Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут комп'ютерних наук
та інформаційних технологій
(повне найменування інституту, назва факультету(відділення))

Кафедра інформаційних систем та комп'ютерного моделювання
(повна назва кафедри (предметної циклової комісії))

Пояснювальна записка

до дипломної роботи
перший (бакалаврський)
(освітньо-кваліфікаційний рівень)

на тему: **Розроблення інтернет магазину квітів засобами "React"**
(тема роботи)

Виконав студент 4 курсу, групи ІСТ-41
спеціальності: 126 „Інформаційні системи та технології"
(шифр і назва напрямку підготовки спеціальності)

Древняк П.М.
(прізвище, ініціали)

Керівники: Гіссовська Н.О., Сторожук О.Л.
(прізвище, ініціали)

Рецензент: Дендюк М.В.
(прізвище, ініціали)

Львів-2024

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій
Кафедра інформаційних систем та комп'ютерного моделювання
Рівень вищої освіти перший(бакалаврський)
Спеціальність 126 " Інформаційні системи та технології "

ЗАТВЕРДЖУЮ:
Завідувач кафедри ІСКМ
_____Сторожук О.Л.
„06" 02 2024 р.

ЗАВДАННЯ
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ
Древняку Павлу Михайловичу
(прізвище, ім'я, по батькові)

1.Тема бакалаврської роботи: «Розроблення інтернет магазину квітів засобами "React"»
керівники роботи: Гіссовська Наталія Олександрівна, Сторожук Олександр Леонідович к.т.н., доц.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від "6" лютого 2024 року, №С-87
2.Термін подання студентом проекту(роботи) 10 червня 2024р
3. Вихідні дані до проекту (роботи) Розробити програмну реалізацію та візуалізацію інтернет магазину квітів засобами "React"
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Стан проблемної області
Інформаційне та математичне забезпечення
Програмне та технічне забезпечення
Висновки
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Підготовка матеріалів до доповіді
6. Дата видачі завдання 7 лютого 2024р.

КАЛЕНДАРНИЙ ПЛАН

№, з/п	Етапи бакалаврської роботи	Термін виконання етапів роботи	Примітка
1.	Огляд літератури згідно досліджуваної теми. Збір необхідних матеріалів.	07.02.24-20.02.24	виконано
2.	Постановка задачі і її формалізація	20.02.24-05.03.24	виконано
3.	Виконання вхідного етапу технології	05.03.24-25.03.24	виконано
4.	Реалізація головних класів проекту	25.03.24-14.04.24	виконано
5.	Виконання етапу відлагодження проєкту	14.04.24-10.05.24	виконано
6.	Виконання етапу впровадження та випуску бета-версії.	10.05.24-02.06.24	виконано
7.	Оформлення записки до дипломного проекту.	02.06.24-10.06.24	виконано

Студент _____ (підпис) Древняк П.М. (прізвище та ініціали)

Керівники роботи _____ (підпис) Гіссовська Н.О. (прізвище та ініціали)

_____ (підпис) Сторожук О.Л. (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота містить 47 сторінок пояснювальної записки, 27 рисунків, 15 джерел, 1 додатку.

У даній дипломній роботі розроблено інтернет-магазин квітів "Flowers". Ця система була створена з використанням таких технологій, як React для фронтенд-розробки та Firebase для забезпечення бекенду та бази даних. Для стилізації компонентів використовувався Material-UI.

Ця платформа дозволяє користувачам переглядати весь асортимент квіткових новинок та купувати їх з максимальною зручністю.

Ключові слова: мова програмування, сайт, React, програмний код, Firebase, JavaScript, веб-система.

ABSTRACT

The thesis contains 47 pages of an explanatory note, 27 figures, 15 sources, 1appendix.

This diploma project presents the development of an online flower shop "Flowers". The system was created using technologies such as React for front-end development and Firebase for backend and database support. Material-UI was used for component styling.

This platform allows users to browse the entire range of floral novelties and purchase them with maximum convenience.

Keywords: programming language, website, React, code, Firebase, JavaScript, web system.

ТЕХНІЧНЕ ЗАВДАННЯ

Мета: Розробити веб-орієнтовану інформаційну систему для інтернет-магазину квітів "Flowers", яка повинна включати:

- Оболонку та інтерфейси створені веб-засобами;
- Інтуїтивно зрозумілий та зручний для користувача інтерфейс.
- Використання сучасних фреймворків та бібліотек для розробки фронтенду.

Веб-сторінки, які дозволяють користувачам;

- Переглядати повний асортимент квіткових новинок;
- Отримувати детальну інформацію про кожен продукт;
- Додавати товари до кошика та здійснювати покупки;
- Розробка зручної навігації;
- Застосування категорій;
- Забезпечити інформативний та привабливий дизайн;
- Використання принципів адаптивного дизайну для підтримки різних пристроїв;

Технології та інструменти:

- Фронтенд: React для розробки користувацького інтерфейсу;
- Бекенд: Firebase для забезпечення серверної логіки та бази даних;
- Стилізація: Material-UI для забезпечення сучасного та зручного дизайну;

Очікуваний результат: Інтуїтивно зрозуміла, зручна та функціональна інформаційна система для інтернет-магазину квітів "Flowers", яка дозволяє користувачам легко знаходити та купувати квіткові новинки

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

IT – інформаційні технології;

IC – інформаційна система;

JS – Java Script;

HTML – HyperText Mark-up Language;

CSS - Cascading Style Sheets;

JSX – JavaScript XML;

SPA – Single Page Application.

ВСТУП

У нашому сучасному світі електронна торгівля перетворюється на невід'ємну складову бізнесу, що стає не просто ключовою, але й необхідною платформою для реалізації товарів та послуг через простір мережі. Це відкриває нові перспективи для розвитку бізнесу та сприяє максимальній доступності та зручності для клієнтів. У рамках даної роботи був створений інтернет-магазин, який займається продажею квітів, квіткових композицій та декоративних рослин. Цей магазин отримав назву "Flowers".

"Flowers" - це бренд, який став відомим завдяки своїй унікальній концепції та високій якості своїх товарів. Мережа квіткових магазинів "Flowers" вже завоювала довіру клієнтів та зарекомендувала себе на ринку.

Об’єктом дослідження є розроблення інтернет магазину квітів “ Flowers” в середовищі React.

Метою роботи є розробка зручного та доступного середовища для онлайн покупок квітів та квіткових композицій. Магазин пропонує широкий вибір квітів та рослин різних сортів і видів, які доступні для покупки за допомогою Інтернету.

Предметом дослідження є створення та розгортання веб-орієнтованої системи з використанням технологій React та Firebase. Основна увага приділяється розробці функціональності веб-сайту та інтеграції з базою даних Firebase для забезпечення потреб користувачів у зручномий та ефективномий спосіб.

Практичне значення роботи демонструється у використанні технології React для створення та розвитку веб-магазинів. Реалізація цього проекту дозволить вивчити та використати сучасні методи та підходи до розробки клієнтських веб-додатків, а також спростить процес взаємодії користувачів з інтернет-магазином, забезпечуючи їм зручність та швидкість використання.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1 Огляд проблемної області.

Ми любимо радувати наших мам, дівчат, дружин та сестер ніжними букетами квітів. Цей подарунок — символ нашої любові та уваги. Чоловіки по всьому світу видихнули з полегшенням, дізнавшись, що тепер квіти можна замовляти онлайн.

Замість того, щоб виходити з дому, люди тепер можуть здійснювати покупки, натискаючи всього лише кілька кнопок на своїх пристроях. І цей тренд швидко розповсюджується на всі сфери бізнесу, включаючи індустрію квітів.

В період сучасності інтернет-шопінг визнається однією з найбільш зручних та вигідних форм покупок для користувачів мережі, саме тому квіткові інтернет магазини стрімко розвиваються, і можуть бути корисні наступним представникам бізнесу:

- фрилансерам, які працюють на себе;
- невеликим павільйонам;
- магазинам, які хочуть працювати з доставлянням;
- квітковим дизайнерам;
- флористичним студіям;

Створення сайту з продажу квітів потребує ретельної підготовки. Квіти — сезонний і швидкопсувний товар, що вимагає налагодженої системи збуту. Кожен гвинтик у машині продажу, від закупівлі до доставки споживачеві, повинен працювати ідеально, тому дуже важливо налаштувати двосторонню комунікацію з покупцем: так щоб він отримував максимум інформації про наші пропозиції, умови оплати та доставки.

Асортимент магазину включає квіти різних сортів і кольорів, від класичних троянд до екзотичних орхідей, що задовольнить будь-який смак та побажання клієнтів. Ми гарантуємо свіжість та якість кожної квітки, що відправляється з нашого магазину

Інтернет-магазин має радувати високою яскістю фотоматеріалів, які вміють "продавати самі себе" та стимулювати регулярні замовлення. Найприкріше — це "блукання" потенційного клієнта сайтом. Якщо каталог нагадує карту скарбів, відвідувач швидко закриє вкладку. Тому сайт має бути простим і зрозумілим.

Інтернет-магазин квітів "Flowers" - це відмінна можливість для тих, хто шукає свіжі та якісні квіти за доступними цінами, не залишаючи затишку власної домівки. Наш магазин пропонує широкий вибір квіткових композицій, букетів та горшкових рослин для будь-яких випадків та свят. Щоб вся увага була спрямована на товар, інтернет-магазин розроблений у мінімалістичному стилі. Простий та зрозумілий інтерфейс акцентує увагу на красі рослин, без зайвих графічних хитрощів.

Зробіть свої різдвяні подарунки особливими за допомогою наших чудових квіткових композицій, або зробіть яскравий акцент на будь-якому святі з нашою допомогою. Замовляйте квіти онлайн в нашому магазині "Flowers" і отримуйте задоволення від кожної купівлі!

Таким чином, інтернет-магазин квітів "Flowers" надає зручний та простий спосіб придбати квіткові композиції та потішити себе, або своїх близьких свіжими та прекрасними квітами.

1.2 Чому відкривати інтернет-магазин є пріорітетом

Інтернет назавжди змінив те, як споживачі купують товари чи послуги. Згідно з останніми даними European B2C Ecommerce Report, понад 3,1 мільйона, або 35% португальців, вже купують онлайн, очікується, що в найближчі роки це число зростатиме в геометричній прогресії, і це відповідає обсягу бізнесу понад 3 мільярди євро. У всьому світі ця цифра зростає до 1,61 мільярда, що становить близько 43% від усіх користувачів Інтернету. Враховуючи, що інтернет-магазини не мають кордонів, це означає, що майже 2 мільярди людей у всьому світі є потенційними покупцями.

- Електронна комерція продовжить зростати

Очікується, що електронна комерція продовжить зростати в найближчі роки,

не тільки досягнувши 2 мільярдів покупців у всьому світі в 2020 році, але й досягнувши 4 мільярдів доларів продажів.

Тільки в Португалії електронна комерція значно зростає з кожним роком, з темпами зростання від 13,3% до 15,9%. Ця тенденція збережеться і в найближчі роки, зберігаючи двозначні темпи зростання, а це означає, що зараз ідеальний час, щоб увійти у світ електронної комерції та скористатися хвилею зростання цієї торгової трансформації.

- Рішення споживачів сьогодні ґрунтується на Інтернеті

Фізичних магазинів без присутності в Інтернеті вже недостатньо. Процес покупки споживачами змінився, і сьогодні першим кроком завжди є проведення онлайн-дослідження. Таким чином, компанії, які мають найкращу присутність в Інтернеті, і особливо ті, які пропонують можливість купувати безпосередньо через Інтернет, - це ті, які ловлять споживачів у той час, коли вони найбільш зацікавлені та сприйнятливі. Це ті, чиї веб-сайти оптимізовані для пошукових систем, хто знає, як робити ставку на цифровий маркетинг і хто розуміє, що чим кращий користувацький досвід, тим більше у користувачів спокуси купувати їхні продукти, а не конкурентів.

- Більше охоплення через Інтернет та мобільний зв'язок

Фізичні магазини завжди обмежені одним фактором: географічною присутністю. Якщо ваші потенційні клієнти знаходять ваш бізнес, лише зайшовши до вашого магазину, ваш асортимент завжди буде обмежений. З інтернет-магазином такого не відбувається- будь-який користувач може зайти в ваш магазин в інтернеті і зробити покупку, незалежно від того, де він знаходиться, а значить, асортимент надзвичайно широкий, якщо порівнювати з фізичною присутністю.

Крім того, існує мобільний вплив, тобто постійна присутність мобільних пристроїв, особливо смартфонів. Маючи практично необмежений доступ у будь-якому місці та в будь-якій ситуації, користувачі мають все більше можливостей для покупок, а мобільні пристрої виключно орієнтовані на те, щоб мотивувати

користувачів дізнаватися, що їм потрібно саме тоді, коли їм це потрібно. З веб-сайтом електронної комерції, адаптованим під мобільні пристрої, це означає, що в будь-яку секунду дня новий користувач може знайти вашу компанію та здійснити покупку. Навіщо чекати, поки нові клієнти дізнаються про ваш бізнес на вулиці, якщо він завжди може бути присутнім саме в той момент, коли ваша продукція потрібна?

- Інтернаціоналізації

Продовжуючи попередній пункт, сфера електронної комерції не обмежується країною її походження. Насправді інтернет-магазин – це справжня революція для інтернаціоналізації! З багатомовним інтернет-магазином немає бар'єрів для продажів - ви можете дістатися в будь-яку точку світу і в будь-який час. Таким чином, електронна комерція розширює ваш бізнес і ваш асортимент, долаючи будь-які географічні обмеження, щоб залучити людей, зацікавлених у вашій продукції, з будь-якої точки світу.

- Магазин завжди відкритий

Ще однією перевагою в порівнянні з фізичними магазинами, і ще одним, що робить електронну комерцію все більш незамінною, є той факт, що тут немає «робочих годин» - інтернет-магазин працює завжди, 24 години на добу, 7 днів на тиждень.

Це означає, що ви можете мати клієнта в будь-який час, і ви не обмежені доступністю клієнта на момент відкриття вашого магазину. Якщо врахувати той факт, що багато користувачів шукають те, що їм потрібно, в будь-який час, в тому числі і вночі (коли, звичайно, вони не знайдуть жодного відкритого фізичного магазину), це не тільки зручна перевага, але і відмінний момент для вашого бізнесу.

- Нові інструменти продажів

Інтернет-магазин не схожий на традиційний магазин - він кращий. Маючи у своєму розпорядженні численні цифрові інструменти, починаючи від A/B-тестування та теплових карт для оцінки реакції споживачів на вашому веб-сайті,

маркетингової електронної пошти та кампаній, а також оплати за клік для збільшення ваших продажів, автоматизації маркетингу до процесу автоматизованого залучення клієнтів та багато іншого, існує широкий спектр варіантів, які можуть зробити ваш інтернет-магазин автоматизованою машиною для продажів. Це справжнє майбутнє продажів, яке доступне лише в цифровому вигляді, тому вам потрібно адаптуватися та увійти в цей світ, якщо ви не хочете залишитися позаду.

- Якщо ви не в мережі, ваші конкуренти будуть

Це ключовий момент: чим довше ви відкладаєте свій неминучий перехід до цифрового адаптованого світу, тим швидше ваші конкуренти зможуть просунутися вперед і влаштуватися на першому місці для вашого регіону. Хоча є й інші важливі фактори, які означають, що електронна комерція – це не проста «гонка озброєнь», чим раніше ви завойовуєте своїх клієнтів, тим швидше ви зможете завоювати свою лояльність і завоювати сильніші позиції. І, звичайно, якщо ви не адаптуєтеся, ви не зможете пережити конкурентів, які розширюються в Інтернеті.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Платформа REACT

Реактивні технології, такі як React, можуть суттєво покращити розробку веб-магазину для продажу квітів завдяки своїм перевагам у створенні інтерфейсів користувача. Нижче наведені ключові аспекти використання React для створення веб-магазину Flowers:

React дарує нам можливість розробляти компоненти, які можна використовувати знову та знову, а також зручно керувати ними. Це дає змогу створювати складні структури веб-сайтів з простих блоків, спрощуючи процес розробки та підтримки коду.

React дозволяє розробникам розбивати інтерфейс на окремі компоненти, кожен з яких відповідає за свою частину UI. Для веб-магазину квітів це можуть бути компоненти для:

- Списку продуктів
- Карточки товару
- Кошика покупок
- Улюблені товари
- Форми замовлення
- Навігаційного меню

React надає можливості для ефективного управління станом додатку за допомогою useState і useReducer хуків. Для більших додатків, таких як веб-магазин, може знадобиться використання зовнішніх бібліотек для управління станом, таких як Redux або Context API.

React є потужною бібліотекою для створення інтерфейсів користувача, яка надає численні додаткові можливості для покращення розробки веб-додатків. Ось деякі з них:

Hooks дозволяють використовувати стан і інші можливості React без написання класів. Вони забезпечують гнучкість та покращують повторне використання логіки.

- useState: для управління локальним станом компонентів.
- useEffect: для побічних ефектів, таких як отримання даних з API.
- useContext: для використання контексту і глобального стану.
- useReducer: для складного управління станом.
- useMemo та useCallback: для оптимізації продуктивності.

Контекст API дозволяє передавати дані через дерево компонентів без необхідності передавати їх через кожен рівень дерева. Це зручно для глобального стану, наприклад, кошика покупок або теми додатку.

Організація компонентів у React є критично важливою для створення масштабованого та зручного у підтримці додатку. Розглянемо докладніше, як можна структурувати компоненти для веб-магазину квітів. (рисунок 2.1):

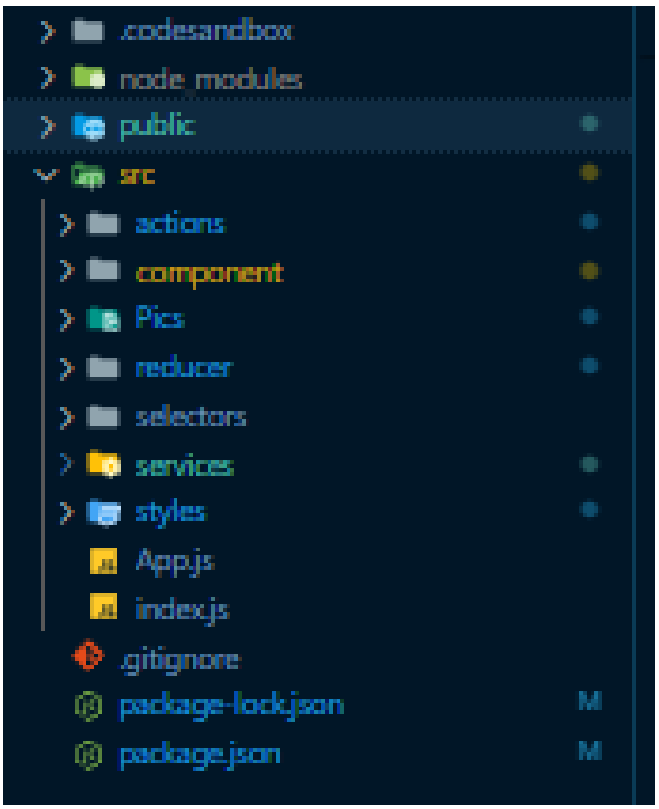


Рисунок 2.1 - Структура проекту веб-магазину квітів “Flovers”.

App.js: є кореневим компонентом, з якого починається рендеринг всього додатку.

index.js: Це точка входу програми React. Він відповідає за ініціалізацію додатку React у браузері та відображення його на веб-сторінці. У цьому файлі зазвичай імпортується головний компонент додатку та використовується функція ReactDOM.render() для відображення цього компонента у визначений HTML-елемент на сторінці.

components: це місце, де зазвичай зберігаються компоненти вашого додатку React. Компоненти - це невеликі частини коду, які відповідають за рендеринг певних частин інтерфейсу користувача. Розташовуючи їх у відповідній папці, ви організовуєте ваш проект, що полегшує його розуміння та підтримку.

services: містить файли, які відповідають за роботу зі службами або зовнішніми даними у вашому додатку. Це може включати запити до сервера, роботу з базою даних, управління станом додатку та інші операції, пов'язані з логікою вашого додатку. Розташовуючи ці файли в окремій папці, ви створюєте структурований підхід до управління даними та послугами вашого додатку, що полегшує розробку та підтримку проекту.

store: зазвичай містить файли, пов'язані з управлінням станом додатку за допомогою бібліотеки управління станом, наприклад Redux або MobX. Ці файли можуть містити дієвість, редуктори, посередники та інші елементи, необхідні для управління та зберігання стану додатку. Використання папки store допомагає структурувати код, пов'язаний зі станом, та зробити його більш легким у розумінні та підтримці.

Під час створення веб-сайтів на базі React використовуються ключові набори засобів та інструментів для забезпечення продуктивної та ефективної розробки. Ці компоненти і бібліотеки допомагають спростити процес створення і підтримки веб-додатків, роблячи їх більш гнучкими та функціональними. На високому рівні, вони допомагають створити інтерактивні та привабливі інтерфейси, що відповідають сучасним стандартам та очікуванням користувачів, а саме:

ReactDOM: це бібліотека React, яка відповідає за рендеринг React-елементів в DOM (Document Object Model). Вона надає методи для взаємодії з DOM, щоб дозволити React додавати, оновлювати та видаляти елементи на сторінці відповідно до змін у стані додатку. ReactDOM.render() - один з найбільш використовуваних методів ReactDOM, який використовується для відображення React-елементів в кореневий елемент DOM, який вже існує в HTML-структурі сторінки.

React Router: це бібліотека для навігації між різними сторінками веб-додатка, розробленого на React. Вона дозволяє створювати динамічні маршрути, які відображаються в адресному рядку браузера, та реагувати на їхні зміни, переходячи від одного URL до іншого без перезавантаження сторінки.

React Router надає компоненти для визначення маршрутів, такі як Route, Switch, Redirect, Link. Вона дозволяє реалізувати SPA (односторінкові додатки), де зміни сторінок відбуваються без перезавантаження всієї сторінки (рисунок 2.1).

Рисунок 2.1 - Бібліотека React Router

Redux це бібліотека управління станом для додатків, що використовують бібліотеку React. Вона дозволяє ефективно керувати станом додатка і спрощує взаємодію між компонентами.

Основною концепцією Redux є централізоване зберігання стану додатка в одному об'єкті, який називається "store". Стан у Redux є неізменним, тобто він не може бути змінений напряму. Зміни стану відбуваються шляхом відправлення "дій" (actions) до "редюсерів" (reducers), які обробляють ці дії та оновлюють стан.

Redux дозволяє легко відслідковувати та управляти складними станами додатка, зокрема обробляти асинхронні операції, такі як запити до сервера. Вона також спрощує тестування, оскільки логіка додатка відокремлена від компонентів React, що робить код більш прозорим і підтримуваним (рисунок 2.2).

Рисунок 2.2 - Бібліотека Redux

2.2. Платформа FireBase.

Розробка та розгортання веб-додатків, в якої потужний арсенал можливостей що забезпечують зручне та ефективне використання розробниками. Основні можливості Firebase включають зберігання даних в реальному часі, аутентифікацію користувачів, хостинг веб-сторінок, аналітику та інші.

Однією з ключових переваг Firebase [11] є його легкість використання та інтеграції з додатками, зокрема додатками, що використовують бібліотеку React. Цей сервіс дуже просто підключити до свого застосунку.

Зокрема, Firebase може бути використаний для зберігання даних в реальному часі за допомогою Realtime Database або Firestore, для автентифікації користувачів, для розгортання та хостингу веб-сторінок, для відстеження та аналізу поведінки користувачів за допомогою сервісу аналітики Firebase, та іншого.

Загальна ідея Firebase полягає в тому, щоб допомогти розробникам швидко створювати та розгортати веб-додатки з великим функціоналом, не витрачаючи багато часу на налаштування інфраструктури. (рисунок 2.3, 2.4, 2.5).

Рисунок 2.3 - Налаштування Firebase

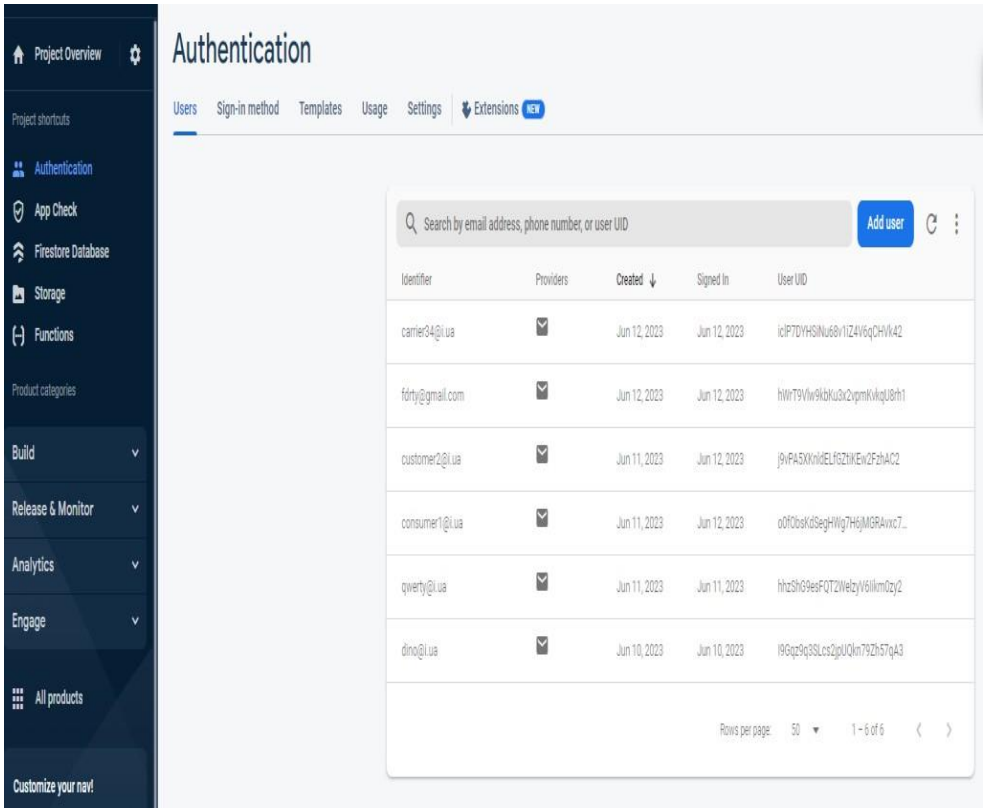


Рисунок 2.4 - Механізм аутентифікації користувачів Firebase

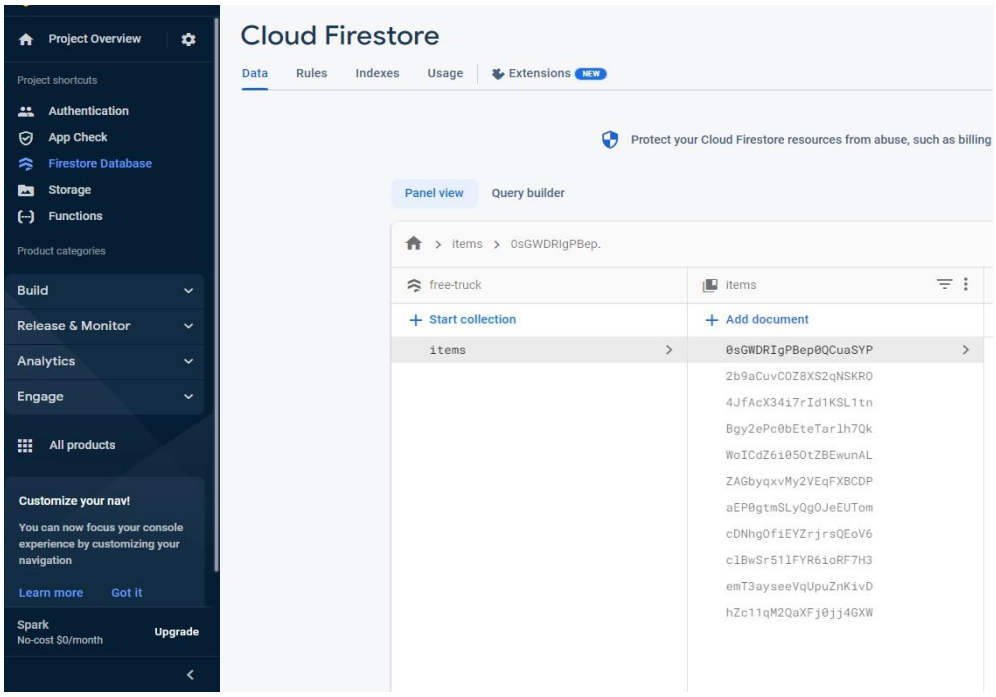


Рисунок 2.5 - Firebase. Сховище данних

Використання цих інструментів та бібліотек полегшує розроблення магазину в серевищі React. і додає безліч корисних можливостей. Завдяки їм розробники можуть працювати з чистим, ефективним та легко керованим кодом, що полегшує

швидкість розробки та підтримки web-застосунків. [13, 14].

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Побудова web-магзину

Одним із важливих завдань, які було вирішено, стало конструювання безпечної системи перевірки автентичності. Для цього використано всі можливості та функціонал платформи Firebase. в поєднані з бібліотекою Redux.

Серед ключових можливостей цього компонента:

- Відтворення окремого інтерфейсу для аутентифікаційних даних користувача.
- Функціонал, що дозволяє користувачам реєструватися на сайті.
- Авторизація у систему при введенні логіна та паролю.
- Відображення інформаційних повідомлень про помилки або про успішне виконання операцій.

Цей компонент виявляється дуже гнучким і можливим для використання в кожному компоненті магазину, де є необхідність перевірки автентичності. Забезпечується зручність взаємодії веб-магазину з користувачами під час процедури автентифікування їх даних.

Ця технологія вирізняється не лише застосуванням передових технологій, але й ефективним використанням найкращих практик у веб-розробці. Він є важливою складовою вашого веб-сайту, пропонуючи вам не тільки потужні можливості автентифікації, а й забезпечуючи високий рівень захисту конфіденційної інформації вашого користувача.

Цей компонент може легко інтегруватися у ваш веб-сайт та надати користувачам інтуїтивно зрозуміле інтерфейс, який дозволить їм безпечно увійти в систему, реєструватися або виходити з облікового запису. Він дозволяє налаштовувати рівні доступу та контролювати права користувачів, що робить його ідеальним варіантом для будь-якого веб-сайту з будь-якими потребами в автентифікації.Завдяки цьому компоненту ваш веб-сайт стане надійним та зручним для користувачів будь-якого рівня. Він допоможе підвищити безпеку та забезпечити

конфіденційність особистих даних користувачів, що робить його незамінним інструментом для будь-якого веб-проекту. (рисунок 3.1, 3.2).

```
import "../../styles/login.css";
import React, { useState } from "react";
import pLogo from "../../Pics/pink-logo.png";
import { useHistory } from "react-router-dom";
import { db, auth } from "../../index";
import { Link } from "react-router-dom";

const Login = (props) => {
  const history = useHistory();
  const [values, setValues] = useState({
    email: "",
    password: "",
    errorMessage: "",
    errorCode: ""
  }); ← #10-15 const [values, setValues] = useState
  let submitNewValues = { ...values };

  const handleChange = (e) => {
    if (e.target.type === "email") {
      submitNewValues.email = e.target.value;
    } else if (e.target.type === "password") {
      submitNewValues.password = e.target.value;
    }
    setValues(submitNewValues);
  }; ← #18-25 const handleChange = (e) =>

  const handleSubmit = (e) => {
    e.preventDefault();
    auth
      .signInWithEmailAndPassword(values.email, values.password)
      .then((userCredential) => {
        let user = userCredential.user;
        db.collection("users")
          .get(user.uid)
          .then((querySnapshot) => {
            submitNewValues.errorMessage = " ";
            history.push("/");
          });
      }) ← #31-39 .then
      .catch((error) => {
        submitNewValues.errorCode = error.code;
        submitNewValues.errorMessage = error.message;
        setValues(submitNewValues);
        console.log(error.code, error.message);
      }); ← #40-45 .catch
  }; ← #27-46 const handleSubmit = (e) =>
```

```
const generatePreviewImgUrl = (file, callback) => {
  const reader = new FileReader();
  reader.onload = () => callback(reader.result);
  reader.readAsDataURL(file);
}; ← #18-22 const generatePreviewImgUrl = (file, callback) =>
let inputChange = { ...values };
const handleImgChange = (e) => {
  const file = e.target.files[0];
  if (!file) {
    return;
  } else if (file.size > 80000) {
    inputChange.sizeErrorMessage =
      "the size of the painting is large. choose less";
  } else {
    inputChange.sizeErrorMessage = "";
    generatePreviewImgUrl(file, (previewImgUrl) => {
      setChoose({ previewImgUrl });
    });
  } ← #31-36 else
  setValues(inputChange);
}; ← #24-38 const handleImgChange = (e) =>

const handleInpChange = (e) => {
  if (e.target.type === "text") {
    inputChange.username = e.target.value;
  } else if (e.target.type === "password") {
    inputChange.password = e.target.value;
  } else if (e.target.type === "email") {
    inputChange.email = e.target.value;
  }
  setValues(inputChange);
}; ← #40-49 const handleInpChange = (e) =>
const handleSignUp = (e) => {
  e.preventDefault();
  auth
    .createUserWithEmailAndPassword(values.email, values.password)
    .then((userCredential) => {
      let user = userCredential.user;
      db.collection("users")
        .doc(user.uid)
        .set({
          username: inputChange.username,
          email: inputChange.email,
          image: chosen.previewImgUrl ? chosen.previewImgUrl : unKnown,
          bag: [],
          favorites: []
        }) ← #58-64 .set
        .then((docRef) => {
          inputChange.errorMessage = "";
          setValues(inputChange);
          history.push("/");
        }) ← #65-69 .then
        .catch((error) => {
          inputChange.errorMessage = error.message;
        });
    }) ← #54-73 .then

    .catch((error) => {
      inputChange.errorMessage = error.message;
      inputChange.errorCode = error.code;
      setValues(inputChange);
    }); ← #75-79 .catch
}; ← #50-80 const handleSignUp = (e) =>
```

Рисунок 3.1 - Операції з авторизацією

```
return (
  <>
    <div className="login-main">
      <div className="login-container">
        <div className="login-logo">
          <div className="logo">
            <a href="/">
              <img src={pLogo} alt={pLogo} />
            </a>
          </div>
        </div>
        <div className="login-form">
          <div className="form">
            <div className="chosen-img">
              <div className="chosen-img-content">
                <img
                  src={chosen.previewImgUrl ? chosen.previewImgUrl : unKnown}
                  alt="user-img"
                />
              </div>
              <span className="error-massage">
                {values.sizeErrorMessage ? values.sizeErrorMessage : ""}
              </span>
            </div>
            <form onSubmit={handleSignUp}>
              <input
                type="file"
                accept=".jpg, .jpeg, .png"
                onChange={handleImgChange}
                id="userImgInp"
              />
              <label htmlFor="userImgInp">Choose image</label>
              <input
                type="text"
                onChange={handleInpChange}
                value={values.username}
                placeholder="username"
                required
              />
              <input
                type="email"
                onChange={handleInpChange}
                value={values.email}
                placeholder="E-mail"
                required
              />
              <input
                type="password"
                onChange={handleInpChange}
                value={values.password}
                placeholder="password"
                required
              />

              <span className="error-massage">
                {values.errorMessage ? values.errorMessage : ""}
              </span>
              <input className="submit" type="submit" value="CREATE" />
            </form>
          </div>
        </div>
        <div className="in-login-create-newAcc-btn">
          <div className="create-newAcc-btn">
            <a href="/login">Log In</a>
          </div>
        </div>
      </div>
    </div>
  </>
); ← #81-150 return
```

```
  return (
    <>
      <div className="login-main">
        <div className="login-container">
          <div className="login-logo">
            <div className="logo">
              <Link to="/">
                <img src={pLogo} alt={pLogo} />
              </Link>
            </div>
          </div>
          <div className="login-form">
            <div className="form">
              <form onSubmit={handleSubmit}>
                <input
                  type="email"
                  value={values.email}
                  onChange={handleChange}
                  placeholder="E-mail"
                  required
                />
                <input
                  type="password"
                  value={values.password}
                  onChange={handleChange}
                  placeholder="Password"
                  required
                />
                <span className="error-massage">
                  {values.errorMessage ? values.errorMessage : ""}
                </span>
                <input className="submit" type="submit" value="LOG IN" />
              </form>
            </div>
          </div>
          <div className="in-login-create-newAcc-btn">
            <div className="create-newAcc-btn">
              <a href="/signup">Create a new account</a>
            </div>
          </div>
        </div>
      </div>
    </>
  );
};

export default Login;
```

Рисунок 3.2 - Відображення коду авторизації

Дані авторизації ефективно використовуються в Redux-Toolkit., доповнення що управляється станом додатків, побудованих на React.

Redux Toolkit - це офіційний набір утиліт, що надає додаткові зручності та спрощення для роботи з Redux. Він має на меті зменшити кількість шаблонного коду, який зазвичай потрібно писати для налаштування магазину Redux та виконання типових завдань, таких як створення дії (actions), редукторів (reducers), та налаштування середовища розробки.

Redux Toolkit пропонує такі корисні інструменти та концепції:

configureStore(): Ця функція дозволяє створити магазин Redux з попередньо підключеними редукторами та middleware, що полегшує налаштування.

createSlice(): Цей метод дозволяє оголосити "зріз" стану (slice), який включає в себе редуктор, дії та початковий стан, що значно спрощує процес створення редукторів.

createAsyncThunk(): Ця функція дозволяє створювати асинхронні дії, що дозволяє легко виконувати асинхронні операції, такі як запити API.

createEntityAdapter(): Цей інструмент допомагає управляти нормалізованими даними в магазині Redux, забезпечуючи зручний доступ до сутностей та їх оновлення.

Redux DevTools Extension Integration: Redux Toolkit інтегрується з Redux DevTools Extension, що дозволяє легко відстежувати та візуалізувати зміни стану додатку.

Спрощена конфігурація магазину: За допомогою функції configureStore(), ви можете швидко створити магазин Redux з попередньо налаштованими редукторами та middleware. Це дозволяє швидко розпочати роботу з Redux без необхідності налаштовувати кожен елемент окремо.

Зручне створення редукторів: Метод createSlice() дозволяє оголосити "зріз" стану (slice), який містить у собі редуктор, дії та початковий стан. Це робить процес створення редукторів більш простим та зрозумілим.

Підтримка асинхронних дій: Функція createAsyncThunk() дозволяє створювати асинхронні дії, що дозволяє легко виконувати асинхронні операції, такі як запити API, та ефективно керувати їхнім станом у Redux.

Управління нормалізованими даними: За допомогою інструменту createEntityAdapter(), ви можете зручно управляти нормалізованими даними в магазині Redux. Це полегшує доступ до сутностей та їх оновлення у вашому додатку.

Інтеграція з Redux DevTools Extension: Redux Toolkit легко інтегрується з Redux DevTools Extension, що дозволяє вам відстежувати та візуалізувати зміни

стану додатку у реальному часі, щоб полегшити відладку та вдосконалення вашого коду [3].

Ці можливості роблять роботу з Redux більш ефективною та продуктивною, допомагаючи вам швидше розробляти та підтримувати додатки (рисунок 3.3 – 3.7).

```
import { applyMiddleware, compose, createStore } from "redux";
import thunk from "redux-thunk";
import reducer from "../reducer/reducer";

const store = createStore(
  reducer,
  compose(
    applyMiddleware(thunk),
    window.__REDUX_DEVTOOLS_EXTENSION__
      ? window.__REDUX_DEVTOOLS_EXTENSION__()
      : (f) => f
  ) ← #7-12 compose
); ← #5-13 const store = createStore

export default store;
```

Рисунок 3.3 - Сховище даних у програмі на React

```
import initialState from "./initialState";
export const SET_BRANDS = "SET_BRANDS";
export const SET_CATEGORY = "SET_CATEGORY";
export const SET_ITEMS = "SET_ITEMS";
export const SET_NEWS_ITEMS = "SET_NEWS_ITEMS";
export const SET_USER = "SET_USER";
export const SET_ITEMS_BY_BRAND = "SET_ITEMS_BY_BRAND";
export const SET_ITEMS_BY_CATEGORY = "SET_ITEMS_BY_CATEGORY";
export const SET_SELECTED_ITEM = "SET_SELECTED_ITEM";
const reducer = (state = initialState, action) => {
  switch (action.type) {
    case SET_BRANDS: {
      return {
        ...state,
        brands: action.payload,
      };
    } ← #12-17 case SET_BRANDS:
    case SET_CATEGORY: {
      return {
        ...state,
        categories: action.payload,
      };
    } ← #18-23 case SET_CATEGORY:
    case SET_ITEMS: {
      return {
        ...state,
        items: action.payload,
      };
    } ← #24-29 case SET_ITEMS:
    case SET_NEWS_ITEMS: {
      return {
        ...state,
        news: action.payload,
      };
    } ← #30-35 case SET_NEWS_ITEMS:
    case SET_USER: {
      return {
        ...state,
        user: action.payload,
      };
    } ← #36-41 case SET_USER:
    case SET_ITEMS_BY_BRAND: {
      return {
        ...state,
        filterByBrand: action.payload,
      };
    } ← #42-47 case SET_ITEMS_BY_BRAND:
    case SET_ITEMS_BY_CATEGORY: {
      return {
        ...state,
        filterByCategory: action.payload,
      };
    } ← #48-53 case SET_ITEMS_BY_CATEGORY:
    case SET_SELECTED_ITEM: {
      return {
        ...state,
        selectedItem: action.payload,
      };
    } ← #54-59 case SET_SELECTED_ITEM:
    default:
      return state;
  } ← #11-62 switch (action.type)
}; ← #10-63 const reducer = (state = initialState, action) =>

export default reducer;
```

Рисунок 3.4 - Функції для обробки інформації у Firestore

```
import firestoreSvc from "../../services/firestoreSvc";    "firestore": Unknown word.
import { SET_BRANDS, SET_CATEGORY, SET_ITEMS, SET_NEWS_ITEMS } from "../reducer";

export const getCategoriesAction = () => async (dispatch) => {
  const categoryState = [];
  const doc = await firestoreSvc.getCategories();    "firestore": Unknown word.
  doc.forEach((category) => {
    categoryState.push({
      categoryId: category.id,
      ...category.data(),
    });
  }); ← #7-12 doc.forEach
  dispatch({
    type: SET_CATEGORY,
    payload: categoryState,
  });
}; ← #4-17 export const getCategoriesAction = () => async (dispatch) =>

export const getBrandsActions = () => async (dispatch) => {
  const brandState = [];
  const doc = await firestoreSvc.getBrands();    "firestore": Unknown word.
  doc.forEach((brand) => {
    brandState.push({
      brandId: brand.id,
      ...brand.data(),
    });
  }); ← #22-27 doc.forEach
  dispatch({
    type: SET_BRANDS,
    payload: brandState,
  });
}; ← #19-32 export const getBrandsActions = () => async (dispatch) =>

export const getItemsActions = () => async (dispatch) => {
  const itemsState = [];
  const querySnapshot = await firestoreSvc.getItems();    "firestore": Unknown word.
  querySnapshot.forEach((item) => {
    itemsState.push({
      itemId: item.id,
      ...item.data(),
    });
  }); ← #37-42 querySnapshot.forEach
  dispatch({
    type: SET_ITEMS,
    payload: itemsState,
  });
}; ← #34-47 export const getItemsActions = () => async (dispatch) =>

export const getNewItemsActions = () => async (dispatch) => {
  const newsState = [];
  const querySnapshot = await firestoreSvc.getNewItems();
  querySnapshot.forEach((item) => {
    newsState.push({
      itemId: item.id,
      ...item.data(),
    });
  }); ← #52-57 querySnapshot.forEach
  dispatch({
    type: SET_NEWS_ITEMS,
    payload: newsState,
  });
} ← #49-62 export const getNewItemsActions = () => async (dispatch) =>
```

Рисунок 3.5 - Взаємодія Firebase

```
export const selectUser = (state) => state.user;
export const selectBrands = (state) => state.brands;
export const selectCategories = (state) => state.categories;
export const selectItems = (state) => state.items;
export const selectNews = (state) => state.news;
export const selectCategoryById = (id) => (state) =>
  state.categories.find((category) => category.categoryId === id);
export const selectBrandById = (id) => (state) =>
  state.brands.find((brand) => brand.brandId === id);
export const selectItemById = (id) => (state) =>
  state.items.find((item) => item.itemId === id);
export const selectBagCount = (state) =>
  state.user.data ? state.user && state.user.data.bag.length : "";
```

Рисунок 3.6 - Робота з Firebase для вибору даних

```
1  import { db } from "..";
2
3  const firestoreSvc = {    "firestore": Unknown word.
4    async getCategories() {
5      return db.collection("category").get();
6    },
7    async getBrands() {
8      return db.collection("brands").get();
9    },
10   async getItems() {
11     return db.collection("items").get();
12   },
13   async getNewItems() {
14     return db.collection("items").where("status", "==", "new").get();
15   },
16 }; ← #3-16 const firestoreSvc =
17
18 export default firestoreSvc;    "firestore": Unknown word.
19
20
21
```

Рисунок 3.7 - Витяг даних з Firestore

Створено спеціальний елемент каруселі, що відтворює послідовність продуктів на сайті. Цей елемент побудований на основі бібліотеки react-multi-carousel та зберігається у станах веб магазину за допомогою Redux’.

React-Multi-Carousel - це бібліотека для React, яка надає можливість легко створювати каруселі або слайдери з декількома елементами. Ось деякі основні функціональні можливості цієї бібліотеки:

Множинні каруселі: React-Multi-Carousel дозволяє створювати каруселі з декількома елементами, які можуть бути прокручуваними горизонтально або вертикально.

Повне керування: Ви маєте повний контроль над каруселлю, включаючи можливість зупиняти автоматичну прокрутку, відтворення, паузу та інші функції.

Сповіщення про події: Бібліотека надає можливість обробляти різні події каруселі, такі як клік на елемент, початок або завершення прокрутки.

Підтримка кастомізації: Ви можете легко кастомізувати вигляд каруселі, використовуючи CSS або власні компоненти для елементів каруселі.

Підтримка клавіатурного керування: Ви можете взаємодіяти з каруселлю за допомогою клавіш клавіатури, наприклад, використовуючи клавіші вліво/вправо для перемикання слайдів.

React-Multi-Carousel. - це потужний інструмент для створення динамічних і адаптивних каруселей з декількома елементами в додатках на базі React (рисунок 3.8).

Рисунок 3.8 - Код каруселі товарів

Кошик - це незамінна складова нашого веб-додатка, яка відповідає за покупки товару. Він створений з урахуванням потреб авторизованих користувачів і має на меті забезпечити зручність та ефективність процесу покупок.

У нього включено багато корисних функцій, серед яких перегляд обраних букетів у кошику, розрахунок загальної вартості обраного асортименту та зручний інтерфейс оплати. Щоб стилізувати та налаштувати інтерфейс, ми використовуємо бібліотеку Material-Ui а також власну стилізацію.

Кошик взаємодіє з Redux, використовуючи хук useSelector для отримання інформації про користувача та його корзину. Із стану bag зберігається інформація

про товари у корзині, а totalPrice відповідає за розрахунок сумарної вартості замовлення. Для автоматичного оновлення станів ми використовуємо useEffect, щоб коректно реагувати на зміни.

У випадку, якщо користувач не авторизований, його перекидує у вікно авторизації для забезпечення безпеки та захисту конфіденційності. BugItems, з свого боку, відображає кожну позицію у кошику і забезпечує його коректну інтеграцію.

При цьому, BugItems використовує бібліотеку Produces яка зберігає дані кошику при пергрузці системи. Кошик - це не лише спосіб відображення корзини, але й інструмент, що полегшує і оптимізує процес покупок, надаючи користувачеві зручність та ефективність (рис 3.9, 3.10).

```jsx
const Bag = () => {
  const user = useSelector(selectUser);
  const classes = useStylesForBag();
  const history = useHistory();
  const [bag, setBag] = useState([]);
  const [totalPrice, setTotalPrice] = useState([]);
  const emptyBag = (
    <div className={classes.emptyBag}>
      <h1 className={classes.emptyBagTitle}>Кошик порожній</h1>    "Кошик": Unknown word.
    </div>
  ); ← #18-22 const emptyBag =

  useEffect(() => {
    if (user.data) {
      let clonedUserBag = produce(user, (draftUser) => {
        return draftUser.data.bag;
      });
      setBag(clonedUserBag);
      let setPrice = 0;
      clonedUserBag.map((item) => {
        return (setPrice += +item.price);
      });
      setTotalPrice(setPrice);
    } else {
      history.push("/");
    }
  }, [user, history]); ← #24-38 useEffect
  return (
    <div className={classes.bagComponent}>
      <div className={classes.bagHeader}>
        <h1> Кошик покупок{ } </h1>    "Кошик": Unknown word.
      </div>
      <div className={classes.paper}>
        <div className={`${classes.leftContent} for-scroll`}>
          <div className={classes.bagItems}>
            {bag.length
              ? bag.map((item, i) => (
                <BagItem key={item.itemId} ind={i} {...item} />
              ))
              : emptyBag} ← #47-51 <div className={classes.bagItems}>
          </div>
        </div>
        <Card className={classes.card} variant="outlined">
          <CardContent>
            <Typography variant="h5" component="h2">
              Огляд замовлення    "Огляд": Unknown word.
            </Typography>
            <br />
            <Typography component="p">Доставка - безкоштовна</Typography>    "Доставка": Unknown word.
            <br />
            <Typography component="p">Загальна вартість - ${totalPrice}</Typography>    "Загальна": Unknown word.
          </CardContent>
          <CardActions style={{ border: "1px solid black" }}>
            <Link to="/bag/payment">
              <Button size="small">Натисніть, щоб замовити</Button>    "Натисніть": Unknown word.
            </Link>
          </CardActions>
        </Card>
      </div>
    </div>
  ); ← #39-72 return
}; ← #12-73 const Bag = () =>

export default Bag;
```

Рисунок 3.9 - Кошик магазину

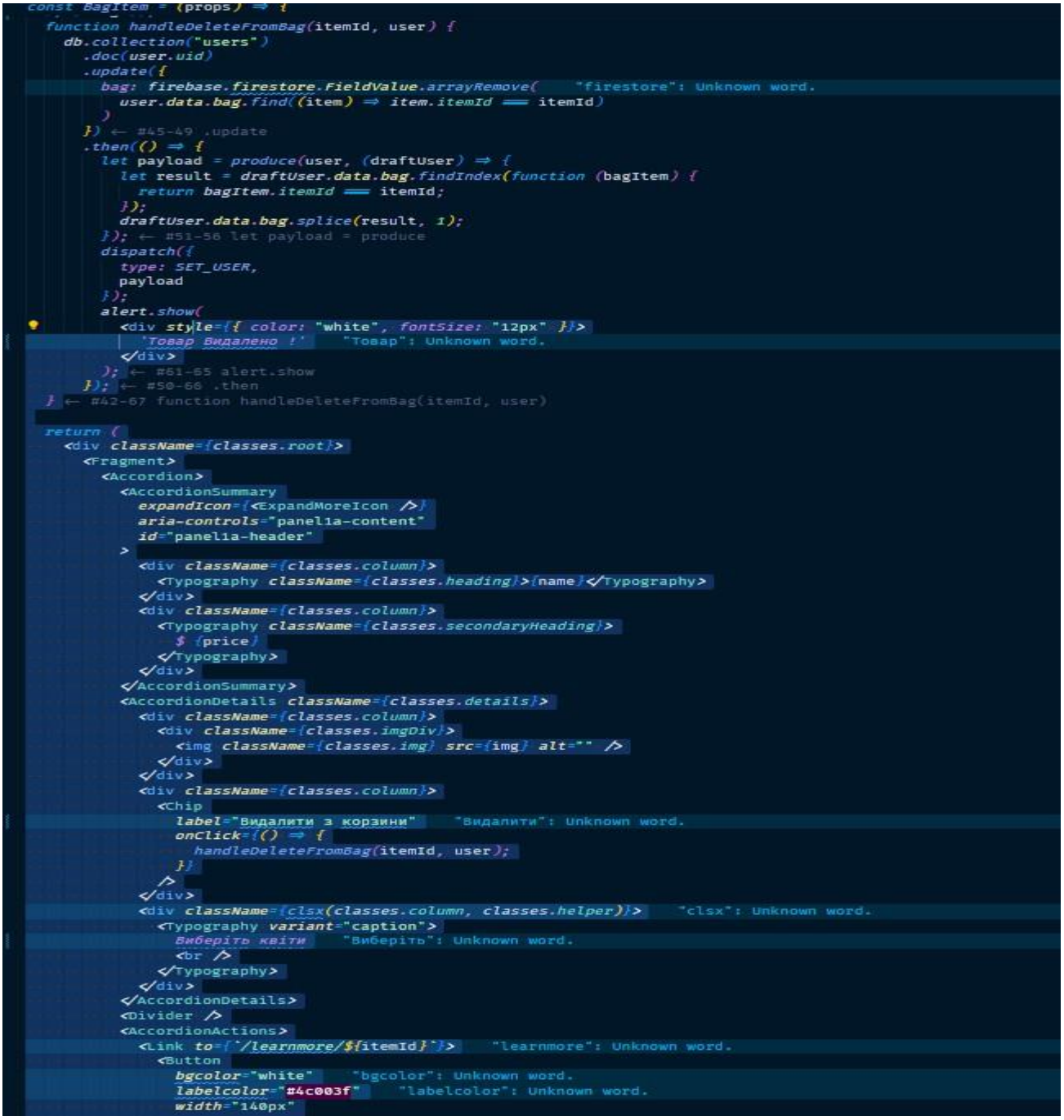


Рисунок 3.10 - Продукти в кошику

У веб-магазині створена система оплати, яка відіграє ключову роль у веб-додатку, де він відповідає за сторінку оформлення платежів.

Цей система реалізована за допомогою додаткової бібліотеки react-credit-cards., що створює форму для заповнення платіжної інформаці під час оплати покупок

Бібліотека react-credit-cards - це компонент для React., який дозволяє створювати інтерфейси для візуалізації та введення даних кредитних карт. Вона

надає зручні і гнучкі інструменти для відображення різних аспектів кредитних карт, таких як номер карти, термін дії, ім'я власника та CVV-код.

Основні можливості бібліотеки react-credit-cards включають:

Візуалізація кредитних карт: За допомогою компонентів цієї бібліотеки можна візуалізувати кредитну карту з реалістичним виглядом, включаючи передню та задню сторони карти.

Введення даних карти: Компоненти дозволяють користувачеві вводити дані кредитної картки, такі як номер карти, термін дії, ім'я власника та CVV-код.

Валідація даних карти: Бібліотека може надавати можливість перевіряти коректність введених даних кредитної картки.

Кастомізація вигляду: Ви можете кастомізувати вигляд компонентів бібліотеки, включаючи стилізацію, розмір та поведінку.

Підтримка різних типів карт: Бібліотека підтримує візуалізацію та введення даних різних типів кредитних карт, включаючи Visa, Mastercard, American Express та інші.Динамічна зміна даних карти: Ви можете легко змінювати дані кредитної картки, що відображаються у компонентах бібліотеки, наприклад, під час введення нових даних або після підтвердження операції.

Бібліотека react-credit-cards є корисним інструментом для розробників, які потребують можливостей відображення та введення даних кредитних карт у своїх додатках на базі React (рисунок 3.11).

```
import React, { useState } from "react";
import Cards from "react-credit-cards";
import "react-credit-cards/es/styles-compiled.css";
import {
  Button as ButtonMaterial,
  Dialog,
  DialogActions,
  DialogContent,
  DialogContentText,
} from "@material-ui/core"; ← #4-10 import
import { useDispatch, useSelector } from "react-redux";
import "../../styles/payment.css";
import Button from "../shared/Button";
import { selectUser } from "../../selectors/firebase";
import produce from "immer";     "immer": Unknown word.
import { db } from "../../";
import { SET_USER } from "../../reducer/reducer";
import { useHistory } from "react-router";
const Payment = () => {
  const user = useSelector(selectUser);
  const dispatch = useDispatch();
  const history = useHistory();
  const [open, setOpen] = useState(false);
  const [data, setData] = useState({
    cvc: "",
    expiry: "",
    name: "",
    number: "",
  }); ← #24-29 const [data, setData] = useState
  const handleInputChange = (e) => {
    setData({
      ...data,
      [e.target.name]: e.target.value,
    });
  }; ← #30-35 const handleInputChange = (e) =>
```

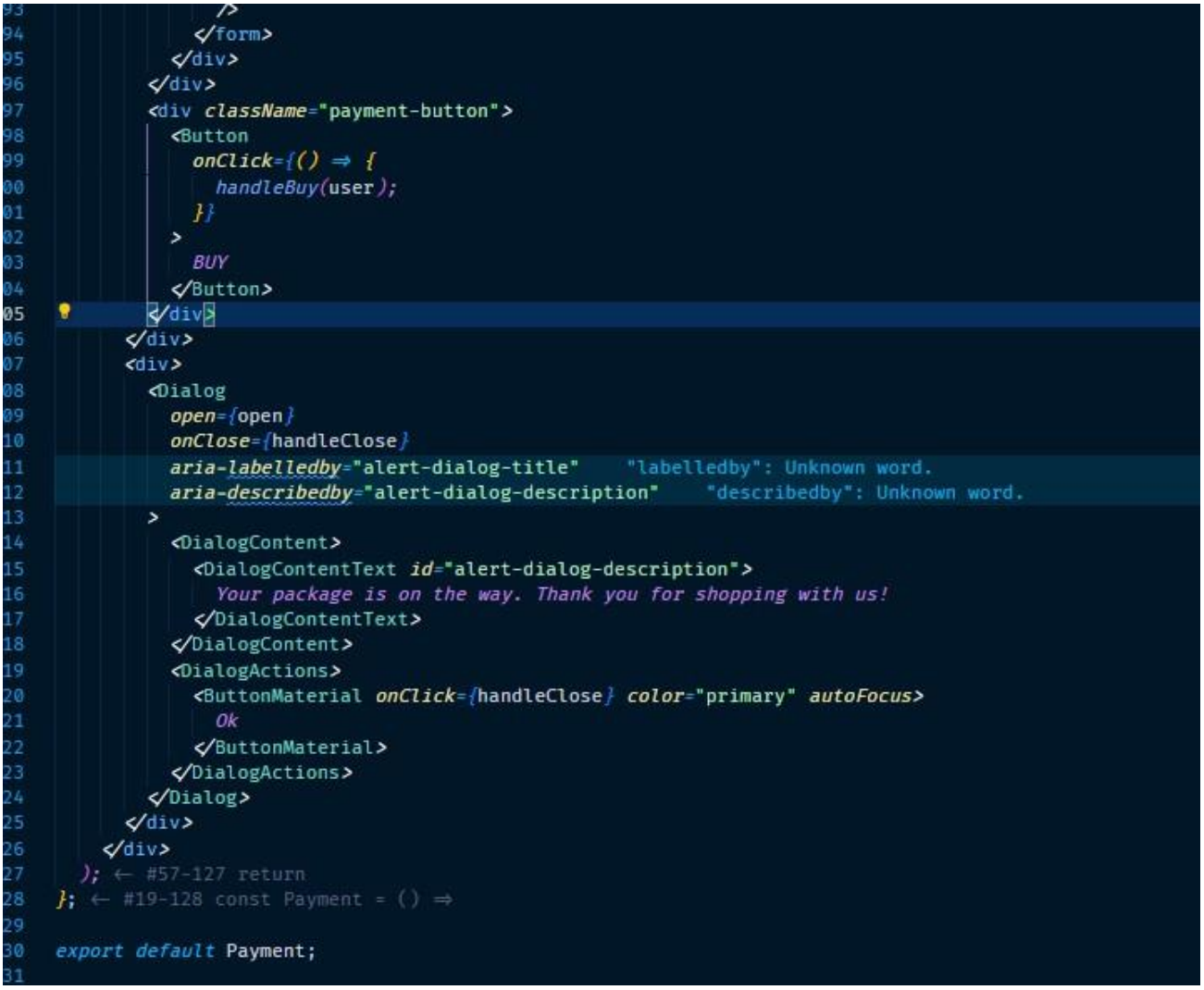
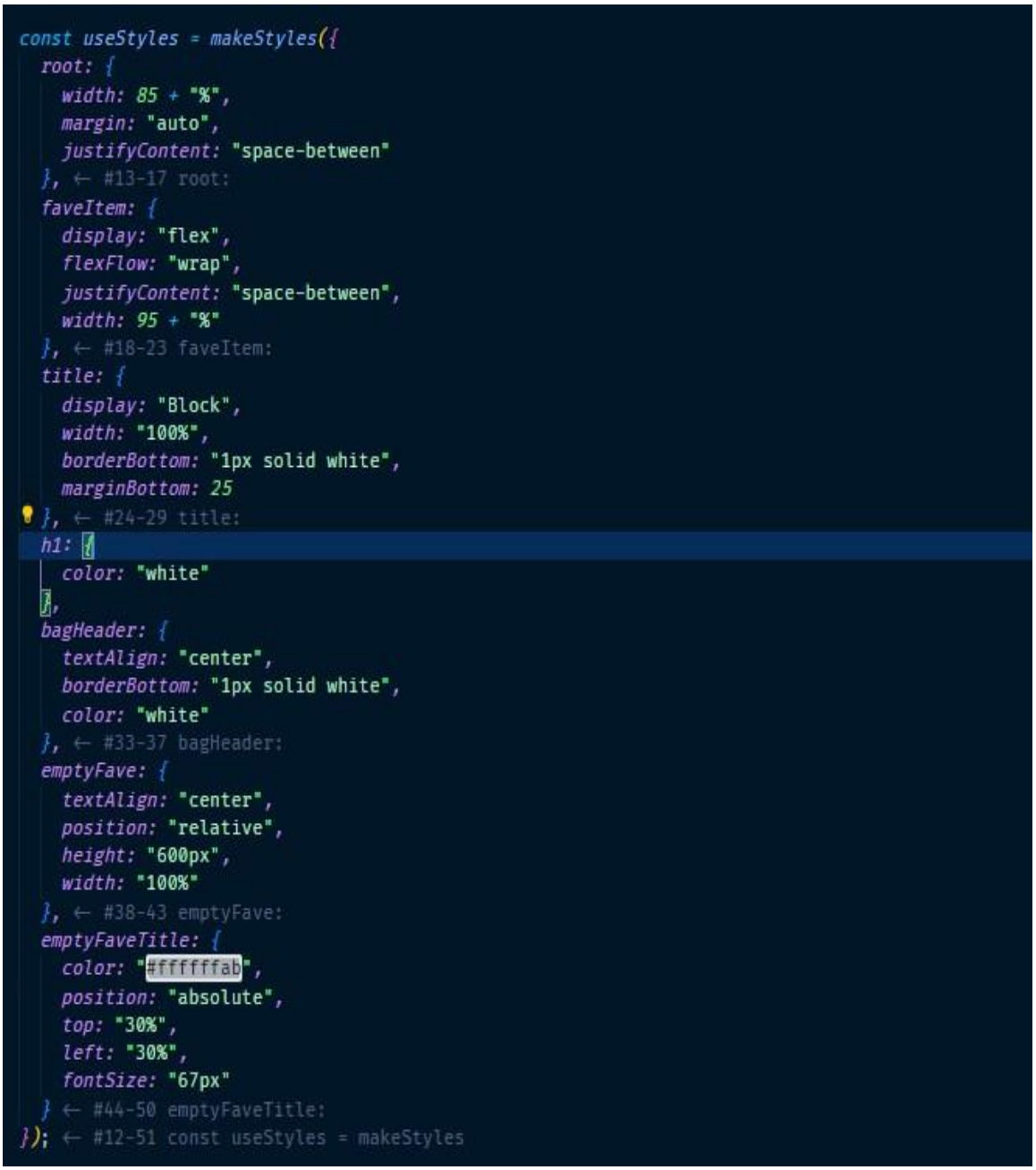


Рисунок 3.11 - Реалізація оплати в магазині

Для оформлення інтерфейсу користувача використовується Material-UI, що ґрунтується на концепції дизайну Google Material та пропонує широкий спектр готових компонентів і стилів. Для керування зовнішнім виглядом компонентів і розділів сторінки використовується makeStyles. Цей інструмент дозволяє визначати стилі для різного функціональних компонентів ситеми, що забезпечує гнучкий підхід до стилізації інтерфейсу.

Material-UI також забезпечує можливість створювати різні інтерактивні елементи як поява модального вікна для відтворення приємного візуалу вебмагазину[12].

Використання Material-UI спрощує процес створення стильного та функціонального інтерфейсу для React-додатків, дозволяючи швидко реалізувати рішення згідно з найсучаснішими вимогами дизайну (рисунок 3.12).



```
const useStyles = makeStyles({
  root: {
    width: 85 + "%",
    margin: "auto",
    justifyContent: "space-between"
  }, ← #13-17 root:
  faveItem: {
    display: "flex",
    flexFlow: "wrap",
    justifyContent: "space-between",
    width: 95 + "%"
  }, ← #18-23 faveItem:
  title: {
    display: "Block",
    width: "100%",
    borderBottom: "1px solid white",
    marginBottom: 25
  }, ← #24-29 title:
  h1: {
    color: "white"
  },
  bagHeader: {
    textAlign: "center",
    borderBottom: "1px solid white",
    color: "white"
  }, ← #33-37 bagHeader:
  emptyFave: {
    textAlign: "center",
    position: "relative",
    height: "600px",
    width: "100%"
  }, ← #38-43 emptyFave:
  emptyFaveTitle: {
    color: "#ffffffab",
    position: "absolute",
    top: "30%",
    left: "30%",
    fontSize: "67px"
  } ← #44-50 emptyFaveTitle:
}); ← #12-51 const useStyles = makeStyles
```

Рисунок 3.12- Стилізація Material-UI

3.2. Тестування проєкту

Відображення робочих вікон діючого застосунку представлено на рисунках 3.13 – 3.22.

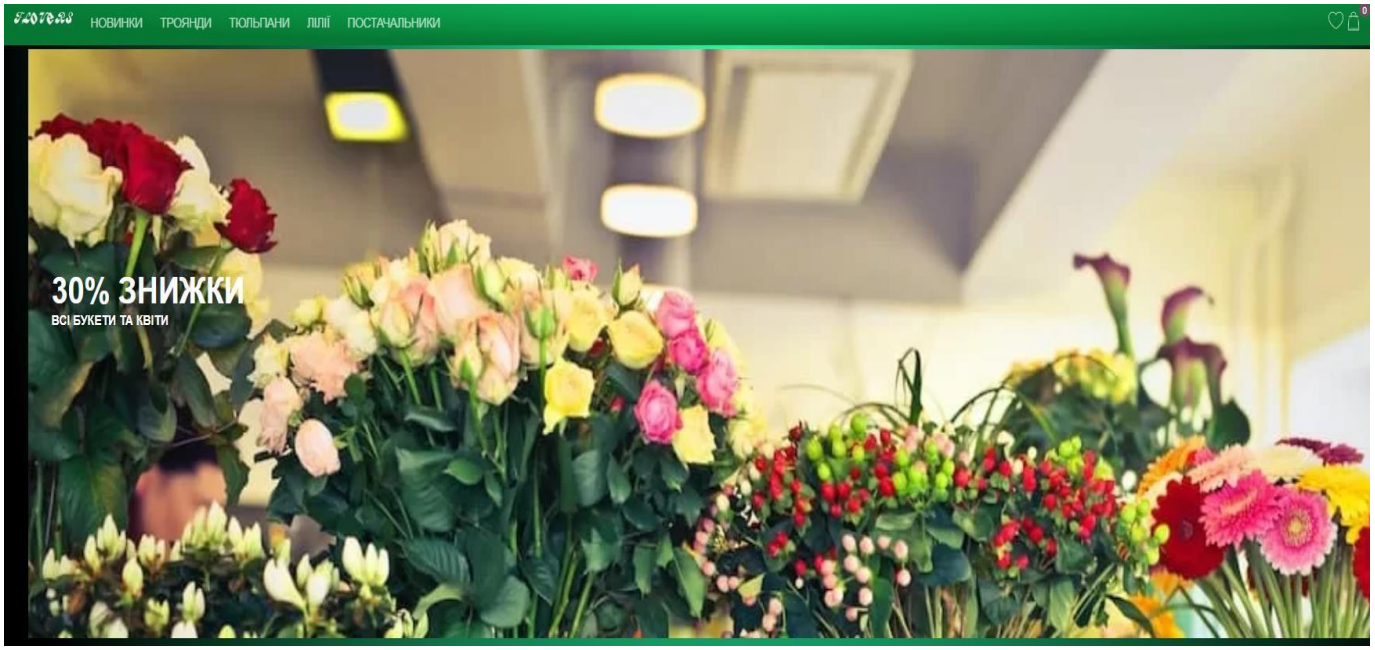


Рисунок 3.13 - Хедер та навігаційне меню веб-магазину

.

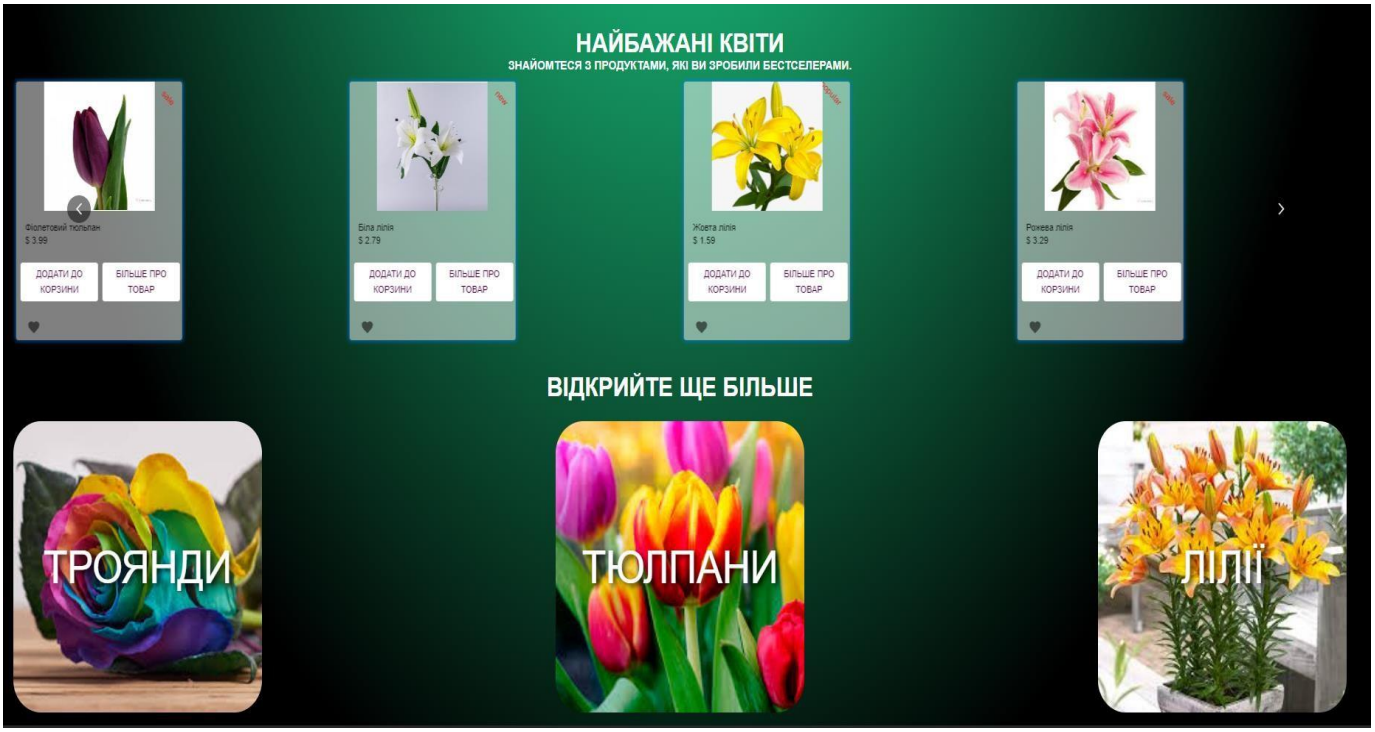
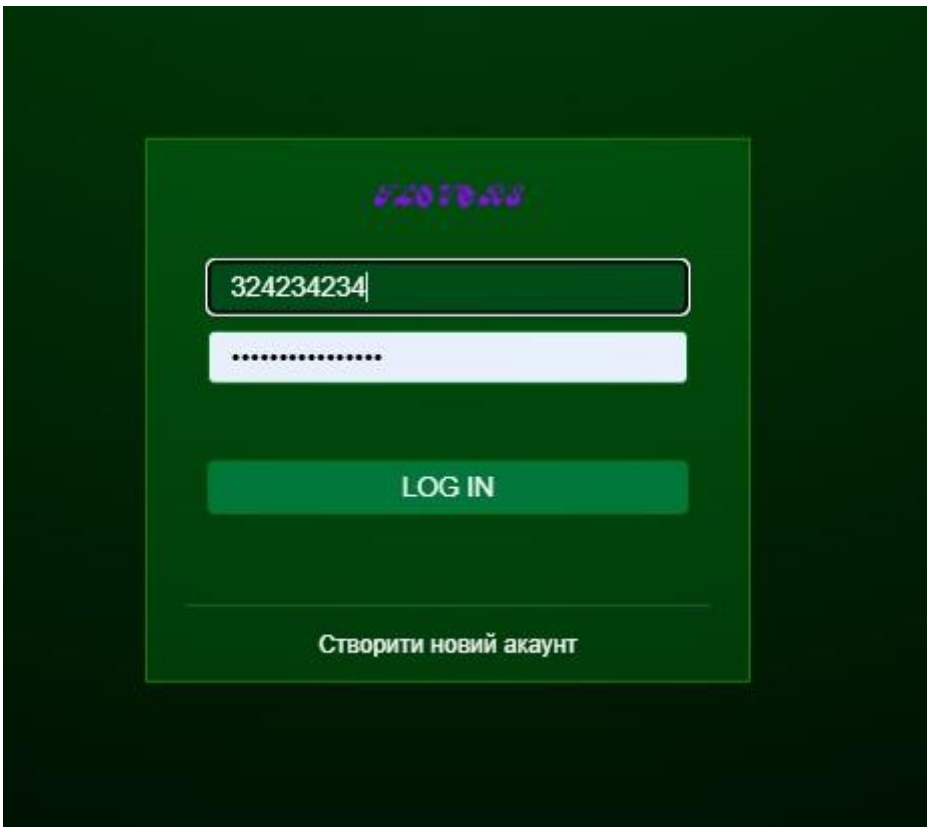


Рисунок 3.14 - Вибір категорій та найбажані товари на основній сторінці


324234234
LOG IN
Створити новий акаунт

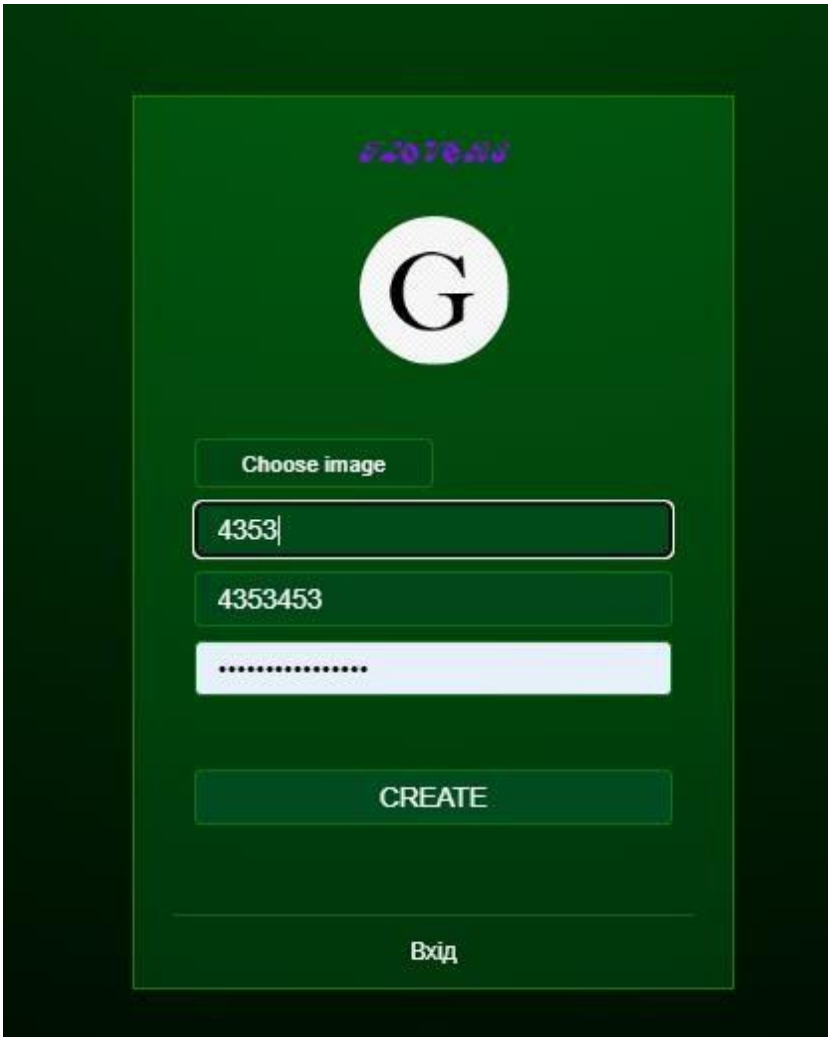

G
Choose image
4353
4353453
CREATE
Вхід

Рисунок 3.15 - Логінізація та реєстрація користувачів

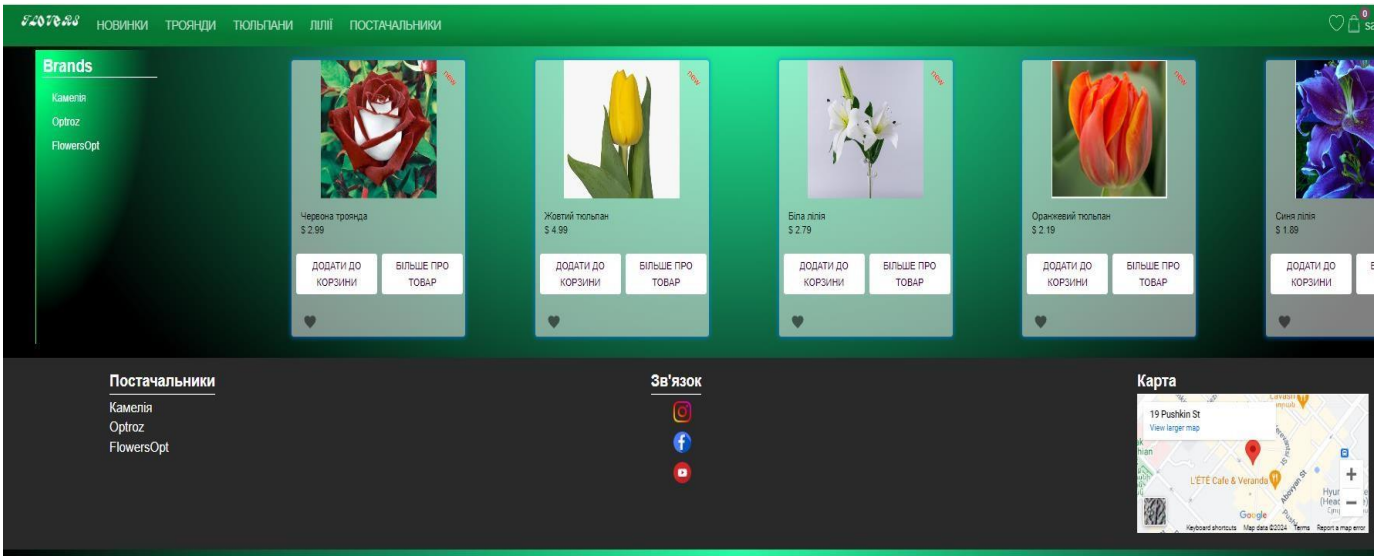


Рисунок 3.16 - Товари по категорії

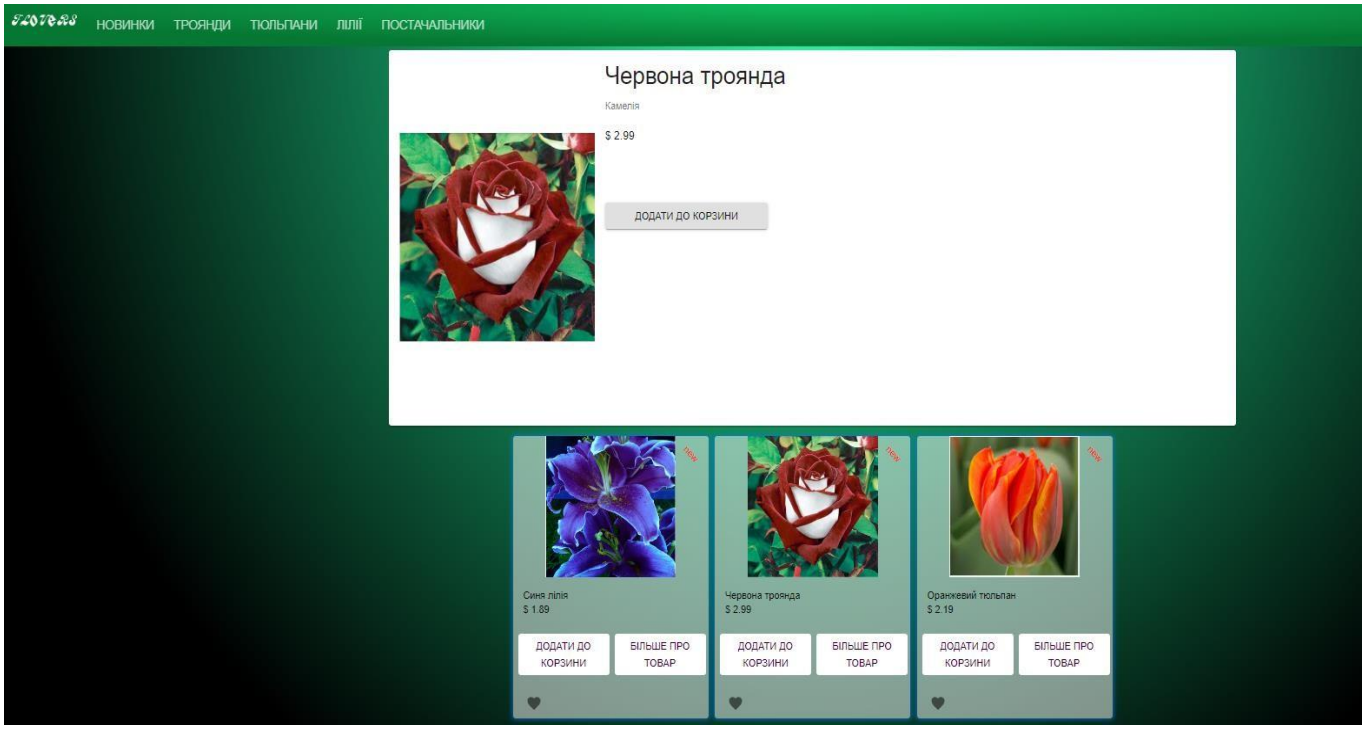


Рисунок 3.17 - Інформація про товар

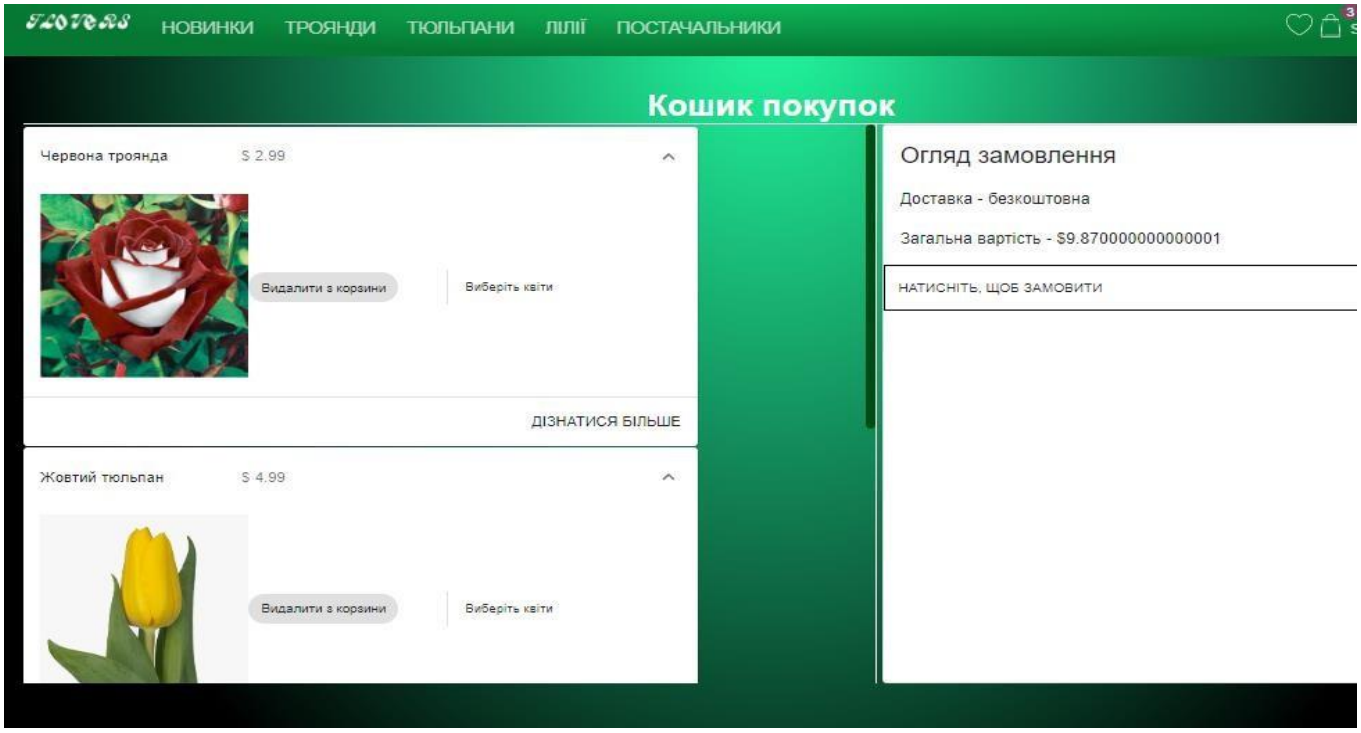


Рисунок 3.18 - Кошик

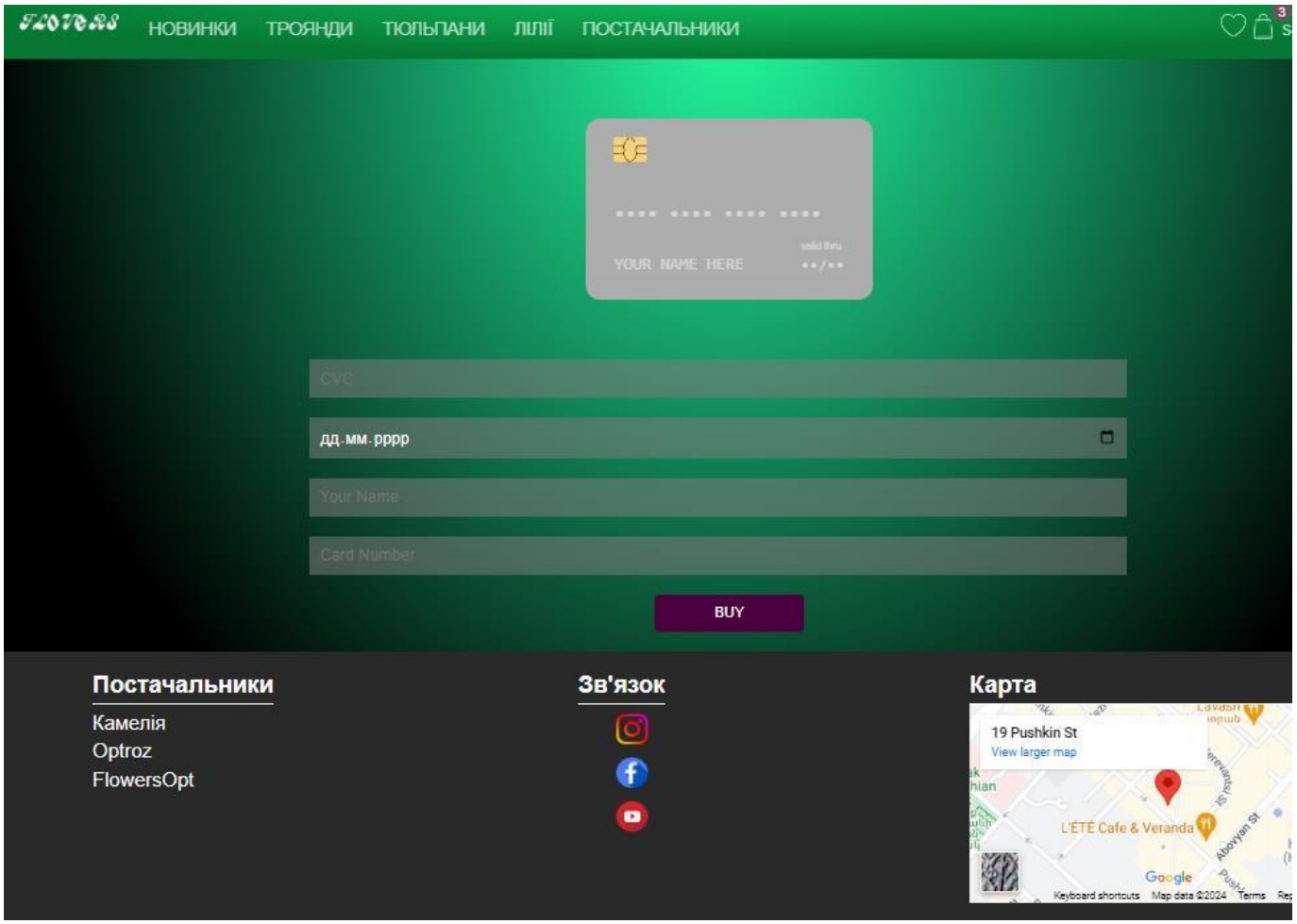


Рисунок 3.19 - Оплати товару

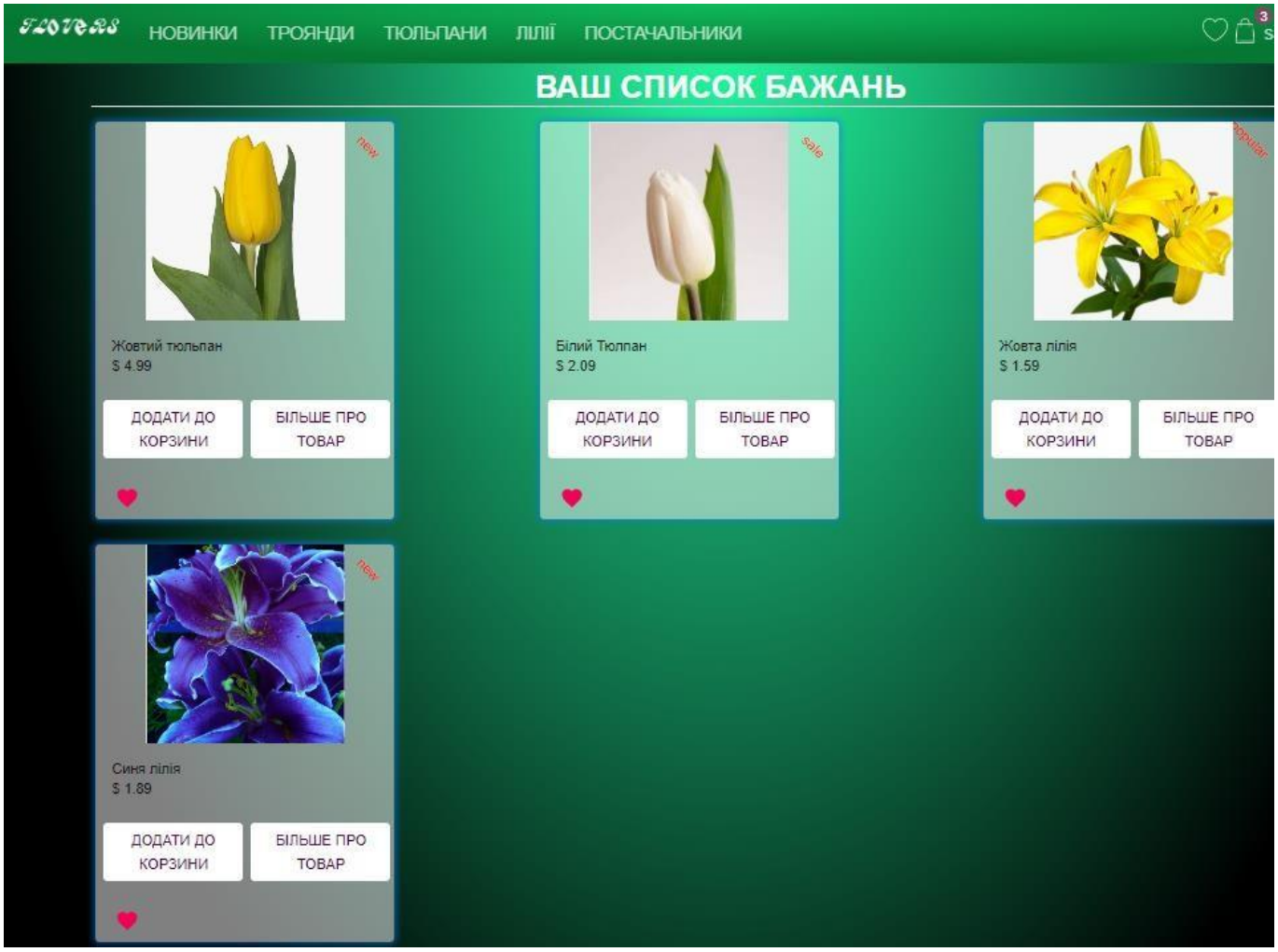


Рисунок 3.20 - Список сподобавшихся товарів

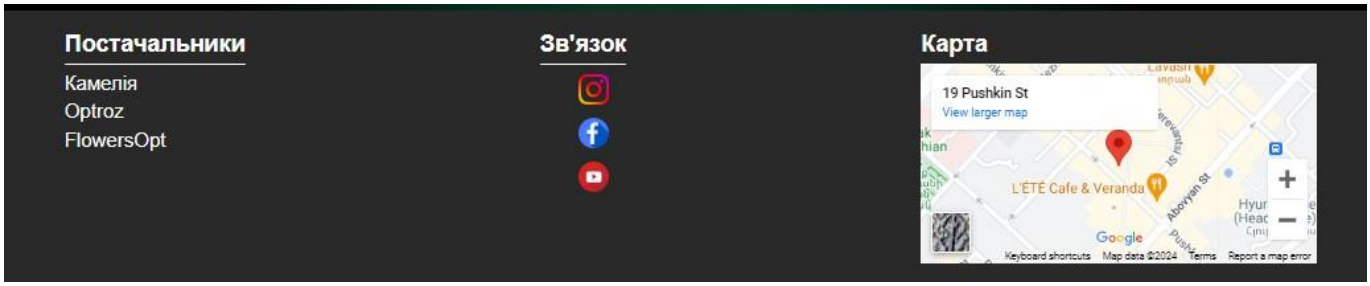


Рисунок 3.21 - Підвал магазину

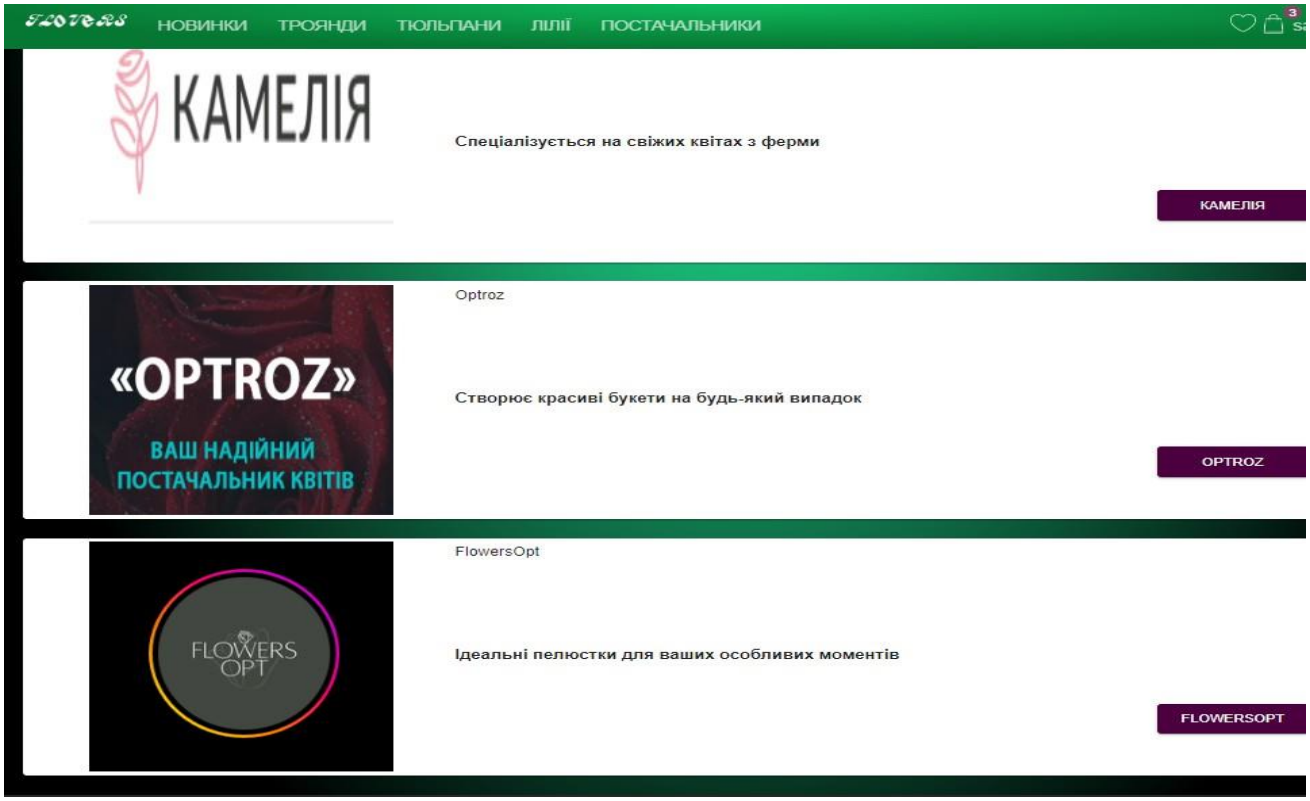


Рисунок 3.22 - Постачальники

ВИСНОВКИ

Веб-магазин квітів "Flowers" представляє собою відмінний приклад сучасного та інноваційного підходу до електронної комерції. Він поєднує в собі ефективний інтерфейс користувача з високоякісними послугами та безпечною системою оплати, надаючи клієнтам зручність та надійність у купівлі квітів онлайн.

Завдяки використанню передових технологій, таких як React, Redux та Firebase, "Flowers" може швидко реагувати на потреби своїх клієнтів та надавати їм персоналізований досвід покупок. Крім того, система автентифікації користувачів забезпечує безпеку та конфіденційність даних, що є важливим аспектом у сфері онлайн-торгівлі.

Майбутні перспективи "Flowers" включають постійне вдосконалення та розширення функціональності, щоб надати клієнтам ще більше можливостей та комфорту під час покупок. Розвиток інтернет-магазину "Flowers" відображає важливий тренд у сучасній економіці, де електронна торгівля стає все більш популярною та доступною для широкого кола споживачів.

Додатково, важливе значення має можливість подальшого розвитку проекту. Створення розгалуженої системи веб-магазинів з більшим різновидом асортименту, можуть стати ключовим напрямком для подальшого розвитку "Flowers".

Завдяки цьому проекту отримано необхідну практику у створенні веб-магазину, поглиблено знання про сучасні технології та професійні навички. "Flowers" не лише забезпечує якісний сервіс для своїх клієнтів, але й є відмінним прикладом успішного поєднання технологій та бізнесу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Robin Wieruch, “The Road to React” - Leanpub, 2020, 300 сторінок.
2. Alex Banks, Eve Porcello, “Learning React: Functional Web Development with React and Redux” - O’Reilly Media, 2017, 350 сторінок.
3. Kirupa Chinnathambi, “Learning React: A Hands-On Guide to Building Web Applications Using React and Redux (2nd Edition)” - Addison-Wesley Professional, 2018, 400 сторінок.
4. Bonnie Eisenman, “Learning React Native: Building Native Mobile Apps with JavaScript” - O’Reilly Media, 2015, 250 сторінок.
5. Azat Mardan - “React Quickly: Painless Web Apps with React, JSX, Redux, and GraphQL” - Manning Publications, 2017, 528 сторінок.
6. React. JavaScript-бібліотека для створення користувацьких інтерфейсів [Електронний ресурс] – Режим доступу: Url: https://uk.legacy.reactjs.org (дата звернення: 20.03.2024).
7. React Js Best Practices for startups – [Електронний ресурс] – Режим доступу: https://rencybeth.medium.com/7-reactjs-best-practices-2546c8ec78c7 (дата звернення: 15.04.2024).
8. What is React Suite? – [Електронний ресурс] – Режим доступу: https://rsuitejs.com/guide/introduction (дата звернення: 15.04.2024).
9. Документація React: [Електронний ресурс] – Режим доступу: https://reactjs.org/docs/getting-started.html (дата звернення: 30.05.2024).
10. Документація Redux: [Електронний ресурс] – Режим доступу: https://redux.js.org/introduction/getting-started (дата звернення: 10.05.2024)
11. Документація Firebase: [Електронний ресурс] – Режим доступу: https://firebase.google.com/docs (дата звернення: 18.05.2024).
12. Офіційний сайт Material-UI: [Електронний ресурс] – Режим доступу: https://mui.com/ (дата звернення: 20.05.2024).

13. Франко Ребеза, "Firebase Cookbook: Over 70 recipes to help you create real-time web and mobile applications with Firebase" - Packt Publishing, 2017, 358 сторінок.
14. Roldán C. S., “React 18 Design Patterns and Best Practices” - Packt Publishing, 2023, 524 сторінок.
15. Mike McDonald, "Firebase Essentials: Get to grips with Firebase's real-time database, authentication, and storage solutions" - Packt Publishing, 2016, 294 сторінок.

ДОДАТКИ

Код проекту

```
import React from "react";
import ReactDOM from "react-dom";
import App from "./App";
import firebase from "firebase/app";
import "firebase/auth";
import "firebase/storage";
import AlertTemplate from "react-alert-template-basic";
import { transitions, positions, Provider as AlertProvider } from "react-alert";
import { Provider } from "react-redux";
import store from "./reducer/indexStore";

const firebaseConfig = {
  apiKey: "AIzaSyBdJeXtbTxrpJH1fialzApIUVN84V7Ioyg",    "Txrp": Unknown word.
  authDomain: "cosmetics-91882.firebaseapp.com",    "firebaseapp": Unknown word.
  projectId: "cosmetics-91882",
  storageBucket: "cosmetics-91882.appspot.com",    "appspot": Unknown word.
  messagingSenderId: "134093997606",
  appId: "1:134093997606:web:ab96e1e384a17969aec8e7"
}; ← #12-19 const firebaseConfig =

firebase.initializeApp(firebaseConfig);

export const db = firebase.firestore();    "firestore": Unknown word.
export const auth = firebase.auth();
export const storage = firebase.storage();
const options = {
  position: positions.BOTTOM_RIGHT,
  timeout: 3000,
  offset: "30px",
  transition: transitions.FADE,
  color: "white",
  background: "white"
}; ← #26-33 const options =

ReactDOM.render(
  <React.StrictMode>
    <AlertProvider template={AlertTemplate} {...options}>
      <Provider store={store}>
        <App />
      </Provider>
    </AlertProvider>
  </React.StrictMode>,
  document.getElementById("root")
```

```
import { useEffect } from "react";
import New from "./component/section/New";
import Brands from "./component/section/Brands";
import Nav from "./component/menu/Nav";
import Main from "./component/section/Main";
import Footer from "./component/section/Footer";
import Login from "./component/menu/Login";
import Signup from "./component/menu/Signup";
import Brand from "./component/section/Brand";
import { useDispatch } from "react-redux";
import Bag from "./component/menu/Bag";
import Payment from "./component/section/Payment";
import Category from "./component/section/Category";
import LearnMore from "./component/section/LearnMore";
import "./styles/responsive.css";
import {···
} from "react-router-dom"; ← #16-21 import
import {···
} from "./reducer/asyncActions/firestoreActions";    "firestore": Unknown word. ← #22-27 i
import Favorites from "./component/menu/Favorites";
export default function App() {
  const dispatch = useDispatch();
  useEffect(() => {
    dispatch(getBrandsActions());
    dispatch(getCategoriesAction());
    dispatch(getItemsActions());
    dispatch(getNewItemsActions());
  }, [dispatch]); ← #31-36 useEffect

  return (
    <>
      <Router>
        <Nav />
        <Switch>
          <Route path="/new" component={New} />
          <Route exact path="/categories/:categoryId" component={Category} />
          <Route exact path="/brands" component={Brands} />
          <Route path="/brands/:brandId" component={Brand} />
          <Route exact path="/bag" component={Bag} />
          <Route path="/bag/payment" component={Payment} />
          <Route exact path="/learnmore/:itemId" component={LearnMore} />    "learnmore": U
          <Route path="/login" component={Login} />
          <Route path="/signup" component={Signup} />
          <Route exact path="/" component={Main} />
          <Route exact path="/myFavorites" component={Favorites} />
          <Route exact path="/home" component={Main}>
            <Redirect to="/" />
          </Route>
          <Route path="*">
            <Redirect to="/" />
          </Route>
        </Switch>
        <Footer />
      </Router>
```

```jsx
import Carousel from "react-multi-carousel";
import "react-multi-carousel/lib/styles.css";
import { useSelector } from "react-redux";
import { selectItems } from "../../selectors/firebase";
import Item from "./Item";
import "../../styles/carousel.css";

const CarouselPrint = () => {
  const responsive = {
    desktop: {
      breakpoint: { max: 3000, min: 1024 },
      items: 4,
      slidesToSlide: 1,
    }, ← #10-14 desktop:
    tablet: {
      breakpoint: { max: 1024, min: 464 },
      items: 2,
      slidesToSlide: 2,
    }, ← #15-19 tablet:
    mobile: {
      breakpoint: { max: 464, min: 0 },
      items: 1,
      slidesToSlide: 1,
    }, ← #20-24 mobile:
  }; ← #9-25 const responsive =
  const items = useSelector(selectItems);
  let arrayItems = [...items].slice(0, 9);
  return (
    <div className="carousel-main">
      <h1 className="carousel-title">НАЙБАЖАНІ К
      <h3 className="carousel-subtitle">
        Знайомтеся з продуктами, які ви зробили бестселерами.
      </h3>
      <div className="slider">
        <Carousel
          responsive={responsive}
          infinite
          autoPlay
          autoPlaySpeed={3000}
          keyBoardControl
          transitionDuration={100}
          customTransition="transform 1000ms ease-in-out"
          focusOnSelect
          containerClass="carousel-container"
          dotListClass="custom-dot-list-style"
          itemClass="carousel-item-padding-40-px"
        >
          {arrayItems.map((item, i) => {
            return (
              <Item key={item.itemId} ind={i} {...item} showThumbs={false} />
            );
          })} ← #40-52 >
        </Carousel>
      </div>
    </div>
  ); ← #28-56 return
}; ← #8-57 const CarouselPrint = () =>
export default CarouselPrint;
```

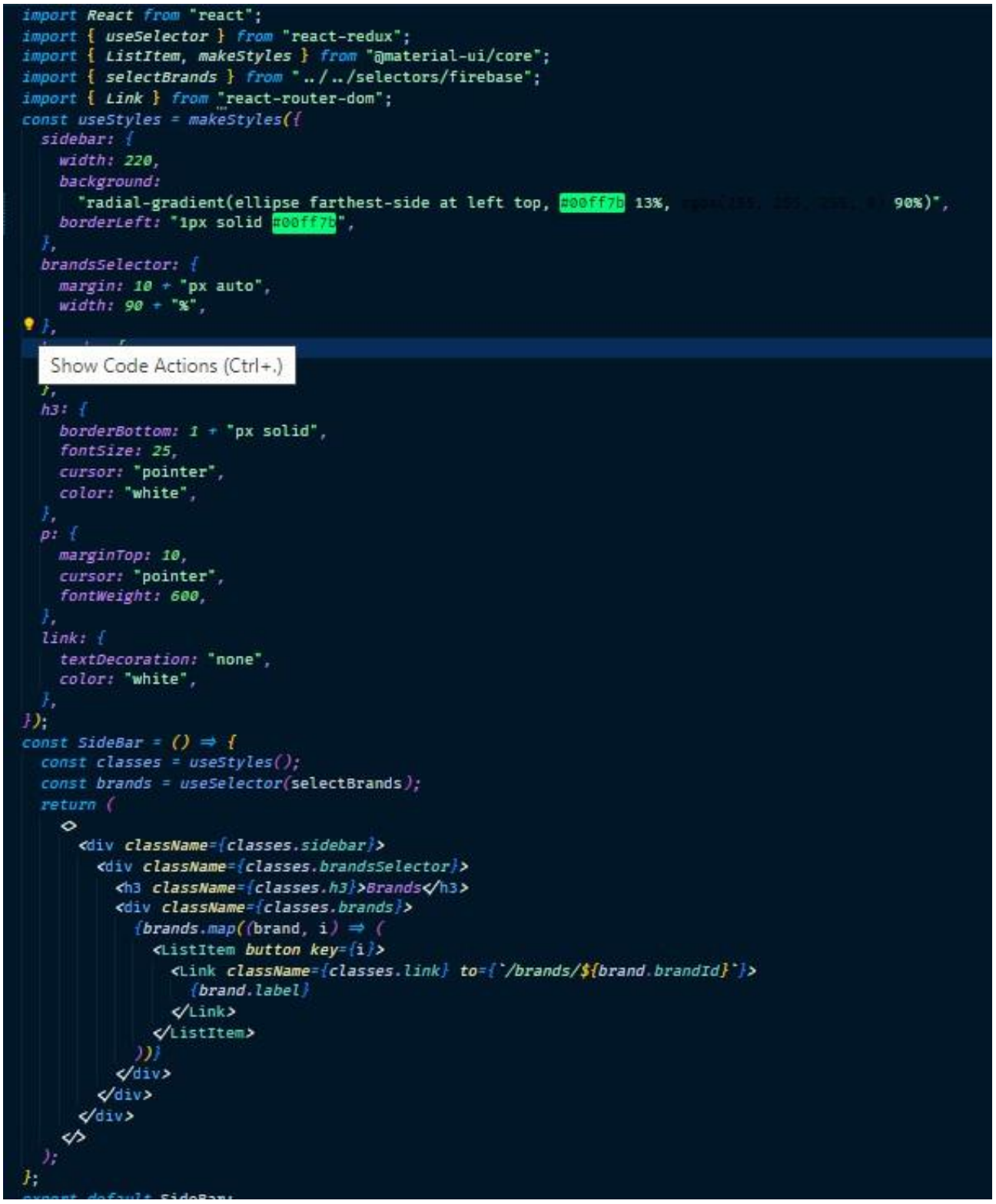
"бестселерами": Unknown word. cSpell
View Problem (Alt+F8) Quick Fix... (Ctrl+.)

"Знайомтеся": Unknown word

```jsx
import { Link } from "react-router-dom";
import "../../styles/discoverEvenMore.css";
function DiscoverEvenMore() {
  return (
    <div className="discover-main">
      <h1 className="discover-title">Відкрийте ще більше</h1>    "Відкрийте": Unknown wor
      <div className="discover-content">
        <Link className="discover-item " to="/categories/dAPYQgeAUzNgKlAfFsJg">
          <div className="discoverEvenMore discover-lips">
            <span>Трояндн</span>    "Трояндн": Unknown word.
          </div>
        </Link>
        <Link className="discover-item " to="/categories/klfNpOuNiwtYoKKMiKfx">    "Niwt"
          <div className="discoverEvenMore discover-face">
            <span>Тюлпани</span>    "Тюлпани": Unknown word.
          </div>
        </Link>
        <Link className="discover-item " to="/categories/ShNIPYCvUYhcWz6pUpPI">    "NIPY"
          <div className="discoverEvenMore discover-eyes">
            <span>Лінії</span>    "Лінії": Unknown word.
          </div>
        </Link>
      </div>
    </div>
  );
}
export default DiscoverEvenMore;
```

```jsx
import React from "react";
import { makeStyles } from "@material-ui/core/styles";
import { Typography as MaterialTypography } from "@material-ui/core";
const useStyles = makeStyles({
  typography: {
    color: "#4c003f",
  },
});

const Typography = (props) => {
  const {
    children,
    color = "black",
    variant = "body2",
    className = "",
    component = "p",
    ...rest
  } = props;

  const typographyClasses = useStyles();
  return (
    <MaterialTypography
      variant={variant}
      className={`${typographyClasses.typography}, ${className}`}
      color={color}
      component={component}
      {...rest}
    >
      {children}
    </MaterialTypography>
  );
};
export default Typography;
```

```
  useEffect(() => {
    const getBrandLogo = (photo) => {
      setImg(photo); // Встановлюємо URL зображення напряму    "Встановлюємо": Unknown word.
    };

    if (selectedItem && selectedItem.image) {
      getBrandLogo(selectedItem.image);
    } else {
      setImg(""); // Очищаємо URL зображення, якщо selectedItem або selectedItem.photo не існує    "Очищаємо":
    }

    // Зверніть увагу, що тут немає необхідності використовувати return    "Зверніть": Unknown word.
  }, [selectedItem]); ← #70-82 useEffect
  return (
    <div className={classes.root}>
      <Paper className={classes.paper}>
        <Grid container spacing={2}>
          <Grid item>
            <ButtonBase className={classes.image}>
              <img className={classes.img} alt="complex" src={img} />
            </ButtonBase>
          </Grid>
          <Grid item xs={12} sm container>
            <Grid item xs container direction="column" spacing={2}>
              <Grid item xs>
                <Typography gutterBottom variant="h4">
                  {selectedItem ? selectedItem.name : ""}
                </Typography>
                <Typography variant="body2" color="textSecondary">

                  {selectedItem ? selectedItem.brand : ""}
                </Typography>
                <br />
                <Typography variant="subtitle1">
                  $ {selectedItem ? selectedItem.price : ""}
                </Typography>
                <Button
                  variant="contained"
                  color="default"
                  className={classes.button}
                  onClick={() => {
                    handleAddToBagItem(
                      { ...selectedItem },
                      user,
                      alertDraft,
                      dispatch,
                      history
                    ); ← #111-117 handleAddToBagItem
                  }} ← #110-118 onClick=
                >
                  Додати до корзини    "Додати": Unknown word.
                </Button>
              </Grid>
            </Grid>
          </Grid>
        </Grid>
      </Paper>

      <div style={{ display: "flex", justifyContent: "center" }}>
        {newArray.map((item, i) => (
          <Item key={item.itemId} ind={i} {...item} />
        ))}
      </div>
    </div>
  ); ← #83-134 return
}; ← #48-135 const LearnMore = () =>

export default LearnMore;
```


import React from "react";
import { useSelector } from "react-redux";
import { ListItem, makeStyles } from "@material-ui/core";
import { selectBrands } from "../../selectors/firebase";
import { Link } from "react-router-dom";
const useStyles = makeStyles({
sidebar: {
width: 220,
background:
"radial-gradient(ellipse farthest-side at left top, #00ff7b 13%, 90%)",
borderLeft: "1px solid #00ff7b",
},
brandsSelector: {
margin: 10 + "px auto",
width: 90 + "%",
},
Show Code Actions (Ctrl+.)
},
h3: {
borderBottom: 1 + "px solid",
fontSize: 25,
cursor: "pointer",
color: "white",
},
p: {
marginTop: 10,
cursor: "pointer",
fontWeight: 600,
},
link: {
textDecoration: "none",
color: "white",
},
});
const SideBar = () => {
const classes = useStyles();
const brands = useSelector(selectBrands);
return (
<>

<h3 className={classes.h3}>Brands</h3>

{brands.map((brand, i) => (
<ListItem button key={i}>
<Link className={classes.link} to={`/brands/${brand.brandId}`}>
{brand.label}
</Link>
</ListItem>
))}

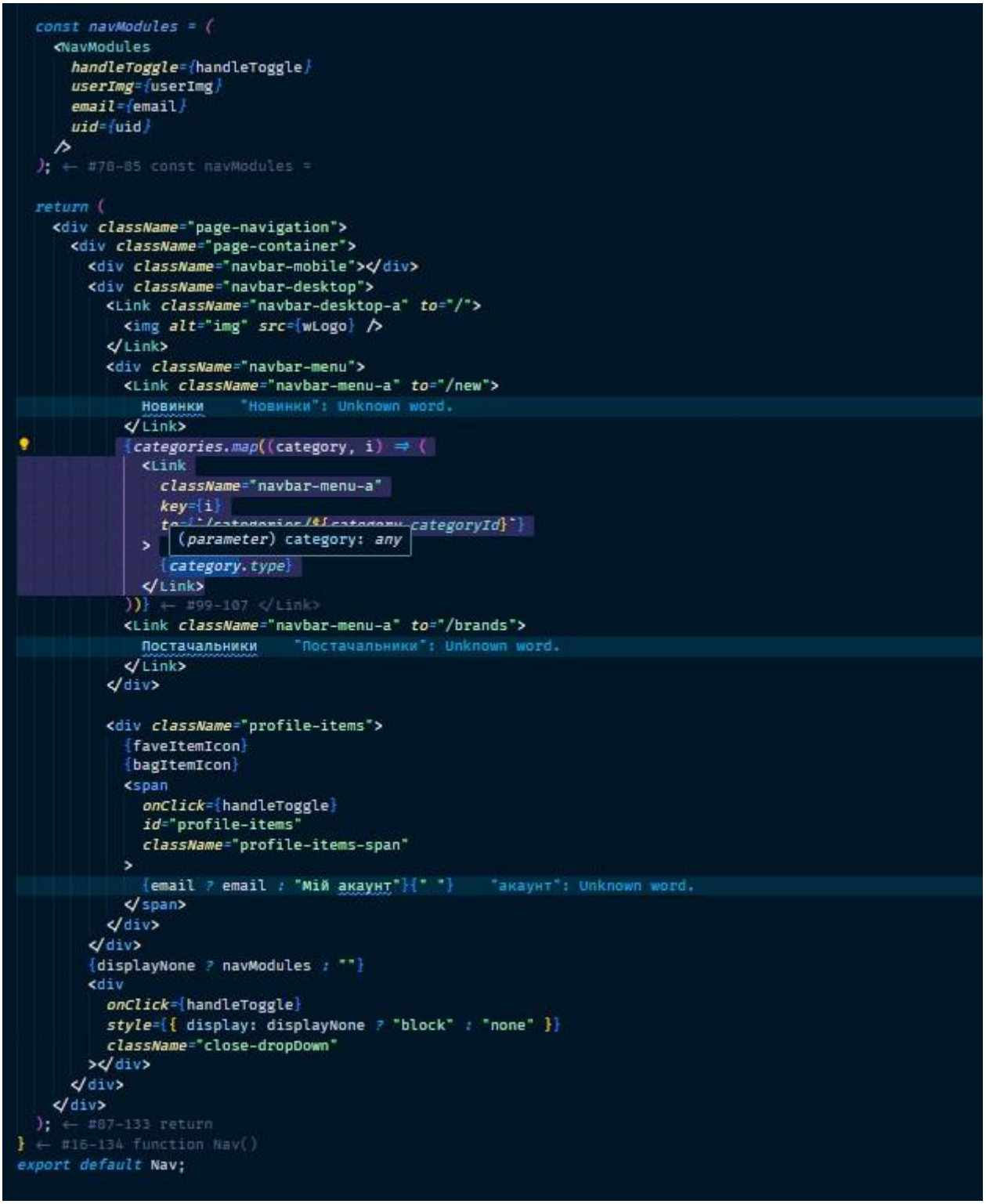
</>
);
};

```
const BagItem = (props) => {
  function handleDeleteFromBag(itemId, user) {
      .then(() => {
        '',
        alert.show(
          <div style={{ color: "white", fontSize: "12px" }}>
            'Товар Видалено !'    "Товар": Unknown word.
          </div>
        ); ← #61-65 alert.show
      }); ← #50-66 .then
  } ← #42-67 function handleDeleteFromBag(itemId, user)

  return (
    <div className={classes.root}>
      <Fragment>
        <Accordion>
          <AccordionSummary
            expandIcon={<ExpandMoreIcon />}
            aria-controls="panel1a-content"
            id="panel1a-header"
          >
            <div className={classes.column}>
              <Typography className={classes.heading}>{name}</Typography>
            </div>
            <div className={classes.column}>
              <Typography className={classes.secondaryHeading}>
                $ {price}
              </Typography>
            </div>
          </AccordionSummary>
          <AccordionDetails className={classes.details}>
            <div className={classes.column}>
              <div className={classes.imgDiv}>
                <img className={classes.img} src={img} alt="" />
              </div>
            </div>
            <div className={classes.column}>
              <Chip
                label="Видалити з корзини"    "Видалити": Unknown word.
                onClick={() => {
                  handleDeleteFromBag(itemId, user);
                }}
              />
            </div>
            <div className={clsx(classes.column, classes.helper)}>    "clsx": Unknown word.
              <Typography variant="caption">
                Виберіть квіти    "Виберіть": Unknown word.
                <br />
              </Typography>
            </div>
          </AccordionDetails>
          <Divider />
          <AccordionActions>
            <Link to={`/learnmore/${itemId}`}>    "learnmore": Unknown word.
              <Button
                bgcolor="white"    "bgcolor": Unknown word.
                labelcolor="#4c003f"    "labelcolor": Unknown word.
                width="140px"
                border="none"
              >
                Дізнатися більше    "Дізнатися": Unknown word.
              </Button>
            </Link>
          </AccordionActions>
        </Accordion>
      </Fragment>
    </div>
  ); ← #69-124 return
}; ← #24-125 const BagItem = (props) =>
export default BagItem;
```

```
import { makeStyles } from "@material-ui/core/styles";
export const useStylesForBag = makeStyles((theme) => ({
  paper: {
    display: "flex",
    justifyContent: "space-between"
  },
  image: {
    width: 128,
    height: 128
  },
  root: {
    width: "100%"
  },
  container: {
    maxHeight: 440
  },
  bullet: {
    display: "inline-block",
    margin: "0 2px",
    transform: "scale(0.8)"
  }, ← #17-21 bullet:

  pos: {
    marginBottom: 12
  },
  bagComponent: {
    width: "95%",
    margin: "auto",
    marginBottom: "50px",
    paddingTop: "30px"
  }, ← #26-31 bagComponent:
  bagHeader: {
    textAlign: "center",
    borderBottom: "1px solid white",
    color: "white"
  }, ← #32-36 bagHeader:
  leftContent: {
    width: "57%",
    borderRight: "1px solid white",
    height: "600px",
    overflow: "auto"
  }, ← #37-42 leftContent:
  bagItems: { width: "80%" },
  card: {
    width: "42%",
    margin: "0 auto"
  },
  emptyBag: {
    position: "relative",
    height: "600px"
  },
  emptyBagTitle: {
    color: "#ffffffab",
    position: "absolute",
    top: "50%",
    left: "20%",
    fontSize: "67px"
  } ← #52-58 emptyBagTitle:
})); ← #2-59 export const useStylesForBag = makeStyles
export const useStylesForBagItem = makeStyles((theme) => ({
  root: {
    width: "100%",
    marginTop: "2px"
  },
  heading: {
    fontSize: theme.typography.pxToRem(15)
  },
  secondaryHeading: {
    fontSize: theme.typography.pxToRem(15),
    color: theme.palette.text.secondary
  },
  icon: {
```

```jsx
const Login = (props) => {
  const handleChange = (e) => {
    if (e.target.type === "email") {
      submitNewValues.email = e.target.value;
    } else if (e.target.type === "password") {
      submitNewValues.password = e.target.value;
    }
    setValues(submitNewValues);
  }; ← #18-25 const handleChange = (e) =>

  const handleSubmit = (e) => {
    e.preventDefault();
    auth
      .signInWithEmailAndPassword(values.email, values.password)
      .then((userCredential) => {
        let user = userCredential.user;
        db.collection("users")
          .get(user.uid)
          .then((querySnapshot) => {
            submitNewValues.errorMessage = " ";
            history.push("/");
          });
      }) ← #31-39 .then
      .catch((error) => {
        submitNewValues.errorCode = error.code;
        submitNewValues.errorMessage = error.message;
        setValues(submitNewValues);
        console.log(error.code, error.message);
      }); ← #40-45 .catch
  }; ← #27-46 const handleSubmit = (e) =>

  return (
    <>
      <div className="login-main">
        <div className="login-container">
          <div className="login-logo">
            <div className="logo">
              <Link to="/">
                <img src={pLogo} alt={pLogo} />
              </Link>
            </div>
          </div>
          <div className="login-form">
            <div className="form">
              <form onSubmit={handleSubmit}>
                <input
                  type="email"
                  value={values.email}
                  onChange={handleChange}
                  placeholder="E-mail"
                  required
                />
                <input
                  type="password"
                  value={values.password}
                  onChange={handleChange}
                  placeholder="Password"
                  required
                />
                <span className="error-massage">
                  {values.errorMessage ? values.errorMessage : ""}
                </span>
                <input className="submit" type="submit" value="LOG IN" />
              </form>
            </div>
          </div>
          <div className="in-login-create-newAcc-btn">
            <div className="create-newAcc-btn">
              <a href="/signup">Створити новий акаунт</a>    "Створити": Unknown word.
            </div>
          </div>
        </div>
      </div>
    </>
```

```jsx
const Signup = () => {
  const handleSignUp = (e) => {
      .then((userCredential) => {
      }) ← #54-73 .then

      .catch((error) => {
        inputChange.errorMessage = error.message;
        inputChange.errorCode = error.code;
        setValues(inputChange);
      }); ← #75-79 .catch
  }; ← #50-80 const handleSignUp = (e) =>
  return (
    <>
      <div className="login-main">
        <div className="login-container">
          <div className="login-logo">
            <div className="logo">
              <a href="/">
                <img src={pLogo} alt={pLogo} />
              </a>
            </div>
          </div>
          <div className="login-form">
            <div className="form">
              <div className="chosen-img">
                <div className="chosen-img-content">
                  <img
                    src={chosen.previewImgUrl ? chosen.previewImgUrl : unKnown}
                    alt="user-img"
                  />
                </div>
                <span className="error-massage">
                  {values.sizeErrorMessage ? values.sizeErrorMessage : ""}
                </span>
              </div>
              <form onSubmit={handleSignUp}>
                <input
                  type="file"
                  accept=".jpg, .jpeg, .png"
                  onChange={handleImgChange}
                  id="userImgInp"
                />
                <label htmlFor="userImgInp">Choose image</label>
                <input
                  type="text"
                  onChange={handleInpChange}
                  value={values.username}
                  placeholder="username"
                  required
                />
                <input
                  type="email"
                  onChange={handleInpChange}
                  value={values.email}
                  placeholder="E-mail"
                  required
                />
                <input
                  type="password"
                  onChange={handleInpChange}
                  value={values.password}
                  placeholder="password"
                  required
                />

                <span className="error-massage">
                  {values.errorMessage ? values.errorMessage : ""}
                </span>
                <input className="submit" type="submit" value="CREATE" />
              </form>
            </div>
          </div>
          <div className="in-login-create-newAcc-btn">
            <div className="create-newAcc-btn">
```


const navModules = (
<NavModules
handleToggle={handleToggle}
userImg={userImg}
email={email}
uid={uid}
/>
); ← #78-85 const navModules =
return (

<Link className="navbar-desktop-a" to="/">

</Link>

<Link className="navbar-menu-a" to="/new">
Новинки "Новинки": Unknown word.
</Link>
{categories.map((category, i) => (
<Link
className="navbar-menu-a"
key={i}
(parameter) category: any
{category.type}
</Link>
))} ← #99-107 </Link>
<Link className="navbar-menu-a" to="/brands">
Постачальники "Постачальники": Unknown word.
</Link>

{faveItemIcon}
{bagItemIcon}
<span
onClick={handleToggle}
id="profile-items"
className="profile-items-span"
>
{email ? email : "Мій акаунт"}{" "} "акаунт": Unknown word.

{displayNone ? navModules : ""}
<div
onClick={handleToggle}
style={{ display: displayNone ? "block" : "none" }}
className="close-dropDown"
>

); ← #87-133 return
} ← #16-134 function Nav()
export default Nav;