КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ТОРГОВЕЛЬНО-ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук та інформаційних систем

ВИПУСКНИЙ КВАЛІФІКАЦІЙНИЙ ПРОЕКТ

на тему:

«Розробка інформаційної системи управління маркетинговою діяльністю»

Студента 4 курсу, 9 групи,
спеціальності
122 «Комп'ютерні науки» _____ Лісовця Вадима Івановича
(підпис студента)

Науковий керівник:
доктор технічних наук, професор _____ Краскевич Валерій Євгенович
(підпис керівника)

Гарант освітньої програми:
кандидат технічних наук, доцент _____ Демідов Павло Георгійович
(підпис керівника)

Київ 2020

Київський національний торговельно-економічний університет

Факультет інформаційних технологій

Кафедра комп'ютерних наук та інформаційних систем

Спеціальність 122 «Комп'ютерні науки»

Затверджую

Зав. кафедри _____ Пурський О.І.

«20» грудня 2019 р.

ЗАВДАННЯ

на випускний кваліфікаційний проект студенту

Лісовцю Вадиму Івановичу

(прізвище, ім'я, по батькові)

1. Тема випускного кваліфікаційного проекту (далі ВКП):

«Розробка інформаційної системи управління маркетинговою діяльністю»

Затверджена наказом ректора від «04» грудня 2019 р. № 4111

2. Строк здачі студентом закінченого проекту «12» червня 2020 року

3. Цільова установка та вихідні дані до роботи

Мета роботи: дослідження сучасних методів ведення маркетингової діяльності.

Об'єкт дослідження: інтернет-магазин.

Предмет дослідження: технології, засоби та методи розробки web-сайтів.

4. Перелік графічного матеріалу_____

_____

_____

5. Консультанти по роботі із зазначенням розділів, за якими здійснюється консультування:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Пурський О.І.	15.12.2019 р.	15.12.2019 р.
2	Пурський О.І.	15.12.2019 р.	15.12.2019 р.
3	Пурський О.І.	15.12.2019 р.	15.12.2019 р.

6. Зміст випускного кваліфікаційного проекту (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1. РОЛЬ ІНТЕРНЕТ-ТЕХНОЛОГІЙ У МАРКЕТИНГОВІЙ ДІЯЛЬНОСТІ

1.1. Основи методології маркетингу в мережі Інтернет

1.2. Значення web-сайту в системі маркетингу підприємства

1.3. Сучасні етапи створення web-сайту / web-додатку

РОЗДІЛ 2. ВИБІР ПРОГРАМНОГО СЕРЕДОВИЩА, СИСТЕМИ КОМПІЛЯЦІЇ ТА КОМПОНОВКИ

2.1. Вибір JavaScript-фреймворку

2.2. Вибір CSS-препроцессора

2.3. Вибір інтегрованого середовища розробки

2.4. Вибір менеджера пакетів

2.5. ES6+ JavaScript та Babel-транспайлер

2.6. Препроцесор JSX

2.6. Вибір інструменту управління станом даних в JavaScript-додатках

2.7. Вибір збирача проекту

РОЗДІЛ 3. РОЗРОБКА ІНТЕРНЕТ-МАГАЗИНУ

3.1. Інструмент швидкого старту Create React App

3.2. Структура проекту

3.3. React.js: підключення та особливості використання

3.4. Застосування Redux та механізм його роботи

3.5. Інтерфейс та функціональні можливості сайту

ВИСНОВКИ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ДОДАТКИ

7. Календарний план виконання проекту

№	Назва етапів ВКП	Строк виконання етапів проекту	
		За планом	Фактично
1	2	3	4
1	Вибір теми ВКП	01.10.2019	01.10.2019
2	Розробка та затвердження завдання на ВКП	15.12.2019	15.12.2019
3	Вступ	03.02.2020	
4	Розділ 1. Роль інтернет-технологій у маркетинговій діяльності	28.02.2020	
5	Розділ 2. Вибір програмного середовища, системи компіляції та компоновки	06.04.2020	
6	Розділ 3. Розробка інтернет-магазину	12.05.2020	
7	Висновки	15.05.2020	
8	Здача ВКП на кафедру науковому керівнику	20.05.2020	
9	Попередній захист ВКП	03.06.2020	
10	Виправлення зауважень, зовнішнє рецензування ВКП	09.06.2020	
11	Представлення готової зшитої ВКП на кафедру	12.06.2020	
12	Публічний захист ВКП	За розкладом роботи ЕК	

8. Дата видачі завдання « 15 » грудня 2019 р.

Керівник випускного кваліфікаційного проекту

Краскевич В.Є.
(прізвище, ініціали, підпис)

Гарант освітньої програми Демідов П.Г.
(прізвище, ініціали, підпис)

Завдання прийняв студент-дипломник Лісовець В.І.
(прізвище, ініціали, підпис)

9. Відгук керівника випускного кваліфікаційного проекту

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____

Керівник випускного кваліфікаційного проекту

_____ « 30 » травня 2020 р.
(підпис)

10. Висновок про випускний кваліфікаційний проект

Випускний кваліфікаційний проект студента _____Лісовця В.І.
(прізвище, ініціали)

може бути допущений до захисту в екзаменаційній комісії.

Гарант освітньої програми _____Демідов П.Г.
(підпис, прізвище, ініціали)

Завідувач кафедри _____Пурський О.І.
(підпис, прізвище, ініціали)

«_____»_____2020 р.

Анотація

У випускному кваліфікаційному проекті проведено аналіз методологій маркетингу в мережі Інтернет з метою визначення місця інформаційних систем у маркетинговій діяльності підприємства. Теоретично обґрунтовано необхідність використання web-сайту як потужного інструменту маркетингу та здійснено огляд сучасних етапів його створення. Сайт став ключовим фактором успіху у мережевій економіці, що дає змогу найбільш ефективно розробляти та впроваджувати програми маркетингу. Здійснено вибір програмного середовища та розробку інтернет-магазину як інформаційної системи управління маркетинговою діяльністю.

Ключові слова: інформаційна система, маркетинг, мережа Інтернет, web-сайт, інтернет-магазин.

Annotation

The analyze of the marketing methodologies in the Internet in order to determine the place of information systems in the marketing activity of the enterprise is done in the graduation qualification work. The necessity of using the web-site as a powerful marketing tool is theoretically substantiated and the modern stages of its creation are reviewed. The site has become a key success factor in the network economy, enabling the most effective development and implementation of marketing programs. The software environment is selected and the online store as an information system for marketing activity management is developed.

Keywords: information system, marketing, Internet, web-site, online store.

ЗМІСТ

ВСТУП

Останнім часом мережа Інтернет стала одним з найважливіших засобів маркетингу, який має значні переваги в порівнянні із традиційними маркетинговими каналами. У сучасному ринковому середовищі компанії конкурують між собою з метою залучення клієнта не лише через якість, ціну своєї продукції, довіру до компанії тощо. Сайт стає ключовим фактором успіху в мережевій економіці, який дає змогу розробляти та впроваджувати найбільш ефективно програми маркетингу [2, 3].

Дослідженню питань про роль інтернет-технологій у маркетинговій діяльності присвячені значна кількість праць як закордонних дослідників (Dave Chaffey, ***Fiona Ellis-Chadwick, Richard Mayer***, Matt Bailey, Mary Lou Roberts, ***Debra Zahay, Alan Charlesworth, Ja***gdish N. Sheth, ***Abdolreza Eshghi, Balaji C. Krishnan) так і вітчизняних економістів-науковців, зокрема: Антон Воронюк, Александр Полищук, Артем Нестеренко, Андрій Катаєв, Мария Клименченко, Володимир Проскура, Тетяна Товт та інших[2, 3***, 30-36]. В роботах зазначається, що різноманітні процеси здійснюють істотний вплив на діяльність кожного окремого підприємства та економіки в цілому. Одним із таких процесів є впровадження інформаційних технологій. Впливу цих процесів схильні як вертикальні, так і горизонтальні економічні структури. Інтернет став новим ринком, представляючи систему нових економічних відносин в принципово новому просторі. Використання інтернет-технологій охопло безліч областей: від формування загального інформаційного середовища всередині компанії до взаємодії із суб'єктами ринку за допомогою інтернет-мережі, а сайт компанії став центральним елементом комунікативної політики. Це і зумовило **актуальність** обраної теми дослідження, його мету та завдання.

Мета і завдання дослідження. Метою даного дослідження є розробка інформаційної системи управління маркетинговою діяльністю. Для досягнення поставленої мети необхідно було вирішити наступні **завдання:**

- провести аналіз методологій маркетингу в мережі Інтернет;

- визначити місце web-сайту в системі маркетингу підприємства;
- здійснити огляд сучасних етапів розробки web-сайту;
- здійснити вибір JavaScript-фреймворку, інтегрованого середовища розробки, CSS-препроцесора, менеджера пакетів, інструменту управління станом даних та збирача проекту;
- розробити інтернет-магазин.

Об'єкт дослідження – інформаційні системи у маркетинговій діяльності підприємства.

Предмет дослідження – інтернет-магазин на базі React / Redux.

Теоретичною основою дослідження є праці провідних вчених з проблем запровадження та використання сучасних методологій маркетингу в мережі Інтернет.

Для практичного вирішення поставлених задач використано такі **методи дослідження**:

- загальнонауковий аналітичний метод;
- системний підхід;
- методи теорії БД для формування інформаційно-логічної моделі предметної області та БД;
- методи алгоритмічного програмування.

Інтернет-магазин, що є результатом кваліфікаційної роботи, реалізує інформаційну систему управління маркетинговою діяльністю підприємства. Сайт виступає ключовим фактором успіху у мережевій економіці, що дає змогу найбільш ефективно розробляти та впроваджувати програми маркетингу. Саме тому розробка має **практичне значення**.

РОЗДІЛ 1.
РОЛЬ ІНТЕРНЕТ-ТЕХНОЛОГІЙ У МАРКЕТИНГОВІЙ ДІЯЛЬНОСТІ

1.1. Основи методології маркетингу в мережі Інтернет

Маркетинг – це практична діяльність (система управлінських функцій), за допомогою якої організовують і керують комплексом дій 4Р, пов’язаних з оцінкою купівельної спроможності споживачів, з її перетворенням в реальний попит на вироби і послуги і їх наближенням до покупців для отримання прибутку або будь-якої іншої мети.

Комплекс маркетингу (концепція 4Р) – це система, складові якої піддаються контролю і набір яких підприємство використовує з метою викликати потрібну реакцію з боку цільового ринку. Вважається, що дана модель включає всі необхідні параметри продукту, які може контролювати і розвивати маркетолог.

Мета концепції 4Р – розробити стратегію, яка дозволить підвищити прийнятну цінність товару, а також допоможе максимізувати довгостроковий прибуток компанії на ринку.

Комплекс маркетингу включає такі основні елементи (рис. 1.1):

- продукція (англ. product) – рівень якості товарів та послуг, набір їх властивостей, асортимент, номенклатура;
- ціна (англ. price) – ціна, знижки, націнки;
- просування (англ. promotion) – стимулювання збуту, реклама, прямий маркетинг, зв'язки з громадськістю;
- ринок (англ. place) – розташування, ємкість, потреби клієнта [1].

Рис. 1.1. Маркетингова теорія 4Р

Розвиток інформаційних технологій став основою для появи і бурхливого зростання електронного бізнесу і комерції. При цьому розвиток інформаційних і комп'ютерних технологій якісно змінив характер маркетингової діяльності. У зв'язку з цим виникло поняття інтернет-маркетингу.

Інтернет-маркетинг – це досвід використання усіх методів та засобів традиційного маркетингу у мережі Інтернет з метою продажу продукту товару споживачам та керування їх взаємовідносинами.

Інтернет-маркетинг є частиною електронної комерції. Його складовими можуть бути: прямий маркетинг, зв'язки з громадскістю, оптимізація під соціальні мережі, контекстна реклама, медійна реклама, маркетинг у соцмережах, брендинг, пошукова оптимізація, вірусний маркетинг і т.д. [2]. Інтернет-маркетинг є частиною будь-якої якісної маркетингової компанії. Його частина у секторі споживачів та на ринку «бізнес для бізнесу» постійно зростає. Про це свідчить постійне зростання чисельності інтернет-магазинів.

До переваг інтернет-маркетингу відносять:

- зручність для клієнтів (немає необхідності у фізичній присутності та надана можливість здіснювати покупку у будь-який час доби не звертаючи увагу на робочий графік магазину);
- можливість відслідкувати вподобання клієнта на базі минулих покупок (персоналізація пропозицій для клієнта);
- швидкі кількісні результати (надає інформацію про кліки, підписки, відвідування, покупки тощо);
- широке охоплення клієнтів (немає обмежень відносно фізичного місця);
- менше накладних витрат на бізнес (автономні магазини вимагають більше витрат).

Використання методів інтернет-маркетингу направлене на економію фінансів (на рекламі та на зарплаті працівників), а також на розширення своєї діяльності та вихід на національний чи міжнародний ринок. Компанії,

незалежно від розмірів, мають однакові шанси у просуванні своїх товарів. Вихід на ринок за допомогою мережі Інтернет вимагає набагато менших витрат та дає конкретну статистичну інформацію ефективності маркетингової кампанії, що відрізняє даний метод від традиційних методів просування. Такий спосіб цілеспрямовано набуває більшої популярності не лише у бізнесі, але і у простих користувачів, які мають на меті просунути особистий web-сайт і отримати від цього прибуток [3].

Мережа Інтернет надає можливість для збору та розподілу інформації і служить новим каналом продажу та просування товарів і послуг. Зрештою мережа об'єднує інформацію для управлінських дій на всіх рівнях компанії та забезпечує нові електронні зв'язки для того, щоб полегшити зв'язок із клієнтами та партнерами ланцюга постачання. Отже, інтернет-маркетинг безсумнівно лідирує в ефективності порівняно із класичним маркетингом.

1.2. Значення web-сайту в системі маркетингу підприємства

На даний момент Інтернет виступає не лише засобом отримання інформації, але й ефективним бізнес-інструментом. Для розширення каналу збуту компанія повинна не лише відкрити автономний магазин, але й для підвищення показників збуту власної продукції створити сайт. Отже, сайт – це потужний засіб збільшення прибутку в інформаційну епоху [4, 5].

Web-сайт знаходить ефективне застосування в таких областях як:

- продаж через мережу Інтернет;
- реклама продукції;
- сервіс та обслуговування після продажу.

Сайт надає працівникам маркетингових служб можливість:

- за невеликий проміжок часу змінити асортимент, ціну чи опис товару;
- зекономити на витратах (витрати на утримання персоналу, приміщення, поліграфічну рекламу і т.д.);
- швидко надсилати своїм клієнтам інформацію роз'яснювального чи рекламного характеру та отримувати зворотній зв'язок;

- вести статистику відвідувачів сайту чи його окремих частин.

Сайти, що використовують тільки технологію HTML, важко використовувати для віртуального маркетингу, тому що такі сайти статичні, і вони не здатні збирати або аналізувати інформацію. Найчастіше в Інтернеті використовуються динамічні сайти, які крім технології HTML, використовують такі технології як:

- web-мови програмування (PHP, Perl, ASP.NET), здатні динамічно обробляти інформацію, вести персоналізацію, взаємодіяти з електронними базами даних;
- електронні бази даних для зберігання зібраної інформації (MongoDB, MySQL, Oracle, MS SQL);
- модулі аналізу, обробки, статистики, прийняття рішень (SAP, OLAP, LDAP).

Сьогодні під назвою «інтернет-магазин» мається на увазі цілий спектр рішень різного масштабу і призначення. Такий магазин ще має назву віртуального.

Віртуальний магазин є представництвом підприємства у всесвітній павутині через створення сайту для продажу своєї продукції користувачам мережі Інтернет. Іншими словами, віртуальний магазин є співтовариством географічно розділених працівників магазину і клієнтів, що комунікують через Інтернет без необхідності особистого контакту.

Грамотно підготовлений web-сайт в змозі вирішувати питання залучення клієнтів і збільшення продажів, створення іміджу та вивчення споживчого попиту і т.д. Однак навіть незначні промахи при реалізації сайту можуть завдати істотної шкоди діяльності фірми.

Отже, використання сайту у системі маркетингу підприємства дозволяє забезпечити виконання наступних цілей та завдань:

- збільшення обсягу продажів за рахунок впровадження електронної комерції в режимі реального часу;

- використання мережевої реклами товарів і послуг з метою збільшення обсягу продажів традиційними способами;
- скорочення витрат на ведення бізнесу;
- створення позитивного сучасного іміджу;
- створення інформаційних баз даних;
- надання нових послуг;
- збір інформації про ринок, поповнення маркетингових баз даних.

Необхідно відзначити, що перенесення традиційної торгівлі у всесвітню павутину робить її гнучкішою, оскільки електронна торгівля, базуючись на цифровій інформації, полегшує співпрацю людей.

1.3. Сучасні етапи створення web-сайту / web-додатку

Для розробки і запуску сучасного та успішного сайту необхідно дотримуватись певних етапів (рис. 1.2), результатом виконання яких буде якісний і ефективний продукт з фірмовим стилем та зручним інтерфейсом.

Рис. 1.2. Етапи створення web-сайту

Отже, роботу зі створення сайту можна розділити на таких 12 етапів:

1. Цілі і завдання проекту. На перший погляд простий, але дуже важливий етап, суттю якого є пошук чітких та продуманих цілей, принципів і тонкощів роботи сайту. Визначення та обговорення таких речей вимагає занурення в задачу як від виконавця, так і від замовника. Результатом цього етапу є заповнений опитувальний лист (бриф).

2. Аналіз предметної області – це збір необхідної інформації про тематику сайту, особливості роботи аналогічних сайтів, визначення основних моментів та виключення несуттєвого. Даний етап забезпечує розробку правильного і виваженого технічного завдання на розробку.

3. Технічне завдання і тип сайту. Розробник і замовник обговорюють всі необхідні деталі та нюанси, генеруючи ТЗ на створення сайту. Щонайменше повинні бути узгоджені наступні аспекти:

- тип (особистий блог, форум, сайт новин, лендінг, сайт-візитка, інтернет-магазин, корпоративний портал і т.д.);
- функціонал сайту – набір можливостей та інтерфейсів (пошук, каталог, стрічка новин, календар подій, форум, платіжний шлюз тощо);
- дизайн – зовнішній вигляд інтерфейсу, основні вимоги до оформлення, бажаний фірмовий стиль і колірна гамма ;
- основна структура – зміст сторінок сайту та взаємозв’язок між ними;
- структура сторінок – перелік необхідних блоків на сторінках сайту, їх вигляд, відображення, додатковий функціонал тощо.

4. Попередній макет дизайну. Етап складається з декількох підетапів:

- розробка унікальної дизайнерської ідеї (ескізи дизайну в різних варіантах з текстовими поясненнями демонструються замовнику для прийняття рішення);
- ескіз головної сторінки (основа для дизайну всього сайту);
- узгодження із замовником (обговорення зауважень, удосконалення макету, затвердження);
- проектування внутрішніх сторінок (розробка інших сторінок сайту відповідно до аналогічного алгоритму).

5. Визначення хостингу і доменного імені. Домен – це по суті перше враження ще до того моменту як його побачить користувач. Він повинен бути простим і запам’ятовуваним. Хостинг – це сервер, на якому знаходитиметься сайт. Часто розробники пропонують перелік провайдерів, з якими вони працювали.

6. Установка і вибір рушія – вибір розробниками необхідного CMS для сайту, якщо у замовника не вказав іншого. Розробка може вестись на WordPress, Joomla, Drupal, OpenCart і т.д. Не існує єдиного універсального рушія, який підходив би для створення всіх типів сайтів.

7. Верстка сторінок. Коли фірмовий стиль затверджений, CMS сайту підібраний, можна приступити до розробки на HTML і CSS. Як правило, цей етап повинен відповідати певним вимогам:

- кросбраузерна сумісність – однаково коректне відображення сайту в будь-якому браузері, збереження всіх основних принципів його інтерфейсу і відсутність помилок, незалежно від браузера;
- адаптивність – властивість інтерфейсу сайту підлаштовуватися під розширення пристрою користувача;
- динамічний код – збереження можливості в будь-який момент швидко правити код, виправляти помилки і додавати нові елементи, інформацію;
- оптимізація – код сайту повинен виконуватися браузером швидко і ефективно на будь-яких пристроях, не витрачаючи зайвих ресурсів процесора та пам’яті;
- відповідність стандартам – дотримання загальноприйнятих міжнародних стандартів по форматуванню коду та дотримання норм верстки;
- коректність – правильне і логічне використання всіх елементів HTML / CSS, коректна робота JS-коду.

8. Розробка функціоналу та програмування. Front-end розробники «оживлюють» дизайн, перетворюючи макети в інтерактивні web-сторінки

9. Наповнення вмістом – завантаження оригінального контенту для ефективного SEO-просування. На цьому етапі ключову роль грає професійний копірайтинг. Всі матеріали сайту повинні бути SEO-оптимізовані, повинні підходити до загальної концепції сайту та відповідати цілям і завданням проекту.

10. Тестування, відлагодження, виправлення помилок. Тестування проводиться спільно з розробником і замовником, виявляються всі помилки, недоліки, упущення. Сайт поліпшується до максимально ефективного стану.

11. Публікація web-сайту в інтернеті. На даному етапі сайт повністю завершений, має унікальний дизайн і фірмовий стиль, всі необхідні функції, стає доступний в інтернеті і готовий до просування. Відбувається реєстрація сайту в найбільших пошукових системах і каталогах для початку просування.

12. Просування, розкрутка і реклама. Для розкрутки сайту можна використовувати банерну, контекстну і таргетингову рекламу, застосовуючи всі можливості SEO-просування і оптимізації, SMO тощо. Завдання полягає у професійному виводу сайту в верхні позиції по цільових запитах [6].

Отже, із розвитком ринкових відносин в мережі Інтернет та появою нових інформаційних технологій змінюються і можливості створення сайту та вимоги до нього. Це зумовлює необхідність пошуку нових підходів до створення сайту з урахуванням сучасних ринкових вимог та інноваційних можливостей інформаційних технологій.

РОЗДІЛ 2.
ВИБІР ПРОГРАМНОГО СЕРЕДОВИЩА, СИСТЕМИ КОМПІЛЯЦІЇ ТА КОМПОНОВКИ

2.1. Вибір JavaScript-фреймворку

Javascript-фреймворки є однією з найбажаніших платформ для створення сучасних односторінкових web-додатків (SPA), соціальних мереж, eCommerce-продуктів, SaaS-платформ і багато чого іншого. JavaScript є ядром для більшості веб-додатків і має в своїй екосистемі велику кількість різноманітних інструментів для вирішення web-задач. Загалом через доступність великого арсеналу інструментів багато розробників не змогли дати собі відповідь який JavaScript front-end-фреймворк найкраще підходить для їх наступного проекту [7].

Javascript-фреймворк – це набір методів і функцій, які написані на мові Javascript і готові до використання розробниками. Ці функції спрощують взаємодію веб-додатку із сервером і прискорюють маніпулювання елементами веб-сторінки чи додатку.

Основною перевагою використання фреймворків є той факт, що вони забезпечують миттєвий зворотний зв’язок з тими, хто в даний час взаємодіє із сайтом. На традиційних web-сайтах без фреймворків початковий контент зберігається на сервері, і будь-який новий контент, який необхідно завантажити, вимагає перезавантаження сторінки. А при використанні фреймворку перезавантажуються тільки потрібні блоки сайту. Такий підхід зараз використовують соціальні мережі, наприклад, Facebook.

Нижче згадані деякі з переваг використання фреймворків:

- Вартість розробки web-сайтів та web-додатків. Вона знижується завдяки JavaScript-фреймворкам, оскільки вони безкоштовні і мають відкритий вихідний код.

- Швидкість розробки. У JavaScript-фреймворків хороша документація (опис роботи функцій та приклади використання), безліч форумів і груп

підтримки. Деякі з Javascript-фреймворків підтримуються такими відомими компаніями як Google або Facebook. Завдяки такій кількості інформації швидкість розробки стрімко зростає.

- Ефективність. Використання попередньо створених функцій і шаблонів дозволяє реалізовувати проекти якісніше. Розробники в кінцевому підсумку пишуть менше коду, що призводить до швидчого та ефективнішого виконання проектів.

Останні кілька років перші рядки у рейтингу front-end-фреймворків займають Angular і React. У 2018-2019 до цього списку додався Vue і продовжує розвиватись. Тому буде робитись вибір лише серед цих трьох технологій.

Angular – повнокомпонентий фреймворк, який використовується величезними корпораціями. Розробники люблять його за якісну документацію і відсутність необхідності вивчення додаткових бібліотек.

Переваги:

- велика екосистема;
- повний набір інструментів;
- якісна генерація коду;
- елегантний стиль програмування.

Недоліки:

- використовує багато енергоресурсів;
- велика крива навчання;
- погана оптимізація продуктивністі.

Сьогодні позиції Angular значно знизились. Велика кількість розробників стали скаржитись на нього все частіше, а спільнота The State of Javascript опублікувала результати опитувань, згідно з якими Angular сьогодні вважають вмираючим фреймворком. Тим не менше навіть у 2020-му році ще є попит на фахівців, які знають цей продукт.

Vue випущений у 2014-му році, але отримав світове визнання зовсім недавно. Перш за все, він знайшов популярність завдяки простоті і дуже

низькому розміру файлу. Останні оновлення допомогли значно обійти React і Angular за багатьма показниками. Але об’єктивно цей фреймворк ще не досяг рівня двох попередніх.

Переваги:

- мала крива навчання;
- легке налаштування;
- проста інтеграція з елементами інших мов програмування.

Недоліки:

- обмежений набір інструментів;
- все ще невелика спільнота користувачів;
- швидко еволюціонує (перевага і недолік одночасно – через швидкий розвиток багато прикладів розробок можуть бути застарілими).

На даний момент більшість азіатських компаній (наприклад, Xiaomi і Alibaba) перейшли на цей фреймворк. Опитування показали, що:

- 90% тих, хто використовував Vue, з радістю працюватимуть з ним в майбутньому;
- 70% тих, хто чув про Vue, зацікавлені у його вивченні.

Тому можна впевнено сказати, що у 2020-му році Vue буде одним із ключових JavaScript-трендів.

React – це JavaScript-бібліотека з відкритим вихідним кодом для розробки користувацьких інтерфейсів. Розроблена і підтримується компанією Facebook.

Переваги:

- гнучкий;
- невеликі розміри файлів;
- прості оновлення, що не порушують стабільність роботи;
- відмінно комбінується з іншими бібліотеками при необхідності.

Недоліки:

- як правило, потрібні знання додаткових інструментів для повноцінного процесу написання коду;

- крива навчання сильно залежить від того, який другорядний набір рішень вибере користувач;
- не оптимізована документація.

На даний момент React – найпопулярніший набір рішень. Важливо, що нарікань в його сторону значно менше в порівнянні з Angular. Його використовують такі корпорації, як Airbnb і Twitter, тому Facebook підтримує його функціональність і стабільність на високому рівні.

Згідно із опитуванням, виконаним State of JS [8], у 2019 році React є лідером за рівнем задоволеності розробниками (рис. 2.1). State of JS щорічно подає статистику відносно усього, що стосується JavaScript. Це перше опитування, респондентами якого виступають виключно розробники.

Рис. 2.1. Статистика State of JS. Рівень задоволеності

Відповідно до трендів NPM React є найбільш скачуваним фреймворком у 2019-му році (рис. 2.2). За даними Stack Overflow у 2020-му році React тримає лідерські позиції, популярність Angular падає, а популярність Vue навпаки – зростає (рис. 2.3).

	stars	forks	issues	updated	created	size
react	132428	24544	763	Jul 11, 2019	May 24, 2013	2.7 KB
vue.js	143303	20730	316	Jul 10, 2019	Jul 29, 2013	22.8 KB
@angular/core	49511	13565	2946	Jul 11, 2019	Sep 18, 2014	87.4 KB

Рис. 2.2. Статистика State of JS. Рівень задоволеності

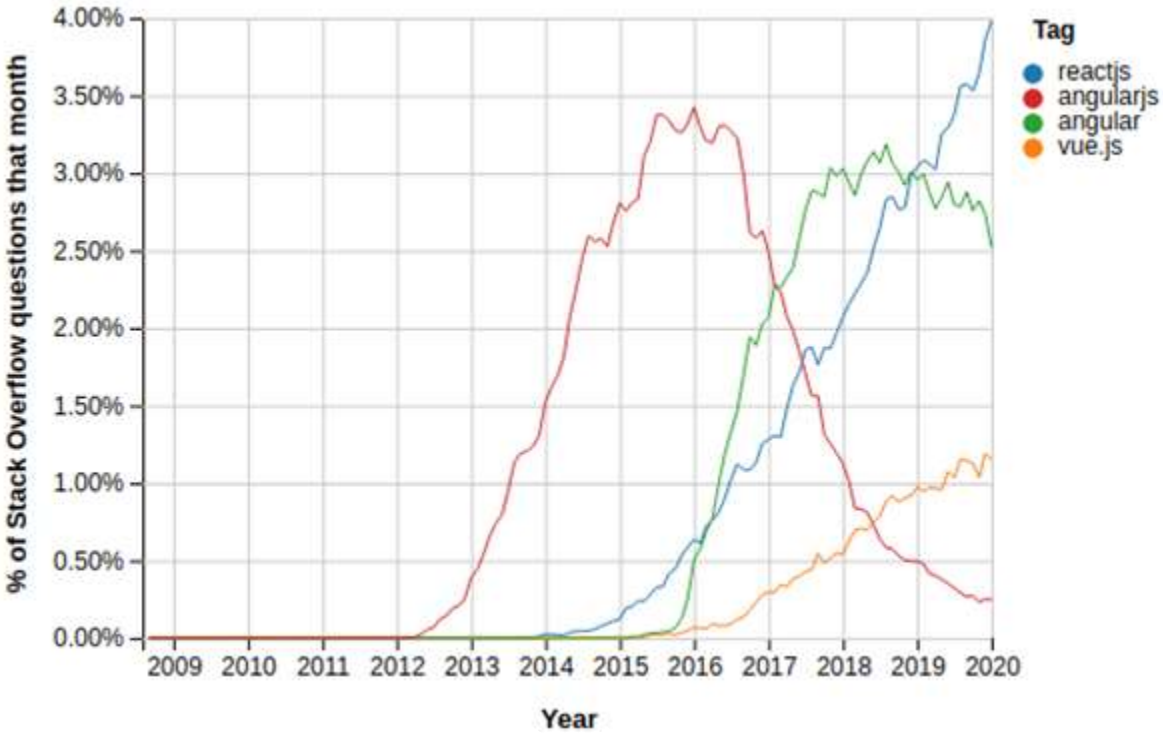

Рис. 2.3. Тренди Stack Overflow

GitHub показує, що React має більше всього fork-ів, а у Vue більше зірок (рис. 2.4).

	Github					
Angular	Watch	3,233	Star	52,246	Fork	14,458
React	Watch	6,607	Star	136,993	Fork	25,847
Vue	Watch	5,946	Star	149,009	Fork	21,984

Рис. 2.4. Тренди GitHub

Отже, враховуючи усі переваги і недоліки кожного із перерахованих вище фреймворків та досвід розробників із багаторічним стажем, прийнято рішення взяти за основу для розробки сайту бібліотеку React.

Однією з її характерних особливостей є можливість використовувати JSX (мова програмування з близьким до HTML синтаксисом), який компілюється в JavaScript. Також розробники можуть досягти підвищення продуктивності високонавантажених додатків за допомогою Virtual DOM, що допоможе знизити ймовірність виникнення можливих незручностей та покращує користувальницький досвід. Як відомо, операції з DOM-деревом потребують великих ресурсів, тому усі операції виконуються із віртуальним DOM, щоб в реальний DOM за один раз додати усі зміни.

React використовує ізоморфний підхід, що допомагає здійснювати рендеринг сторінок швидше, тим самим дозволяючи користувачам відчувати себе комфортніше під час роботи із додатком. Пошукові системи індексують такі сторінки краще. Створені компоненти можуть бути з легкістю змінені і використані заново в нових проектах. Високий відсоток перевикористання коду підвищує покриття тестами, що, в свою чергу, призводить до підвищення рівня контролю якості. Завдяки можливості перевикористання

коду стало набагато простіше створювати мобільні додатки. Код, який був написаний під час створення сайту, може бути знову використаний для створення мобільного додатку.

2.2. Вибір CSS-препроцесора

Препроцесор – це інструмент, що перетворює код з одного синтаксису в інший. Зазвичай на вході препроцессора поступає з код, написаний з використанням синтаксичних конструкцій, відомих цьому перепроцесору.

CSS-препроцесор – мова написання стилів, схожа на CSS, але з можливостями, яких у нього не вистачає; безпосередньо в браузері не працює і потребує компіляції, результатом якої є CSS-файл. Для компіляції потрібно запустити будь-яке ПЗ (gulp, webpack тощо). Воно стежить за збереженням препроцесорних файлів і збирає із них CSS-файл.

Актуальні CSS-препроцесори: Sass / SCSS (2007 р., лідер ринку), Less (2009 р.), Stylus (2010 р.) і PostCSS (2013 р., спочатку використовувався для постпроцесингу, але він надає можливість повторити майже всю функціональність справжніх препроцесорів). Далі наведено особливості роботи кожного з них.

Less (Leaner Style Sheets) – це динамічна мова стилів. Основним недоліком є відсутність умовних конструкцій і циклів. Головна перевага даного інструменту – ясність і простота у застосуванні, його функціональні можливості можна розширити за допомогою різних плагінів.

Sass (Syntactically Awesome Style Sheets) – це метамова на основі CSS, основним завданням якої є розширення можливостей написання CSS коду. Він був розроблений в якості модуля для HAML і, крім того, написаний на Ruby (є порт на C ++). Препроцесор має два варіанти синтаксису: Sass і SCSS (їх функціонал повністю однаковий, різниця полягає лише у синтаксисі).

Особливості Sass / SCSS:

- Тригонометрія – SCSS дозволяє писати власні (синусоїдальні і косинусоїдальні) функції.

- Змінні – в стандартному CSS теж є поняття змінних, але в Sass з ними можна працювати по-іншому, наприклад, повторювати їх через директиву @for або генерувати властивості динамічно.

- Вкладеність – SCSS дозволяє вкладати правила CSS один в одного.

- Покращені математичні операції – можна додавати, віднімати, множити та ділити значення CSS і на відміну від стандартного CSS, Sass / SCSS дозволяє обійтись без calc().

- Директиви @for, @while і вираз @ if-else.

- Міксини (домішки) – можна один раз створити набір CSS-властивостей і працювати з ними повторно або змішувати з іншими значеннями. Міксини можна використовувати для створення окремих тем одного макета. Домішки також можуть містити цілі CSS-правила чи будь-що інше, дозволене в Sass-документі. Вони навіть можуть приймати аргументи, що дозволяє створювати велику різноманітність стилів за допомогою невеликої кількості міксинів.

- Функції – можна створювати визначення CSS у вигляді функцій для багаторазового використання.

Stylus – наймолодший інструмент, що базується на Node.js, тому відпадає необхідність у використанні сторонньої технології (Sass вимагає для своєї роботи Ruby). Stylus надає набір JavaScript API, що робить можливим подальше налаштування цього інструменту. Його синтаксис не потребує дужок, ком, двокрапок, крапок із комою – він цілком і повністю заснований на використанні табуляції та пробілів; але якщо необхідно використовувати будь-який з видів пунктуаций, то його можна легко застосувати – компіляція відбудеться коректно. Для Stylus є готова бібліотека міксинів під назвою Nib. Остання стабільна збірка цього препроцесора була випущена у 2014 р., що свідчить про стрімке зменшення його популярності.

Звертаючись до статистики, варто звернути увагу, що згідно оцінки загальної популярності від Google (рис. 2.5) Sass значно лідирує. Статистика сформована на основі даних термінів, введених у пошуковику, і містить

думку усіх користувачів мережі Інтернет. GitHub показує (рис. 2.6), що Less лідирує за кількістю fork-ів та кількістю зірок, проте Sass має більшу кількість репозиторіїв. Це свідчить про те, що розробники надають однакову перевагу обом препроцесорам. Іншу картину можна побачити, якщо звернутись до опитування, проведеного Dagstuhl Research Online Publication Server, SitePoint та Ashley Nolan. Згідно з ним середовище розробників значну перевагу надає Sass.

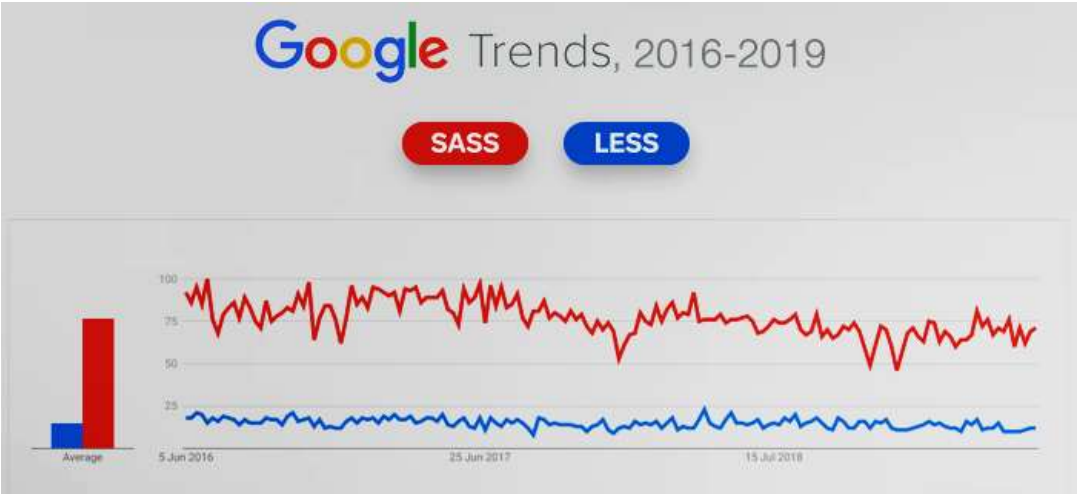

Рис. 2.5. Тренди Google

Рис. 2.6. Тренди GitHub

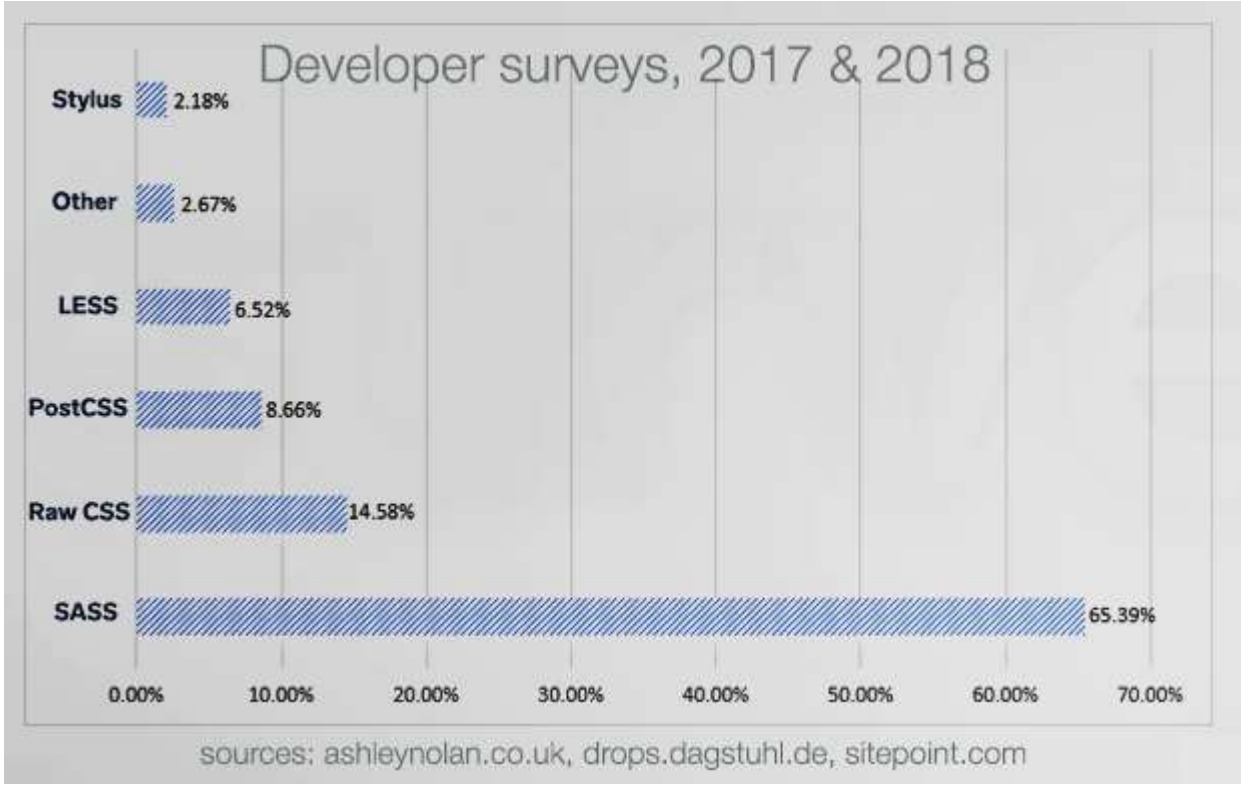

Рис. 2.7. Опитування розробників: Dagstuhl Research Online Publication Server, SitePoint та Ashley Nolan

Отже, проаналізувавши особливості лідируючих в останні роки препроцесорів та звернувшись до статистики, вирішено у своєму проекті використати препроцесор Sass із CSS-подібним синтаксисом – SCSS.

2.3. Вибір інтегрованого середовища розробки

JavaScript – потужна і примхлива мова програмування. З одного боку, безліч фреймворків і бібліотек, з іншого – не найпростіший синтаксис та небезпеки, пов’язані з «динамікою». Тому для роботи з JavaScript важливо підібрати редактор коду, оскільки в них програмісти проводять більшу частину свого робочого часу. Правильний вибір забезпечить чистоту коду, високу швидкість розробки, мінімум помилок і задоволення від роботи.

Є два основні типи редакторів: IDE та «легкі» редактори (далі – текстові редактори). Багато хто використовує по одному інструменту кожного типу.

Терміном IDE (Integrated Development Environment) називають потужні редактори з безліччю функцій, які працюють в рамках цілого проекту. IDE має можливість завантажити проект (який може складатись із безлічі файлів) та перемикатись між файлами, пропонує автодоповнення по коду всього проекту (не лише відкритого файлу), також він інтегрується із системою контролю версій (наприклад, git), середовищем для тестування та іншими інструментами на рівні всього проекту. До IDE належать Visual Studio Code та WebStorm.

Текстові редактори менш потужні по відношенню до IDE, але вони відрізняються швидкістю, зручним інтерфейсом та простою. В основному їх використовують для того, щоб швидко відкрити і відредагувати потрібний файл. Головна відмінність між текстовим редактором та IDE полягає у тому, що IDE працює на рівні усього проекту, тому вона завантажує більше даних при запуску, аналізує структуру проекту, якщо це необхідно, і т.д. Текстовий редактор підходить лише для одного файлу.

На практиці текстові редактори можуть мати багато плагінів, включаючи автодоповнення та аналізатори синтаксису на рівні директорії, тому межі між IDE та текстовими редакторами розмиваються. До текстових редакторів відносять: Atom, Sublime Text, Notepad++, Vim та Emacs.

Розробка сайту вимагає роботи з великою структурою файлів та вимагає інтеграції додаткових інструментів. Освітній портал GeekBrains пропонує до використання [10] 5 редакторів: WebStorm, Visual Studio Code, Sublime Text, Atom Editor, Brackets.

У доповіді [11] здійснено детальний аналіз редакторів коду за різними параметрами, серед яких: Sublime Text, Atom Editor, Visual Studio Code та WebStorm. Доповідач (начальник front-end-відділу, Full-stack Node.js розробник, користувач Sublime Text із досвідом у 6 років) робить опір на WebStorm та Visual Studio Code і, згідно якісній оцінці, обоє варіантів заслуговують уваги розробників.

IT-компанія Jelvix серед 15-ти IDE та JS-редакторів зробила наступні висновки: WebStorm – це найкращий спеціалізований інструмент для JavaScript. Atom та Visual Studio – найкращі для великих та складних проектів. Brackets використовують коли мова йде не про складні проекти і зручний для початківців. Komodo IDE заслуговує почесної згадки як дружній для користувача та багатомовний IDE з багатими можливостями [12].

Отже, для розробки проекту вирішено обрати IDE WebStorm.

2.4. Вибір менеджера пакетів

Система управління пакетами – це набір програмного забезпечення, що дозволяє управляти процесом установки, видалення, налаштування та оновлення різних компонентів програмного забезпечення. Часто така система є досить низькорівневою утилітою, тому її супроводжують більш високорівневі утиліти – менеджери пакетів, які можуть завантажувати їх з мережевого сховища та відстежувати залежності між ними.

Bower – вже неактуальний менеджер пакетів, сенсу поглиблюватись у нього немає. Встановлюється за допомогою npm, але має свою базу пакетів. Раніше мав особливості, що не реалізовані в npm, проте вже застарів. На сайті Bower є рекомендація переходити на Yarn або npm.

Npm і Yarn – два найпопулярніших менеджери пакетів для JavaScript [13].

Npm – це менеджер пакетів, що входить до складу Node.js. Його основне завдання – читати імена та версії пакетів з package.json і розгортати їх разом із залежностями в папку node_modules. Має глобальний кеш пакетів (по аналогії з GAC.NET). Пакети можуть бути службовими (devDependencies) і звичайними, що включаються в production-збірку.

Yarn був розроблений в Facebook щоб позбавиться від недоліків npm. По суті Yarn – це новий інсталятор, який, як і раніше, базується на структурі заданої npm. У Yarn доступні всі ті ж пакети що і в npm, тому перехід з npm на Yarn не вимагає великих зусиль.

Відмінності:

- Процес установки пакетів. При установці пакетів npm виконує необхідні дії послідовно, тобто кожен пакет повинен повністю встановитись один за одним. Yarn у свою чергу виконує кілька установок за один крок, що робить процес установки залежностей коротшим.

- Швидкість. Yarn на порядок швидший усіх версій npm менше 5.0. Коли команда npm випустила 5 версію вони заявили, що він буде в п'ять разів швидший своїх попередників. Але все ж npm залишився повільнішим.

- Безпека. Головна проблема npm полягає у тому, що він автоматично запускає код залежностей і дозволяє додавати залежності на льоту. Хоча такі особливості дають багато переваг, вони також створюють уразливості в безпеці. Так як Yarn встановлює залежності тільки з файлів yarn.lock або package.json, він вважається безпечнішим. Також Yarn перевіряє контрольні суми перед установкою, щоб гарантувати цілісність кожного пакету.

До першого офіційного релізу Yarn користувачі скаржились на проблеми з продуктивністю, але ці проблеми незабаром були вирішені. Оскільки Yarn підтримується такою великою компанією як Facebook, всі проблеми досить швидко вирішують. Тому Yarn на даний момент повинен бути досить стабільним.

Незважаючи на те, що Yarn вважається покращеною версією npm, він все ж має кілька невирішених проблем. Наприклад, одночасне використання npm і Yarn створює конфлікти. Ще однією проблемою є велика необхідність у дисковому просторі, оскільки Yarn зберігає залежності локально.

Yarn все більше набиратиме популярності завдяки своїй великій продуктивності і численним корисним командам. Проте npm все ще існує і стає потужнішим з кожною новою версією [14, 15] (рис. 2.8).

Отже, базуючись на проаналізованій інформації та опираючись на приведену статистику, прийнято рішення у процесі розробки проекту використовувати менеджер пакетів npm.

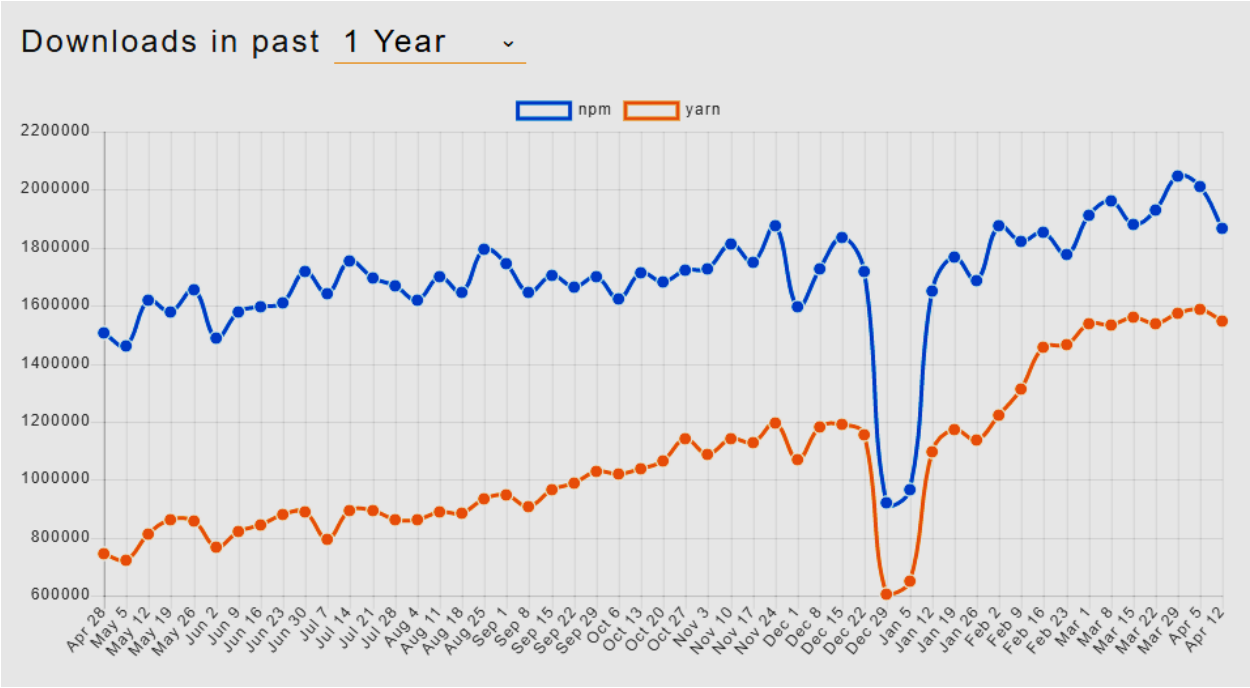

Рис. 2.8. Тренди npm за 2019-2020 рр. [16]

2.5. ES6+ JavaScript та Babel-транспайлер

JavaScript – об’єктно-орієнтована мова, але зі специфічною механікою спадкування через прототипи.

Базові поняття, які необхідно знати про JavaScript:

- області видимості змінних;
- особливості роботи об’єктів і прототипів;
- перетворення типів;
- замикання та їх особливості;
- типи даних;
- асинхронні операції;
- робота з об’єктами браузера;
- робота з подіями, спливання подій;
- маніпулювання з елементами DOM.

ECMAScript 6 (ES6) – це новий стандарт мови JavaScript, що має новий функціонал і був випущений у 2015-му році. Ці нові можливості мають різну ступінь складності та можуть бути корисні і для простих скриптів, і для складних додатків. Нові можливості стандарту:

- підтримка класів;
- шаблони рядків;
- підтримка лексичних оголошень змінних;
- оператор const;
- скорочений формат визначення об’єктів;
- модулі;
- arrow-функції;
- новий тип Symbol;
- генератори;
- об’єкти WeakSet і WeakMap;
- механізм Promise;
- значення за замовчуванням для аргументів функцій;
- передача у функцію довільної групи параметрів у формі масиву;
- структури даних Map та Set;
- підтримка абстракції масивів;
- безліч методів для рядків, масивів та математичних операцій.

ES6 став проривом. Відтоді комітет прийняв рішення про щорічний перегляд і випуск стандарту. У 2019-му році вийшов стандарт ES10.

На сьогоднішній день найбільшу підтримку у браузерах має версія ES5, проте наявність сучасного інструментарію для розробки (транспайлери JS) дозволяє писати код розробникам на тій версії стандарту, яка їм більше підходить.

Найпопулярнішим транспайлером є Babel [15]. Він також є і поліфілом. Підтримує ES6+ / TypeScript, JSX, Flow, переводячи код, написаний на цих мовах, в ES5, зрозумілий всім браузерам. Зазвичай Babel працює на сервері в складі системи збирання JS-коду (наприклад, webpack або brunch).

2.6. Препроцесор JSX

JSX (JavaScript XML) – це надбудова на JavaScript, яка дозволяє використовувати у ньому XML-подібний синтаксис [17].

JSX рекомендується використовувати при написанні React-компонентів, оскільки за допомогою нього простіше представити DOM-модель і у його коді легше розібратись. Можна використовувати React і без JSX, проте JSX робить React елегантнішим.

Код JSX, знайдений у JavaScript-файлах, за допомогою Babel-транспайлера перетворюється у стандартні JavaScript-об'єкти. JSX дозволяє писати стислі структури XML / HTML (наприклад, DOM-подібні деревовидні структури).

Особливості:

- У JSX можна вставляти JavaScript-вираз всередині фігурних дужок.
- JSX також є виразом. Це означає, що його можна використовувати всередині операторів if та циклів for, привласнити його змінній, прийняти в якості аргументу і повернути його із функції. Цей вираз не є ні рядком, ні HTML.
- За замовчуванням React DOM уникає будь-яких значень, вставлених в JSX до того, як проведе їх рендеринг. Таким чином можна бути впевненим у тому, що у додаток випадково ніколи нічого не попаде. Перед рендерингом все конвертується у стрічки. Це допомагає запобігти XSS (cross-site-scripting) атаки.
- JSX представляє собою об'єкт.
- Усередині JSX-тегів можуть міститися інші теги.
- Теги JSX мають ім'я та атрибути. Якщо значення атрибута укладено в лапки, то воно є стрічкою. В іншому випадку потрібно обернути значення у фігурні дужки. Це значення – вкладений вираз JavaScript.

2.7. Вибір інструменту управління станом даних в JavaScript-додатках

У міру того як вимоги до односторінкових JavaScript-додатків стають все вищими, є необхідність керувати все більшою кількістю станів (State) за допомогою JavaScript. Ці стани можуть включати в себе відповіді сервера, кешовані дані, а також дані, створені локально, але ще не збережені на

сервері. Це також відноситься до UI-станів, таким як активний маршрут (route), виділений таб, відображений спінер або нумерація сторінок і т.д.

Керувати станами, які постійно змінюються, складно. Якщо модель може оновити іншу модель, то вигляд може оновити модель, яка оновлює іншу модель, а це, в свою чергу, може викликати оновлення іншого вигляду. У якийсь момент стає незрозуміло, що відбувається у додатку. Стає неможливо відслідкувати коли, чому і як стан оновився. Коли система стає непрозорою і недетермінованою, важко виявити помилки або додавати нову функціональність.

Це досить неприємна ситуація, беручи до уваги нові вимоги, що стають звичними для front-end-розробки: обробка оптимістичних оновлень (optimistic updates), рендер на сервері, вилучення даних перед виконанням переходу на сторінку і т.д.

Ця складність виникає через те, що відбувається змішування двох концепцій, які дуже нелегкі для розуміння: зміни (mutation) та асинхронність (asynchronicity). Ці дві концепції можуть працювати разом, але в такому випадку вони створюють хаос. Бібліотеки, аналогічні React, намагаються вирішити цю проблему на рівні вигляду, видаляючи асинхронність і пряме маніпулювання DOM. Проте React дає можливість керувати станом даних розробнику за допомогою Redux.

Так само як і Flux, CQRS та Event Sourcing, Redux намагається зробити процес зміни стану передбачуваними за допомогою введення деяких обмежень на те, як і коли можуть відбутись оновлення.

Redux має змішаний спадок. Він схожий з деякими патернами та технологіями, але, в той же час, значно відрізняється від них: Flux, Elm, Immutable, Baobab, Rx та інші.

Redux ідеальний для односторінкових додатків, в яких управління станом згодом може стати складним. Redux не прив’язаний до якогось фреймворку, хоча він написаний з орієнтацією на React, його можна використовувати з Angular і навіть jQuery.

Дані в React «перетікають» через компоненти. Більш специфічно це називається «односпрямованим потоком даних» – дані перетікають в одному напрямку від батьків до нащадків. З цією характеристикою не цілком очевидно як будуть взаємодіяти два компонента, які не перебувають у відносинах «батько-нащадок» (рис. 2.9).

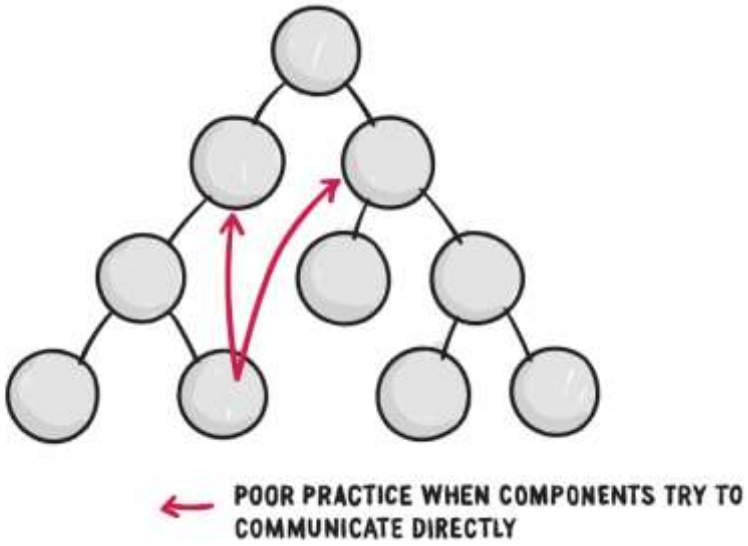

Рис. 2.9. Компоненти, що не перебувають увідносинах «батько-нащадок»

React не рекомендує використовувати безпосередню взаємодію компонентів. Навіть якби в React були можливості для підтримки цього підходу, він би розцінювався як погана практика, так як безпосередня взаємодія компонентів схильна до помилок і веде до спагетті-коду.

Redux пропонує вирішити це шляхом зберігання всього стану (state) програми в одному місці – сховищі (store). Компоненти передають (dispatch) зміни стану у сховище, а не безпосередньо іншим елементам. Компонент, якому важливі ці зміни, може підписатися (subscribe) на них в store (рис. 2.10).

Redux гарантує, що компоненти беруть інформацію про свої стани зі сховища, а усі компоненти відправляють зміни свого стану у сховище. Компонент повинен турбуватись лише про відправку змін свого стану у

сховище і не думати яким чином ці зміни вплинуть на інші компоненти. Так Redux робить потік даних простішим та зрозумілішим.

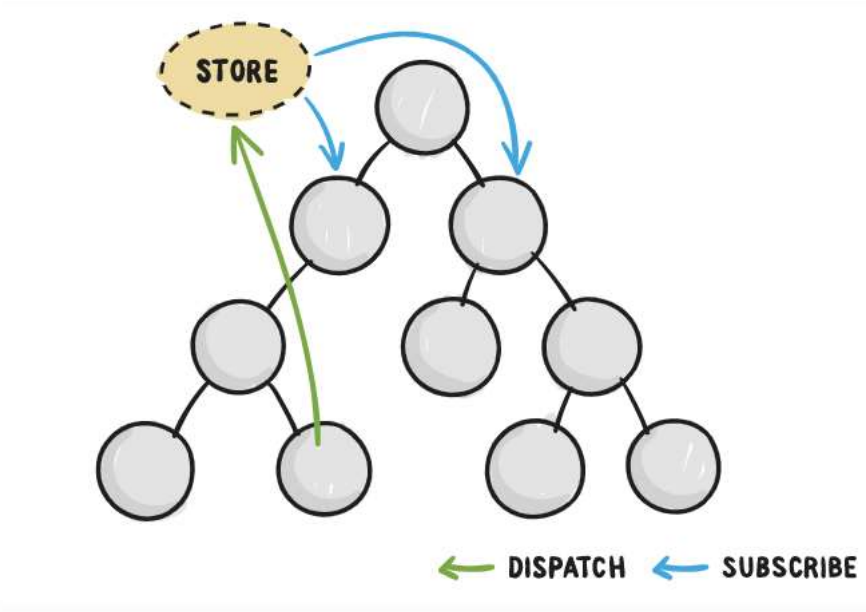

Рис. 2.10. Взаємодія компонентів додатку за технологією Redux

Ці обмеження відображені в трьох принципах Redux:

1. Єдине джерело правди. Стан всього додатку зберігається в дереві об’єктів всередині одного стору. Це полегшує створення універсальних програм.

2. Стан – тільки для читання. Єдиний спосіб змінити стан – це передати action (об’єкт, який описує що трапиться). Це гарантує, що вигляд або функції, що реагують на події мережі (network callbacks), ніколи не змінять стан безпосередньо. Оскільки всі зміни централізовані і застосовуються послідовно в строгому порядку, тому немає необхідності стежити за «гонкою станів».

Об’єкт store сам по собі володіє невеликим API, в якому всього чотири методи (потрібно зауважити, що метод задання стану відсутній):

- store.dispatch(action);
- store.subscribe(listener);
- store.getState();

- replaceReducer (nextReducer).

3. Зміни (mutation) здійснюються чистими функціями. Для визначення того, як дерево стану буде трансформовано за допомогою action, необхідно писати чисті редуктори (reducer). Редуктор бере попередній стан та action і повертає новий стан. Потрібно не забувати повертати новий об'єкт стану замість того, щоб змінювати попередній [18].

Властивості чистих функцій:

- вона не робить викликів за межі мережі або бази даних;
- вона повертає значення, що залежить лише від значень її параметрів;
- її аргументи не повинні бути змінені в процесі роботи функції;
- виклик чистої функції з однаковим набором аргументів повинен завжди повертати одне і те ж значення.

Вони називаються «чистими», тому що не роблять нічого, окрім повернення значення в залежності від їх параметрів. Вони не тягнуть ніяких побічних ефектів для іншої частини системи.

Redux – це невелика бібліотека, в якій угоди і API були ретельно продумані для зручного створення інструментів та підключення розширень екосистеми [19].

Існує багато інформації (статей, блогів тощо) щодо порівнянь інструменів для роботи зі станом React-додатка. У [20] стверджується, що Redux та MobX заслуговують однакової уваги.

Для простого розв'язання наскрізної передачі властивостей відмінно підходить технологія React Context API. Але для додатків більшого масштабу, які мають кілька (і / або більш складних) станів, редукторів і т.д., Redux буде найкращим рішенням [21, 22]. Ним варто користуватись тільки у випадках, коли неможливо управляти станом додатку за допомогою стандартного менеджера станів в React.

Оскільки інтернет-магазин міститиме багато функціоналу та у майбутньому планується його розширення, тому для управління станом додатку буде використано Redux.

2.8. Вибір збирача проекту

Коли ще сайти були невеликими, необхідності в окремій збірці для front-end не було. Однак обсяг і складність CSS та JS постійно збільшуються і вигляд, в якому зручно розробляти, став дуже відрізнятись від вигляду, в якому потрібно показувати результат користувачеві. З'явилися такі завдання, як конкатенація (склеювання) файлів, мінімізація коду і навіть попередня компіляція. Крім цього чим більше у проекті буде залежностей, тим більше помилок виникає при підключенні тегів <script> в правильному порядку. Результатом цього стали спеціалізовані системи збирання для front-end.

Як тільки необхідність у збиранні проекту стала відчутна, тут же на підтримку front-end почали приходити інструменти, що використовувалися для back-end. Більшість розробників так чи інакше взаємодіяли з webpack при роботі з проектами на React, Angular чи Vue. У більшості подібних ситуацій вистачає налаштувань за замовчуванням, тому освоєння самого webpack як правило відкладається на потім. Але знання базових налаштувань webpack можуть значно поліпшити процес розробки.

Webpack створює граф залежностей для JavaScript, CSS та ін., видаючи однофайлові збірки коду так, щоб була можливість імпортувати всі необхідні ресурси JavaScript лише одним тегом <script> [23].

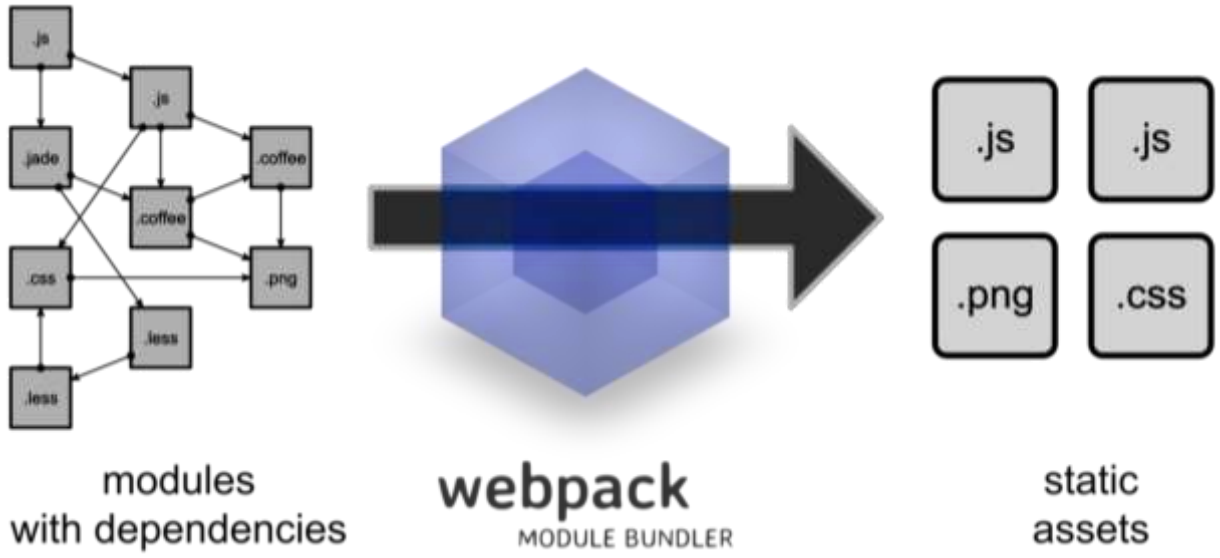

Рис. 2.11. Взаємодія компонентів додатку за технологією Redux

Попередниками і все ще широко використовуваними засобами є Grunt, Broccoli та Gulp [24].

Альтернативою використанню webpack є використання комбінації менеджерів завдань (таких як Grunt або Gulp) зі збирачем пакетів, подібним Browserify. Проте webpack вирішує завдання збірки більш інтегрованим і природним чином. У Browserify для цього доведеться використовувати Gulp / Grunt і довгий список додаткових модифікаторів та плагінів. Webpack пропонує досить великий функціонал за замовчуванням вже з коробки.

Webpack працює на основі файлу конфігурації, на відміну від gulp / grunt, де необхідно писати код для виконання своїх завдань. Залежно від конфігурації він може робити правильні припущення про те, що необхідно зробити, як працювати з різними модулями JS, як компілювати код і як управляти активами тощо. Так само є функція гарячого перезавантаження проекту live-reload. Можливість заміни вихідних імен файлів іменами хеш-файлів, які дозволяють браузерам легко виявляти змінені файли шляхом включення в ім’я файлу специфічного для збірки хеша. І це лише деякі основні моменти, які роблять webpack кращим вибором.

Особливості webpack:

- допомагає зібрати воєдино усі ресурси;
- може розділити вихідний файл на кілька файлів, з метою уникнення повільного завантаження сторінки через великий розмір JS-файлу;
- стежить за змінами і повторно виконує завдання;
- може виконувати транспіляцію JS ES6+ в ES5 за допомогою Bable;
- може конвертувати вбудовані зображення в URI-стрічку;
- дозволяє використовувати require() для CSS-файлів;
- може запускати webpack-dev-server;
- має вбудований livereload (перезавантаження браузера у реальному часі);
- може виконати транспіляцію CoffeeScript в JavaScript;
- може виконати Tree Shaking;

- може працювати з Hot Module Replacement.

Для збирання scss-файлів в один css-файл необхідно встановити пакет node-sass.

Webpack не обмежується одним лише front-end – його також успішно застосовують в back-end-розробці на Node.js [23].

Webpack, безумовно, є потужним помічником для розробки і його дуже легко налаштовувати під будь-які завдання. Оскільки він має усі потрібні налаштування за замовчуванням, тому для початківця webpack буде найкращим варіантом. А усі можливості, перераховані вище, роблять його потужним інструментом, який буде дуже корисним під час розширення проекту.

РОЗДІЛ 3.
РОЗРОБКА ІНТЕРНЕТ-МАГАЗИНУ

3.1. Інструмент швидкого старту Create React App

Існують набори інструментів React, які допомагають для вирішення таких завдань як:

- масштабування до великої кількості файлів та компонентів;
- використання сторонніх бібліотек із npm;
- раннє виявлення поширених помилок;
- відображення змін CSS і JS на льоту в процесі розробки;
- оптимізація коду для production.

Create React App – зручне середовище для вивчення React і кращий спосіб почати створення нового односторінкового додатку на React.

Інструмент налаштовує середовище для використання нових можливостей JavaScript, оптимізує додаток для production і забезпечує комфорт під час розробки. Потрібен лише Node.js не нижче версії 8.10 і npm не нижче версії 5.6.

Для створення проекту потрібно виконати команди:

```
npx create-react-app book-store
cd book-store
npm start
```

В результаті буде створено папку book-store із початковим набором файлів. Структура проекту зображена на рисунку 3.1.

npx у першому рядку не є помилкою. Це інструмент запуску пакетів, доступний у версіях npm 5.2 і вище.

npx – інструмент, призначений для того, щоб допомогти стандартизувати досвід використання npm-пакетів: так само, як і npm спрощує установку та управління залежностями, розміщеними в реєстрі, npx спрощує використання CLI-утиліт та інших виконуваних файлів. Це значно спрощує ряд речей, які робились раніше за допомогою звичайного npm.

Create React App не обробляє back-end-логіку або бази даних. Він тільки надає команди для збирання front-end, тому можна використовувати його із будь-яким back-end. Create React App включає Babel та webpack із попередніми налаштуваннями, що є дуже зручно.

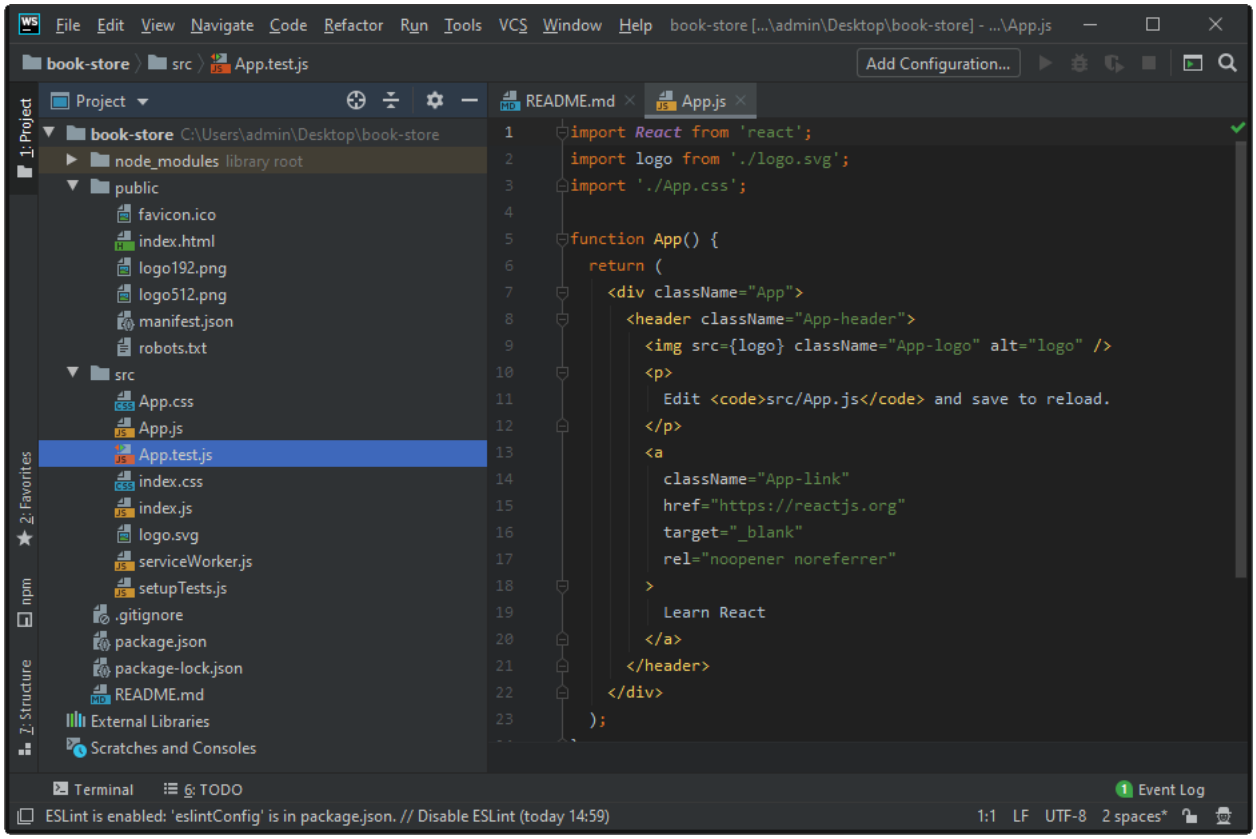

Рис. 3.1. Структура файлів у новому проекті

Для запуску сервера на локальному хості 3000 у командній стрічці потрібно виконати команду:

```
npm start
```

З’явиться повідомлення, що сайт запущений на локальному хості за адресою http://localhost:3000 (рис. 3.2).

Коли додаток готовий до розгортання в production, запуск команди npm run build створить оптимізовану збірку застосунку в папці build.

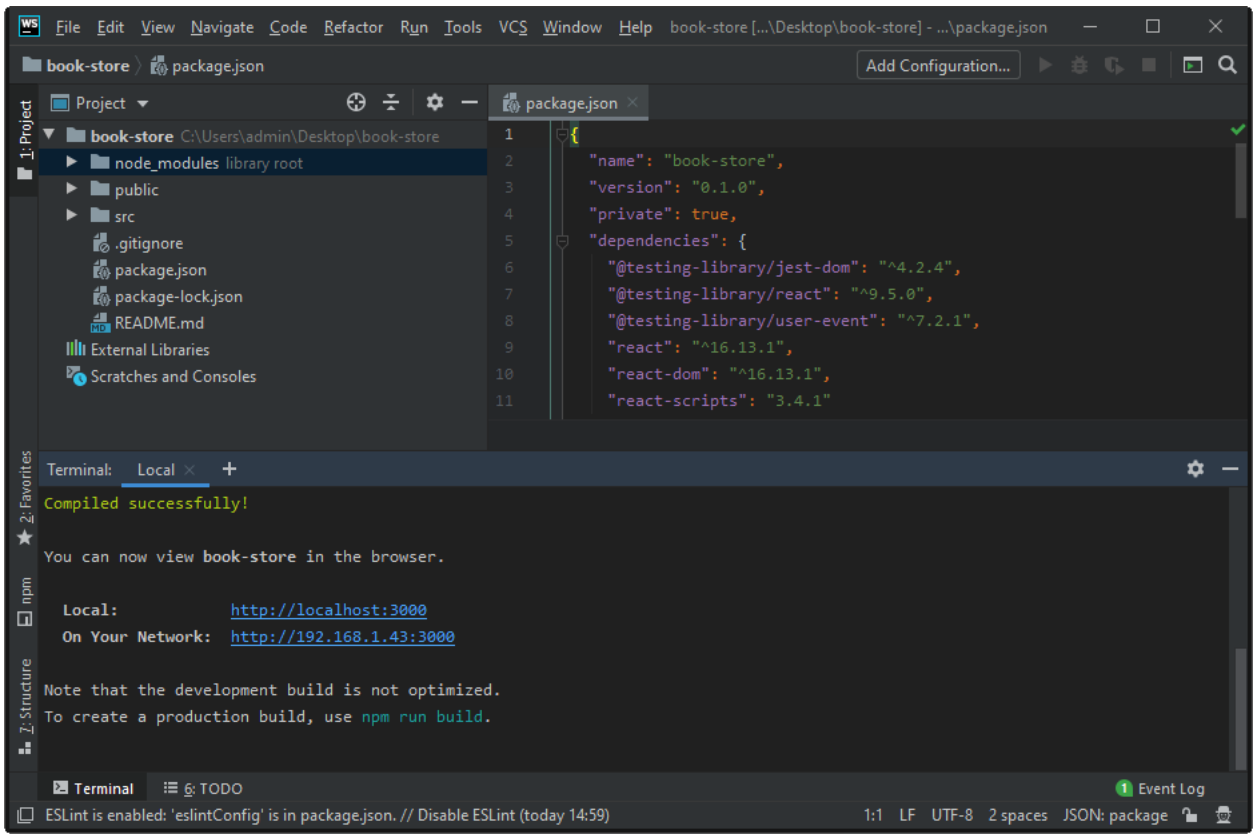

Рис. 3.2. Запуск проекту на виконання

Отже, разом з Create React App було отримано готовий, налаштований сервер, Babel-інтерпретатор, webpack для збірки, засоби для написання css-коду без префіксів для кросбраузерності та налаштоване середовище для тестування і збирання проекту. У середовищі буде все, що потрібно для створення сучасного React-додатку:

- React, JSX і ES6;
- додаткові можливості мови за межами ES5: деструктуризація, стрілкові функції, шаблони стрічок, проміси, підтримка асинхронності (async/await), спосіб створення змінних через let та const;
- dev-сервер, який працює незалежно від незначних помилок;
- можливість імпортувати CSS-файли та зображення безпосередньо з JavaScript;
- autoprefixer CSS, що дозволяє відмовитись від -webkit та інших префіксів;

- сценарій збірки для об’єднання JS, CSS та зображень для production за допомогою sourcemaps.

3.2. Структура проекту

Перебуваючи в папці проекту, потрібно відкрити файл, який знаходиться за адресою public / index.html. Тут особливу увагу слід звернути на рядок `<div id = 'root'></div>`. Саме тут буде знаходитись React-додаток. Весь цей рядок буде замінено на код програми, а все інше залишиться незмінним (рис. 3.3).

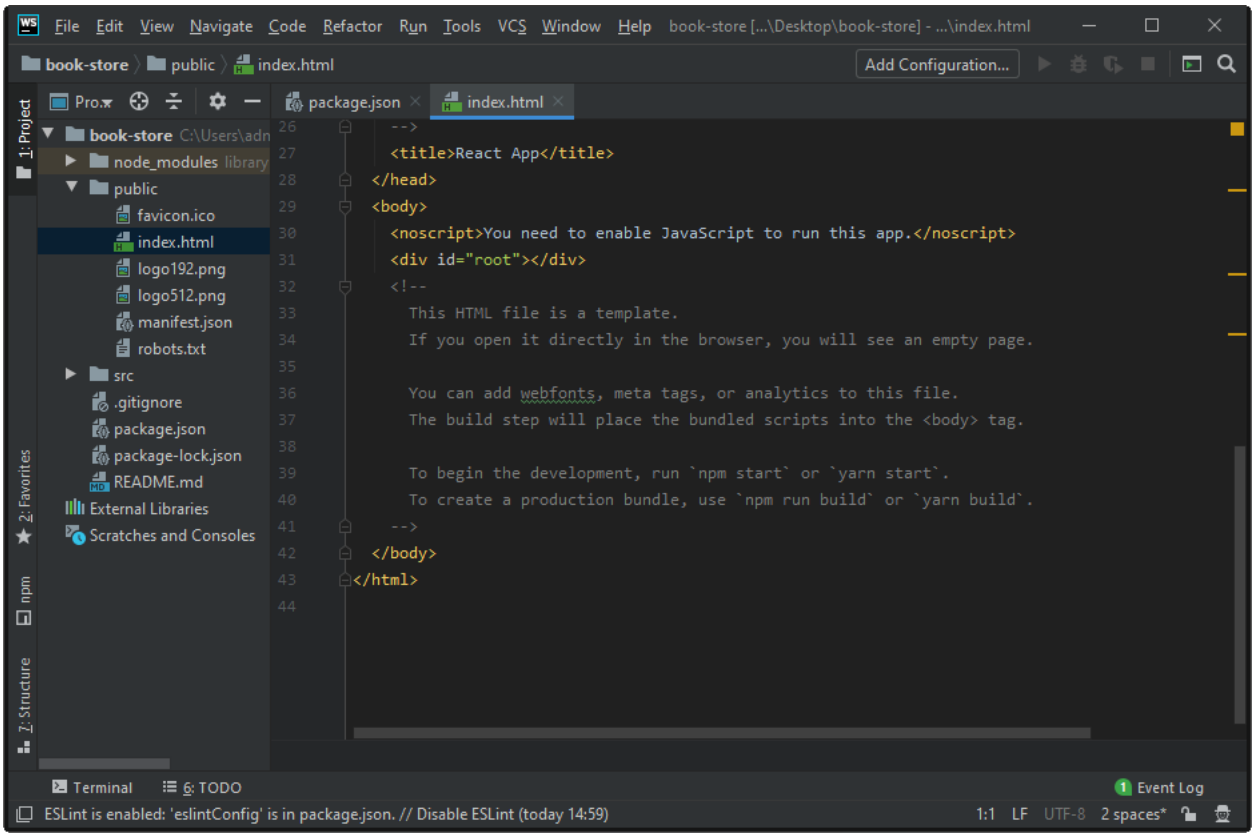

Рис. 3.3. Головний файл проекту

Саме файл src / index.js виконує розгортання React-додатку (рис. 3.4). Крім того увесь вихідний код програми буде розміщуватись в директорії src. Ось рядок коду, який відповідає за підключення:

```
ReactDOM.render(<App />, document.getElementById ('root'));
```

Цей рядок повідомляє React про те, що потрібно взяти компонент App і помістити його в div-елемент із id root, який був визначений у щойно розглянутому файлі index.html.

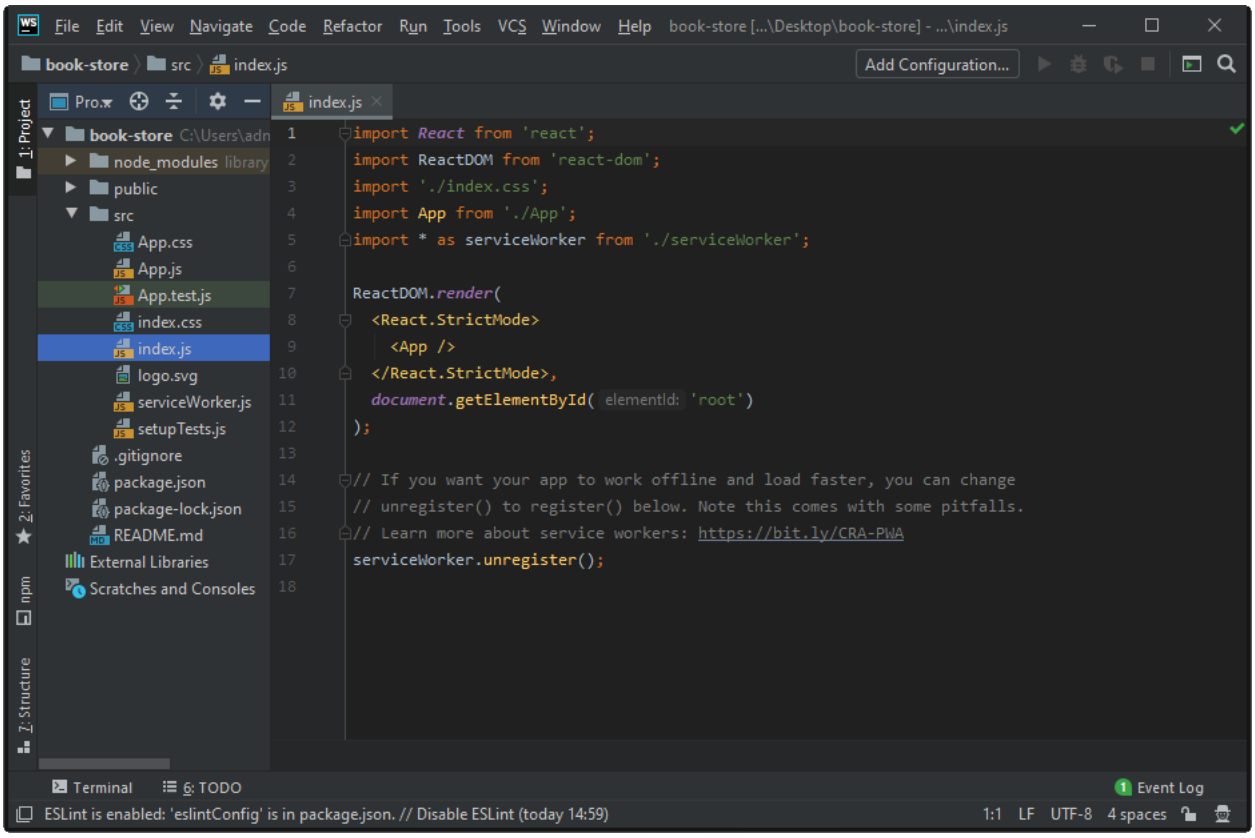

Рис. 3.4. Головний js-файл

Конструкція <App /> дуже схожа на HTML-код, але це – зразок JSX-коду. Необхідно звернути увагу на те, що ця конструкція починається з великої літери: <App /> не те ж саме, що й <app />. В React прийнято угоду по іменування сутностей. Такий підхід дозволяє системі розрізняти звичайні HTML-теги і компоненти React. Якщо імена компонентів не будуть починатися з великої літери, React не зможе вивести їх на сторінку.

Якщо у js-файлі планується використання JSX, то необхідно імпортувати React, скориставшись наступною командою:

```
import React from 'react';
```

Код компоненту App знаходиться у файлі src / App.js (рис. 3.5).

```
import React, {Component} from 'react';
import logo from './logo.svg';
import './App.css';

class App extends Component {
  render() {
    return (
      <div className="App">
        <header className="App-header">
          <img src={logo} className="App-logo" alt="logo"/>
          <p>
            Edit <code>src/App.js</code> and save to reload.
          </p>
          <a
              className="App-link"
              href="https://reactjs.org"
              target="_blank"
              rel="noopener noreferrer"
          >
            Learn React
          </a>
        </header>
      </div>
    );
  }
}
```

Рис. 3.5. Код компоненту App

Одна з найбільш приємних можливостей React полягає в тому, що ця бібліотека не примушує розробника до суворого дотримання якихось угод, що стосуються структури проекту. Багато що в цьому плані залишається на розсуд програміста. Цей підхід відрізняється від того, який, наприклад, прийнятий у фреймворків Ember.js або Angular. У цих фреймворках передбачені і угоди, що стосуються структури проектів і правила іменування файлів та компонентів.

Однак існують деякі підходи до структури React-проектів [25]:

- групування по функціональності або маршруту;
- групування по типу файлу;
- архітектура Arc.js [26].

Ознайомившись із можливими варіантами побудови структури проекту та перейнявши досвід із вже існуючих, прийнято рішення створити структуру, зображену на рисунку 3.6.

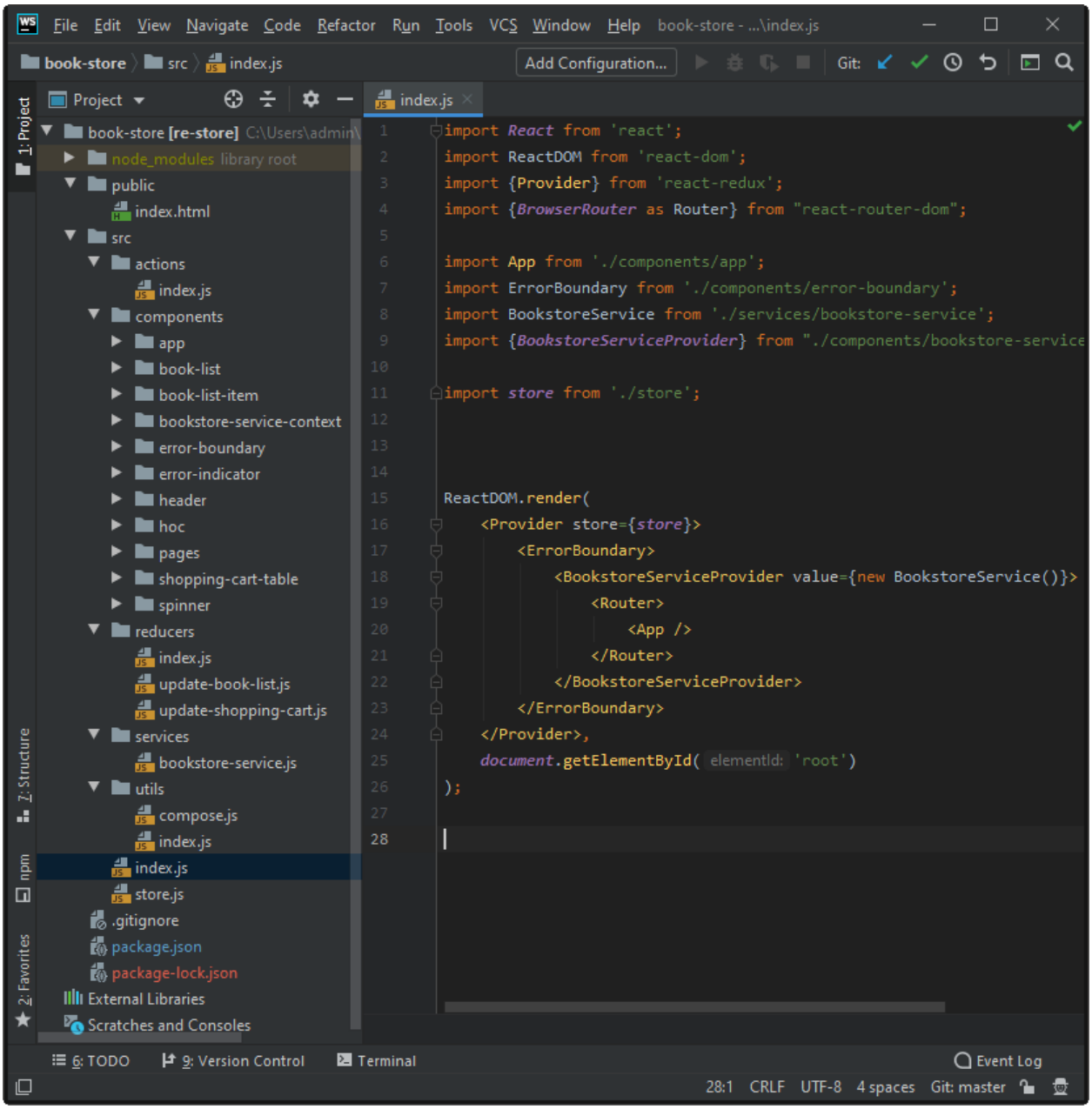

Рис. 3.6. Кінцева структура проекту

Деякі файли, згенеровані інструментом Create React App, що зображені на рисунку 3.5, було видалено – залишився лише файл src / app.js, який згодом було переміщено у папку src / components / app. Решта папок було створено із названо за рекомендаціями, викладеними вище.

Отже, структура проекту містить такий набір папок:

- node_modules – розташовані пакети для роботи доатку;
- public – публічні файли;
- src / store – Redux-сховище;
- src / actions – містяться actions та action creators;
- src / components – набір компонентів (кожен компонент знайходиться в окремій папці і, крім того, включає власний scss-файл);
- src / components / hoc – набір компонентів вищого порядку;
- src / components / app – каталог, в якому знаходиться компонент найвищого рівня ієрархії;
- src / reducers – набір чистих функцій;
- src / services – деякі абстракції для зовнішнього API (наприклад, для бекенда)
- src / utils – набір утиліт та допоміжних функцій.
- .gitignore – файл для git (містяться правила, за якими обираються файли, що не допустимі для завантаження у git-репозиторій);
- package.json –файл із залежностями проекту;
- README.md – опис проекту.

3.3. React.js: підключення та особливості використання

Для підключення бібліотеки React достатньо скачати пакети react і react-dom та додати у файл src / index.js наступні рядки:

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './components/app';
ReactDOM.render(<App />, document.getElementById('root'));
```

Метод ReactDOM.render першим аргументом приймає компонент, а другим – елемент DOM-дерева. Компоненти – основні будівельні блоки React. Їх можна створювати, застосовуючи два підходи:

1. Полягає у використанні компонентів-класів (Class Component).
2. Полягає у використанні функціональних компонентів (Functional Component).

Причини, відповідно до яких не рекомендується використовувати компоненти, що базуються на класах, віддаючи перевагу функціональним компонентам:

- Компоненти, засновані на класах, на відміну від функціональних компонентів, складніше тестувати.

- Класи компонентів погано підтримують концепцію поділу відповідальності. Якщо розробник не приділяє якості коду належної уваги, він буде складати весь код в один клас, який з часом може дуже вирости (навіть більше 1000 рядків).

- Використання класів компонентів підштовхує розробника до того, щоб розміщувати в одному і тому ж місці логіку програми та опис його інтерфейсу. А це, знову ж таки, погано з точки зору концепції поділу відповідальності.

- Компоненти, створені з використанням класів, не можна назвати сутностями, що нагадують чисті функції. Це ускладнює прогнозування поведінки подібних компонентів. Функціональні компоненти, з іншого боку, є чистими функціями. Це виражається в тому, що вони завжди, при передачі їм одних і тих же вхідних даних, видають одну і ту ж розмітку.

- Застосування функціональних компонентів сприяють тому, щоб розробник розмірковував про архітектуру додатків, що, в свою чергу, покращує її.

- При використанні функціональних компонентів немає необхідності користуватися ключовим словом this, яке завжди було джерелом плутанини.

Прикалад компоненту на основі класу:

```
class CartPage extends Component {
    render() {
        return (
            <div className="cart-page row">
                <div className="col-md-12">
                    <ShoppingCartTable/>
                </div>
            </div>
        );
```

```
        }
    }
```

Прикалад функціонального компоненту:

```
    const withBookstoreService = () => (View) => {
        return (props) => {
            return (
                <BookstoreServiceConsumer>
                    {
                        (BookstoreService) => {
                            return  <View {...props}
bookstoreService={BookstoreService}/>
                        }
                    }
                </BookstoreServiceConsumer>
            )
        }
    };
```

Основні моменти, що розкривають суть React:

- JSX – дозволяє використовувати HTML-подібний синтаксис, що буде трансформовано у легкі JavaScript об’єкти;
- Virtual DOM – аналог DOM дерева;
- React.Component – спосіб створення нового компоненту;
- render (метод) – визначає UI окремого компоненту;
- ReactDOM.render – рендеринг React-компоненту у DOM;
- state – внутрішнє сховище даних (об’єктів) компоненту;
- constructor (this.state) – встановлює початковий стан компоненту;
- setState – допоміжний метод, що використовується для оновлення стану компоненту і повторного рендерингу інтерфейсу користувача;
- props – дані, що передаються від батьківського до дочірнього компоненту;
- propTypes – дозволяє контролювати типи окремих props, що передаються до дочірніх компонентів;
- defaultProps – дозволяє задавати компоненту props за замовчуванням.

Головні особливості React представлені на рисунку 3.7.

Рис. 3.7. Головні особливості React

Життєвий цикл компоненту:

- componentDidMount – виконується після встановлення (mount) компоненту;

- componentWillUnmount – виконується перед тим, як компонент буде знищено;

- getDerivedStateFromProps – виконується, коли компонент встановлюється або коли props змінюються; використовується для оновлення стану компонента, коли props змінюються.

Події: onClick, onSubmit, onChange.

3.4. Застосування Redux та механізм його роботи

Redux – це спосіб управління станом додатку. Він заснований на декількох концепціях, вивчивши які, можна з легкістю вирішувати проблеми зі станом. Далі показано як влаштований Redux та механізм його роботи (рис. 3.8).

3.4.1. Незмінне дерево станів

У Redux загальний стан додатку представляється одним об’єктом JavaScript – state (стан). Це незмінне дерево станів доступне тільки для

читання. Вносити змінити напряму заборонено. Зміни можливі лише при відправці action (дії). У цій концепції більше не потрібне використання методу setState().

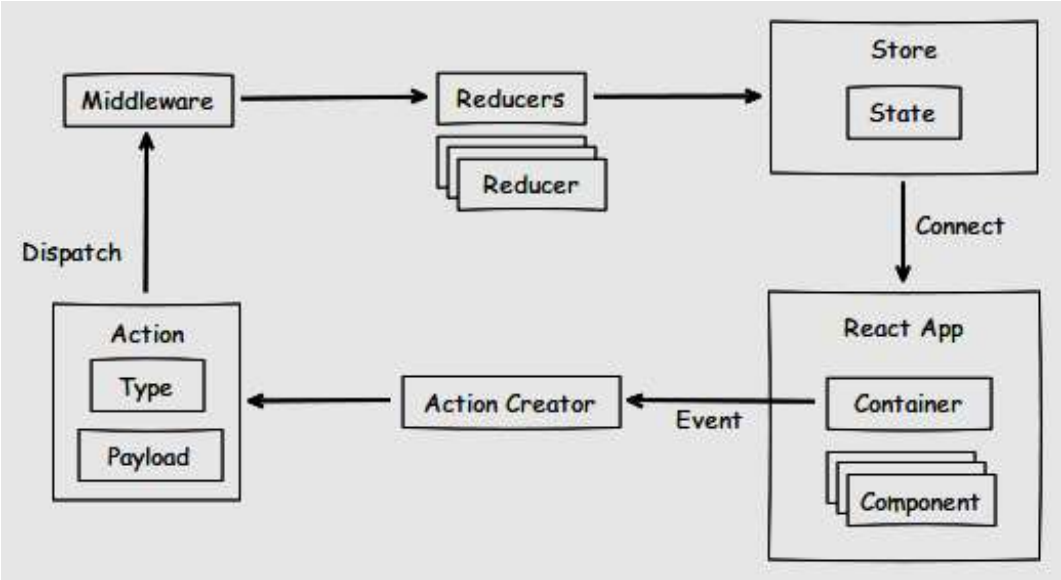

Рис. 3.8. Механізм роботи Redux

3.4.2. Дії

Дія – це JavaScript-об’єкт, який здійснює лаконічний опис суті зміни:

```
{
    type: 'FETCH_BOOKS_SUCCESS',
    payload: newBooks
}
```

Єдина вимога до об’єкта дії – наявність властивості type, значенням якої зазвичай є стрічка. Перелік об’єктів action даного проекту представлено у додатку Б.

3.4.3. Типи дій – константи

У простому додатку тип дії задається стрічкою. У міру розростання функціональності додатку краще переходити на константи:

```
const FETCH_BOOKS_SUCCESS = 'FETCH_BOOKS_SUCCESS'
const action = {
    type: FETCH_BOOKS_SUCCESS,
    payload: 'Third item'
}
```

Ці дії потрібно виносити в окремі файли, а потім їх імпортувати:

```
import {
    FETCH_BOOKS_SUCCESS,
    FETCH_BOOKS_REQUEST
} from './actions'
```

3.4.4. Генератори дій

Генератори дій (actions creators) – це функції, що створюють дії.

```
const booksLoaded = (newBooks) => {
    return {
        type: 'FETCH_BOOKS_SUCCESS',
        payload: newBooks
    }
};
```

Зазвичай ініціюються разом з функцією надсилання дії:

```
dispatch(booksLoaded(newBooks));
```

Генератори дій можна створювати при визначенні функції:

```
const dispatchBooksLoaded = newBooks =>
    dispatch(booksLoaded(newBooks));
dispatchBooksLoaded ('Milk');
```

Перелік генераторів дій, описаних вище, представлено у додатку Б.

3.4.5. Редуктори

При запуску дії обов’язково щось відбувається і стан додатку змінюється. Цим займаються редуктори.

Редуктор (reducer) – це чиста функція, яка обчислює наступний стан дерева на підставі його попереднього стану та дії, що застосовується (підрозділ 2.7):

```
(currentState, action) => newState
```

Редуктор не повинен:

- змінювати значення аргументів, що передаються;

- змінювати стан (замість цього створюється новий стан за допомогою методу Object.assign ({}, ...));
- мати побічні ефекти (ніяких API-викликів з якими-небудь змінами);
- викликати нечисті функції (це функції, результат яких залежить від чогось окрім їх аргументів – наприклад, Date.now () або Math.random ()).

Розроблений інтернет-магазин містить два файли з редукторами:

- update-book-list.js – редуктори для завантаження списку книг (рис. 3.9);
- update-shopping-cart.js – редуктори для роботи із кошиком покупця: додавання чи вилучення одиниці замовлення чи зміни їх кількості (рис. 3.10).

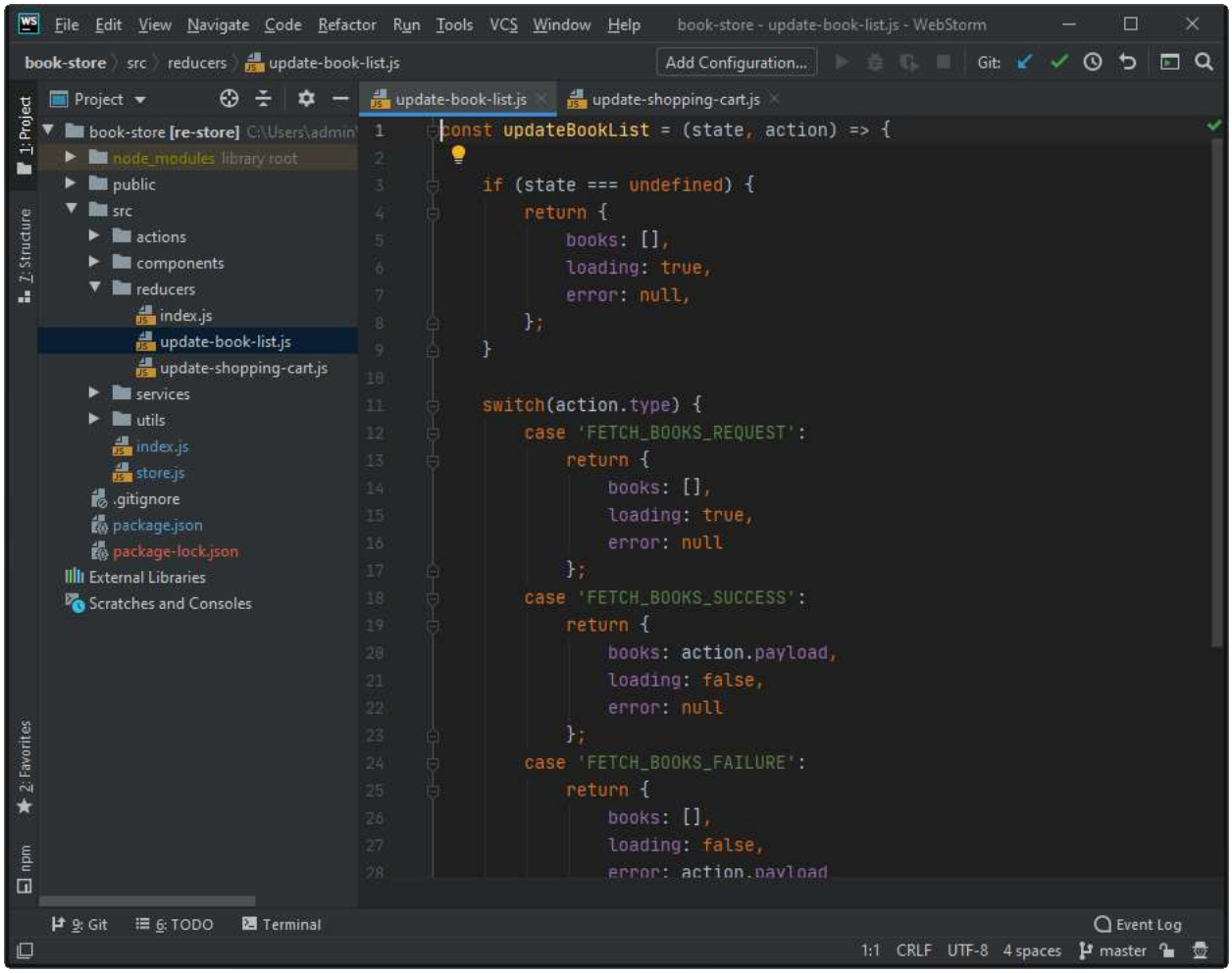

Рис. 3.9. Редуктори для роботи зі списком книг

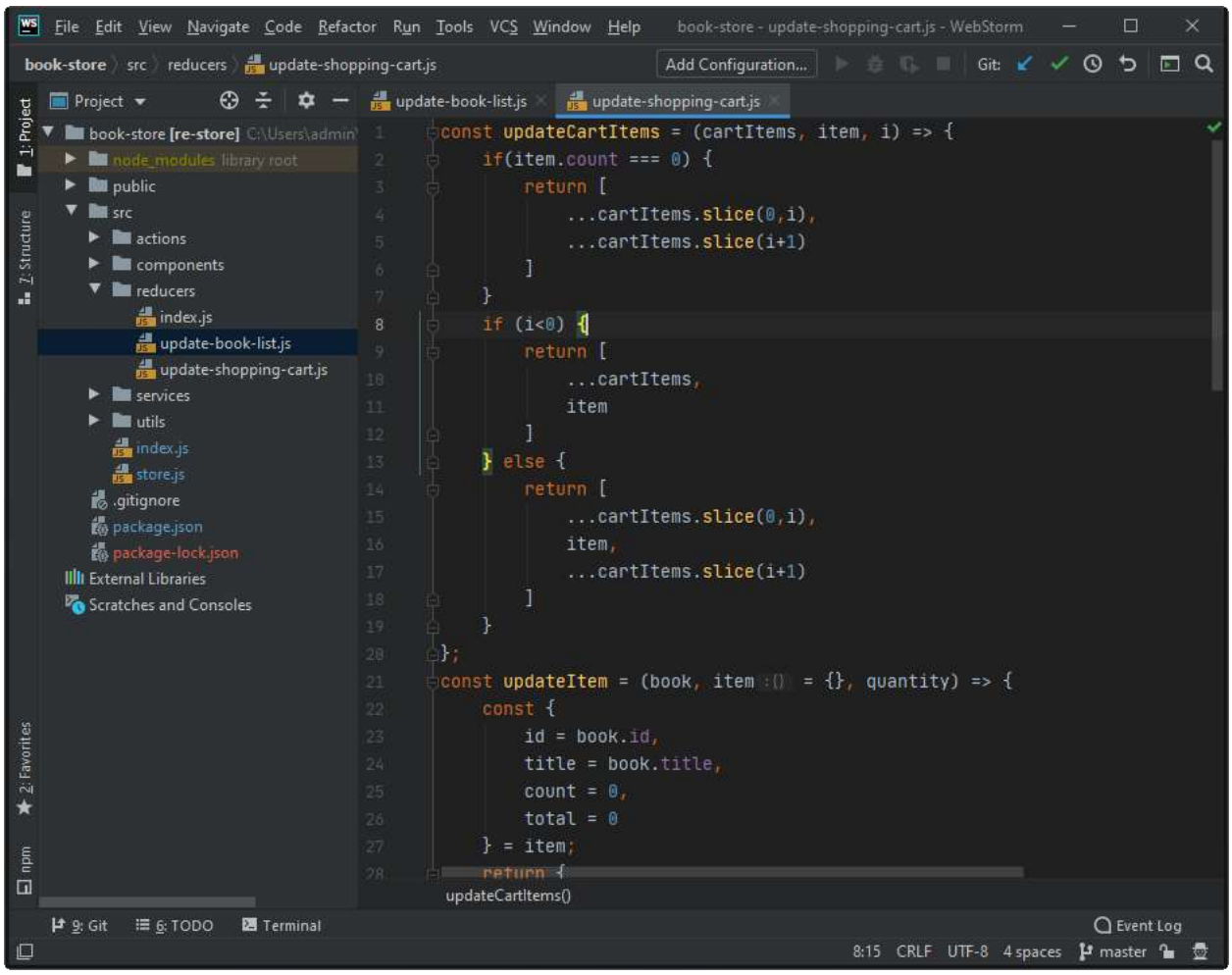

Рис. 3.10. Редуктори для роботи із кошиком

3.4.6. Сховище

Сховище (store) – це об’єкт, який:

- містить у собі дерево стану;
- відображає стан через getState();
- може постійно оновлюватись через dispatch();
- дозволяє реєструватись (або видалятись) в якості слухача зміни стану через subscribe().

Сховище в додатку завжди унікальне. Механізм створення сховища відображений на рисунку 3.11, а підключення здійснюється у головному js-файлі проекту (рис. 3.12).

```
import {createStore, applyMiddleware} from 'redux';
import thunkMiddlware from 'redux-thunk';
import reducer from './reducers';

const logMiddleware = (store) => (next) => (action) => {
    console.log(action.type);
    return next(action)
};
const stringMiddleware = (store) => (next) => (action) => {
    if (typeof action === 'string') {
        return next({
            type: action
        })
    }
    return next(action)
};

const store = createStore(reducer, applyMiddleware(
    thunkMiddlware,
    stringMiddleware,
    logMiddleware
));
```

Рис. 3.11. Ініціалізація Redux-сховища

```
import React from 'react';
import ReactDOM from 'react-dom';
import {Provider} from 'react-redux';
import {BrowserRouter as Router} from "react-router-dom";

import App from './components/app';
import ErrorBoundary from './components/error-boundary';
import BookstoreService from './services/bookstore-service';
import {BookstoreServiceProvider} from "./components/bookstore-service-c
import store from './store';

ReactDOM.render(
    <Provider store={store}>
        <ErrorBoundary>
            <BookstoreServiceProvider value={new BookstoreService()}>
                <Router>
                    <App />
                </Router>
            </BookstoreServiceProvider>
        </ErrorBoundary>
    </Provider>,
    document.getElementById( elementId: 'root')
);
```

Рис. 3.12. Підключення Redux-сховища

3.4.7. Потік даних

Потік даних в Redux завжди однонаправлений.

Передача дій з потоками даних у сховище відбувається через виклик методу dispatch():

```
store.dispatch(action);
```

Саме сховище передає дії редуктора і генерує наступний стан, а потім оновлює його і повідомляє про це всіх слухачів.

Метод store.dispatch() викликається у файлі store.js (додаток Г).

3.5. Інтерфейс та функціональні можливості сайту

Розроблений сайт є прототипом потужного інтернет-магазину. Він містить перелік книг (рис. 3.13) та кошик із вибраними книгами (рис. 3.14).

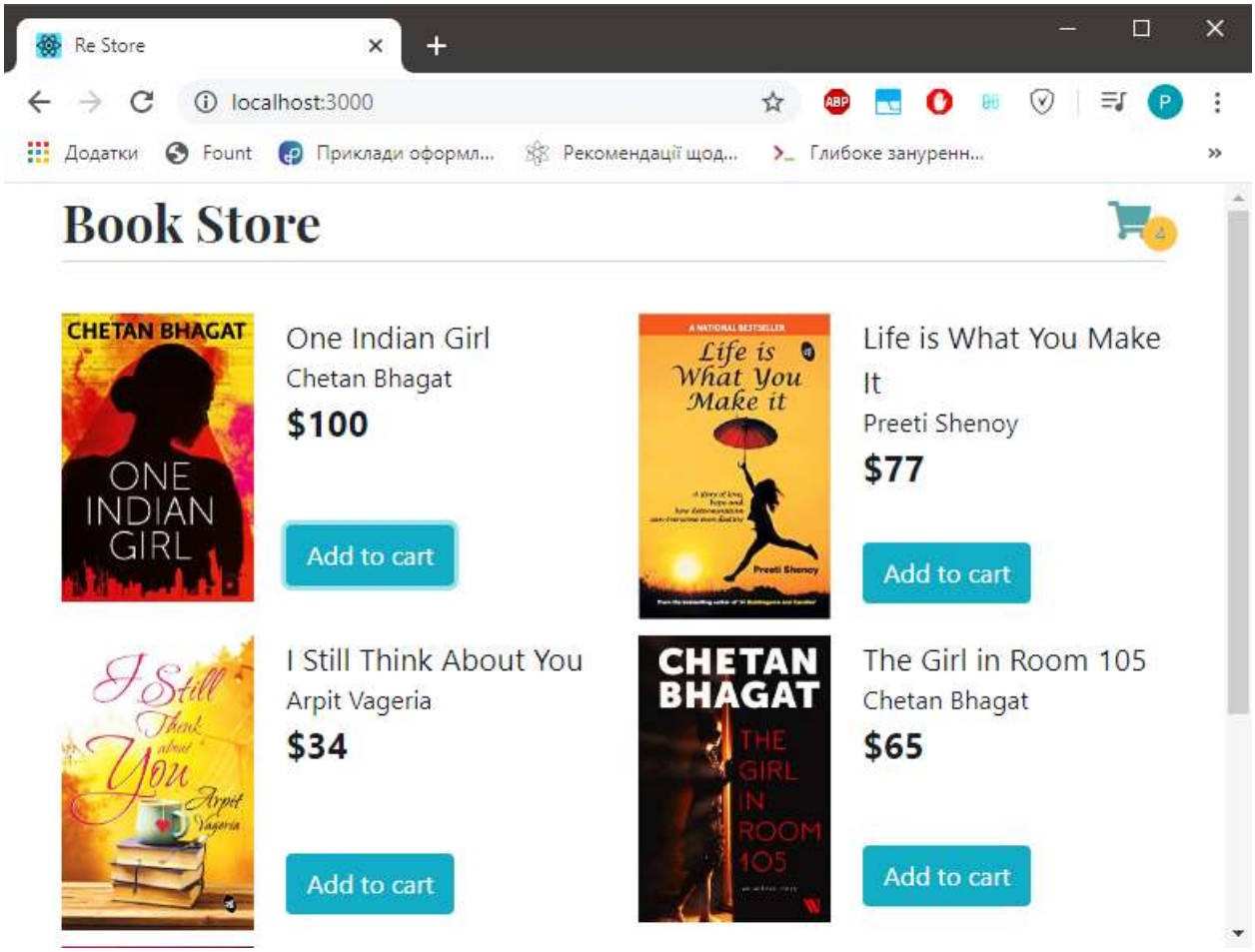

Рис. 3.13. Каталог книг

Кожна книжкова одиниця включає назву, ім’я її автора, ціну, постер книги та кнопку для додавання до кошика. Іконка кошика розташована у правому верхньому куті додатку. Після вибору першої книги біля піктограми з’являється інформація про кількість вибраних книг.

Після натискання на іконку кошика користувач переходить на сторінку /cart. Ця сторінка відображає список обраних книг у вигляді таблиці. Кожна стрічка таблиці містить порядковий номер, назву книги, кількість одиниць, загальну вартість та набір кнопок для збільшення / зменшення одиниць книги та видалення цієї стрічки зі списку. Внизу таблиці вказується загальна вартість усіх покупок.

Є можливість перейти назад на головну сторінку при натисканні на Book Store.

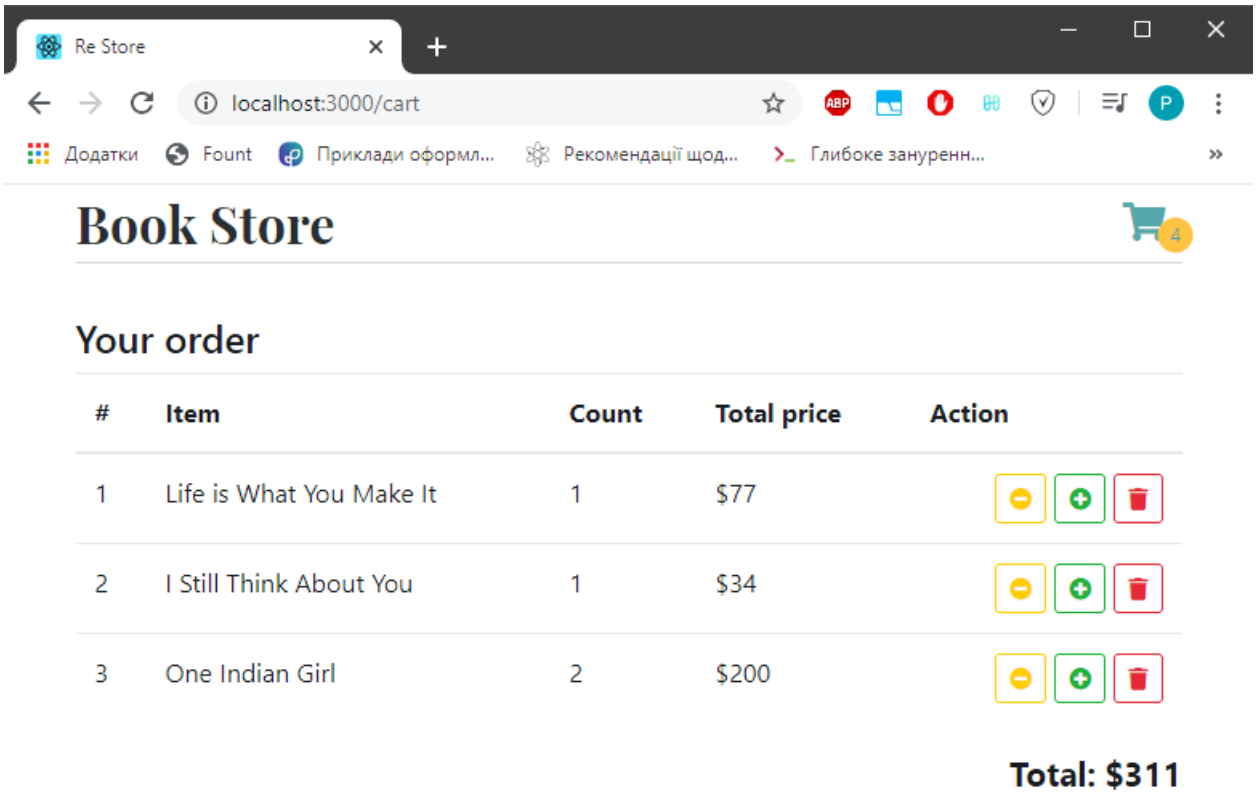

Рис. 3.14. Кошик

На даний момент сайт немає широкого функціоналу. Але весь код написано вручну. Хоча онлайн конструювання сайтів має ряд таких переваг як ціна, швидкість створення, відсутність програмування, уже готова структура сайту, проте вибраний підхід дає можливість самому визначати дизайн, структуру сайту та його наповнення. Він дозволяє уникнути обмежень, пов'язаних із відсутністю можливості редагування коду вручну, щомісячною платою та можливостями розширення функціоналу.

В подальшуму плануються розширення функціоналу, а саме:

- реєстрація та логінування відвідувачів інтернет-магазину;
- система пошуку товарів на сайті;
- фільтри товарів за характеристиками, тобто можливість сортувати товари за різними критеріями (назва, дата, ціна, товарні характеристики, бренд, категорія тощо);
- мобільна версія магазину;
- індивідуальні пропозиції покупцеві на основі зроблених замовлень;
- управління правами доступу і розподіл ролей на сайті (vip-покупець, звичайний покупець, незареєстрований покупець);
- зручна картка товару;
- методи зворотного зв'язку;
- інтелектуальний розрахунок вартості доставки, на основі ваги, габаритів, віддаленості тощо.

Отже, даний інтернет-магазин хоча і не має багатого функціоналу, проте є хорошим фундаментом для подальшої розробки унікальної інформаційної системи управління маркетинговою діяльністю.

ВИСНОВКИ

У випускному кваліфікаційному проекті представлено результати теоретичних і прикладних досліджень, що полягають у розробці інформаційної системи, яка призначена для управління маркетинговою діяльністю. Результатом досліджень стала розробка web-сайту, а саме інтернет-магазину, за допомогою сучасних технологій та методів.

В результаті проведених досліджень було отримано такі висновки:

- Мережа Інтернет надає можливість для збору та розподілу інформації і служить новим каналом продажу та просування товарів і послуг. Зрештою мережа об’єднує інформацію для управлінських дій на всіх рівнях компанії та забезпечує нові електронні зв’язки для того, щоб полегшити зв’язок із клієнтами та партнерами ланцюга постачання. Тому інтернет-маркетинг безсумнівно лідирує в ефективності порівняно із класичним маркетингом.

- Перенесення традиційної торгівлі у всесвітню павутину робить її гнучкішою, оскільки електронна торгівля, базуючись на цифровій інформації, полегшує співпрацю людей. Web-сайт – потужний засіб для реалізації поставленої цілі.

- Із розвитком ринкових відносин в мережі Інтернет та появою нових інформаційних технологій змінюються і можливості, технології та методи створення сайту, вимоги до нього. Це зумовлює необхідність пошуку нових підходів з урахуванням сучасних ринкових вимог та інноваційних можливостей інформаційних технологій.

- Найкращими засобами та технологіями розробки web-сайту на сьогоднішній день є бібліотека React з використанням преспроцесора JSX, CSS-препроцесор, бібліотека Redux як інструмент управління станом даних, менеджер пакетів npm, збирач проекту webpack, стандарти JavaScript ES6+ та транспайлер Babel. Найкращим спеціалізованим інструментом для роботи з JavaScript є інтегроване середовище розробки WebStorm.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Котлер Ф. Основы маркетинга: краткий курс. – М.: Вильямс, 2016. – 488 с.

2. Чередниченко Ю.В. Маркетинг в Интернете: сайт, который зарабатывает / Ю.В. Чередниченко. – 2-е изд. – Санкт-Петербург [и др.]: Питер, 2013. – 175 с.

3. Дурович А.П. Маркетинг в условиях глобализации: монография / А.П. Дурович; Федерация профсоюзов Беларуси «Международный университет «МИТСО». – Минск: МИТСО, 2016. – 147 с.

4. Лещев Д.В. Создание интерактивного web-сайта: Учеб. курс /Д.В. Лещев. – СПб.: Питер, 2003. – 544 с.

5. Ноблес Р. Эффективный Web-сайт / Р. Ноблес, К.-Л. Греди. – М.: Триумф, 2004. – 560 с.

6. Веб-розробка: етапи, стандарти, реальні проекти [Електронний ресурс] / WEB-SYSTEMS.SOLUTIONS // Блог. – Електрон. дані. – Режим доступу: https://web-systems.solutions/blog/web-development-stages-standards-projects/. – Назва з екрану. – Дата публікації: 11.02.2019. – Дата перегляду: 19.03.2020

7. Фоменко А. Что лучше всего изучать в 2020 году: Angular, React или Vue.js [Електронний ресурс] / Александр Фоменко // JAVASCRIPT. – Електрон. дані. – Режим доступу: https://badcode.ru/chto-luchshie-vsiegho-izuchat-v-2020-ghodu-angular-react-ili-vue-js/. – Назва з екрану. – Дата публікації: 29.02.2020. – Дата перегляду: 20.03.2020.

8. Front End Frameworks [Електронний ресурс] / State of JavaScript. – Електрон. дані. – Режим доступу: https://2019.stateofjs.com/front-end-frameworks/. – Назва з екрану. – Дата публікації: 29.12.2019. – Дата перегляду: 25.03.2020.

9. Kryukov D. Sass vs Less: Which CSS Preprocessor to Choose in 2019 [Електронний ресурс] / Denis Kryukov. – Електрон. дані. – Режим доступу:

https://blog.soshace.com/sass-vs-less-which-css-preprocessor-to-choose-in-2019/. – Назва з екрану. – Дата публікації: 02.07.2019. – Дата перегляду: 28.03.2020.

10. Бубнов И. 5 редакторов кода для JavaScript [Електронний ресурс] / Илья Бубнов. – Електрон. дані. – Режим доступу: https://geekbrains.ru/posts/javascript_editors/. – Назва з екрану. – Дата публікації: 05.07.2017. – Дата перегляду: 28.03.2020.

11. Шинкевич А. Война текстовых редакторов: редактор кода vs IDE [Електронний ресурс] / Александра Шинкевич. – Електрон. дані. – Режим доступу: https://wsd.events/2018/12/01/pres/code-editors/#cover/. – Назва з екрану. – Дата публікації: 01.12.2018. – Дата перегляду: 28.03.2020.

12. Yusov K. Top 15 JavaScript IDEs and JS Editors for Frontend Development [Електронний ресурс] / Kirill Yusov. – Електрон. дані. – Режим доступу: https://jelvix.com/blog/best-javascript-ides/. – Назва з екрану. – Дата публікації: 15.01.2020. – Дата перегляду: 29.03.2020.

13. Chuba N. npm vs Yarn – какой менеджер пакетов стоит использовать? [Електронний ресурс] / Nazar Chuba. – Електрон. дані. – Режим доступу: https://ua-blog.com/npm-vs-yarn-npm-vs-yarn-какой-менеджер-пакетов-стоит-испол/. – Назва з екрану. – Дата публікації: 29.03.2018. – Дата перегляду: 30.03.2020.

14. Severien T. Yarn vs npm: Everything You Need to Know [Електронний ресурс] / Tim Severien. – Електрон. дані. – Режим доступу: http://prgssr.ru/development/yarn-ili-npm-vse-chto-vam-nuzhno-znat.html/. – Назва з екрану. – Дата публікації: 20.10.2016. – Дата перегляду: 30.03.2020.

15. Фурдак В. Как стать full stack разработчиком, зная back-end. Пошаговая инструкция [Електронний ресурс] / Владислав Фурдак. – Електрон. дані. – Режим доступу: https://dou.ua/lenta/articles/how-to-become-full-stack-developer/. – Назва з екрану. – Дата публікації: 27.03.2019. – Дата перегляду: 31.03.2020.

16. Potter J. npm vs yarn [Електронний ресурс] / John Potter. – Електрон. дані. – Режим доступу: https://www.npmtrends.com/npm-vs-yarn/. – Назва з екрану. – Дата публікації: 12.05.2019. – Дата перегляду: 31.03.2020.

17. Введение в JSX [Електронний ресурс]. – Електрон. дані. – Режим доступу: https://thewebland.net/development/javascript/reactjs/introducing-jsx/. – Дата публікації: 06.11.2017. – Дата перегляду: 01.04.2020.

18. Redux-in-russian [Електронний ресурс]: оригинальная документация по Redux с переводом на руський. – Електрон. дані. – Режим доступу: https://rajdee.gitbooks.io/redux-in-russian/content/. – Дата публікації: 16.10.2018. – Дата перегляду: 03.04.2020.

19. Gagliardi V. React Redux Tutorial for Beginners [Електронний ресурс]: The Definitive Guide (2020) / Valentino Gagliardi. – Електрон. дані. – Режим доступу: https://www.valentinog.com/blog/redux/. – Дата публікації: 10.03.2020. – Дата перегляду: 03.04.2020.

20. Wieruch R. Redux vs MobX without Confusion [Електронний ресурс] / Robin Wieruch. – Електрон. дані. – Режим доступу: https://www.robinwieruch.de/redux-mobx/. – Дата публікації: 28.03.2017. – Дата перегляду: 03.04.2020.

21. Ceddia D. Redux vs The React Context API [Електронний ресурс] / Dave Ceddia. – Електрон. дані. – Режим доступу: https://daveceddia.com/context-api-vs-redux/. – Дата публікації: 17.07.2018. – Дата перегляду: 03.04.2020.

22. Jorgenson S. Get to Know React’s New Context API [Електронний ресурс] / Sarah Jorgenson. – Електрон. дані. – Режим доступу: https://scotch.io/tutorials/get-to-know-reacts-new-context-api/. – Дата публікації: 18.05.2018. – Дата перегляду: 03.04.2020.

23. Copes F. A beginner’s introduction to Webpack [Електронний ресурс] / Flavio Copes. – Електрон. дані. – Режим доступу: https://www.freecodecamp.org/news/a-beginners-introduction-to-webpack-2620415e46b3/. – Дата публікації: 13.06.2018. – Дата перегляду: 04.04.2020.

24. Nandan Singh A. An intro to Webpack: what it is and how to use it [Електронний ресурс] / Ashish Nandan Singh. – Електрон. дані. – Режим доступу: https://www.freecodecamp.org/news/an-intro-to-webpack-what-it-is-and-how-to-use-it-8304ecdc3c60/. – Дата публікації: 15.01.2019. – Дата перегляду: 04.04.2020.

25. Структура файлов [Електронний ресурс]: есть ли рекомендации по структуре React-проектов? / Документация. – Електрон. дані. – Режим доступу: https://ru.reactjs.org/docs/faq-structure.html/. – Дата публікації: 19.03.2020. – Дата перегляду: 05.04.2020.

26. Вишняков А. Лучшая архитектура для React проекта [Електронний ресурс] / Александр Вишняков. – Електрон. дані. – Режим доступу: https://blog.maddevs.io/лучшая-архитектура-для-react-проекта-2f6f1feedc13/. – Дата публікації: 14.03.2018. – Дата перегляду: 05.04.2020.

27. Marcus I. Easy React JS for Beginner Developers: A Step-By-step Visual Guide to Learn React Js and Building Your Own React Applications from Scratch / Ibas Marcus. – Independently Published, 2019. – 202 с.

28. Бэнкс А. React и Redux: функциональная веб-разработка / А. Бэнкс, Е. Порселло. – СПб.: Питер, 2018. – 336 с.

29. Тиленс Томас Марк. React в действии / Томас Марк Тиленс. – СПб.: Питер, 2019. – 368 с.

30. Ярлыков А. Инструменты Интернет-маркетинга [Електронний ресурс] / А. Ярлыков // Эффективные инструменты Интернет-маркетинга. – 2013. – Електрон. дані. – Режим доступу: http://takmak51.ru/

31. Плескач В.Л. Технології електронного бізнесу: Монографія / В.Л. Плескач. – К.: КНЕУ, 2004. – 223 с.

32. Вирин Федор. Интернет-маркетинг. Полный сборник практических инструментов / Федор Вирин. – М.: Эскмо, 2009. – 224 с.

33. Петрик Е.А. Интернет-маркетинг / Е.А. Петрик. – М.: Московская финансово-промышленная академия, 2004. – 299 с.

34. Успенский И.В. Интернет-маркетинг: Учебник [Електронний ресурс] / И.В. Успенский. – СПб.: Изд-во СПГУЭиФ, 2003. – Електрон. дані. Режим доступу: www.aup.ru/books/m80/

35. Литовченко І.Л. Інтернет-маркетинг: Навчальний посібник / І.Л. Литовченко, В.П. Пилипчук. – К.: Центр учбової літератури, 2008. – 184 с.

36. Павленко А.Ф. Маркетинг: Підручник / А.Ф. Павленко, А.В. Войчак. – К.: КНЕУ, 2003. – 246 с.

ДОДАТКИ

Додаток А

Публічні файли

public / index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>

    <meta charset="utf-8" />
    <meta
            name="viewport"
            content="width=device-width, initial-scale=1" />
    <link
            rel="stylesheet"
            href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"
            integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9
JvoRxT2MZw1T"
            crossorigin="anonymous">
    <link
            rel="stylesheet"
            href="https://use.fontawesome.com/releases/v5.8.2/css/all.css"
            integrity="sha384-oS3vJWv+0UjzBfQzYUhtDYW+Pj2yciDJxpsK1OYPAYjqT085Qq/1
cq5FLXAZQ7Ay"
            crossorigin="anonymous">
    <link
            href="https://fonts.googleapis.com/css?family=Playfair+Display:700" rel="stylesheet">

    <title>Re Store</title>

  </head>
  <body>
    <div id="root"></div>
  </body>
</html>
```

Додаток Б

Redux: actions та action creators

src / actions / index.html

```
const booksRequested = () => {
   return {
      type: 'FETCH_BOOKS_REQUEST'
   }
};

const booksLoaded = (newBooks) => {
   return {
      type: 'FETCH_BOOKS_SUCCESS',
      payload: newBooks
   }
};

const booksError = (error) => {
   return {
      type: 'FETCH_BOOKS_FAILURE',
      payload: error
   }
};

const fetchBooksOld = (dispatch, bookstoreService) => () => {
   dispatch(booksRequested());
   bookstoreService.getBooks()
      .then(data => dispatch(booksLoaded(data)))
      .catch(error => dispatch(booksError(error)))
};

const fetchBooks = (bookstoreService) => () => (dispatch) => {
   dispatch(booksRequested());
   bookstoreService.getBooks()
      .then(data => dispatch(booksLoaded(data)))
      .catch(error => dispatch(booksError(error)))
};

const bookAddedToCart = (id) => {
   return {
      type: 'BOOK_ADDED_TO_CART',
      payload: id
   }
};

const bookRemovedFromCart = (id) => {
   return {
      type: 'BOOK_REMOVED_FROM_CART',
      payload: id
   }
};

const allBooksRemovedFromCart = (id) => {
   return {
      type: 'ALL_BOOKS_REMOVED_FROM_CART',
      payload: id
   }
};
```

```
export {
    fetchBooks,
    bookAddedToCart,
    bookRemovedFromCart,
    allBooksRemovedFromCart
}
```

Додаток В

Redux: набір чистих функцій

src / reducers / index.js

```
import updateBookList from './update-book-list';
import updateShoppingCart from './update-shopping-cart';

const reducers = (state, action) => {
   return {
      bookList: updateBookList(state, action),
      shoppingCart: updateShoppingCart(state, action)
   };
};

export default reducers;
```

src / reducers / update-book-list.js

```
const updateBookList = (state, action) => {

   if (state === undefined) {
      return {
         books: [],
         loading: true,
         error: null,
      };
   }

   switch(action.type) {
      case 'FETCH_BOOKS_REQUEST':
         return {
            books: [],
            loading: true,
            error: null
         };
      case 'FETCH_BOOKS_SUCCESS':
         return {
            books: action.payload,
            loading: false,
            error: null
         };
      case 'FETCH_BOOKS_FAILURE':
         return {
            books: [],
            loading: false,
            error: action.payload
         };
      default:
         return state.bookList;
   }
};

export default updateBookList;
```

src / reducers / update-shopping-cart.js

```
const updateCartItems = (cartItems, item, i) => {
   if(item.count === 0) {
      return [
         ...cartItems.slice(0,i),
```

```
            ...cartItems.slice(i+1)
        ]
    }
    if (i<0) {
        return [
            ...cartItems,
            item
        ]
    } else {
        return [
            ...cartItems.slice(0,i),
            item,
            ...cartItems.slice(i+1)
        ]
    }
};
const updateItem = (book, item = {}, quantity) => {
    const {
        id = book.id,
        title = book.title,
        count = 0,
        total = 0
    } = item;
    return {
        id,
        title,
        count: count + quantity,
        total: total + quantity*book.price
    }
};
const updateOrder = (state, bookId, quantity) => {
    const {
        bookList: {books},
        shoppingCart: {cartItems: oldCartItems}
    } = state;

    const book = books.find(({id}) => bookId === id);
    const i = oldCartItems.findIndex(e => e.id === bookId);
    const item = oldCartItems[i];
    const newItem = updateItem(book, item, quantity);
    const newCartItems = updateCartItems(oldCartItems, newItem, i);
    return {
        cartItemsCount: newCartItems.reduce((ac,e) => ac + e.count, 0),
        orderTotal: state.shoppingCart.orderTotal + quantity * book.price,
        cartItems: newCartItems
    };
};


const updateShoppingCart = (state, action) => {

    if (state === undefined) {
        return {
            cartItems: [],
            orderTotal: 0,
            cartItemsCount: 0
        };
    }

    switch (action.type) {
        case 'BOOK_ADDED_TO_CART':
            return updateOrder(state, action.payload, 1);
```

```
        case 'BOOK_REMOVED_FROM_CART':
            return updateOrder(state, action.payload, -1);

        case 'ALL_BOOKS_REMOVED_FROM_CART':
            const item = state.shoppingCart.cartItems.find(({id}) => id === action.payload);
            return updateOrder(state, action.payload, -item.count);
        default: return state.shoppingCart;
    }
};

export default updateShoppingCart;
```

Додаток Г

Redux: store

src / store.js

```
import {createStore, applyMiddleware} from 'redux';
import thunkMiddlware from 'redux-thunk';
import reducer from './reducers';

const logMiddleware = (store) => (next) => (action) => {
   return next(action)
};
const stringMiddleware = (store) => (next) => (action) => {
   if (typeof action === 'string') {
      return next({
         type: action
      })
   }
   return next(action)
};

const store = createStore(reducer, applyMiddleware(
   thunkMiddlware,
   stringMiddleware,
   logMiddleware
));

const myAction = (dispatch) => {
   setTimeout(
      () => dispatch({type: 'DELAYED_ACTION'}),
      2000
   )
};

const delayedActionCreator = (timer) => (dispatch) => {
   setTimeout(
      () => dispatch({type: 'DELAYED_ACTION'}),
      timer
   )
};

store.dispatch(delayedActionCreator(5000));
export default store;
```

Додаток Д

Компоненти React-додатку

Компонент app

src / components / app / index.js

```
import App from './app';

export default App;
```

src / components / app / app.js

```
import React from 'react';
import {Route, Switch} from 'react-router-dom';
import './app.scss';

import {HomePage, CartPage} from '../pages';
import Header from '../header';


const App = () => {
  return (
      <main className="container">
         <Header/>
         <Switch>
            <Route path="/" exact component={HomePage}/>
            <Route path="/cart" exact component={CartPage}/>
            <Route render={() => <h2>Default</h2>}/>
         </Switch>
      </main>
  );

};


export default App;
```

Компонент book-list

src / components / book-list / index.js

```
import BookList from './book-list';

export default BookList;
```

src / components / book-list / book-list.js

```
import React, {Component} from 'react';
import {connect} from "react-redux";
import './book-list.scss';

import BookListItem from '../book-list-item';
import {withBookstoreService} from '../hoc';
import {fetchBooks, bookAddedToCart} from '../../actions';
import {compose} from '../../utils';
import Spinner from '../spinner';
import ErrorIndicator from '../error-indicator';


const BookList = ({books, onAddedToCart}) => {
    return (
```

```
        <div className="book-list row">
            {
                books.map(book => {
                    return (
                        <div className="list-item book-list__list-item col-md-6 col-lg-4" key={book.id}>
                            <BookListItem
                                book={book}
                                onAddedToCart={() => onAddedToCart(book.id)}/>
                        </div>
                    )
                })
            }
        </div>
    )
};


class BookListContainer extends Component {

    componentDidMount() {
        this.props.fetchBooks();
    }

    render() {
        const {books, loading, error, onAddedToCart} = this.props;
        if (loading) return <Spinner/>;
        if (error) return <ErrorIndicator/>;

        return <BookList books={books} onAddedToCart={onAddedToCart}/>
    }
}

const mapStateToProps = ({bookList: {books, loading, error}}) => {
    return {
        books,
        loading,
        error
    }
};

const mapDispatchToProps = (dispatch, ownProps) => {
    const {bookstoreService} = ownProps;
    return {
        fetchBooks: (id) => dispatch(fetchBooks(bookstoreService)()),
        onAddedToCart: (id) => dispatch(bookAddedToCart(id))
    }
};

export default compose(
    withBookstoreService(),
    connect(mapStateToProps, mapDispatchToProps)
)(BookListContainer);
```

src / components / book-list / book-list.scss

```
.list-item {
  min-width: 270px;
  flex-grow: 1;
  flex-shrink: 1;
  padding-bottom: 10px;
  padding-right: 10px;
}
```

Компонент book-list-item

src / components / book-list-item / index.js

```
import BookListItem from './book-list-item';

export default BookListItem;
```

src / components / book-list-item / book-list-item.js

```
import React from 'react';
import './book-list-item.scss';

const BookListItem  = ({book, onAddedToCart}) => {
   const {author, title, price, coverImage} = book;
   return (
      <div className="book-list-item d-flex">
         <div className="book-cover book-list-item__book-cover ">
            <img src={coverImage} alt="" className="book-cover__img"/>
         </div>
         <div className="book-details">
            <span className="book-title">{title}</span>
            <div className="book-author">{author}</div>
            <div className="book-price">{`$${price}`}</div>
            <button
               className="btn btn-info btn-add-to-cart book-details__btn-add-to-cart"
               onClick={onAddedToCart}>
               Add to cart
            </button>
         </div>
      </div>
   );
};


export default BookListItem;
```

src / components / book-list-item / book-list-item.scss

```
.book-list-item {
  &__book-cover {
    margin-right: 20px;
  }
}

.book-cover {
  width: 120px;
  flex-shrink: 0;

  &__img {
    max-width: 100%;
  }
}

.book-title {
  font-size: 1.2rem;
}

.book-price {
  font-size: 1.4rem;
  font-weight: bold;
```

```
}

.book-details {
  margin-bottom: 10px;
  position: relative;

  &__btn-add-to-cart {
    position: absolute;
    bottom: 0;
    left: 0;
  }
}
```

Компонент bookstore-service-context

src / components / bookstore-service-context / index.js

```
import { BookstoreServiceProvider, BookstoreServiceConsumer} from './bookstore-service-context';

export {
    BookstoreServiceProvider,
    BookstoreServiceConsumer
}
```

src / components / bookstore-service-context / bookstore-service-context.js

```
import React from 'react';

const {
    Provider: BookstoreServiceProvider,
    Consumer: BookstoreServiceConsumer
} = React.createContext();

export {
    BookstoreServiceProvider,
    BookstoreServiceConsumer
}
```

Компонент error-boundary

src / components / error-boundary / index.js

```
import ErrorBoundary from './error-boundary';

export default ErrorBoundary;
```

src / components / error-boundary / error-boundary.js

```
import React, {Component} from 'react';
import ErrorIndicator from '../error-indicator';

export default class ErrorBoundary extends Component {

    state = {
        error: false
    };

    componentDidCatch(error) {
        this.setState({error: true})
    }

    render() {
        const {error} = this.state;
        if (error) return <ErrorIndicator/>
```

```
        return this.props.children;
    }
};
```

Компонент error-indicator

src / components / error-indicator / index.js

```
import ErrorIndicator from './error-indicator';

export default ErrorIndicator;
```

src / components / error-indicator / error-indicator.js

```
import React from 'react';
import './error-indicator.scss';

const ErrorIndicator = () => {
    return (
        <div>Error!!</div>
    )
};

export default ErrorIndicator;
```

Компонент header

src / components / header / index.js

```
import Header from './header';

export default Header;
```

src / components / header / header.js

```
import React from 'react';
import {Link} from "react-router-dom";
import {connect} from 'react-redux';
import './header.scss';

const Header = ({cartItemsCount}) => {

    const count = cartItemsCount? <span className="shopping-cart__count">{cartItemsCount}</span> : null;

    return (
        <header className="row">
            <div className="col-md-12">
                <div className=" header d-flex justify-content-between align-items-center">
                    <Link to='/'>
                        <div href="#" className="logo text-dark">ReStore</div>
                    </Link>
                    <Link to="/cart">
                        <div className="shopping-cart header__shopping-cart">
                            <i className="cart-icon fa fa-shopping-cart shopping-cart__cart-icon"/>
                            { count }
                        </div>
                    </Link>
                </div>
            </div>
        </header>
    );
};

const mapStateToProps = ({shoppingCart: {cartItemsCount}}) => {
```

```
    return {
        cartItemsCount
    }
}

export default connect(mapStateToProps)(Header);
```

src / components / header / header.scss

```
.header {
  border-bottom: 1px solid #ccc;
  margin-bottom: 2rem;

  & a:hover {
    text-decoration: none;
  }

  &__shopping-cart {
    position: relative;
  }
}

.logo {
  font-size: 2rem;
  font-weight: bold;
  font-family: 'Playfair Display', Georgia,  serif;

  &:hover {
    text-decoration: none;
  }
}

.shopping-cart {
  &__cart-icon {
    margin-right: 10px;
  }

  &:hover {
    text-decoration: none;
  }

  &__count {
    position: absolute;
    width: 22px;
    height: 22px;
    border-radius: 50%;
    background: #fdb64e;
    bottom: 0;
    right: 0;
    transform: translate(30%,20%);
    text-align: center;
    font-size: 0.75rem;
    line-height: 1.4rem;
    color: cadetblue;
  }
}

.cart-icon {
  font-size: 1.5rem;
  color: cadetblue;
}
```

Компонент hoc

src / components / hoc / index.js

```
import withBookstoreService from './with-bookstore-service';

export {
   withBookstoreService
}
```

src / components / hoc / with-bookstore-service.js

```
import React from 'react';
import {BookstoreServiceConsumer} from '../bookstore-service-context';

const withBookstoreService = (mapMethodsToProps) => (View) => {
   return (props) => {
      return (
         <BookstoreServiceConsumer>
            {
               (BookstoreService) => {
                  return <View
                     {...props}
                     bookstoreService= {BookstoreService}/>
               }
            }
         </BookstoreServiceConsumer>
      )
   }
};

export default withBookstoreService;
```

Компонент pages

src / components / pages / index.js

```
import HomePage from './home-page';
import CartPage from './cart-page';


export {
   HomePage,
   CartPage
}
```

src / components / pages / cart-page.js

```
import React, {Component} from 'react';
import ShoppingCartTable from "../shopping-cart-table";

class CartPage extends Component {
   render() {
      return (
         <div className="cart-page row">
            <div className="col-md-12">
               <ShoppingCartTable/>
            </div>
         </div>
      );
   }
}

export default CartPage;
```

src / components / pages / home-page.js

```
import React, {Component} from 'react';

import BookList from '../book-list';
import ShoppingCartTable from "../shopping-cart-table";

class HomePage extends Component {
   render() {
      return (
         <div className="home-page">
            <BookList/>
         </div>
      );
   }
}

export default HomePage;
```

Компонент shopping-cart-table

src / components / shopping-cart-table / index.js

```
import ShoppingCartTable from './shopping-cart-table';

export default ShoppingCartTable;
```

src / components / shopping-cart-table / shopping-cart-table.js

```
import React from 'react';
import {connect} from "react-redux";
import './shopping-cart-table.scss';

import {bookRemovedFromCart, allBooksRemovedFromCart, bookAddedToCart} from '../../actions';

const ShoppingCartTable = ({items, total, onIncrease, onDecrease, onDelete}) => {

   const renderRow = (item, i) => {
      const {id, title, count, total} = item;
      return (
         <tr key={id}>
            <td>{i+1}</td>
            <td>{title}</td>
            <td>{count}</td>
            <td>${total}</td>
            <td>
               <button
                  className="btn btn-outline-danger btn-sm float-right"
                  onClick={() => onDelete(id)}>
                  <i className="fa fa-trash"/>
               </button>
               <button
                  className="btn btn-outline-success btn-sm float-right"
                  onClick={() => onIncrease(id)}>
                  <i className="fa fa-plus-circle" />
               </button>
               <button
                  className="btn btn-outline-warning btn-sm float-right"
                  onClick={() => onDecrease(id)}>
                  <i className="fa fa-minus-circle" />
               </button>
            </td>
         </tr>
      )
```

```
    };

    return (
        <div className="shopping-cart-table">
            <h4>Your order</h4>
            <table className="table">
                <thead>
                    <tr>
                        <th>#</th>
                        <th>Item</th>
                        <th>Count</th>
                        <th>Total price</th>
                        <th>Action</th>
                    </tr>
                </thead>
                <tbody>
                    {
                        items.map(renderRow)
                    }
                </tbody>
            </table>
            <div className="shopping-cart-table__total">Total: ${total}</div>
        </div>
    );
};

const mapStateToProps = ({shoppingCart: {cartItems, orderTotal}}) => {
    return {
        items: cartItems,
        total: orderTotal
    }
};

const mapDispatchToProps = {
    onIncrease: bookAddedToCart,
    onDecrease: bookRemovedFromCart,
    onDelete: allBooksRemovedFromCart
};

export default connect(mapStateToProps, mapDispatchToProps)(ShoppingCartTable);
```

src / components / shopping-cart-table / shopping-cart-table.scss

```
.shopping-cart-table {
  &__total {
    text-align: right;
    font-size: 1.3rem;
    font-weight: bold;
  }

  & .btn:not(:first-child) {
    margin-right: 5px;
  }
}
```

Компонент spinner

src / components / spinner / index.js

```
import Spinner from './spinner';

export default Spinner;
```

src / components / spinner / spinner.js

```
import React from 'react';
import './spinner.scss';

const Spinner = () => {
   return (
      <div className="lds-css ng-scope">
         <div className="lds-spinner"  style={{width:'100%', height:'100%'}}>
            <div></div>
            <div></div>
            <div></div>
            <div></div>
            <div></div>
            <div></div>
            <div></div>
            <div></div>
            <div></div>
            <div></div>
            <div></div>
            <div></div>
         </div>
      </div>
   );
};

export default Spinner;
```

src / components / spinner / spinner.scss

```
@keyframes lds-spinner {
  0% {
   opacity: 1;
  }
  100% {
   opacity: 0;
  }
}
@-webkit-keyframes lds-spinner {
  0% {
    opacity: 1;
  }
  100% {
    opacity: 0;
  }
}
.lds-spinner {
  position: relative;
}
.lds-spinner div {
  left: 94px;
  top: 48px;
  position: absolute;
  -webkit-animation: lds-spinner linear 1s infinite;
  animation: lds-spinner linear 1s infinite;
  background: #17a2b8;
  width: 12px;
  height: 24px;
  border-radius: 40%;
  -webkit-transform-origin: 6px 52px;
  transform-origin: 6px 52px;
}
.lds-spinner div:nth-child(1) {
  -webkit-transform: rotate(0deg);
```

```
  transform: rotate(0deg);
  -webkit-animation-delay: -0.916666666666667s;
  animation-delay: -0.916666666666667s;
}
.lds-spinner div:nth-child(2) {
  -webkit-transform: rotate(30deg);
  transform: rotate(30deg);
  -webkit-animation-delay: -0.833333333333333s;
  animation-delay: -0.833333333333333s;
}
.lds-spinner div:nth-child(3) {
  -webkit-transform: rotate(60deg);
  transform: rotate(60deg);
  -webkit-animation-delay: -0.75s;
  animation-delay: -0.75s;
}
.lds-spinner div:nth-child(4) {
  -webkit-transform: rotate(90deg);
  transform: rotate(90deg);
  -webkit-animation-delay: -0.666666666666667s;
  animation-delay: -0.666666666666667s;
}
.lds-spinner div:nth-child(5) {
  -webkit-transform: rotate(120deg);
  transform: rotate(120deg);
  -webkit-animation-delay: -0.583333333333333s;
  animation-delay: -0.583333333333333s;
}
.lds-spinner div:nth-child(6) {
  -webkit-transform: rotate(150deg);
  transform: rotate(150deg);
  -webkit-animation-delay: -0.5s;
  animation-delay: -0.5s;
}
.lds-spinner div:nth-child(7) {
  -webkit-transform: rotate(180deg);
  transform: rotate(180deg);
  -webkit-animation-delay: -0.416666666666667s;
  animation-delay: -0.416666666666667s;
}
.lds-spinner div:nth-child(8) {
  -webkit-transform: rotate(210deg);
  transform: rotate(210deg);
  -webkit-animation-delay: -0.333333333333333s;
  animation-delay: -0.333333333333333s;
}
.lds-spinner div:nth-child(9) {
  -webkit-transform: rotate(240deg);
  transform: rotate(240deg);
  -webkit-animation-delay: -0.25s;
  animation-delay: -0.25s;
}
.lds-spinner div:nth-child(10) {
  -webkit-transform: rotate(270deg);
  transform: rotate(270deg);
  -webkit-animation-delay: -0.166666666666667s;
  animation-delay: -0.166666666666667s;
}
.lds-spinner div:nth-child(11) {
  -webkit-transform: rotate(300deg);
  transform: rotate(300deg);
  -webkit-animation-delay: -0.083333333333333s;
  animation-delay: -0.083333333333333s;
```

```
}
.lds-spinner div:nth-child(12) {
  -webkit-transform: rotate(330deg);
  transform: rotate(330deg);
  -webkit-animation-delay: 0s;
  animation-delay: 0s;
}
.lds-spinner {
  width: 46px !important;
  height: 46px !important;
  -webkit-transform: translate(-23px, -23px) scale(0.23) translate(23px, 23px);
  transform: translate(-23px, -23px) scale(0.23) translate(23px, 23px);
}
```

Додаток Е

Сервіси

src / services / bookstore-service.js

```
export default class BookstoreService {

   data = [
      {
         id: 1,
         title: 'One Indian Girl',
         author: 'Chetan Bhagat',
         price: 100,
         coverImage: 'https://images-eu.ssl-images-amazon.com/images/I/51btCZ-13mL.jpg'
      },{
         id: 2,
         title: 'Life is What You Make It',
         author: 'Preeti Shenoy',
         price: 77,
         coverImage: 'https://images-na.ssl-images-
amazon.com/images/I/41k9HxNTJQL._SX311_BO1,204,203,200_.jpg'
      },{
         id: 3,
         title: 'I Still Think About You',
         author: 'Arpit Vageria',
         price: 34,
         coverImage: 'https://images-eu.ssl-images-amazon.com/images/I/51RBC5mLueL.jpg'
      },{
         id: 4,
         title: 'The Girl in Room 105',
         author: 'Chetan Bhagat',
         price: 65,
         coverImage: 'https://images-eu.ssl-images-amazon.com/images/I/413Ge3dy3qL.jpg'
      },{
         id: 5,
         title: 'Be My Perfect Ending',
         author: 'Arpit Vageria',
         price: 90,
         coverImage: 'https://images-eu.ssl-images-amazon.com/images/I/51K1inn6%2BqL.jpg'
      }
   ];

   getBooks() {
      return new Promise((resolve, reject) => {
         setTimeout(() => {
            if (Math.random()>0) {
               resolve(this.data)
            } else {
               reject(new Error('Something was happen!'))
            }
         }, 800)
      })
   }
}
```

Додаток Ж

Утиліти та допоміжні функції

src / utils/ index.js

```
import compose from './compose';

export {
   compose
}
```

src / utils/ store.js

```
const compose = (...funcs) => (component) => funcs.reduceRight((ac,e) => e(ac), component);

export default compose;
```

Додаток И

Точка входу JavaScript

src / index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import {Provider} from 'react-redux';
import {BrowserRouter as Router} from "react-router-dom";

import App from './components/app';
import ErrorBoundary from './components/error-boundary';
import BookstoreService from './services/bookstore-service';
import {BookstoreServiceProvider} from "./components/bookstore-service-context";

import store from './store';



ReactDOM.render(
   <Provider store={store}>
      <ErrorBoundary>
         <BookstoreServiceProvider value={new BookstoreService()}>
            <Router>
               <App />
            </Router>
         </BookstoreServiceProvider>
      </ErrorBoundary>
   </Provider>,
   document.getElementById('root')
);
```

Додаток К

Файл конфігурації проекту

package.json

```
{
  "name": "re-store",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "node-sass": "^4.12.0",
    "prop-types": "^15.7.2",
    "react": "^16.8.6",
    "react-dom": "^16.8.6",
    "react-redux": "^7.0.3",
    "react-router-dom": "^5.0.0",
    "react-scripts": "^3.4.1",
    "redux": "^4.0.1",
    "redux-thunk": "^2.3.0"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": "react-app"
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

Додаток Л

Набір правил для приховування файлів і папок

від системи контролю версій Git

.gitignore

```
# See https://help.github.com/articles/ignoring-files/ for more about ignoring files.

# dependencies
/node_modules
/.pnp
.pnp.js

# testing
/coverage

# production
/build

# misc
.DS_Store
.env.local
.env.development.local
.env.test.local
.env.production.local

npm-debug.log*
yarn-debug.log*
yarn-error.log*
```