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 *Бичкова Іллі Олеговича*
(ПІБ)

академічної групи *122-18-1*
(шифр)

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

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

на тему: *Розробка веб-орієнтованої системи з продажу товарів з використанням 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 *122-18-1* *Бичкова Іллі Олеговича*
(група) (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка веб-орієнтованої системи з продажу товарів з використанням React.js*

затверджена наказом ректора НТУ «ДП» від «07» червня 2022 р. № 268-с

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів виробничої практики 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 _____ *Бичков І.О.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2022 р.*

Термін подання кваліфікаційної роботи до ЕК: *11.06.2022 р.*

РЕФЕРАТ

Пояснювальна записка: 82 с., 15 рис., 3 дод., 12 джерел.

Об'єкт розробки: веб-система функціонування інтернет-магазину товарів.

Мета кваліфікаційної роботи: розробка веб-орієнтованої системи з продажу товарів з використанням React.js.

У вступі розглядається аналіз та сучасний стан проблеми, конкретизується мета кваліфікаційної роботи та галузь її застосування, наведено обґрунтування актуальності теми та уточнюється постановка завдання.

У першому розділі проведено аналіз предметної галузі, визначено актуальність завдання та призначення розробки, розроблена постановка завдання, задані вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі виконано аналіз існуючих рішень, обрано вибір платформи для розробки, виконано проектування і розробка програми, наведено опис алгоритму і структури функціонування системи, визначені вхідні і вихідні дані, наведені характеристики складу параметрів технічних засобів, описаний виклик та завантаження застосунку, описана робота програми.

В економічному розділі визначено трудомісткість розробленої інформаційної підсистеми, проведений підрахунок вартості роботи по створенню застосунку та розраховано час на його створення.

Практичне значення полягає у створенні системи, що забезпечує відвідувачам сайту доступ до каталогу товарів за категоріями, можливість редагування заказу та перегляду особистої інформації, що забезпечує комфортне користування інтернет-магазином . Адміністративна панель магазину дає можливість адміністрації магазину надавати актуальну інформацію щодо наявності та вартості товару.

Актуальність інтернет-магазину визначається великим попитом на онлайн послуги, що оптимізують та спрощують обслуговування клієнтів, сприяють створенню позитивного іміджу компанії.

Список ключових слів: ІНТЕРНЕТ-МАГАЗИН, АДМІНІСТРАТИВНА ПАНЕЛЬ, САЙТ, ВЕБ-ЗАСТОСУНОК, ФРОНТЕНД, БЕКЕНД, БРАУЗЕР.

ABSTRACT

Explanatory note: 82 p., 15 figs, 3 apps, 12 sources.

Object of development: web-system of functioning of the online shop.

The purpose of the diploma project: development of the web-oriented e-commerce system using React.js

The introduction considers the analysis and the current state of the problem, specifies the purpose of the qualification work and the scope of its application, provides a justification for the relevance of the topic and clarifies the problem.

In the first section the analysis of the subject area is carried out, the urgency of the task and purpose of development is defined, the statement of the task is developed, requirements to software realization, technologies and software are set.

The second section analyzes the existing solutions, selects the platform for development, performs design and development of the program, describes the algorithm and structure of the system, determines the input and output data, provides characteristics of the parameters of technical means, describes the call and application load, describes the program .

The economic section determines the complexity of the developed information subsystem, calculates the cost of work to create an application and calculates the time for its creation.

The practical significance is to create a system that provides site visitors with access to the catalog of products by category, the ability to edit the order and view personal information, which provides comfortable use of the online store. The administrative panel of the store allows the store administration to provide up-to-date information on the availability and cost of goods.

The relevance of the online store is determined by the high demand for online services that optimize and simplify customer service, contribute to the creation of a positive image of the company.

Keywords: ONLINE STORE, ADMINISTRATIVE PANEL, WEBSITE, WEB APP, FRONTEND, BACKEND, BROWSER.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД - база даних;

ПК - персональний комп’ютер;

API - Application Programming Interface;

CMS - Content Management System;

CSS - Cascading Style Sheets;

DOM - Document Object Model;

IT - Information Technology;

FE - Frontend;

BE - Backend;

JS - JavaScript;

MVC - Model – View – Controller;

HTML - Hypertext Markup Language.

ВСТУП

З початку існування мережі інтернет, дедалі більше комерційних діячів залучують її можливості для автоматизації процесів та спрощення доступу до сервісів.

Тому не виключенням стали й онлайн-маркетплейси. Це явище кардинально змінило принципи торгівлі так, що на теперішній час існує левова частка підприємців, які зовсім не мають фізичної реалізації свого бізнесу. Однак, умови для розміщення торгівельної точки в інтернеті мають схожі засоби для існування зі своїми попередниками. Так, тепер замість того, щоб орендувати приміщення чи точку на ринку, потрібно сплачувати за хостинг, а замість виставочної зали чи прилавку – власне сам сайт-додаток. Задля зручності користувачів цей сайт повинен бути добре оптимізованим як з точки зору швидкодії, яку забезпечують софтвер-інженери, так і з точки зору привабливості, над якою працюють дизайнери. Він має містити максимально повну інформацію про продукт, яка структурована у зручному для користувача інтерфейсі. Тому дизайн повинен бути не тільки привабливим ззовні, але також із опрацьованим психологічним аспектом, щоб із найбільшою долею ймовірності довести клієнта до покупки. А щоб максимально збільшити ці шанси, фронт та бекенд інженери зменшують час очікування завантаження контенту.

Продуктом даної кваліфікаційної роботи є створення e-commerce веб-застосунку, яка надає доступ до швидкого та зручного перегляду та придбання товарів споживачами по всьому світу.

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

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальні відомості з предметної області

На даний момент в інтернеті є багато сторінок для торгівлі, розважальної діяльності, муніципальних сервісів и т.д. Існують сайти від звичайних лендінг-сторінок до масивних систем. Всіх їх можна розділити на односторінкові та багатосторінкові.

Односторінкові зазвичай створюються із звичайними демонстраційними цілями: для портфоліо, сайту-візитки, чи просто задля розміщення невеликого контенту з посиланням на інші сервіси. Багатосторінковими є онлайн-маркетплейси, різноманітні системи для навчання, бухгалтерії і т.д. Ці сторінки містять багато контенту, який потрібно розмістити на окремих частинах. Також, односторінкові сайти вантажать усю інформацію одразу і не мають інтеграції з окремою базою даних, чого не скажеш про багатосторінкові, які довантажують потрібну інформацію по мірі запитів користувача, взаємодіючий з бекендом і базою даних.

Зазвичай статичні сайти створюються на чистому HTML з використанням Cascading Style Sheets для зовнішньої привабливості, та фреймворків з шаблонами компонентів. Сайти з використанням тільки цих технологій мають свої плюси та мінуси.

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 на інші частини сайту;

– невелике число використовуваних програмних компонентів утруднює несанкціоноване втручання до такої системи.

Такий метод створення сайтів у наш час вважається застарілим та майже не використовується, бо має певні слабкі сторони:

– складно внести зміни в структуру і зовнішній вигляд сайту, бо нема розподілення на «компоненти» і якщо змінюється стиль якогось компоненту у одному місці, треба змінювати це в усіх інших сторінках, що містить система;

– неможливо використовувати такі фішки, 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.

Коли користувач заходить на сайт, запит на запитувану інформацію відсилається до бази даних, відповідь вставляється в шаблон, утворюючи нову сторінку сайту і надсилається користувачеві, де відображається на локальній машині. Таким чином, якщо потрібно оновити контент або інформацію 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.

Одним із недоліків такого типу сайтів є більш складна архітектура, яка потребує використання різноманітних фреймворків. Але правильний старт проекту із таким підходом забезпечить кращу масштабованість та більший простір для збільшення сайту.

У процесі планування проекту виникають дилеми з вибору того чи іншого засобу для відповідної задачі. У даній кваліфікаційній роботі, серед інших альтернатив для зберігання інформації про продукт, а також для авторизації користувача, було обрано сервіс Firebase. Це платформа розробки мобільних додатків з величезним функціоналом. Починалася вона як стартап, а сьогодні її використовують при розробці кращих кроссплатформенних додатків. Головне достоїнство платформи в тому, що вона дозволяє розробнику не відволікатися на створення бекенд, тобто прихованої від користувача програмної частини проекту, наприклад, серверного коду. І це спрощує і прискорює створення мобільних додатків, дає можливість повністю зосередитися саме на UX / UI, тобто, на призначеному для користувача інтерфейсі і досвіді. Задля доступу для функціоналу є зручний API з багатьма можливостями. Також перевагою є незаперечно краща реалізація швидкодії, на відміну від саморобного “велосипеду”, яка постійно вдосконалюється інженерами Google.

Задля взаємодії із цією платформою, та для розробки фронтенд частини сайту було використано мову програмування JavaScript, за допомогою бібліотеки React JS.

React - одна з бібліотек JavaScript з відкритим кодом. Він використовується для побудови інтерактивних інтерфейсів користувача. Це ефективна, декларативна та гнучка бібліотека. Він стосується компонента V, тобто перегляду модельного перегляду-контролера (MVC). Це не цілі рамки, а лише бібліотека прифронтових сторін. Це дозволяє створювати або створювати складні інтерфейси користувача з використанням ізольованих і невеликих фрагментів коду, відомих як компоненти. Основна перевага компонентів полягає в тому, що зміна будь-якого одного компонента не впливає на всю програму.

Як засіб стейт-менеджменту було використано патерн Redux. Це бібліотека JavaScript з відкритим кодом для управління станом програми. Redux зазвичай використовується з бібліотеками, такими як Angular або React, для створення інтерфейсів користувача. Керувати станом кожного компонента в додатку стає важко, коли розмір програми стає надзвичайно великим. Redux допомагає в оновленні та підтримці стану кожного компонента в додатку.

Для оформлення UI використано бібліотеку Bootstrap. Це інструмент з відкритим кодом та дивовижний фреймворк для веб-розробок, що містить ідеальну суміш шаблонів дизайну на основі CSS та HTML, які використовуються для форм, типографіки, навігацій, кнопок та інших компонентів інтерфейсу, а також опціональних розширень JavaScript.

1.2. Призначення розробки та область застосування

Як об'єкт розробки інформаційної системи для відображення, продажу та адміністрування інтернет-магазину музикальних електронних пристроїв розглядається веб-додаток, в якому користувач може ознайомитися з асортиментом торгівельної марки “Blast Wave Electronics”, її історією, зв’язатися з виробником, та придбати продукцію. Сервіс містить сторінку з каталогом актуальних моделей товару, можливість фільтрації за типом, сортування. Також є сторінки із детальним описом кожного продукту та сторінка кошика. Адміністративна функція надає можливість до редагування інформації про товари, додання нових та видалення непотрібних.

Даних додаток заточений під торгівлю невеликим об’ємом товарів, що зумовлено специфікою компанії та продукції, яка являє собою кастомні педалі ефектів, що використовуються музикантами для модуляції, зміни гучності, акустико-частотних характеристик своїх інструментів.

Фронтенд частина для має в собі найнеобхідніший функціонал, який надає можливість користувачеві необхідний комфортний та зручний досвід від початку знайомства з асортиментом до моменту придбання.

1.3. Підстава для розробки

Підставами для розробки «виконання кваліфікаційної роботи» є:

- освітня програма 122 «Комп’ютерні науки»;
- навчальний план та графік навчального процесу;
- наказ ректора Національного технічного університету «Дніпровська політехніка» № 268-с від 18.05.2022р.;
- завдання на кваліфікаційну роботу на тему «Розробка веб-орієнтованої інформаційної системи з продажу товарів з використанням React.js».

1.4. Постановка завдання

Завданням цього кваліфікаційної роботи є розробка веб-орієнтованої інформаційної системи з продажу товарів. Програмне забезпечення призначене для надання універсального інструменту для відображення контенту та базового адміністрування магазину.

Програма повинна реалізувати наступні функції:

- Формування web-сторінок з використанням контенту з бекенду.
- Оформлення замовлень з даного магазину.
- Відображення контенту на мобільних та десктопних девайсах.
- Інформацію для зв’язку із компанією.

Для досягнення поставленої мети необхідно:

- вивчити предметну область розв'язуваної задачі;
- створити алгоритм для реалізації поставленого завдання;
- налаштувати платформу Firebase та клієнтську частину.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

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

– надання віддаленого доступу до застосунку через веб-браузер на комп'ютері користувача;

– зчитування вхідних даних з пристроїв, хмарних сервісів, за адресними посиланнями;

– зберігання даних системи у платформі Firebase.

Для виконання перерахованих вище функцій у застосунку повинні бути реалізовані:

– можливість отримати доступ до програми через веб-браузер;

– наявність типової конфігурації, що забезпечує можливість швидкого введення застосунку в експлуатацію;

– програмно-апаратна переносимість.

1.5.2. Вимоги до інформаційної безпеки

Для уникнення некоректної роботи програми необхідно реалізувати:

– семантичний та синтаксичний контроль вхідних даних;

– обробку виняткових ситуацій;

– виведення повідомлень про помилки;

– можливість повторного введення даних;

– можливість безперервної роботи протягом не менше 120 годин (5 діб);

– платформну незалежність.

1.5.3. Вимоги до складу та параметрів технічних засобів

Для нормального функціонування даного застосунку необхідно, щоб персональний комп’ютер, на якому, буде функціонувати інформаційної система, відповідала наступним вимогам:

– Intel Pentium 4/Athlon 64 або пізнішої версії з підтримкою SSE2
– доступ до мережі Internet;
– не менше 4 GB оперативної пам'яті;
– не менше 10 GB вільного місця на жорсткому диску;
– клавіатура;
– маніпулятор "миша".

Наведені вище технічні характеристики є рекомендованими, тобто при наявності технічних засобів не нижче зазначених, розроблений програмний застосунок буде функціонувати відповідно до вимог щодо надійності, швидкості обробки даних і безпеки, 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-орієнтована система, відповідало наступним вимогам:

✓ веб-браузер Google Chrome / Opera / Safari/ Firefox / Microsoft Edge (офіційно підтримуваних версій на даний момент).

Застосунок має бути реалізовано на мові програмування JavaScript з використанням бібліотек React JS та Redux, CSS-бібліотеки Bootstrap та API наданих попередньо налаштованою платформою Firebase.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Функціональне призначення системи

Метою даної роботи була розробка e-commerce веб-застосунку з можливістю перегляду та адміністрування продуктів торгівельної марки “Blast Wave Electronics”

Призначення застосунку:

– демонстрація асортименту товарів;

– налаштування повного шляху від перегляду до покупки товару.

Розроблена система має наступний функціонал:

– відображення сторінок з використанням інформації з серверу

– доступ до адмін-панелі

– можливість переглянути контакти для аналогової комунікації

– можливість створити замовлення товару

Для досягнення поставленої задачі розроблене програмне забезпечення підтримує виконання таких операцій:

Для цього система має можливість на здійснення наступних операцій:

– надання віддаленого доступу до застосунку через веб-браузер на комп'ютері або іншому девайсі користувача;

– можливість користування усіма функціями на будь якому девайсі через браузер (із підключенням до мережі інтернет);

– доступ до серверу, отримання даних з нього;

– формування замовлення та відправка даних менеджеру через Телеграм

– редагування товарів через адмін-панель.

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 інтернет-магазину математичні методи не використовувалися.

2.3. Опис використаних технологій та мов програмування

Дана система реалізована на мові для програмування веб-застосунків – JavaScript, за допомогою фреймворку React JS, утіл-бібліотеки Redux Toolkit, UI-бібліотеки Bootstrap та сервісу Firebase.

Архітектура основана на компонентному підході та з використанням Redux патерну для локального стейт-менеджменту.

JavaScript – об’єктно-прототипна мова програмування, що здебільшого використовується у веб-розробці. Вона має низку властивостей об'єктно-орієнтованої мови, але завдяки концепції прототипів підтримка об'єктів в ньому відрізняється від традиційних мов ООП. Крім того, JavaScript має ряд властивостей, притаманних функціональним мовам, — функції як об'єкти першого рівня, об'єкти як списки, каррінг (currying), анонімні функції, замикання (closures) — що додає мові додаткову гнучкість.

На відміну від більшості серверних мов програмування, JS відноситься до клієнтської сторони, тобто, обробляється на стороні клієнта, а саме інтерпретатором браузера користувача.

Як уже було згадано, JS використовують в основному для роботи на фронтенд-стороні. Якщо говорити більш конкретно, JS використовується для управління вікном браузера, зміни змісту документа при доступі до DOM’у, обробки різного роду подій на сторінці.

Тобто загалом ця мова – м’язи, які керують скелетом, що є HTML та зовнішньою оболонкою (по аналогії із шкірою, обличчям і т.д.) – CSS.

У 2015 році вийшла версія ECMAScript - ES6 (ES2015). ECMAScript — стандарт мови програмування, затверджений міжнародною організацією ECMA згідно зі специфікацією ECMA-262. Найвідомішими реалізаціями стандарту є мови JavaScript, JScript та ActionScript, які широко використовується у веб-розробці.

JavaScript знайшов свого користувача та набрав популярності завдяки таким перевагам:

– підтримка скриптів усіма популярними браузерами;

– повна інтеграція з верстkою сторінок (HTML + CSS) і серверної частиною (бекенд).

– швидкість роботи і продуктивність. Щоб зекономити час та навантаження на сервер JavaScript дозволяє частково обробляти веб-сторінки на комп'ютерах користувача без запитів до сервера;

– велика кількість бібліотек та фреймворків;

– прості завдання вирішуються швидко, а для складних є готові шаблони вирішення (патерни);

– зручність для користувача інтерфейсів

– легкість освоєння.

JavaScript також має деякі недоліки (обмеження):

– немає можливості читання та завантаження файлів

– нестрога типізація і вільне трактування. Мова ігнорує явні нестиковки, але до цього також можна звикнути.

– немає підтримки віддаленого доступу. Мову не можна використовувати для мережевих додатків, тому JavaScript іноді навіть не вважають повноцінною мовою програмування.

Колись для людства можливість написати повідомлення на глиняній табличці, передати цю табличку іншій людині в сусідньому поселенні, щоб вона могла її прочитати, стала справжнім проривом і піком прогресу.

Так і у світі IT — перша веб-сторінка, яка дозволяла перейти на іншу сторінку за допомогою гіперпосилання, була просто дивом та небувалою подією, яка перевернула весь світ. Сьогодні для нас така проста сторінка здається такою ж примітивною, як і та глиняна табличка, а веб-сторінки та їх розробка стали дуже складними та дорогими.

Разом з розвитком веб-сторінок розвивалися інструменти розробки. Ще зовсім недавно найпопулярнішим фреймворком був jQuery.

Фреймворк - власне каркас, значно полегшує процес об'єднання певних компонентів під час створення коду.

jQuery, однак, не вистачало можливостей послідовної обробки даних за представленими уявленнями. Для вирішення цієї проблеми була створена ціла плеяда фреймворків, які дуже швидко стали популярними, наприклад, Backbone, Knockout та Ember. Події розвивалися швидко, і на зміну цим фреймворкам прийшли фреймворки іншого рівня: Angular, React, Vue.

Ми вже звикли бачити серед провідних JavaScript-фреймворків React, Angular і Vue.js. Звіт про стан фронтенда State of Frontend 2020 відображає результати опитування більш ніж 4500 професійних фронтенд-розробників (рис. 2.1.).

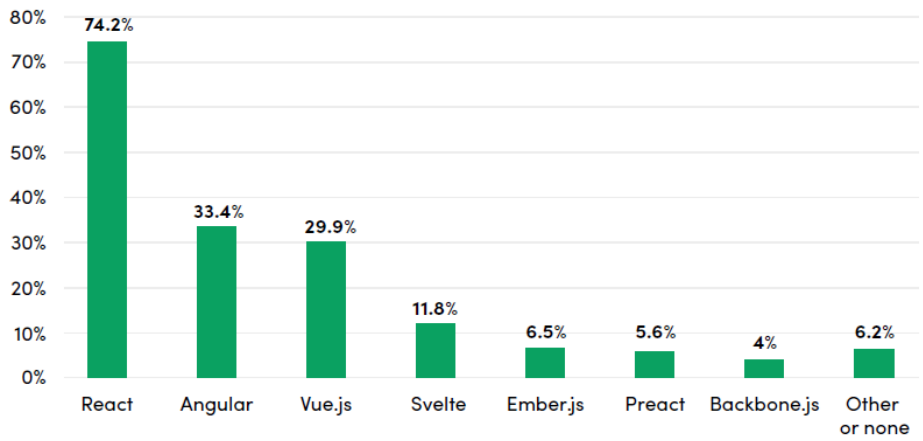

Рис. 2.1. Результати опитування

У травні 2013 року на конференції з JavaScript у США було представлено нову бібліотеку, яка змінила правила гри на ринку веб розробки — React.

Багато хто думає, що творцем або співзасновником React був Ден Абрамов, але це не так. Створив її Джордан Вовк, інженер із Facebook. Ден Абрамов створив Redux двома роками пізніше. Redux став супер популярним, а Ден Абрамов став частиною команди React.

Ще на конференції аудиторія була захоплена багатьма інноваційними функціями React, такими як віртуальний DOM, одностороння передача даних та Flux-шаблони. Після цієї конференції React набула великої популярності, яка продовжує зростати й у наші дні.

До речі, не закінчуються суперечки, чи можна назвати React фреймворком.

З одного боку, React справді лише UI бібліотека. На відміну від інших фреймворків, для повноцінної роботи розробнику необхідно додатково встановити набір інших бібліотек (наприклад, Redux, React-router, Axios тощо). У той же час, у Angular все це є з коробки. Спочатку деякі бібліотеки (наприклад, роутинг) входили до складу React, але були винесені окремо для більшої гнучкості. Коли я називаю React фреймворком, то маю на увазі не лише бібліотеку React, а й всю її екосистему.

Як би ми її не називали, але залишається фактом, що на сьогоднішній день React — це найпопулярніша JavaScript-технологія. Екосистема бібліотек та інструментів для React багата та різноманітна. Якщо ви використовуєте React, то підібрати бібліотеку, яка вирішить вашу проблему, зазвичай не важко.

React заснований на компонентах, це ще одна ключова особливість бібліотеки. Кожен компонент повертає частину призначеного для користувача інтерфейсу зі своїм станом. Об'єднуючи компоненти, програміст створює завершений інтерфейс веб-додатку.

Ще одна важлива особливість React - використання JSX. JSX — це розширення синтаксису JavaScript, яке виглядає як XML. Ви можете використовувати просту трансформацію синтаксису JSX в React. Приклад JSX (рис. 2.2) можна побачити нижче :

Рис. 2.2. Приклад JSX

До важливих особливостей також відноситься використання віртуального DOM (Virtual DOM). Virtual DOM створений, щоб постійне оновлення DOM стало більш продуктивним. На відміну від звичайного та Shadow DOM, Virtual DOM не офіційна специфікація, а скоріше новий метод взаємодії з DOM.

Redux Toolkit – це пакет, який полегшує роботу з Redux. Він був розроблений для вирішення трьох основних проблем:

- Занадто складне настроювання сховища (store)
- Для того щоб змусити Redux робити щось корисне, доводиться використовувати додаткові пакети
- Занадто багато шаблонного коду (boilerplate)

Redux Toolkit надає інструменти для налаштування сховища та виконання найпоширеніших операцій, а також містить корисні утиліти, що дозволяють спростити код.

Будь-який фронтендер або верстальник рано чи пізно приходить до необхідності створення свого невеличкого фреймворку. Складається він зазвичай з тих правил і функцій, які доводиться повторювати в кожному проекті. Зібравши їх в одній бібліотеці, і починаючи роботу над наступним проектом, досить буде просто підключити її і використовувати готові рішення. Це може бути сітка для колонок з контентом, стандартні правила спрайтів, відступи, заголовки тощо. У випадку, коли над одним проектом працюють кілька фронтенд-фахівців, подібні фреймворки повинні бути стандартизовані. І перевагу в такому випадку віддається загальноприйнятим рішенням. Тут ми опиняємося перед вибором: який фреймворк використовувати?

Що таке Bootstrap? Створили його в катівнях компанії Twitter. Спочатку він використовувався для власних продуктів і називався «Twitter Bootstrap», а пізніше був випущений на волю. За це у нього забрали слово Twitter з назви. Bootstrap — це CSS/HTML-фреймворк для створення сайтів. Іншими словами, це набір інструментів для верстки. У ньому є ряд переваг, завдяки яким BS вважається найпопулярнішим серед собі подібних. Переваги Bootstrap:

- Швидкість роботи — завдяки безлічі готових елементів верстка з Bootstrap займає значно менше часу;

- Масштабованість — додавання нових елементів не порушує загальну структуру документів;

- Легко налаштовувати — редагування стилів проводиться шляхом створення нових css-правил, які виконуються замість стандартних. При цьому не потрібно використовувати атрибути типу “! important”;

- Велика кількість шаблонів — про шаблони Bootstrap я напишу далі;

- Величезне співтовариство розробників;

- Широка сфера застосування — Bootstrap використовується в створенні тем для практично будь-яких CMS (OpenCart, Prestashop, Magento, Joomla, Bitrix, WordPress і будь-які інші), в тому числі для односторінкових додатків.

Особливою популярністю користується Bootstrap для створення одностроріНкових або лендингів (landing page).

Firebase – це платформа для розробки мобільних і веб-додатків, що дає змогу швидко створювати високоякісні додатки, залучати лояльних користувачів і збільшувати прибутки. До платформи включено комплекс інтегрованих функцій, які можна поєднувати й суміщати, зокрема мобільний сервер, засоби аналітики, а також інструменти вдосконалення й монетизації для максимальної успішності додатків.

2.4. Опис структури системи та алгоритмів її функціонування

На етапі визначення архітектури, була обміркована структура вхідних та вихідних даних, основна мета застосунку, його призначення, вид користувачів та особливості цільових девайсів задля спрощення взаємодії із системою.

Структура сайту – це описання маршруту користувача, за яким він мандрує серед інтерфейсу. Чим простіше розроблений цей шлях – тим більше буде зацікавленість клієнта. Також дуже зручно, коли з будь-якого місця ти можеш повернутися назад чи вийти з додатку.

Структура інтернет-магазину складається з таких частин:

- Каталог асортименту товарів
- Сторінка з деталями товару;
- сторінка профіля з кошиком;
- контакти з формою зворотного зв'язку.
- Сторінка з інформацією про компанію

Існують універсальні правила, дотримання яких дозволяє створити комфортні умови для здійснення покупок клієнтами та збільшити конверсію магазину. Успішний інтернет-магазин включає в себе потрібний функціонал, привабливий зовнішній вигляд (дизайн) і маркетинг-мікс (взаємозв'язок маркетингових інструментів).

Головним фокусом кінцевого користувача має бути саме асортимент товарів. Туди користувач потрапляє одразу після переходу на дефолтний роут застосунку. Там він може переглянути весь асортимент товарів одразу, відфільтрувати його та відсортувати.

Спільними для усіх сторінок є хедер (header) та футер (footer). Це два компоненти-шаблони які містять у собі інші необхідні компоненти. У даному застосунку хедер має кнопки для навігації по всій системі, логотип та кнопку профілю, клік по котрій, в залежності від актуального стану (користувач вже здійснив логін до системи чи ні), викликає навігацію до профілю або ж відкриває модальне вікно з авторизацією.

При створенні сучасних додатків останнім часом використовують найпопулярніші шаблони проектування для створення архітектури додатку, такі як, наприклад, MVC.

Model-View-Controller (MVC, «Модель-Представлення-Контролер») - схема поділу даних програми, призначеного для користувача інтерфейсу і керуючої логіки на три окремих компоненти: модель, представлення і контролер - таким чином, що модифікація кожного компонента може здійснюватися незалежно. Він вирішує проблему поділу логіки обробки запиту користувача і логіки подання інформації (рис. 2.3.).

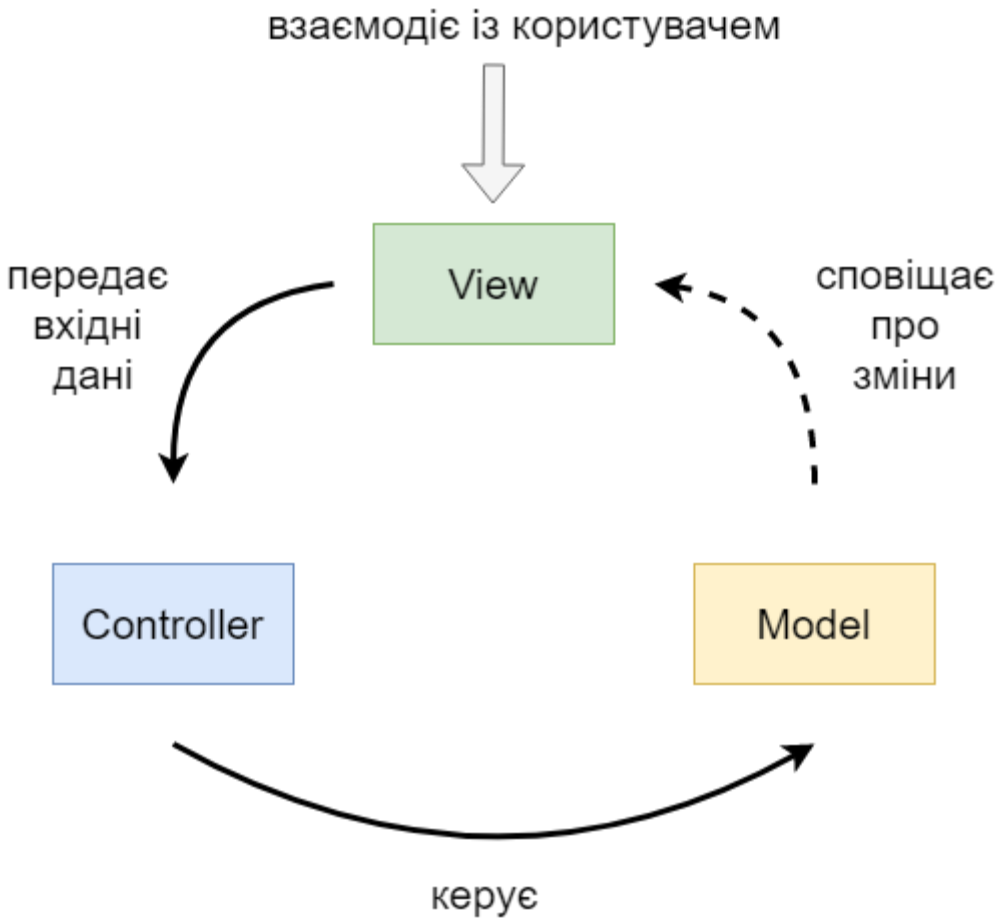

Рис. 2.3. Схема взаємодії компонентів веб-додатку в моделі MVC

Але у цій архітектурі є свої недоліки, а саме важко розширювати функціонал, змінювати структуру додатку та перевикористовувати якісь частини сторінок, як це можна зробити з компонентним підходом по проектування.

Та саме у React використовується так званий компонентний підхід, який було обрано для цього проекту. У React немає контролерів, в'юшок, моделей, шаблонів і т.д. - все є компонент. Компоненти можна і потрібно перевикористати, успадковувати один від одного, компонувати. Компонент - це свого роду будівельна одиниця, з якої збирається інтерфейс.

Можна скільки завгодно сперечатися, який фреймворк краще, але не можна не визнати очевидне - вони все базуються на компонентах. У React, в Vue, в Angular ви займаєтеся тим, що ділите своє додаток на невеликі частини і працюєте з ними як з самостійними одиницями.

Концепція компонентного підходу у фронтенді відкриває неймовірні можливості для повторного використання одного разу написаного коду.

Компоненти дозволяють розділити інтерфейс користувача на незалежні частини, придатні до повторного використання, і сприймати їх як такі, що функціонують окремо один від одного.

Концептуально компоненти є подібними до функцій JavaScript. Вони приймають довільні вхідні дані (так звані “пропси”) і повертають React-елементи, що описують те, що повинно з’явитися на екрані.

Найпростішим способом визначення компонента є написання функції JavaScript:

```
function Welcome(props) {
  return <h1>Привіт, {props.name}</h1>;
}
```

Ця функція є коректним React-компонентом, оскільки вона приймає єдиний аргумент “пропс” (скорочено від properties - властивості), який є об’єктом з даними і повертає React-елемент. Такі компоненти ми називаємо “функціональними компонентами”, оскільки вони буквально є JavaScript функціями.

Ви також можете використовувати ES6 класи, щоб визначити компонент:

```
class Welcome extends React.Component {
  render() {
    return <h1>Привіт, {this.props.name}</h1>;
  }
}
```

Два компоненти, що наведені вище, є еквівалентними з точки зору React.

Flux/Redux архітектура - це вид архітектури, яку Facebook використовує щоб працювати з React. Якщо розглядати flux в розрізі MVC, то React в свою чергу відповідає за V – View, а Flux/Redux за M – Model.

Це архітектура (рис.2.4.), яка відповідає за створення слою даних у JavaScript додатках и розробку серверної частини в веб додатках, доповнює комплексні компоненти виду View у React.

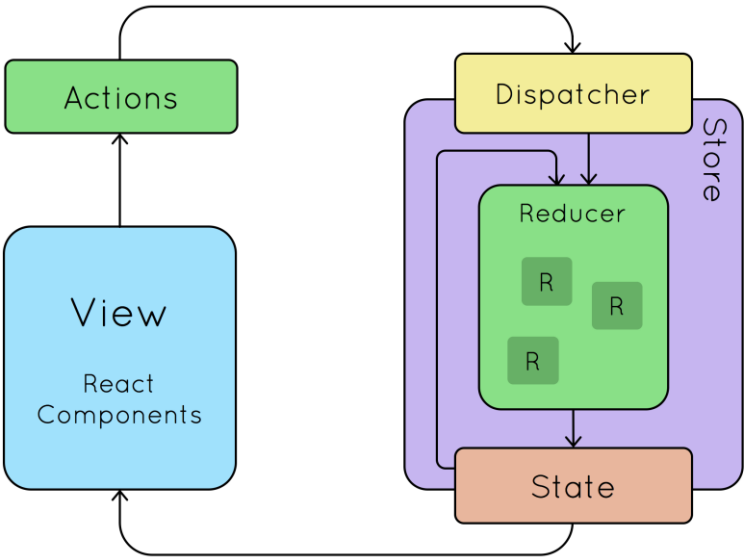

Рис. 2.4. Схема Redux архітектури

У стандартній архітектурі Redux є такі терміни:

– Дія (action);
– Створювач дії (Action creator);
– Функція-диспетчер (Dispatching Function)
– Редуктор (Reducer);
– Сховище (Store)

Типова дія виглядає так:

```
{
  type: 'SELECTED_USER',
  userId: 232
}
```

Типовий створювач дії виглядає так:

```
function addItem(t) {
```

```
  return {
    type: 'SELECTED_USER',
    title: t
  }
}
```

Типовий диспетчер виглядає так:

```
import { dispatch } from 'react-redux';
dispatch(addItem('Milk'));
```

Типовий редуктор має такий вигляд:

```
const reducer = (state = [], action) => {
  switch (action.type) {
    case 'ADD_ITEM':
      return state.concat([{ title: action.title }])
    case 'REMOVE_ITEM':
      return state.map((item, index) =>
        action.index === index
          ? { title: item.title }
          : item
      default:
        return state
  }
}
```

Типове сховище має такий вигляд:

```
import { createStore } from 'redux'
import reducer from './reducers'
let store = createStore(reducer)
```

Немалозначним фактором є грамотна файлова структура проекту задля зручності розробки: знаходження потрібного файлу, оптимального шляху імпорту і т.д.

В даному випадку весь проект знаходиться у приватному репозиторії на GitHub задля зручного доступу розробника з різних девайсів та задля зручності коригування внесених змін, якщо є така необхідність.

Файли даного застосунку(рис. 2.5.) поділені на компоненти, що використовуються більше одного разу у додатку та логічно розподілені по папкам згідно реальній структурі сайту. Також є окрема папка та файли для Redux сховища та файл-помічник для роботи з Firebase.

Рис. 2.5. Структура файлової системи

2.5. Обґрунтування та організація вхідних та вихідних даних програми

Даний застосунок отримує вхідні дані через завантаження інформації з сервісу Firebase, та розподіляє її по Redux сховищу.

Вхідні дані:

- інформація про асортимент;
- інформація про стан авторизації користувача

Вихідні дані:

- web-застосунок магазину;
- замовлення користувача;

Дані програми організовані в локальні сховища даних – для товарів та для стану користувача. Локальні сховища організують між собою одне велике сховище даних, до якого підключені усі основні компоненти програмного додатку. Компоненти отримують дані з глобального сховища, та реагують на кожну зміну даних. Дані зі сховищ потрапляють за допомогою спеціального методу, який називається useSelector, прямо до скоупу потрібного компонента.

2.6. Опис роботи розробленої системи

2.6.1. Використані технічні засоби

Для серверних технічних засобів рекомендована конфігурація, що забезпечує цілодобову роботу програми з резервуванням даних:

- процесор Intel Core i7-1165G7 (2.8 - 4.7 ГГц)
- модулі пам’яті 32GB DDR4_x000D_ - 3200 МГц
- SSD диск 512GB
- доступ до мережі Internet;
- маніпулятор "миша";

– клавіатура.

Наведені вище технічні характеристики є рекомендованими, тобто при наявності технічних засобів не нижче зазначених, розроблений програмний виріб буде функціонувати відповідно до вимог щодо надійності, швидкості обробки даних і безпеки, висунутими замовником.

2.6.2. Використані програмні засоби

Для програмування існує декілька найпопулярніших редакторів коду. Деякі призначені для одної мови, а деякі підтримують і декілька. Кожен має свої переваги та недоліки, тому програміст не обмежений у виборі.

Ці програми називаються IDE, а дана кваліфікаційна робота розроблялась за допомогою Visual Studio Code та тестувалась у браузері Google Chrome (працює на V8).

Необхідними програмними та технічними засобами для клієнта є мобільний телефон, ПК або інший девайс з підключенням до Інтернету та браузером, що підтримує JavaScript.

2.6.3. 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 одне з серверного забезпечення (Apache або Nginx);

– 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.

2.6.4. Опис інтерфейсу користувача

При потраплянні на сайт, користувач одразу бачить невід’ємні компоненти кожної сторінки – хедер та футер. У хедері знаходиться логотип, навігація по розділам сайту та кнопка для авторизації. Основною сторінкою з’являється сторінка з каталогом товарів (рис 2.6.), де користувач може переглянути увесь асортимент, відфільтрувати його за категоріями та відсортувати. До сторінки з детальним описом товару та можливістю здійснити покупку можна потрапити натиснувши на будь який товар.

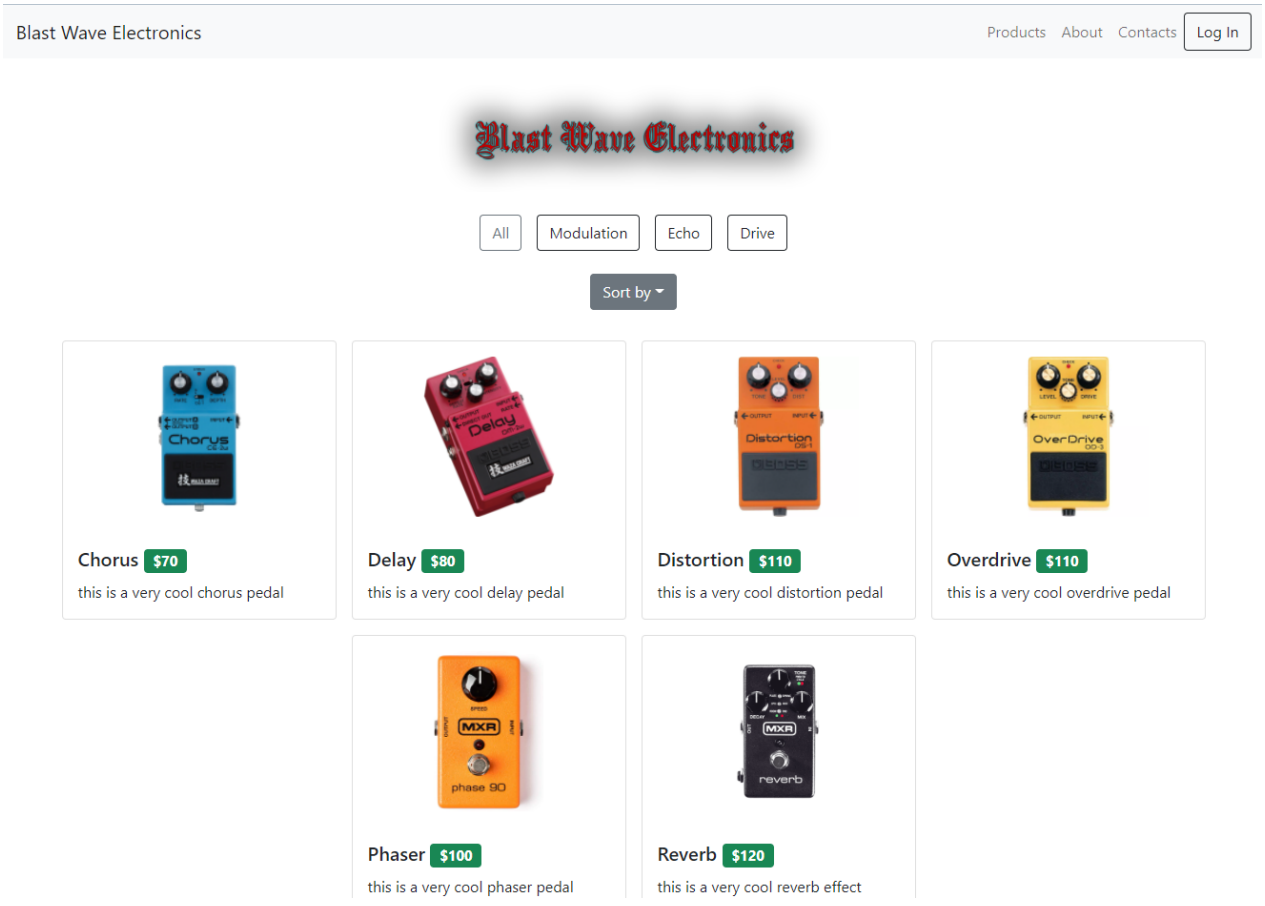

Рис. 2.6. Сторінка каталогу

Сторінка із контактами для зворотного зв’язку (рис 2.7.)

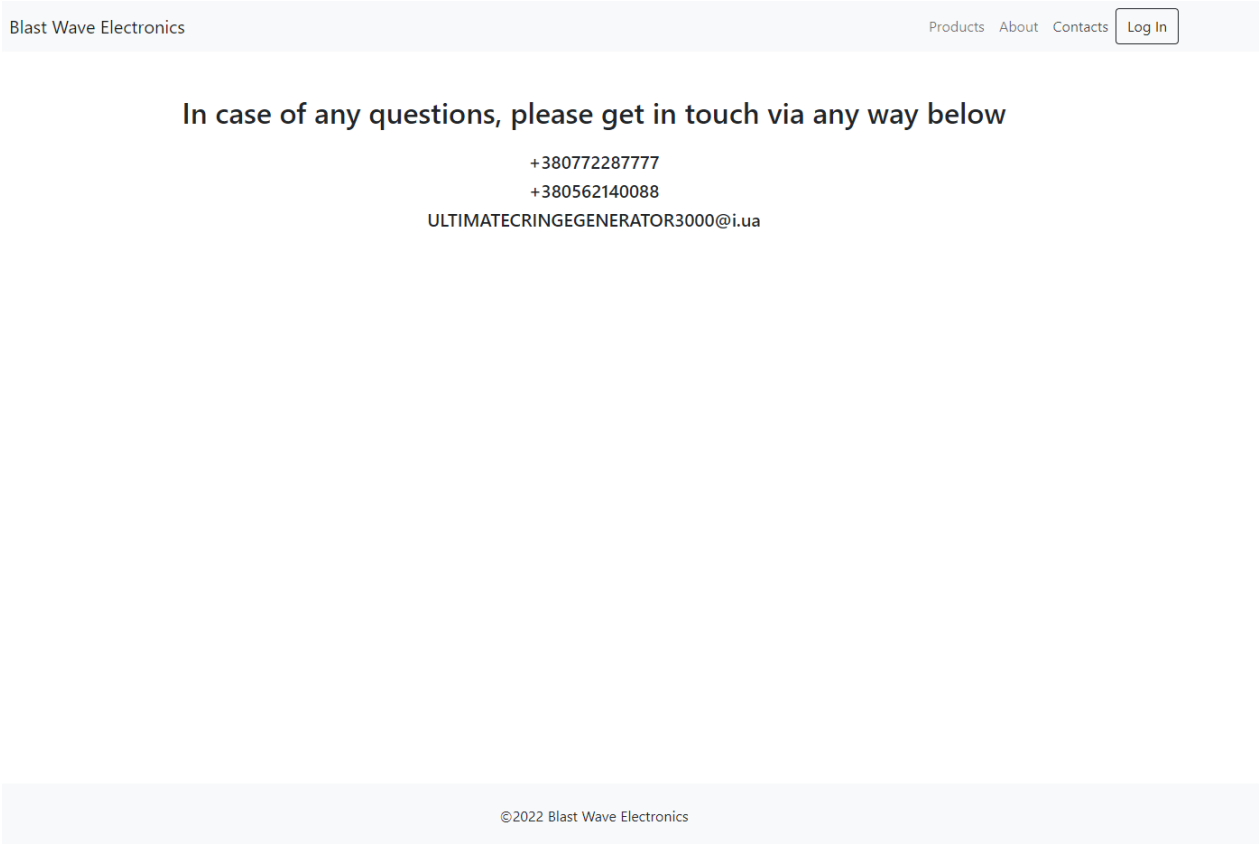

Рис. 2.7. Сторінка контактів

Сторінка з інформацією про компанію (рис 2.8.)

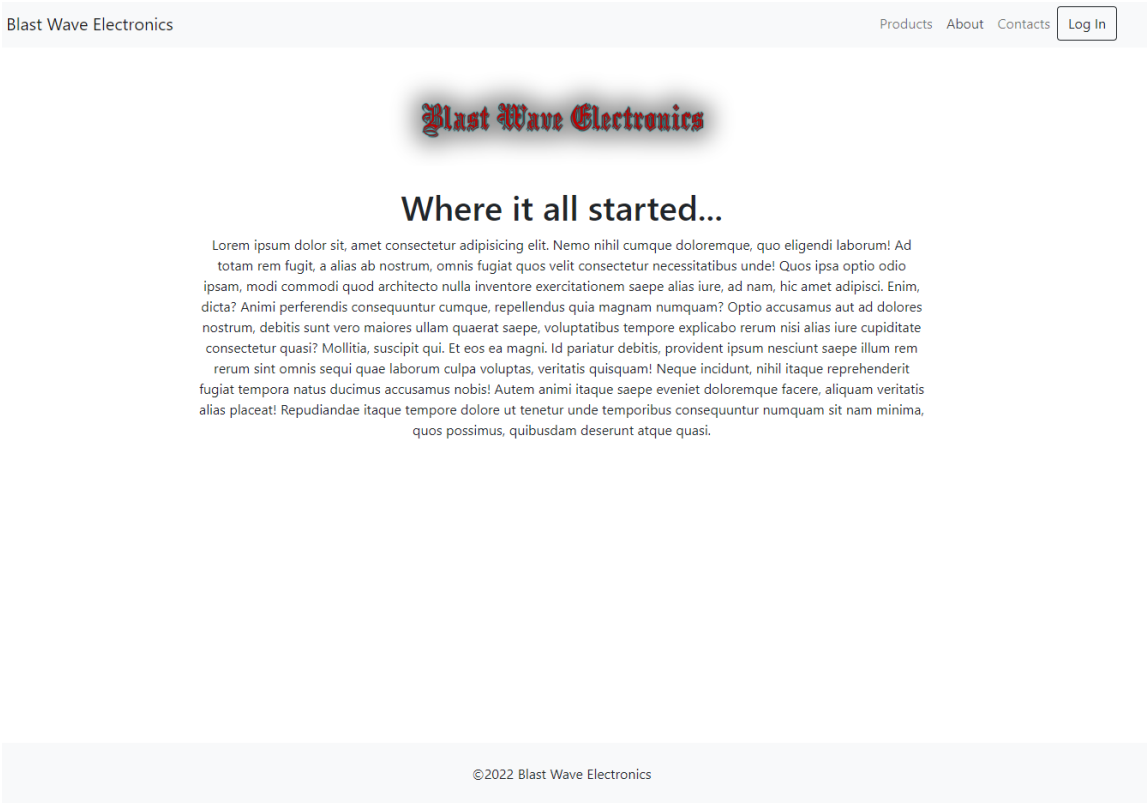

Рис. 2.8. Інформаційна сторінка

При клікі на якийсь з товарів, користувача переносить до сторінки з описом цього товару та можливістю додати його до кошика (рис 2.9.)

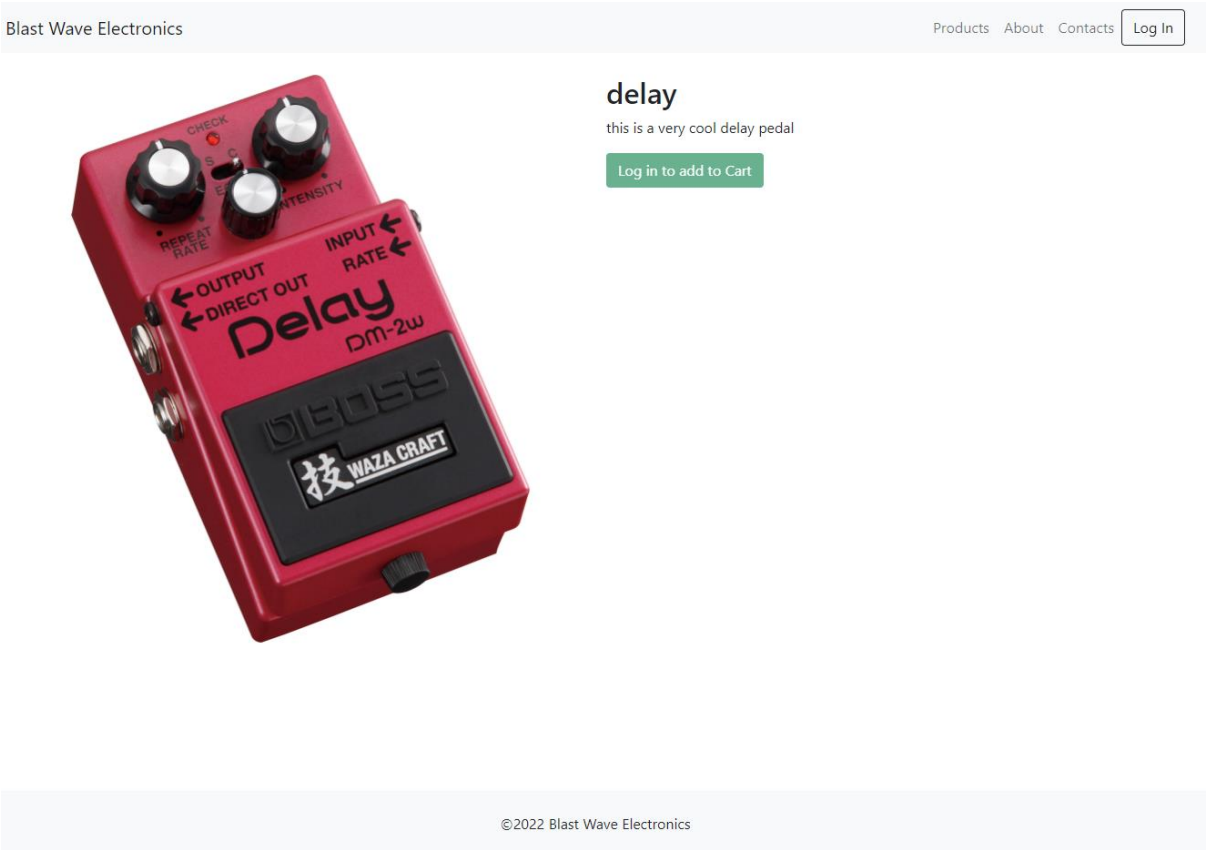

Рис. 2.9. Деталі товару

На даний момент кнопка додавання до кошику неактивна, бо користувачеві потрібно увійти до свого акаунту, або створити новий. Для цього є кнопка у хедері, при натисканні на котру з’являється таке модальне вікно (рис. 2.10.)

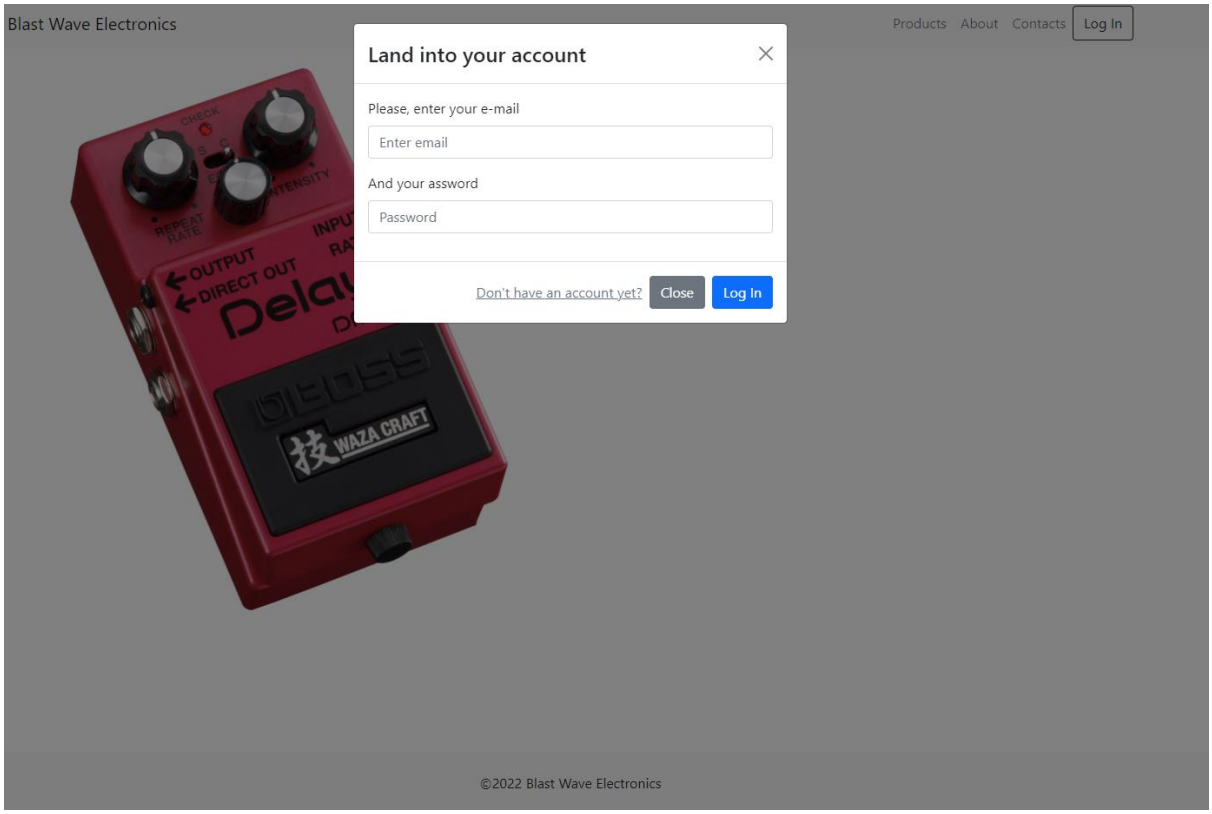

Рис. 2.10. Авторизація

Тут є поля для вводу електронної пошти та паролю. Якщо у користувача ще немає акаунту, то він може його створити натиснувши на кнопку “Don’t have an account yet?”

Вікно залишиться те саме, але тепер змінився надпис на “позитивній” кнопці, який каже про те, що натиснувши на неї буде створено новий акаунт (рис. 2.11.)

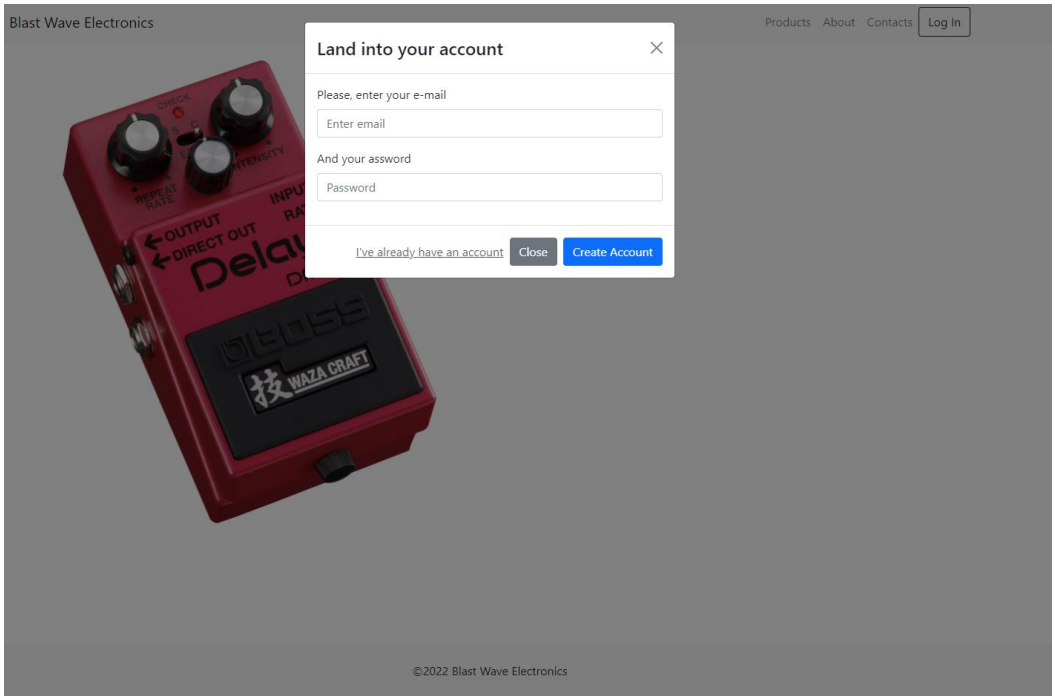

Рис. 2.11. Сторінка деталів товару

Якщо в якомусь з полів користувач не пройде перевірку на коректність введених даних, про це з’явиться повідомлення біля відповідного поля (рис. 2.12.)

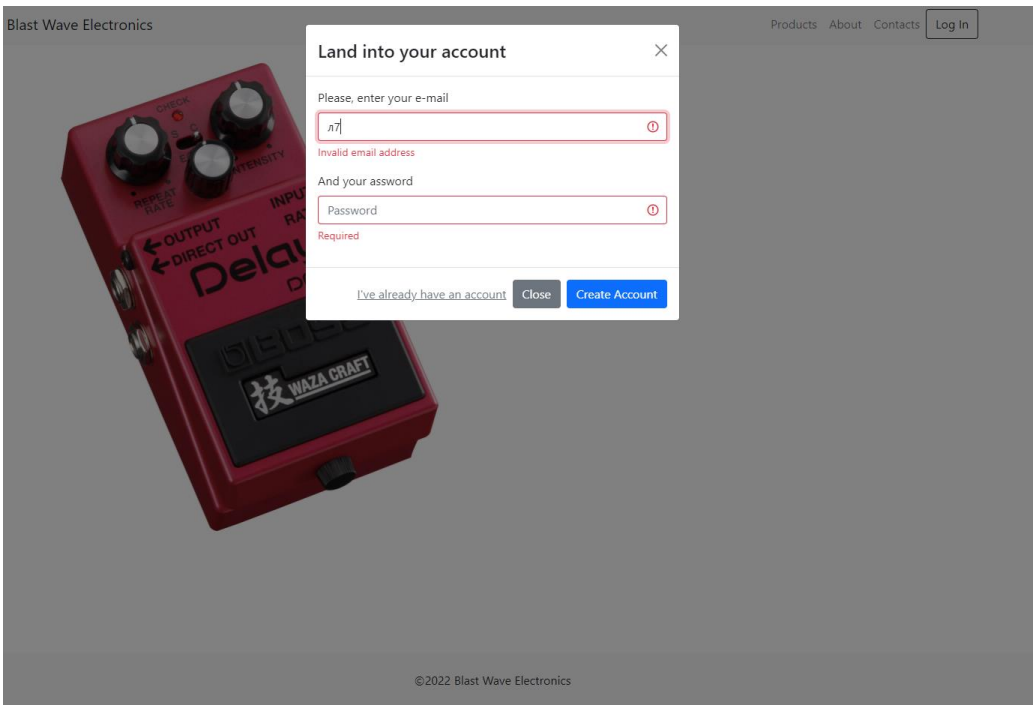

Рис. 2.12. Сторінка контактів

Якщо користувач успішно авторизувався, він потрапляє до сторінки із кошиком своїх замовлень (рис. 2.13.)

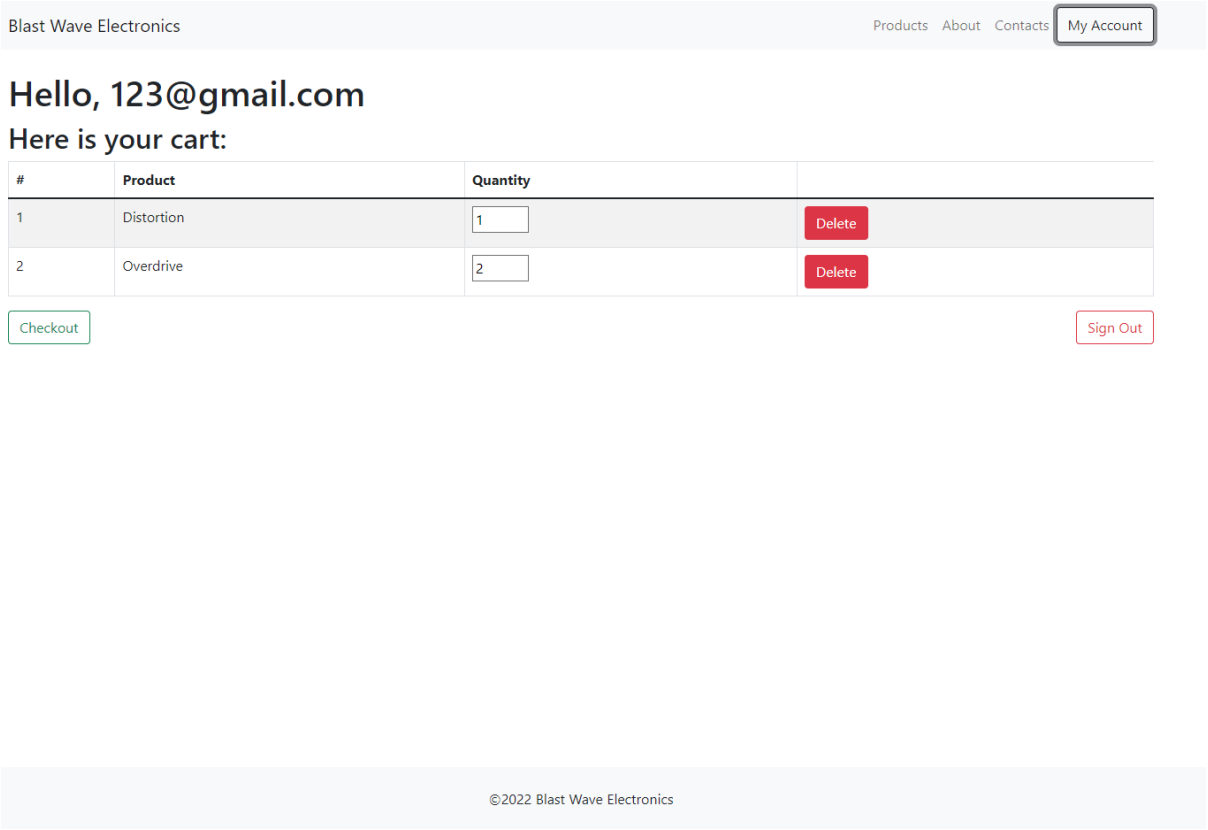

Рис. 2.13. Сторінка профілю та кошика

Тут користувач може змінити кількість педалей у своєму замовлені, видалити позицію цілком, підтвердити замовлення або вийти із свого акаунту. Щоб добавити нову позицію, потрібно вибрати її з каталогу на сторінці “Products”

Якщо замовлення схвалене, при натисканні на кнопку “Checkout” з’являється наступне модальне вікно (рис. 2.14.)

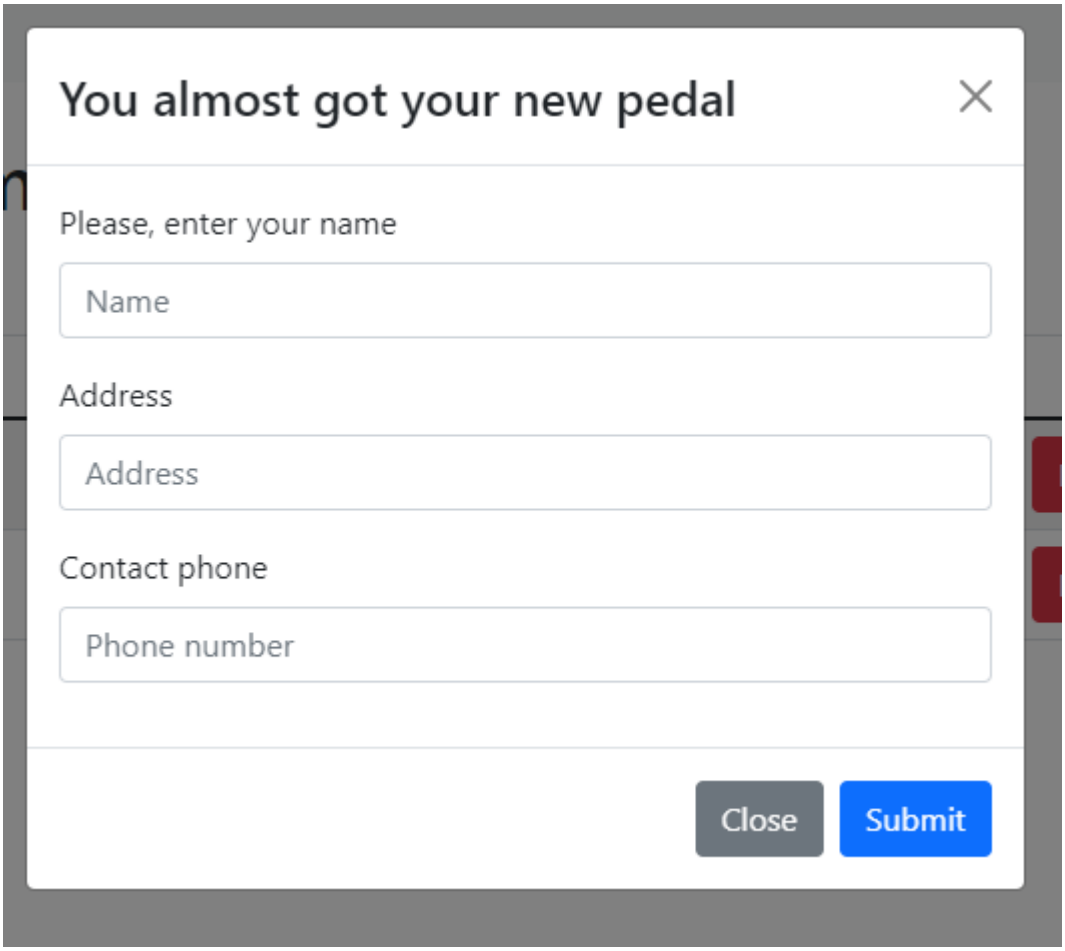

Рис. 2.14. Підтвердження замовлення

Якщо у цьому вікні користувач заповнив усі поля, та пройшов перевірку введених даних, після натискання на кнопку “Submit” його замовлення надсилається до телеграм-боту менеджеру з продаж, а користувач отримує повідомлення про успішну покупку (рис. 2.15.).

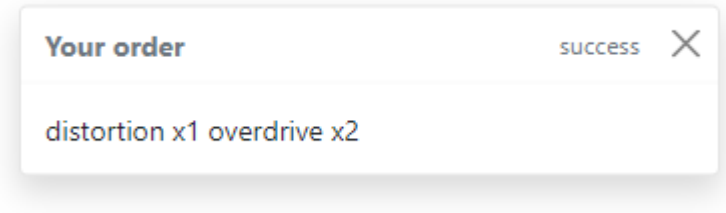

Рис. 2.15. Повідомлення про успішну покупку

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Початкові дані:

1. передбачуване число операторів програми – 1489;
2. коефіцієнт корекції програми в ході її розробки – 0.06;
3. коефіцієнт складності програми – 1.9;
4. годинна заробітна плата програміста– 136грн;

Середня годинна зарплата Junior FE developer в Україні була вирахувати виходячи з даних «Української спільноти програмістів (DOU)» [10]. Станом на кінець 2020 року зарплата Junior FE розробника простягається від 500$ до 1100$. Вирахувавши середню заробітну плату програміста маємо плату 800$ у місяць. При курсі валют НБУ на початок червня 2021 року один американський долар дорівнює 29.25грн, тому середня зарплата в гривнях дорівнює 23400 грн. При стандартному графіку (176 годин/місяць) зарплата за годину буде становити близько 136 грн.

1. коефіцієнт збільшення витрат праці в наслідок недостатнього опису задачі – 1,3;
2. коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1,4;
3. вартість машино-години ЕОМ –20 грн/год.

Вартість оренди комп’ютера на місяць 1337 грн (монітор) та 2048 грн (системний блок). Загалом на місяць оренда коштуватиме 3385 грн. При стандартному графіку (176 годин/місяць) вартість машино-години ЕОМ за годину роботи буде становити 21 грн. В цю вартість входить ремонт за гарантією та базовий комплект гарнітури (клавіатура та миша).

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

Трудомісткість розробки ПЗ можна розрахувати за формулою:

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

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

t_u - витрати праці на дослідження алгоритму рішення задачі;

t_a- витрати праці на розробку блок-схеми алгоритму;

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

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

$t_д$ - витрати праці на підготовку документації.

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

Умовне число операторів (підпрограм):

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

де q - передбачуване число операторів (1489);

C - коефіцієнт складності програми (1,9);

p - коефіцієнт корекції програми в ході її розробки (0,06).

Звідси умовне число операторів в програмі:

$$Q = 1{,}9 \cdot 1489 \cdot (1 + 0{,}06) = 2998{,}846$$

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

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

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:

$$t_u = (2998{,}846 \cdot 1{,}2) / (75 \cdot 1{,}4) = 34{,}27 \text{ людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі визначаються за формулою:

$$t_a = \frac{Q}{(20\ldots25) \cdot k} \text{ , людино-годин, (3.2)}$$

де Q – умовне число операторів програми;

k – коефіцієнт кваліфікації програміста.

Підставивши відповідні значення в формулу (3.2), отримаємо:

$$t_a = 2998{,}846 / (20 \cdot 1{,}4) = 107 \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20\ldots25)\cdot k}\ \text{, людино-годин.}$$

t_n = 2998,846 / (25 · 1,4) = 85 людино-годин.

Витрати праці на налагодження програми на ЕОМ:

- за умови автономного налагодження одного завдання:

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

$t_{отл}$= 2998,846 / (5 · 1,4) = 428,4 чел.-ч.

- за умови комплексного налагодження завдання:

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

$t^k_{отл}$= 1,5 · 428,4 = 642,6 людино-годин.

Витрати праці на підготовку документації визначаються за формулою:

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

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

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

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

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

Підставляючи відповідні значення, отримаємо:

$t_{др}$ = 2998,846 / (18 · 1,4) = 119 людино-годин.

$t_{до}$ = 0,75 · 119 = 89,25 людино-годин.

$t_{д}$ = 119 + 89,25 = 208,25 людино-годин.

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

t = 50 + 34,27 + 107 + 85 + 428,4 + 208,25 = 914,32 людино-годин.

3.2. Розрахунок витрат на створення програми

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

$$К_{ПО} = З_{ЗП} + З_{МВ}$$, грн.

Заробітна плата виконавців визначається за формулою:

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

де: t - загальна трудомісткість, людино-годин;

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

З урахуванням того, що середня годинна зарплата програміста становить 136 грн / год, отримуємо:

$$З_{ЗП} = 914{,}32 \cdot 136 = 124\ 347 \text{ грн.}$$

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

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

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

$C_{мч}$ - вартість машино-години ЕОМ, грн/год (21 грн/год).

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

$$З_{мв} = 428{,}4 \cdot 21 = 8996{,}4 \text{ грн.}$$

Звідси витрати на створення програмного продукту:

$$К_{ПО} = 208\ 625 + 16\ 332{,}75 = 133\ 343{,}4 \text{ грн.}$$

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:

$$T = 914{,}32 / 1 \cdot 176 \approx 5{,}2 \text{ міс.}$$

Висновок: даний веб-застосунок розроблено для забезпечення доступу користувачів до основних функцій магазину компанії з продажу електроніки,

Полегшення зв’язку між клієнтами та компанією, налаштування онлайн процесу покупки та, відповідно, підвищення продажу товарів.

Вартість даного програмного забезпечення близько 133 343,4 тис. грн і не вимагає додаткових витрат як при розробці програми. Очікуваний час розробки становить 914,32 годин, тобто 5,2 місяці. Цей термін пов'язаний зі значним числом операторів, і включає час на дослідження і розробку алгоритму вирішення поставленого завдання, програмування по готовому алгоритму, налагодження програми і підготовку документації.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було поставлено завдання розробити веб-орієнтовану системи з продажу товарів з використанням React.js.

Даний застосунок створений як інструмент для надання доступу до відображення контенту інтернет-магазину та його редагування через адміністративні функції. Він надає зручний та комфортний доступ до асортименту товарів завдяки зрозумілого відображення каталогу, можливості фільтрації та сортування, що забезпечує швидкий процес покупки.

Під час виконання даного проекту були виконані наступні задачі:

- вивчено предметну галузь розв'язуваної задачі;
- створено алгоритм для реалізації поставленого завдання;
- створено базу даних і клієнтську та серверну програму, що працює разом.

Розроблене програмне забезпечення дозволяє:

- Формування web-сторінок з використанням контенту з бекенду.
- Оформлення замовлень з даного магазину.
- Відображення контенту на мобільних та десктопних девайсах.
- Інформацію для зв’язку із компанією.

Застосунок реалізований з використанням фреймворку React.js, бібліотеки Redux Toolkit, та бібліотеки стилей – Bootstrap. В якості бекенду було використано сервіс Firebase.

Архітектура проекту – компонентна, яка передбачує можливість перевикористання однакових частин коду у різних місцях. В якості стейт-менеджменту використано архітектуру Redux.

Також у кваліфікаційній роботі було визначено трудомісткість розробленої системи, на базі середьої зарплати розробника проведений підрахунок вартості роботи по створенню програми, який складає 133 343,4 грн та розраховано час на створення інтернет-магазину – 914,32 людино-годин , тобто 5,2 місяці.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Адам Фрімен. Pro React 16. Перше видання, 2019. – 768 с.
2. К. Чіннатхамбі. Learning React: A Hands-On Guide to Building Web Applications Using React and Redux. Друге видання, 2018. – 304 с.
3. Марк Тіленс Томас. React в дії, 2018. – 360 с.
4. Молер, Дж. Flash 8. Керівництво Web-дизайнера; Эксмо - М., 2019. - 400 с.
5. А. Бенкс. Learning React: Functional Web Development with React and Redux 1st Edition, 2017. – 350 с.
6. К. Дуглас – Як влаштований JavaScript, 2019. - 394 с.
7. Е. Фрімен, Е. Робсон – Вивчаєм програмування на JavaScript, 2017. – 550 с.
8. Бенкс Алекс, Порселло Єва. React та Redux: функціональна веб-розробка. 2018. - 336 с.: іл. - (Серія "Бестселери O'Reilly").
9. React [Електронний ресурс] - Режим доступу: https://react.js.org .
10. Bootstrap [Електронний ресурс] – Режим доступу: https://getbootstrap.com/docs/5.2/getting-started/introduction/.
11. Firebase [Електронний ресурс] – Режим доступу: https://firebase.google.com/docs/build.
12. Firebase [Електронний ресурс] – Режим доступу: https://redux-toolkit.js.org/tutorials/overview.

ДОДАТОК А

КОД ПРОГРАМИ

Index.html - головний HTML файл

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="icon" href="./bud.png" />
    <title>Blast Wave Electronics</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
  </body>
</html>
```

Package.json - інформація про застосунок

```
{
  "name": "blast-wave-electronics",
  "version": "0.1.0",
  "homepage": "https://CaelumVallis.github.io/blast-wave-electronics",
  "private": true,
  "dependencies": {
    "@reduxjs/toolkit": "^1.8.1",
    "@testing-library/jest-dom": "^5.16.4",
    "@testing-library/react": "^13.1.1",
    "@testing-library/user-event": "^13.5.0",
    "@types/jest": "^27.5.0",
    "@types/node": "^17.0.31",
    "@types/react": "^18.0.9",
    "@types/react-dom": "^18.0.3",
    "bootstrap": "^5.1.3",
    "firebase": "^9.8.1",
    "formik": "^2.2.9",
    "node-sass": "^7.0.1",
    "react": "^18.0.0",
    "react-bootstrap": "^2.3.1",
    "react-dom": "^18.0.0",
    "react-redux": "^8.0.1",
```

```
    "react-router-dom": "^6.3.0",
    "react-scripts": "5.0.1",
    "typescript": "^4.6.4",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "predeploy": "npm run build",
    "deploy": "gh-pages -d build",
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject",
    "lint": "eslint \"src/**/*.{js,jsx,ts,tsx,html}\" && npm run check-prettier",
    "check-prettier": "./node_modules/prettier/bin-prettier.js --check
\"src/**/*.{js,jsx,ts,tsx,html}\"",
    "prepare": "husky install"
  },
  "husky": {
    "hooks": {
      "pre-commit": "npm run lint"
    }
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  },
  "devDependencies": {
    "@typescript-eslint/eslint-plugin": "^5.20.0",
    "@typescript-eslint/parser": "^5.20.0",
```

```
    "eslint": "^8.15.0",
    "eslint-config-prettier": "^8.5.0",
    "eslint-plugin-prettier": "^4.0.0",
    "eslint-plugin-react": "^7.29.4",
    "gh-pages": "^4.0.0",
    "husky": "^7.0.4",
    "prettier": "^2.6.2"
  }
}
```

Eslint.json - файл з конфігурацією eslint

```
{
  "env": {
    "browser": true,
    "es2021": true
  },
  "extends": ["eslint:recommended", "plugin:react/recommended",
"plugin:@typescript-eslint/recommended"],
  "parser": "@typescript-eslint/parser",
  "parserOptions": {
    "ecmaFeatures": {
      "jsx": true
    },
    "ecmaVersion": "latest",
    "sourceType": "module"
  },
  "plugins": ["react", "@typescript-eslint"],
  "rules": {
    "react/react-in-jsx-scope": "off",
    "react/prop-types": "off"
  }
}
```

.prettierrc - файл з конфігурацією prettier

```
{
  "semi": true,
  "tabWidth": 2,
  "printWidth": 120,
  "singleQuote": true,
  "trailingComma": "none"
}
```

.husky -> precommit - файл з прекоміт хуком

```
#!/bin/sh
. "$(dirname "$0")/_/husky.sh"

npm run lint
```

Index.js - головний файл javascript

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import { Provider } from 'react-redux';
import { store } from './store/store.js';
import App from './App.js';
import { BrowserRouter } from 'react-router-dom';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <Provider store={store}>
      <BrowserRouter>
        <App />
      </BrowserRouter>
    </Provider>
  </React.StrictMode>
);
```

App.js - головний файл застосунку

```
import './App.scss';
import { Footer } from './components/footer';
import { Header } from './components/header';
import { MainContent } from './components/main-content';

const App = () => {
  return (
    <div className="d-flex flex-column" style={{ height: '100vh' }}>
      <Header />
      <MainContent />
      <Footer />
    </div>
  );
};

export default App;
```

App.scss - головний файл стилей

```
@import '~bootstrap/scss/bootstrap';
```

Telegram.js - файл з методамиtelegram api

```
//api.telegram.org/bot{**TOKEN**}/sendMessage?chat_id=207661598&text
=qq

export const sendOrderDetailsToTelegram = (order) => {
  const markdownOrder = () => {
    return
'Order%20from%20_name_%0AAddress%3A%20_address_%0APhone%3A%20_p
hone_%0AOrder%3A%0A_order_'
      .replace('_name_', order.name)
      .replace('_address_', order.address)
      .replace('_phone_', order.phone)
      .replace('_order_', order.order.map((el) => `-
%20${el.name}%2C%20${el.quantity}%20pieces`).join('%0A'));
  };
  const url =
`https://api.telegram.org/bot5524023316:AAHmV3WnVqhxCaRVELb2VB3BPYFin
9_uU5M/sendMessage?chat_id=207661598&&parse_mode=HTML&text=${markdo
wnOrder()}`;

  fetch(url)
    .then((response) => {
      return response.json();
    })
    .then((data) => {
      if (data.ok) {
        // this.handleStatusMessage([data.ok, `Ваш заказ №${orderId} успешно
принят!`]);
        // console.log(data);
      }
      if (!data.ok) {
        console.log(data);
      }
    });
};
```

Misc.js – файл з утілитами проекту

```
export const BASE_URL = 'blast-wave-electronics';

export const toUpperCase = (s) => `${s[0].toUpperCase()}${s.slice(1)}`;
```

Firebase.js - файл з конфігурацією та методами firebase api

```
import { initializeApp } from 'firebase/app';
import { getAuth, signOut } from 'firebase/auth';
import { getDatabase, ref as databaseRef, child, get, set, ref, push, update } from 'firebase/database';
import { getStorage } from 'firebase/storage';

const firebaseConfig = {
  apiKey: 'AIzaSyDG0PmMh-3M6H_2qrxGgDPlG0riWh0knkI',
  authDomain: 'blast-wave-electronics.firebaseapp.com',
  databaseURL: 'https://blast-wave-electronics-default-rtdb.europe-west1.firebasedatabase.app',
  projectId: 'blast-wave-electronics',
  storageBucket: 'blast-wave-electronics.appspot.com',
  messagingSenderId: '272140529360',
  appId: '1:272140529360:web:189dd29ec57e97ac5838d5',
  measurementId: 'G-Z0FS91FZ40'
};

export const app = initializeApp(firebaseConfig);

export const auth = getAuth(app);

export const database = getDatabase(app);
export const dbRef = databaseRef(getDatabase());

export const storage = getStorage(app);

export const checkForAdmin = async (email) => {
  const snapshot = await get(child(dbRef, `users`));
  if (snapshot.exists()) {
    const { admin_emails } = snapshot.val();
    return admin_emails.includes(email);
  } else {
    throw new Error('No data available');
  }
};

export const signOutUser = async () => await signOut(auth);

export const addUserToDb = async (email) => {
  const usersListRef = ref(database, 'users/list');
  const newUserRef = await push(usersListRef);
```

```
  await set(newUserRef, { email });
  return newUserRef.key;
};

export const getUserKey = async (email) => {
  const usersRef = ref(database, 'users');
  const snapshot = await get(child(usersRef, 'list'));
  const usersList = await snapshot.val();
  for (const key in usersList) {
    if (usersList[key].email === email) return key;
  }
};

export const updateCart = (updatedCart, userKey) => {
  const usersRef = ref(database, 'users');
  const updates = {};
  updates[`/list/${userKey}/cart`] = updatedCart;
  update(usersRef, updates);
};

export const addToCart = async (userKey, newProduct, increaseValue) => {
  const usersRef = ref(database, 'users');

  const snapshot = await get(child(usersRef, `list/${userKey}`));
  const cart = (await snapshot.val().cart) || [];

  let updatedCart = [...cart];

  const duplicateIndex = cart.findIndex((el) => el.name ===
newProduct.name);
  if (duplicateIndex === -1) {
    updatedCart.push({ ...newProduct, quantity: 1 });
  } else {
    updatedCart[duplicateIndex].quantity = increaseValue
      ? updatedCart[duplicateIndex].quantity + 1
      : newProduct.quantity;
  }

  updateCart(updatedCart, userKey);
};

export const getUserCart = async (userKey) => {
  const usersRef = ref(database, 'users');
```

```
  const snapshot = await get(child(usersRef, `list/${userKey}/cart`));
  const userCart = await snapshot.val();
  return userCart;
};
```

Store.js - головний файл стейт менеджменту

```
import { configureStore, combineReducers } from '@reduxjs/toolkit';
import productsSlice from './productsSlice';
import userSlice from './userSlice';

const reducer = combineReducers({
  productsSlice,
  userSlice
});

export const store = configureStore({ reducer });
```

ProductsSlice.js - файл конфігурації стейта товарів

```
import { createSlice, createAsyncThunk } from '@reduxjs/toolkit';
import { child, get } from 'firebase/database';
import { getDownloadURL, ref, uploadBytes } from 'firebase/storage';
import { dbRef, storage } from '../utils/firebase';

const parseProductsList = (products) => {
  return Object.keys(products).map((el) => {
    return { name: el, ...products[el] };
  });
};

const combineWithImages = async (products) => {
  return await Promise.all(
    products.map(async (el) => {
      const imgUrl = await getDownloadURL(ref(storage, `${el.imgPath}`));
      return { ...el, imgUrl };
    })
  );
};

export const uploadImage = createAsyncThunk('products/uploadImage', async
(file, { rejectWithValue }) => {
  const imgRef = ref(storage, `${file.name}`);
  uploadBytes(imgRef, file)
    .then(() => {
```

```
      console.log('Uploaded a file!');
    })
    .catch((err) => rejectWithValue(err));
});

export const fetchProducts = createAsyncThunk(
  'products/fetchProducts',
  async (_, { rejectWithValue, dispatch, getState }) => {
    await get(child(dbRef, 'products'))
      .then(async (snapshot) => {
        if (snapshot.exists()) {
          const response = snapshot.val();

          const parsedProductsList = parseProductsList(response);
          dispatch(setProductsList(parsedProductsList));

          const {
            productsSlice: { productsList }
          } = getState();
          dispatch(setFiltersList(productsList));

          const productsWithImages = await combineWithImages(productsList);
          dispatch(setProductsList(productsWithImages));
        } else {
          throw new Error('No data available');
        }
      })
      .catch((err) => rejectWithValue(err));
  }
);

export const productsSlice = createSlice({
  name: 'products',
  initialState: {
    status: 'idle',
    error: '',
    productsList: [],
    filters: [],
    filterBy: null,
    openProduct: {}
  },
  reducers: {
    setProductsList: (state, { payload }) => {
```

```
            state.productsList = payload;
          },
          setFiltersList: (state, { payload }) => {
            state.filters = [...new Set(payload.map((product) => product.type))];
          },
          setOpenProduct: (state, { payload }) => {
            state.openProduct = state.productsList.find((product) => product.name ===
payload);
          }
        }
      export const { setProductsList, setFiltersList, setOpenProduct } =
productsSlice.actions;

      export default productsSlice.reducer;
```

UsersSlice.js - файл конфігурації стейту користувачів

```
      import { createSlice, createAsyncThunk } from '@reduxjs/toolkit';
      import { getUserCart } from '../utils/firebase';

      const defaultAccountState = {
        isLoggedIn: false,
        isAdmin: false,
        authData: {},
        cart: []
      };

      export const fetchCart = createAsyncThunk('users/fetchUsers', async (userKey,
{ dispatch }) => {
        const cart = await getUserCart(userKey);
        dispatch(setUserCart(cart));
      });

      export const userSlice = createSlice({
        name: 'user',
        initialState: { ...defaultAccountState },
        reducers: {
          setUser: (state, { payload }) => {
            state.isAdmin = payload.isAdmin;
            state.authData = payload;
            state.isLoggedIn = true;
          },
          setUserCart: (state, { payload }) => {
            state.cart = payload;
```

```
    },
    signOut: (state) => {
      state.isLoggedIn = false;
      state.isAdmin = false;
      state.authData = {};
    }
  }
});

export const { setUser, signOut, setUserCart } = userSlice.actions;

export default userSlice.reducer;
```

Header index.js - компонент хедеру

```
import { Link, useNavigate } from 'react-router-dom';
import { Button, Container, Nav, Navbar } from 'react-bootstrap';
import { AuthModal } from '../main-content/authModal';
import { useEffect, useState } from 'react';
import { useDispatch, useSelector } from 'react-redux';
import { onAuthStateChanged } from 'firebase/auth';
import { setUser } from '../../store/userSlice';
import { getUserKey, auth, checkForAdmin } from '../../utils/firebase';
import { BASE_URL } from '../../utils/misc';

export const Header = () => {
  const dispatch = useDispatch();
  const [modalShow, setModalShow] = useState(false);
  const { isLoggedIn } = useSelector((state) => state.userSlice);
  const navigate = useNavigate();

  useEffect(() => {
    checkAuthorization();
  }, []);

  const checkAuthorization = () => {
    onAuthStateChanged(auth, async (user) => {
      if (user?.email) {
        const isAdmin = await checkForAdmin(user.email);
        const dbKey = await getUserKey(user.email);
        dispatch(setUser({ email: user.email, isAdmin, dbKey }));
      }
    });
  };
```

```
  const handleAccountClick = () => {
    if (isLoggedIn) navigate(`/${BASE_URL}/account`);
    else toggleModal(true);
  };

  const toggleModal = (status) => setModalShow(status);

  return (
    <>
      <Navbar bg="light" expand="lg">
        <Container>
          <Navbar.Brand as={Link} to="/">
            Blast Wave Electronics
          </Navbar.Brand>
          <Navbar.Toggle />
          <Navbar.Collapse className="justify-content-end">
            <Nav>
              <Nav.Link as={Link} to={`/${BASE_URL}/products`}>
                Products
              </Nav.Link>
              <Nav.Link as={Link} to={`/${BASE_URL}/about`}>
                About
              </Nav.Link>
              <Nav.Link as={Link} to={`/${BASE_URL}/contacts`}>
                Contacts
              </Nav.Link>
              <Button variant="outline-dark" onClick={handleAccountClick}>
                {isLoggedIn ? 'My Account' : 'Log In'}
              </Button>
            </Nav>
          </Navbar.Collapse>
        </Container>
      </Navbar>
      <AuthModal show={modalShow} toggleModal={toggleModal} />
    </>
  );
};
```

Footer index.js - компонент футера

```
import { Container } from 'react-bootstrap';

export const Footer = () => {
```

```
  return (
    <div className="bg-light py-4">
      <Container>
        <p className="m-0 text-center">©2022 Blast Wave Electronics</p>
      </Container>
    </div>
  );
};
```

Main-contetn index.js - головний файл основного контенту сторінок

```
import { Container } from 'react-bootstrap';
import { Products } from './products';
import { About } from './about/About';
import { Navigate, useRoutes } from 'react-router-dom';
import { Account } from './account';
import { useSelector } from 'react-redux';
import { ProductPage } from './products/ProductPage';
import { Contacts } from './contacts';
import { BASE_URL } from '../../utils/misc';

export const MainContent = () => {
  const { isLoggedIn } = useSelector((state) => state.userSlice);

  return (
    <Container className="pb-4 flex-grow-1">
      {useRoutes([
        { path: `/`, element: <Navigate to={`/${BASE_URL}`} replace /> },
        { path: `/${BASE_URL}/`, element: <Navigate
to={`/${BASE_URL}/products`} replace /> },
        { path: `/${BASE_URL}/products`, element: <Products /> },
        { path: `/${BASE_URL}/products/:productName`, element:
<ProductPage /> },
        { path: `/${BASE_URL}/about`, element: <About /> },
        { path: `/${BASE_URL}/contacts`, element: <Contacts /> },
        {
          path: `/${BASE_URL}/account`,
          element: isLoggedIn ? <Account replace /> : <Navigate
to={`/${BASE_URL}/products`} />
        }
      ])}
    </Container>
  );
};
```

Main-contetn About index.js - компонент інформації про компанію

```
import { Container } from 'react-bootstrap';
import { Logo } from './Logo';

export const About = () => {
  return (
    <Container>
      <div className="col-4 mx-auto">
        <Logo />
      </div>
      <div className="col-8 mt-4 mx-auto text-center">
        <h1>Where it all started...</h1>
        <p>
          Lorem ipsum dolor sit, amet consectetur adipisicing elit. Nemo nihil
cumque doloremque, quo eligendi laborum!
          Ad totam rem fugit, a alias ab nostrum, omnis fugiat quos velit
consectetur necessitatibus unde! Quos ipsa
          optio odio ipsam, modi commodi quod architecto nulla inventore
exercitationem saepe alias iure, ad nam, hic
          amet adipisci. Enim, dicta? Animi perferendis consequuntur cumque,
repellendus quia magnam numquam? Optio
          accusamus aut ad dolores nostrum, debitis sunt vero maiores ullam
quaerat saepe, voluptatibus tempore
          explicabo rerum nisi alias iure cupiditate consectetur quasi? Mollitia,
suscipit qui. Et eos ea magni. Id
          pariatur debitis, provident ipsum nesciunt saepe illum rem rerum sint
omnis sequi quae laborum culpa voluptas,
          veritatis quisquam! Neque incidunt, nihil itaque reprehenderit fugiat
tempora natus ducimus accusamus nobis!
          Autem animi itaque saepe eveniet doloremque facere, aliquam veritatis
alias placeat! Repudiandae itaque
          tempore dolore ut tenetur unde temporibus consequuntur numquam sit
nam minima, quos possimus, quibusdam
          deserunt atque quasi.
        </p>
      </div>
    </Container>
  );
};
```

Main-contetn about Logo.js - компонент логотипу

```
import { Image } from 'react-bootstrap';
```

```
import logo from '../../../assets/logo.png';

export const Logo = () => {
  return <Image style={{ objectFit: 'contain', width: '100%' }} src={logo} />;
};
```

Main-content account index.js - файл профілю / кошика користувача

```
import { useEffect, useState } from 'react';
import { Button, Table, Toast, ToastContainer } from 'react-bootstrap';
import { useDispatch, useSelector } from 'react-redux';
import { useNavigate } from 'react-router-dom';
import { fetchCart, signOut } from '../../../store/userSlice';
import { addToCart, signOutUser, updateCart } from '../../../utils/firebase';
import { toUpperCase } from '../../../utils/misc';
import { PurchaseModal } from './purchaseModal';

export const Account = () => {
  const dispatch = useDispatch();
  const { authData, cart } = useSelector((state) => state.userSlice);
  const navigate = useNavigate();

  const [localCart, setLocalCart] = useState([]);
  const [showToast, setShowToast] = useState(false);

  useEffect(() => {
    dispatch(fetchCart(authData.dbKey));
  }, []);

  useEffect(() => {
    setLocalCart(cart);
  }, [cart]);

  const [modalShow, setModalShow] = useState(false);

  const toggleModal = (status) => setModalShow(status);

  const handleQuantityChange = (value, product, index) => {
    addToCart(authData.dbKey, { ...product, quantity: value }, false);
    setLocalCart(localCart.map((el, i) => (i === index ? { ...el, quantity: value }
: el)));
  };

  const handleDelete = (index) => {
```

```
  let updatedCart = [...localCart].splice(index, 1);
  if (updatedCart.length === 1) updatedCart = [];
  updateCart(updatedCart, authData.dbKey);
  setLocalCart(updatedCart);
};

const onSignOut = async () => {
  await signOutUser();
  dispatch(signOut());
  navigate('/');
};

const onOrderSuccess = () => {
  setShowToast(true);
  setTimeout(() => {
    setShowToast(false);
  }, 3000);
};

return (
  <>
    <h1>Hello, {authData.email}</h1>
    <h2>Here is your cart:</h2>
    <Table striped bordered hover>
      <thead>
        <tr>
          <th>#</th>
          <th>Product</th>
          <th>Quantity</th>
        </tr>
      </thead>
      <tbody>
        {localCart &&
          localCart.map((el, i) => {
            return (
              <tr key={el.name}>
                <td>{i + 1}</td>
                <td>{toUpperCase(el.name)}</td>
                <td>
                  <input
                    size={3}
                    value={localCart[i].quantity}
                    min={1}
```

```
                    onChange={(e) => handleQuantityChange(e.target.value, el, i)}
                  />
                </td>
                <td>
                  <Button onClick={() => handleDelete(i)} variant="danger">
                    Delete
                  </Button>
                </td>
              </tr>
            );
          })}
        </tbody>
      </Table>
      <div className="d-flex justify-content-between">
        <Button variant="outline-success" onClick={() => toggleModal(true)}>
          Checkout
        </Button>
        <Button variant="outline-danger" onClick={() => onSignOut()}>
          Sign Out
        </Button>
      </div>
      <ToastContainer position="middle-center">
        <Toast show={showToast}>
          <Toast.Header>
            <strong className="me-auto">Your order</strong>
            <small className="text-muted">success</small>
          </Toast.Header>
          <Toast.Body>{localCart.map((el) => `${el.name}
x${el.quantity}`).join('\n')}</Toast.Body>
        </Toast>
      </ToastContainer>
      <PurchaseModal order={localCart} show={modalShow}
toggleModal={toggleModal} onOrderSuccess={onOrderSuccess} />
    </>
  );
};
```

Main-content account purchaseModal.js - компонент модального вікна покупки

```
import { Formik, useFormik } from 'formik';
import { Button, Form, Modal } from 'react-bootstrap';
import { sendOrderDetailsToTelegram } from '../../../utils/telegram';
```

```
export const PurchaseModal = ({ show, toggleModal, order, onOrderSuccess }) => {
  const handleSubmit = (data) => {
    sendOrderDetailsToTelegram({ ...data, order: order.map((el) => ({ name: el.name, quantity: el.quantity })) });
    toggleModal(false);
    onOrderSuccess();
  };

  const validate = (values) => {
    const errors = {};

    if (!values.name) {
      errors.name = 'Required';
    }

    if (!values.address) {
      errors.address = 'Required';
    }

    if (!values.phone) {
      errors.phone = 'Required';
    }

    return errors;
  };

  const formik = useFormik({
    initialValues: {
      name: '',
      address: '',
      phone: ''
    },
    onSubmit: handleSubmit,
    validate
  });

  return (
    <Modal show={show} onHide={() => toggleModal(false)}>
      <Modal.Header closeButton>
        <Modal.Title>You almost got your new pedal</Modal.Title>
      </Modal.Header>
      <Modal.Body>
```

```
            <Formik>
              <Form
                onSubmit={formik.handleSubmit}
                onKeyDown={(e) => {
                  if (e.key === 'Enter') formik.submitForm();
                }}
              >
                <Form.Group className="mb-3">
                  <Form.Label>Please, enter your name</Form.Label>
                  <Form.Control
                    className={formik.errors.name && 'is-invalid'}
                    name="name"
                    onChange={formik.handleChange}
                    type="email"
                    placeholder="Name"
                  />
                  <div className={`${formik.errors.name && 'in'}valid-
feedback`}>{formik.errors.name}</div>
                </Form.Group>
                <Form.Group className="mb-3">
                  <Form.Label>Address</Form.Label>
                  <Form.Control
                    className={formik.errors.address && 'is-invalid'}
                    name="address"
                    onChange={formik.handleChange}
                    type="text"
                    placeholder="Address"
                  />
                  <div className={`${formik.errors.address && 'in'}valid-
feedback`}>{formik.errors.address}</div>
                </Form.Group>
                <Form.Group className="mb-3">
                  <Form.Label>Contact phone</Form.Label>
                  <Form.Control
                    className={formik.errors.phone && 'is-invalid'}
                    name="phone"
                    onChange={formik.handleChange}
                    type="phone"
                    placeholder="Phone number"
                  />
                  <div className={`${formik.errors.phone && 'in'}valid-
feedback`}>{formik.errors.phone}</div>
                </Form.Group>
```

```
            </Form>
          </Formik>
        </Modal.Body>
        <Modal.Footer>
          <Button variant="secondary" onClick={() => toggleModal(false)}>
            Close
          </Button>
          <Button onClick={() => formik.submitForm()} variant="primary">
            Submit
          </Button>
        </Modal.Footer>
      </Modal>
    );
  };
```

Main-content authModal index.js - компонент модального вікна авторизації

```
  import { Formik, useFormik } from 'formik';
  import { useState } from 'react';
  import { Button, Form, Modal } from 'react-bootstrap';
  import { addUserToDb, auth, checkForAdmin, getUserKey } from
'../../../utils/firebase';
  import { createUserWithEmailAndPassword, signInWithEmailAndPassword }
from 'firebase/auth';
  import { setUser } from '../../../store/userSlice';
  import { useDispatch } from 'react-redux';
  import { useNavigate } from 'react-router-dom';
  import { BASE_URL } from '../../../utils/misc';

  export const AuthModal = ({ show, toggleModal }) => {
    const dispatch = useDispatch();
    const [isNewUser, setIsNewUser] = useState(false);
    const navigate = useNavigate();

    const createNewUser = async ({ email, password }) => {
      await createUserWithEmailAndPassword(auth, email, password)
        .then(async (userCredential) => {
          const user = userCredential.user;
          const newUserKey = await addUserToDb(user.email);
          dispatch(setUser({ email: user.email, dbKey: newUserKey }));
        })
        .catch(({ code, message }) => {
          throw new Error({ code, message });
        });
```

```
  };

  const signInUser = async ({ email, password }) => {
    await signInWithEmailAndPassword(auth, email, password)
      .then(async (userCredential) => {
        const user = userCredential.user;
        const isAdmin = await checkForAdmin(email);
        const dbKey = await getUserKey(email);
        dispatch(setUser({ email: user.email, isAdmin, dbKey }));
      })
      .catch((error) => {
        console.log(error);
        // throw new Error({ code, message });
      });
  };

  const handleSubmit = async (data) => {
    try {
      if (isNewUser) await createNewUser(data);
      else await signInUser(data);
    } catch (error) {
      console.log(error);
    }
    toggleModal(false);
    navigate(`${BASE_URL}/account`);
  };

  const validate = (values) => {
    const errors = {};

    if (!values.email) {
      errors.email = 'Required';
    } else if (!/^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-
Z]{2,4}$/i.test(values.email)) {
      errors.email = 'Invalid email address';
    }

    if (!values.password) {
      errors.password = 'Required';
    } else if (!/^.{6,}$/.test(values.password)) {
      errors.password = 'Must be 6 characters at least';
    }
```

```
  return errors;
 };

 const formik = useFormik({
  initialValues: {
   email: '',
   password: ''
  },
  onSubmit: handleSubmit,
  validate
 });

 return (
  <Modal show={show} onHide={() => toggleModal(false)}>
   <Modal.Header closeButton>
    <Modal.Title>Land into your account</Modal.Title>
   </Modal.Header>
   <Modal.Body>
    <Formik>
     <Form
      onSubmit={formik.handleSubmit}
      onKeyDown={(e) => {
       if (e.key === 'Enter') formik.submitForm();
      }}
     >
      <Form.Group className="mb-3">
       <Form.Label>Please, enter your e-mail</Form.Label>
       <Form.Control
        className={formik.errors.email && 'is-invalid'}
        name="email"
        onChange={formik.handleChange}
        type="email"
        placeholder="Enter email"
       />
       <div className={`${formik.errors.email && 'in'}valid-
feedback`}>{formik.errors.email}</div>
      </Form.Group>
      <Form.Group className="mb-3">
       <Form.Label>And your assword</Form.Label>
       <Form.Control
        className={formik.errors.password && 'is-invalid'}
        name="password"
        onChange={formik.handleChange}
```

```
                type="password"
                placeholder="Password"
              />
              <div className={`${formik.errors.password && 'in'}valid-
feedback`}>{formik.errors.password}</div>
            </Form.Group>
          </Form>
        </Formik>
      </Modal.Body>
      <Modal.Footer>
        <a href="#" className="link-secondary" onClick={() =>
setIsNewUser(!isNewUser)}>
          {isNewUser ? `I've already have an account` : `Don't have an account
yet?`}
        </a>
        <Button variant="secondary" onClick={() => toggleModal(false)}>
          Close
        </Button>
        <Button onClick={() => formik.submitForm()} variant="primary">
          {isNewUser ? 'Create Account' : 'Log In'}
        </Button>
      </Modal.Footer>
    </Modal>
  );
};
```

Main-content contacts index.js - компонент контактів

```
export const Contacts = () => {
  return (
    <div className="text-center">
      <h2 className="my-4">In case of any questions, please get in touch via
any way below</h2>
      <h5 className="mt-4">+380772287777</h5>
      <h5>+380562140088</h5>
      <h5>ULTIMATECRINGEGENERATOR3000@i.ua</h5>
    </div>
  );
};
```

Main-content products index.js - головний файл товарів

```
import { useState } from 'react';
import { useSelector } from 'react-redux';
import { useSearchParams } from 'react-router-dom';
```

```
import { Logo } from '../about/Logo';
import { Filters } from './Filters';
import { ProductsList } from './ProductsList';
import { Sorting } from './Sorting';

export const Products = () => {
  const [searchParams, setSearchParams] = useSearchParams();
  const [sortBy, setSortBy] = useState();
  const { productsList, filters, status } = useSelector((state) =>
state.productsSlice);

  return (
    <div>
      <div className="col-4 mx-auto">
        <Logo />
      </div>
      <div className="w-100 d-flex justify-content-center flex-wrap mt-4">
        <Filters types={filters} setSearchParams={setSearchParams} />
      </div>
      <div className="w-100 d-flex justify-content-center my-2">
        <Sorting setSortBy={setSortBy} />
      </div>
      <div className="w-100">
        <ProductsList productsList={productsList} status={status}
sortBy={sortBy} filterBy={searchParams.get('type')} />
      </div>
    </div>
  );
};
```

Main-content products sorting.js - компонент сортування

```
import React from 'react';
import { Dropdown } from 'react-bootstrap';
import { toUpperCase } from '../../../utils/misc';

export const Sorting = ({ setSortBy }) => {
  const sorting = ['name', 'price'];

  return (
    <Dropdown>
      <Dropdown.Toggle variant="secondary">Sort by</Dropdown.Toggle>
      <Dropdown.Menu>
        {sorting.map((el) => {
```

```
            return (
              <React.Fragment key={el}>
                <Dropdown.Item onClick={() => setSortBy({ by: el, dir: 'up'
})}>{toUpperCase(el)} ↑</Dropdown.Item>
                <Dropdown.Item onClick={() => setSortBy({ by: el, dir: 'down'
})}>{toUpperCase(el)} ↓</Dropdown.Item>
              </React.Fragment>
            );
          })}
        </Dropdown.Menu>
      </Dropdown>
    );
  };
```

Main-content products filters.js - компонент фільтрації

```
  import { Button } from 'react-bootstrap';
  import { toUpperCase } from '../../../utils/misc';

  export const Filters = ({ types, setSearchParams }) => {
    return (
      <>
        <Button onClick={() => setSearchParams('')} className="m-2"
variant="outline-secondary">
          All
        </Button>
        {types.map((type) => {
          return (
            <Button
              key={`${Math.random()}`}
              onClick={() => setSearchParams({ type })}
              className="m-2"
              variant="outline-dark"
            >
              {toUpperCase(type)}
            </Button>
          );
        })}
      </>
    );
  };
```

Main-content products productList.js - компонент списку товарів

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

```
import { useDispatch, useSelector } from 'react-redux';
import { AddNewCard, ProductCard } from './ProductCard';
import { fetchProducts, uploadImage } from '../../../store/productsSlice';
import { useLocation, useNavigate } from 'react-router-dom';

export const ProductsList = ({ productsList, filterBy, sortBy }) => {
  const [products, setProducts] = useState([]);
  const dispatch = useDispatch();
  const navigate = useNavigate();
  let location = useLocation();

  const { isAdmin } = useSelector((state) => state.userSlice);

  useEffect(() => {
    dispatch(fetchProducts());
  }, []);

  useEffect(() => {
    if (!filterBy) setProducts(productsList);
    else setProducts(productsList.filter((el) => el.type === filterBy));
  }, [filterBy, productsList]);

  useEffect(() => {
    if (sortBy) setProducts(sortProductsListBy(sortBy));
  }, [sortBy, productsList]);

  const handleImgUpload = (file) => {
    dispatch(uploadImage(file));
  };

  const sortProductsListBy = ({ by, dir }) => {
    switch (dir) {
      case 'up':
        return [...products].sort((a, b) => a[by].localeCompare(b[by]));
      case 'down':
        return [...products].sort((a, b) => -1 * a[by].localeCompare(b[by]));
      default:
        break;
    }
  };

  const openProduct = (name) => {
    navigate(`${location.pathname}/${name}`);
```

```
  };

  return (
    <div className="d-flex flex-wrap justify-content-center">
      {products.map((productInfo) => {
        return (
          <ProductCard
            data={productInfo}
            isAdmin={isAdmin}
            key={productInfo.id}
            onImgUpload={handleImgUpload}
            openProductPage={openProduct}
          />
        );
      })}
      {isAdmin && <AddNewCard />}
    </div>
  );
};
```

Main-content products productCard.js - компонент карточки товару

```
import { Formik, useFormik } from 'formik';
import { useState } from 'react';
import { Button, Card, Form } from 'react-bootstrap';
import { toUpperCase } from '../../../utils/misc';

const handleSubmit = (data, img) => {
  // onImgUpload(img)
  // setData
  console.log(data, img);
};

const adminCard = (formik, handleFileChange) => {
  return (
    <Card className="m-2" style={{ width: '18rem' }}>
      <Card.Img
        className="p-3"
        style={{
          maxHeight: '200px',
          objectFit: 'contain'
        }}
        variant="top"
        src={formik.values.imgUrl}
```

```
            />
            <Button style={{ position: 'absolute', top: '0', right: '0' }}
variant="primary" size="sm">
              change
            </Button>
            <Card.Body>
              <Form.Group className="mb-1">
                <Form.Control name="name" placeholder="Name"
value={formik.values.name} onChange={formik.handleChange} />
                <Form.Text className="text-muted">Product Name</Form.Text>
              </Form.Group>
              <Form.Group className="mb-1">
                <Form.Control name="price" placeholder="Price"
value={formik.values.price} onChange={formik.handleChange} />
                <Form.Text className="text-muted">Product Price</Form.Text>
              </Form.Group>
              <Form.Group className="mb-1">
                <Form.Control
                  name="description"
                  as={'textarea'}
                  placeholder="Description"
                  value={formik.values.description}
                  onChange={formik.handleChange}
                />
                <Form.Text className="text-muted">Description</Form.Text>
              </Form.Group>
              <Form.Group className="mb-1">
                <Form.Control type="file" name="img" onChange={handleFileChange}
/>
                <Form.Text className="text-muted">Select Image</Form.Text>
              </Form.Group>
              <Button onClick={() => formik.submitForm()} variant="primary">
                Submit
              </Button>
            </Card.Body>
          </Card>
        );
      };

      export const AddNewCard = () => {
        const formik = useFormik({
          initialValues: {
            name: '',
```

```
      price: '',
      description: ''
    },
    onSubmit: handleSubmit
  });

  return adminCard(formik);
};

export const ProductCard = ({ data: { name, price, description, imgUrl },
isAdmin, onImgUpload, openProductPage }) => {
  const [img, setImg] = useState(null);

  const handleFileChange = (e) => {
    setImg(e.currentTarget.files[0]);
  };

  const formik = useFormik({
    initialValues: {
      name: toUpperCase(name),
      price,
      description,
      imgUrl
    },
    onSubmit: (values) => handleSubmit(values, img, onImgUpload)
  });

  if (isAdmin) {
    return (
      <Formik>
        <Form onSubmit={formik.handleSubmit}>{adminCard(formik,
handleFileChange)}</Form>
      </Formik>
    );
  } else {
    return (
      <Card className="m-2 text-truncate" style={{ width: '18rem' }}>
        <Card.Img
          className="p-3"
          style={{
            maxHeight: '200px',
            objectFit: 'contain',
            cursor: 'pointer'
```

```
            }}
            onClick={() => openProductPage(name)}
            variant="top"
            src={imgUrl}
          />
          <Card.Body>
            <Card.Title>
              {toUpperCase(name)} <span className="badge bg-
success">${price}</span>
            </Card.Title>
            <Card.Text className="text-truncate">{description}</Card.Text>
          </Card.Body>
        </Card>
      );
    }
  };
```

Main-content products productPage.js - компонент сторінки товару

```
  import { useEffect, useState } from 'react';
  import { Button, Col, Container, Row } from 'react-bootstrap';
  import { useDispatch, useSelector } from 'react-redux';
  import { useParams } from 'react-router-dom';
  import { fetchProducts, setOpenProduct } from '../../../store/productsSlice';
  import { addToCart } from '../../../utils/firebase';

  export const ProductPage = () => {
    const dispatch = useDispatch();
    const { productName } = useParams();
    const productsSlice = useSelector((state) => state.productsSlice);
    const { isLoggedIn, authData } = useSelector((state) => state.userSlice);
    const [buyBtnStatus, setBuyBtnStatus] = useState({ title: '', disabled: true });

    const animateBuyBtnTitle = () => {
      const prevStatus = buyBtnStatus;
      setBuyBtnStatus({ title: 'Added to Cart!', disabled: true });
      setTimeout(() => {
        setBuyBtnStatus(prevStatus);
      }, 3000);
    };

    const handleAddToCart = () => {
      animateBuyBtnTitle();
```

```
        addToCart(authData.dbKey, productsSlice.openProduct, true);
      };

      useEffect(() => {
        if (productsSlice.productsList.length === 0) dispatch(fetchProducts());
      }, []);

      useEffect(() => {
        setBuyBtnStatus({
          title: isLoggedIn ? 'Add to Cart' : 'Log in to add to Cart',
          disabled: isLoggedIn ? false : true
        });
      }, [isLoggedIn]);

      useEffect(() => {
        if (productsSlice.productsList !== 0)
dispatch(setOpenProduct(productName));
      }, [productsSlice.productsList]);

      return (
        <Container>
          <Row>
            <Col xs={6}>
              <img style={{ maxWidth: '100%' }}
src={productsSlice.openProduct?.imgUrl} />
            </Col>
            <Col xs={6}>
              <h2>{productsSlice.openProduct?.name}</h2>
              <p>{productsSlice.openProduct?.description}</p>
              <Button onClick={handleAddToCart} variant="success"
disabled={buyBtnStatus.disabled}>
                {buyBtnStatus.title}
              </Button>
            </Col>
          </Row>
        </Container>
      );
    };
```

ДОДАТОК Б

ВІДГУК
керівника економічного розділу
на кваліфікаційну роботу бакалавра
на тему:
«Розробка веб-орієнтованої системи з продажу товарів з використанням React.js»
студента групи 122-18-1 Бичкова Іллі Олеговича

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	
Диплом Бичков.doc	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Диплом Бичков.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF.
Програма	
diploma.zip	Архів. Містить коди програми.
Презентація	
Презентація Бичков.ppt	Презентація кваліфікаційної роботи.