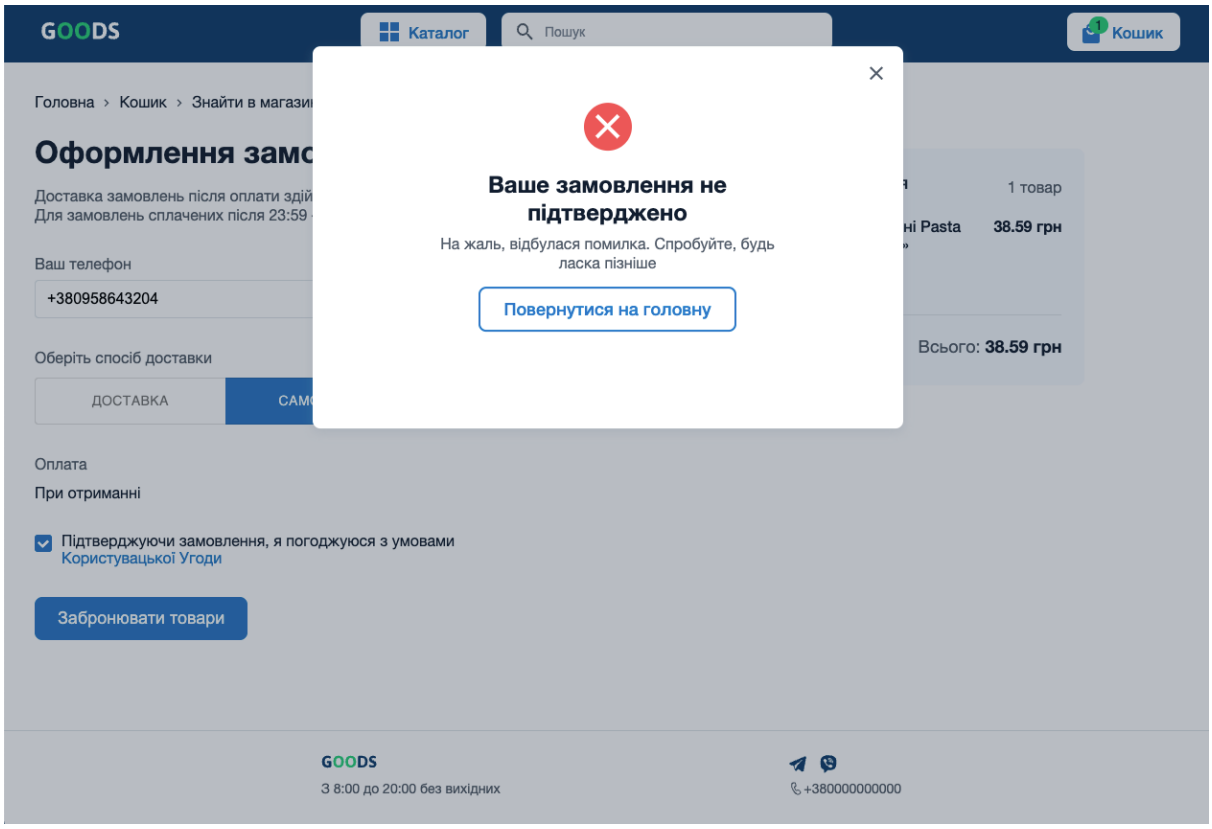

Рис. 2.30. Модальне вікно у випадку помилки під час оформлення замовлення

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Початкові дані:

1. передбачуване число операторів програми – 2310;
2. коефіцієнт корекції програми в ході її розробки – 0.05;
3. коефіцієнт складності програми – 1,7;
4. годинна заробітна плата програміста – 100 грн/год;
5. коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1.4;
6. коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1.4;
7. вартість машино-години ЕОМ – 25 грн/год.

Середня годинна заробітна плата програміста була вирахувана виходячи з даних одного з найбільших українських сайтів для анонімного пошуку роботи «Djinni.co». Станом на кінець 2021 року зарплата Junior Web UI розробника коливається від 400$ до 800$. Виходячи з цього середня заробітна плата програміста буде 600$ у місяць. При курсі валют НБУ на початок червня 2022 року один американський долар дорівнює 29.25 грн, тому середня зарплата в гривнях дорівнює 17550 гривень. При стандартному графіку (176 годин/місяць) зарплата за годину становитиме приблизно 100 грн.

Оскільки для цього проєкту потрібна досить велика потужність ПК для розробки back-end’у на локальному сервері, гарним рішенням буде оренда комп’ютера на час розробки додатку. Вартість оренди комп’ютера на місяць 1400 грн (монітор) та 3000 грн (системний блок). Загалом на місяць оренда коштуватиме 4400 грн, що при стандартному графіку (176 годин/місяць) буде коштувати близько 25 грн/год. До вартості оренди доданий гарантійний ремонт та базовий комплект гарнітури (клавіатура та миша).

Нормування праці в процесі розробки програмного забезпечення істотно ускладнено через творчий характер праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

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

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 ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C \cdot (1 + p),$$

де q – передбачуване число операторів (2310);

C – коефіцієнт складності програми (1,7);

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

Звідси умовне число операторів в програмі:

$$Q = 1{,}7 \cdot 2310 \cdot (1 + 0{,}05) = 4123{,}35$$

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

$$t_u = \frac{Q \cdot B}{(75 \ldots 85) \cdot k} \text{ , людино-годин,}$$

де B – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

k - коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності. При стажі роботи від 5 до 8 років він складає 1,4.

Приймемо збільшення витрат праці внаслідок недостатнього опису завдання не більше 50% (B = 1,2). З урахуванням коефіцієнта кваліфікації k = 1,4, отримуємо витрати праці на вивчення опису завдання:

$$t_u = (4123{,}35 \cdot 1{,}2) / (75 \cdot 1{,}4) = 47{,}124 \text{ людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі визначаються за формулою:

$$t_a = \frac{Q}{(20 \ldots 25) \cdot k} \text{ людино-годин, (3.2)}$$

де Q – умовне число операторів програми;

k – коефіцієнт кваліфікації програміста.

Підставивши відповідні значення в формулу (3.2), отримаємо:

$$t_a = 4123{,}35 / (20 \cdot 1{,}4) = 147{,}2625 \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20 \ldots 25) \cdot k} \text{ людино-годин.}$$

$$t_n = 4123{,}35 / (25 \cdot 1{,}4) = 117{,}81 \text{ людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ:

- за умови автономного налагодження одного завдання:

$$t_{\text{отл}} = \frac{Q}{(4 \ldots 5) \cdot k} \text{ людино-годин.}$$

$$t_{\text{отл}} = 4123{,}35 / (5 \cdot 1{,}4) = 589{,}05 \text{ людино-годин.}$$

- за умови комплексного налагодження завдання:

$$t_{\text{отл}}^{k} = 1{,}5 \cdot t_{\text{отл}} \text{ людино-годин.}$$

$$t_{\text{отл}}^{k} = 1{,}5 \cdot 589{,}05 = 883{,}575 \text{ людино-годин.}$$

Витрати праці на підготовку документації визначаються за формулою:

$$t_{\text{д}} = t_{\text{др}} + t_{\text{до}} \text{ , людино-годин,}$$

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

$$t_{\text{др}} = \frac{Q}{(15 \ldots 20) \cdot k} \text{ , людино-годин,}$$

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

$$t_{\text{д}} = 0{,}75 \cdot t_{\text{др}} \text{ , людино-годин.}$$

Підставляючи відповідні значення, отримаємо:

$$t_{\text{др}} = 4123{,}35 / (18 \cdot 1{,}4) = 163{,}625 \text{ людино-годин.}$$

$$t_{\text{до}} = 0{,}75 \cdot 163{,}625 = 122{,}72 \text{ людино-годин.}$$

$$t_{\text{до}} = 163{,}625 + 122{,}72 = 286{,}34 \text{ людино-годин.}$$

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

$$t = 50 + 47{,}124 + 147{,}2625 + 117{,}81 + 589{,}05 + 286{,}34 = 1237{,}59 \text{ людино-годин.}$$

3.2. Розрахунок витрат на створення програми

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

$$K_{\text{ПО}} = З_{\text{ЗП}} + З_{\text{МВ}} \text{ , грн.}$$

Заробітна плата виконавців визначається за формулою:

$$З_{\text{ЗП}} = t \cdot С_{\text{ПР}} \text{ , грн,}$$

де t – загальна трудомісткість, людино-годин;

$C_{ПР}$ – середня годинна заробітна плата програміста, грн/година

З урахуванням того, що середня годинна зарплата програміста становить 100 грн / год, отримуємо:

$$З_{\text{ЗП}} = 1237{,}59 \cdot 100 = 123759 \text{ грн.}$$

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

$$З_{\text{мв}} = t_{\text{отл}} \cdot С_{\text{мч}} \text{ грн, (3.3)}$$

де $t_{\text{отл}}$– трудомісткість налагодження програми на ЕОМ, год;

$С_{\text{мч}}$- вартість машино-години ЕОМ, грн/год (25 грн/год).

Підставивши в формулу (3.3) відповідні значення, визначимо вартість необхідного для налагодження машинного часу:

$$З_{\text{мв}} = 589.05 \cdot 25 = 14726{,}25 \text{ грн.}$$

Звідси витрати на створення програмного продукту:

$$К_{ПО} = 123759 + 14726{,}25 = 138485{,}25 \text{ грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p} \text{ міс,}$$

де B_k – число виконавців (дорівнює 1);

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

Звідси витрати на створення програмного продукту:

$$T = 1237{,}59 / 1 \cdot 176 \approx 7 \text{ міс.}$$

Висновок: програмне забезпечення розроблене для надання доступу користувачам до основного функціоналу продуктового сервісу, ефективної взаємодії між потенційними споживачами та компаніями-партнерами. Вартість даного програмного забезпечення складає близько 138485,25 грн і не вимагає додаткових витрат при розробці. Очікуваний час розробки складає 7 місяців. Цей термін пов’язаний зі значною кількістю операторів та включає час на дослідження і розробку алгоритму вирішення поставленого завдання, розробку по готовому алгоритму, налагодження програми та підготовку документації.

ВИСНОВКИ

Під час виконання практики було поставлено завдання розробити веб-орієнтовану інформаційну систему пошуку продуктових товарів та послуг.

Ця інформаційна система призначена для надання універсального інструменту для відображення контенту бази магазинів та надання послуг. Практичне призначення даної системи полягає в забезпеченні відвідувачам сайту простого і комфортного доступу до пошуку товарів за ключовими словами або за допомогою каталогу товарів за рахунок оптимальних параметрів візуалізації його вмісту, що зробить процес покупки швидшим і зручнішим, і підвищить ефективність роботи інформаційної системи.

Під час виконання даного проєкту були виконані наступні задачі:

вивчено предметну галузь розв'язуваної задачі;

- створено алгоритм для реалізації поставленого завдання;
- створено макети інтерфейсу системи, базу даних і клієнтську та серверну програму.

Розроблене програмне забезпечення дозволяє:

- формування динамічних web-сторінок на основі шаблону з використанням контенту отриманого з серверу;
- оформлення замовлень з даного магазину.

Програма реалізована на мові програмування JavaScript з використанням бібліотеки ReactJS, фреймворку Node.js та бази даних MongoDB.

В ході виконання даної кваліфікаційної роботи ще було визначено трудомісткість розробленої системи, проведено розрахунок вартості роботи з створення програми, використовуючи середню зарплату розробника. Він складає 138485,25 грн. Також було розраховано час для створення програмного додатку - 1237,59 людино-годин, тобто приблизно 7 місяців.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Frontend frameworks and libraries [Електронний ресурс] - Режим доступу: https://2021.stateofjs.com/en-US/libraries/front-end-frameworks

2. Документація Node.js [Електронний ресурс] - Режим доступу: https://nodejs.org/uk/docs/

3. А.Аллан. Клієнтська розробка для професіоналів. Node.js – СПБ.:2017. – 220с.

4. TypeScript – популярна мова програмування [Електронний ресурс] - Режим доступу: https://www.typescriptlang.org/

5. React [Електронний ресурс] - Режим доступу: https://uk.reactjs.org/docs/getting-started.html

6. Основы React.js [Електронний ресурс] - Режим доступу: https://learn.javascript.ru/screencast/react

7. Документація MongoDB [Електронний ресурс] - Режим доступу: https://www.mongodb.com/docs/

8. Анонімний пошук роботи на Джині [Електронний ресурс] - Режим доступу: https://djinni.co/

9. Вартість аренди ноутбуку почасово [Електронний ресурс] - Режим доступу: https://notebooksbu.com/garantiya-3-goda

10. Руководство по Figma [Електронний ресурс] - Режим доступу: https://www.figma.com/community/file/813826100927416632

11. Robin Wieruch. The Road to React: Your journey to master React.js in JavaScript (2022 Edition) – 365c.

12. Basarat Ali Syed. Beginning Node.js – 326c.

13. Mario Casciaro, Luciano Mammino. Node.js Design Patterns: Design and implement production-grade Node.js applications using proven patterns and techniques, 3rd Edition – 660c.

14. Greg Lim. Beginning Node.js, Express & MongoDB Development – 155c.

15. Amit Phaltankar. MongoDB Fundamentals: A hands-on guide to using MongoDB and Atlas in the real world – 748с.

16. Kyle Simpson. You Don’t Know JS Yet: Get Started – 143с.

17. John Wiley & Sons. HTML and CSS: Design and Build Websites – 490с.

18. Alex Banks. Learning React: Functional Web Development with React and Redux – 350с.

19. Alex Banks. Learning React: Modern Patterns for Developing React Apps – 310с.

20. Документація Google Maps Platform [Електронний ресурс] – Режим доступу: https://developers.google.com/maps/documentation

21. Документація React [Електронний ресурс] – Режим доступу: https://en.reactjs.org/

22. Документація Redux [Електронний ресурс] – Режим доступу: https://redux.js.org/

23. Документація redux-saga [Електронний ресурс] – Режим доступу: https://redux-saga.js.org/

24. Документація Express [Електронний ресурс] – Режим доступу: https://expressjs.com/ru/

25. Документація Mongoose [Електронний ресурс] – Режим доступу: https://mongoosejs.com/

26. Курс Ghulam Abbas. Node.js, Express, MongoDB Bootcamp 2022 [Електронний ресурс] – Режим доступу: https://www.udemy.com/course/nodejs-express-mongodb-bootcamp-with-real-projects/

27. Курс Stephen Grider. Modern React with Redux [Електронний ресурс] – Режим доступу: https://www.udemy.com/course/react-redux/

28. Курс Tom Phillips. Redux Saga with React: Fast-track Redux Saga intro course [Електронний ресурс] – Режим доступу: https://www.udemy.com/course/redux-saga/

29. Курс Jonas Schmedtmann. The Complete JavaScript Course 2022: Form Zero to Exprert! [Електронний ресурс] – Режим доступу: https://www.udemy.com/course/the-complete-javascript-course/

30. Методичні рекомендації до виконання кваліфікаційних робіт здобувачів першого рівня вищої освіти спеціальності 121 Інженерія програмного забезпечення / О.С. Шевцова, І.М. Удовик; Д : НТУ «Дніпровська політехніка», 2021. – 65 с.

ДОДАТОК А

КОД ПРОГРАМИ

Код back-end додатку:

app.js

```
const express = require('express');
const morgan = require('morgan');
const rateLimit = require('express-rate-limit');
const helmet = require('helmet');
const mongoSanitize = require('express-mongo-sanitize');
const xss = require('xss-clean');
const cors = require('cors');

const AppError = require('./utils/appError');
const globalErrorHandler = require('./controllers/errorController');

require('./models/proposalModel');

const categoryRouter = require('./routes/categoryRoutes');
const productRouter = require('./routes/productRoutes');
const shopRouter = require('./routes/shopRoutes');
const orderRouter = require('./routes/orderRoutes');
const userRouter = require('./routes/userRoutes');

const app = express();

// GLOBAL MIDDLEWARES

// Set security HTTP headers
app.use(helmet());

// Development logging
if (process.env.NODE_ENV === 'development') {
  app.use(morgan('dev'));
}

// Limit requests from same IP
const limiter = rateLimit({
  max: 1000,
  windowMs: 60 * 60 * 1000,
  message: 'Too many requests from this IP, please try again in an hour!',
});
```

```
app.use(cors());
app.options('*', cors());

app.use('/api', limiter);

// Body parser, reading data from body into req.body
app.use(
  express.json({
    limit: '10kb',
  })
);

// Data sanitization against NoSQL query injection
app.use(mongoSanitize());

// Data sanitization against XSS
app.use(xss());

// ROUTES
app.use('/api/v1/categories', categoryRouter);
app.use('/api/v1/products', productRouter);
app.use('/api/v1/shops', shopRouter);
app.use('/api/v1/orders', orderRouter);
app.use('/api/v1/users', userRouter);

app.all('*', (req, res, next) => {
  next(new AppError(`Can't find ${req.originalUrl} on this server!`, 404));
});

app.use(globalErrorHandler);

module.exports = app;
```

categoryRoutes.js

```
const express = require('express');
const categoryController = require('../controllers/categoryController');

const router = express.Router();

router.route('/').get(categoryController.getAllCategories);
router.route('/:id').get(categoryController.getCategory);

module.exports = router;
```

orderRoutes.js

```
const express = require('express');
const orderController = require('../controllers/orderController');

const router = express.Router();

router.route('/').post(orderController.createOrder);

module.exports = router;
```

productRoutes.js

```
const express = require('express');
const productController = require('../controllers/productController');
const authController = require('../controllers/authController');

const router = express.Router();

router
  .route('/')
  .get(productController.getAllProducts)
  .post(
    authController.protect,
    authController.restrictTo('admin'),
    productController.addProduct
  );
router.route('/:id').get(productController.getProduct);

module.exports = router;
```

shopRoutes.js

```
const express = require('express');
const shopController = require('../controllers/shopController');
const authController = require('../controllers/authController');

const router = express.Router();

router.route('/proposals').post(shopController.getShopProposals);

router.use(authController.protect);

router.use(authController.restrictTo('shop'));
```

```
router
  .route('/me')
  .get(shopController.getMyShops)
  .post(shopController.addMyShop);

module.exports = router;
```

userRoutes.js

```
const express = require('express');
const authController = require('../controllers/authController');
const userController = require('../controllers/userController');

const router = express.Router();

router.post('/signup', authController.signup);
router.post('/login', authController.login);

router.post('/forgotPassword', authController.forgotPassword);
router.patch('/resetPassword/:token', authController.resetPassword);

// Protect routes after this middleware
router.use(authController.protect);

router.get('/me', userController.getMe);
router.patch('/updatePassword', authController.updatePassword);

module.exports = router;
```

categoryController.js

```
const Category = require('../models/categoryModel');
const factory = require('../controllers/handlerFactory');

exports.getAllCategories = factory.getAll(Category);
exports.getCategory = factory.getOne(Category, {
  path: 'products',
  populate: {
    path: 'proposals',
    options: { select: { price: 1 } },
  },
});
```

orderController.js

```
const catchAsync = require('../utils/catchAsync');
const Order = require('../models/orderModel');
const AppError = require('../utils/appError');

exports.createOrder = catchAsync(async (req, res, next) => {
  if (!req.body?.products?.length) {
    return next(new AppError('Order must contain at least one product.', 400));
  }

  const order = await Order.create(req.body);

  res.status(200).json({
    status: 'success',
    data: order,
  });
});
```

productController.js

```
const Product = require('../models/productModel');
const factory = require('./handlerFactory');
const catchAsync = require('../utils/catchAsync');

exports.getAllProducts = catchAsync(async (req, res, next) => {
  let products = await Product.find().populate({
    path: 'proposals',
    options: { select: { price: 1 } },
  });

  if (req.query?.searchValue) {
    const searchValue = req.query?.searchValue.toLowerCase();
    products = products.filter((product) =>
      product.name.toLowerCase().includes(searchValue)
    );
  }

  // SEND RESPONSE
  res.status(200).json({
    status: 'success',
    results: products.length,
    data: products,
  });
});
```

```
exports.getProduct = factory.getOne(Product, {
  path: 'proposals',
  options: { select: { price: 1 } },
});

exports.addProduct = catchAsync(async (req, res, next) => {
  const newProduct = await Product.create({
    name: req.body.name,
    category_id: req.body.category_id,
    weight: req.body.weight,
    packing: req.body.packing,
    manufacturer: req.body.manufacturer,
    image: req.body.image,
  });

  res.status(200).json({
    status: 'success',
    data: newProduct,
  });
});
```

shopController.js

```
const catchAsync = require('../utils/catchAsync');
const AppError = require('../utils/appError');
const Product = require('../models/productModel');
const Shop = require('../models/shopModel');

exports.getShopProposals = catchAsync(async (req, res, next) => {
  if (!req.body.length) {
    return next(
      new AppError('Please provide a list of desired products.', 400)
    );
  }

  const requestedProducts = req.body;
  const requestedProductsDetailedInfo = await Promise.all(
    requestedProducts.map((product) => Product.findById(product.product_id))
  );
  const shops = await Shop.find().populate({ path: 'products' });

  const proposals = shops.map((shop) => {
    const desiredProductsAvailability = requestedProductsDetailedInfo.map(
      (product) => {
```

```
      const shopProduct = shop.products.find((item) => {
        return item.product_id.toString() === product.id.toString();
      });

      const countDesired = requestedProducts.find(
        (item) => item.product_id.toString() === product.id.toString()
      ).count;

      let count;

      if (!shopProduct) {
        count = 0;
      } else if (shopProduct.count >= countDesired) {
        count = countDesired;
      } else {
        count = shopProduct.count;
      }

      return {
        id: product._id,
        name: product.name,
        weight: product.weight,
        packing: product.packing,
        manufacturer: product.manufacturer,
        image: product.image,
        price: shopProduct ? shopProduct.price : null,
        count_desired: countDesired,
        count,
      };
    }
  );

  return {
    id: shop._id,
    name: shop.name,
    address: shop.address,
    schedule: shop.schedule,
    coordinates: shop.coordinates,
    proposal: desiredProductsAvailability,
  };
});

res.status(200).json({
  status: 'success',
  data: proposals,
```

```
  });
});

exports.addMyShop = catchAsync(async (req, res, next) => {
  const newShop = await Shop.create({
    name: req.body.name,
    address: req.body.address,
    coordinates: req.body.coordinates,
    schedule: req.body.schedule,
    owner: req.user.id,
  });

  res.status(200).json({
    status: 'success',
    data: newShop,
  });
});

exports.getMyShops = catchAsync(async (req, res, next) => {
  const shops = await Shop.find({ owner: req.user.id });

  res.status(200).json({
    status: 'success',
    results: shops.length,
    data: shops,
  });
});
```

userController.js

```
const User = require('../models/userModel');
const catchAsync = require('../utils/catchAsync');

exports.getMe = catchAsync(async (req, res, next) => {
  const userData = await User.findById(req.user.id).select('-password');

  res.status(200).json({
    status: 'success',
    data: userData,
  });
});
```

errorController.js

```
const AppError = require('../utils/appError');

const handleCastErrorDB = (err) => {
  const message = `Invalid ${err.path}: ${err.value}.`;
  return new AppError(message, 400);
};

const handleDuplicateFieldsDB = (err) => {
  const value = err.errmsg.match(/(["'])(?:(?=(\\?))\2.)*?\1/)[0];
  const message = `Duplicate field value: ${value}. Please use another value!`;
  return new AppError(message, 400);
};

const handleValidationErrorDB = (err) => {
  const errors = Object.values(err.errors).map((item) => item.message);
  const message = `Invalid input data. ${errors.join('. ')}`;
  return new AppError(message, 400);
};

const handleJWTError = () => new AppError('Invalid token.', 401);

const handleJWTExpiredError = () =>
  new AppError('Your token has expired.', 401);

const sendErrorDev = (err, res) => {
  res.status(err.statusCode).json({
    status: err.status,
    message: err.message,
    stack: err.stack,
    error: err,
  });
};

const sendErrorProd = (err, res) => {
  // Operational, trusted error: send message to client
  if (err.isOperational) {
    res.status(err.statusCode).json({
      status: err.status,
      message: err.message,
    });

    // Programming or other unknown error: don't leak error details
  } else {
```

```
    // 1) Log error
    console.error('ERROR 💥', err);
    // 2) Send generic message
    res.status(500).json({
      status: 'error',
      message: 'Something went wrong!',
    });
  }
};

module.exports = (err, req, res, next) => {
  err.statusCode = err.statusCode || 500;
  err.status = err.status || 'error';

  if (process.env.NODE_ENV === 'development') {
    sendErrorDev(err, res);
  } else if (process.env.NODE_ENV === 'production') {
    let error = { ...err };
    if (err.name === 'CastError') {
      error = handleCastErrorDB(err);
    }
    if (err.code === 11000) {
      error = handleDuplicateFieldsDB(err);
    }
    if (err.name === 'ValidationError') {
      error = handleValidationErrorDB(err);
    }
    if (err.name === 'JsonWebTokenError') {
      error = handleJWTError();
    }
    if (err.name === 'TokenExpiredError') {
      error = handleJWTExpiredError();
    }
    sendErrorProd(error, res);
  }
};
```

handleFactory.js

```
const catchAsync = require('../utils/catchAsync');
const AppError = require('../utils/appError');
const APIFeatures = require('../utils/apiFeatures');

exports.deleteOne = (Model) =>
  catchAsync(async (req, res, next) => {
```

```
    const doc = await Model.findByIdAndDelete(req.params.id);

    if (!doc) {
      return next(new AppError('No document found with that ID', 404));
    }

    res.status(204).json({
      status: 'success',
      data: null,
    });
  });

exports.updateOne = (Model) =>
  catchAsync(async (req, res, next) => {
    const doc = await Model.findByIdAndUpdate(req.params.id, req.body, {
      new: true,
      runValidators: true,
    });

    if (!doc) {
      return next(new AppError('No document found with that ID', 404));
    }

    res.status(200).json({
      status: 'success',
      data: doc,
    });
  });

exports.createOne = (Model) =>
  catchAsync(async (req, res, next) => {
    const doc = await Model.create(req.body);
    res.status(200).json({
      status: 'success',
      data: doc,
    });
  });

exports.getOne = (Model, populateOptions) =>
  catchAsync(async (req, res, next) => {
    let query = Model.findById(req.params.id);
    if (populateOptions) {
      query = query.populate(populateOptions);
    }
    const doc = await query;
```

```
    if (!doc) {
      return next(new AppError('No document found with that ID', 404));
    }

    res.status(200).json({
      status: 'success',
      data: doc,
    });
  });

exports.getAll = (Model, populateOptions) =>
  catchAsync(async (req, res, next) => {
    let query = Model.find();

    if (populateOptions) {
      query = query.populate(populateOptions);
    }

    // EXECUTE QUERY
    const features = new APIFeatures(query, req.query)
      .filter()
      .sort()
      .limitFields()
      .paginate();
    const docs = await features.query;

    // SEND RESPONSE
    res.status(200).json({
      status: 'success',
      results: docs.length,
      data: docs,
    });
  });
```

categoryModel.js

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

const categorySchema = new mongoose.Schema(
  {
    name: {
      type: String,
      required: [true, 'Category must have a name.'],
      unique: true,
```

```
      minlength: [
        3,
        'Category name must have more or equal then 3 characters.',
      ],
    },
    parent_id: {
      type: mongoose.Schema.ObjectId,
      ref: 'Category',
      default: null,
    },
  },
  {
    toJSON: {
      virtuals: true,
    },
    toObject: {
      virtuals: true,
    },
  }
);

categorySchema.virtual('products', {
  ref: 'Product',
  foreignField: 'category_id',
  localField: '_id',
});

const Category = mongoose.model('Category', categorySchema);

module.exports = Category;
```

orderModel.js

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

const orderSchema = new mongoose.Schema({
  // TODO: add validation for number
  phone: {
    type: String,
    required: [true, 'Order must have a number.'],
  },
  shop_id: {
    type: mongoose.Schema.ObjectId,
    ref: 'Shop',
    required: [true, 'Order must have a shop id.'],
```

```
  },
  created_at: {
    type: Date,
    default: Date.now(),
  },
  payment_method: {
    type: String,
    default: 'in_place',
  },
  delivery_method: {
    type: String,
    default: 'in_place',
  },
  products: [
    {
      product_id: {
        type: mongoose.Schema.ObjectId,
        ref: 'Product',
      },
      count: Number,
    },
  ],
});

const Order = mongoose.model('Order', orderSchema);

module.exports = Order;
```

productModel.js

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

const productSchema = new mongoose.Schema(
  {
    name: {
      type: String,
      unique: true,
      required: [true, 'Product must have a name.'],
      minlength: [5, 'Product name must have more or equal then 5 characters.'],
      maxlength: [
        80,
        'Product name must have less or equal then 40 characters.',
      ],
    },
    category_id: {
```

```
      type: mongoose.Schema.ObjectId,
      ref: 'Category',
      required: [true, 'Product must belong to a category.'],
    },
    weight: {
      type: String,
      required: [true, 'Product must have a weight.'],
    },
    packing: {
      type: String,
      required: [true, 'Product must have a packing.'],
    },
    manufacturer: {
      type: String,
      required: [true, 'Product must have a manufacturer.'],
    },
    image: String,
    proposals: [
      {
        type: mongoose.Schema.ObjectId,
        ref: 'Proposal',
      },
    ],
  },
  {
    toJSON: {
      virtuals: true,
    },
    toObject: {
      virtuals: true,
    },
  }
);

productSchema.index({ name: 'text' });

const Product = mongoose.model('Product', productSchema);

module.exports = Product;
```

proposalModal.js

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

const proposalsSchema = new mongoose.Schema(
```

```
  {
    product_id: {
      type: mongoose.Schema.ObjectId,
      ref: 'Product',
    },
    shop_id: {
      type: mongoose.Schema.ObjectId,
      ref: 'Shop',
    },
    price: {
      type: Number,
      required: [true, 'Product must have a price.'],
    },
    count: Number,
  },
  {
    toJSON: {
      virtuals: true,
    },
    toObject: {
      virtuals: true,
    },
  }
);

const Proposal = mongoose.model('Proposal', proposalsSchema);

module.exports = Proposal;
```

shopModel.js

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

const shopSchema = new mongoose.Schema(
  {
    name: {
      type: String,
      required: [true, 'Shop must have a name.'],
    },
    address: {
      type: String,
      required: [true, 'Shop must have an address.'],
    },
    coordinates: [Number],
    schedule: {
```

```
      type: String,
      required: [true, 'Shop must have a schedule.'],
    },
    owner: {
      type: mongoose.Schema.ObjectId,
      ref: 'User',
    },
  },
  {
    toJSON: {
      virtuals: true,
    },
    toObject: {
      virtuals: true,
    },
  }
);

shopSchema.virtual('products', {
  ref: 'Proposal',
  foreignField: 'shop_id',
  localField: '_id',
});

const Shop = mongoose.model('Shop', shopSchema);

module.exports = Shop;
```

Код front-end додатку

index.tsx

```
import ReactDOM from 'react-dom';
import { Provider } from 'react-redux';
import App from './App';
import store from './store';
import './index.scss';

store()
  .then((s) => {
    ReactDOM.render(
      <Provider store={s}>
        <App />
      </Provider>,
      document.getElementById('root')
```

```
    );
  })
  .catch((err) => {
    // eslint-disable-next-line
    console.error(err.message);
  });
```

App.tsx

```
import { useEffect } from 'react';
import i18n from 'i18next';
import { initReactI18next } from 'react-i18next';
import { BrowserRouter as Router, Switch } from 'react-router-dom';
import InitComponent from './containers/InitComponent';
import {
  getFromLocalStorage,
  setToLocalStorage,
} from './global/helpers/localStorageHelper';

const App = () => {
  const ru = require('./global/translations/ua.json');

  useEffect(() => {
    if (!getFromLocalStorage('lang')) {
      setToLocalStorage('lang', 'ua');
    }
  }, []);

  i18n.use(initReactI18next).init({
    resources: {
      ru,
    },
    lng: getFromLocalStorage('lang'),
    fallbackLng: 'ru',
    interpolation: {
      escapeValue: false,
    },
  });

  return (
    <Router>
      <Switch>
        < MainPage />
      </Switch>
    </Router>
```

```
  );
};

export default App;
```

MainPage.tsx

```
import { Route, Switch } from 'react-router-dom';
import withAuth from '../../global/hocs/withAuth';
import Header from '../../components/Header';
import Footer from '../../components/Footer';
import HomePage from '../HomePage';
import ProductsPage from '../ProductsPage';
import ProductPage from '../ProductPage';
import CartPage from '../CartPage';
import ShopsPage from '../ShopsPage';
import OrderPage from '../OrderPage';
import DefaultLayout from '../../components/DefaultLayout';

const MainPage = () => {
  return (
    <>
      <Header />

      <Switch>
        <Route exact path="/shops" component={ShopsPage} />

        <DefaultLayout>
          <Route exact path="/products" component={ProductsPage} />
          <Route path="/products/:productId" component={ProductPage} />
          <Route exact path="/cart" component={CartPage} />
          <Route exact path="/order" component={OrderPage} />
          <Route exact path="/" component={HomePage} />
        </DefaultLayout>
      </Switch>

      <Footer />
    </>
  );
};

export default withAuth(MainPage);
```

HomePage.tsx

```
import { useState } from 'react';
import { useTranslation } from 'react-i18next';
import SearchBar from '../../components/SearchBar';
import AboutUs from './components/AboutUs';
import OrderSteps from './components/OrderSteps';
import styles from './HomePage.module.scss';

const HomePage = () => {
  const { t } = useTranslation();
  const [searchValue, setSearchValue] = useState<string>('');

  return (
    <div className={styles.mainContainer}>
      <div className={styles.mainTitle}>{t('HomePage.TITLE_TEXT')}</div>
      <div className={styles.searchBarContainer}>
        <SearchBar value={searchValue} setValue={setSearchValue} />
      </div>
      <AboutUs />
      <OrderSteps />
    </div>
  );
};

export default HomePage;
```

SearchBar.tsx

```
import { ChangeEvent, useState } from 'react';
import { useHistory } from 'react-router-dom';
import { ClickAwayListener } from '@mui/material';
import { useTranslation } from 'react-i18next';
import styles from './SearchBar.module.scss';
import searchIcon from '../../global/media/search.svg';
import { PRODUCTS_ROUTE } from '../../global/constants';

interface ProductsSearchProps {
  value: string;
  setValue: (value: string) => void;
}

const SearchBar = ({ value, setValue }: ProductsSearchProps) => {
  const history = useHistory();
  const { t } = useTranslation();
```

```
const [isSearchActive, setIsSearchActive] = useState<boolean>(false);

const handleSearch = (event: ChangeEvent<HTMLInputElement>) => {
  setValue(event.target.value);
};

const handleSearchClear = () => {
  setValue('');
};

const goToProductsPage = () => {
  if (value.length) {
    history.push(`${PRODUCTS_ROUTE}?search=${value}`);
  }
};

return (
  <div className={styles.mainContainer}>
    <ClickAwayListener
      onClickAway={() => {
        setIsSearchActive(false);
      }}
    >
      <div
        onClick={() => {
          setIsSearchActive(true);
        }}
        className={styles.searchInputContainer}
      >
        <div
          className={`${styles.searchInput} ${
            isSearchActive && styles.searchInputActive
          }`}
        >
          <input
            placeholder={t('SearchBar.SEARCH_PLACEHOLDER')}
            value={value}
            onChange={handleSearch}
            type="text"
            onKeyPress={(e) => {
              if (e.key === 'Enter' && value) {
                goToProductsPage();
              }
            }}
```

```
          />
          {value.length ? (
            <button
              onClick={handleSearchClear}
              className={styles.searchClearButton}
              type="button"
            >
              {t('SearchBar.SEARCH_CLEAR_BTN')}
            </button>
          ) : null}
        </div>
        <button
          className={styles.searchButton}
          onClick={goToProductsPage}
          type="button"
        >
          <img src={searchIcon} alt="search icon" />
        </button>
      </div>
    </ClickAwayListener>
  </div>
 );
};

export default SearchBar;
```

AboutUs.tsx

```
import { useTranslation } from 'react-i18next';
import styles from './AboutUs.module.scss';

const AboutUs = () => {
 const { t } = useTranslation();

 return (
  <div className={styles.aboutUsContainer}>
   <div className={styles.title}>{t('AboutUs.TITLE_TEXT')}</div>
   <div className={styles.subtitle}>
    <div className={styles.firstPart}>{t('AboutUs.SUBTITLE1_TEXT')}</div>
    <div
className={styles.secondPart}>{t('AboutUs.SUBTITLE2_TEXT')}</div>
   </div>
  </div>
 );
};
```

```
export default AboutUs;
```

ProductsPage.tsx

```
import { useEffect } from 'react';
import { useDispatch, useSelector } from 'react-redux';
import { useTranslation } from 'react-i18next';
import { useLocation } from 'react-router-dom';
import * as actions from './actions';
import { selectors } from './reducer';
import { getProductsCount } from '../../global/helpers';
import styles from './ProductsPage.module.scss';
import Loader from '../../components/Loader';
import ProductCard from '../../components/ProductCard';
import { ProductCardVariant } from '../../components/ProductCard/ProductCard';
import ProductsBreadCrumbs from
'../../components/Breadcrumbs/ProductsBreadCrumbs';

const ProductsPage = () => {
  const { t } = useTranslation();
  const dispatch = useDispatch();

  const params = new URLSearchParams(useLocation().search);
  const searchValue = params.get('search');
  const categoryId = params.get('categoryId');
  const categoryName = params.get('categoryName');

  const loading = useSelector(selectors.productsPageLoading);
  const data = useSelector(selectors.productsPageData);
  const sortedData = data
    ? data.slice().sort((_a, b) => (b.price ? 1 : -1))
    : data;

  useEffect(() => {
    if (searchValue) {
      dispatch(actions.getProductsBySearchValue.request(searchValue));
    } else if (categoryId) {
      dispatch(actions.getProductsByCategoryId.request(categoryId));
    }
  }, [dispatch, searchValue, categoryId]);

  const pageContent = (
    <>
      {sortedData && (
```

```
      <>
        <div className={styles.topBlock}>
          <div className={styles.header}>
            <div className={styles.headerText}>
              {searchValue ? (
                <>
                  {sortedData.length
                    ? t('ProductsPage.TITLE_TEXT')
                    : t('ProductsPage.NOTHING_FOUND')}{' '}
                  <br />“{searchValue}”
                </>
              ) : (
                <>
                  {categoryId && categoryName && (
                    <>
                      {!sortedData.length &&
                        t('ProductsPage.NOTHING_FOUND_IN_CATEGORY')}{' '}
                      {categoryName}
                    </>
                  )}
                </>
              )}
            </div>
            <div className={styles.subheaderText}>
              {getProductsCount(sortedData.length)}{' '}
              {t('ProductsPage.SUBTITLE_TEXT')}
            </div>
          </div>
        </div>
        <div className={styles.cardsContainer}>
          {sortedData.map((product) => (
            <div key={product.id} className={styles.card}>
              <ProductCard variant={ProductCardVariant.SEARCH} {...product} />
            </div>
          ))}
        </div>
      </>
    )}
  </>
);

return (
  <div className={styles.main}>
    <ProductsBreadCrumbs categoryName={categoryName} />
    {loading ? <Loader /> : pageContent}
```

```
    </div>
  );
};

export default ProductsPage;
```

helpers.ts

```
export const capitalizeFirstLetter = (string: string): string => {
  return string[0].toUpperCase() + string.slice(1).toLowerCase();
};

export const sortByAlphabet = (x: any, y: any) => {
  return x.name.localeCompare(y.name);
};

export const getLastDigit = (number: number) => {
  const splitCount = number.toString().split('');
  return Number(splitCount[splitCount.length - 1]);
};

export const getProductsCount = (count: number) => {
  if (count > 20 || count <= 4) {
    const lastDigit = getLastDigit(count);
    if (lastDigit === 1) {
      return `${count} товар`;
    }
    if (lastDigit === 2 || lastDigit === 3 || lastDigit === 4) {
      return `${count} товари`;
    }
  }
  return `${count} товарів`;
};

export const getProposalsCount = (count: number) => {
  const lastDigit = getLastDigit(count);
  return `Знайдено в ${lastDigit === 1 ? '1 магазині' : `${count} магазинах`}`;
};
```

ДОДАТОК Б

ВІДГУК
керівника економічного розділу
на кваліфікаційну роботу бакалавра
на тему:
«Розробка веб-орієнтованого програмного додатку пошуку продуктових товарів та послуг з використанням фреймворків Node.js, React.js»
студентки групи 121-18-2 Бондаренко Діани Олександрівни

Керівник економічного розділу
доцент каф. ПЕП та ПУ, к.е.н **Л. В. Касьяненко**

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	**Опис**
Пояснювальні документи	
DiplomaBondarenko.doc	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
DiplomaBondarenko.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF.
Програма	
diplomaBondarenko.zip	Архів. Містить коди програми.
Презентація	
DiplomaPresentationBondarenko.pptx	Презентація кваліфікаційної роботи.