Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

Розробка та оптимізація універсальних веб-інтерфейсів з використанням бібліотеки React або Angular

Текстова частина до курсової роботи

За спеціальністю «Інженерія програмного забезпечення» 121

Керівник курсової роботи
к.н., с.в. Демківський Є. О.
_____
(підпис)
“____” _____ 2022 р.
Виконав студент 3-го курсу
Атанасов Антон Сергійович
“____” _____ 2022 р.

Київ 2020

Зміст

Вступ

Ця робота має на меті огляд популярних бібліотек для розробки графічних інтерфейсів веб-застосунків. Створення власного веб-застосунку з використанням сучасних технологій.

Інтерфейс реалізовано за допомогою мови програмування JavaScript з використанням бібліотек React та React Router, сховище даних та засоби створення запитів до цього сховища з метою отримання даних — за допомогою мови запитів та маніпуляції даних GraphQL.

1. Аналіз предметної області

1.1 JavaScript

JavaScript — багатопарадигмена, інтерпретована мова програмування високого рівня. Найбільш відомою ця мова є в якості мови сценаріїв для веб-сторінок, але існують середи виконання JavaScript, що не є браузерами, як, наприклад, Node.js.

JavaScript був створений у 1995 році Бренданом Айхом в Netscape у співпраці з Марком Андрессеном, співзасновником Netscape Communications, та Біллом Джоєм, співзасновником Sun Microsystems. В процесі розробки мова зазнала великого впливу з боку таких мов, як C та Java. Остання також стала причиною такої назви мови для браузерів, оскільки на той час Java була надзвичайно популярною, і розробники мали отримати ліцензію в Sun Microsystems, щоб уникнути проблем з авторським правом.

Принципи роботи JavaScript

JavaScript реалізується (інтерпретується) за допомогою так званих «рушіїв». Найпершим рушієм був SpiderMonkey, створений Бренданом Айхом, але з часом з’явились інші інтерпретатори, як-от V8 (Google), Rhino (Mozilla Foundation), Carakan (Opera) тощо.

Переваги JavaScript

- Універсальність. Незважаючи на те, що мова була запланована як засіб з’єднання різних частин веб-сторінок, зараз можна і створити повноцінний веб-застосунок, створити сервер за допомогою Node.js, відобразити статистику у вигляді таблиць та діаграм або навіть зробити повноцінну гру.
- Інфраструктура. Для JavaScript протягом 25 років було створено багато бібліотек, фреймворків та інших рішень, що перетворило цю «іграшкову мову» в потужний інструмент.
- Простота. Невеликі задачі можливо вирішити дуже швидко, без зайвої роботи. Для більш великих можна знайти готове рішення, адаптувати під власні потреби.
- Зручність. З часом легко звикнути до синтаксису та логіки даної мови.

• Веб-інтерфейси. JavaScript додає динамічності елементам на сторінці: анімації, приховання, додавання, перевірка введених даних, реагування на різноманітні дії користувача.

Недоліки JavaScript

• Динамічна типізація та неявне приведення типів. В той час, як Java або C++ — статично типізовані (один тип змінної, що визначається в момент оголошення і не змінюється), в динамічно типізованому JavaScript змінна може мати будь-який тип, що залежить від даних, які присвоєні їй в будь-який момент виконання програми. Це може викликати помилки, які буде складно відлагодити.

• Помилки під час виконання програми. В Java та C++, якщо програміст неправильно написав код, виникне помилка компіляції, а програма не запуститься. Якщо щось піде не за планом в процесі виконання, програма «викине» помилку та зупинить роботу. JavaScript не є компільованою мовою програмування, тому всі помилки виникають лише тоді, коли інтерпретатор виконає стрічку коду. В Java, C++, навіть Python помилки дуже детальні, з точним місцем їх виникнення у коді. Помилки в JavaScript більш узагальнені та неконкретні, відлагодження програми займає більше часу навіть з потужним “дебаггером”.

1.2 ReactJS

React — бібліотека JavaScript, головне призначення якої — спростити створення веб-інтерфейсу та одночасно додати більше можливостей для забезпечення функціональності окремих його частин

Особливості бібліотеки

- «Менше коду, більше можливостей» — React виконує більшу частину роботи за програміста (деякі аспекти роботи бібліотеки можливо, але дуже складно зробити самотужки)
- Швидкість — за допомогою технології Virtual DOM (Document Object Model). Зазвичай, якщо деякий візуальний елемент на сторінці змінюється (наповнення, атрибути, зовнішній вигляд тощо), то браузер перезавантажує увесь документ. Якщо таких перезавантажень багато, то страждає швидкодія. Virtual DOM змінює лише ті елементи, які змінюються в процесі виконання програми, замість того, щоб оновити усе і одразу. Це дозволяє робити багато таких «змін» з набагато меншими втратами швидкості
- Повторне використання компонентів. Компоненти, написані у React, мають власну логіку, і дуже легко використати один і той самий компонент в різних місцях програми, але з різним (або тим самим) призначенням.
- Потік даних «в один бік». Один компонент може мати всередині інший, а той ще один, і так далі. Через таке дерево компонентів можна передавати дані від батьківського компонента до дочірнього. Такий підхід дозволяє знайти джерело помилок, якщо вони пов’язані з даними
- React можна використовувати у мобільній розробці за допомогою бібліотеки React Native, що походить від React. Тобто за допомогою синтаксису React можна створити як веб-застосунок, так і повноцінний мобільний додаток

Принципи роботи React

- JSX (JavaScript Extension) — розширення синтаксису JavaScript, що дозволяє писати елементи HTML у вигляді HTML просто у коді програми

```
import React from "react";        You, 3 months ago • Implemented

You, 3 months ago | 1 author (You)
class PageNotFound extends React.Component {
  render() {
    return <h1>Sorry, but this page doesn't seem to exist</h1>;
  }
}

export default PageNotFound;
```

Рисунок 1.2.1 Приклад з програми курсової роботи — метод компоненту React повертає простий елемент HTML без потреби у багаторазовому звертанні до об’єкту document

- Внутрішній стан компоненту (state) — об’єкт, що зберігає в собі дані, які можна змінити в процесі виконання програми

```
class CartCardGallery extends React.Component {

  constructor(props) {
    super(props);
    this.state = {
      currentImageIndex: 0
    }
  }

  render() {
    return (
      <div className="cartCardGallery">
        <img
          src={this.props.images[this.state.currentImageIndex]}
          alt={this.state.currentImageIndex} />
        <div       You, 2 months ago • Implemented cart page
          className="slideshowArrow slideLeft"
          onClick={() => this.setState({
            currentImageIndex: this.state.currentImageIndex === 0
              ? this.props.images.length - 1
              : this.state.currentImageIndex - 1
          })}
```

Рисунок 1.2.2 Стан дотримується концепції незмінності: єдиний спосіб щось змінити всередині — повністю перезаписати об’єкт замість того, щоб напряму змінювати окремі значення

- Властивості компонента (props) — незмінні значення, що передаються від батьківського компонента дочірньому

```
render() {
  return (
    <div className="app">
      <div className="headerContainer">
        <Header
          categories={this.state.categories}
          currencies={this.state.currencies}
          currencyIndex={this.state.currencyIndex}
          chooseCurrency={(index) => this.chooseCurrency(index)}
          bagSize={this.state.cart.reduce((acc, product) => acc + product.quantity, 0)}
          cart={this.state.cart}
          toggleMiniCart={this.toggleMiniCart}
          showingMiniCart={this.state.showingMiniCart}
          clearCart={this.clearCart}
          incrementProductQuantity={(index) => this.incrementProductQuantity(index)}
          decrementProductQuantity={(index) => this.decrementProductQuantity(index)}
        />
```

Рисунок 1.2.3 Властивістю може бути що завгодно — число, стрічка, функція, посилання на функцію, масив тощо. Головне — змінювати їх всередині компоненту не можна

```
class Header extends React.Component {      You, 3 months ago • Implement
  render() {
    return (
      <header>
        <Navigation categories={this.props.categories} />
        <div className="logo">
          <img src={logo} alt="Company logotype" />
        </div>
        <Actions
          currencies={this.props.currencies}
          currencyIndex={this.props.currencyIndex}
          chooseCurrency={this.props.chooseCurrency}
          bagSize={this.props.bagSize}
          cart={this.props.cart}
          showingMiniCart={this.props.showingMiniCart}
          toggleMiniCart={this.props.toggleMiniCart}
          clearCart={this.props.clearCart}
          incrementProductQuantity={this.props.incrementProductQuantity}
          decrementProductQuantity={this.props.decrementProductQuantity}
        />
      </header>
    );
  }
}
```

Рисунок 1.2.4 Компонент, що працює лише з властивостями компонентів

- Життєвий цикл компонента — компоненти з’являються, оновлюються та видаляються з дерева компонентів. Існують спеціальні методи — методи життєвого циклу — що викликаються до, під час або після вищевказаних подій. Ці методи можна використати, щоб: задати початковий стан, коли компонент лише створюється; запобігти зайвим оновленням компоненту, якщо стан часто змінюється, але властивості залишаються без змін; видалити глобальні об’єкти (наприклад, setInterval), коли компонент видаляється, тощо.

1.3 GraphQL

GraphQL — мова запитів та середа виконання цих запитів

Переваги перед REST API

Як і архітектура REST, GraphQL відсилає запити на сервер, той повертає дані. Але GraphQL має певні плюси:

- Не більше, не менше: GraphQL, на відміну від REST, дозволяє задати запитуваний об’єкт, в результаті якого сервер відповість лише з тими даними, що потрібні в момент виконання, і не більше. Це дозволяє зменшити розмір об’єкта, що повертається сервером, а це, у свою чергу, покращить швидкодію запитів, особливо якщо їх відсилається багато за одиницю часу
- Чітка типізація та самодокументування: сервер на GraphQL задає типи даних для об’єктів, що повертаються, та значень всередині. Це значить, що клієнт завжди знає, які дані (числа, стрічки, булеві значення тощо) повертатиме запит. До того ж, «схеми», за якими визначаються об’єкти (з яких за минулим пунктом можна взяти лише те, що потрібно) можна вважати своєрідною документацією, до якої можна звернутися, щоб дізнатись, що є, а чого немає
- Все в одному місці: в той час, коли для REST на сервері потрібно визначати декілька точок доступу (endpoints), заголовки запитів та відповідей, для конкретної задачі може знадобитись декілька запитів на різні точки, що, разом з недоліком з першого пункту, впливатиме на швидкодію. GraphQL же зберігає все в одному місці, і достатньо лише одного запиту, щоб отримати дані з різних об’єктів

1.4 React Router

React Router — бібліотека, що дозволяє створити так звані «застосунки на одній сторінці»

Застосунок на одній сторінці

Більшість великих веб-застосунків мають різні сторінки, по яких користувач може переходити, повертатись назад — навігація. Якщо це застосунок, що програмується на сервері (server-side rendering), то разом із точками доступу до даних (описані у пункті 1.3) потрібні точки доступу до окремих сторінок, а це додаткові запити та зусилля для того, щоб зробити елементи динамічними, або зробити так, щоб якийсь елемент залишався на декількох сторінках.

React та React Router становлять тандем, що дозволяє створювати застосунки, що програмуються на клієнтській частині (client-side rendering). Тобто це один запит, щоб отримати те, ще побачить користувач, коли той вперше зайде на сторінку, а решта буде створена та замінена на актуальний зміст сайту в процесі користування

Для навігації сторінками в такому застосунку і призначений React Router. Він маніпулює вбудованим об’єктом історії браузеру, щоб імітувати поведінку навігації. Замість сторінок, що очікують на відкриття, все вже було відправлено до користувача, а перехід від однієї сторінки до іншої забезпечується посиланнями, що замінюють один компонент на інший

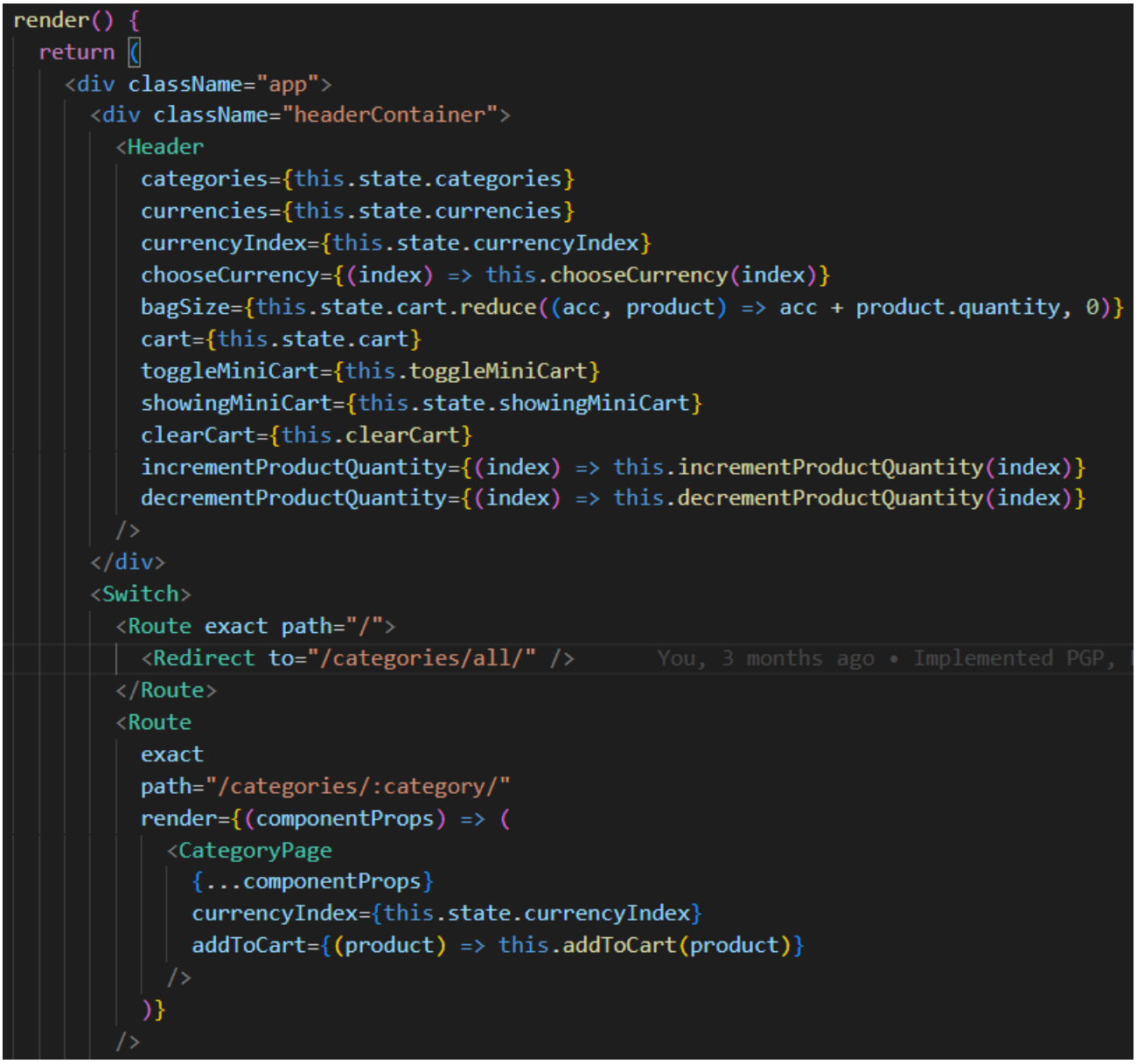

Рисунок 1.4.1 React-компонент Header містить в собі посилання на інші «сторінки» застосунку. Вбудований ReactRouter-компонент Switch переключатиметься на компонент Route, чий атрибут path збігається з посиланням в застосунку. Той самий компонент потім перетвориться на React-компонент, що визначений в атрибуті render

2. Розробка веб-інтерфейсу

2.1 Тематика веб-застосунку

Застосунок є прототипом інтернет-магазину. Обраний набір технологій є ідеальним для таких застосунків з декількох причин:

- Категорії продуктів, самі продукти та кошик покупця можуть виступати окремими «сторінками» — React Router
- Продукти, що показані на сторінці категорій, мають спільні дані, тому можна розбити запит до даних продукту на два виклики: для демонстрації товару у категорії (назва, перше зображення, ціна, властивості) та для власної сторінки (те саме, але додатково опис товару та решта зображень). Так само і власне категорії — можна відправити запит на категорії та продукти окремо взятої категорії — GraphQL
- Для створення шаблону категорії та компоненту, потрібні лише два компоненти — React. До того ж, користувач має змогу обрати тип товару, який він хоче купити (колір, розмір, місткість, діагональ тощо), додати його до кошика та змінити бажану для покупки кількість

2.2 Вхідні дані

Дана схема GraphQL визначає загальний вид об’єктів, до яких клієнт може звертатись за даними. Сервер завжди повертає відповідь у форматі JSON

```
const typeDefs = gql`
    type Price {
        currency: Currency!,
        amount: Float!
    }

    type Attribute {
        displayValue: String,
        value: String,
        id: String!
    }
```

```
    type AttributeSet {
        id: String!,
        name: String,
        type: String,
        items: [Attribute]
    }

    type Product {
        id: String!,
        name: String!,
        inStock: Boolean,
        gallery: [String],
        description: String!,
        category: String!,
        attributes: [AttributeSet]
        prices: [Price!]!,
        brand: String!
    }

    type Category {
        name: String,
        products: [Product]!
    }

    type Currency {
        label: String!,
        symbol: String!
    }

    input CategoryInput {
        title: String!
    }

    type Query {
        categories: [Category],
        category(input: CategoryInput): Category,
        product(id: String!): Product,
        currencies: [Currency]
    }
`;
```

2.3 Веб-інтерфейс

У застосунку реалізована навігація різними «сторінками», перегляд категорій, перегляд окремих продуктів, зміна валюти, вибір типу продукту, додавання та видалення продукту до кошика, зміна кількості продукту в кошику, динамічний підрахунок підсумкової вартості усіх товарів

Користувач може додати товар як зі сторінки категорії, так і зі сторінки самого продукту. Якщо користувач обирає товар, знаходячись на сторінці категорії, з кожному масиву типів товару обирається перший тип (при переході на сторінку товару типи виставлені за замовчуванням)

Кошик може бути як випадаючим віконцем, так і окремою сторінкою, для цього використовується один компонент

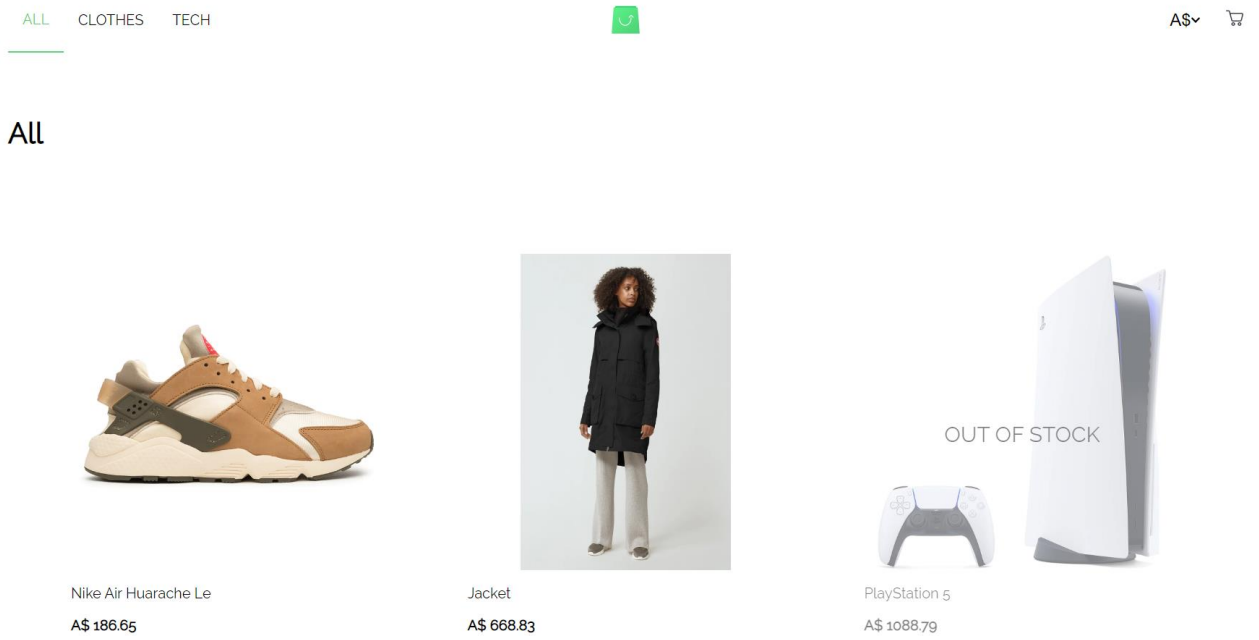

Рисунок 2.3.1 Користувач автоматично переходить з кореня сайту на категорію «Всі товари», що і є головною сторінкою

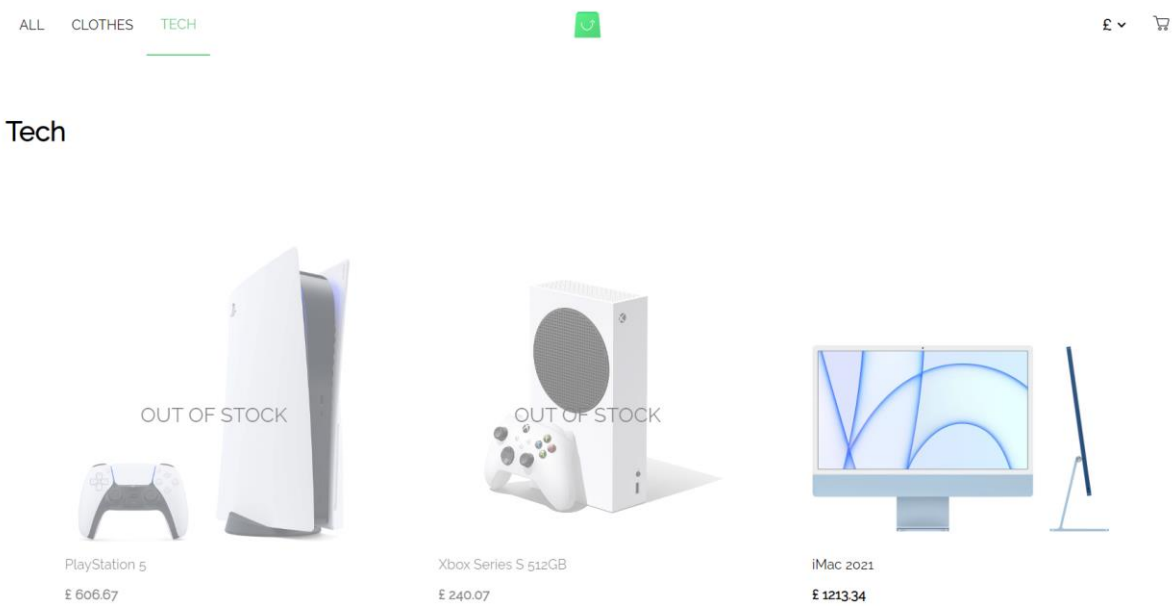

Рисунок 2.3.2 Користувач перейшов до іншої категорії. Зверніть увагу — деяких товарів «немає на складі», і користувач не зможе їх додати до кошику ані зі сторінки категорії, ані товару

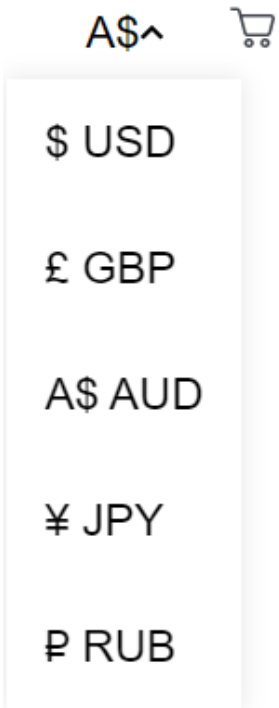

Рисунок 2.3.3 При натисканні на значок валюти в правому кутку згори випадає віконце, що дозволяє змінити валюту, в якій відображені ціни на товари. Зміна відбувається одразу, без перезавантаження сторінки

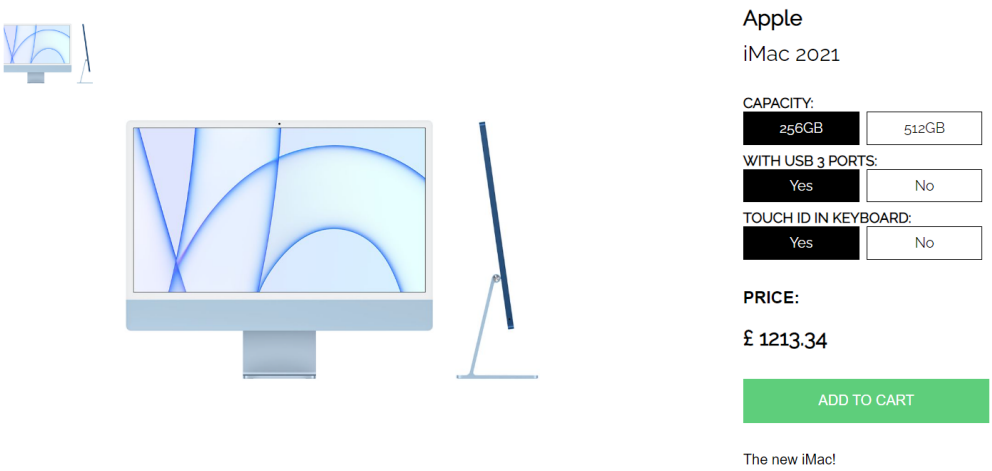

Рисунок 2.3.4 Сторінка товару

Рисунок 2.3.5 Кошик — модальне віконце. Обраний тип товару відображається у віконці, є можливість змінити кількість товару, але не його тип. При досягненні кількості товару нуля той видаляється з кошику

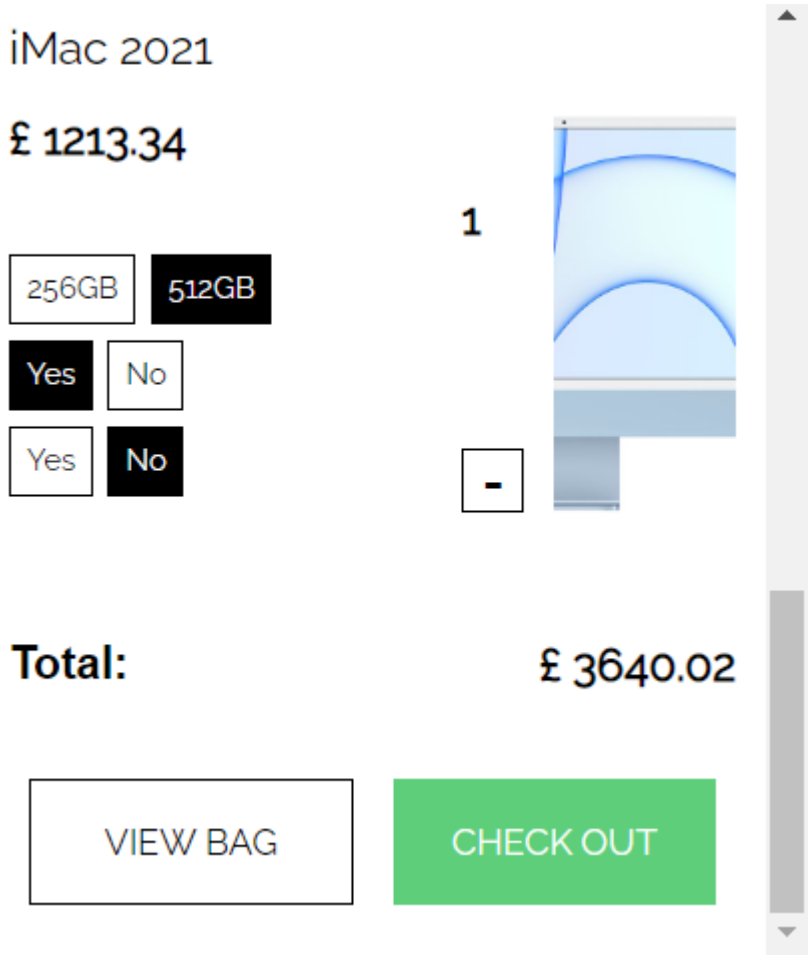

Рисунок 2.3.5 Кнопка «View Bag» дозволяє побачити кошик у вигляді повноцінної сторінки. Кнопка «Check out» видаляє усі товари, імітуючи відсилання замовлення на сервер, а користувача повертає до головної сторінки

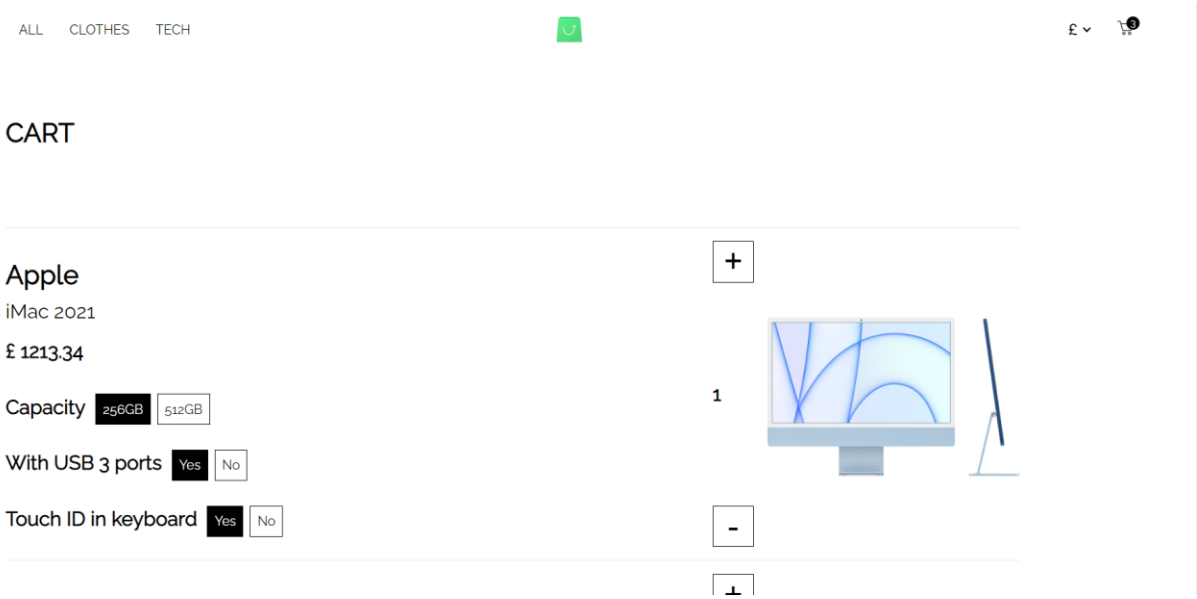

Рисунок 2.3.5 Кошик — сторінка

Висновок

В результаті роботи мета була досягнута. Проаналізовані бібліотеки для створення веб-інтерфейсу. Отримані знання при підготовці та створенні застосунку є важливими та актуальними сьогодні. Реалізований веб-застосунок для інтернет-магазину, який легко можна адаптувати під будь-які товари

Список джерел

1. Mozilla Developer Network[Електронний ресурс] — режим доступу: https://developer.mozilla.org/en-US/

2. ReactJS official website[Електронний ресурс] — режим доступу: https://reactjs.org/

3. React Router documentation[Електронний ресурс] — режим доступу: https://v5.reactrouter.com/web/guides/quick-start

4. GraphQL documentation[Електронний ресурс] — режим доступу: https://graphql.org/learn/