НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ
імені адмірала Макарова

Навчально-науковий інститут автоматики та електротехніки
(повне найменування інституту, факультету)

Кафедра комп'ютеризованих систем управління
(повна назва кафедри)

«Допущений до захисту»
Завідувач кафедри
Черно О.О.
«__»_____2022 р.

Пояснювальна записка

до кваліфікаційної роботи

на здобуття першого (бакалаврського) рівня вищої освіти
(освітньо-кваліфікаційний рівень)

на тему: РОЗРОБКА WEB-ДОДАТКУ ДЛЯ АВТОМАТИЗАЦІЇ РОБОТИ ПІЦЕРІЇ

Виконав студент **2** курсу, **2347СТ** групи
спеціальності **«151 - Автоматизація та комп'ютерно-інтегровані технології»**
(шифр і назва спеціальності)

Поянов І.І.
(прізвище та ініціали)

Керівник **Черно О.О.**
(прізвище та ініціали)

Рецензент **Новогрецький С.М.**
(прізвище та ініціали)

м. Миколаїв - 2022 року

Національний університет кораблебудування імені адмірала Макарова

НН Інститут автоматики і електротехніки
Кафедра комп'ютеризованих систем управління
Освітньо-кваліфікаційний рівень **«бакалавр»**
Спеціальність **151 «Автоматизація та комп'ютерно-інтегровані технології»**
Освітня програма **«Комп'ютеризовані системи управління та автоматика»**

ЗАТВЕРДЖУЮ
Завідувач кафедри КСУ
_____ **Черно О.О.**
«14» лютого **2022 р.**

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНА РОБОТУ СТУДЕНТУ

П о я н о в у І л л і І г о р о в и ч у
(прізвище, ім'я, по батькові)

1. Тема роботи: *Розробка web-додатку для автоматизації роботи піцерії*

керівник роботи *Черно Олександр Олександрович, д.т.н., доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від " 4 " травня 2022 р. № 254-уч

2. Строк подання роботи *20 червня 2022 р.*

3. Вихідні дані до роботи *Швидкість роботи додатку не менша у порівнянні з конкурентами, можливість експлуатації додатку при slow 3G інтернеті*

4. Зміст розрахунково-пояснювальної записки (основні задачі дослідження)
Аналіз даних та технологій конкурентів у розробці web-додатку
Вибір бібліотеки та технолоігй для будування веб-сайту
Розробка, налаштування та відладка додатку. Проведення тестів та обговорення щодо додавання нового функціоналу
Охорона праці

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Аналіз даних та технологій конкурентів у розробці web-додатку
Вибір бібліотек та технологій для будування веб-сайту

Розробка, відлагодження та налаштування додатку

Тестування додатку

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
Охорона праці	*Черно О.О., зав. каф. КСУ*	14.02.2022	10.06.2022

7. Дата видачі завдання: «14» лютого 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційна проектування	Строк виконання	Примітка
1	*Аналіз даних та технологій конкурентів у розробці web-додатку*	*01.04.2022*	*Виконано*
2	*Вибір бібліотеки та технолоігй для будування веб-сайту*	*30.04.2022*	*Виконано*
3	*Розробка, налаштування та відладка додатку. Проведення тестів та обговорення щодо додавання нового функціоналу*	*31.05.2022*	*Виконано*
4	*Охорона праці*	*10.06.2022*	*Виконано*

Студент _____ (підпис) Поянов І.І. (прізвище та ініціали)

Керівник роботи _____ (підпис) Черно О.О. (прізвище та ініціали)

АНОТАЦІЯ

Обсяг кваліфікаційної роботи 69 сторінки, у якій: 23 рисунки та 16 використаних джерел.

В кваліфікаційній роботі розроблено web-додаток для автоматизації піцерії.

В процесі розробки web-додатку було виявлено основні проблеми піцерії; зібрано дизайн додатку за макетом; встановлено всі необхідні бібліотеки для коректної та швидкої роботи додатку; налагодження користувацького інтерфейсу та написання всіх функцій для повного функціонування додатку; проведення тестів щодо швидкості функціонування додатку на мобільній і ПК версії; розглянуто питання охорони праці щодо роботи за ПК.

Ключові слова: веб-розробка, веб-сайт, фреймворк, React, верстка, адаптація сторінки.

ABSTRACT

Volume of qualification work 69 page, in which: 23 drawings and 16 used sources.

In the thesis a web-application for pizzeria automation was developed.

In the process of developing a web-application, the main problems of the pizzeria were identified; the design of the application according to the layout is collected; installed all the necessary libraries for the correct and fast operation of the application; debugging the user interface and writing all the functions for the full functioning of the application; conducting tests on the speed of the application on the mobile and PC versions; the issue of labor protection in relation to work on the PC is considered.

Keywords: web development, website, framework, React, layout, page adaptation.

Зміст

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

DOM(Document Object Model) **-** незалежний від платформи та мови програмний інтерфейс, що дозволяє програмам та скриптам отримати доступ до вмісту HTML-, XHTML- та XML-документів, а також змінювати вміст, структуру та оформлення таких документів.

SPA - веб-додаток або веб-сайт, що використовує єдиний HTML-документ як оболонку для всіх веб-сторінок і організує взаємодію з користувачем через HTML, CSS, JavaScript, що динамічно підвантажуються, зазвичай за допомогою AJAX.

MPA - багатосторінкові програми, що працюють як звичні нам веб-сайти.

ВСТУП

Глобальна мережа Інтернет вже настільки міцно увійшла до нашого життя, що публікація інформації в WWW стала нормою. Тому організація взаємодії інформаційної системи з web-сервером зараз актуальна.

Під інформаційною системою зазвичай розуміється комплекс програм, орієнтований на збирання, зберігання, пошук та обробку текстової та/або фактографічної інформації. Переважна більшість інформаційних систем працює у режимі діалогу з користувачем.

Розвиток Інтернету розширив можливості роботи з віддаленими підрозділами, спілкування, допомогу та підтримку з різних питань через Інтернет та багато іншого.

У складі інформаційних систем можна виділити дві відносно незалежні складові: комп'ютерну інфраструктуру, що є сукупністю мережевої, телекомунікаційної, програмної, інформаційної та організаційної інфраструктур; взаємопов'язані функціональні підсистеми, що забезпечують вирішення завдань та досягнення цілей.

Перша складова відображає системно-технічний, структурний бік будь-якої інформаційної системи. Вимоги до комп'ютерної інфраструктури єдині та стандартизовані, а методи її побудови добре відомі та багаторазово перевірені на практиці.

Друга складова цілком відноситься до прикладної області і значною мірою залежить від напряму роботи. Ця складова повністю базується на комп'ютерній інфраструктурі та визначає прикладну функціональність інформаційної системи.

Завдяки інтеграції Інтернет-технологій та архітектури клієнт-сервер, процес впровадження та супроводу інформаційної системи суттєво спрощується при збереженні досить високої ефективності та простоти спільного використання інформації.

Активний розвиток Інтернету призвів до можливості створення web-сайтів для надання різноманітних інформації та послуг.

Також використання web-технологій відкриває широкі перспективи для електронної комерції та обслуговування клієнтів через Інтернет.

Інтернет-комерція, торгівля в Інтернеті - це комерційна діяльність в Інтернеті, коли процес купівлі/продажу товарів або послуг (весь цикл комерційної/фінансової транзакції або її частина) здійснюється електронним чином із застосуванням Інтернет-технологій.

З вищесказаного випливає, що сайт є необхідним атрибутом для сучасної компанії визначає тему кваліфікаційної роботи «Розробка web-додатку для автоматизації роботи піцерії».

Для досягнення мети в роботі проведено огляд діяльності компанії та сучасних методів розробки веб-сайтів, здійснено вибір методу що найбільш підходить для даної розробки, розроблено текст програми, здійснено її відлагодження, виконано налаштування і тестування розробленого web-додатку. В роботі також розглянуте питання охорони праці при роботі з комп’ютерною технікою.

Робота виконана на замовлення Fox Group та впроваджена в цій компанії.

					151.2347СТ.КР.ПЗ.Р1	
Змн	Арк	№ докум.	Підпис	Дата		
Розроб.		Поянов І.І.			*Аналіз даних та технологій конкурентів у розробці web-додатку*	Літ. / Арк: 10 / Аркушів
Керівни		Черно О.О.				
Зав.каф.		Черно О.О.				НУК ім. адмірала Макарова
Н.контр.		Вінниченко І.Л.				

РОЗДІЛ 1

АНАЛІЗ ДАНИХ ТА ТЕХНОЛОГІЙ КОНКУРЕНТІВ У РОЗРОБЦІ WEB-ДОДАТКУ

1.1 Характеристика піцерії та основні проблеми

В кваліфікаційному проекті розглядається піцерія. Дане підприємство має приміщення в якому знаходяться: кімнати для персоналу, склад для зберігання продуктів, пекарня з печами для випічка піц та касове відділення де люди можуть подивитись асортимент товару та зробити замовлення.

Заклад не має приміщень для споживання їжі, тому працює тільки на видачу піци з собою, тому головною проблемою закладу є мала кількість замовлень через те що їх потрібно пробити тільки в приміщені піцерії.

Кращим рішенням проблеми цього закладу є створення web-додатку через який люди зможуть заздалегідь зробити замовлення. Web-додаток дозволить зробити замовлення простішими і люди зможуть йдучи додому заздалегідь замовити собі піцу не очікуючи на вулиці.

В наш час більшість людей працюють віддалено вдома, а часу та сил готувати не завжди вистачає, а саме такі люди частіше роблять замовлення в таких додатках, цим самим ми збільшимо обсяг клієнтів, вони зможуть замовити з дому собі їжу, а кур'єр її доставить до дому.

1.2 Аналіз існуючих технологій що використовується при розробці web-додатків

Основна різниця між звичайним сайтом та онлайн-магазином регулюється функціями кошика товарів та можливістю оплати замовлення на сайті. Цей функціонал можна додати до сайту, використовуючи різні технології.

Залежно від бюджету та потреб бізнесу можна вибрати з [1]:

- плагіна кошика товарів;

- платформи для створення онлайн магазинів;

- розробка персоналізованої платформи для майбутнього онлайн магазину.

На перший погляд, плагін кошика товарів та платформа для електронної комерції схожі. Завдяки ним, у сайту з'являються необхідні функції для онлайн-торгівлі(рис. 1.1.).

Але при цьому вони абсолютно різні.

Рис. 1.1. Кошик сайтів

Інтеграція кошика товарів підійде, якщо у вас вже є сайт і ви хочете додати каталог товарів та проводити онлайн оплату. Цей функціонал надають різні плагіни та розширення для сайтів. Крім того, кошик товарів має досить обмежений набір інструментів для онлайн-торгівлі, методів оплати та варіантів дизайну. Але, залежно від обраного плагіну та сайту, можете персоналізувати кошик товарів.

Вартість таких розширень може досягати 500 доларів, залежно від кількості функцій та товарів на сайті. Можна використовувати це рішення, якщо у вас обмежені часові рамки та бюджет, при цьому ви не плануєте розширювати бізнес у майбутньому. Пам'ятайте, що такі плагіни не

підтримують адаптивний дизайн, через що ви втрачаєте потенційний трафік з мобільних пристроїв.

Платформи для онлайн-магазинів дають продавцям повноцінне рішення для торгівлі в інтернеті. Залежно від типу платформи, вона може включати хостинг, обробку онлайн-платежів, інструменти для маркетингу, безпека, шаблони дизайну та інструменти анатиліки. Ви можете створити професійний онлайн магазин на одній з безлічі платформ і CMS, щоб клієнти могли без проблем придбати товар, що їм сподобався.

Привабливий дизайн онлайн-магазину та зручний шлях користувача - це ключові фактори збільшення конверсій. Крім того, більшість платформ пропонують користувачам адаптивні шаблони дизайну, завдяки чому покупці можуть без проблем зробити покупку з мобільного пристрою.

Онлайн-магазин, створений за допомогою існуючої платформи - це ідеальне рішення для невеликого або середнього бізнесу, з невисоким або середнім рівнем трафіку.

Вартість таких рішень починається від 29 доларів на місяць. Але, якщо вам потрібні більш просунуті функції, персоналізовані модулі та унікальний дизайн, це коштуватиме набагато дорожче.

Персоналізована платформа для онлайн магазину - це найдорожчий і найзатратніший за часом варіант. У той же час такий сайт дає власнику масу переваг. Ви можете додати будь-які функції, оскільки ви не обмежені можливостями платформи. Ви можете вирішувати, як покупці будуть знаходити товари, додавати їх до кошика та оформляти замовлення. Крім того, ви маєте повний контроль над тим, як виглядає онлайн магазин, завдяки чому ви можете персоналізувати його під нішу бізнесу.

Такий вид e-commerce розробки найкраще підходить великим компаніям із великим каталогом товарів та великою кількістю трафіку. Також персоналізована платформа дуже популярна для створення торгових майданчиків, таких як Amazon та eBay.

Ви можете інтегрувати онлайн магазин з будь-яким способом оплати та CRM.

Але для такого виду розробки онлайн магазинів, вам буде потрібна допомога команди e-commerce розробників, які створять сайт з нуля. Крім того, у вас має бути бачення готового сайту, включаючи функції панелі адміністратора, архітектуру онлайн магазину та шляхи користувача.

Висновки за розділом 1, мета і задачі проектування

В результаті проведеного аналізу можна зробити наступні висновки. Web-додаток є важливою частиною для розширення бізнесу, онлайн-магазин повинний містити кошик та оплата товару, web-додаток повинен бути адаптивним до різних пристроїв та швидкостей звязку, потрібно обрати відповідну бібліотеку те фреймворк для швидкої розробки та налагодження додатку, користувацький інтерфейс повинен бути простим та зручним у використанні, прив'язка товару до потрібної категорії.

Метою роботи є розробка веб-додатку для автоматизації роботи піцерії, який забезпечує адаптацію до різних пристроїв та швидкостей звязку.

Для досягнення мети потрібно розвязати наступні задачі:

- розробити текст програми та здійснити відлагодження;
- вибір методу що найбільш підходить для даної розробки;
- виконати налаштування розробленого web-додатку;
- розглянути питання роботи охорони праці.

					151.2347СТ.КР.ПЗ.Р2		
Змн	Арк	№ докум.	Підпис	Дата			
Розроб.		Поянов І.І.			*Вибір бібліотек та технологій для будування веб-сайту*	Літ.	Арк 15 / Аркушів
Керівник		Черно О.О.					
Зав.каф.		Черно О.О.				НУК ім. адмірала Макарова	
Н. контр.		Вінниченко І.Л.					

РОЗДІЛ 2

ВИБІР БІБЛІОТЕК ТА ТЕХНОЛОГІЙ ДЛЯ БУДУВАННЯ ВЕБ-САЙТУ

2.1 Web-pack як збірник для аналізу модулів додатку і створення залежностей

Webpack - це статичний модульний збирач для додатків на JavaScript [2].

Навіщо потрібний Вебпак

Програми, написані на JavaScript, постійно ускладнюються, тому для збору модулів все частіше використовують спеціальний інструмент - бандлер. Подібні інструменти дозволяють розробникам упаковувати, компілювати та загалом організовувати всі ресурси, необхідні для проекту. Можна використовувати не лише сторонні бібліотеки, а й власні файли. Подібна модульна система дозволяє досягти кращої організації проекту, оскільки він розбивається на невеликі модулі.

Вебпак зараз є однією з найпотужніших подібних бандлерів, тобто модульний збирач. Він має відкритий вихідний код і дозволяє вирішувати безліч завдань. Як і інші інструменти розробника, вебпак має свої плюси та мінуси.

Переваги: він добре підходить для роботи з односторінковими програмами. Також веб-пакет може здійснювати просунуте розділення коду. Через ці та інші переваги він є одним з найбільш популярних інструментів JS-розробки на даний момент.

Недоліки: трохи складно розібратися у його роботі, частина документації застаріла через велику кількість змін в оновленнях.

Базові поняття

Вебпак - дуже гнучкий інструмент. Щоб почати працювати з ним, необхідно ознайомитися з чотирма базовими поняттями

- Entry - вхід;
- Output - вивід;
- Loaders - завантажувачі;
- Plugins - плагіни.

Entry (вхід) мається на увазі точка входу (entry point), яку вебпак використовуватиме побудови внутрішнього графа залежностей. Після введення точки входу вебпак зможе зрозуміти, які модулі та бібліотеки безпосередньо і не безпосередньо зв'язуються.

В результаті кожна залежність перетворюється на файли, які називаються бандлами (bundles можна перекласти як пакети або вузли).

Output (вивід) вказує на те, де вебпак повинен розміщувати збірку створених бандлів і як він називатиме ці файли (за замовчуванням це ./dist). Налаштувати цю частину процесу можна в полі output у конфігурації:

Loaders (завантажувачі) дозволяють вебпаку обробляти як файли JavaScript, т.к. сам по собі бандлер розуміє лише JS.

Завантажувачі трансформують всі типи файлів у модулі, які потім можна додати до графи залежностей програми (отже, і в бандл рис. 2.1.).

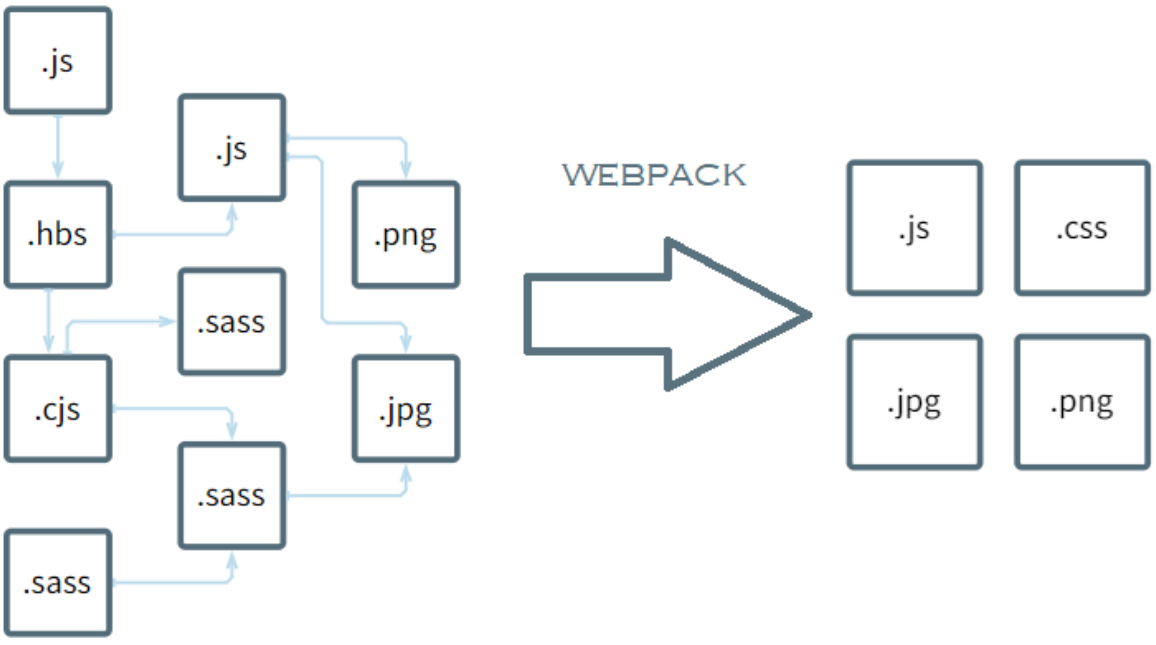

Рис. 2.1. Loaders для вебпака

Використання завантажувачів має дві цілі:

- властивість test визначає, які файли мають бути трансформовані;
- властивість use вказує, який завантажувач повинен використовуватись для виконання трансформації.

Якщо завантажувачі використовуються для трансформації певних типів модулів, плагіни можуть бути використані для виконання набагато ширшого списку завдань.

Щоб використовувати плагін, необхідно використовувати require() і додати його до масиву плагінів. Більшість плагінів можна кастомізувати через налаштування. Оскільки один плагін може використовуватися кілька разів для різних цілей, необхідно створити кілька окремих екземплярів, використовуючи оператор new.

Вебпак - корисний та гнучко настроюваний інструмент для розробки, який постійно допрацьовується та покращується.

2.2 Бібліотеки React і ReduxToolkit для створення користувацького інтерфейсу

React - це JavaScript-бібліотека для створення інтерфейсів користувача [3]. Треба звернути увагу, що це бібліотека, а не фреймворк. React часто називають фреймворком, але це помилка. По-перше, його використання ні до чого вас не зобов'язує, не формує "фрейм" проекту. По-друге, React виконує єдине завдання: показує на сторінці компонент інтерфейсу, синхронізуючи його з даними програми, і лише цієї бібліотеки в загальному випадку недостатньо для того, щоб повністю реалізувати проект.

Незабаром після появи, React і подібні до нього рішення (Vue.js, Svelte) практично захопили світ фронтенду: тому що вони допомагають

вирішувати проблеми, ґрунтуючись на ідеї декларативного програмування, а не на імперативному підході.

Декларативний підхід полягає у описі кінцевого результату (що хочемо отримати).При імперативному підході описуються конкретні кроки задля досягнення кінцевого результату (як ми хочемо щось отримати).

Виявилося, що декларативний підхід чудово підходить для створення інтерфейсів, і він прижився у суспільстві. Цей підхід працює не тільки в Інтернеті: порівняно недавно компанія Apple представила фреймворк SwiftUI, заснований на тих же принципах.

Додаток на чистому HTML та JS

У рамках імперативного підходу покрокові інструкції до створення програми виглядають так:

1) оголошуємо початкові значення програми: надали константам посилання на DOM-елементи, встановлюємо початкове значення лічильника;

2) пишемо обробник increment, в якому ми збільшуємо поточне значення, та встановлюємо його у відповідний елемент;

3) встановлюємо початкове значення лічильника (0);

4) встановлюємо обробник для кнопки.

Приклад такого коду наведений на рис. 2.2.

```
<html>
 <head>
   <title>index</title>
   <meta charset="UTF-8" />
 </head>
 <body>
   <div class="root"><h1 id="counter-value"></h1>
   <button id="increment-btn">+1</button></div>
   <script>
     const counterValueElement = document.getElementById("counter-value");
     const incrementBtn = document.getElementById("increment-btn");
     let counterValue = 0;
     function increment() {counterValue += 1;
       counterValueElement.innerText = counterValue;  }
     counterValueElement.innerText = counterValue;
     incrementBtn.addEventListener("click", increment);
   </script>
 </body>
</html>
```

Рис. 2.2. Приклад коду чистому HTML та JS

Треба звернути увагу, що HTML-розмітка та JS-логіка зберігаються окремо один від одного.

Додаток на React

Через специфіку бібліотеки код на React може виглядати незвично для того, хто пише на JavaScript: тому що в тезі практично немає верстки.

Ось що в ньому відбувається:

- викликаючи функцію React.useState, ми повідомляємо React, що збираємося використовувати якесь значення, що змінюється. У відповідь React дає нам значення (value) і функцію, яка дозволить його встановити (setValue).

- оголошуємо обробник increment, який встановлює нове значення лічильника за допомогою вищезгаданої функції.

- створюємо розмітку, використовуючи синтаксис, схожий на HTML (насправді це JSX). Розмітка повторює те, що бачили раніше, але тепер саме значення лічильника та встановлення обробника на клік знаходяться прямо в ній. Це те місце, де ми описуємо кінцевий результат.

Весь код знаходиться усередині функції App. У React вона та інші схожі функції називаються компонентами. Компонент - це фрагмент інтерфейсу, який містить розмітку і, при необхідності, пов'язану з нею логіку. Всі програми React будуються на компонентах. При цьому сам компонентний підхід з'явився задовго до React, але його поєднали з декларативністю (рис. 2.3.).

```
<body>
   <div id="root"></div>
   <script src="https://unpkg.com/react@17/umd/react.development.js" crossorigin></script>
   <script src="https://unpkg.com/react-dom@17/umd/react-dom.development.js" crossorigin ></script>
   <script   src="https://unpkg.com/@babel/standalone@7.13.6/babel.min.js" crossorigin ></script>
   <script type="text/babel">
     function App() {
       const [value, setValue] = React.useState(0);
       function increment() { setValue(value + 1); }
       return (
         <div className="app">
           <h1>{value}</h1>
           <button onClick={increment}>+1</button>
         </div>  );  }
     ReactDOM.render(
       <div>
         <App />
       </div>
       document.getElementById("root") );
   </script>
 </body>
```

Рис. 2.3. Приклад коду на React

Порівняння двох варіантів програми

У першому випадку ми написали алгоритм для роботи з елементами, значенням та його зміни - кроки, необхідні для досягнення результату.

У другому, використовуючи JSX-розмітку та допоміжні функції React, ми одразу описали результат, який хочемо бачити. У цьому полягає відмінність декларативного підходу від імперативного.

Якщо порівнювати ці додатки, то при використанні React можна виділити такі особливості:

- Розмітка і логіка, що відноситься до неї, знаходяться поруч і пов'язані один з одним. Це спрощує подальшу роботу з кодом;
- Виділено лічильник із кнопкою в компонент. Це означає, що ми можемо дуже легко його перевикористати: достатньо на 44 рядку написати ще один тег <App />, і на сторінці з'являться вже два незалежні лічильники;
- Більше не потрібно використовувати ідентифікатори для звернення до DOM-елементів, що також робить код, що легко підтримується;
- Стан компонента ізольований: немає жодної можливості модифікувати значення ззовні, якщо ми так не замислювалися. Завдяки цьому потік даних у додатку стає більш передбачуваним, що спрощує розробку та налагодження.

Також варто зазначити, що у React-додатках ми не працюємо безпосередньо з DOM-деревом. Натомість ми описуємо розмітку за допомогою JSX, а React вже сам вирішує, як перетворити її на реальні DOM-елементи. Це стає можливим завдяки абстракції, яка називається віртуальний DOM.

Для яких проектів підійде React

Резюмуючи всі особливості, можна назвати кілька типів проектів, яким підійде React.

- Якщо проект планує розширюватись, і над ним працює чи працюватиме команда розробників. Тому що в такому разі використання відомої технології допоможе простіше домовлятися між розробниками та краще підтримувати код;
- Середнім і великим проектам буде корисний компонентний підхід, який лежить в основах React. Це спростить структурування та перевикористання коду та дасть виграш у довгостроковій перспективі;
- Legacy-проекти, які мають пройти через рефакторинг. Тому що React можна додавати до вже існуючого проекту, оновлюючи кодову базу поступово та точково.

React, можливо, не підійде для простих додатків (односторінкових та одноразових сайтів), тому що в цьому випадку для того, щоб розібратися з налаштуванням проекту та оточення, вже знадобиться багато часу та праці.

Ще React буде не найуспішнішим вибором для реалізації елементів проекту, які дуже чутливі до споживаних ресурсів. У цьому випадку можливе гібридне використання: коли сама програма здебільшого написана на React, а вимогливі до продуктивності місця з ним не взаємодіють - бібліотека ніяк не заважає робити такі інтеграції. Однак і тут слід підходити до питання без фанатизму: React є досить швидким у більшості випадків, а оптимізувати вузькі місця завжди можна за фактом.

Redux Toolkit - це пакет, який полегшує роботу з Redux [4]. Він був розроблений для вирішення трьох основних проблем:

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

В даний час розробка левової частки веб-застосунків, заснованих на фреймворку React, ведеться з використанням бібліотеки Redux. Ця бібліотека є найпопулярнішою реалізацією FLUX-архітектури і,

незважаючи на низку очевидних переваг, має дуже істотні недоліки, такі як:

- складність і "багатослівність" рекомендованих патернів для написання та організації коду, що тягне за собою велику кількість бойлерплейту;
- відсутність вбудованих засобів управління асинхронною поведінкою та побічними ефектами, що призводить до необхідності вибору відповідного інструменту з безлічі аддонів, написаних сторонніми розробниками.

Для усунення цих недоліків розробники Redux презентували бібліотеку Redux Toolkit. Цей інструмент є набір практичних рішень та методів, призначених для спрощення розробки додатків з використанням Redux. Розробники цієї бібліотеки мали на меті спростити типові випадки використання Redux. Цей інструмент не є універсальним рішенням у кожному з можливих випадків використання Redux, але дозволяє спростити код, який потрібно написати розробнику.

2.3 Обрання зручного збирача програми для отримання кінцевої верстки

Найкращий і зручний збирач проектів на даний момент вважається gulp.

Gulp - інструмент для автоматизації рутинних завдань, що виникають під час веб-розробки [5]. Це може бути не тільки frontend технологія, це може бути і backend технологія.

Якщо ви працюєте з такими технологіями, як html, css, javascript і т.д. Якщо ви впровадите в практику своєї роботи такий інструмент як gulp, ви

значно прискорите швидкість роботи і, по-друге, цей інструмент "відкриє вам дорогу" до нових можливостей, які значно підвищать рівень веб-розробки та знань.

Gulp - просто програма, написана мовою програмування Javascript. Для того, щоб почати користуватися, бажано знати, хоча б основи мови Javascript. Якщо ви знаєте, то користуватися програмою Gulp для вас буде набагато простіше.

Сенс наступний: ми створюємо для системи Gulp деякі завдання. Тобто, описуємо ці завдання мовою Javascript. Потім, Gulp просто виконує ці завдання у потрібній послідовності, оскільки ми це прописали. Тобто, Gulp - це просто система управління завданнями з веб-розробки. Також її називають task manager. task - завдання, manager - керувати.

Важливо розуміти, що Gulp - просто якесь ядро, до якого ми прикручуємо додаткові модулі, плагіни, які "вчать" Gulp виконувати якусь певну функцію чи роботу. Встановлюючи ці плагіни, ми отримуємо нові можливості в системі Gulp, які ми можемо використовувати.

Якщо повернутися на офіційну сторінку сайту Gulp і перейдемо в розділ Plugins, то тут ми можемо знайти велику кількість плагінів, які дозволяють виконувати якісь певні рутинні завдання.

Вибираємо потрібний плагін, встановлюємо його в цю систему та користуємося ним.

Розглянемо основні з них. Список цих завдань може бути досить великий.

Завдання 1. Мініфікація коду.

Це одна з найчастіших завдань, для якої найчастіше використовують Gulp і подібні до неї системи - це завдання мініфікації коду. Тобто написали якийсь код, якоюсь мовою програмування. Для вас цей код сприймається добре, але якщо ви розмістите цей код на робочому сервері, на якому будн розміщуватися сайт, то, відповідно, цей код

завантажуватиметься досить довго через те, що в ньому є багато зайвої інформації у вигляді відступів, коментарів і т.д.

Gulp дозволяє забрати все зайве з коду, підготувати його для того, щоб це можна було викласти на робочий сервер.

Завдання 2. Об'єднання коду з різних файлів на один.

Ви можете об'єднувати код із CSS, Javascript файлів і.т.д. в один. Це важливо зробити також через швидкість завантаження документа. Під час роботи з протоколом http кожен запит до файлу - це додатковий час завантаження сторінки.

Якщо ви об'єднуєте код в один файл, завантажити його простіше та швидше, ніж завантажити кілька файлів.

Це типове завдання, яке доводиться вирішувати у сучасній веб-розробці. Написати програму набагато простіше, якщо її код розбитий на модулі та незалежні частАини.

Завдання 3. Робота з CSS препроцесорами: sass, less, …

Система Gulp дозволяє вам використовувати їх у своїй роботі та ви отримаєте такий потужний інструмент для того, щоб покращити свої навички веб-розробки.

Завдання 4. Підтримка нових стандартів Javascript.

Оскільки мова Javascript є клієнтською мовою програмування, то він залежить від того браузера, на якому буде працювати. Якщо відвідувач сайту користується старими браузерами, у нього нові стандарти не будуть працювати. За допомогою Gulp ви можете вирішити це завдання.

Gulp - не єдиний інструмент, який дозволяє вирішувати такі завдання. Важливо розуміти, що Gulp - це одне з найпростіших і найлегших рішень, які дозволяють це зробити.

Код написаний під Gulp інтуїтивно зрозумілий та проект, який у вас буде виходити, досить компактний та зручний. У ньому буде все найнеобхідніше, що потрібне для веб-розробки.

Gulp - альтернатива, яка дозволить вирішувати ці завдання простіше та швидше.

Висновки за розділом 2

У даному розділі кваліфікаційної роботи було обрано технології для розробки web-додатку. До складу технологій розробки входять: React, Redux Toolkit, React Router v6, Fetch, React Hooks, Prettier, CSS-Modules, React Pagination.

За допомогою прикладів на рис. 2.2 та рис. 2.3. побачили різницю написання коду за допомогою звичайного hetml, css, js та React. React виявився кращим рішенням для написання додатку, за рахунок своєї швидкості розробки та можливості зменшити об’єм написаного коду програми та збільшити швидкість додатку.

Розглянуто основні завдання Gulp у розробці, а саме: мініфікація коду, обєднання коду з різних файлів, робота з препроцесорами, підтримка стандартів js.

					151.2347СТ.КР.ПЗ.Р3		
Змн	Арк	№ докум.	Підпис	Дата			
Розроб.		Поянов І.І.			*Розробка, налаштування та відлагодження додатку, проведення тестів*	Літ.	Арк 27 / Аркушів
Керівник		Черно О.О.					
Зав.каф.		Черно О.О.				НУК ім. адмірала Макарова	
Н.контр		Вінниченко І.Л.					

РОЗДІЛ 3

РОЗРОБКА, НАЛАШТУВАННЯ ТА ВІДЛАГОДЖЕННЯ ДОДАТКУ, ПРОВЕДЕННЯ ТЕСТІВ

3.1 Налаштування середовища розробки VS Code та встановлення всіх необхідних бібліотек.

Створення будь якого web-додатку починається з обрання середовища, де буде проходити написання коду. Редактор коду обирається в залежності від функціоналу, який буде оброблюватись та зручність розробки з певними бібліотеками та збиральниками.

В даній кваліфікаційній роботі було обрано «Virtual Studio code».

VSCode - найпопулярніший редактор коду серед веб-розробників, його можна використовувати з нуля, проте з часом його базових функцій недостатньо [6].

Visual Studio Code - редактор вихідного коду. Він має багатомовний інтерфейс користувача та підтримує ряд мов програмування, підсвічування синтаксису, IntelliSense, рефакторинг, налагодження, навігацію за кодом, підтримку Git та інші можливості. Багато можливостей Visual Studio Code недоступні через графічний інтерфейс, часто вони використовуються через палітру команд або JSON-файли (наприклад, налаштування користувача). Палітра команд є подібністю командного рядка, що викликається поєднанням клавіш.

VS Code також дозволяє замінювати кодову сторінку за збереження документа, символи перекладу рядка та мову програмування поточного документа.

Visual Studio Code має підтримку плагінів, доступних через Visual Studio Marketplace. Вони можуть включати доповнення до редактора, підтримку додаткових мов програмування, статичні аналізатори коду.

Розширення в редакторах дозволяють пришвидшити розробку, їх перевага в тому що можуть вирівнювати код, що робить його більш сприйнятим при читанні та візуально зрозумілим з точки зору вкладеності елементів структури програм. Деякі розширення дозволяють створювати готовий шаблон для компоненту, інші дозволяють автоматично імпортувати компоненти завдяки чому виникає менше помилок при створення додатку.

Для написання програми будуть використовуватись такі розширення:

- Auto Import - Автоматично знаходить, аналізує та забезпечує дії з кодом та завершення коду для всіх доступних імпортів. Працює з Typescript і TSX;
- ES7 + React/Redux snippets - розширення надає фрагменти JavaScript та React/Redux в ES7+, і значно спростить написання коду, завдяки гарячим клавішам дозволяє створити готовий шаблон який автоматично експортується;
- HTML CSS Support - Плагін автоматично доповнює назву ID або HTML-атрибута для визначень, знайдених у робочій області, на які посилається link;
- IntelliSense for CSS class names in HTML - Аналізує робоче оточення для додавання можливості автодоповнення CSS класів;
- JS JSX Snippets - сніппет, тобто він дозволяє ввести лише коротке поєднання клавіш, а замість нього з'явиться цілий рядок коду. Вручну майже нічого не треба робити, що економить час;
- Prettier - засіб для форматування коду, який націлений на використання жорстко заданих правил оформлення програм.

Оскільки додаток буде створюватись за допомогою React, потрібно встановити необхідне середовище для його функціонування Node JS.

Node.js - середовище виконання JavaScript. Оточення Node.js включає все, що потрібно для виконання програми, написаної на JavaScript.

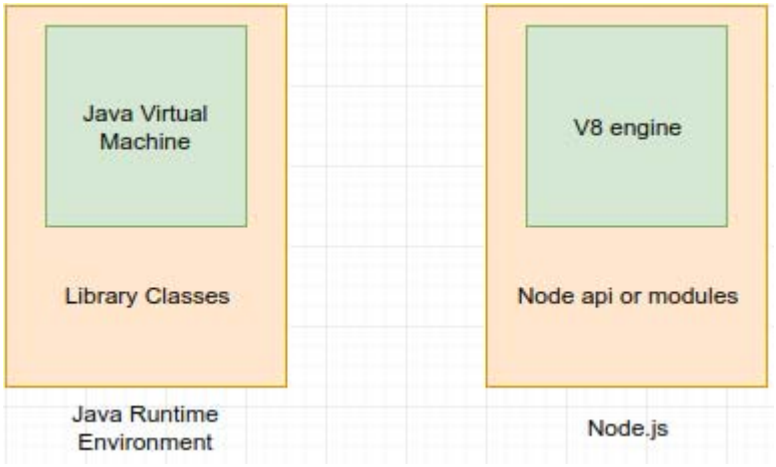

Рис. 3.1. Різниця звичайного JS від Node JS

Раніше ви могли запустити JavaScript тільки в браузері, але одного разу розробники розширили його, і тепер ви можете запускати JS на своєму комп'ютері як окрему програму. Так виник Node.js.

Тепер ви можете зробити набагато більше з JavaScript, ніж просто інтерактивні веб-сайти.

Тепер JavaScript має можливість робити те, що можуть робити інші скриптові мови програмування, такі як Python.

Обидва - браузерний JavaScript та Node.js запускаються у середовищі виконання V8. Цей двигун використовує JS код, і перетворює його на швидший машинний код. Машинний - низькорівневий код, який може запускати комп'ютер без необхідності спочатку його інтерпретувати.

I/O означає введення/виведення. Це може бути будь-що: від читання/запису локальних файлів до HTTP-запиту в API. I/O займає час і, отже, блокує інші функції.

Розглянемо приклад, в якому ми запитуємо user1 та user2 з бекенду, а потім друкуємо їх на екрані/консолі. Відповідь на цей запит вимагає часу, але обидва запити даних можуть виконуватися незалежно і в один і той же час.

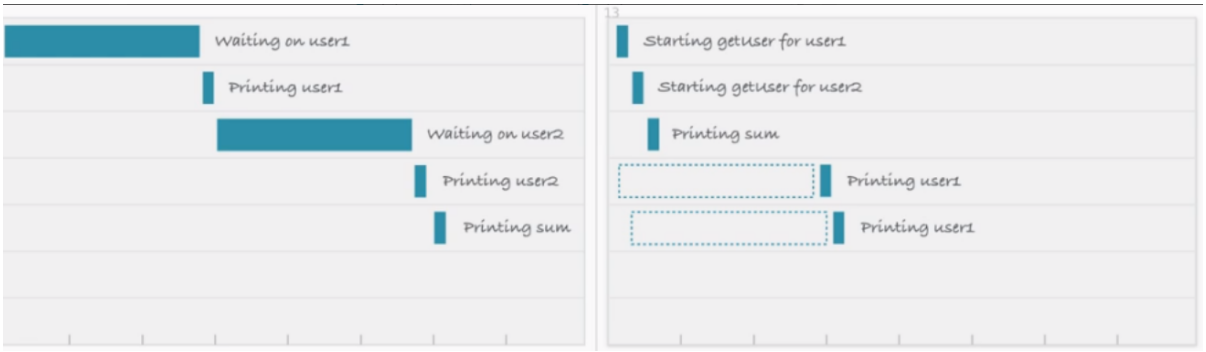

Рис. 3.2. Запит з Back-end

У методі блокування запит даних user2 не запускається, доки дані user1 не будуть надруковані на екрані.

Якщо це був веб-сервер, нам потрібно було б почати новий потік кожного нового користувача. Але JavaScript однопоточний (він має однопотоковий цикл подій, про який ми поговоримо трохи пізніше). Таким чином, це зробить JavaScript не дуже підходящим для багатопотокових задач.

З іншого боку, використовуючи неблокуючий запит, можна ініціювати запит даних для user2, не чекаючи відповіді на запит user1. Ви можете ініціювати обидва запити паралельно.

Введення-вивід, що не блокує, усуває необхідність багатопоточності, оскільки сервер може обробляти кілька запитів одночасно.

3.2 Верстка. Адаптація додатку на інших пристроях

Верстка сайту - один із найважливіших етапів розробки онлайн-ресурсу, в результаті якого нарисований дизайнером макет перетворюється на HTML та CSS-код. Це завдання потребує особливих навичок. Щоб якісно згорнути HTML-код, потрібні глибокі знання особливостей роботи браузерів, семантики веб-сторінок, принципів позиціонування елементів.

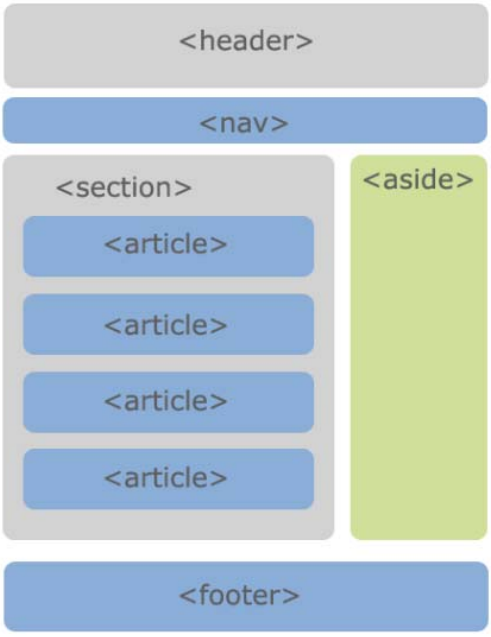

Рис. 3.3. Семантична верстка сторінки

Семантична верстка або семантичний HTML-код - підхід до створення веб-сторінок на мові HTML, заснований на використанні HTML тегів відповідно до їх семантики (призначення), а також передбачає логічну та послідовну ієрархію сторінки [7].

Сучасні види верстки сайту

Раніше, щоб зверстати сайт, використовували HTML-таблиці: кожен елемент містився в окремій клітинці, що розв'язувало проблеми з позиціюванням контенту. Однак код виходив складним для підтримки. На зміну таблиць прийшла блокова верстка: елементи містилися в порожні, незалежні контейнери та розміщувалися на сторінці за допомогою різних CSS-властивостей.

Інша проблема, яку потрібно вирішити при верстці - зображення сторінок при різних розширеннях екрану.

Підхід, при якому розміри блоків та інших елементів вказуються у відсотках до ширини екрану або батьківських елементів називають верстkою. Попри всі переваги, таке рішення не можна назвати ідеальним. Подібна буде погано виглядати на широких та вузьких екранах: в таких

випадках контент буде або занадто розповзатися, або занадто стискатися. Вивчати його буде незручно.

Верстка - більш сучасне і правильне рішення, яке передбачає написання різних правил оформлення для екранів з різним розширенням. Це більш трудомісткий процес. Однак це автоматично вирішує задачу з підготовки мобільної версії сайту: спеціально робити нічого не потрібно, оскільки сторінка буде підлаштовуватися під невеликий екран.

До написаного коду 2 вимоги. По-перше, верстка повинна бути валідною: грубі семантичні помилки не допускаються, бо це призводить до проблем з просуванням сайту в пошукових системах. Вплив відчутний. По-друге, сайт повинен бути кросбраузерним, тобто однаково виглядати у всіх популярних версіях браузерів, які використовує цільова аудиторія замовника.

Отже маємо макет, який нам потрібно зібрати в звичайний SPA.

Односторінкова програма - це веб-додаток або веб-сайт, що використовує єдиний HTML-документ як оболонку для всіх веб-сторінок і організує взаємодію з користувачем через HTML, CSS, JavaScript, що динамічно підвантажуються, зазвичай за допомогою AJAX.

В розробці будемо використовувати БЕМ

БЕМ (Блок, Елемент, Модифікатор) – компонентний підхід до веб-розробки [8]. У його основі лежить принцип поділу інтерфейсу на незалежні блоки. Він дозволяє легко та швидко розробляти інтерфейси будь-якої складності та повторно використовувати існуючий код, уникаючи «Copy-Paste».

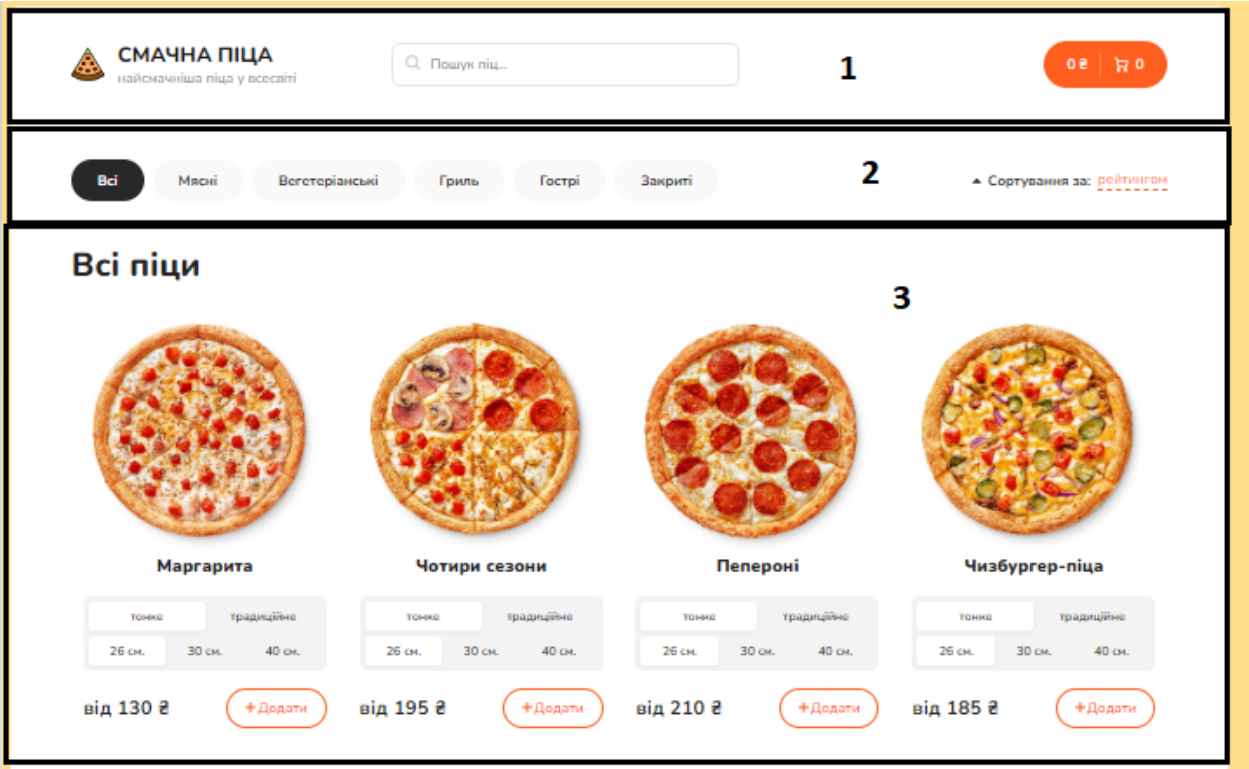

Рис. 3.4. Основна сторінка web-додатку

Макет складається з 3 частин:

- Header – шапка сайту, в яку входить логотип з назвою та корзина замовлень
- Nav – меню навігації завдяки якому користувачі зможуть швидко знайти потрібну піцу
- Main - основна секція в якій знаходиться сам товар для замовлення в якому міститься фото продукту, його назва, кнопка додавання в корзину та деталі для замовлення.

Добавляючи семантичні HTML-теги на вашій сторінці, ви даєте доповнювальну інформацію, яка допомагає пошуковикам розглядати ролики та відносну важливість різних частин ваших сторінок.

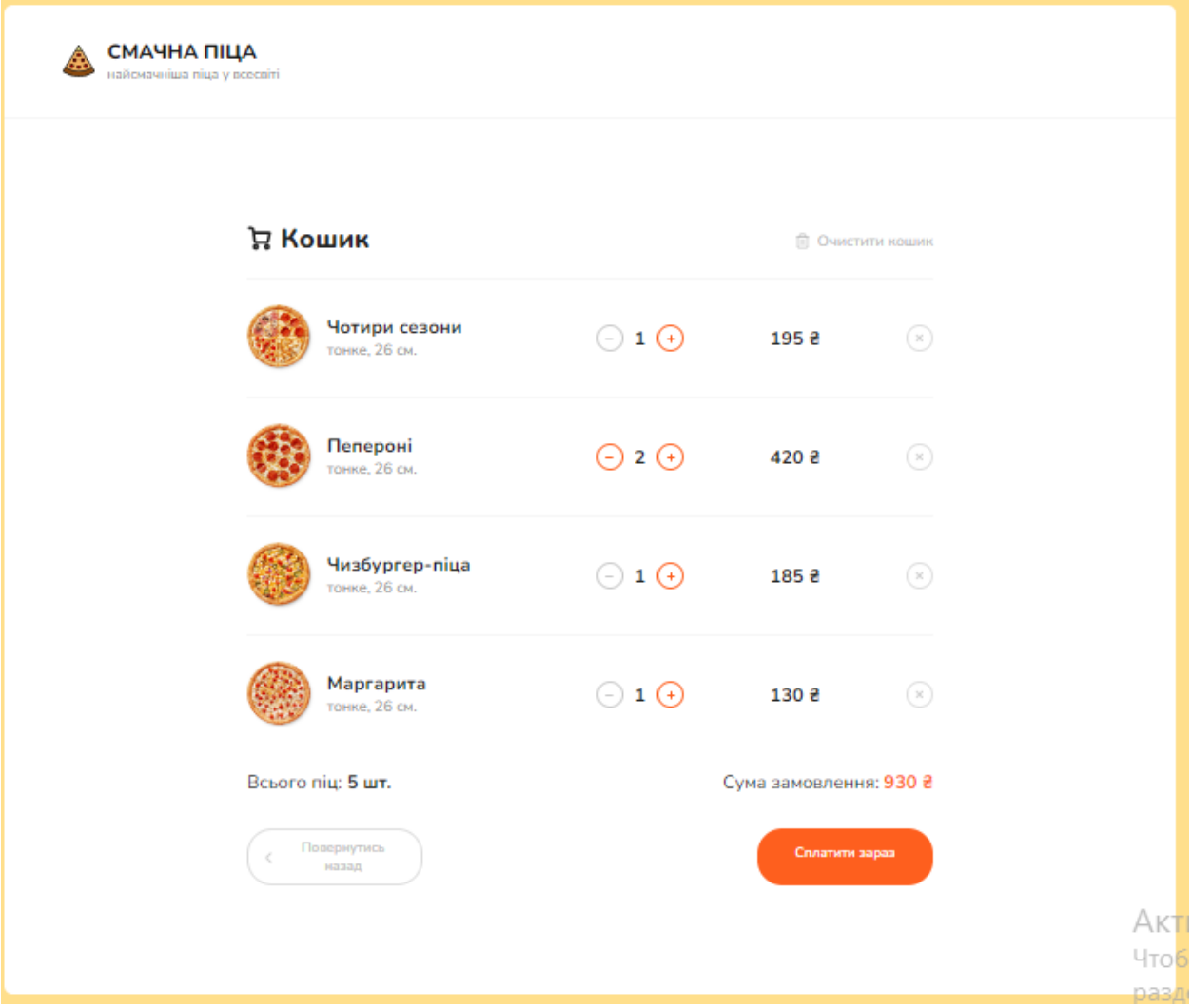

Рис. 3.5. Кошик замовлень

Кошик замовлень повинна містити замовлення, які користувач вибрав на основній сторінці та додав до кошика, та порахувати суму замовлення та кількість замовлених піц.

Після успішної верстки додатку потрібно його адаптувати під пристрої з меншими екранами та мобільні телефони, робиться це за допомогою @mediaquery.

Медіа-запити – це правила CSS, які дозволяють керувати стилями елементів залежно від значень технічних параметрів пристроїв. Іншими словами, це конструкції, які дозволяють визначати на підставі деяких умов, які стилі необхідно використовувати на веб-сторінці, а які ні [9].

Медіа-запити призначені для створення адаптивних дизайнів. Адаптивний дизайн відрізняється від інших тим, що він може "пристосовуватися" (видозмінюватися) залежно від того, яку ширину екрана має пристрій (браузер).

Але під час створення адаптивних веб-сторінок також необхідно звернути увагу на метатег viewport. Цей тег забезпечує коректне відображення адаптивних дизайнів сайтів на екранах пристроїв, що мають високу щільність пікселів. Іншими словами, він встановлює відповідність між CSS та фізичною роздільною здатністю веб-сторінки.

Створення медіа-запиту починається з ключового слова @media, після якого вказується одна або кілька умов. Як умову можна вказувати тип пристрою або вимоги до певної характеристики. Вимога до певної характеристики записується у круглих дужках.

Зазвичай адаптація робиться до 350px – це дозволить користувачам мобільних телефонів зручно користуватися додатком.

Результат адаптація наведений на рис. 3.6.

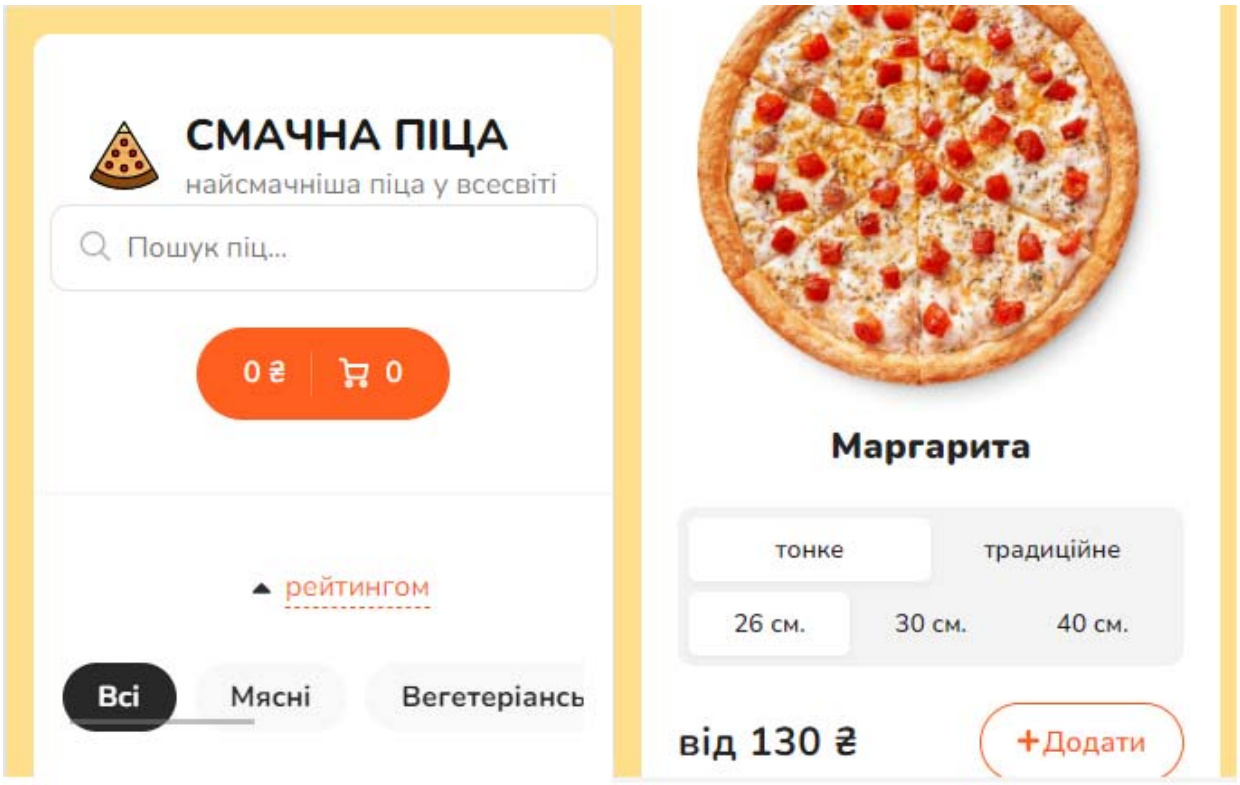

Рис. 3.6. Адаптація web-додатку

Таким чином створена адаптація веб-додатку для мобільних пристроїв, що дозволить користувачам зручно використовувати його.

3.3 Моделювання керуючих функцій для роботи з інтерфейсом

Моделювання керуючих функцій – це основна частина всього web-додатку. Саме тут звичайний SPA перетворюється в привичний для нас MPA.

MPA чи Multi Page Applications за принципом роботи повністю протилежні SPA. MPA — це багатосторінкові програми, які працюють як звичні нам веб-сайти. Вони надсилають запит на сервер і повністю оновлюють сторінку, коли з нею відбувається будь-яка дія (перехід на іншу сторінку, внесення та зміна даних). Подібна архітектура програми значно впливає на швидкість і продуктивність, оскільки більшість даних підвантажується повторно при кожному переході(рис. 3.7).

Рис. 3.7. Принцип роботи MPA

Для перетворення SPA в MPA потрібно відкрити консоль та написати команду для завантаження модулів node js:

npm install -g name-project

Після завантаження всіх модулів в середовищі з’явиться, структура папок зображена на рис. 3.8.

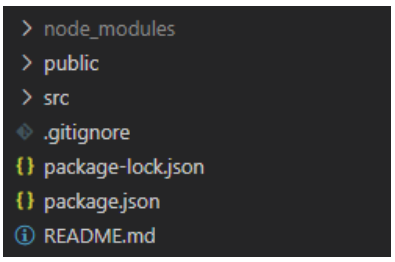

Рис. 3.8. Структура папок React

Нас цікавить тільки src, package.json, саме з цим ми і будемо працювати.

Package.json містить інформацію про всі бібліотеки які ми будемо завантажувати в ході роботи, завдяки ньому можна спостерігати чи встановилась потрібна бібліотека.

Src – це папка, яка містить початкові компоненти, в ній і будемо розбивати нашу веб-сторінку на компоненти.

Спочатку потрібно зрозуміти для чого потрібен VirtualDom.

У React для кожного об'єкта DOM існує відповідний віртуальний об'єкт DOM. Віртуальний об'єкт DOM — це представлення об'єкта DOM, подібно до його полегшеної копії. Віртуальний об'єкт DOM має ті ж властивості, що і реальний об'єкт DOM, але у нього немає реальної можливості безпосередньо змінювати те, що відображається на екрані [10].

«Віртуальна модель DOM (VDOM) — це концепція програмування, в якій ідеальне або «віртуальне» уявлення інтерфейсу користувача зберігається в пам'яті і синхронізується з «реальною» бібліотекою DOM, такою як ReactDOM. Цей процес називається узгодженням».

Маніпулювання DOM відбувається повільно. Керування віртуальним DOM відбувається набагато швидше, тому що на екрані нічого не відображається.

Коли в інтерфейс користувача додаються нові елементи, створюється віртуальна модель DOM, представлена у вигляді дерева. Кожен елемент є вузлом у цьому дереві. Якщо будь-який з цих елементів змінюється, створюється нове віртуальне дерево DOM. Потім дерево порівнюється з попереднім віртуальним деревом DOM.

Як тільки це буде зроблено, віртуальна DOM обчислює найкращий із можливих методів внесення цих змін до реальної DOM. Це гарантує мінімальну кількість операцій із реальною DOM. Отже, зниження вартості поновлення реальної моделі DOM.

На рис. 3.9. показано віртуальне дерево DOM та процес порівняння.

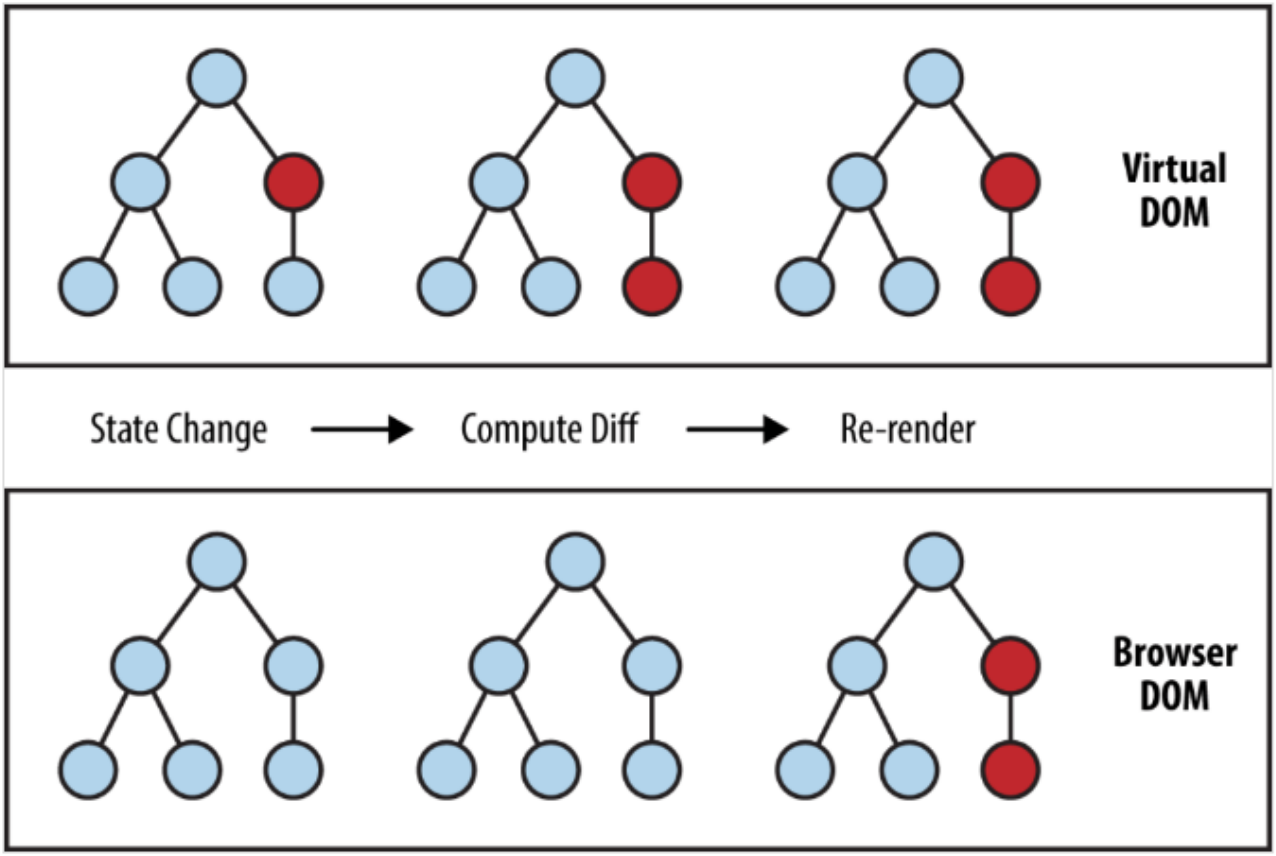

Рис. 3.9. Процес порівняння DOM

Червоні кружки є вузлами, які змінилися. Ці вузли представляють елементи інтерфейсу користувача, стан яких було змінено. Потім обчислюється різниця між попередньою версією віртуального дерева DOM та поточним віртуальним деревом DOM. Потім все батьківське піддерево повторно візуалізується для отримання оновленого інтерфейсу користувача. Це оновлене дерево потім пакетно оновлюється до реальної моделі DOM.

Оскільки тепер знаємо як працює DOM, можемо встановити:

nmp install react-router-dom

React Router це стандартна бібліотека маршрутизації (routing) у React. Він зберігає інтерфейс програми, синхронізованим з URL на браузері. React Router дозволяє маршрутизувати "потік даних" (data flow) у додатку зрозумілим способом. Він подібний до твердження, якщо у вас є даний URL, він буде подібний до цього Route (маршруту), і інтерфейс буде таким!

React Router насправді має користь і широко використовується в програмах React більше з боку Server, ніж програми React з боку Client. Точніше React Router зазвичай використовується в додатку React в середовищі NodeJS Server, він дозволяє вам визначати динамічні URL, але відповідні філософії "Single Page Application" (Односторінковий додаток) у React. Для розробки програми React вам потрібно написати багато Component, але потрібен лише один файл для обслуговування користувачів, це index.html.

React Router допомагає визначити динамічні URL, і вибрати відповідний Component для render (зображення) на браузері користувача відповідно кожному URL.

Для того щоб він почав працювати потрібно імпортувати його в наш основний javascript файл index.js, потім обернути наш додаток <App /> в

<BrowserRouter> виглядатиме це так:

```
root.render(
<BrowserRouter>
    <Provider store={store}>
        <App />
    </Provider>
</BrowserRouter>,
);
```

Рис. 3.10. Код підключення Router

Тепер почнемо розбивати наш додаток на компоненти, React дозволяє створювати великого кількості коду невеликі компоненти, що дозволяє не заплутатись у всьому коду, а працювати лише з необхідним компонентом, а раніше встановлений router дозволить оновлювати лише

компонент який ми змінили, а не цілу сторінку. Як це виглядає наведено на рис. 3.11.

```
function App() {
  const [searchValue, setSearchValue] = React.useState('');

  return (
    <div className="wrapper">
      <SearchContext.Provider value={{ searchValue, setSearchValue }}>
        <Header />
        <div className="content">
          <Routes>
            <Route path="/" element={<Home />} />
            <Route path="/cart" element={<Cart />} />
            <Route path="*" element={<NotFound />} />
          </Routes>
        </div>
      </SearchContext.Provider>
    </div>
  );
}
```

Рис. 3.11. Код компонентної розбивки

В кожному з елементів знаходиться свій код що містить посилання на відповідну сторінку, переходячи за посиланням користувач потрапляє на потрібну йому сторінку: кошика, основу або ж якщо сторінку не знайдено то відобразиться сторінка 404 NotFound.

Наступним етапом після перенесення і налаштування кнопок додамо скелет для піц, це дозволить людям у яких швидкість Інтернету занадто низька спочатку відобразити схожий ще не завантажений компонент, як це виглядатиме на рис. 3.12.

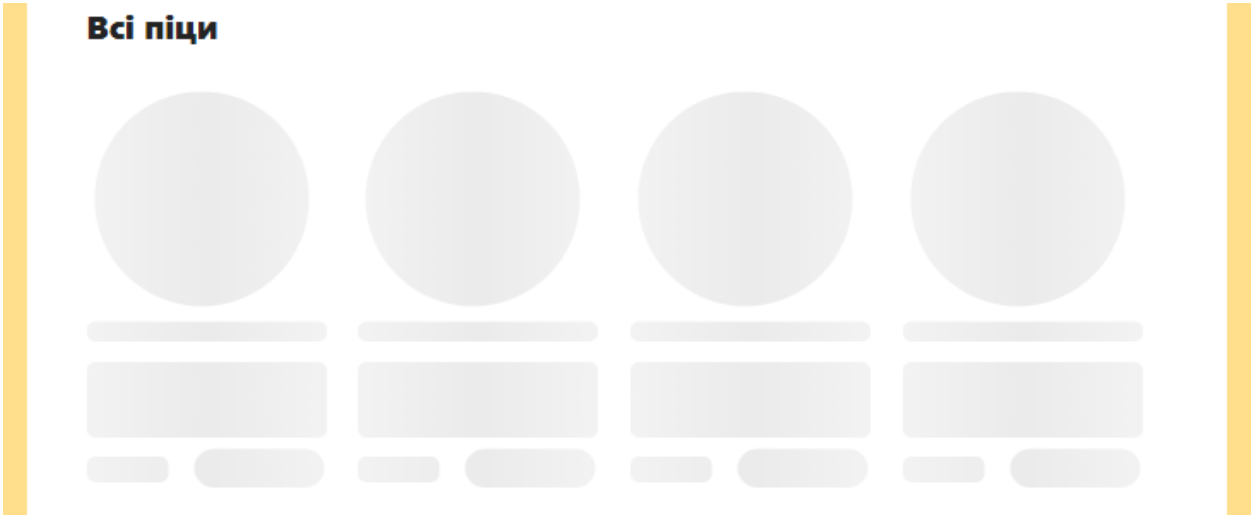

Рис. 3.12. Скелет піц

Для цього було створено новий компонент, а в нього прописано наступну функцію:

```
const Skeleton = () => (
<ContentLoader
className="pizza-block"
speed={2}
width={280}
height={500}
viewBox="0 0 280 500"
backgroundColor="#f3f3f3"
foregroundColor="#ecebeb">
<circle cx="134" cy="136" r="125" />
<rect x="-6" y="275" rx="10" ry="10" width="280" height="32" />
<rect x="-1" y="325" rx="0" ry="0" width="280" height="88" />
<rect x="-1" y="434" rx="10" ry="10" width="95" height="30" />
<rect x="124" y="424" rx="25" ry="25" width="152" height="45" />
</ContentLoader>
);
```

Після чого ми повинні взяти масив наших піц, та прописати йому, якщо швидкість завантаження не встигає завантажити компонент піц то відобразити її скелет:

```
 const pizzas = items.map((obj) => <PizzaBlock key={obj.id} {...obj} />);
 const skeletons = [...new Array(6)].map((_, index) => <Skeleton key={index} />);
<div className="content__items">{isLoading ? skeletons : pizzas}</div>
```

Наступний крок - додавання та рахування піц, які ми додали до кошику, це є основною та самою важкою частиною коду. Реалізовувати будемо це через хуки.

Хуки - функції, за допомогою яких можна «підчепитися» до стану та методів життєвого циклу React із функціональних компонентів. Хуки не працюють усередині класів - вони дають вам можливість використовувати React без класів [11].

Хуки - функції JavaScript, які накладають два додаткові правила:

- хуки слід викликати лише на верхньому рівні. Не викликайте хуки всередині циклів, умов чи вкладених функцій;
- хуки слід викликати лише з функціональних компонентів React. Не викликайте хуки із звичайних JavaScript-функцій.

Схема за якою це працює зображене на рис. 3.13., реалізація у вигляді програмного коду знаходиться в додатку в кінці програми.

Рис. 3.13. Схема визначення стану

Результат виконаної роботи знаходиться на рис. 3.14 - 3.15.

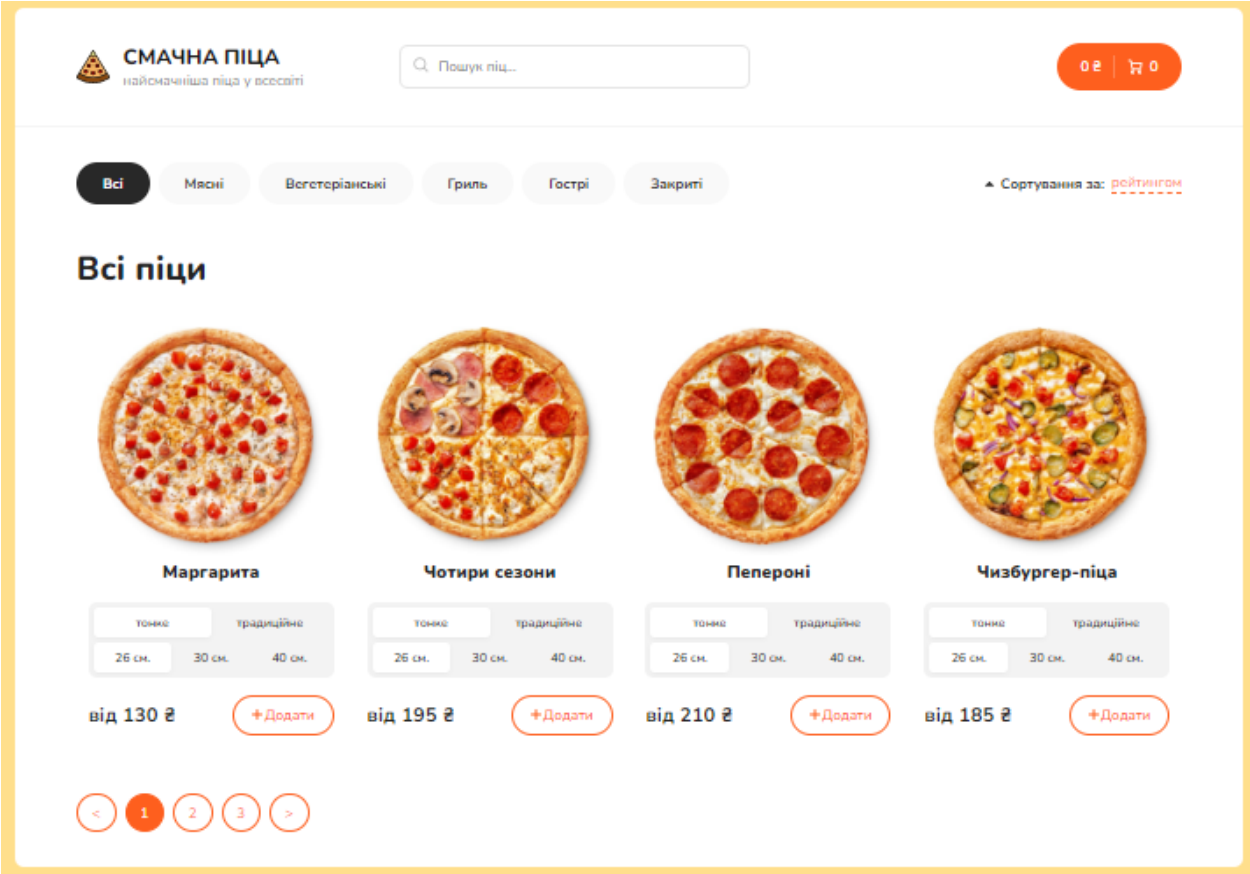

Рис. 3.14. Кінцевий результат

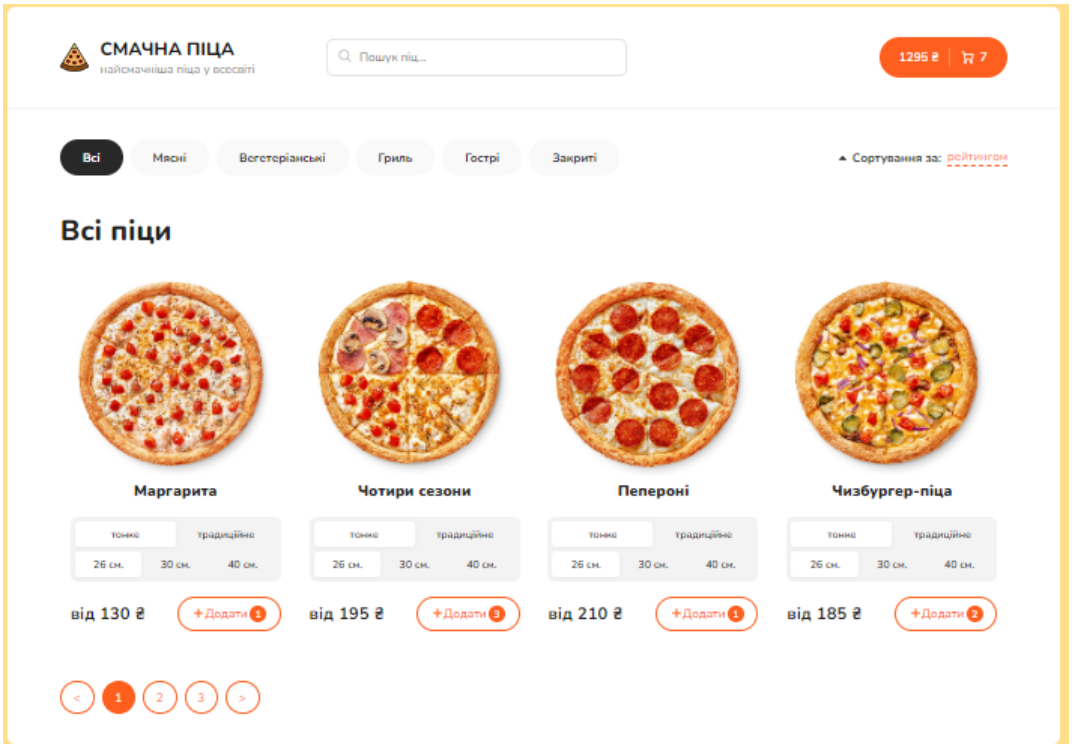

Рис. 3.15. Результат додавання та сума за піци

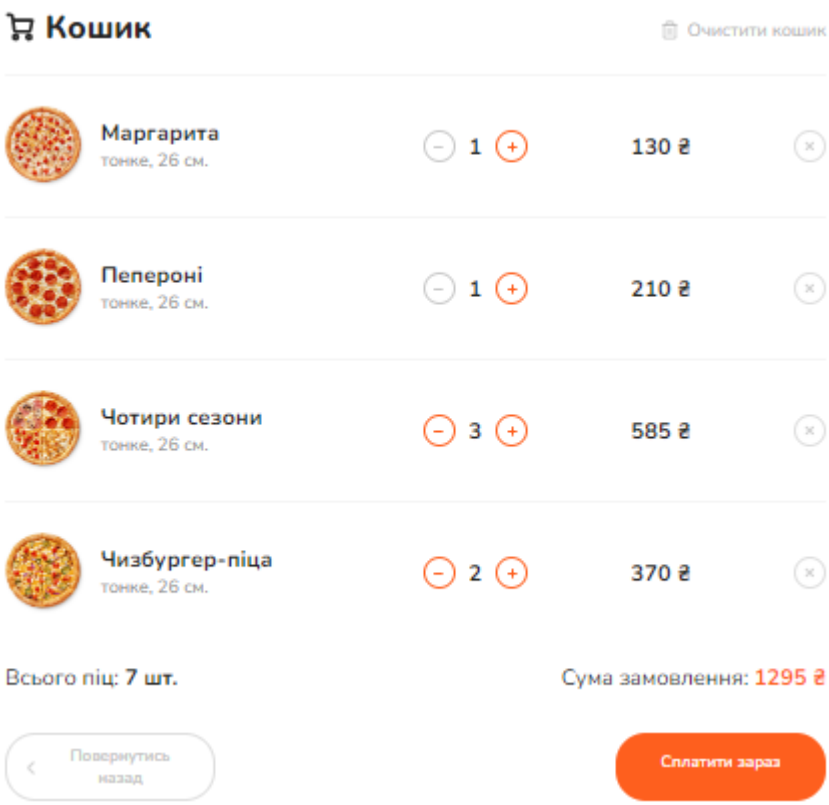

Рис. 3.16. Робота кошика

За допомогою React реалізовано всі сторінки web-додатку та функціонал повністю виконує свою роботу, замовлення піц на основній сторінці, додавання та підрахунок замовлення в корзині, при потребі видалення або замовлення.

3.4. Визначення показників швидкодії додатку

Чим швидше завантажується веб-сторінка, тим зручніша вона для користувачів.

Ні Google, ні інші пошукові системи не розповідають, наскільки вони вимірюють показники швидкості і наскільки вони впливають на ранжування в пошуку. Але сьогодні можна бути впевненими, що зручність сторінок стає важливим фактором пошуку. Оцінка зручності сторінки включає кілька технічних характеристик, зокрема Core Web Vitals —

метрики завантаження, інтерактивності та візуальної стабільності. Якщо коротко, то вони оцінюють не те, як швидко сторінка завантажується, а те, як швидко рендерується перша видима частина.

Баланс між зручністю та швидкістю

Працювати над швидкістю, безперечно, потрібно. Якщо сторінки сайту вантажаться повільно, позиції просядуть разом із показниками перегляду та конверсії. Але для успішного пошукового просування недостатньо просто покращувати показники завантаження - потрібно знаходити баланс між швидкістю та зручністю для користувачів.

UX важливий для всіх пошукових систем: у керівництві Bing «позитивний досвід користувача» визнаний більш значущим, ніж швидке завантаження сторінок, документація Yahoo! теж вказує на пріоритет зручності для користувачів щодо оцінки швидкості.

Largest Contentful Paint

Пошукові системи звертають увагу не на те, як швидко завантажиться вся сторінка, а на те, як швидко стане доступним перший рендер (тобто перший екран, з яким можна взаємодіяти). Для користувачів так само важливо, щоб перший видимий фрагмент якомога швидше став читабельним та інтерактивним. Largest Contentful Paint (LCP) — буквально «відмальовування найбільшого елемента контенту» — це зображення або текстовий блок, який займає найбільше місце на першому екрані.

Типи файлів, які можуть бути найбільшим елементом вмісту:

- Зображення (<img>)
- Зображення всередині svg-об'єктів (<image> всередині <svg>
- Відео (<video>)
- Фонові зображення завантажені за допомогою функції url()
- Блокові елементи із текстовими вузлами

Що впливає на завантаження LCP:

- Час відповіді сервера. Воно залежить від багатьох факторів - провайдера хостингу, системи управління контентом, задіяних баз даних та ін;

- Блокують рендеринг JavaScript та CSS. Саме файли JavaScript та CSS відповідають за інтерактивність та візуальну привабливість сторінки. При обробці HTML браузер зустрічає JS-and CSS-файли, які потрібно вивантажувати із сервера, - через це гальмується процес рендерингу.

- час завантаження ресурсів. На показник LCP впливає час завантаження ресурсів сторінки – коду, картинок, відео;

- тип рендерингу. Рендеринг JavaScript може здійснюватися на стороні клієнта та сервері. У першому випадку рендеринг відбувається безпосередньо у браузері, а у другому браузер отримує вже попередньо відрендерований контент сторінки з HTML-документа, який був запрошений у сервера.

2.5 секунди і менше – добрий показник LCP.

За допомогою веб сервісу pagespeed.web.dev дізнаємося продуктивність створеного додатку для ПК та телефонів(рис. 3.17, 3.18).

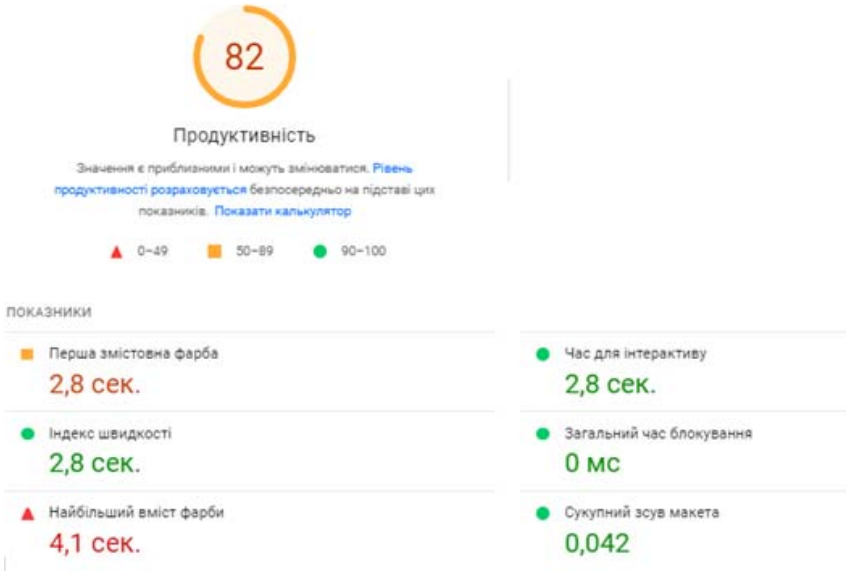

Рис. 3.17. Показники для мобільної версії

Як бачимо з тестів для мобільної версії результат задовільний, для відмінного результату потрібно більше оптимізувати картинки але це не є критичним для даного додатку.

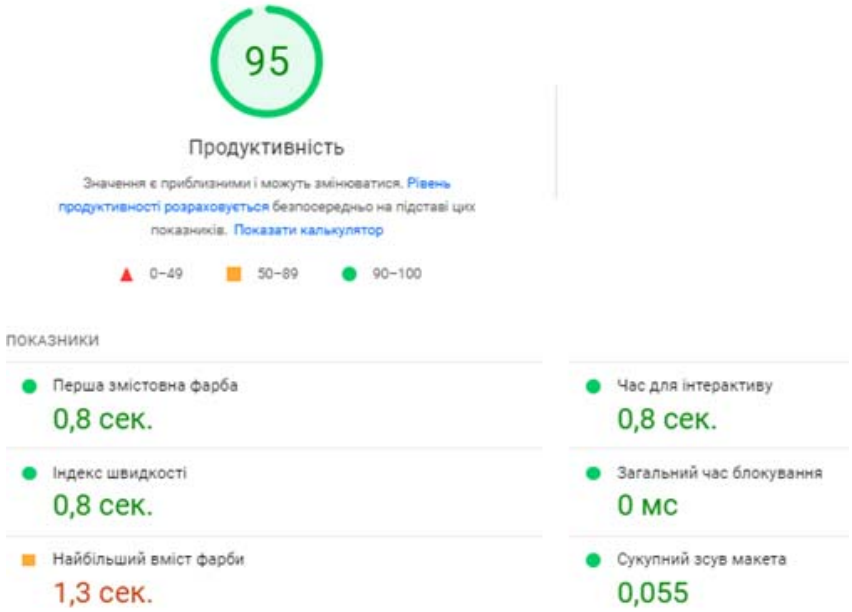

Рис. 3.18. Показники для ПК версії

Показники для ПК версії відмінні, це означає що недоліків з оптимізації та завантаження з БД налаштовано правильно та не потребує додаткових налагоджень і проект повністю готовий до завантаження на хостинг та повноцінної роботи.

Висновки за розділом 3

За допомогою React був зконструйований web-додаток. В якому було зверстано та розбито на компоненти проект. Всі дані про товар знаходяться в вже готовій базі даних, після чого вона переходить в Redux. Для стану кнопок та замовлень були використані хуки, що є правильним підходом що до написання коду.

Завдяки навігація можна зручно знайти потрібний товар та відсортувати за потрібним критерієм.

Всі показники швидкості додатку були заміряні та задовільні.

					151.2347СТ.КР.ПЗ.Р4		
Змн	Арк	№ докум.	Підпис	Дата			
Розроб.		Поянов І.І.			Охорона праці	Літ.	Арк	Аркушів
Керівник		Черно О.О.					49	
Зав.каф.		Черно О.О.				НУК ім. адмірала Макарова		
Н. контр.		Вінниченко І.Л.						

РОЗДІЛ 4

ОХОРОНА ПРАЦІ

4.1. Основні положення охорони праці

Охорона праці – це система правових, соціально-економічних, організаційно-технічних, санітарно-гігієнічних і лікувально-профілактичних заходів, спрямованих на збереження здоров’я і працездатності людини в процесі праці [12].

Закон України “Про охорону праці” від 14.10.1992 визначає основні положення щодо реалізації конституційного права громадян на охорону їх життя і здоров'я в процесі трудової діяльності, регулює за участю відповідних державних органів відносини між власником підприємства, установи і організації або уповноваженим ним органом і працівником з питань безпеки, гігієни праці та виробничого середовища і встановлює єдиний порядок організації охорони праці в Україні. Дія Закону поширюється на всі підприємства, установи і організації незалежно від форм власності та видів їх діяльності, на усіх громадян, які працюють, а також залучені до праці на цих підприємствах. Законодавство про охорону праці складається з Закону України “Про охорону праці”, Кодексу законів про працю України та інших нормативних актів. Державна політика в галузі охорони праці базується на принципах:

– пріоритету життя і здоров'я працівників відповідно до результатів виробничої діяльності підприємства, повної відповідальності власника за створення безпечних і нешкідливих умов праці;

– комплексного розв'язання завдань охорони праці на основі національних програм з цих питань та з урахуванням інших напрямів економічної і соціальної політики, досягнень в галузі науки і техніки та охорони навколишнього середовища;

– соціального захисту працівників, повного відшкодування шкоди особам, які потерпіли від нещасних випадків на виробництві і професійних захворювань;

– використання економічних методів управління охороною праці, проведення політики пільгового оподаткування, що сприяє створенню безпечних умов праці, участі держави у фінансуванні заходів щодо охорони праці;

– забезпечення координації діяльності державних органів, організацій та громадських об'єднань, що вирішують різні проблеми охорони здоров'я, гігієни та безпеки праці, а також співробітництва між власниками та працівниками, між усіма соціальними групами при прийнятті рішень з охорони праці на всіх рівнях.

4.2. Аналіз небезпечних і шкідливих факторів у приміщенні з компьютерною технікою

На людину в процесі її трудової діяльності можуть впливати небезпечні та шкідливі виробничі чинники. Небезпечні чинники – це такі, внаслідок яких з працюючою людиною може трапитись нещасний випадок і різке погіршення здоров’я. Шкідливі чинники – це такі, які можуть викликати професійне захворювання. Між шкідливими і небезпечними виробничими чинниками спостерігається взаємозв’язок. У багатьох випадках наявність шкідливих чинників сприяє проявленню травмонебезпечних чинників. Наприклад, надмірна вологість у приміщенні та наявність струмопровідного пилу (шкідливі чинники) підвищують небезпеку враження людини електричним струмом (небезпечний чинник).

Працівники, робота яких тісно пов’язана з використанням комп’ютерної техніки, зіштовхуються з впливом таких фізично небезпечних і шкідливих чинників, як підвищений рівень шуму, відсутність або недостатня освітленість робочої зони, електричний струм,

статична електрика тощо [13]. Велика кількість користувачів ПК на своєму робочому місці також піддаються впливу таких чинників, як розумова перенапруга, перенапруга зорових та слухових аналізаторів, монотонність праці. Вплив зазначених несприятливих чинників призводить до зниження працездатності. Тривале перебування в зоні комбінованого впливу різних несприятливих чинників може призвести до професійного захворювання.

Випромінювання монітору. Монітор ПК на основі електронно–променевої трубки виступає потенційним джерелом рентгенівського, ультрафіолетового, інфрачервоного, видимого, радіочастотного та низькочастотного електромагнітного випромінювань. Проте дослідження показали, що інтенсивність цих випромінювань набагато нижча рівнів, які здатні спричинити біологічний вплив на людину. Для різних моніторів величини рентгенівського випромінювання навіть на відстані 5 – 10 см від екрану не перевищують встановлених норм. Рівні ультрафіолетового та інфрачервоного випромінювань нижчі за природний фон або відповідають йому й не можуть чинити шкоди. На робочих місцях мають місце низькоінтенсивні електромагнітні випромінювання, які не перевищують припустимих значень електричних та магнітних полів для частот 60 кГц – 300 МГц. Максимальний рівень напруженості електричного поля реєструється біля задньої панелі монітору.

Виробниче освітлення. В умовах сучасного виробництва важливим чинником покращення умов праці є оптимізація кількісних та якісних характеристик освітлення робочих місць. В виробництві радіотехнічного та електронного профілю оптимізація зорової праці набуває особливого значення в зв’язку з інтенсифікацією праці та прогресуючою тенденцією до мініатюризації апаратури. Значна частина технологічних процесів пов’язана з роботою найвищої точності і характеризується високим ступенем напруженості зорової роботи. Втомлюваність органів зору залежить від ступеню напруженості процесів, супроводжуючих зорове сприйняття.

Вирішення питання раціонального освітлення виробничих приміщень і робочих місць покращує умови зорової роботи, послаблює зорове та нервове стомлення, сприяє підвищенню уваги й покращенню координаційної діяльності. Медичні дослідження показали, що хороше освітлення також покращує діяльність дихальних органів, сприяючи покращенню поглинання кисню.

Рівномірне розподілення яскравості в полі зору людини також має важливе значення для підтримки її працездатності. Якщо в полі зору постійно знаходяться поверхні які значно відрізняються за освітленістю, то при переведенні погляду з сильно– на слабкоосвітлену поверхню око повинне переадаптуватися. Часто переадаптація веде до розвитку втомлюваності зору й ускладнює виконання виробничих операцій.

Шум. Шум – це небажаний для людини звук. Рівень шуму вимірюється в децибелах (дБ). В залежності від рівня та характеру шуму, його тривалості, а також від індивідуальних особливостей людини, шум може по–різному впливати на неї. Шум у 20–30 дБ являє собою природний фон. Шум до 80 дБ сприймається як голосний. Шум у 130 дБ викликає болюче відчуття, а з 150 дБ стає нестерпним.

Шум створює значне навантаження на нервову систему людини, роблячи на неї психологічний вплив. Слабкий шум по–різному впливає на людей. Головними чинниками можуть бути: вік, стан здоров’я, вид праці. Вплив шуму залежить також від індивідуального відношення до нього.

Сильний шум викликає труднощі з розпізнанням колірних сигналів, знижує швидкість сприйняття кольору, гостроту зору, порушує сприйняття візуальної інформації. Довгий час людина скаржиться на нездужання. Симптоми – головний біль, нудота, надмірна дратівливість. Тривалий вплив шуму з рівнем звукового тиску 85–90 дБ сприяє збільшенню числа помилок у роботі, знижує продуктивність праці на 30–60 % і погіршує її якість. Шуми високих рівнів можуть явитися підґрунтям для розвитку стійкого безсоння, неврозів і склерозу.

При високих рівнях шуму слухова чутливість падає через 1–2 роки, а при середніх – через 5–10. Тому особливо важливо заздалегідь уживати заходів захисту від шуму. На разі майже кожен, хто піддається шуму, ризикує стати глухим.

Акустичні роздратування поволі накопичуються в організмі. Змінюється сила, урівноваженість і рухливість нервових процесів – тим більше, чим інтенсивнішим є шум. Люди, що піддаються постійному впливові шуму, часто стають важкими в спілкуванні.

Висновки за розділом 4

У розділі 4 розглянуті питання аналізу шкідливих та небезпечних виробничих чинників яким піддаються оператори ПК які задіяні при розробці або експлуатації АСУ(Т). Визначені шляхи вирішення цих проблем для покращення та створення належних умов праці персоналу. Виконаний розрахунок загального освітлення виробничого приміщення.

ВИСНОВКИ

У даній кваліфікаційній роботі було розроблено web-додаток для налагодження роботи піцерії.

У ході вивчення проблем закладу було прийнято рішення створити web-додаток який поліпшив би роботу піцерія. В ході вивчення конкурентів були виявлені ліпші технології для створення додатку який би не програвав конкурентам у функціоналі на швидкості

Було зверстано макет web-додатку піцерії в який входив: основна сторінка, корзина для замовлення, сторінка 404 на випадок неправильного переходу. у верстці використовувався збірник проектів gulp, верстка була зроблена за допомогою методології БЕМ, це дозволило швидко орієнтуватися у великому кількості коду та не зменшити кількість помилок на етапі збору дизайну.

За допомогою React додаток здобув логіку яка дозволила йому функціонувати як повноцінний сайт. Для людей з повільним інтернетом був розроблений скелетон для піц, це було досить важливо та не спостерігалось у конкурентів, що додасть додатку більшу аудиторію за рахунок кращої продуктивності. Найважливіший етап з додаванням товару в корзину, видалення та продовження замовлення був виконаний та налагоджений.

Додаток протестований на швидкість та показав досить гарні результати, що досить добре, тому що його за рахунок цього буде просувати пошук в інтернеті. Показник швидкості на телефоні показали 82 з 100 балів що є гарним результатом але є куди поліпшувати та вдосконалювати, а для ПК 95 з 100 балів що є відмінним результатом на момент створення додатку вдосконалення не потрібно, а проект можна почати експлуатувати.

Проведено аналіз небезпек і шкідливих для здоров'я людини факторів, характерних для робіт, пов'язаних з роботою за ПК, розроблено заходи щодо підвищення безпечності робіт.

Робота виконана на замовлення Fox Group та впроваджена в цій компанії.

ПЕРЕЛІК ПОСИЛАНЬ

1. Dinarys, «Вибір відповідних технологій для онлайн магазину» [Режим доступу]: https://dinarys.com/ru/blog/technology-stack-for-e-commerce-websites;
2. Webpack, «Webpack. Посібник» [Режим доступу]: https://webpack.js.org/;
3. React, «React. Бібліотека JavaScript» [Режим доступу]: https://ru.react.js.org/;
4. Redux Toolkit, «Керівництво по Redux Toolkit» [Режим доступу]: https://redux-toolkit.js.org/;
5. Gulp 4 «Актуальне та вичерпне керівництво для найменших» [Режим доступу]: https://webdesign-master.ru/blog/tools/gulp-4-lesson.html;
6. VSCode, «Code editing. Redefined» [Режим доступу]: https://code.visualstudio.com/;
7. HtmlAcademy «Що таке семантична верстка і навіщо вона потрібна» [Режим доступу]: https://htmlacademy.ru/blog/articles/semantics;
8. Методологія, «Керівництво з методології БЕМ» [Режим доступу]: https://ru.bem.info/methodology;
9. ІТШеф, «CSS медіа-запити (media queries)» [Режим доступу]: https://itchief.ru/html-and-css/media-queries;
10. MDN, «Що таке Об'єктна Модель Документу (DOM)» [Режим доступу]: https://developer.mozilla.org/ru/docs/Web/API/Document_Object_Model/Introduction;
11. React, «Введення в хуки» [Режим доступу]: https://ru.reactjs.org/docs/hooks-intro.html;
12. Zp.gov.ua, «Навчання з питань охорони праці» [Режим доступу]: https://zp.gov.ua/upload/editor/navchannya_z_pitan_ohoroni_praci.pdf;

13. StudFiles, «Шкідливі та небезпечні виробничі чинники при роботі з компютерною технікою» [Режим доступу]: https://studfile.net/preview/5211197/page:3/;
14. JavaScript: Повний посібник, 7-е видання - Девід Фленаган;
15. "Як працює JavaScript" ("How JavaScript Works") - Дуглас Крокфорд;
16. React та Redux. Функціональна веб-розробка (2018) [PDF] Алекс Бенкс, Єва Порселло.

ДОДАТОК А

Код програми web-додатку

Index.js

```
import { BrowserRouter } from 'react-router-dom';
import { Provider } from 'react-redux';
import App from './App';
import { store } from './redux/store';
const rootElem = document.getElementById('root');
if (rootElem) {
 const root = ReactDOM.createRoot(rootElem);
 root.render(
  <BrowserRouter>
   <Provider store={store}>
    <App />
   </Provider>
  </BrowserRouter>,
 );
}
```

App.js

```
import Loadable from 'react-loadable';
import React, { Suspense } from 'react';
import { Routes, Route } from 'react-router-dom';
import Home from './pages/Home';
import './scss/app.scss';
import MainLayout from './layouts/MainLayout';
const Cart = Loadable({
 loader: () => import(/* webpackChunkName: "Cart" */ './pages/Cart'),
 loading: () => <div> Йде завантаження кошика...</div>,
});
const FullPizza = React.lazy(() => import(/* webpackChunkName: "FullPizza" */ './pages/FullPizza'));
const NotFound = React.lazy(() => import(/* webpackChunkName: "NotFound" */ './pages/NotFound'));
function App() {
 return (
  <Routes>
   <Route path="/" element={<MainLayout />}>
    <Route path="" element={<Home />} />
    <Route
     path="cart"
     element={
      <Suspense fallback={<div>Йде завантаження кошика...</div>}>
       <Cart />
      </Suspense>
     }
    />
    <Route
     path="pizza/:id"
     element={
      <Suspense fallback={<div>Йде завантаження...</div>}>
       <FullPizza />
      </Suspense>
     }
    />
    <Route
     path="*"
     element={
      <Suspense fallback={<div>Йде завантаження...</div>}>
       <NotFound />
      </Suspense>
     }
```

```
      />
    </Route>
   </Routes>
  );
}
export default App;
```

cart.jsx

```
import React from 'react';
import { Link } from 'react-router-dom';
import { useDispatch, useSelector } from 'react-redux';
import { CartItem, CartEmpty } from '../components';
import { selectCart } from '../redux/cart/selectors';
import { clearItems } from '../redux/cart/slice';
const Cart: React.FC = () => {
 const dispatch = useDispatch();
 const { totalPrice, items } = useSelector(selectCart);
 const totalCount = items.reduce((sum: number, item: any) => sum + item.count, 0);
 const onClickClear = () => {
  if (window.confirm(Очистити кошик?')) {
   dispatch(clearItems());
  }
 };
 if (!totalPrice) {
  return <CartEmpty />;
 }
 return (
  <div className="container container--cart">
   <div className="cart">
    <div className="cart__top">
     <h2 className="content__title">
      <svg
       width="18"
       height="18"
       viewBox="0 0 18 18"
       fill="none"
       xmlns="http://www.w3.org/2000/svg">
       <path
        d="M6.33333 16.3333C7.06971 16.3333 7.66667 15.7364 7.66667 15C7.66667 14.2636 7.06971 13.6667 6.33333 13.6667C5.59695 13.6667 5 14.2636 5 15C5 15.7364 5.59695 16.3333 6.33333 16.3333Z"
        stroke="white"
        strokeWidth="1.8"
        strokeLinecap="round"
        strokeLinejoin="round"></path>

 <path d="M14.3333 16.3333C15.0697 16.3333 15.6667 15.7364 15.6667 15C15.6667 14.2636 15.0697 13.6667 14.3333 13.6667C13.597 13.6667 13 14.2636 13 15C13 15.7364 13.597 16.3333 14.3333 16.3333Z"
        stroke="white"
        strokeWidth="1.8"
        strokeLinecap="round"
        strokeLinejoin="round"></path>
       <path
        d="M4.78002 4.99999H16.3334L15.2134 10.5933C15.1524 10.9003 14.9854 11.176 14.7417 11.3722C14.4979 11.5684 14.1929 11.6727 13.88 11.6667H6.83335C6.50781 11.6694 6.1925 11.553 5.94689 11.3393C5.70128 11.1256 5.54233 10.8295 5.50002 10.5067L4.48669 2.82666C4.44466 2.50615 4.28764 2.21182 4.04482 1.99844C3.80201 1.78505 3.48994 1.66715 3.16669 1.66666H1.66669"
        stroke="white"
        strokeWidth="1.8"
        strokeLinecap="round"
```

```
            strokeLinejoin="round"></path>
          </svg>
          Корзина
        </h2>
        <div onClick={onClickClear} className="cart__clear">
          <svg
            width="20"
            height="20"
            viewBox="0 0 20 20"
            fill="none"
            xmlns="http://www.w3.org/2000/svg">
            <path
              d="M2.5 5H4.16667H17.5"
              stroke="#B6B6B6"
              strokeWidth="1.2"
              strokeLinecap="round"
              strokeLinejoin="round"></path>
            <path
              d="M6.66663 5.00001V3.33334C6.66663 2.89131 6.84222 2.46739 7.15478 2.15483C7.46734
1.84227 7.89127 1.66667 8.33329 1.66667H11.6666C12.1087 1.66667 12.5326 1.84227 12.8451
2.15483C13.1577 2.46739 13.3333 2.89131 13.3333 3.33334V5.00001M15.8333
5.00001V16.6667C15.8333 17.1087 15.6577 17.5326 15.3451 17.8452C15.0326 18.1577 14.6087
18.3333 14.1666 18.3333H5.83329C5.39127 18.3333 4.96734 18.1577 4.65478 17.8452C4.34222
17.5326 4.16663 17.1087 4.16663 16.6667V5.00001H15.8333Z"
              stroke="#B6B6B6"
              strokeWidth="1.2"
              strokeLinecap="round"
              strokeLinejoin="round"></path>
            <path
              d="M8.33337 9.16667V14.1667"
              stroke="#B6B6B6"
              strokeWidth="1.2"
              strokeLinecap="round"
              strokeLinejoin="round"></path>
            <path
              d="M11.6666 9.16667V14.1667"
              stroke="#B6B6B6"
              strokeWidth="1.2"
              strokeLinecap="round"
              strokeLinejoin="round"></path>
          </svg>
          <span>Очистити кошик</span>
        </div>
      </div>
      <div className="content__items">
        {items.map((item: any) => (
          <CartItem key={item.id} {...item} />
        ))}
      </div>
      <div className="cart__bottom">
        <div className="cart__bottom-details">
          <span>
            {' '}
            Всього піц: <b>{totalCount} шт.</b>{' '}
          </span>
          <span>
            {' '}
            Сума замовлення: <b>{totalPrice} </b>{' '}
          </span>
        </div>
        <div className="cart__bottom-buttons">
          <Link to="/" className="button button--outline button--add go-back-btn">
```

```
            <svg
              width="8"
              height="14"
              viewBox="0 0 8 14"
              fill="none"
              xmlns="http://www.w3.org/2000/svg">
              <path
                d="M7 13L1 6.93015L6.86175 1"
                stroke="#D3D3D3"
                strokeWidth="1.5"
                strokeLinecap="round"
                strokeLinejoin="round"></path>
            </svg>
            <span>Повернутись назад</span>
          </Link>
          <div className="button pay-btn">
            <span>Сплатити зараз</span>
          </div>
        </div>
      </div>
    </div>
  </div>
 );
};
export default Cart;
```

Home.jsx

```
import React from 'react';
import qs from 'qs';
import { useSelector } from 'react-redux';
import { useNavigate } from 'react-router-dom';
import { Categories, Sort, PizzaBlock, Skeleton, Pagination } from '../components';
import { sortList } from '../components/Sort';
import { useAppDispatch } from '../redux/store';
import { selectFilter } from '../redux/filter/selectors';
import { selectPizzaData } from '../redux/pizza/selectors';
import { setCategoryId, setCurrentPage, setFilters } from '../redux/filter/slice';
import { fetchPizzas } from '../redux/pizza/asyncActions';
import { SearchPizzaParams } from '../redux/pizza/types';
const Home: React.FC = () => {
  const navigate = useNavigate();
  const dispatch = useAppDispatch();
  const isMounted = React.useRef(false);
  const { items, status } = useSelector(selectPizzaData);
  const { categoryId, sort, currentPage, searchValue } = useSelector(selectFilter);
  const onChangeCategory = React.useCallback((idx: number) => {
    dispatch(setCategoryId(idx));
  }, []);
  const onChangePage = (page: number) => {
    dispatch(setCurrentPage(page));
  };
  const getPizzas = async () => {
    const sortBy = sort.sortProperty.replace('-', '');
    const order = sort.sortProperty.includes('-') ? 'asc' : 'desc';
    const category = categoryId > 0 ? String(categoryId) : '';
    const search = searchValue;
    dispatch(
      fetchPizzas({
        sortBy,
        order,
        category,
        search,
```

```
      currentPage: String(currentPage),
    }),
   );
   window.scrollTo(0, 0);
  };
 const pizzas = items.map((obj: any) => <PizzaBlock key={obj.id} {...obj} />);
  const skeletons = [...new Array(6)].map((_, index) => <Skeleton key={index} />);
  return (
   <div className="container">
    <div className="content__top">
     <Categories value={categoryId} onChangeCategory={onChangeCategory} />
     <Sort value={sort} />
    </div>
    <h2 className="content__title">Всі піци</h2>
    {status === 'error' ? (
     <div className="content__error-info">
      <h2>Виникла помилка □</h2>
      <p> На жаль, не вдалося отримати пітси. Спробуйте повторити спробу пізніше.</p>
     </div>
    ) : (
     <div className="content__items">{status === 'loading' ? skeletons : pizzas}</div>
    )}
    <Pagination currentPage={currentPage} onChangePage={onChangePage} />
   </div>
  );
};
export default Home;
```

FullPizza.jsx

```
import React from 'react';
import { Link } from 'react-router-dom';
import axios from 'axios';
import { useParams, useNavigate } from 'react-router-dom';
const FullPizza: React.FC = () => {
  const [pizza, setPizza] = React.useState<{
    imageUrl: string;
    title: string;
    price: number;
  }>();
  const { id } = useParams();
  const navigate = useNavigate();
  React.useEffect(() => {
    async function fetchPizza() {
      try {
        const { data } = await axios.get('https://626d16545267c14d5677d9c2.mockapi.io/items/' + id);
        setPizza(data);
      } catch (error) {
        alert('Помилка при отриманні піци!');
        navigate('/');
      }
    }
    fetchPizza();
  }, []);
  if (!pizza) {
    return <>Завантаження...</>;
  }
  return (
    <div className="container">
      <img src={pizza.imageUrl} />
      <h2>{pizza.title}</h2>
      <h4>{pizza.price} </h4>
      <Link to="/">
        <button className="button button--outline button--add">
```

```
        <span>Назад</span>
       </button>
      </Link>
    </div>
  );
};
export default FullPizza;
```

PizzaBlock.jsx

```
import React from 'react';
import { Link } from 'react-router-dom';
import { useDispatch, useSelector } from 'react-redux';
import { selectCartItemById } from '../../redux/cart/selectors';
import { CartItem } from '../../redux/cart/types';
import { addItem } from '../../redux/cart/slice';
const typeNames = ['тонке', 'традиційне'];
type PizzaBlockProps = {
  id: string;
  title: string;
  price: number;
  imageUrl: string;
  sizes: number[];
  types: number[];
  rating: number;
};
export const PizzaBlock: React.FC<PizzaBlockProps> = ({
  id,
  title,
  price,
  imageUrl,
  sizes,
  types,
}) => {
  const dispatch = useDispatch();
  const cartItem = useSelector(selectCartItemById(id));
  const [activeType, setActiveType] = React.useState(0);
  const [activeSize, setActiveSize] = React.useState(0);
  const addedCount = cartItem ? cartItem.count : 0;
  const onClickAdd = () => {
    const item: CartItem = {
      id,
      title,
      price,
      imageUrl,
      type: typeNames[activeType],
      size: sizes[activeSize],
      count: 0,
    };
    dispatch(addItem(item));
  };
  return (
    <div className="pizza-block-wrapper">
      <div className="pizza-block">
        <Link key={id} to={`/pizza/${id}`}>
          <img className="pizza-block__image" src={imageUrl} alt="Pizza" />
          <h4 className="pizza-block__title">{title}</h4>
        </Link>
        <div className="pizza-block__selector">
          <ul>
            {types.map((typeId) => (
              <li
                key={typeId}
                onClick={() => setActiveType(typeId)}
```

```
            className={activeType === typeId ? 'active' : ''}>
            {typeNames[typeId]}
           </li>
          ))}
         </ul>
         <ul>
          {sizes.map((size, i) => (
           <li
            key={size}
            onClick={() => setActiveSize(i)}
            className={activeSize === i ? 'active' : ''}>
            {size} см.
           </li>
          ))}
         </ul>
        </div>
        <div className="pizza-block__bottom">
         <div className="pizza-block__price">від {price} </div>
         <button onClick={onClickAdd} className="button button--outline button--add">
          <svg
           width="12"
           height="12"
           viewBox="0 0 12 12"
           fill="none"
           xmlns="http://www.w3.org/2000/svg">
           <path
            d="M10.8 4.8H7.2V1.2C7.2 0.5373 6.6627 0 6 0C5.3373 0 4.8 0.5373 4.8 1.2V4.8H1.2C0.5373 4.8
0 5.3373 0 6C0 6.6627 0.5373 7.2 1.2 7.2H4.8V10.8C4.8 11.4627 5.3373 12 6 12C6.6627 12 7.2 11.4627
7.2 10.8V7.2H10.8C11.4627 7.2 12 6.6627 12 6C12 5.3373 11.4627 4.8 10.8 4.8Z"
            fill="white"
           />
          </svg>
          <span>Додати</span>
          {addedCount > 0 && <i>{addedCount}</i>}
         </button>
        </div>
       </div>
      </div>
     );
};
```

Caregoty.jsx

```
import React from 'react';
type CategoriesProps = {
 value: number;
 onChangeCategory: (idx: number) => void;
};
const categories = ['Всі', 'Мясні', 'Вегетеріанські', 'Гриль', 'Гострі', 'Закриті'];
export const Categories: React.FC<CategoriesProps> = React.memo(({ value, onChangeCategory }) => {
 return (
  <div className="categories">
   <ul>
    {categories.map((categoryName, i) => (
     <li key={i} onClick={() => onChangeCategory(i)} className={value === i ? 'active' : ''}>
      {categoryName}
     </li>
    ))}
   </ul>
  </div>
 );
});
```

Sort.jsx

```
import React from 'react';
import { useDispatch } from 'react-redux';
import { setSort } from '../redux/filter/slice';
import { Sort as SortType, SortPropertyEnum } from '../redux/filter/types';
type SortItem = {
  name: string;
  sortProperty: SortPropertyEnum;
};
type PopupClick = MouseEvent & {
  path: Node[];
};
type SortPopupProps = {
  value: SortType;
};
export const sortList: SortItem[] = [
  { name: 'рейтингом (DESC)', sortProperty: SortPropertyEnum.RATING_DESC },
  { name: 'рейтингом (ASC)', sortProperty: SortPropertyEnum.RATING_ASC },
  { name: 'ціні (DESC)', sortProperty: SortPropertyEnum.PRICE_DESC },
  { name: 'ціні (ASC)', sortProperty: SortPropertyEnum.PRICE_ASC },
  { name: 'алфавиту (DESC)', sortProperty: SortPropertyEnum.TITLE_DESC },
  { name: 'алфавиту (ASC)', sortProperty: SortPropertyEnum.TITLE_ASC },
];
export const Sort: React.FC<SortPopupProps> = React.memo(({ value }) => {
  const dispatch = useDispatch();
  const sortRef = React.useRef<HTMLDivElement>(null);
  const [open, setOpen] = React.useState(false);
  const onClickListItem = (obj: SortItem) => {
    dispatch(setSort(obj));
    setOpen(false);
  };
  React.useEffect(() => {
    const handleClickOutside = (event: MouseEvent) => {
      const _event = event as PopupClick;
      if (sortRef.current && !_event.path.includes(sortRef.current)) {
        setOpen(false);
      }
    };
    document.body.addEventListener('click', handleClickOutside);
    return () => document.body.removeEventListener('click', handleClickOutside);
  }, []);
  return (
    <div ref={sortRef} className="sort">
      <div className="sort__label">
        <svg
          width="10"
          height="6"
          viewBox="0 0 10 6"
          fill="none"
          xmlns="http://www.w3.org/2000/svg">
  <path d="M10 5C10 5.16927 9.93815 5.31576 9.81445 5.43945C9.69075 5.56315 9.54427 5.625 9.375 5.625H0.625C0.455729 5.625 0.309245 5.56315 0.185547 5.43945C0.061849 5.31576 0 5.16927 0 5C0 4.83073 0.061849 4.68424 0.185547 4.56055L4.56055 0.185547C4.68424 0.061849 4.83073 0 5 0C5.16927 0 5.31576 0.061849 5.43945 0.185547L9.81445 4.56055C9.93815 4.68424 10 4.83073 10 5Z"
          fill="#2C2C2C"
        />
        </svg>
        <b>Сортировка по:</b>
        <span onClick={() => setOpen(!open)}>{value.name}</span>
      </div>
      {open && (
```

```
      <div className="sort__popup">
        <ul>
          {sortList.map((obj, i) => (
            <li
              key={i}
              onClick={() => onClickListItem(obj)}
              className={value.sortProperty === obj.sortProperty ? 'active' : ''}>
              {obj.name}
            </li>
          ))}
        </ul>
      </div>
    )}
  </div>
 );
});
```

Header.jsx

```
import React from 'react';
import { Link, useLocation } from 'react-router-dom';
import { useSelector } from 'react-redux';
import logoSvg from '../assets/img/pizza-logo.svg';
import { Search } from './';
import { selectCart } from '../redux/cart/selectors';
export const Header: React.FC = () => {
 const { items, totalPrice } = useSelector(selectCart);
 const location = useLocation();
 const isMounted = React.useRef(false);
 const totalCount = items.reduce((sum: number, item: any) => sum + item.count, 0);
 React.useEffect(() => {
  if (isMounted.current) {
   const json = JSON.stringify(items);
   localStorage.setItem('cart', json);
  }
  isMounted.current = true;
 }, [items]);
 return (
  <div className="header">
   <div className="container">
    <Link to="/">
     <div className="header__logo">
      <img width="38" src={logoSvg} alt="Pizza logo" />
      <div>
       <h1>React Pizza V2</h1>
       <p>найсмачніша піца у всесвіті</p>
      </div>
     </div>
</Link>
    {location.pathname !== '/cart' && <Search />}
    <div className="header__cart">
     {location.pathname !== '/cart' && (
      <Link to="/cart" className="button button--cart">
       <span>{totalPrice} ₴</span>
       <div className="button__delimiter"></div>
       <svg
        width="18"
        height="18"
        viewBox="0 0 18 18"
        fill="none"
        xmlns="http://www.w3.org/2000/svg">
        <path
```

```
            d="M6.33333 16.3333C7.06971 16.3333 7.66667 15.7364 7.66667 15C7.66667 14.2636
7.06971 13.6667 6.33333 13.6667C5.59695 13.6667 5 14.2636 5 15C5 15.7364 5.59695 16.3333 6.33333
16.3333Z"
            stroke="white"
            strokeWidth="1.8"
            strokeLinecap="round"
            strokeLinejoin="round"
           />
           <path
            d="M14.3333 16.3333C15.0697 16.3333 15.6667 15.7364 15.6667 15C15.6667 14.2636
15.0697 13.6667 14.3333 13.6667C13.597 13.6667 13 14.2636 13 15C13 15.7364 13.597 16.3333
14.3333 16.3333Z"
            stroke="white"
            strokeWidth="1.8"
            strokeLinecap="round"
            strokeLinejoin="round"
           />
           <path
            d="M4.78002 4.99999H16.3334L15.2134 10.5933C15.1524 10.9003 14.9854 11.176 14.7417
11.3722C14.4979 11.5684 14.1929 11.6727 13.88 11.6667H6.83335C6.50781 11.6694 6.1925 11.553
5.94689 11.3393C5.70128 11.1256 5.54233 10.8295 5.50002 10.5067L4.48669 2.82666C4.44466
2.50615 4.28764 2.21182 4.04482 1.99844C3.80201 1.78505 3.48994 1.66715 3.16669
1.66666H1.66669"
            stroke="white"
            strokeWidth="1.8"
            strokeLinecap="round"
            strokeLinejoin="round"
           />
          </svg>
          <span>{totalCount}</span>
         </Link>
        )}
      </div>
     </div>
    </div>
  );
};
```

CartItem.jsx

```
import React from 'react';
import { useDispatch } from 'react-redux';
import { addItem, minusItem, removeItem } from '../redux/cart/slice';
import { CartItem as CartItemType } from '../redux/cart/types';
type CartItemProps = {
  id: string;
  title: string;
  type: string;
  size: number;
  price: number;
  count: number;
  imageUrl: string;};
export const CartItem: React.FC<CartItemProps> = ({
  id,
  title,
  type,
  size,
  price,
  count,
  imageUrl,
}) => {
  const dispatch = useDispatch();

  const onClickPlus = () => {
```

```
  dispatch(
    addItem({
      id,
    } as CartItemType),
  );
 };
 const onClickMinus = () => {
  dispatch(minusItem(id));
 };
 const onClickRemove = () => {
  if (window.confirm(Ви дійсно хочете видалити товар?')) {
    dispatch(removeItem(id));
  }
 };
 return (
  <div className="cart__item">
    <div className="cart__item-img">
      <img className="pizza-block__image" src={imageUrl} alt="Pizza" />
    </div>
    <div className="cart__item-info">
      <h3>{title}</h3>
      <p>
        {type}, {size} см.
      </p>
    </div>
    <div className="cart__item-count">
      <button
        disabled={count === 1}
        onClick={onClickMinus}
        className="button button--outline button--circle cart__item-count-minus">
        <svg
          width="10"
          height="10"
          viewBox="0 0 10 10"
          fill="none"
          xmlns="http://www.w3.org/2000/svg">
          <path
            d="M5.92001 3.84V5.76V8.64C5.92001 9.17016 5.49017 9.6 4.96001 9.6C4.42985 9.6 4.00001 9.17016 4.00001 8.64L4 5.76L4.00001 3.84V0.96C4.00001 0.42984 4.42985 0 4.96001 0C5.49017 0 5.92001 0.42984 5.92001 0.96V3.84Z"
            fill="#EB5A1E"></path>
          <path
            d="M5.75998 5.92001L3.83998 5.92001L0.959977 5.92001C0.429817 5.92001 -2.29533e-05 5.49017 -2.29301e-05 4.96001C-2.2907e-05 4.42985 0.429817 4.00001 0.959977 4.00001L3.83998 4L5.75998 4.00001L8.63998 4.00001C9.17014 4.00001 9.59998 4.42985 9.59998 4.96001C9.59998 5.49017 9.17014 5.92001 8.63998 5.92001L5.75998 5.92001Z"
            fill="#EB5A1E"></path>
 </svg>
      </button>
      <b>{count}</b>
      <button
        onClick={onClickPlus}
        className="button button--outline button--circle cart__item-count-plus">
        <svg
          width="10"
          height="10"
          viewBox="0 0 10 10"
          fill="none"
          xmlns="http://www.w3.org/2000/svg">
          <path
```

```
        d="M5.92001 3.84V5.76V8.64C5.92001 9.17016 5.49017 9.6 4.96001 9.6C4.42985 9.6 4.00001
9.17016 4.00001 8.64L4 5.76L4.00001 3.84V0.96C4.00001 0.42984 4.42985 0 4.96001 0C5.49017 0
5.92001 0.42984 5.92001 0.96V3.84Z"
        fill="#EB5A1E"></path>
       <path
        d="M5.75998 5.92001L3.83998 5.92001L0.959977 5.92001C0.429817 5.92001 -2.29533e-05
5.49017 -2.29301e-05 4.96001C-2.2907e-05 4.42985 0.429817 4.00001 0.959977 4.00001L3.83998
4L5.75998 4.00001L8.63998 4.00001C9.17014 4.00001 9.59998 4.42985 9.59998 4.96001C9.59998
5.49017 9.17014 5.92001 8.63998 5.92001L5.75998 5.92001Z"
        fill="#EB5A1E"></path>
      </svg>
     </button>
    </div>
    <div className="cart__item-price">
     <b>{price * count} </b>
    </div>
    <div className="cart__item-remove">
     <div onClick={onClickRemove} className="button button--outline button--circle">
      <svg
       width="10"
       height="10"
       viewBox="0 0 10 10"
       fill="none"
       xmlns="http://www.w3.org/2000/svg">
       <path
        d="M5.92001 3.84V5.76V8.64C5.92001 9.17016 5.49017 9.6 4.96001 9.6C4.42985 9.6 4.00001
9.17016 4.00001 8.64L4 5.76L4.00001 3.84V0.96C4.00001 0.42984 4.42985 0 4.96001 0C5.49017 0
5.92001 0.42984 5.92001 0.96V3.84Z"
        fill="#EB5A1E"></path>
       <path
        d="M5.75998 5.92001L3.83998 5.92001L0.959977 5.92001C0.429817 5.92001 -2.29533e-05
5.49017 -2.29301e-05 4.96001C-2.2907e-05 4.42985 0.429817 4.00001 0.959977 4.00001L3.83998
4L5.75998 4.00001L8.63998 4.00001C9.17014 4.00001 9.59998 4.42985 9.59998 4.96001C9.59998
5.49017 9.17014 5.92001 8.63998 5.92001L5.75998 5.92001Z"
        fill="#EB5A1E"></path>
      </svg>
     </div>
    </div>
   </div>
  );
};
```

Pagination.jsx

```
import React from 'react';
import ReactPaginate from 'react-paginate';
import styles from './Pagination.module.scss';
type PaginationProps = {
 currentPage: number;
 onChangePage: (page: number) => void;
};
export const Pagination: React.FC<PaginationProps> = ({ currentPage, onChangePage }) => (
 <ReactPaginate
  className={styles.root}
  breakLabel="..."
  nextLabel=">"
  previousLabel="<"
  onPageChange={(event) => onChangePage(event.selected + 1)}
  pageRangeDisplayed={4}
  pageCount={3}
  forcePage={currentPage - 1}
 />
);
```