МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Кафедра Комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
Аліна САВЧЕНКО
«_____» _____ 2024р.

КВАЛІФІКАЦІЙНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ЗДОБУВАЧА ОСВІТНЬОГО СТУПЕНЯ **"БАКАЛАВР"**

Тема: «Вебсервіс пошуку книг та додавання їх до категоріальних полиць**»**

Виконавець: студент групи УС-413 Дремлюга Владислав Борисович

Керівник: д.т.н., професор Зіатдінов Юрій Кашафович

Нормоконтролер: Олександр ШЕВЧЕНКО

Київ – 2024

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних наук та технологій*

Кафедра *Комп'ютерних інформаційних технологій*

Галузь знань, спеціальність, освітньо-професійна програма: 12 "Інформаційні технології", 122 "Комп'ютерні науки", "Інформаційні управляючі системи та технології"

ЗАТВЕРДЖУЮ
Завідувач випускової кафедри
_____ Аліна САВЧЕНКО
«____»_____2024р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи (проєкту)

Дремлюги Владислав Борисовича
(прізвище, ім'я, по батькові)

1. Тема роботи: «Вебсервіс пошуку книг та додавання їх до категоріальних полиць» затверджена наказом ректора від «5» квітня 2024 р. №517/ст.

2. Термін виконання роботи: 06.05.2023 – 09.06.2024

3. Вихідні дані до роботи: технічні, функціональні, а також вимоги щодо користувацького досвіду та інтерфейсу; безпека даних користувачів.

4. Зміст пояснювальної записки: вступ, аналіз проблематики створення та дослідження вебсервісу пошуку книг та додавання їх до категоріальних полиць, використані технології та методи розробки, реалізація.

5. Перелік обов'язкового графічного матеріалу: ключові вимоги та потреби користувачів; клієнт-серверна архітектура; модель «товстого» і «тонкого» клієнтів; інструменти як React, Google Books API, VS Code, GitHub, Netlify; кореневі файли проєкту; структурні схеми будови сторінок.

6. Календарний план-графік

№ п/п	Завдання	Термін виконання	Підпис керівника
1.	Отримання завдання на дипломну роботу, створення плану дипломної роботи та побудова плану-графіку виконання робіт.	03.04.2024 – 09.04.2024	
2.	Розроблення та затвердження календарного плану виконання дипломної роботи.	10.04.2024 – 14.04.2024	
3.	Проведення консультацій з науковим керівником.	15.04.2024 – 19.04.2024	
4.	Огляд та аналіз наукової літератури по темі дипломної роботи та написання Розділу 1.	20.04.2024 – 03.05.2024	
5.	Написання Розділу 2 дипломної роботи.	03.05.2024 – 08.05.2024	
6.	Написання Розділу 3 і Розділу 4 дипломної роботи. Завершення створення пояснювальної записки дипломної роботи.	09.05.2024 – 25.05.2024	
7.	Оформлення та друк пояснювальної записки.	26.05.2024 – 29.05.2024	
8.	Створення презентації, доповіді та підготовка до захисту дипломної роботи.	30.05.2024 – 04.06.2024	
9.	Підготовка матеріалів дипломної роботи для передачі секретарю ДЕК (папка, конверт, диск із файлом диплому, рецензія, відгук).	05.06.2024 – 07.06.2024	

7. Дата видачі завдання: «06» травня 2024 р.

Керівник дипломної роботи _____ Юрій ЗІАТДІНОВ
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Владислав ДРЕМЛЮГА
(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Вебсервіс пошуку книг та додавання їх до категоріальних полиць»: 78 с., 29 рис., 7 літературних джерел.

Об’єкт дослідження: процес створення та функціонування вебсервісу з пошуку книг та додавання їх до категоріальних полиць.

Мета роботи: розробити вебсервіс з пошуку книг та додавання їх до категоріальних полиць.

Методи дослідження: функціональне тестування, порівняльний аналіз, обробка літератури, аналіз даних, UX-тестування.

Отримані результати та їх новизна: використання сучасних підходів до UX/UI для підвищення зручності користувачів; дизайн вебсервісу забезпечує зручну навігацію та легкість у використанні; впроваджено кешування результатів для зниження навантаження на сервер; реалізовано захист даних користувачів через безпечну автентифікацію та авторизацію; сервіс адаптований для використання на різних пристроях (мобільних, планшетах, десктопах) та легко масштабується для підтримки великої кількості користувачів.

Результати кваліфікаційної роботи рекомендується: адаптувати для комерційних платформ з продажу книг, застосовувати в онлайн-спільнотах любителів книг, використовувати як основу для подальших досліджень у галузі веброзробки та інформаційних систем.

ВЕБСЕРВІС, ПОШУК КНИГ, БАЗА ДАНИХ, REACT, API, КОМПОНЕНТИ, HTTP-ЗАПИТИ, АВТОРИЗАЦІЯ, КЕШУВАННЯ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

ПП	Програмний Продукт
API	**Application Interface -** набір правил і протоколів, що дозволяють різним програмам взаємодіяти між собою. API визначає методи, які одна програма може використовувати для запиту даних або послуг у іншої програми.
HTML	**HyperText Markup Language** — стандартна мова розмітки, яка використовується для створення та структурування вебсторінок. HTML описує структуру вебсторінки за допомогою елементів, таких як заголовки, параграфи, посилання, зображення та інші елементи контенту.
CSS	**Cascading Style Sheets —** мова стилів, яка використовується для задання вигляду та оформлення вебсторінок, що написані мовою HTML.
JS	**JavaScript** – мова програмування для створення інтерактивних веб-сторінок
Пакет	Розширення або додаток, який додає певні функції або можливості до редактора коду. Це може бути набір інструментів, тема оформлення, розширення для роботи з певними мовами програмування або будь-яке інше розширення, яке розширює функціональність VS Code.
Стан	Об'єкт, який містить дані, які можуть змінюватися в процесі роботи додатку. Використовуючи стан, компоненти React можуть відслідковувати зміни даних та автоматично оновлювати свій вміст відповідно до цих змін, що дозволяє створювати інтерактивні та динамічні веб-інтерфейси.

ВСТУП

Загальна доступність інформації завдяки Інтернету перетворила наше суспільство, забезпечуючи нам безмежний доступ до знань, розваг та можливостей. Це відкриває широкі горизонти для інновацій та розвитку, а також створює підґрунтя для постійного удосконалення та розвитку нових інструментів і сервісів. У цьому контексті створення Вебсервісу для пошуку книг та їх організації на категоріальних полицях стає не лише актуальною, але й необхідною складовою інтернет-екосистеми.

Метою обрання створення саме цього Вебсервісу є можливість задоволення потреб користувачів у зручному та ефективному інструменті для пошуку та організації книг. Він поєднує в собі потужність сучасних веб-технологій з інтуїтивно зрозумілим інтерфейсом, що дозволяє швидко та зручно знаходити необхідну літературу та впорядковувати її відповідно до власних потреб та вподобань. Іншими словами, він допомагає користувачам створювати власні віртуальні бібліотеки, що відображають їхні інтереси та уподобання. Такий підхід сприяє розвитку культури читання та саморозвитку.

Об'єктом дослідження є процес створення Вебсервісу з пошуку книг та їх додавання до категоріальних полиць. Цей процес включає аналіз проблем, з якими можуть зіштовхнутися розробники під час створення сервісу, а також огляд існуючих недоліків і пропозицій щодо удосконалення підходів, методів та алгоритмів для досягнення оптимальних результатів. У подальших розділах дослідження будуть розглянуті ці аспекти більш детально, з метою надання практичних рекомендацій та розв'язання виявлених проблем.

РОЗДІЛ 1
АНАЛІЗ ПРОБЛЕМАТИКИ СТВОРЕННЯ ТА ДОСЛІДЖЕННЯ ВЕБСЕРВІСУ ПОШУКУ КНИГ ТА ДОДАВАННЯ ЇХ ДО КАТЕГОРІАЛЬНИХ ПОЛИЦЬ

1.1. Визначення проблем та викликів у процесі створення вебсервісу для пошуку книг

Книги, з давніх часів, стали невід'ємною частиною людської культури та цивілізації. Вони є носіями знань, історій та думок, передаваних з покоління в покоління. Від ан121тичних рукописів до сучасних видань, книги відображають різноманітні аспекти життя, мислення та вірувань різних епох і культур.

Цей дипломний проект присвячений загальному огляду проблематики читання не лише серед молоді, а й старших груп населення, огляду поточної ситуації в галузі вебсервісів для пошуку книг, а також концепції розробки та викликів, що стоять під час розробки даного програмного забезпечення.

Однак, з самого початку, перед самою розробкою програмного забезпечення, необхідно дослідити саму концепцію вебсервісів з пошуку книг, провести аналіз області застосування програмного продукту та його вплив на соціокультурне оточення.

1.2. Аналіз проблематики

Культура читання є дуже важливою частиною нашого суспільства. Йдеться не лише про читання книг, а й про читання статей та іншого навчального матеріалу.

Проте за останнє десятиліття чи близько того змінився, передусім, стиль нашого читання. Сучасний світ сповнений відволікаючих факторів, через які людям важко зосередитися на читанні книги чи статті, не перериваючи сповіщень на телефоні. Інтернет дав змогу людям отримувати інформацію одним натисканням кнопки, і це призвело до того, що сучасні покоління все менше звертають уваги до читання всього, що є недостатньо розважальним. Люди більше звикають гортати стрічки соціальних мереж, ніж сидіти з книгою чи газетою.

Любов до читання книг – особистий вибір. Але головна проблема в тому, що це не популярний особистий вибір у сучасному світі. Варто дослідити деякі з ключових проблем, які впливають на культуру читання в цілому:

- *Брак часу*. У наш час у людей немає часу сісти і почитати книгу. Вони завжди в дорозі, чи то їдуть на роботу, чи доглядають за дітьми вдома. У них багато справ, тому їм важко знайти час для читання.

- *Відсутність звички читати*. Читання завжди було важливою навичкою, яку діти розвивали під час дорослішання, оскільки це допомагає їм краще вчитися в школі, успішніше навчатися та рости освіченішими особистостями. Однак, коли звичка читати книжки не прищеплена - вони, як дорослі особистості, не відчувають зв’язку з книгою та загальною культурою читання.

- *Дороге хобі*. Звичайно, книги – недешеве задоволення. Вони займають багато місця, можуть коштувати дорого й потребують місце для зберігання. Для деяких людей вони варті витрачених коштів, але для інших - це річ, яку вони не можуть собі дозволити. Взяти книги напрокат в бібліотеці – доцільний вихід із ситуації. На мою думку, рішення сходити в бібліотеку досить виправдане для мети заохочення молоді, й не тільки, до читання – саме перебування в атмосфері біля полиць з різноманітними книгами й можливість безкоштовно доторкнутись до цікавої й корисної інформації – вирішує головну проблему, адже схиляє й прищеплює людям цікавість до читання та пізнання чогось нового.

Виходячи з вищезазначених аспектів, було обрано тематику створення вебсервісу, що, одночасно, був би естетичним та продуктивним. Це – вебсервіс для пошуку книг та додавання їх до категоріальних полиць.

1.3. Виявлення ключових вимог та потреб користувачів у вебсервісі з пошуку та організації книг

Виявлення ключових вимог та потреб користувачів у вебсервісі з пошуку та організації книг є важливим етапом у процесі розробки продукту. Для досягнення успіху у цій сфері потрібно ретельно дослідити та зрозуміти потреби користувачів. Ось деякі ключові аспекти, які можуть виявитися важливими у цьому контексті:

- Зручний пошук книг. Користувачі очікують від вебсервісу швидкого та ефективного пошуку книг. Пошук повинен бути інтуїтивно зрозумілим та забезпечувати релевантні результати.

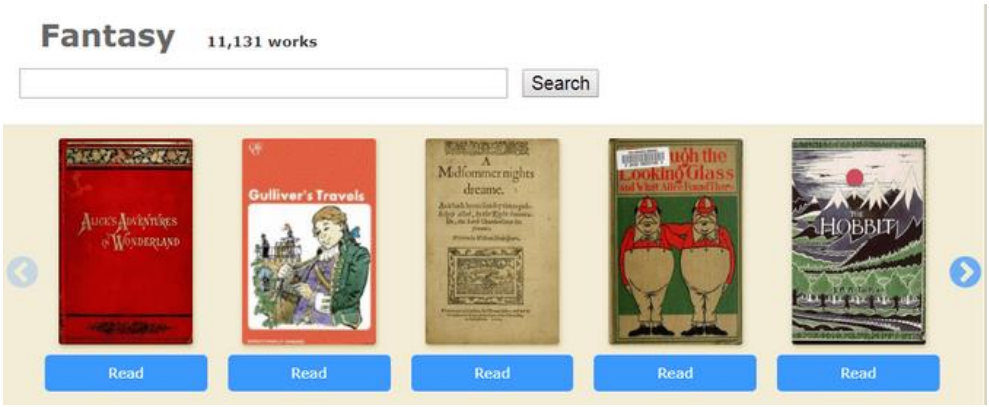

Рис. 1.1. Сторінка з пошуком та результатом пошуку книг

- Додавання до категоріальних полиць. Користувачі хочуть мати можливість організувати свою колекцію книг за різними категоріями або полицями. Це може включати улюблені книги, прочитані книги, книги для майбутнього читання тощо.

Рис. 1.2. Модель книжної полиці

- Можливість безпосереднього читання. Ця вимога відповідає потребам користувачів у зручному доступі до вмісту книг без необхідності завантажувати або купувати їх. Можливість безпосереднього читання книги може бути реалізована за допомогою онлайн-читалок або вбудованих переглядачів, які дозволяють читати книги прямо у веб-браузері без необхідності завантажувати додаткове програмне забезпечення або покупати книги.

Рис. 1.3. Приклад кнопки для взаємодії з користувачем

- Зручний інтерфейс користувача. Вебсервіс повинен мати зрозумілий та привабливий інтерфейс, який дозволяє легко навігувати та використовувати всі функціональні можливості сервісу.

Рис. 1.4. Модель естетичного кольорового інтерфейсу

- Мобільна сумісність. Оскільки багато користувачів використовують мобільні пристрої для доступу до інтернету, важливо забезпечити сумісність вебсервісу з мобільними пристроями та оптимізацію для різних розмірів екранів;

Рис. 1.5. Доступ до сервісу з мобільних пристроїв

1.4. Аналіз можливих шляхів удосконалення підходів, методів та алгоритмів створення вебсервісу

Насамперед, перед розробкою вебсервісу, варто замислитись і окреслити основні пріоритети, цілі, які треба мати на увазі та слідувати їм, адже навіть через

неправильну постановку задач та цілей в кінцевому результаті не буде отримано бажаного програмного продукту, який був би справді актуальним користувачеві.

1.4.1. Загальний опис цілей та пріоритетів, дослідження недоліків

Отож, необхідно окреслити, включити та реалізувати такі цілі під час аналізу та розробки вебсервісу:

- **Естетичний інтерфейс**. Важливість красивого та естетичного інтерфейсу важко переоцінити. У даному проекті були прагнення створити естетичний, лаконічний та привабливий інтерфейс, що був би радше помічником та дороговказом користувачеві, аніж лише візуальною частиною. При окресленні цього критерію було взято до уваги інтерфейс альтернативних вебсервісів для пошуку книг та додавання їх до категоріальних полиць. Дивлячись на візуальну складову таких вебсервісів як Google Books, Amazon, Goodreads, LibraryThing та Bookfinder.com можна прийти до висновку, що варто реалізувати інтерфейс більш високого рівня – візуальна складова у перелічених вебсервісів є недоопрацьованою: занадто перевантажений інтерфейс з непотрібними складовими, які відвертають увагу нових користувачів при знайомстві з вебсервісом; сам дизайн відчувається як той, що з минулого. Іншими словами, інтерфейс, або візуальна складова даного вебсервісу повинна відповідати моді сучасного дизайну, бути не перевантаженою та товариською для користувача.
- **Велика база даних книг**. Дозволяє користувачам знаходити великий обсяг різноманітних книг та авторів. База даних великого обсягу повинна відповідати таким критеріям:

  - *Широкий вибір* - велика БД забезпечує користувачам широкий вибір книг для пошуку. Це означає, що користувачі можуть

знаходити не лише популярні книги, але і менш відомі твори, новинки, а також рідкісні або старовинні видання.

- *Різноманітність жанрів та авторів* - з великою БД можна охопити широкий спектр жанрів та авторів. Це важливо для того, щоб задовольнити різноманітні літературні смаки користувачів і дозволити користувачам знаходити книги, які відповідають їхнім інтересам.
- *Актуальність і повнота інформації* - БД повинна бути постійно оновлювана, щоб вона була актуальною та повною, тобто це - додавання нових видань книг, а також актуалізація інформації про наявність книг у різних магазинах чи бібліотеках.
- *Підтримка різних форматів та мов* - важливо мати БД, яка включає книги в різних форматах (паперові, електронні, аудіо) та мовах. Це забезпечить доступність книг для широкого кола користувачів з різних країн і культур.
- *Покриття різних видавництв* - велика база даних повинна включати книги від різних видавництв, щоб користувачі могли знаходити книги, незалежно від їхнього походження чи видавництва.

Виходячи з вищезазначеного, вибір зупинився на API від компанії Google (Google Books API). Google є світовим гігантом у сфері інформаційних технологій. Не дивно, що Google має одну з найбільших баз даних книг у світі.

Додатково, офіційний сайт з використання API для пошуку книг від Google має велику та чітку документацію та поради з використання та реалізації програмного продукту у супроводі з програмним інтерфейсом від цієї компанії, що є надзвичайно корисним розробнику при проектуванні та реалізації вебсервісу з пошуку книг та додавання їх до категоріальних полиць.

- **Інтеграція з обліковим записом Google**. Зручність авторизації через обліковий запис Google спрощує процес реєстрації та входу на

платформу для користувачів. Це є надзвичайно корисною функцією та життєво необхідною для сучасного вебсервісу не лише для пошуку книг, а й загалом для всіх них. Основні з переваг:

- *Безпека*. OAuth 2.0 надає механізм авторизації без передачі логінів і паролів між сторонами. Користувачі можуть надати застосункам/вебсервісам обмежений доступ до своїх облікових записів Google без ризику розголошення особистої інформації.

- *Зручність для користувачів*. Більшість користувачів вже мають облікові записи Google, тому вони можуть авторизуватися в додатках лише кількома кліками. Це дозволяє уникнути необхідності запам'ятовувати та вводити нові логіни та паролі для кожного окремого додатку.

- *Доступ до інших сервісів Google*. Після авторизації через обліковий запис Google, користувач може отримати доступ до інших сервісів Google, таких як Gmail, Google Drive, Google Calendar і т.д., без необхідності повторно вводити облікові дані.

- *Стандартність і популярність*. OAuth 2.0 є стандартом в галузі авторизації, тому більшість розробників і користувачів знайомі з його принципами. Це спрощує процес розробки та використання авторизації в додатках і забезпечує високий рівень довіри.

- *Керування дозволами*. OAuth 2.0 дозволяє користувачам контролювати, які дозволи надаються додатку під час авторизації. Це означає, що користувачі можуть обирати, яку інформацію вони хочуть надавати і яку не хочуть.

- **Адаптивний дизайн**. Серед головних цілей, що беруться до уваги, є те, що користувачі повинні мати можливість взаємодіяти з вебсервісом не лише через комп’ютер чи ноутбук, а й за допомогою смартфону чи планшету, адже у сучасності з приходом та розвитком нових технологій більшість трафіку при використанні вебсайтів

припадає на мобільні телефони. Отож, є базова необхідність адаптувати комп'ютерні версії дизайну до мобільних. Адаптивний дизайн/інтерфейс посідає належне місце в списку пріоритетів для розробки програмного продукту.

1.4.2. Можливі шляхи удосконалення вебсервісу

Зважаючи на загальний опис окреслених цілей та пріоритетів, сформовано можливі шляхи удосконалення програмного продукту:

- **Естетичний інтерфейс**. Постійне оновлення дизайну та вдосконалення взаємодії з користувачем для забезпечення максимальної зручності та привабливості. Важливо бути на зв'язку завжди з користувачами через систему відгуків, яку можна реалізувати згодом, й отримувати відгуки й оцінки, проводити анонімні опитування серед аудиторії на предмет наявності дизайнерських помилок й покращення користувацького досвіду (UX). Доцільно формувати думку стосовно дизайну загалом та його окремих елементів, користувацького досвіду саме через зворотній зв'язок з користувачами вебсервісу з пошуку книг.
- **Можливість інтеграції з іншими сервісами**. Додавання підтримки авторизації через інші платформи, такі як Facebook або Twitter, розширить вибір для користувачів. Вони матимуть можливість вибрати платформу, з якої їм найзручніше авторизуватися, що полегшить процес для широкого кола користувачів. Мета розширення функціональності:
  - Багато користувачів вже мають облікові записи на платформах, таких як Facebook або Twitter, і вони можуть вважати зручнішим використовувати ці облікові записи для авторизації на вашому вебсервісі, ніж створювати новий обліковий запис або вводити електронну пошту та пароль.
  - Інтеграція з платформами, такими як Facebook або Twitter, може відкрити нові можливості для соціальної інтеграції вашого вебсервісу.

У майбутньому, наприклад, можна дозволити користувачам ділитися вмістом або активністю з вашого сервісу на їхніх сторінках у соціальних мережах або запрошувати друзів з їх соціальних мереж для участі в моєму вебсервісі.

- **Розширення функціоналу**. Дуже важливо покращувати створений програмний продукт у подальшому. Серед основних категорій, що можна реалізувати та імплементувати до вебсервісу з пошуку книг та додавання їх до категоріальних полиць є:

  - *Система коментарів*. Перспективною є система, що дозволяє користувачам залишати коментарі та відгуки до конкретних книг. Коментарі можуть містити відгуки, рецензії, думки та поради щодо книги. Це є надзвичайно корисним, адже таким чином вебсервіс стає майданчик для дискусій, де можна комунікувати, обмінюватись думками щодо новопрочитаних книг, авторів і т.д.

  - *Система відгуків та зворотнього зв'язку*. Ця система дозволяє користувачам залишати відгуки про весь вебсервіс, його функціонал або якість обслуговування. Обов'язково включає в себе форму зворотного зв'язку для спілкування з адміністрацією сервісу. Життєво необхідна для подальшої оптимізації та покращення програмного продукту.

  - *Система рекомендацій.* В даному випадку дуже важливо реалізувати систему, яка аналізуватиме вибірки читачів та їхні уподобання для надання персоналізованих рекомендацій щодо книг. Це досить непроста задача, але надзвичайно варта реалізації – розширить кругозір читача та познайомить з новими авторами чи книгами для подальшого читання.

- **Покращення вихідного коду у вебсервісі з пошуку книг та додавання їх до категоріальних полиць**. Наразі вебсервіс виконує всі поставлені перед ним цілі, але важливо те, що розподілення коду на різні компоненти, перенесення деякої бізнес-логіки в окремі частини, та й загалом

відредагування програмного коду грає надзвичайно важливу роль у подальшій масштабованості проекту.

Приміром, варто чітко уявити ситуацію, що в деякий момент у майбутньому з’явиться потреба у реалізації нової функції вебсервісу, виправленні наявних помилок тощо. Тому відповідно розподілений, відредагований програмний код слугуватиме меншим зусиллям у виконанні поставлених задач.

1.5. Висновки до розділу 1

Провівши аналіз проблематики створення вебсервісу з пошуку книг та додавання їх до категоріальних полиць було доведено, що існує попит на такий вебсервіс у зв’язку з ростом інтересу до цифрового читання та потребою у зручному інструменті для організації особистих колекцій книг. Проект такого сервісу може мати значний соціальний і комерційний вплив, якщо буде реалізований з урахуванням потреб користувачів та сучасних стандартів технологій.

Було описано основні вимоги користувачів до майбутнього програмного продукту, серед яких такі як естетичний інтерфейс, велика база даних книг та інтеграція з програмним інтерфейсом від Google (Google Books API) та адаптивним інтерфейсом.

Також було виявлено кілька ключових аспектів, які слід врахувати при розробці такого сервісу:

По-перше, важливо забезпечити широкий вибір книг та зручний механізм їх пошуку, щоб користувачі могли легко знаходити та додавати книги до своїх колекцій.

По-друге, дослідження також вказало на необхідність реалізації у майбутньому додаткових функціональних можливостей, таких як система коментарів, відгуків та зворотного зв’язку, які допоможуть створити спільноту користувачів та забезпечити зворотний зв'язок з аудиторією. Крім того, система

рекомендацій може допомогти персоналізувати досвід користувача та рекомендувати нові книги на основі їхніх уподобань та історії читання.

Загалом, результати дослідження підкреслюють важливість створення зручного, функціонального та інтерактивного веб-сервісу для пошуку книг та організації їх колекцій. Шлях до успіху включає розробку імплементацію різноманітних функцій та можливостей, які відповідають потребам та очікуванням користувачів, та постійне вдосконалення та розвиток сервісу з урахуванням змін у галузі та вимог ринку.

РОЗДІЛ 2
ОГЛЯД ВИКОРИСТАНИХ ТЕХНОЛОГІЙ ТА МЕТОДІВ РОЗРОБКИ ВЕБСЕРВІСУ ПОШУКУ КНИГ ТА ДОДАВАННЯ ЇХ ДО КАТЕГОРІАЛЬНИХ ПОЛИЦЬ

2.1. Загальний опис систем розподілених обчислень

Спершу, перед розробкою, необхідно зрозуміти суть архітектури, яка буде реалізована при розробці даного вебсервісу. Дивлячись на світові тенденції та стандарти й задля ефективної роботи було обрано клієнт-серверну архітектуру.

Передусім, до визначення терміну архітектури клієнт-сервер необхідно сказати, що дана архітектура є видом моделі розподілених обчислень, яка, в свою чергу, характеризується як програма, що виконується на декількох комп’ютерах одночасно. Тобто, це використання декількох комп'ютерів (вузлів), об'єднаних в мережу, для спільного вирішення задач. Ця модель базується на принципі поділу великої обчислювальної задачі на менші підзадачі, які виконуються паралельно на різних вузлах, а потім об'єднуються для отримання кінцевого результату. Прикладами можуть бути такі сервіси чи платформи як: Docker, Amazon, Microsoft Azure, Apache, велика кількість сервісів Google (Google Photos, Google MapReduce) тощо. Перелічені платформи та багато інших є світовими гігантами та знаним стандартом використання моделі розподілених обчислень, що використовується усюди. Хочу перелічити основні властивості даної моделі та, відповідно, усіх перелічених сервісів:

- Прозорість. Дана характеристика означає, що у зв’язку зі специфікою конструювання моделі користувачі не знають, де фізично розташовані ресурси чи дані, скільки копій даних існує та де вони розташовані.
- Це є величезною перевагою, адже у таких умовах система зазвичай відновлюється автоматично без впливу на користувача. Як приклад: користувачі сервісу Amazon не знають на якому фізичному сервері зберігаються дані та файли і де вони розташовані географічно – користувач використовує єдиний URL для завантаження та отримання даних; віртуальні машини можуть переміщуватись між фізичними серверами без впливу на запущені додатки, тобто робота триває безперервно; одночасні запити користувачів можуть оброблятися паралельно, без конфліктів та неузгодженостей, адже система автоматично обробляє ці питання тощо.
- Масштабованість. Система, відповідно, може легко розширюватись для підтримки більшої кількості користувачів та великого навантаження на сервери.
- Надійність роботи. Як вже було сказано вище, забезпепчується безперервність роботи та доступність послуг навіть у випадку збоїв чи відмови конкретних вузлів – механізми дублювання, резервування та відновлення.
- Синхронізація. Неймовірно важлива характеристика – відбувається координація між вузлами й тим самим забезпечується узгодженість стану даних та порядку виконання задач, тобто це методи управління чергами задач, блокування ресурсів і т.д.
- Розподіл ресурсів. Всі вузли системи спільно використовують ресурси, такі як пам’ять, мережеві ресурси і т.д.

Вище було коротко описано основні властивості систем розподілених обчислень, хоча їх набагато більше.

Підсумовуючи, розподілені обчислення – це важлива технологія, що є стандартом у сфері конструювання масштабних світових комп’ютерних систем як і сьогодення, так і майбутнього.

2.1.1. Архітектура «клієнт-сервер»

Варто повернутись до архітектури «клієнт-сервер». Як вже було сказано, вона є видом систем розподіленого обчислення й володіє її переліченими перевагами та властивостями відповідно. Дана архітектура складається з трьох компонентів:

- Набір серверів. Вони виконуються основну роботу, тобто надають інформацію чи послуги програмам, що звертаються до них.
- Набір клієнтів. Програми, що використовують сервіси, які надаються серверами, наприклад клієнтський комп’ютер.
- Мережа. Те, що забезпечує взаємодію між клієнтом та сервером.

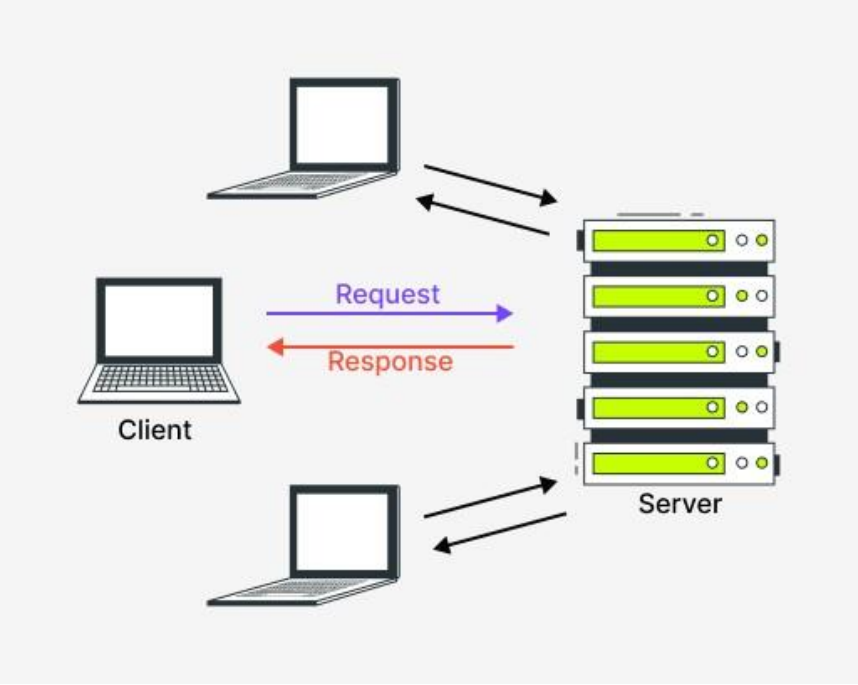

Рис. 2.1. Коротка схема роботи архітектури «клієнт-сервер»

Як видно з рисунку, клієнт, в даному випадку користувач, відправляє запит на сервер для отримання даних і далі сам сервер надсилає відповідь (успіх – статус 200 чи помилка – 400, 401 тощо). Дані відповіді серверу можуть надсилатися у різних форматах, але найпопулярніший та найзручніший – JSON формат. Він є читабельним як для людей, так і для машин.

Варто зазначити, що сервер та клієнт зазвичай у такій архітектурі розташовані на різних комп'ютерах, але є ситуації, коли все знаходиться на одному комп'ютері – в такому випадку маємо справу з локальним сервером. Підхід з локальним сервером є часто використовуваним, адже він є ефективним для розробки та тестування програмного продукту через можливість тестування застосунків перед розгортанням на живих віддалених серверах, невелику затримку, оскільки не надсилаються запити через Інтернет та можливість працювати навіть без доступу до мережі Інтернет.

2.1.2. Поняття моделей «тонкого» та «товстого» клієнтів

Дворівнева модель архітектури «клієнт-сервер» поділяється на модель тонкого та модель товстого клієнту. Коротко кажучи, тонкий клієнт – модель, в рамках якої вся логіка застосування та управління даними зосереджена на сервері. Тобто у цьому випадку веб-сайт виконує мінімальні обчислення та обробку даних, передаючи більшість запитів і бізнес-логіки до сервера. У свою чергу товстий клієнт – навпаки, більшість бізнес-логіки та обчислень концентрує на клієнтському боці, наприклад, у браузері.

Розроблений вебсервіс пошуку книг та додавання їх до категоріальних полиць є прикладом товстого клієнту. Адже використовуючи Javascript-бібліотеку React (про що йтиметься далі) більшість логіки – фільтрація, сортування, обробка даних відбувається на клієнті без додаткового залучення серверу.

Доцільно зазначити переваги «тонкого/товстого» клієнта:

- **«Тонкий» клієнт:**
  - До переваг можна віднести мінімізований ризик виникнення несправностей та низькі технічні вимоги до обладнання;
  - До недоліків: в такій архітектурі до одного серверу може бути підключено необмежену кількість клієнтів, але у разі виникнення помилки постраждають усі користувачі; відсутність можливості працювати без підключення до Інтернету; зниження продуктивності у разі високих навантажень на сервер.
- **«Товстий» клієнт**:
  - До переваг відносяться: високу функціональність; можливість роботи в режимі офлайн через основну обробку даних на клієнті; швидка обробка даних; відсутність залежності від віддалених серверів.
  - До недоліків: досить трудомісткий процес налаштування та інсталяції; необхідність у постійному технічному обслуговуванні та встановленні оновлень; складність у синхронізації даних; великий розмір дистрибутива.

Підсумовуючи, нижче на рисунку 2.2. коротко зображені принципи роботи вищезазначених моделей.

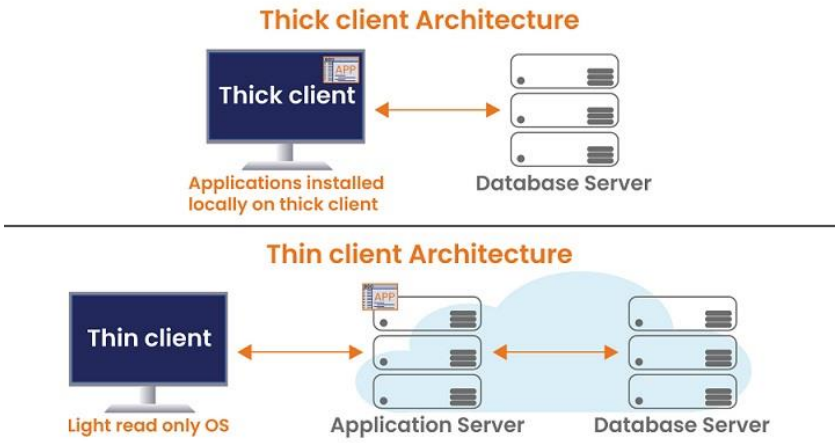

Рис 2.2. Модель «товстого» та «тонкого» клієнтів

2.2. Огляд основних використаних технологій та методів

Як було зазначено вище, модель розробленого вебсервісу передбачає «товстий» клієнт. Це означає, що, хоч і більшість коду буде визначено на клієнті та потребуватиме більше ресурсів від обчислювальної машини, зате дані будуть оброблятись динамічно, швидко та надійно, й потребуватимусть меншого навантаження на сервер.

Для початку необхідно обрати мову програмування для досягнення відповідних цілей та задоволення визначених вимог.

Рухаючись далі, варто визначитись з відповідною бібліотекою чи фреймворком, адже в сучасності будь-який проект розробляється не на самій лише мові програмування, а за допомогою відповідних інтрументів, що полегшують написання коду та розширюють простір для створення сучасних ПП. Вони є досить прогресивними та їхній функціонал написаний на відповідній мові програмування. Зазначу, що бібліотека чи фреймворк є світовими стандартами веброзробки, що дозволяють створювати проекти високої якості.

2.2.1. Мова програмування JavaScript

JavaScript – це високорівнева скриптова мова програмування, яка широко використовується для створення інтерактивних програм та сайтів. За допомопогою JS можна додавати анімацію та спливаючі повідомлення, валідувати форми, оновлюти контент та змусити інтерфейс реагувати на дії користувача. Наприклад, на мобільних пристроях варто перемальовувати навігаційне меню, з чим і допомагає JS, аби було зручно користуватись вебсайтом зі смарфтону чи планшету тощо. Коротко кажучи, Javascript «оживляє» веб-сайт, робить його динамічним, функціональним та чутливим до дій користувача.

JS має свої особливості:

- Інтерпретованість. JavaScript виконується в реальному часі в середовищі браузера без необхідності попередньої компіляції. Розробник вносить зміни до коду JavaScript і відразу бачить результат, що є досить зручним.
- Інтеграція з HTML та CSS. У комбінації з ними мова JavaScript дозволяє створювати повнофункціональні веб-сторінки.
- Динамічна типізація. Тип змінних JavaScript визначається автоматично при виконанні коду. Програмістам не потрібно надавати його заздалегідь, що спрощує роботу.
- Велика кількість бібліотек та фреймворків JavaScript. Готові рішення для різних завдань заощаджують час IT-фахівців.
- Кросплатформність. Код JavaScript може бути виконаний на різних платформах та пристроях.
- Автоматичне збір сміття. Полегшує керування пам'яттю, автоматично видаляючи об'єкти, які не використовуються.

- Імперативність, функціональність та принципи ООП. JavaScript підтримує основні парадигми програмування. Це дозволяє створювати складні структури даних, покращувати організацію коду, застосовувати функціональні концепції та описувати точні кроки для досягнення результату.
- Асинхронність. За допомогою таких механізмів, як проміси та async/await, JavaScript ефективно обробляє операції, які не потребують негайного завершення без блокування інших завдань.
- Вбудована підтримка DOM (Document Object Model). Для динамічної взаємодії з елементами веб-сторінки, обробки подій та змін.

Переваги мови JavaScript:

- Простота вивчення. Зрозумілий і логічний синтаксис JavaScript робить його хорошим варіантом для розробників-початківців.
- Універсальність. Мова JavaScript чудово працює на будь-яких пристроях та платформах. Все, що потрібно – це браузер.
- Гнучкість та сумісність. JavaScript легко інтегрується з іншими технологіями, такими як HTML та CSS, а також підтримує різноманітні парадигми програмування, асинхронність та динамічну типізацію.
- Корисні інструменти. Бібліотеки та фреймворки JS дозволяють вирішувати широкий спектр завдань: від створення сайтів та додатків до розробки серверних рішень.

Недоліки:

- JS буває менш продуктивним, ніж інші мови програмування (особливо у високонавантажених обчислювальних завданнях);
- JavaScript вразливий для деяких видів кібератак, наприклад, міжсайтового скриптингу (XSS). Але проблеми з кібербезпекою характерні не лише для JS.

- У масштабних та довгострокових проектах динамічна типізація JS може створювати труднощі та збільшувати витрати на підтримку софту;
- Не всі браузери однаково обробляють JavaScript код, що впливає на сумісність.

2.2.2. React – бібліотека Javascript

ReactJS, також відома як React, є популярною бібліотекою JavaScript для створення інтерфейсів користувача. Її також називають фронтенд (інтерфейсною) бібліотекою JavaScript. Вона була розроблена Facebook у 2013 році й наразі широко використовується для створення динамічних та інтерактивних веб-додатків.

React — це бібліотека JavaScript для створення інтерфейсів користувача (UI) в Інтернеті. Це декларативна бібліотека на основі компонентів, яка дозволяє розробникам створювати повторно використані компоненти інтерфейсу користувача, і вона дотримується підходу Virtual DOM (Document Object Model), який оптимізує продуктивність рендерингу (відмалювання інтерфейсу) шляхом мінімізації оновлень DOM. React швидкий і добре працює з іншими інструментами та бібліотеками.

React є однією з найвимогливіших бібліотек JavaScript, оскільки вона оснащена масою функцій, які роблять її швидкою та готовою до використання. Нижче наведено кілька основних властивостей React:

1. **Компонентна архітектура**. React надає функцію для розбиття інтерфейсу користувача на менші самодостатні компоненти. Кожен компонент може мати власний стан і властивості.
2. **JSX (розширення синтаксису JavaScript)**. JSX — це розширення синтаксису для JavaScript, яке дозволяє розробникам писати HTML-подібний код у своїх файлах JavaScript. Це робить компоненти React більш читабельними та зрозумілими.

```
<h1>Welcome to {name}</h1>
```

Рис. 2.3. Фрагмент JSX-коду

3. **Віртуальний DOM**. React підтримує спрощене представлення фактичного DOM у пам’яті. Коли відбуваються зміни, React ефективно оновлює лише необхідні частини DOM.

Рис. 2.4. Спрощена модель роботи Virtual DOM

4. **Одностороння прив'язка даних**. Дані в React протікають лише в одному напрямку - передаються зверху вниз, тобто від батьківських до дочірніх компонентів. Властивості (Props) у дочірньому компоненті не можуть повертати дані своєму батьківському компоненту, але можуть мати зв’язок із батьківськими компонентами для зміни станів відповідно до наданих вхідних даних.

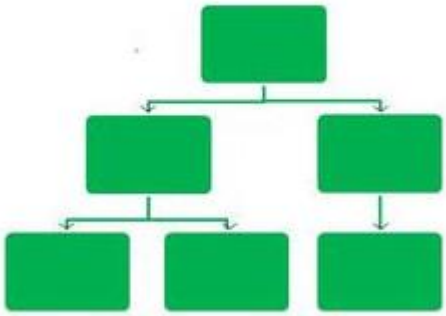

Рис. 2.5. Одностороння прив’язка даних у React

5. **Продуктивність**. Як було сказано раніше, React використовує віртуальний DOM і оновлює лише змінені частини. Таким чином, DOM працює швидше. DOM виконується в пам’яті, тому ми можемо створювати окремі компоненти, завдяки чому DOM працює швидше.

6. **Компоненти**. React ділить веб-сторінку на кілька компонентів, оскільки він заснований на компонентах. Кожен компонент є частиною дизайну інтерфейсу користувача, який має власну логіку та дизайн, як показано на зображенні нижче. Таким чином, логіка компонентів, написана на JavaScript, полегшує та пришвидшує роботу та може використовуватися повторно.

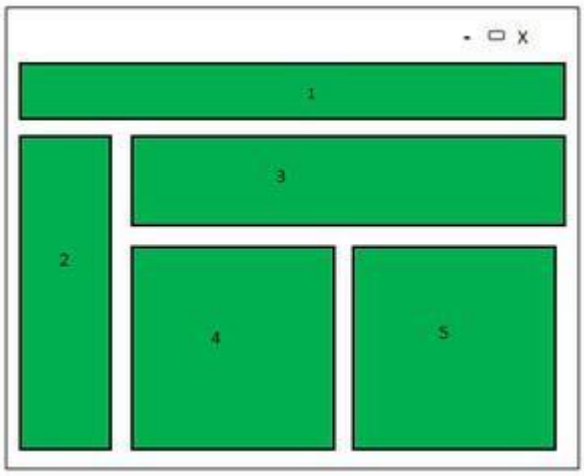

Рис. 2.6. Компонентний підхід у React

7. **Односторінкові програми (SPA)**. React рекомендовано для створення SPA, що дозволяє плавно оновлювати вміст без перезавантаження сторінки. Його орієнтація на багаторазові компоненти робить його ідеальним для додатків у реальному часі.

Кожен компонент React має власний життєвий цикл. Життєвий цикл компонента можна визначити як ряд методів, які викликаються на різних етапах існування компонента. React автоматично викликає ці методи в різні моменти

життєвого циклу компонента. Розуміння цих фаз допомагає керувати станом, виконувати побічні ефекти та ефективно оптимізувати компоненти:

1. Ініціалізація. Це етап, на якому створюється компонент із заданими Props і стандартним станом. Це робиться в конструкторі класу компонентів;
2. Фаза монтування:
    - Конструктор. Метод конструктора ініціалізує компонент. Тут можна налаштувати початковий стан і прив’язати обробники подій.
    - render(). Цей метод повертає JSX-представлення компонента. Він викликається під час початкового рендерингу та наступних оновлень.
    - componentDidMount(). Після вставлення компонента в DOM цей метод викликається. Використовується зазвичай для побічних ефектів, наприклад таких як отримання даних або налаштування таймерів.
3. Етап оновлення:
    - componentDidUpdate(prevProps, prevState). Викликається після оновлення компонента через нові властивості або зміни стану. Побічні ефекти зазвичай опрацьовуються тут.
    - shouldComponentUpdate(nextProps, nextState). Визначає, чи потрібно повторно відтворити компонент. Налаштувавши цей метод, зазвичай можна оптимізувати продуктивність.
    - render(). Знову ж таки, метод render() відображає зміни в стані або атрибутах під час оновлень.
4. Фаза демонтування:
    - componentWillUnmount(). Викликається безпосередньо перед видаленням компонента з DOM. Очистіть ресурси (наприклад, слухачі подій, таймери).

Підсумовуючи вищезазначену інформацію, React – фундаментальна бібліотека, за допомогою якої написано більшість веб-сайтів сьогодення та яка,

за допомогою компонентного підходу, віртуального DOM та методів життєвого циклу компонентів справді показала себе як лідера фронтенд-розробки. Нею буде обов’язково скористовано при написанні програмного коду.

2.2.3. Google Books API

Так як розробка вебсервісу пошуку книг та додавання їх до категоріальних полиць не передбачає використання додаткових баз даних та їх ручне створення, необхідно знайти інструмент, який буде містити в собі велику базу даних, пошук книг та методи їх додавання до категоріальних полиць. Тобто, інакшими словами, це - набір правил і протоколів, які описують, як дві програми або системи взаємодіють одна з одною (програмний інтерфейс). Використовуючи методологію REST за допомогою HTTP запитів отримуватимуться дані про книги й виконуватимуться певні дії, надсилаючи дані на сервер та отримуючи відповіді.

Рис. 2.7. Логотип «Google Books»

Серед безлічі схожих програмних інтерфейсів (API) було обрано саме Google Books API. Серед основних причин вибору:

- Велика база даних книг. Google Books API має, мабуть, найбільше базу даних книг з поміж усіх його конкурентів. І це не дивно, адже Google – корпорація світового масштабу.
- Можливість широкого пошуку книг та взаємодії з віртуальними книжними полицями;

- Інтеграція з іншими сервісами Google, наприклад, такими як Google Drive та взаємодія з методом авторизації Google OAuth2, що також є стандартом та поширеною методикою авторизації та автентифікації користувачів;
- Досить хороша документація, часто відвідувані форуми та центри підтримки.

Варто відзначити, що у зв'язку з використанням Google Books API та методом авторизації OAuth2 не потрібно власноруч створювати додаткову базу даних в таких сервісах як, наприклад: MongoDB чи PostgreSGL, адже OAuth2 – метод авторизації через обліковий запис Google. Тобто книги, що будуть додані до категоріальних полиць, будуть зберігатись у обліковому записі користувача, і користувачу необхідно лише при наступній авторизації увійти у той самий обліковий запис Google для того, аби побачити свої книги чи загалом взаємодіяти з вебсервісом. Це є дуже зручно, адже не потрібно створювати власну базу даних, систему авторизації, що є досить складним і не відноситься до інтерфейсної частини (frontend) розробки обраного вебсервісу.

Нижче, на рисунку 2.8, показано комбінацію бібліотеки React JS та програмного інтерфейсу Google Books API. У результаті вийде швидкий, динамічний та естетичний вебсервіс.

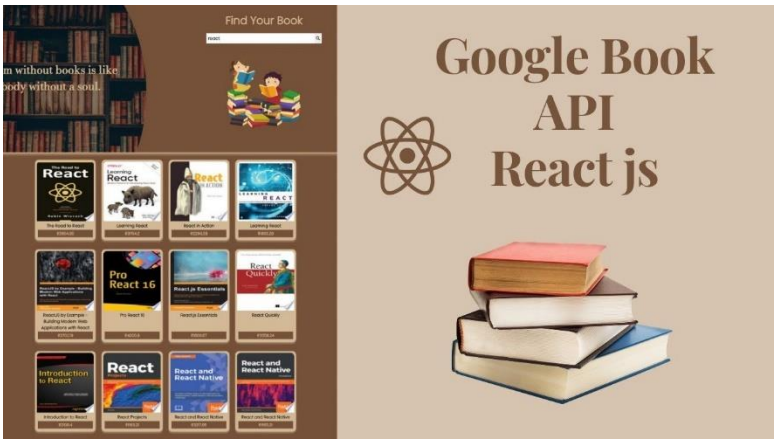

Рис. 2.8. Комбінація React та API від Google

2.2.4. Текстовий редактор Visual Studio Code

Visual Studio Code — це потужна та безкоштовна IDE, яка підтримує розробку понад 10 мов і багатий ринок, яким керує представництво Microsoft. Visual Studio Code дуже зручна, є цілісною та ефективною при написанні коду. Основні особливості:

- *Доступ для MacOS, Windows та Linux*;
- *Легке редагування, створення та налагоджування проєктів*. У своїй основі Visual Studio Code має блискавичний редактор вихідного коду, ідеальний для повсякденного використання. Завдяки підтримці сотень мов VS Code допомагає працювати продуктивно за допомогою підсвічування синтаксису, зіставлення дужок, автоматичного відступу, вибору фрагментів тощо. Інтуїтивно зрозумілі комбінації клавіш, легке налаштування та зіставлення комбінацій клавіш, надані спільнотою, дозволяють легко орієнтуватися в коді.
- *Широке поле для налаштувань*. Можна налаштувати кожну функцію на свій смак і встановити будь-яку кількість сторонніх розширень. Хоча більшість сценаріїв працюють «з коробки» без конфігурації, VS Code також розвивається разом з користувачем.
- VS Code містить розширену вбудовану підтримку розробки Node.js за допомогою JavaScript і TypeScript, що базується на тих самих базових технологіях, які керують Visual Studio. VS Code також містить чудові інструменти для веб-технологій, таких як JSX/React, HTML, CSS, SCSS, Less і JSON.

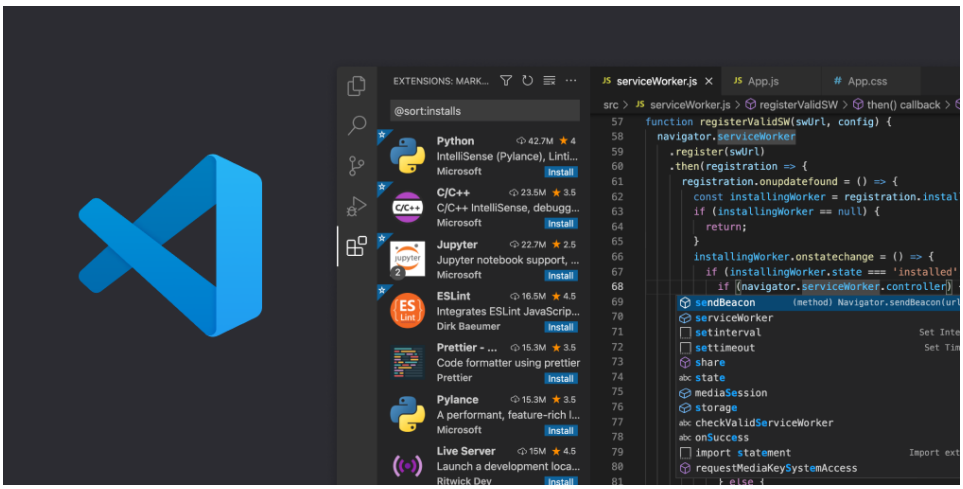

Рис 2.9. Вигляд робочої області VS Code

2.2.5. GitHub

GitHub — це веб-платформа для хостингу та спільної розробки програмного забезпечення, яка базується на системі керування версіями Git. Вона дозволяє розробникам зберігати, управляти та відслідковувати зміни у своєму коді, а також співпрацювати з іншими розробниками.

Основними функціями GitHub є репозиторії, які містять файли проектів, історію змін та інструменти для управління цими змінами, такі як гілки (branches), коміти (commits) та запити на злиття (pull requests).

Окрім базових функцій керування кодом, GitHub пропонує безліч додаткових інструментів для покращення співпраці та управління проектами. Наприклад, GitHub Issues дозволяє створювати та відслідковувати завдання та помилки, а GitHub Actions автоматизує робочі процеси, такі як тестування та розгортання коду. Платформа також підтримує інтеграцію з багатьма іншими сервісами та інструментами, що робить її потужним інструментом для розробників усього світу.

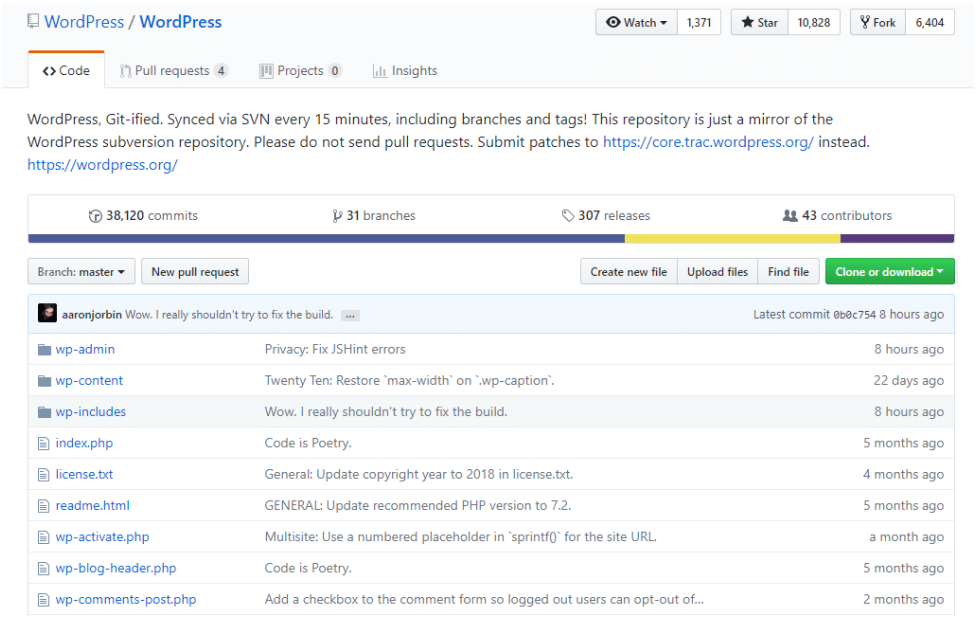

Рис. 2.10. Вигляд інтерфейсу сервісу GitHub

2.2.6. Netlify

Netlify — це платформа для хостингу та автоматизованого розгортання вебсайтів і вебдодатків. Вона забезпечує безперервну інтеграцію та безперервне розгортання (CI/CD), спрощуючи процес публікації вебпроєктів. Netlify дозволяє розробникам розгорнути сайти з різних систем керування версіями, таких як GitHub, GitLab або Bitbucket, з мінімальними зусиллями. Основні переваги Netlify включають простоту використання, швидкість розгортання та потужні можливості для керування контентом і конфігураціями.

Однією з найбільших переваг даної платформи є автоматизоване розгортання. Це означає, що після того як користувач виклав вебсайт до мережі Інтернет та вирішив доопрацювати якісь деталі в програмному коді, то йому не обов’язково кожен раз після успішного доопрацювання складати збірку програмного коду – все зробить сам Netlify. А все відбувається так, що при розгортуванні сайту у платформі Netlify користувач вказує файли через обліковий запис GitHub та команду збірки, і в момент, коли користувач оновлює файли на GitHub, то Netlify самостійно розгортує нову версію вебсайту. Це є

надзвичайно зручно, адже користувачу треба лише надіслати оновлені файли на GitHub і не потрібно хвилюватись за цілісність збірки в глобальній мережі Інтернет.

Також у Netlify є чудова опція зберігання змінних середовища, які використовуються у кожному проєкті в сьогоденні та багато інших переваг.

Рис. 2.11. Логотип сервісу «Netlify»

2.3. Висновки до розділу 2

Підсумовуючи вищезазначену інформацію в даному розділі, варто підкреслити, що найбільш доцільним для створення вебсервісу пошуку книг та додавання їх до категоріальних полиць буде використання системи розподілених обчислень, а архітектури «клієнт-сервер» та моделі «товстого» клієнту, адже вебсервіс повинен бути актуальним, відповідати вимогам часу та відповідним стандартам у частині базової архітектури.

У свою чергу, мова програмування Javascript спеціально створена для написання програмного коду для веб-застосунків, тому вибір був зроблений на ній. Також, для того, щоб веб-сервіс був сучасним у плані швидкодії, варто використати бібліотеку або відповідний Javascript-фреймворк. Було обрано бібліотеку React через її численні переваги та широку спільноту. Найголовніша частина – це пошук чи вибір програмного інтерфейсу (API). Як було зазначено вище, Google Books API є найпопулярнішим API з великою базою даних, гарною документацією та підтримкою методу авторизації Google OAuth2 та відносною простотою у використанні.

Насамкінець, було визначено, що немає потреби створювати власну систему авторизації, власноруч налаштовувати додаткову базу даних – все зберігатиметься у обліковому записі Google користувача.

РОЗДІЛ 3
ОГЛЯД РЕАЛІЗАЦІЇ ВЕБСЕРВІСУ

3.1. Осмислення та поділ проекту на частини

Перед реалізацією веб-сервісу для пошуку книг та їх додавання до категоріальних полиць важливо окреслити частини розробки, що значно спростить процес реалізації.

Першочерговим кроком є ініціалізація програмного інтерфейсу Google Books API. Для інтеграції Google Books API необхідно створити проект у Google Cloud Platform, отримати облікові дані та налаштувати проект для використання в створюваному веб-сервісі.

Додавання книг до категоріальних полиць може бути реалізовано шляхом як і збереження обраних книг у локальному стані клієнтського додатка або у браузерному сховищі, так і надсиланням HTTP запитів на сервер Google. Було вирішено використовувати останній варіант, адже локальний стан клієнтського застосунку є не сталим та постійно змінюється, а браузерне сховище час від часу може очищатись, що, відповідно видалятиме додані книги користувача. А у разі надсилання HTTP запитів та взаємодії з сервером Google додані книги завжди зберігатимуться у обліковому записі Google користувача.

Друга важлива частина розробки полягає у створенні фронтенд частини веб-сервісу на React. React дозволяє створювати інтуїтивно зрозумілий та динамічний інтерфейс користувача, де можна легко реалізувати функціонал пошуку книг та організації їх у категоріальні полиці.

Використовуючи компоненти React, можна створити зручні форми для пошуку, відображення списків книг та управління полицями. Оскільки буде використано Google Books API для отримання даних про книги і Google OAuth2 для автентифікації користувачів, не потрібно створювати власну базу даних чи систему авторизації. Усі необхідні дані про книги отримуються через API, а автентифікація користувачів обробляється через Google OAuth2, що значно спрощує процес розробки та обслуговування веб-сервісу.

3.2. Ініціалізація проєкту в Google Books API

Спочатку необхідно перейти на сайт офіційної документації Google Cloud Console. Далі натиснути на кнопку «Go to Create a Project».

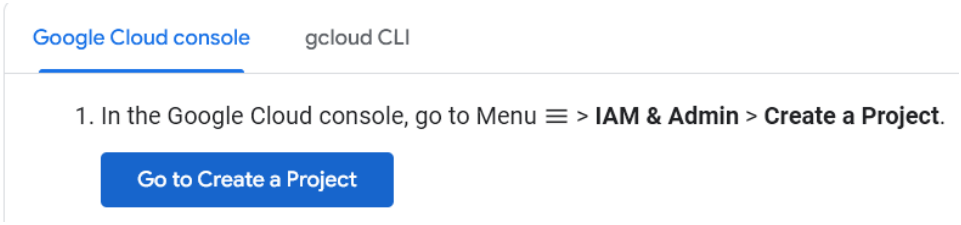

Рис. 3.1. Вигляд контекстного меню при вході до сервісу Google Cloud Console

Далі у відповідних полях вказати назву проєкту та, опціонально, організацію.

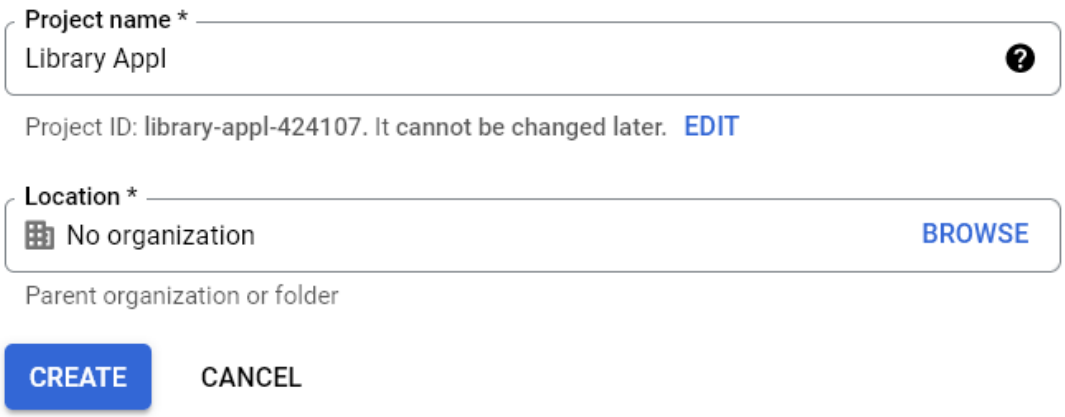

Рис. 3.2. Меню для вказання ім’я та організації

Після натискання клавіші «Create» у верхньому лівому куту можна побачити коротку інформацію про створений проєкт.

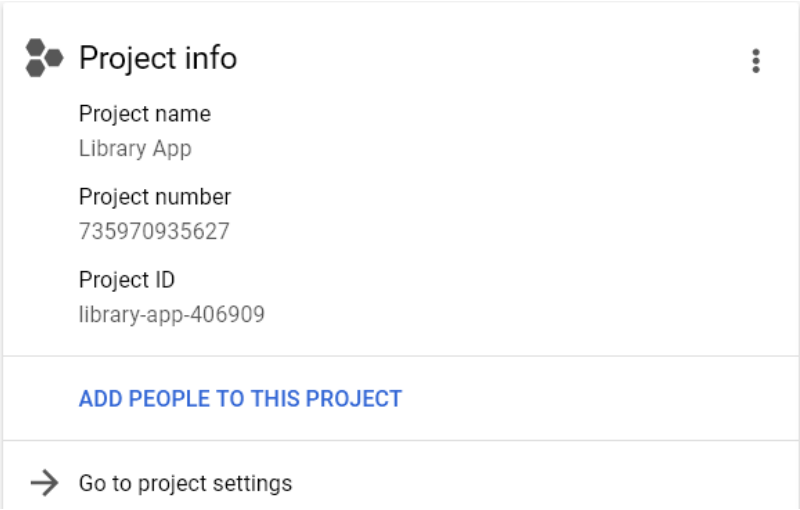

Рис. 3.3. Коротка інформація про створений проект в Google Cloud Console

Після цього перейти на вкладку «Credentials» і вже тут відображаються API-ключі та OAuth2 Cliend Ids даного проєкту. OAuth2 Cliend Ids це саме те, що потрібно створити для можливості авторизації через Google Oauth2.

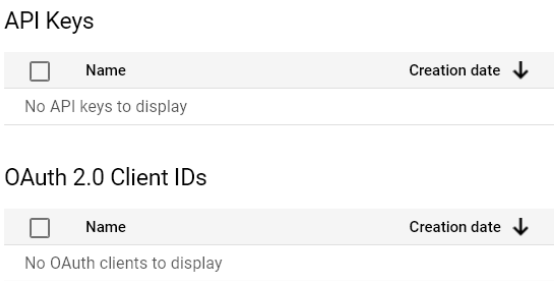

Рис. 3.4. Меню огляду наявних API-ключів та OAuth2-клієнтів

Клацнути на «Create Credentials» та у випаючому списку обрати «OAuth Client ID».

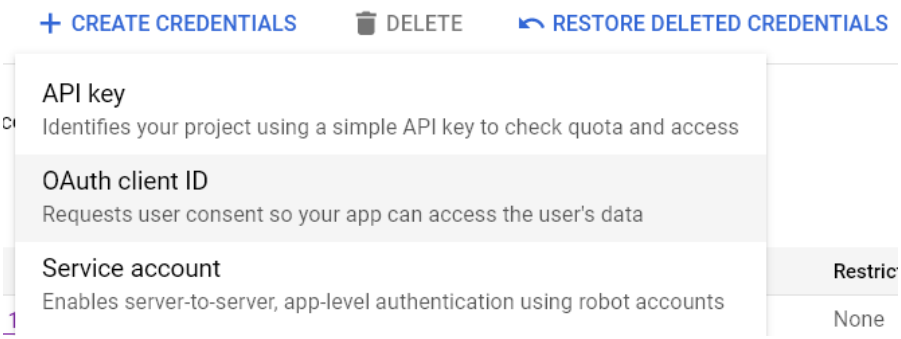

Рис. 3.5. Порядок створення OAuth2-клієнту

Далі послідовно необхідно заповнити усі поля. Спочатку вказується Application type (тип застосунку) – потрібно обрати «Web application», далі ім'я застосунку (у даному випадку «Library App»). Далі - дуже важливо, в опціях «Authorized JavaScript origins» вводиться та адреса локального серверу, за якою буде створюватись вебсервіс в браузері, по замовчуванню у більшості середовищ розробок – це http://localhost:3000. Бажано також і ввести таку адресу як http://localhost, адже у майбутньому, при тестуванні версії вебсервісу, яка додаватиметься у Інтернет, після двокрапки можуть бути інші цифри, тому аби Google OAuth2 не блокував і дозволяв авторизуватись з інших адрес локального серверу потрібно здійснити такі кроки. Ті ж адреси потрібно увести і в наступне поле «Authorized Redirect URLs».

Після успішної реєстрації показується спливаюче вікно з успішним створенням OAuth Client та з такими обліковими даними як «Client ID» «Client Secret» Значення цих облікових даних потрібно десь записати, аби вони були під рукою і ні в якому разі нікому не показувати та не надсилати. Це дані, що потрібні для правильної авторизації та автентифікації користувача при надісланих HTTP запитах. Тому надзвичайно важливо суворо дотримуватись правил конфіденційності. У майбутньому, на етапі написання коду, буде перенесено значення цих даних у змінні середовища, аби вони не були перехоплені різними типами хакерських атак при користуванні вебсервісом.

На цьому етапі все готово для початку ініціалізації проєкту в текстовому редакторі.

3.3. Створення інтерфейсної частини вебсервісу

Передусім, перед ініціалізацією проєкту необхідно визначити функціональність вебсервісу:

- Пошук книг за введеним запитом;
- Перегляд конкретної книги з інформацією про неї;
- Додавання книги до категоріальної полиці;
- Видалення конкретної книги з категоріальної полиці;
- Видалення усіх книг з конкретної полиці.

3.3.1. Ініціалізація проєкту

З самого початку було створено папку, де буде розташовуватись проєкт. Я назвав її «Library_Appl». Після цього у відкритому вікні редактору VS Code потрібно відкрити новостворену папку через меню «File» - «Open Folder» та вибрати її. Тепер можна перейти до написання коду.

Для ініціалізації проєкту було виконано команду «create-react-app» та задано ім’я проєкту у терміналі редактору. Після деякого часу очікування, створюється шаблон з початковими файлами, такими як App.js, index.css, style.css, index.html та інші. Деякі файли є зайвими, тому було прибрано їх.

Додатково, були встановлені пакети, що допоможуть у розробці. Серед них – webpack (збирач файлів, важливий для розробки), react, react-dom – сама бібліотека React JS; @react-oauth/google – для авторизації за допомогою Google OAuth2, react-router-dom – для навігації по сайту, axios – пакет для надсилання HTTP запитів та інші. У фінальному варіанті ось усі встановлені пакети:

```
"dependencies": {
    "@babel/core": "^7.23.5",
    "@babel/node": "^7.22.5",
    "@babel/preset-env": "^7.23.5",
```

```
    "@babel/preset-react": "^7.22.5",
    "@react-oauth/google": "^0.12.1",
    "axios": "^1.4.0",
    "babel-loader": "^9.1.2",
    "clean-webpack-plugin": "^4.0.0",
    "clsx": "^2.1.0",
    "css-loader": "^6.8.1",
    "dotenv-webpack": "^8.0.1",
    "express": "^4.18.2",
    "file-loader": "^6.2.0",
    "fs-extra": "^11.1.1",
    "html-webpack-plugin": "^5.6.0",
    "immer": "^10.0.3",
    "jwt-decode": "^4.0.0",
    "npm-run-all": "^4.1.5",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-icons": "^5.0.1",
    "react-router-dom": "^6.14.0",
    "sass": "^1.70.0",
    "sass-loader": "^14.1.0",
    "style-loader": "^3.3.3",
    "use-immer": "^0.9.0",
    "webpack": "^5.88.1",
    "webpack-cli": "^5.1.4",
    "webpack-dev-server": "^4.15.1"
}
```

3.3.2. Базова структура

Кореневою папкою з усіма основними файлами є папка «app». Тут зберігаються файли компонентів, файли глобальних контекстів, папка з картинками, папка «node_modules» - для розробника вона неважлива – там зберігаються файли завантажених пакетів, головний файл App.js, index.html та глобальний файл стилів global.scss.

Нижче розташовані файли конфігурації. Їх потрібно використовувати не так часто, але вони дуже необхідні для правильної роботи.

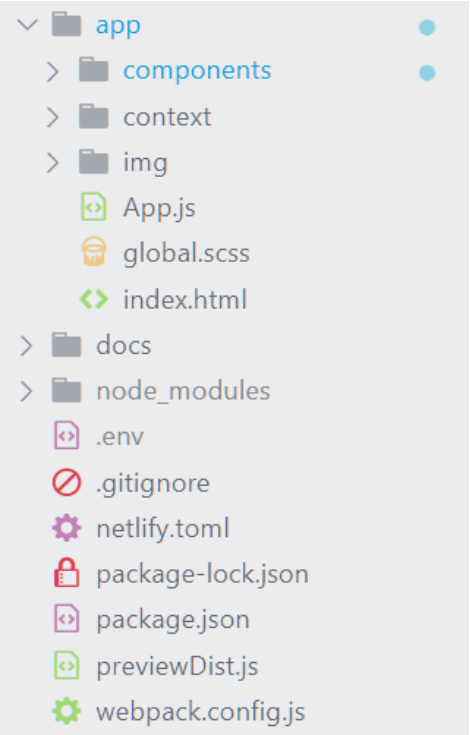

Рис. 3.6. Кореневі папки з файлами проекту

Файл App.js є головним, адже у ньому знаходяться всі компоненти, маршрути та деяка глобальна логіка створюваного вебсервісу. App.js повертає ось таку розмітку, тобто вкладені компоненти вебсервісу:

```
<GoogleOAuthProvider clientId={process.env.REACT_APP_CLIENT_ID}>
        <StateContext.Provider value={state}>
          <DispatchContext.Provider value={dispatch}>
            <FlashMessages messages={state.flashMessages.value} />
            <BrowserRouter>
              {state.loggedIn ? <Header /> : <HeaderLoggedOut />}
              <Routes>
                <Route path="/" element={state.loggedIn ? <SearchPage /> :
<LoginPage />} />
                <Route path="/search/:searchValue/*" element={<SearchPage />} />
                <Route path="/bookshelves/*" element={<Bookshelves />} />
                <Route path="/bookshelves/:bookshelf_type/*"
element={<Bookshelves />} />
```

```
                    <Route path='/search/:searchValue/:id/*'
element={<ViewSingleBook />} />
                    <Route
                      path='/bookshelves/:bookshelf_type/:id/*'
                      element={<ViewSingleBook bookshelves={true} />}
                    />
                    <Route path='*' element={<NotFound />} />
                  </Routes>
                </BrowserRouter>
              </DispatchContext.Provider>
            </StateContext.Provider>
          </GoogleOAuthProvider>
```

Тепер варто розібратись з вищезазначеними компонентами. Глобальна обгортка <GoogleOAuthProvider> необхідна для авторизації через технологію Google OAuth2. Вона не є обов'язковою і можна було створювати окремі файли, локальний сервер, але цей пакет значно полегшує авторизацію. Все, що необхідно – це обгорнути усі компоненти даним провайдером, додати значення client_id та у компоненті <LoginPage /> виконати певні дії.

Далі видно такі обгортки як <StateContext.Provider> та <DispatchContext.Provider>. Це провайдери, що використовують React Context – для того, аби використовувати глобальний стан у програмі для певних дій, адже зазвичай стан є локальний для кожного компоненту. Компонент <FlashMessages /> саме і використовує визначені глобальні провайдери та слугує для відображення спливаючих повідомлень про дії користувача у верхній частині екрану.

Перед визначенням маршрутів, було описано умову, згідно якої неавторизований користувач бачитиме компонент <HeaderLoggedOut /> (верхню частину сайту), в той час як авторизований – інший компонент <Header />. Це зроблено для того, аби лише авторизовані користувачі могли користуватись навігаційним меню.

Після цього можна визначити роутинги (маршрути). Тобто, це адресні рядки, по шляху яких користувачі бачитимуть створені компоненти. Наприклад,

на маршруті «/» повинна бути домашня сторінка, «/bookshelves» - сторінка з категоріальними полицями, «/search/:searchResult» - вказане пошукове значення, введене користувачем.

Файли компонентів показані на рисунку 3.6. Для кожного компоненту було створено окрему папку, у якій знаходився головний JavaScript файл компоненту та файл стилів, котрий під'єднується за допомогою імпорту. Дана структура запобіжить плутанині, адже для кожного компоненту – свій файл стилів.

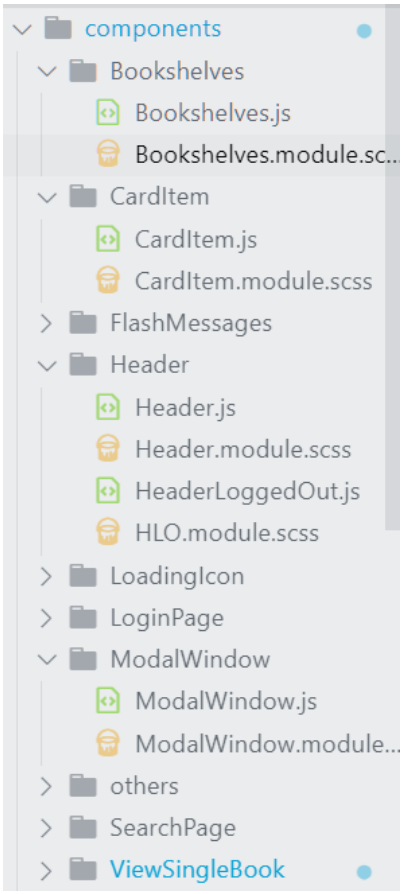

Рис 3.7. Фрагмент показу файлів папки компонентів

3.3.3. Короткий огляд основних концепцій

Як було зазначено вище, отримування інформації буде здійснюватись з серверу Google за допомогою HTTP запитів. Відповідно, у кожному компоненті треба надіслати запит на сервер, отримати результат (чи обробити помилку) та

помістити дані у стан компоненту (за допомогою функції useState()). Таким чином, дані зберігатимуться у стані і в результаті у розмітці JSX (HTML-подібний код для відмалювання елементів сторінки) ці дані необхідно відобразити.

Основна схема, за якою буде надсилатись запит на сервер та отримуватимуться дані:

```
async function getData() {
        try {
          if (activeLink[bookshelf_type] > 1) {
            const storedBookshelfResults =
localStorage.getItem(`bookshelfResults_${bookshelf_type}`)
            if (storedBookshelfResults) {
              setResult(JSON.parse(storedBookshelfResults))
            } else {
              setResult('')
            }
          } else if (activeLink[bookshelf_type] == 1) {
            setResult([])
            setWelcomePage(false)
            setIsLoading(true)
            const apiDestination = getApiDestination(bookshelf_type)
            const response = await
Axios.get(`/mylibrary/bookshelves/${apiDestination}/volumes`, {
              headers,
              CancelToken: ourRequest.token,
            })
            setIsLoading(false)
            setResult(response.data.items !== undefined ? response.data.items : '')

            if (response.data.items != undefined) {
              setDeleteAll(true)
              localStorage.setItem(
                `bookshelfResults_${bookshelf_type}`,
                JSON.stringify(response.data.items)
              )
            } else {
              setDeleteAll(false)
              console.log(`У полиці ${bookshelf_type} немає книг.`)
```

```
            }
          }
        } catch (err) {
          console.log('Помилка запиту: ', err)
        }
      }
```

Дивлячись вищенаведений програмний код, можна помітити, що створюється асинхронна функція (адже йде асинхронне звернення до серверу, тому програма не знає скільки це займе часу), де за допомогою пакету Axios та його методу get отримуються дані згідно адрес, що представлені у документації. У адресі визначено динамічний параметр id, що береться з поля «id» JSON-відповіді на запит щодо конкретної книги. Відповідно id буде змінюватись динамічно та відповідати кожній конкретній книзі.

Додатково, можна помітити зміну деяких додаткових станів, таких як setIsLoading(false) – для відображення кільця завантаження, та роботу з localStorage – це було зроблено для того, щоб дані конкретної книги зберігались у локальному сховищі браузера, щоб при можливості надсилалось менше запитів на сервер, що, в свою чергу, значно прискорює відображення даних та збільшує швидкодію. У різних компонентах буде використано ідентичні підходи з localStorage. Наприклад, ось власноруч створений механізм за допомогою стану компоненту (функції useState()) у компоненті категоріальних полиць:

```
const [activeLink, setActiveLink] = useState({
    favorites: 0,
    'to-read': 0,
    'reading-now': 0,
    'have-read': 0,
  })
```

З початку монтування компоненту є значення 0 біля назв кожного з ключів об’єкту полиць. При кліку на кнопку назви полиці 0 змінюється на 1 біля відповідної назви ключа об’єкту. Клацнувши на наступну кнопку, значення також змінюється з 0 на 1. При цьому, результат відображення книг, тобто дані,

зберігаються у локальне сховище браузера localStorage, і тому, коли користувач повернеться знову на деяку полицю – запит до сервера не йтиме, дані візьмуться з локального сховища, що значно прискорить відображення.

Переходячи до тематики асинхронних HTTP запитів до серверу, варто відзначити, що для деяких запитів необхідно додавати токен доступу (access token). Задачею токену доступу є автентифікація користувача з боку серверу. При авторизації за допомогою пакету @react-oauth/google користувач отримує токен доступу у форматі рядку з терміном актуальності в 60 хвилин. Після цього користувачу необхідно авторизуватись наново для отримання свіжого токену доступу. Також було вирішено зберігати токен у глобальному стані, адже токен може знадобитись у будь-якому компоненті. Ось така структура додається до деяких запитів, що потребують токен доступу:

```
const headers = {
      Authorization: `Bearer ${appState.token.value}`,
      'Content-Type': 'application/json',
   }
```

Значення **appState.token.value** – це і є токен доступу у форматі рядку, що зберігається у глобальному стані appState.

3.3.4. Огляд основних компонентів

Сторінка входу (компонент LoginPage.js)

Окрім деякої JSX-розмітки, даний компонент виконує надважливу роль. При натисканні кнопки входу, у глобальний стан переміщується токен доступу access_token та відбувається показ спливаючого повідомлення з успіхом і після цього користувач направляється на головну сторінку методом navigate(«/»):

```
const appDispatch = useContext(DispatchContext)
   const navigate = useNavigate()

   const login = useGoogleLogin({
      onSuccess: codeResponse => {
```

```
        appDispatch({ type: 'login', value: codeResponse.access_token })
        appDispatch({
          type: 'flashMessages',
          value: 'Login successfull',
          style: 'success',
        })
        navigate('/')
      },
      scope:
        'https://www.googleapis.com/auth/userinfo.profile
https://www.googleapis.com/auth/userinfo.email
https://www.googleapis.com/auth/books',
      onError: error => console.log('Login failed ', error),
    })
```

Домашня сторінка (компонент SearchPage.js).

Схематично, було окреслено будову домашньої сторінки на рисунку 3.7.

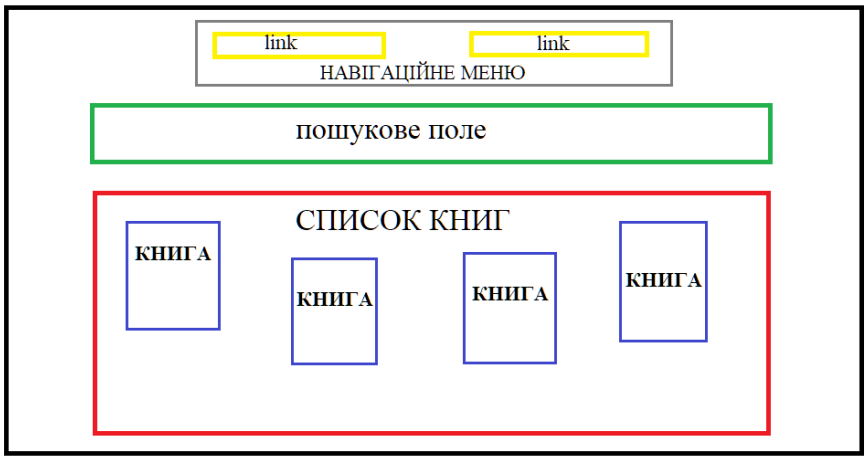

Рис. 3.8. Схематична будова домашньої сторінки

Як видно з рисунку 111 домашня сторінка складатиметься з навігаційного меню, пошукового поля та області зі списком книг. Це найбільш оптимальна та інтуїтивно зрозуміла користувачеві структура.

Було розроблено ось таку структуру у вигляді JSX-коду:

```
<div className={styles.page}>
          <div className={styles.main}>
            <form onSubmit={trigger} className={styles.form}>
```

```
            <div className={styles.input_container}>
              <input
                 className={styles.input}
                 onChange={e => setSearchTerm(e.target.value)}
                 onKeyDown={e => {
                    if (e.key === 'Enter') {
                       trigger(e)
                    }
                 }}
                 type='text'
                 placeholder='Type a search query here!'
                 value={searchTerm}
              />
              <button
                 className={clsx(styles.cross, searchTerm && styles.active)}
                 onClick={e => {
                    e.preventDefault()
                    setSearchTerm('')
                 }}
              >
                 X
              </button>
              <button className={styles.submit} type='submit'>
                 <img src={search} alt='Search' />
              </button>
            </div>
            <div className={styles.submit_small}>
              <button type='submit'>
                 <img src={search} alt='Search' />
              </button>
            </div>
          </form>
        </div>

        {welcomePage && (
          <div className={styles.image}>
            <img src={search_book}></img>
          </div>
        )}
        {isLoading && <LoadingIcon />}
```

```
        {!isLoading && !welcomePage && (
          <div className={styles.cards}>
            {results
              ? results.map(book => {
                  return (
                    <CardItem
                      key={book.id}
                      id={book.id}
                      image={book.volumeInfo.imageLinks?.smallThumbnail}
                      title={book.volumeInfo.title}
                      author={book.volumeInfo.authors}
                      description={book.volumeInfo.description}
                    />
                  )
                })
              : ''}
          </div>
        )}
      </div>
```

Як видно з JSX-коду, до кожного елементу JSX, наприклад контейнеру div чи елементу button було додано відповідний CSS клас для стилізації через атрибут «className». Додатково, було використано препроцесор SCSS для спрощення та зручності написання CSS-коду та принцип модульності файлів стилів.

Було прив’язано поле пошуку до стану, відповідно, у полі вводу буде зберігатись останній введений користувачем запит.

Стан welcomePage відповідає за показ при першому вході на вебсервіс картинки пошуку, а при початку пошуку вона зникає і з’являється список книг.

Стан isLoading відповідає за показ елементу завантаження (кільця). Відповідно, у JSX-коді буде показано кільце завантаження умовним оператором.

Відповідь сервера на запит, а точніше JSON-код поміщений у стан results. Відповідно робиться перевірка на наявність наповнення цього стану знаком питання «?» і після цього кожен елемент відображається методом JavaScript map(). Для кожного елементу book буде відбуватись рендер копоненту <CardItem

/> - це маленька картка книги. Програмним інтерфейсом Google Books API створено обмеження в отримання від 20 до 50 книг, тому в результаті така ж кількість буде відображена в області книг.

Сторінка перегляду книги (компонент ViewSingleBook.js).

У даному компоненті не відбувається комплексної логіки чи додаткових функцій, отож присутні лише запит на сервер та показ, відповідно, JSX-розмітки у вигляді вертикального стовпчика з такою інформацією про книгу: картинка, назва тому, автор та короткий опис. Додатково, розташовані кнопки додавання тому до полиці та кнопка читання книги онлайн.

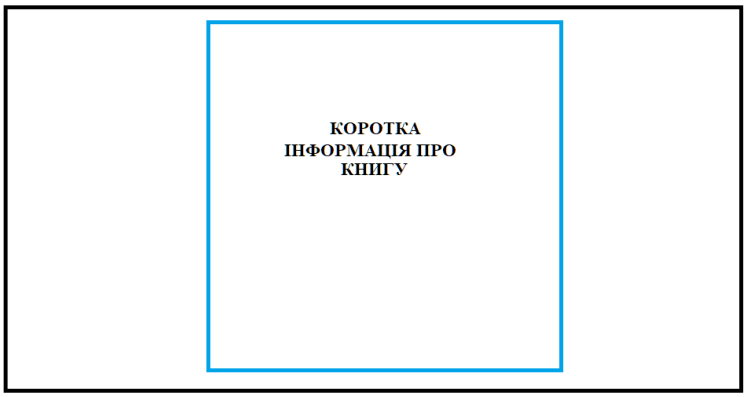

Рис. 3.9. Схематичний загальна будова сторінки перегляду книги

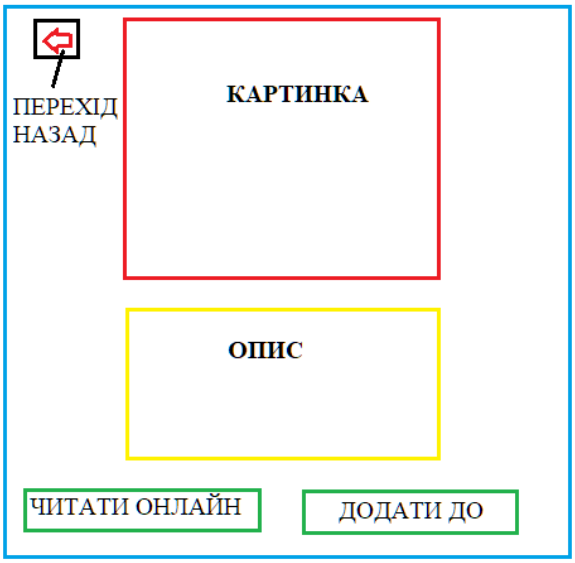

Рис. 3.10. Схематичний внутрішня будова елементу перегляду книги

Модальне вікно (компонент ModalWindow.js)

Серед незвичного це те, що модальне вікно – це не окрема сторінка, на яку переходить користувач, а елемент, що спливає поверх будь-якої сторінки, без маршрутизації. Відповідно, було зроблена така модель: у компоненті, у якому користувач буде запускати модальне вікно (<ViewSingleBook />) створюється локальний стан [modalIsOpen, setModalIsOpen]. Коли користувач натискає на кнопку «Додати до» - стан змінюється на true. Згідно з цією умовою, коли стан дорівнює true, то у JSX-розмітці відбувається рендер модального вікна. Коли користувач натискає на кнопку закриття модального вікна, відбувається передача властивостей (Props) батьківському компоненту, яка змінює стан modalIsOpen на false і модальне вікно закривається.

У самому модальному вікні був створений контейнер, у якому розташований список елементів з назвами полиць: «Favorites», «To Read», «Reading Now», «Have Read». При кліку на кожну з кнопок, відбувається запит на сервер з метою додавання відповідної книги до полиці, що прикріплена до кожної з кнопок.

Рис. 3.11. Схематична будова модального вікна

Сторінка перегляду полиць (компонент Bookshelves.js)

В своїй основі тут використовуються визначені вище концепції, такі як стан welcomePage, стан results з результатами відповіді сервера. Крім того, є такі стани як [deleteAll, setDeleteAll] та [btnDisabled, setBtnDisabled]. Перший відповідає за коректний показ кнопки видалення усіх книг з полиці, а другий – за зміну стилю та заборону на натисканні кнопки під час запиту до серверу.

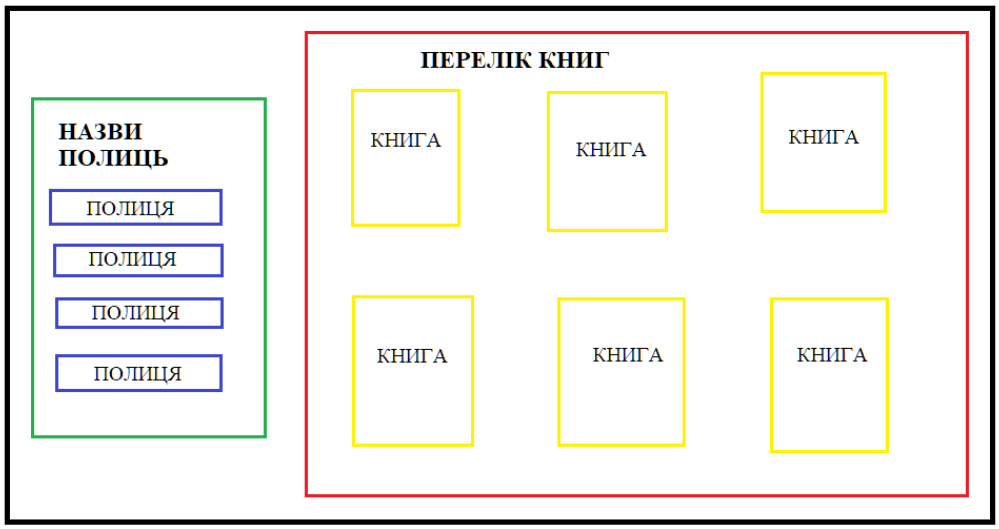

Рис. 3.12. Схематична будова сторінки категоріальних полиць

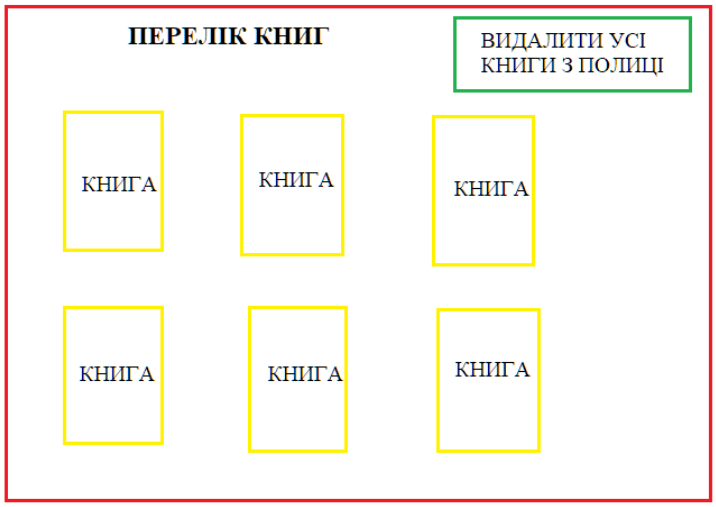

Рис. 3.13. Схематична будова частини вмісту будь-якої з категоріальних полиць

Додаткові компоненти:

- Компонент спливаючих повідомлень (FlashMessages.js);
- Компонент завантаження (кільце завантаження).

Адаптація дизайну для мобільних пристроїв

Для того, аби користувачі могли заходити на вебсервіс також з мобільних пристроїв, було адаптовано дизайн кожного з компонентів під ширину усіх можливих мобільних пристроїв.

У файлі стилів global.scss визначено такі міксини (тобто заздалегідь створені заготовки задля спрощення писання коду):

```
@mixin extraLarge {
  @media screen and (min-width: 2000px) {
    @content;
  }
}

@mixin large {
  @media screen and (min-width: 1500px) {
    @content;
  }
}

@mixin medium {
  @media screen and (max-width: 1060px) {
    @content;
  }
}
@mixin small {
  @media screen and (max-width: 700px) {
    @content;
  }
}

@mixin extraSmall {
  @media screen and (max-width: 500px) {
    @content;
```

```
    }
}

@mixin microSmall {
    @media screen and (max-width: 300px) {
        @content;
    }
}
```

Як видно з коду, за допомогою @media screen and (max-width: …) визначені основні переломні точки, у яких відбувається зміна дизайну елементів.

У файлі стилів конкретного елементу тепер варто лиш написати ключове слово @include та значення (microSmall, small, extraSmall, medium тощо) задля застосування оновлених стилів, наприклад:

```
@include small {
                    right: 10px;
                }
                @include extraSmall {
                    top: 2px;
                    right: 15px;
                    transform: scale(0.9);
                }
```

3.3.6. Висновки до розділу 3

У ході огляду процесу реалізації були висвітлені наступні етапи: процедура реєстрації у Google Cloud Console для можливості користування Google Books API ініціалізації проєкту; ініціалізація проєкту; зроблено короткий огляд загальної структури папок та файлів проєкту; описані основні концепції методики, серед яких загальна процедура запитів до серверу, отримання даних, маніпуляції з даними, кешування та авторизацію у вебсервіс.

Додатково, були продемонстровані структура основних сторінок сайту та їх JSX-розмітка.

ВИСНОВКИ

Провівши всебічний аналіз необхідності створення веб-сервісу для пошуку книг та організації їх у категоріальні полиці, було виявлено значний попит на такий інструмент. Це обумовлено зростанням інтересу до цифрового читання та потребою в зручному способі організації особистих колекцій книг. Проект може мати суттєвий соціальний і комерційний вплив за умови врахування потреб користувачів та сучасних технологічних стандартів.

З’ясовано основні вимоги користувачів до такого сервісу: естетичний і зручний інтерфейс, велика база даних книг, інтеграція з Google Books API та адаптивний дизайн. Для забезпечення успішності сервісу важливо реалізувати декілька ключових аспектів. По-перше, необхідно надати широкий вибір книг та зручний механізм їх пошуку, щоб користувачі могли легко знаходити та додавати книги до своїх колекцій. По-друге, дослідження вказує на важливість додаткових функцій, таких як система коментарів, відгуків та рекомендацій, що сприятимуть створенню активної спільноти користувачів та персоналізації досвіду.

Технологічно доцільним рішенням є використання системи розподілених обчислень, архітектури «клієнт-сервер» та моделі «товстого» клієнта. Для написання програмного коду вибрано мову Javascript з використанням бібліотеки React, яка забезпечує швидкодію та має широку підтримку спільноти. Для роботи з великою базою даних книг інтегровано Google Books API, що забезпечує простоту у використанні та підтримку авторизації через Google OAuth2.

Процес реалізації проекту включає декілька етапів: реєстрацію у Google Cloud Console для користування Google Books API, ініціалізацію проекту, організацію структури папок та файлів, та налаштування основних сторінок сайту з їх JSX-розміткою. Основні концепції методики включають процедуру запитів до серверу, отримання і маніпуляцію даними, кешування та авторизацію.

Загалом, результати дослідження підкреслюють важливість створення функціонального та інтерактивного веб-сервісу для пошуку книг та організації їх колекцій, що задовольнить потреби користувачів та відповідатиме сучасним вимогам ринку.

СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ БІБЛІОГРАФІЧНИХ ДЖЕРЕЛ

1. Бородкін Г. О. Інженерія програмного забезпечення / Г. О. Бородкін, І. Л. Бородкіна: навч. посіб. Київ: Центр навчальної літератури, 2018. 204 с.
2. Reading Culture Challenges For Today's World. URL: https://slimkm.com/blog/reading-culture-challenges-for-todays-world (дата звернення: 23.04.2024).
3. Top 9 Book APIs in 2024. URL: https://isbndb.com/blog/book-api. 24.04.2024.
4. Introduction to OAuth 2.0. URL: https://cloud.google.com/apigee/docs/api-platform/security/oauth/oauth-introduction (дата звернення: 25.04.2024).
5. Архітектура клієнт-сервер. URL: http://educational.mariroz.com/InformTechVInfrastrRynku/lect/lect8.pdf (дата звернення: 26.04.2024).
6. React Introduction. URL: https://www.geeksforgeeks.org/reactjs-introduction/ (дата звернення: 26.04.2024).
7. Visual Studio Code 101#1: Introduction and Get start. URL: https://medium.com/@minghunghsieh/vcs-visual-studio-code-101-introduction-and-installation-ee429fc664db (дата звернення: 26.04.2024).

ДОДАТКИ

Додаток А

Фрагмент коду LoginPage.js

```
return (
      <Page title='Login'>
        <div className={styles.page}>
          <div className={styles.img}>
            <img src={login_poster} alt='' />
          </div>
          <div className={styles.button}>
            <button onClick={login}>Enter</button>
          </div>
        </div>
      </Page>
   )
}

export default LoginPage
```

Фрагмент коду CardItem.js

```
return (
    <>
      <div className={styles.item}>
        <Link
          to={
            props.bookshelves
              ? `/bookshelves/${bookshelf_type}/${id}`
              : `/search/${searchValue}/${id}`
          }
        >
          <div className={styles.body}>


      <div className={styles.top}>
                <img src={props.image || default_image} />
              </div>
              <div className={styles.content}>
                <div className={styles.title}>
                  <p>{props.title}</p>
                </div>
                <div className={styles.author}>
                  <p>{props.author}</p>
                </div>
              </div>
            </div>
          </Link>
        </div>
    </>
  )
```

Фрагмент коду ViewSingleBook.js

```
return (
      <Page title={book.volumeInfo.title}>
        {modalIsOpen ? <ModalWindow closeModal={closeModal} /> : ''}
        {props.bookshelves ? (
          <button
            onClick={deleteVolumeHandler}
            disabled={btnDisabled}
            className={clsx(styles.btn, btnDisabled && styles.btn_disabled)}
          >
            Delete from bookshelf
          </button>
        ) : (
          ''


        )}
      {!isLoading && (
          <div className={clsx(styles.body, closeComponent && styles.close)}
ref={book_body}>
            <div className={styles.book}>
              <div className={styles.top}>
                <Link
                  to={props.bookshelves ? `/bookshelves/${bookshelf_type}` :
`/search/${searchValue}`}
                  className={styles.go_back}
                >
                  <img src={arrow_left} alt='Go back' />
                </Link>
                {book.volumeInfo.imageLinks?.thumbnail && (
                  <div className={styles.image}>
                    <img src={book.volumeInfo.imageLinks?.thumbnail} />
                  </div>
                )}
                {book.volumeInfo.imageLinks?.thumbnail == undefined && (
                  <img className={styles.image} src={default_big_image}
alt="Book's thumbnail" />
                )}
```

```
                {!props.bookshelves && book.volumeInfo.title && (
                  <button className={styles.button} onClick={openModal}>
                    Add to bookshelf
                  </button>
                )}
                <div className={styles.go_back_bottom_container}>
                  <Link
                    to={
                      props.bookshelves ? `/bookshelves/${bookshelf_type}` :
`/search/${searchValue}`
                    }
                    className={styles.go_back_bottom}                    >
                    <img src={arrow_left} alt='Go back' />
                  </Link>
                </div>
              </div>
              <div className={styles.content}>
                <div className={styles.title}>
                  <h2>{book.volumeInfo.title}</h2>
                </div>

                <h3 className={styles.author}>{book.volumeInfo.authors}</h3>
                <p
className={styles.description}>{book.volumeInfo.description}</p>
                <button className={styles.read_online_btn} onClick={() =>
readOnlineHandler(book)}>
                  Read online
                </button>
                {!props.bookshelves && (
                  <button className={styles.button__bottom}
onClick={openModal}>
                    Add to bookshelf
                  </button>
                )}
              </div>
            </div>
          </div>
```

```
        )}
    </Page>
)
```

Фрагмент коду ModalWindow.js

```
return (
      <div className={styles.overlay}>
        {isLoading && <LoadingIcon modalWindow={true} />}
        {!isLoading && (
          <div className={clsx(styles.body, closeComponent && styles.close)}>
            <div className={styles.content}>
              <span
                 className={styles.close_btn}
                 onClick={() => {
                    setCloseComponent(true)
                    setTimeout(() => props.closeModal(), 400)
                 }}
              >
                 <img src={cross} alt='Close' />
              </span>
              <p className={styles.text}>Choose your bookshelf to add</p>
              <ul className={styles.list}>
                 <li
                    className={styles.item}
                    onClick={() => {
                       addVolumeHandler('0', 'Favorites')
                    }}
                 >
                    <button
                       className={clsx(styles.button, active['favorites'] &&
styles.active)}
                       disabled={active['favorites']}
                       onClick={() =>
                          setActive(() => ({
                             ['favorites']: 1,
                          }))
                       }                    >

                       <svg xmlns='http://www.w3.org/2000/svg' viewBox='0 0 24
24' fill='none'>
                          <path
```

```
d='M11.993 5.09691C11.0387 4.25883 9.78328 3.75 8.40796 3.75C5.42122 3.75 3
6.1497 3 9.10988C3 10.473 3.50639 11.7242 4.35199 12.67L12 20.25L19.4216
12.8944L19.641 12.6631C20.4866 11.7172 21 10.473 21 9.10988C21 6.1497
18.5788 3.75 15.592 3.75C14.2167 3.75 12.9613 4.25883 12.007 5.09692L12
5.08998L11.993 5.09691ZM12 7.09938L12.0549 7.14755L12.9079
6.30208L12.9968 6.22399C13.6868 5.61806 14.5932 5.25 15.592 5.25C17.763 5.25
19.5 6.99073 19.5 9.10988C19.5 10.0813 19.1385 10.9674 18.5363
11.6481L18.3492 11.8453L12 18.1381L5.44274 11.6391C4.85393 10.9658 4.5
10.0809 4.5 9.10988C4.5 6.99073 6.23699 5.25 8.40796 5.25C9.40675 5.25 10.3132
5.61806 11.0032 6.22398L11.0921 6.30203L11.9452 7.14752L12 7.09938Z'
                                fill='#fff'
                            ></path>
                        </svg>
                        <p>Favorites</p>
                    </button>
                </li>
                <li
                    className={styles.item}
                    onClick={() => {
                        addVolumeHandler('2', 'To Read')
                    }}
                >
                    <button
                        className={clsx(styles.button, active['to-read'] &&
styles.active)}
                        disabled={active['to-read']}
                        onClick={() =>
                            setActive(() => ({
                                ['to-read']: 1,
                            }))
                        }
                    >

                        <svg
                            width='25px'
                            height='25px'
                            viewBox='0 0 19 19'
                            xmlns='http://www.w3.org/2000/svg'
```

```
                        >
                            <path
                                fill='#fff'
                                fillRule='evenodd'
                                d='M161.960546,159.843246 L164.399107,161.251151
C164.637153,161.388586 164.71416,161.70086 164.580127,161.933013
C164.442056,162.172159 164.144067,162.258604 163.899107,162.117176
L161.419233,160.68542 C161.165323,160.8826 160.846372,161 160.5,161
C159.671573,161 159,160.328427 159,159.5 C159,158.846891
159.417404,158.291271 160,158.085353 L160,153.503423 C160,153.22539
160.231934,153 160.5,153 C160.776142,153 161,153.232903 161,153.503423
L161,158.085353 C161.582596,158.291271 162,158.846891 162,159.5
C162,159.6181 161.986351,159.733013 161.960546,159.843246 Z M160.5,169
C165.746705,169 170,164.746705 170,159.5 C170,154.253295 165.746705,150
160.5,150 C155.253295,150 151,154.253295 151,159.5 C151,164.746705
155.253295,169 160.5,169 Z M160.5,168 C165.19442,168 169,164.19442 169,159.5
C169,154.80558 165.19442,151 160.5,151 C155.80558,151 152,154.80558
152,159.5 C152,164.19442 155.80558,168 160.5,168 Z'
                                transform='translate(-151 -150)'
                            />
                        </svg>
                        <p>To Read</p>
                    </button>
                </li>
                <li
                    className={styles.item}
                    onClick={() => {
                        addVolumeHandler('3', 'Reading Now')
                    }}

                >
                    <button

                        className={clsx(styles.button, active['reading-now'] &&
styles.active)}
                        disabled={active['reading-now']}
```

```
                        onClick={() =>
                            setActive(() => ({
                                ['reading-now']: 1,
                            }))
                        }
                    >
                        <svg
                            width='25px'
                            height='25px'
                            viewBox='0 0 16 16'
                            xmlns='http://www.w3.org/2000/svg'
                            fill='none'
                        >
                            <g fill='#fff'>
                                <path d='M6.25 5a.75.75 0 01.75.75v4.5a.75.75 0 01-1.5
0v-4.5A.75.75 0 016.25 5zM10.5 5.75a.75.75 0 00-1.5 0v4.5a.75.75 0 001.5 0v-4.5z'
/>

                                <path
                                    fillRule='evenodd'
                                    d='M0 8a8 8 0 1116 0A8 8 0 010 8zm8-6.5a6.5 6.5 0
100 13 6.5 6.5 0 000-13z'
                                    clipRule='evenodd'
                                />
                            </g>
                        </svg>
                        <p>Reading Now</p>
                    </button>
                </li>
                <li

                    className={styles.item}
                    onClick={() => {
                        addVolumeHandler('4', 'Have Read')
                    }}
                >
                    <button
```

```
                        className={clsx(styles.button, active['have-read'] &&
styles.active)}
                        disabled={active['have-read']}
                        onClick={() =>
                            setActive(() => ({
                                ['have-read']: 1,
                            }))
                        }
                    >
                        <svg fill='#fff' viewBox='0 0 24 24'
xmlns='http://www.w3.org/2000/svg'>
                            <path d='M7.493,22.862a1,1,0,0,0,1.244-.186l11-
12A1,1,0,0,0,19,9H13.133l.859-6.876a1,1,0,0,0-1.8-.712l-
8,11A1,1,0,0,0,5,14H9.612l-2.56,7.684A1,1,0,0,0,7.493,22.862ZM6.964,12l4.562-
6.273-.518,4.149A1,1,0,0,0,12,11h4.727l-6.295,6.867,1.516-4.551A1,1,0,0,0,11,12Z'
/>
                        </svg>
                        <p>Have Read</p>
                    </button>
                </li>
            </ul>
        </div>
    </div>
    )}
  </div>
 )
}

export default ModalWindow
```

Фрагмент коду Bookshelves.js

```
return (
      <Page title={bookshelf_type ? `${bookshelf_type}` : 'Bookshelves'}>
        <div className={styles.body}>
          <div className={styles.navbar}>
            <ul className={styles.links}>
              <li className={styles.item}>
                <NavLink
                  to='/bookshelves/favorites'
                  end
                  onClick={() =>
                    setActiveLink(prevState => ({
                      ...prevState,
                      ['favorites']: prevState['favorites'] + 1,
                    }))
                  }
                  className={clsx(
                    styles.link,
                    location.pathname.startsWith('/bookshelves/favorites') &&
styles.active
                  )}
                >
                  <svg xmlns='http://www.w3.org/2000/svg' viewBox='0 0 24 24'
fill='none'>
                    <path
                      d='M11.993 5.09691C11.0387 4.25883 9.78328 3.75
8.40796 3.75C5.42122 3.75 3 6.1497 3 9.10988C3 10.473 3.50639 11.7242 4.35199
12.67L12 20.25L19.4216 12.8944L19.641 12.6631C20.4866 11.7172 21 10.473 21
9.10988C21 6.1497 18.5788 3.75 15.592 3.75C14.2167 3.75 12.9613 4.25883 12.007
5.09692L12 5.08998L11.993 5.09691ZM12 7.09938L12.0549 7.14755L12.9079
6.30208L12.9968 6.22399C13.6868 5.61806 14.5932 5.25 15.592 5.25C17.763 5.25
19.5 6.99073 19.5 9.10988C19.5 10.0813 19.1385 10.9674 18.5363
11.6481L18.3492 11.8453L12 18.1381L5.44274 11.6391C4.85393 10.9658 4.5
10.0809 4.5 9.10988C4.5 6.99073 6.23699 5.25 8.40796 5.25C9.40675 5.25 10.3132
5.61806 11.0032 6.22398L11.0921 6.30203L11.9452 7.14752L12 7.09938Z'
```

```
                        fill='#fff'
                      ></path>
                    </svg>
                    <p>Favorites</p>
                  </NavLink>
                </li>
                <li className={styles.item}>
                  <NavLink
                    to='/bookshelves/to-read'
                    onClick={() =>
                      setActiveLink(prevState => ({
                        ...prevState,
                        ['to-read']: prevState['to-read'] + 1,
                      }))
                    }
                    className={clsx(
                      styles.link,
                      location.pathname.startsWith('/bookshelves/to-read') &&
styles.active
                    )}
                  >
                    <svg
                      width='25px'
                      height='25px'
                      viewBox='0 0 19 19'
                      xmlns='http://www.w3.org/2000/svg'
                    >
                      <path
                        fill='#fff'
                        fillRule='evenodd'
                        d='M161.960546,159.843246 L164.399107,161.251151
C164.637153,161.388586 164.71416,161.70086 164.580127,161.933013
C164.442056,162.172159 164.144067,162.258604 163.899107,162.117176
L161.419233,160.68542 C161.165323,160.8826 160.846372,161 160.5,161
C159.671573,161 159,160.328427 159,159.5 C159,158.846891
159.417404,158.291271 160,158.085353 L160,153.503423 C160,153.22539
160.231934,153 160.5,153 C160.776142,153 161,153.232903
```

```
161,153.503423 L161,158.085353 C161.582596,158.291271 162,158.846891
162,159.5 C162,159.6181 161.986351,159.733013 161.960546,159.843246 Z
M160.5,169 C165.746705,169 170,164.746705 170,159.5 C170,154.253295
165.746705,150 160.5,150 C155.253295,150 151,154.253295 151,159.5
C151,164.746705 155.253295,169 160.5,169 Z M160.5,168 C165.19442,168
169,164.19442 169,159.5 C169,154.80558 165.19442,151 160.5,151
C155.80558,151 152,154.80558 152,159.5 C152,164.19442 155.80558,168
160.5,168 Z'
                            transform='translate(-151 -150)'
                        />
                    </svg>
                    <p>To Read</p>
                </NavLink>
            </li>
            <li className={styles.item}>
                <NavLink
                    to='/bookshelves/reading-now'
                    onClick={() =>
                        setActiveLink(prevState => ({
                            ...prevState,
                            ['reading-now']: prevState['reading-now'] + 1,
                        }))
                    }
                    className={clsx(
                        styles.link,
                        location.pathname.startsWith('/bookshelves/reading-now') &&
styles.active
                    )}
                >
                    <svg
                        width='25px'
                        height='25px'
                        viewBox='0 0 16 16'
                        xmlns='http://www.w3.org/2000/svg'
                        fill='none'
```

```
                    >
                        <g fill='#fff'>
                            <path d='M6.25 5a.75.75 0 01.75.75v4.5a.75.75 0 01-1.5
0v-4.5A.75.75 0 016.25 5zM10.5 5.75a.75.75 0 00-1.5 0v4.5a.75.75 0 001.5 0v-4.5z'
/>

                            <path
                                fillRule='evenodd'
                                d='M0 8a8 8 0 1116 0A8 8 0 010 8zm8-6.5a6.5 6.5 0 100
13 6.5 6.5 0 000-13z'
                                clipRule='evenodd'
                            />
                        </g>
                    </svg>
                    <p>Reading Now</p>
                </NavLink>
            </li>
            <li className={styles.item}>
                <NavLink
                    to='/bookshelves/have-read'
                    onClick={() =>
                        setActiveLink(prevState => ({
                            ...prevState,
                            ['have-read']: prevState['have-read'] + 1,
                        }))
                    }
                    className={clsx(
                        styles.link,
                        location.pathname.startsWith('/bookshelves/have-read') &&
styles.active
                    )}
                >
                    <svg fill='#fff' viewBox='0 0 24 24'
xmlns='http://www.w3.org/2000/svg'>
                        <path d='M7.493,22.862a1,1,0,0,0,1.244-.186l11-
12A1,1,0,0,0,19,9H13.133l.859-6.876a1,1,0,0,0-1.8-.712l-
```

```
8,11A1,1,0,0,0,5,14H9.612l-2.56,7.684A1,1,0,0,0,7.493,22.862ZM6.964,12l4.562-6.273-
.518,4.149A1,1,0,0,0,12,11h4.727l-6.295,6.867,1.516-4.551A1,1,0,0,0,11,12Z' />
                        </svg>
                        <p>Have Read</p>
                    </NavLink>
                </li>
            </ul>
        </div>

        <div
            className={clsx({
                [styles.content]: true,
                [styles.content_passive]: welcomePage || !result,
            })}
        >
            {welcomePage ? (
                <div className={styles.icon}>
                    <img src={bookshelf} alt='Bookshelf icon' />
                </div>
            ) : (
                ''
            )}
            {isLoading && !welcomePage ? <LoadingIcon bookshelves={true} /> :
''}
            {deleteAll && result ? (
                <button
                    onClick={deleteAllHandler}
                    disabled={btnDisabled}
                    className={clsx(styles.btn, btnDisabled && styles.btn_disabled)}
                >
                    Delete all
                </button>
            ) : (
                ''
            )}
            <div className={clsx('', result.length > 0 && styles.cards)}>
```

```
                {result &&
                    result.map(book => {
                        return (

                            <CardItem
                                key={book.id}
                                id={book.id}
                                title={book.volumeInfo.title}
                                image={book.volumeInfo.imageLinks?.smallThumbnail}
                                author={book.volumeInfo.authors}
                                bookshelves={true}
                            />
                        )
                    })}
                </div>
                {!welcomePage && !result && (
                    <div className={styles.text}>
                        <p>You haven't already added books on that bookshelf.</p>
                    </div>
                )}
            </div>
        </div>
    </Page>
  )
}

export default Bookshelves
```