МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
_____Аліна САВЧЕНКО

"____"_____2021 р.

ДИПЛОМНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИЦІ ОСВІТНЬОГО СТУПЕНЯ
"МАГІСТР"

ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ "ІНФОРМАЦІЙНІ УПРАВЛЯЮЧІ СИСТЕМИ ТА ТЕХНОЛОГІЇ"

Тема: "Веб додаток електронної бібліотеки для пошуку книжок"

Виконавиця: Серьожко Ганна Сергіївна

Керівник: проф. Моржов Володимир Іванович

Нормоконтролер: _____ Ігор РАЙЧЕВ

Київ - 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра Комп'ютерних інформаційних технологій

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

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Аліна САВЧЕНКО
«____»_____2021р.

ЗАВДАННЯ

на виконання дипломної роботи студентки

Серьожко Ганни Сергіївни
(прізвище, ім'я, по батькові)

1. Тема роботи: «Веб додаток електронної бібліотеки для пошуку книжок» затверджена наказом . ректора від 12.10.2021 за № 2228/ст.

2. Термін виконання роботи: з 12.09.2021 по 31.12.2021.

3. Вихідні дані до роботи: Проаналізувати аналоги додатків електронних бібліотек для пошуку книжок в інтернет: Google Books, e-book. Проаналізувати Google Books API яке надає сервіс Google. Створені заходи повинні містити таку інформацію: список книжок по запиту користувача, назва книг, короткий опис, жанр, автор, видавництво, відгуки, варіанти купівлі або замовлення .

4. Зміст пояснювальної записки: вступ, аналіз предметної області, аналіз вимог до додатку, постановка та планування задач, проектування архітектури WEB додатку, розробка додатку, тестування компонентів, модулів та інтерфейсу висновки, список використаних джерел..

5. Перелік обов'язкового ілюстративного матеріалу: слайди, презентація.

6. Календарний план-графік

№ п/п	Завдання	Термін виконання	Підпис керівника
1.	Отримання завдання на дипломну роботу та побудова плану-графіку виконання робіт.	12.09.2021 – 15.09.2021	
2.	Огляд, аналіз вимог та постановка задачі	16.09.2021 – 19.09.2021	
3.	Проектування архітектури веб додатку, створення декомпозиції задачі.	20.09.2021 – 30.09.2021	
4.	Вибір технології та інструментів для розробки додатку	25.10.2021 – 01.10.2021	
5.	Написання Розділу 1 дипломної роботи.	01.10.2021 – 22.10.2021	
6.	Розробка та реалізація програми з вибраними інструментами. Написання тестів та тестування додатку. Написання Розділу 2 дипломної роботи.	22.10.2021 – 17.11.2021	
7.	Написання Розділу 3 дипломної роботи. Завершення створення пояснювальної записки дипломної роботи.	18.11.2021 – 01.12.2021	
8.	Оформлення та друк пояснювальної записки.	02.12.2021 – 11.12.2021	
9.	Створення презентації, доповіді та підготовка до захисту дипломної роботи	12.12.2021 – 20.12.2021	

7. Дата видачі завдання: 12.10.2021р.

Керівник дипломної роботи _____ Володимир МОРЖОВ
(підпис керівника)

Завдання прийняла до виконання _____ Ганна СЕРЬОЖКО
(підпис випускниці)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи “Веб додаток електронної бібліотеки для пошуку книжок” складається із вступу, чотирьох розділів, загальних висновків, списку використаних джерел і містить 75 сторінок тексту, 20 рисунків та 5 таблиць. Список використаних джерел містить 10 найменувань.

Мета дипломної роботи – проведення дослідження про існуючі веб додатки для пошуку та завантаження книг та журналів з електронних бібліотек. Провести порівняння між існуючими електронними бібліотеками для пошуку та завантаження книг, відмітити переваги та недоліки у кожного. Провести аналіз необхідних даних та параметрів для пошуку книг та на основі цих даних імплементувати додаток у вигляді електронної бібліотеки.

Предмет дослідження – програмне забезпечення, яке містить всі необхідні компоненти для створення повноцінного додатку зі зручним інтерфейсом для пошуку та завантаження книг.

Об’єкт дослідження – є архітектура додатку з використання шаблонів проектування які були використання при створенні програмного забезпечення.

Компоненти, сервіси та модулі які були створені є універсальними та можуть модифіковувати та використовуватись у інших додатках

Результат роботи – Розроблено веб сервіс електронної бібліотеки для пошуку книг з використанням стороннього API.

Ключові слова: ДОДАТОК, СЕРВІС, ЕЛЕКТРОННА БІБЛІОТЕКА, КОМПОНЕНТИ, ІНФОРМАЦІЙНА СИСТЕМА, КОРИСТУВАЦЬКИЙ ІНТЕРФЕЙС.

ЗМІСТ

ВСТУП 6
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 8
1.1. Актуальність проблеми 8
1.2 Аналіз аналогів 9
Висновок до розділу 1 19
РОЗДІЛ 2. ПОСТАНОВКА ЗАДАЧІ ТА МЕТОДИ ДОСЛІДЖЕННЯ 20
2.1 Мета та задачі дослідження 20
2.2 Методи дослідження 23
Висновок до розділу 2 41
РОЗДІЛ 3. ПРОЕКТУВАННЯ ВЕБ ДОДАТКУ 42
3.1 Шаблони проектування 42
3.2 Етапи проектування додатку 44
3.3 Модель варіантів використання 46
3.4 Проектування користувацького інтерфейсу 47
Висновок до розділу 3 55
РОЗДІЛ 4. РОЗРОБКА ВЕБ ДОДАТКУ 56
4.1 Архітектура проекту 56
4.2 Структура проекту 59
4.3 Розробка функціоналу 60
4.4 Розробка веб доступності додатку 64
4.5 Аналіз метрик продуктивності додатку 67
4.6 Тестування компонентів 68
Висновок до розділу 4 74
ВИСНОВКИ 75
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 76

ВСТУП

Сучасні інформаційні технології дозволили прискорити процеси генерації і формування нових інформаційних ресурсів і створили базу для інтенсивного росту інформації. У теперішній час кількість інформаційних ресурсів щороку подвоюється. Темпи росту інформаційних ресурсів перекрили темпи зростання інформаційних потреб людини, які обмежені і наближаються до своєї межі.

Людство стикнулось з новою проблемою – проблемою засвоєння ними лавино зростаючих ресурсів. Таким чином, наступила епоха інформаційного насичення, чи епоха інформаційної кризи. Вирішення цієї проблеми можливе лише за рахунок науково-технічного прогресу.

Паралельно ми спостерігаємо стрімкий ріст розробки програмних забезпечень, WEB та мобільних додатків які виконують роль зв'язуючого шару між існуючою інформацією та користувачами.

Окремою проблемою стала складність використання та розуміння технології звичайними користувачами, тому основна мета даної дипломної роботи - адаптувати інформацію яку отримуємо від сервера та преподнести до користувачів у зручному та читабельному вигляді.

У процесі аналізу та дослідження існуючих додатків для пошуку електронних книг для було помічено багато ключових проблем які мають майже всі застосунки.

– На сайтах розташовано велика кількість зайвої інформації яка не стосується конкретної книги або журналу - це відволікає користувачів від їх основної мети - вибрати книгу та візуально заважає сконцентруватись на виборі книги і перезавантажує увагу людини.

– Приховування ключових факторів - важливої інформації про книгу, автора та видавництво, де її можно замовити та її ціну. Компонент з цією інформацією не завжди виділенний та часто прихований від користувача.

В даній роботі буде описано рішення визначених проблем, та створення WEB сервісу взаємодії між користувачами та API.

Для досягнення поставленої мети було поставлено такі задачі, як дослідити необхідну інформацію по пошуку книги, дослідити схожі застосунки, розробити архітектуру та проектну частину додатку, макет та компоненти користувацького інтерфейсу. Написати вихідний код та провести тестування програмного продукту. Та продумати план про подальший розвиток та підтримку додатку.

Об’єктом дослідження виступає процес пошуку бажаної книги з характеристиками які бажає користувач.

Предметом дослідження є API (Прикладний програмний інтерфейс) з даними про існуючі книги в електронній системі, наданий сервісом Google Books.

В даній роботі набуло подальшого розвитку використання інтеграції з допоміжними сервісами для завантаження книги на пристрій або перегляд у реальному часі.

В результаті виконання роботи, завдяки зручному користувацькому інтерфейсу пасажири мають змогу швидко переглянути всі варіанти книг що існують за їх запитом, вибрати бажану та передивись всю необхідну інформацію про неї. Бачити всі ресурси де її можно завантажити або подивитися - що є практичним значенням даної роботи.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Актуальність проблеми

На сьогоднішній день можно спостерігати швидкий ріст кількості інформації в інтернет середовищі, навіть можна точно стверджувати, що відбувається інформаційний вибух, тобто постійне збільшення швидкості і обсягів публікацій (обсягу інформації) в масштабах всієї планети, який з кожним роком тільки збільшується.

Сучасні тренди розвитку інформаційного суспільства характеризуються стрімкими темпами збільшення даних. Обсяг інформації в світі зростає щорічно на 30%. В середньому на людину в рік у світі виробляється 2,5·108 байт, що безперечно є величезною кількістю інформації.

Інтернет містить незрівнянно більше інформації, ніж пропонують усі існуючі бібліотеки разом узяті, проте кількісна перевага в цьому випадку не приводить до переваги якісної – неструктурована інформація сумнівного змісту унеможливлює ведення релевантного пошуку.

Тільки за останні 15 років кількість онлайн-відвідувань значно перевищує кількість користувачів, які фізично відвідують бібліотеку (зрозуміло, що йдеться про бібліотечно-інформаційні установи, чиї сайти чи ресурси є в інтернет-просторі). Це відбувається за рахунок нарощування доступного бібліотечного інтернет-ресурсу, представленого на сайтах бібліотек, – електронні каталоги, бази даних, оцифровані раритети, електронні версії видань тощо.

Дуже швидкі темпи розвитку електронних бібліотек та електронної екосистеми зберігання та організації інформації призвели до того що користувачам стало значно складніше користуватись ресурсами та знаходити інформацію за власними запитами.

Однією із проблем є складність та нерозуміння веб інтерфесів користувачами. Великі об'єми інформації на сайтах не дають можливості користувачу концентруватися та чітко сприйняти всю інформацію яка подається на сайті. Користувачі втрачають розуміння з користувацьким інтерфейсом що призводить до складності взаємодії користувача з системою.

Попит на електронні джерела сьогодні дуже великий та простий та зрозумілий вигляд додатків призводить до швидкої вирішення проблем користувачів.

Клієнтська частина додатків є дуже важливою складовою частиною програмного забезпечення, так як вона відповідає за взаємодії програми з користувачем та формує враження про використання програми.

1.2 Аналіз аналогів

На першій фазі роботи над проектом було проведено аналіз аналогів існуючих веб додатків для пошуку книг, під час якого було виявлено переваги та недоліки існуючих сервісів.

На сьогоднішній день можна спостерігати стрімке зростання інтересу до неформальної освіти, отримання знань та висококваліфікованих спеціалістів в цілому. Зростає і попит на отримання інформації онлайн з допомоги сервісів та електронних бібліотек, а в майбутньому ця тенденція буде тільки посилюватися, оскільки знання – найважливіший ресурс у сучасному світі.

За останні роки пропозиції тільки зросли, пов'язано це зі збільшенням інформації в інтернеті та переходу роботи до онлайн простору.

Всі аналоги мають одну спільну мету, вони допомагають знаходити книги та журнали, надають коротку інформацію про книгу, деталі як її можно отримати або переглянути.

Бібліотека української літератури “УкрЛіб”

Онлайн-бібліотека УкрЛіб була заснована на зламі тисячоліть - в далекому 2000-му році. Сьогодні вона позиціонує себе як найбільша в інтернеті електронна бібліотека української літератури. За допомогою цього проекту засновники намагаються популяризувати українську мову і культуру та плекати любов до рідної літератури новим поколінням.

У бібліотеці є кілька основних розділів, це “Українська література”, “Світова література”, “Навчання”, “Літературознавство” та “Матеріали російською”. У свою чергу літературні розділи мають підрозділи “Сучасна література”, “Народна творчість”, “Біографії” та “Критичні статті”.

На головній сторінці вказано “Книгу дня”, яку бібліотека пропонує почитати у цей день та розділ “Видатні діячі української літератури”, де можна ознайомитись з біографією та найкращими творами українських письменників. Також можна поцікавитись, які саме книжки зараз читають інші відвідувачі УкрЛібу. Для цього треба перейти до розділу “Рейтинг творів”, де вказані твори дня, найбільш популярні книжки та останні надходження до бібліотеки.

Книжки можна читати безпосередньо на сайті, або завантажити в найбільш прийнятному форматі: DOCX, RTF, HTML, EPUB, MOBI, PDF.

Українська електронна бібліотека “Libruk”

На ресурсі Libruk зібрано різні за жанром твори українських письменників-класиків та сучасних авторів. Всі книжки поділені на два великих розділи - українська література та світові бестселери. Також існує можливість завантажити на Libruk свої твори, що без сумніву сподобається молодим українським авторам.

На головній сторінці присутній повний перелік авторів, алфавітний каталог та такий незвичний інструмент пошуку, як “Автори за місцем народження”. В цьому режимі авторів книжок можна розшукувати на мапі України. Також для знайомства з життєвим та творчим шляхом письменників України на ресурсі викладені їх скорочені біографії.

Всі книжки доступні у найбільш популярних електронних форматах FB2, EPUB та MOBI. Останній формат включено у бібліотеку через те, що електронні книжки Amazon Kindle користуються широкою популярністю в Україні через гарне співвідношення ціна/якість. Деякі книжки можна скачати у форматах PDF або DJVU, які більше підходять для читання на настільному комп'ютері або планшеті.

Крім цього, книжки можна читати за допомогою інтегрованої онлайн-читалки, яка дозволяє налаштувати під себе відображення тексту та запам'ятовує останню прочитану сторінку.

Книжкове братство “Флібуста”

Книжкове братство Флібуста - це одна з найбільших онлайн-бібліотек в Інтернеті. Цей ресурс спеціалізується на російськомовних книжках, але на його сторінках можна знайти книги будь-якого жанру на всіх європейських мовах, включаючи українську. Всього на Флібусті 350 тисяч книжок більш ніж 130 тисяч авторів, при цьому майже 5 тисяч з них українською мовою. Щоб відшукати книжки українською мовою потрібно скористатися відповідним розділом, або вказати при пошуку код “uk” в параметрі “мова”.

Головна перевага Флібусти в порівнянні з іншими онлайн-бібліотеками полягає в тому, що на її сторінках присутні найновіші книжки. До кожного тома додається фото обкладинки, оцінки та коментарі тих, хто вже прочитав книжку. Знайти щось цікаве допоможе розширений пошук та численні рейтинги, до яких включено “Книжки дня”, “Книжки тижня”, а також “Найпопулярніші книжки”, “Найпопулярніші автори” та “Книжки, що отримали найбільше оцінок”.

Українська та зарубіжна література українською мовою “Shift Library CMS”

Метою діяльності цією електронної бібліотеки є поширення і популяризація україномовних творів серед читачів. Особлива увага приділяється книжкам нерадянського періоду, що видавались українською діаспорою, а також виданням останніх років. Пріоритетним завданням бібліотеки є надання читачам електронних книг в повному обсязі та забезпечення школярів і студентів україномовним матеріалом, який вони вивчають за програмами навчальних закладів.

Всі книги бібліотеки згруповані за наступними жанрами: Гумор, Детективи і Трилери, Дитяча література, Довідкова література, Документальна література, Домоведення (Дім і сім'я), Комп'ютери та Інтернет, Любовні романи, Наука, Освіта, Поезія, Драматургія, Пригоди, Проза, Релігія і духовність, Старовинна література, Фантастика.

На думку творців ресурсу компактність, зручність та унівесальність електронних текстів яскраво доводять, що в електронної книги велике майбутнє і українці просто зобов'язані йти в ногу з часом. Саме тому вони намагаються забезпечити зручне користування і доступ до вмісту своєї електронної бібліотеки усіма можливими способами.

Художня література українською мовою “E-bookua.org.ua”

В онлайн-бібліотеці E-bookua.org.ua зібрана велика колекція книг здебільше художнього напрямку. Сайт має такі розділи як “Класика”, “Біографії”, “Історичні романи, пригоди”, “Фантастика, фентезі”, “Детективи, боєвики”, “Любовні романи”, “Сучасна проза”, “Наука і освіта”, “Батькам і дітям”. Якщо ви віддаєте перевагу не читанню, а прослуховуванню книжок, то для вас на сайті присутній розділ “Аудіокниги”.

Публічна електронна бібліотека української художньої літератури “УКРЛІТ.ORG”

Публічна електронна бібліотека української художньої літератури УКРЛІТ.ORG розпочала свою роботу ще в 2005 році зі збірки творів кількох авторів. Відтоді бібліотека опрацьовувала і наповнювалась новими книгами. Основними розділами бібліотеки є українська література, переважно класика, тлумачні словники, транслітерація та інші твори для школярів, студентів, вчителів та широкого загалу.

Аналіз та порівняння додатків для пошуку та перегляду електронних книг

Таблиця 1.1

Основні переваги та недоліки розглянутих аналогів.

Сервіс	Переваги	Недоліки
Бібліотека української літератури “УкрЛіб”	1. Книжки можна читати безпосередньо на сайті, або завантажити в найбільш прийнятному форматі: DOCX, RTF, HTML, EPUB, MOBI, PDF. 2. На головній сторінці вказано “Книгу дня”. 3. У бібліотеці є кілька основних розділів. 4. розділ “Видатні діячі української літератури”, де можна ознайомитись з біографією та найкращими творами українських письменників.	1. Обмежена кількість книг та журналів (переважно лише українською мовою) 2. Присутня реклама на сайті 3. Немає відгуків від користувачів та мало відгуків про книги

Українська електронна бібліотека “Libruk”	1. Існує можливість завантажити на Libruk свої твори, що без сумніву сподобається молодим українським авторам. 2. На головній сторінці присутній повний перелік авторів, алфавітний каталог та такий незвичайний інструмент пошуку, як “Автори за місцем народження”. 3. Всі книжки доступні у найбільш популярних електронних форматах FB2, EPUB та MOBI. Останній формат включено у бібліотеку через те, що електронні книжки Amazon Kindle користуються широкою популярністю в Україні через гарне співвідношення ціна/якість. 4. Крім цього, книжки можна читати за допомогою інтегрованої онлайн-читалки, яка дозволяє налаштувати під себе відображення тексту	1. Лише деякі книжки можна скачати у форматах PDF та DJVU, які більше підходять для читання на настільному комп'ютері або планшеті. 2. Відсутні відгуки від читачів.

	та запам'ятовує останню прочитану сторінку. 5. Також існує можливість завантажити на Libruk свої твори, що без сумніву сподобається молодим українським авторам.	
Книжкове братерство “Флібуста”	1. Всього на Флібусті 350 тисяч книжок більш ніж 130 тисяч авторів, при цьому майже 5 тисяч з них українською мовою. 2. Головна перевага Флібусти в порівнянні з іншими онлайн-бібліотеками полягає в тому, що на її сторінках присутні найновіші книжки. 3. До кожного тома додається фото обкладинки, оцінки та коментарі тих, хто вже прочитав книжку. 4. Знайти щось цікаве допоможе розширений пошук та численні рейтинги, до яких включено “Книжки дня”, “Книжки тижня”, а також “Найпопулярніші	1. Цей ресурс спеціалізується на російськомовних книжках, але на його сторінках можна знайти книги будь-якого жанру на всіх європейських мовах, включаючи українську. 2. Щоб відшукати книжки українською мовою потрібно скористатися відповідним розділом, або вказати при пошуку код “uk” в параметрі “мова”.

	книжки”, “Найпопулярніші автори” та “Книжки, що отримали найбільше оцінок”.	
Електронна бібліотека “Українська література”	1. На цьому ресурсі є можливість отримати вільний доступ до кращих творів класичної української літератури, з якими зазвичай починають знайомство ще в школі. 2. Всі книжки електронної бібліотеки “Українська література” можна читати прямо на сайті, або завантажити в один клік в форматах DOCX, FB2, PDF, DJVU для читання на мобільному телефоні, планшеті або електронній книзі. 3. Окремо відзначимо підрозділ “ЗНО 2017”, в якому зібрані автори та їхні твори, які включені у програму Зовнішнього незалежного оцінювання з української	1. Електронна бібліотека “Українська література” орієнтована на студентів, школярів. 2. Лімітована кількість книг. 3. Відсутність журналів.

	літератури 2017-го року. 4. Для зручної роботи з матеріалами сайту всі письменники структуровані за алфавітним покажчиком.	

Українська та зарубіжна література українською мовою “Shift Library CMS”	1. Пріоритетним завданням бібліотеки є надання читачам електронних книг в повному обсязі та забезпечення школярів і студентів україномовним матеріалом, який вони вивчають за програмами навчальних закладів. 2. Всі книги бібліотеки згруповані за наступними жанрами. 3. Компактність, зручність та універсальність електронних текстів 4. Для зручної роботи з матеріалами сайту всі письменники структуровані за алфавітним покажчиком.	1. Невелика кількість зарубіжних авторів. 2. Відсутня можливість завантаження у форматі PDF. 3. Відсутність журналів.
Електронна бібліотека	1. Основною метою проекту “Електронна	1. Доступ до цих книжок мають

Національної бібліотеки України для дітей	бібліотека НБУ для дітей” є ознайомлення широкого загалу із кращими зразками зібрання бібліотеки в межах існуючого законодавства з авторського права. 2. У фонді Національної бібліотеки України для дітей є багато книжок, які збереглися в країні у одиночних примірниках і зберігаються у зібранні рідкісних і цінних видань. 3. В планах проекту включення у електронну бібліотеку сучасних книжок, що стане можливим після досягнення відповідних домовленостей з авторами та видавництвами. 4. У наявності є й малотиражні цікаві наукові дослідження, які можуть стати у нагоді сучасним вченим.	лише окремі дослідники: книгознавці, мистецтвознавці, літературознавці, художники та видавці. 2. Невелика кількість сучасних книжок та авторів.
Художня література	1. У онлайн-бібліотеці E-bookua.org.ua	1. Завантаження книг лише платне.

українською мовою “E-bookua.org.ua”	зібрана велика колекція книг здебільше художнього напряму. 2. Якщо ви віддаєте перевагу не читанню, а прослуховуванню книжок, то для вас на сайті присутній розділ “Аудіокниги”.	2. Не всі примірники мають книгу у аудіо форматі.
Публічна електронна бібліотека української художньої літератури “УКРЛІТ.ORG”	1. Основними розділами бібліотеки є українська література, переважно класика, тлумачні словники, транслітерація та інші твори для школярів, студентів, вчителів та широкого загалу.	1. Бібліотека відносно нова, створена лише у 2005 році, тома має лише невелику кількість примірників та дуже мало інформації про книги. 2. Відсутні відгуки.

Висновок до розділу 1

В першому розділі було проведено аналіз предметної області, проаналізована сучасні аналоги веб додатків для пошуку книжок та визначено переваги та недоліки.

РОЗДІЛ 2. ПОСТАНОВКА ЗАДАЧІ ТА МЕТОДИ ДОСЛІДЖЕННЯ

2.1 Мета та задачі дослідження

Метою роботи є створення клієнтської частини веб сервісу для швидкого пошуку, перегляду та завантаження або купівлі книги. Дане завдання повинне використовувати правильний підхід до проектування архітектури та програмного забезпечення, дотримання норм та основний принципів, та використання сучасних технологий.

Основною задачею є зменшити складність завантаження або перегляду книг або журналів, зменшити час пошук книги та розробити візуально-доступний інтерфейс для користувачів.

Завданнями для досягнення поставленої мети є:

- Виконати аналіз всіх доступних сервісів та додатків для прошуку електронних книг. Проаналізувати та дослідити які кроки потрібно зробити користувачу для завершення завантаження книги. Дослідити фактори які відволікають користувача від здійснення пошуку книги. Визначати слабкі та сильні місця кожного іх сервісів.
- Дослідити API, які надають інформацію про електронні книги.
- Визначити функціональні вимоги до проектування архітектури додатку. Зробити декомпозицію задач за розбити їх на підзадачі.
- Зробити план розробки, тестування, та подальшої підтримки програмного продукту та всіх компонентів.
- Створити архітектурну частину додатку.

- Створити макет користувацького інтерфейсу відповідний до вимог та бажань користувачів. Створити бібліотеку з універсальних компонентів.
- Реалізувати програмний продукт.
- Провести тестування програмного продукту у різних браузерах та на різних пристроях.
- Провести аналітику додатку за допомогою допоміжних інструментів.
- Протестувати доступність додатку для користувачів з особливими потребами.
- Продумати план подальшої підтримки додатку, автоматизувати оновлення бібліотек та компонентів.

Функціональні вимоги:

1. Користувач потрапляє на головну сторінку електронної бібліотеки.
2. Користувач має можливість почати пошук книги натиснувши кнопку “Почати пошук” яка знаходиться на головній сторінці на банері.
3. Якщо користувач знає ключові слова за якими він хоче здійснити пошук то він може здійснити пошук ввівши ключові слова одразу у пошук який знаходиться в шапці сайту.
4. Якщо користувач хоче здійснити пошук серед книжок по жарнам, то він може натистути на бажаних жанр у секції де знаходяться всі жанри.
5. Після натискання кнопки “Почати пошук” користувач буде перенаправленный на наступну сторінку де він може обрати певні параметри та характеристики за якими він хоче здійснити пошук.
6. У користувача є можливість здійснити швидкий пошук ввівши лише ключові слова до поля пошуку, або здійснити пошуку за расширенными параметрами.
7. У додатку має бути реалізована валідація форми пошуку. Форма з розширеними параметрами пошуку має обов’язкові поля для заповнення такі як: назва книги та ім’я автора. Необов’язковими параметрами є вибір категорії, мова, формат книги та умови завантаження.

8. У додатку має бути реалізован routing - переміщення по сторінкам в рамках одного додатку.
9. Користувач повинен мати змогу сортувати і фільтрувати книги, та шукати поїздки за певними параметрами: популярність книги та дата публікації.
10. Якщо користувач шукає книгу за певними параметрами, та оновлює сторінку, то всі характеристики повинні бути збережені після оновлення.
11. На сайті має бути реалізована пагінація, на сторінці має буду виведено 10 варіантів книг, користувач має змогу перейти на наступну або попередню сторінку.
12. Користувач повинен мати змогу застосувати фільтри для пошуку книги, такі як: мова книги, тип книги, можливість збереження книги.
13. Користувач повинен мати змогу додавати книги до обраного, переглядати обранні книги, та модифікувати список з обраними книгами.
14. На сторінці поїздки повинні бути відображені у вигляді списку, при наведенні курсора на книгу, вона повинна виділятись. При натисканні на книгу, користувач повинен перенаправлятись на окрему сторінку з детальною інформацією про книгу.
15. Якщо користувача зацікавила певна книга, то йому повинні пропонувати схожі за жанром книги.
16. Користувач має змогу переглянути або завантажити книгу за доступними форматами.

Нефункціональні вимоги:

1. Додаток повинен підтримувати сучасні браузери: Chrome, Mozilla, Opera, Safari, IE11, Edge.
2. Швидкодія додатка на мобільних пристроях має будет >90%.
3. Додаток має бути адаптивним до всіх пристроїв
4. Простий та зрозумілий користувацький інтерфейс.
5. Дотримуватись стандарту по доступності AA.

2.2 Методи дослідження

Існує декілька фреймворків та бібліотек які можно використовувати при створенні додаток у вигляді SPA.

Постановка проблеми. На сьогоднішній день існує два принципових підходи до створення веб-додатків: традиційні веб-додатки, велика частина логіки яких виконується на сервері, а також односторінкові додатки, де логіка користувача інтерфейсу виконується переважно в веб-браузері, а взаємодія з веб-сервером здійснюється головним чином через веб-API.

Сьогодні веб-сайти все більше виглядають як додатки з багатьма можливостями взаємодії, а не статичними сторінками, які ми мали близько 10 років тому. З одного боку, причини цього полягають у спробах користувачів отримувати та створювати інформацію на основі їхніх особистих характеристик та вимог. З іншого боку, власники веб-сайтів хочуть надати користувачам більш зручні інтерфейси користувача для роботи з інформацією.

Існує два основних способи побудови веб-сайтів сьогодні: більш сучасний спосіб — це **багатосторінковий додаток (MPA)** — більш традиційний спосіб, а також **односторінковий додаток (SPA)**. Почнемо з багатосторінкового підходу.

Багатосторінковий додаток (MPA) складається з декількох сторінок із статичною інформацією (текстом, зображеннями тощо) та посиланнями на інші сторінки з тим самим вмістом. Під час переходу на іншу сторінку браузер робить новий запит до сервера і знову завантажує всі ресурси, навіть ті компоненти, які повторюються на всіх сторінках (наприклад, заголовок, нижній колонтитул). Таким чином, продуктивність витрачається на завантаження тих самих елементів. Відповідно, це впливає на швидкість і продуктивність. Основними технологіями для такого типу веб-сайту є HTML та CSS. Дані технології використовувались для розробки найперших веб-сайтів, та продовжують використовуватись сьогодні, для створення сучасних веб-сайтів.

Під час розробки веб-сайту може виникати потреба в більш складних інтерактивних компонентах (наприклад, у формах). У цьому випадку ви все ще

можете використовувати традиційний спосіб розробки, але ви будете обмежені стандартними функціями HTML технології. Проте існують більш підходящі технології для нестандартних, складних та інтерактивних речей, такі, як JavaScript та AJAX. Розвиток Web 2.0 став можливим завдяки еволюції цих технологій. На сьогоднішній день, за допомогою цих технологій, можна створювати складні форми, різні анімації, діаграми та графіки з динамічним оновленням, не перезавантажувач всієї сторінки, а лише ту частину сторінки, в якій відбулися зміни.

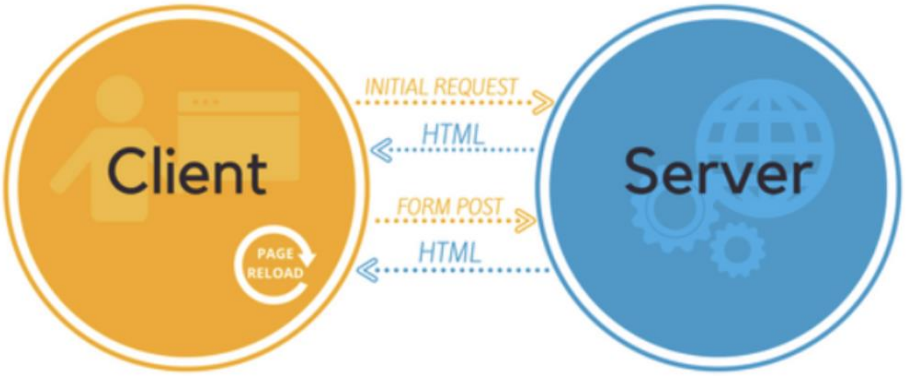

Рис. 2.1. Принцип роботи багатосторінкового додатку

Односторінкова програма (SPA) — це додаток, що працює в браузері з метою забезпечити користувачу досвід близький до користування настільною програмою. Його архітектура побудована таким чином, що при переході на нову сторінку оновлюється лише частина вмісту. Таким чином, немає необхідності повторно завантажувати ті самі елементи. Це дуже зручно для розробників та користувачів. Прикладом технології SPA є реалізований сервіс Gmail компанією Google. Для розробки SPA використовується одна з найпопулярніших мов програмування — JavaScript. На рисунку 2 наведено принцип роботи односторінкових додатків.

Невеликий веб-додаток може бути зроблено за допомогою бібліотеки jQuery. Але слід зазначити, що jQuery є дуже поганим для розробки великих проектів. Перш за все, це пов’язано зі структурою коду. Для створення складних проектів

рекомендується використовувати більш потужні фреймворки, такі як React, Angular чи Vue. Їхня архітектура дозволяє створювати гнучкі веб-додатки і до того ж вони мають в собі великий набір готових рішень, що значно спрощує процес розробки. Крім того, на основі цих фреймворків можна створювати повноцінні мобільні додатки.

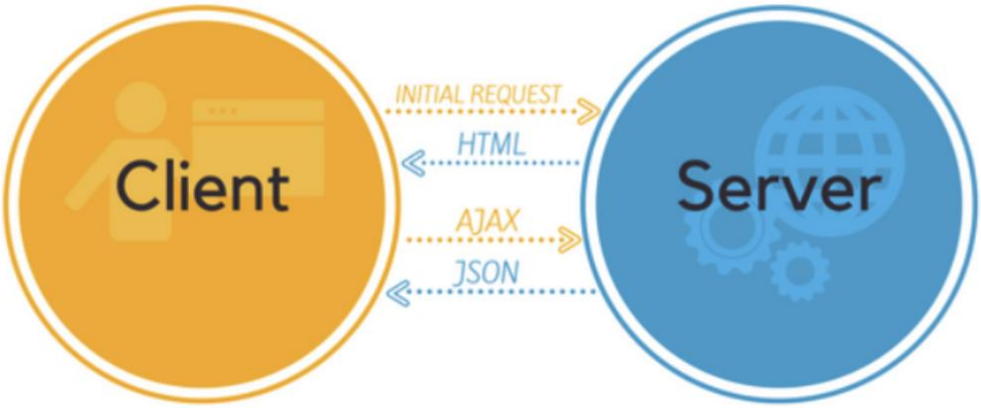

Рис. 2.2. Принцип роботи односторінкового додатку

Таблиця 2.1

Основні переваги та недоліки розглянутих аналогів

Тип додатку	Переваги	Недоліки
Багатосторінковий додаток	1. Багатосторінкова (MPA) архітектура дозволяє легко оптимізувати кожну сторінку для пошукових систем. Розробник може додати мета-теги для будь-якої сторінки; 2. Як правило, для розробки багатосторінкової	1. Розробка як десктопної, так і мобільної версії веб-сайту займає значно більше часу; 2. Збільшує вартість змін при додаванні нових функціональних можливостей в додатку.

	програми потрібен менший стек технологій, таким чином вартість таких додатків виходить дешевшою.	
Односторінкова програма (SPA)	1. Висока швидкість. Оскільки SPA не оновлює всю сторінку, а лише необхідну частину, це значно підвищує швидкість роботи додатку. 2. Висока швидкість розробки. Готові бібліотеки та фреймворки забезпечують потужні інструменти та велику кількість готових рішень для розробки веб-програм. 3. Мобільні додатки. SPA дозволяє створювати мобільні додатки, оскільки розробник може повторно використати той же код бекенда для веб-додатка та власної мобільної програми. 4. SPA може ефективно кешувати будь-які дані. Програма	1. Погана SEO оптимізація. SPA працює на основі JavaScript і завантажує інформацію за запитом від клієнтської частини. Пошукові системи навряд чи можуть імітувати цю поведінку. Оскільки більшість сторінок просто недоступні для сканування пошуковими роботами. 2. Продуктивність та оптимізація. Необхідно пам’ятати про мобільні пристрої та пристрої з низькою продуктивністю, оскільки в односторінкових програмах використання JavaScript який має

	надсилає лише один запит, кешує всі дані, після чого може їх використовувати і навіть працювати у режимі офлайн.	великий розмір. Також слід пам'ятати про розмір пакету, якщо проект великий, то його слід розбити на окремі компоненти з динамічним навантаженням, щоб покращити швидкість загрузки додатку.

На даному етапі розвитку WEB, односторінкові додатки швидко витісняють класичні додатки і є великим внеском у розробку масштабних, швидких, динамічних веб — систем. Однак технології реалізації front end частини web-додатків потребують додаткового дослідження в контексті вибору кращих альтернатив для реалізації систем під конкретні вимоги користувачів чи замовників продукту.

Полягаючи на отримані знання у розробці веб додатків. Як основний інструмент для розробки додатка було обрано бібліотеку - React та мову програмування - TypeScript. Але для реалізації окремого функціоналу (роутинг додатку, опрацювання та збереження даних, відображення інформації) необхідно підключати сторонні бібліотеки.

На етапі перед розробкою додатку важливо було дослідити та проаналізувати існуючі бібліотеки та обрати ті, що найбільше підходять під вимог.

Основні вимоги до методу роботи з отриманими даними від Google Books API.

1. Можливість отримати дані в різних форматах, та змінювати формат по бажанню.

2. Можливість зробити копію даних, зберігати її у власному сховищі та модифікувати.
3. Можливість робити модифікації з даними до того як дані будуть відображені користувачам.
4. Можливість створювати власні сховища, де зберігати модифіковані або відфільтровані дані.
5. Можливість робити запит та отримувати відповідь без перезавантаження сторінки.
6. Можливість робити запит на стороні клієнта.

Після визначення ключових вимог у роботі з даними, створюємо порівняльну таблицю.

Таблиця 2.2

Порівняння методів роботи з отриманими даними від API

Назва	**Переваги**	**Недоліки**
Зв’язка бібліотеки React з сховищем даних Redux	Можливість редагувати дані у власному сховищі перед попаданням у сховище. Легко додати або видалити з проекту. Під час розробки є можливість відслідковувати стан сховища використовуючи ReduxDevTools.	Для збереження даних після оновлення сторінки необхідно застосовувати додаткові сервисі

	Існує єдине джерело істини для всього вашого стану програми.	
Використання бібліотеки React з використанням React Hooks - useSelector, useDispatch.	Слідує кращим практикам та останнім технологіям.	Такий підхід повинен бути запланований на початку будови архітектури додатку, є великі складності у підключенні його у проект потім.

Після аналізу двох варіантів для зберігання даних в додатку, було вирішено до проекту підключати сторонній контейнер для сховища даних - Redux.

Основна концепція сховища даних з використанням Redux

1. Єдине джерело істини. При використанні Redux, основні дані для всього додатку представлені єдиним JavaScript об'єктом з посиланням на стан або дерево станів. Цей об'єкт може бути простим або складним, залежно від вимог вашої програми. Наприклад, станом для простого todo-додатку може бути один масив об'єктів списку завдань. Станом для сайту соціальних медіа може бути словник, який містить інформацію про статті, повідомлення, дані профілю та інші соціальні дані. Незалежно від розміру програми, всі дані стану зберігаються в одному об'єкті. Трохи пізніше ми розглянемо методи управління великими за розміром станами додатків.

```
const state = [
    {
        id: 1,
        task: 'Do laundry',
        completed: true
    },
    {
        id: 2,
        task: 'Paint fence',
        completed: false
    }
];
```

Рис. 2.1. Приклад об’єкта стану додатку.

2. Стан тільки для читання (read-only). Рівень представлення ніколи не буде безпосередньо керувати станом вашого додатку. Наприклад, уявіть, що обробник станів у формі додавання дії todo, безпосередньо не буде додавати нову задачу на ваш масив todos. Крім того, обробник буде пропускати action (дію), яка говорить: "Додаток, я хотів би додати «купити молоко» в масив Todos". Action(Дія) – простий об'єкт JavaScript, що виражає намір змінити стан об'єкту. Він містить мінімальну інформацію, необхідну для опису того, що повинно змінитися в результаті взаємодії з користувачем. Єдиним обов'язковим атрибутом action є тип, а всі інші дані, включені в action будуть специфічні для вашої програми, а тип дії буде включатися. Коли користувач додає завдання «Купити молоко», то дія «публікування» може виглядати наступним чином:

```
{
    type: 'ADD_TODO',
    task: 'Buy milk',
    id: 3
}
```

Рис. 2.2. Приклад об’єкта, що виражає намір змінити стан об'єкту.

3. Зміни за допомогою функцій. Отже, що ж відбувається з "діями", як тільки вони відбуваються в UI? Існує єдина функція, яка працює з цими "діями". Перемикач, стан якого залежить від дії поля type. Кожен тип дії, який може виділятися в вашому додатку потребує обчислення нового стану додатку на основі поточного стану. Функція, яка це виконує повинна бути чистою функцією. Якщо ви не знайомі з чистими функціями, я рекомендую вам подивитися відео Дена Абрамова, творця Redux, який пояснює, що це таке. Чиста функція - функція, що викликана з одними і тими ж аргументами і повертає один і той же результат. При передачі A і B завжди результатом в чистій функції буде C. Якщо функція є нечистою, передача A і B може дати C, або ж вони можуть дати інше значення D. Вихід залежить від вхідних даних і більше від нічого іншого. Чисті функції не мають ніяких побічних ефектів, тому їм не потрібен мережевий доступ або запити до баз даних. Крім того, чисті функції не змінюють свої вхідні аргументи. Замість цього вони використовують вхідні дані для обчислення деякого значення, а потім повертають розраховане значення. Продовжуючи наш todo приклад, case для типу дії 'ADD_TODO' не буде заносити нове значення в масив todos. Це не чиста функція, тому що вона змінює існуючий масив. Замість цього 'ADD_TODO' зробить копію масиву todos, додасть todo до кінця цього нового масиву, а потім повертає новий масив в якості наступного стану програми.

```
(currentState, action) => {
    switch(action.type){
        case 'ADD_TODO':
            const nextState = [
                ...currentState,
                {
                    id: action.id,
                    task: action.task,
                    completed: false
                }
            ];
            return nextState;
            break;
        default:
            return currentState;
    }
};
```

Рис. 2.3. Приклад функції зміни стану сховища.

Чиста функція, яка знає, як перетворити поточний стан додатку та будь-які дії в оновленому стані додатку називається root reducer – кореневим редуктором. Той факт, що root reductor обчислює наступний стан, а не змінює існуючий стан є дуже важливим в рамках Redux. Використовуючи цю модель, розрахунки стану залишаються швидкими, так як ми можемо просто передати посилання будь-яких незмінених даних в поточному стані до наступного стану. Ми також отримуємо гарантію, що наш стан не зміниться, бо знаємо, що він не може бути змінений за межами ланцюга action -> reductor.

Основні вимоги до бібліотек з модифікаціями даних від API.

1. Можливість використовувати готові методи для форматування дати.
2. Підтримка багатомовності, можливість автоматично переводити дані за запитом користувача.

Таблиця 2.3

Порівняння методів роботи з датою

Назва	Переваги	Недоліки
Moment.js	Бібліотека містить велику кількість методів для форматування дати. Легко інтегрувати або видалити в існуючий проект. Підтримую мультимовність. Містить велику кількість методів. Є можливість створювати власні методи.	Бібліотека важить дуже багато, що призводить до великого об'єму додатку. Необхідно завантажувати лише обрані методи. Moment.js змінюється, що може викликати помилки. Він має складний ООР API (який подвоює проблему мінливості). Через складний API це призводить до зниження продуктивності. Його розмір збирання великий при використанні з Webpack, оскільки файли локалі включені в комплект.
Використання JS об'єкту Date()	Не потребує додаткової інсталяції. Є вбудованим об'єктом. Підтримується всіма браузерами та пристроями. Не займає багато місця.	Складність у форматуванні дати. Має лише декілька методів для форматування часу.
Використання бібліотеки date-fns	Date-fns є незмінною, завжди повертаючи нову дату, а не змінюючи ту, яку ви передаєте.	Через складний API це призводить до зниження продуктивності.

	В нього простий API. Ви завжди маєте одну функцію, яка робить одну річ. Це швидко. Ви можете бути впевнені, що ваші користувачі будуть мати кращий досвід користувача. Це ідеальний компаньйон для Webpack. Завдяки стилю "функція для файлу" ви можете вибрати те, що вам потрібно, і припинити роздутий проект з марною функціональністю.	

Робота з датами JavaScript - це завжди складно. Методи дати часто бувають багатослівними і іноді несумісними, що також робить їх схильними до помилок. Але добрі новини під рукою. Існує кілька бібліотек, які допомагають позбутися застарілих маніпуляцій. Ці бібліотеки належать до дат JavaScript, а jQuery - до нативного DOM API.

Проблемою у роботі з датою я зустрілась на середині розробці додатку, під час виведення даних на сторінку. API нам повертав дату у форматі ММ/ДД/РРРР - що суперечило початковим вимогам.

Для форматування дати було обрано бібліотеку Moment.js яка містить готові методи для легкого форматування дати.

Для реалізації поставленого завдання важливо вибрати інструменти та методи, що найкраще підходять для цього.

Таблиця 2.4

Обрані інструменти для розробки та основні переваги перед аналогами

Назва інструменту	**Призначення**	**Переваги**
WebStorm.	Середовище для розробки. Для розробників на javascript jetbrains пропонує платформу webstorm. Вона підтримує популярні фреймворки для фронтенда (angular, react, vue.js) і бекенда (node.js, meteor). Можливості спільної розробки. Це знову ж таки стосується команд, які збираються працювати з загальним репозиторієм. Багато платформ, які ми розглянемо нижче, інтегруються з git. Підтримка потрібної вам операційної системи (ос). Особливу увагу цьому пункту варто приділити, якщо ви працюєте в команді. Найкраще	Середовище містить зручний інтерфейс для роботи з Git. Розумне автодоповнення коду. Інструменти для тестування karma, mocha, protractor і jest. Можливість налаштування власного середовища з налаштуванням форматування коду, рефакторингом, автодоповленням. Webstorm немає Безкоштовної ліцензії, ціна для індивідуального використання-59 $ в рік, для компаній-129 $ в рік на користувача.

	віддавати перевагу кроссплатформенним рішенням. Підтримувані мови (програмування, зрозуміло). Тут не забувайте про довгострокову перспективу-раптом коли-небудь ви вирішите додати в проект можливості, реалізовані на якій-небудь іншій мові. Варто вибрати середовище, яке підтримує кілька мов програмування.	
React	React - одна з бібліотек JavaScript з відкритим кодом. Він використовується для побудови інтерактивних інтерфейсів користувача. Це ефективна, декларативна та гнучка бібліотека. Він стосується компонента V, тобто перегляду модельного перегляду-контролера (MVC). Це не	Бібліотеку легко додати у існуючий проект, або створити новий проект з використанням цієї бібліотеки з CLI. Використовую компонентний підхід. Дозволяє використовувати синтаксис ES6. JSX JSX означає JavaScript XML. Це розширення до

	цілі рамки, а лише бібліотека прифронтових сторін. Це дозволяє створювати або створювати складні інтерфейси користувача з використанням ізольованих і невеликих фрагментів коду, відомих як компоненти. Основна перевага компонентів полягає в тому, що зміна будь-якого одного компонента не впливає на всю програму.	синтаксису мови JS. Він забезпечує спосіб візуалізації компонентів, використовуючи синтаксис, подібний до HTML. React використовує JSX для запису його компонентів. Він також може використовувати чистий JavaScript, але віддає перевагу JSX. Він використовується Babel, попередньою процесором для перетворення тексту, схожого на HTML, знайденого у файлах JavaScript, у стандартні об'єкти JS. HTML-код можна вбудовувати в JavaScript, щоб зробити HTML-код більш зрозумілим і підвищеним, а також підвищить ефективність JavaScript та зробить програму надійною. CCP

		Він розшифровується на стороні сервера. Це дозволяє попередньо віддати початковий стан компонентів на стороні сервера. Браузер може візуалізувати, не чекаючи, поки всі JavaScript будуть виконані або завантажені. Це змушує веб-сторінки завантажуватися швидше. Це допомагає користувачеві переглядати веб-сторінки навіть тоді, коли React все ще завантажує JavaScript, пов'язуючи події або створюючи віртуальний DOM на бекенді. Одностороння прив'язка даних Це дозволяє односторонній потік даних, тобто одностороння прив'язка даних. Завдяки цій функції є кращий

		контроль над додатком. Завдяки цьому стан додатків міститься в конкретних магазинах, тому всі інші компоненти залишаються нещільно зв'язаними. Це підвищує гнучкість та ефективність програми.
AXIOS	HTTP клієнт.	Робить XMLHttpRequests запити з браузеру Робить http запити з node.js Supports the Promise API Перехопити запит і відповідь Перетворення даних запиту та відповіді Скасувати запити Автоматичне перетворення даних JSON Підтримка на стороні клієнта для захисту від XSRF
GitHub	Найпопулярніше віддалене сховище для git-репозиторіїв.	Можливість хостингу та розгонвування проекту.

	Особливості: він дозволяє вам встановлювати права доступу до проектів, відстежувати і відправляти помилки, приймати запити на поліпшення, підписуватися на повідомлення сховища, використовувати графічний інтерфейс, а не командний рядок. Репо за замовчуванням відкриті, але платні акаунти можуть мати приватні репозиторії.	
React Developer Tool, Redux Developer Tools.	Розширення в браузері.	Зручний інструмент для відслідковування стану додатку та сховища. Можливість відстежувати зміну даних після дій користувача. Легко встановити в браузері.
Figma	Figma - цехмарний багатоплатформовий сервіс для дизайнерів інтерфейсів і web-	фрейми(Артборди) - є готові варіанти, але можна створювати

	розробників, з яким можна працювати безпосередньо в браузері. І це лише одне з важливих переваг платформи.	власні під конкретні завдання. Модульна сітка- для зручного упорядкування елементів дизайну в фреймах. векторні форми- для відтворення різних елементів інтерфейсу. криві- для створення кривих і простих векторних форм. Можна додатково завантажити їх з Sketch або Adobe Illustrator. зображення- для швидкого додавання графічних елементів в макет.

Висновок до розділу 2

В другому розділі було визначено функціональні та нефункціональні вимоги до додатку. Визначено стек технологій та інструментів для розробки сервісу. Проведено аналіз та дослідження сучасних методів розробки веб додатків.

Створено карту та план процесів розробки веб додатку.

РОЗДІЛ 3. ПРОЕКТУВАННЯ ВЕБ ДОДАТКУ

Процес моделювання та створення проекту перед початком фази розробки - один із найважливіших кроків у створенні додатку або системи. Від цього кроку, та правильності його виконання залежить подальша доля та розвиток додатку. Мета процесу - на початковому етапі правильно описати процеси та архітектуру додатка. Як результат гарного проектування будет простота розробки та подальшої підтримки додатку.

3.1 Шаблони проектування

Патерни (або шаблони) проектування описують типові способи вирішення поширених проблем при проектуванні програм.

Патерн проектування — це типовий спосіб вирішення певної проблеми, що часто зустрічається при проектуванні архітектури програм.

На відміну від готових функцій чи бібліотек, патерн не можна просто взяти й скопіювати в програму. Патерн являє собою не якийсь конкретний код, а загальний принцип вирішення певної проблеми, який майже завжди треба підлаштовувати для потреб тієї чи іншої програми.

Патерни часто плутають з алгоритмами, адже обидва поняття описують типові рішення відомих проблем. Але якщо алгоритм — це чіткий набір дій, то патерн — це високорівневий опис рішення, реалізація якого може відрізнятися у двох різних програмах.

Якщо провести аналогії, то алгоритм — це кулінарний рецепт з чіткими кроками, а патерн — інженерне креслення, на якому намальовано рішення без конкретних кроків його отримання.

Основні вимоги до якості написання програмного забезпечення з використання патернів проектування

1. Перевірені рішення. Ви витрачаєте менше часу, використовуючи готові рішення, замість повторного винаходу велосипеда. До деяких рішень ви могли б дійти й самотужки, але багато які з них стануть для вас відкриттям.
2. Стандартизація коду. Ви робите менше розрахунків при проектуванні, використовуючи типові уніфіковані рішення, оскільки всі приховані в них проблеми вже давно знайдено.
3. Загальний словник програмістів. Ви замовляєте назву патерна, замість того, щоб годину пояснювати іншим програмістам, який крутий дизайн ви придумали і які класи для цього потрібні.

Класифікація патернів

Патерни відрізняються за рівнем складності, деталізації та охоплення проектованої системи. Проводячи аналогію з будівництвом, ви можете підвищити безпеку на перехресті, встановивши світлофор, а можете замінити перехрестя цілою автомобільною розв’язкою з підземними переходами.

Найбільш низькорівневі та прості патерни — ідіоми. Вони не дуже універсальні, позаяк мають сенс лише в рамках однієї мови програмування.

Найбільш універсальні — архітектурні патерни, які можна реалізувати практично будь-якою мовою. Вони потрібні для проектування всієї програми, а не окремих її елементів.

Крім цього, патерни відрізняються і за призначенням. Є три основні групи патернів:

Породжуючі патерни піклуються про гнучке створення об’єктів без внесення в програму зайвих залежностей.

Структурні патерни показують різні способи побудови зв’язків між об’єктами.

Поведінкові патерни піклуються про ефективну комунікацію між об’єктами.

3.2 Етапи проектування додатку

У процесі розробки будь-якого ПО можна виділити наступні основні види діяльності:

1. Визначення вимог до ПО (інжиніринг вимог): призначений для розуміння розв'язуваної проблеми.
2. Проектування: призначене для планування вирішення проблеми.
3. Реалізація: перетворення плану в працюючий програмний код.
4. Перевірка і оцінка якості: призначене для виявлення помилок кодування (програмного коду) або невідповідностей між певними вимогами і їх реалізацією.
5. Розгортання: надання користувачам можливості працювати зі створеним ПО.
6. Підтримка: призначене для відстеження використання діючої системи і збереження її працездатності.
7. Розвиток: призначене для поліпшення з часом розробленого рішення; надання нових вхідних даних для процесу розробки в формі нових вимог.

Інжиніринг вимог призначений для розуміння необхідних можливостей і характеристик створюваного ПО. Даний аналіз спрямований на визначення функціональних вимог (які функції система повинна виконувати) і не функціональних вимог (якість пропонованого рішення). Інжиніринг вимог також передбачає виявлення загальної ідеї, яка стоїть за розробляється системою; основних зацікавлених осіб, яким потрібна нова система, і умови, в яких буде використовуватися система. Виявлені вимоги обробляються з метою створення високорівневих моделей даної системи, яка абстрагується від непотрібних подробиць даної проблемної області.

Проектування призначене для опису рішення, яке повинно відповідати функціональним вимогам і вимогам ефективності, а також обмеженням того середовища, в якій вона буде працювати. Раніше зібрані вимоги уточнюються і поліпшуються, щоб задовольняти можливим технологічним обмеженням. **Проектування включає такі дії, як:**

1. Проектування схеми даних і класів.
2. Проектування компонент.
3. Проектування графічного інтерфейсу.
4. Проектування архітектури системи.

Допомагає краще сформувати специфічні особливості системи, такі як структура, поведінка, взаємодія, дані і потік управління. Дозволяє розділити сфери відповідальності, основний принцип програмної інженерії - вирішення проблеми шляхом поділу на різні підзадачі - може допомогти впоратися зі складністю і досягти необхідних технічних якостей, таких як адаптованість, простота підтримки, розширюваність і багаторазова використовуванність.

На етапі реалізації проекту розроблені проектні рішення перетворюються у відповідний програмний код (вручну або за допомогою інструментів автоматизації програмування). Можуть знадобитися бібліотеки програм, різні мови програмування, різні комунікаційні протоколи і технічні пристрої.

Перевірка і оцінка якості зазвичай проводиться паралельно з реалізацією, т. К. Правильність і надійність проміжних результатів, а не тільки кінцевого продукту є дуже важливим для гарантування якості всього програми. **Якість в значній мірі пов'язано з наступними критеріями:**

1. Оцінкою функціональності, т. Е. Правильності поведінки додатки щодо заданих функціональних вимог;
2. Оцінкою продуктивності, т. З. часу очікування відгуку програми в звичайних умовах і при пікових навантаженнях;
3. Оцінкою зручності використання (usability), т. Е. Легкості використання, комунікаційної ефективності та відповідності стандартам використання.

Розгортання додатки надає користувачам можливість використання цього додатка.

Залежно від типу додатка процес розгортання може включати:

1. Конфігурація проміжного комунікаційного ПО;
2. Інструктування і навчання майбутніх користувачів, особливо, якщо встановлюється зовсім новий додаток, а не нова версія вже наявної програми.

Підтримка розгорнутого і ефектів у програмному забезпеченні означає забезпечення його робочого стану, яке складається в гарантуванні його доступності та зменшенні збоїв. Може включати: періодичну перевірку файлів журналу; звітів про помилки та очищення тимчасових файлів; виправлення помилок; установку виправлень.

Розвиток програми. Додаток призначений для вирішення реальних завдань користувачів. Однак у зв'язку з розвитком організації, ускладненням розв'язуваних завдань, поліпшенням розуміння користувачем можливостей даного ПЗ неминуче потрібно розвивати створене ПО. Нові потреби користувачів з'являються тільки після того, як вони деякий час попрацюють з створеним ПО. Після цього вони починають давати свої пропозиції та коментарі. Поява нових вимог може викликати необхідність запустити весь процес розробки заново. Незважаючи на суворе дотримання рекомендацій правильної організації процесу розробки ПО, часто тільки після розгортання і накопичення деякого досвіду роботи користувачів з ПО стає ясно, що деякі вимоги не були повністю виконані і додаток повинен бути доопрацьовано.

3.3 Модель варіантів використання

Для більшого розуміння як користувач буде взаємодіяти з електронною бібліотекою було створено діаграму моделі варіантів використання.

Діаграма варіантів використання зображує послідовність дій, які можуть бути виконані системою при взаємодії з відповідним актором (користувачем).

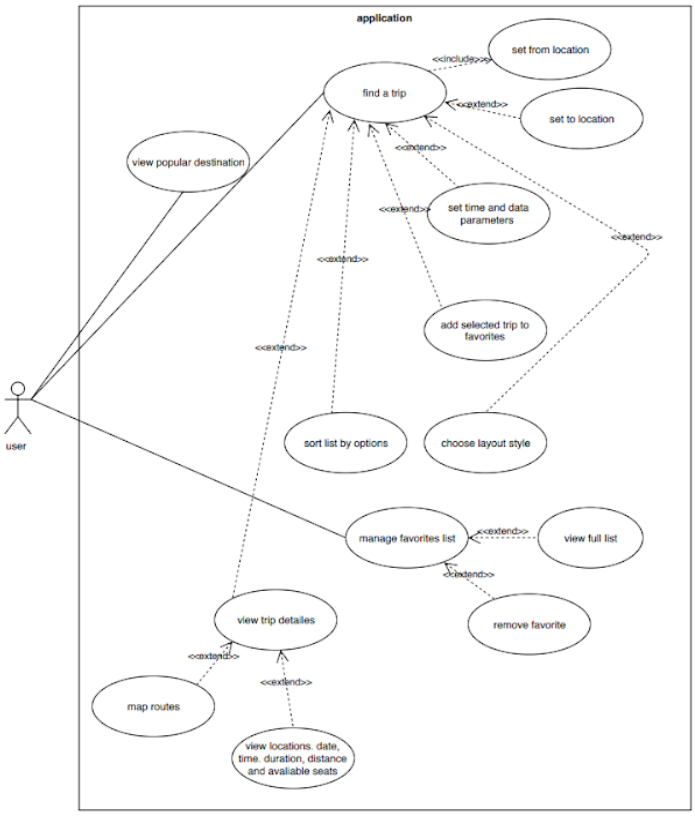

Рис. 3.1. Зображення модель варіантів використання WEB додатку електронної бібліотеки.

3.4 Проектування користувацького інтерфейсу

Користувацький інтерфейс - дуже важлива складова веб додатків та сервісів яка напряму впливає чи будуть користувачі правильно користуватись застосунком та розуміти його можливості та весь функціонал.

Розробці інтерфейсів необхідно завжди необхідно приділяти час, не обмежуватись розробкою інтерфейсів лише для певних груп, а брати весь спектр користувачів та зробити інтерфейс доступним для всіх.

При створенні інтерфейсу веб додатку слід звернути увагу на UX (дизайн взаємодію з користувачем) та UI (користувацький інтерфейс). З основного, дизайн повинен бути зрозумілим, простим, сучасним без використання застарілих засобів взаємодії з користувачем.

Першими кроками було створення макетів для головної сторінки веб додатку, та створення спільних компонентів які використовуються на всьому сайті. Цьому

потрібно приділяти більше часу, адже вигляд, функціональність, та консистентність спільних компонентів є однією із цілей створенні цього додатку.

Загальною метою при розробці макетів користувацьких інтерфейсів було створення модульних компонентів інтерфейсу, які мають бути консистентними та використовуються на всіх сторінках додатку.

Одним із рішень для досягнення консистентності по всьому додатку є створення бібліотеки компонентів та модулів на початку та створення сторінок із цих модулів.

Бібліотека базових компонентів включає в себе такі елементи:

1. Кнопки
2. Поля вводу
3. Форми
4. Меню
5. Типографію
6. Навігацію
7. Списки
8. Картки
9. Всі інші компоненти вводу інформації

На рисунку 6 зображено вигляд головної сторінки додатку. Сторінка містить компоненти: шапку, підвал та контент. Компоненти шапка та підвал повторюються на всіх сторінках.

Контент головної сторінки включає:

1. Банер з кнопкою, натиснувши яку користувач буде перенаправлений на сторінку пошуку, невеликим описом додатку та графічне зображення.
2. Секцію з популярними жарнами книг.
3. Секція з кнопкою почати пошук.

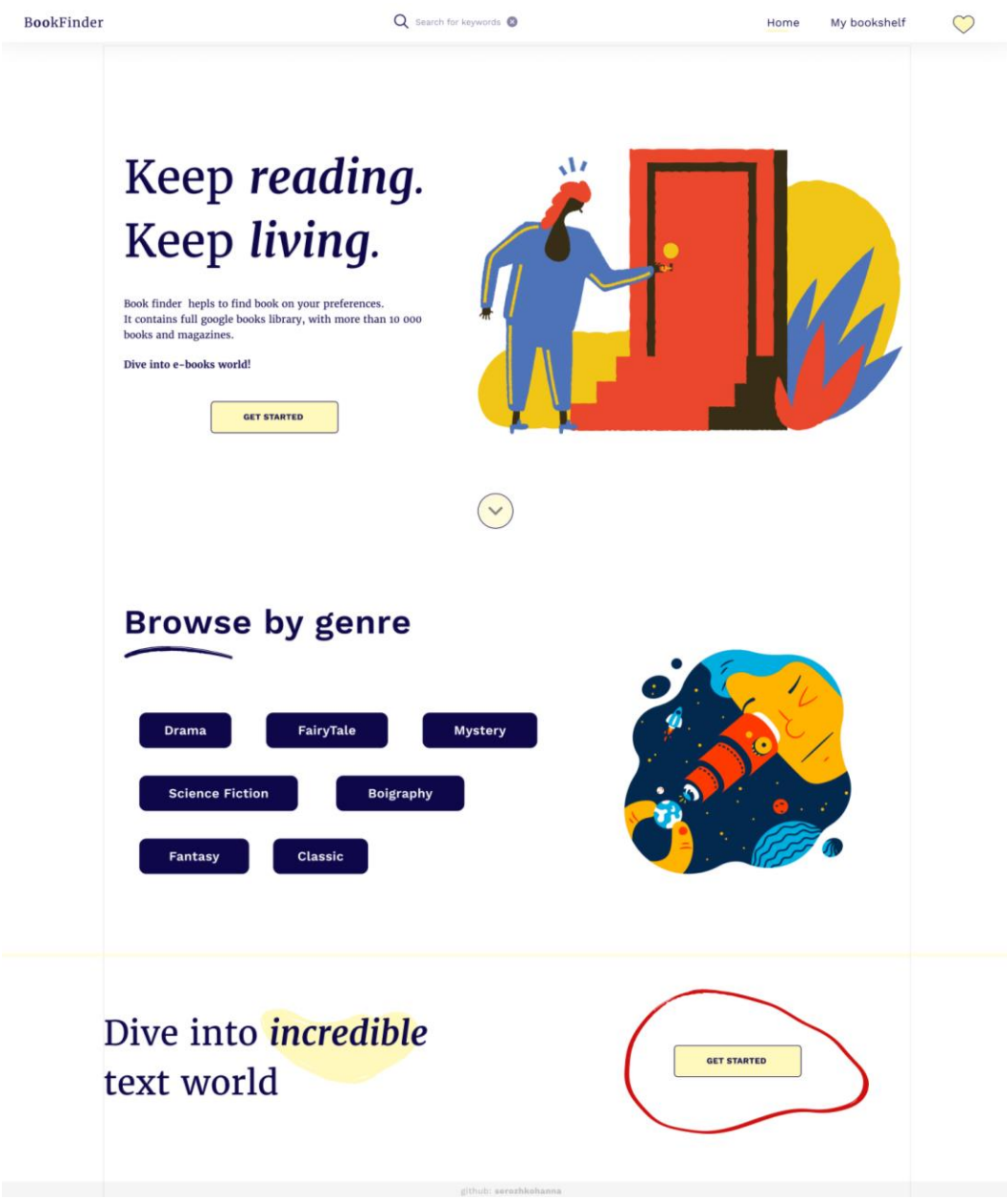

Рис. 3.2. Макет головної сторінки сайта.

В шапці находится логотип додатку, форма пошуку за ключовими словами, меню, та кнопка з обраними книгами. Користувач має змогу ввести у форму пошуку за ключовими словами декілька слів, пошук за цими словами буде виконуватись і результати пошуку будут на сторінці результатів, куди користувач буде перенаправлений.

В банері знаходиться кнопка для початку швидкого пошуку, опис функцій додатку та графічне зображення. Якщо користувач хоче перейти до швидкого пошуку він може це зробити натиснувши кнопку почати пошук у банері.

Наступною секцією є секція з основними жанрами книг, натиснувши їх користувач може здійснити пошук серед книг за жанром. Натиснувши один із жанрів користувач буде автоматично переадресований на сторінку з результатами пошуку.

Наступною є секція з кнопкою почати пошук яка виконує таку ж саму роль як і кнопка у банері зверху. Користувача буде перенаправлено на сторінку з пошуком.

На рисунку 7 зображено вигляд другої сторінки додатку - сторінка з результатами пошуку та формою пошуку. Сторінка містить спільну шапку та підвал, перемикач форм з типом пошуку, форму з детальним пошуком або швидким пошуком, слова за якими виконується пошук, результати пошуку у вигляді списку, блок з фільтрами та блок з сортуванням, пагінацію.

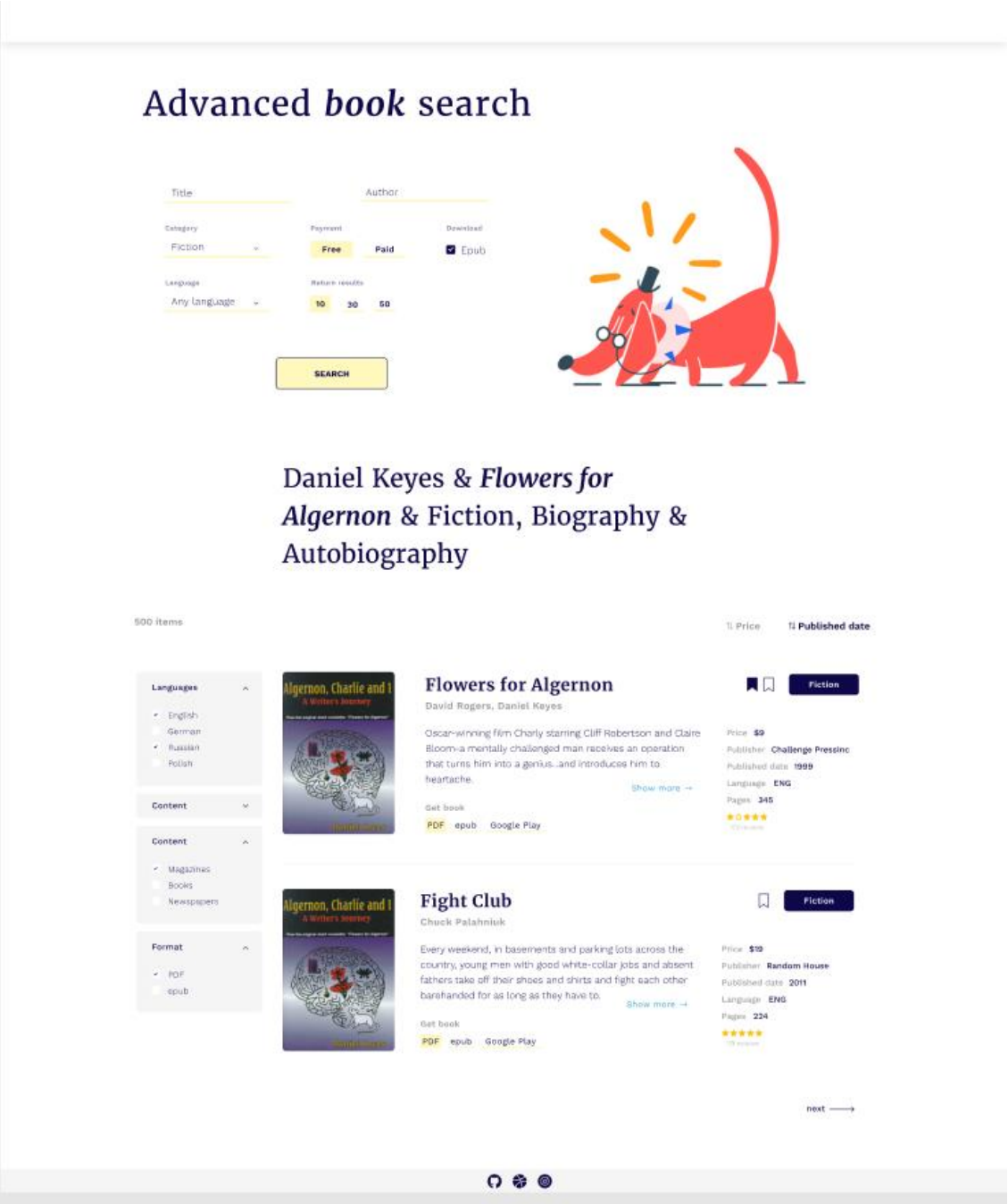

Рис. 3.3. Макет сторінки пошуку та результатів пошуку.

У користувача є можливість виконувати два види пошуку, якщо він знає конкретну назву, автора книги то він може скористатись детальним пошуком за книгами, ввівши більше характеристик. Поля назва книги та автор є обов'язковими, необов’язковими для заповнення є вибір категорії, формат книги, мова, та кількість результатів на сайті.

Другим видом пошуку є швидкий пошук у бібліотеці, для цього користувачеві потрібно буде ввести лише ключові слова у форму.

Після того як заповнить форму пошуку, пошук по бібліотеці буде здійснений, та результати пошуку будуть відображені у виді списку нижче.

Користувач буде бачити 10 результатів на сторінці, користувач може скористатись пагінацією для перегляду наступних результатів.

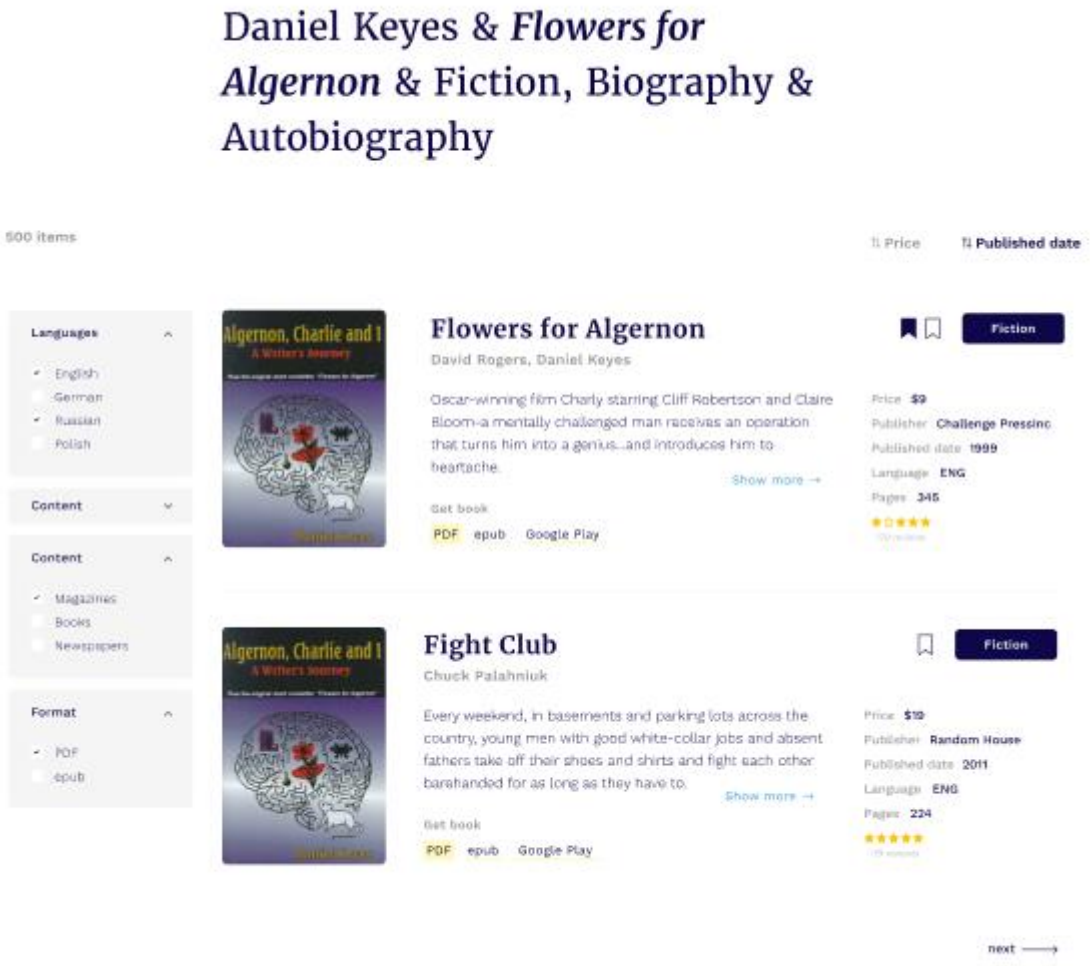

Рис. 3.4. Запит та результати пошуку.

Зліва знаходяться фільтри, які користувач може застосувати та відфільтрувати необхідні книги. Також користувач може відсортувати результати пошуку за наступними критеріями: ціна та дата публікації книги.

Користувачеві відображаються результати у вигляді списку з короткою інформацією про книги. Кожна картка містить наступну інформацію:

1. Назва книги або журналу
2. Автор
3. Короткий опис книги
4. Тип завантаження або перегляду публікації
5. Жанр книги
6. Ціну
7. Коротку інформацію про видавництво.

На рисунку 8 зображено вигляд сторінки з детальним описом обраної книги чи журналу. Сторінка містить детальну інформацію про книгу, обкладинку книги та знизу секцію зі схожими книгами.

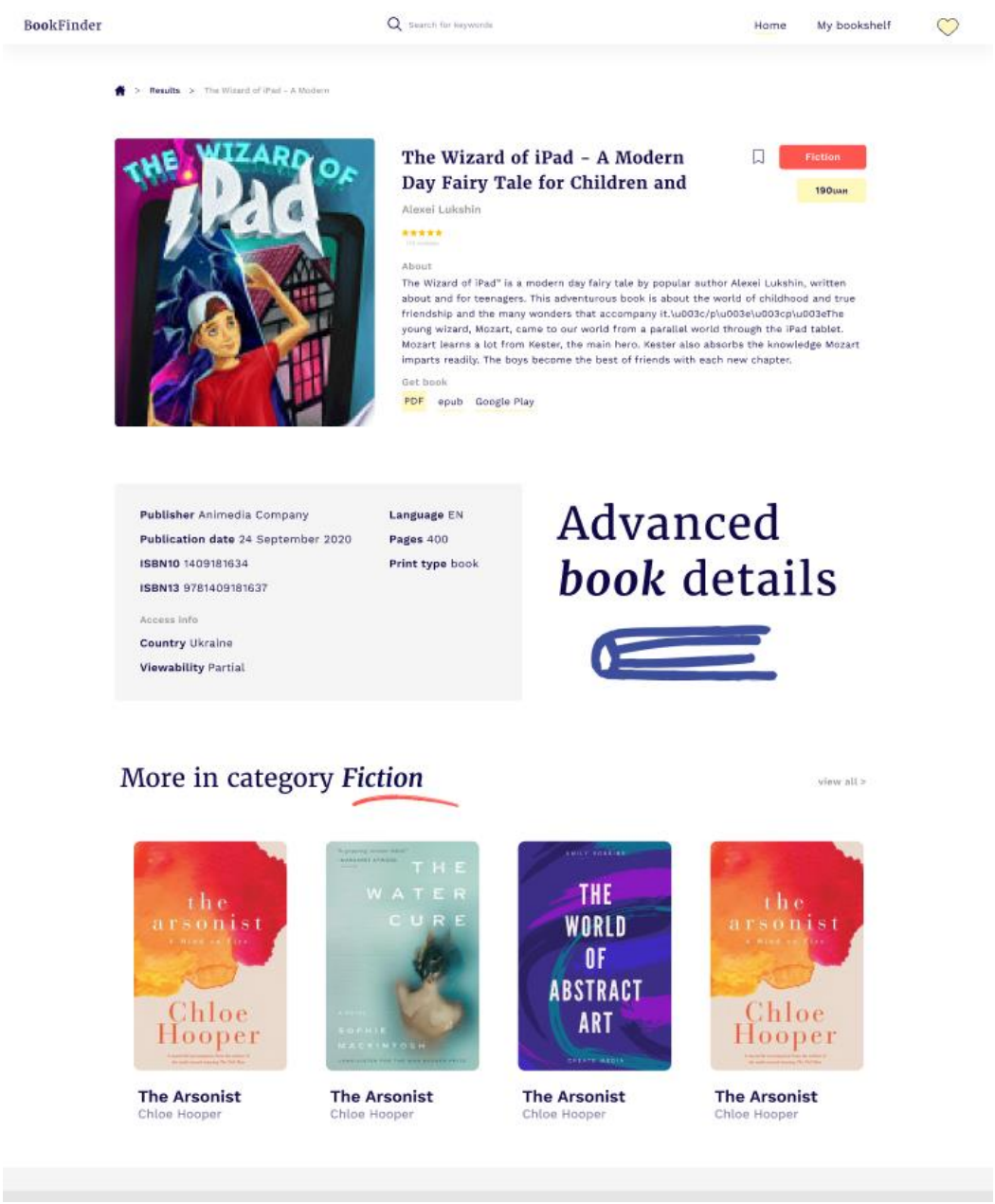

Рис. 3.5. Макет сторінки з детальною інформацією.

Сторінка містить детальну інформацію про книгу, обкладинку книги та знизу секцію зі схожими книгами.

На рисунку 9 зображено компонент із обраними. Тут знаходяться книги або журнали які користувач помітив як обрані. Користувач має можливість видалити елемент із обраного, або перейти на сторінку з детальним описом книги натиснувши на назву.

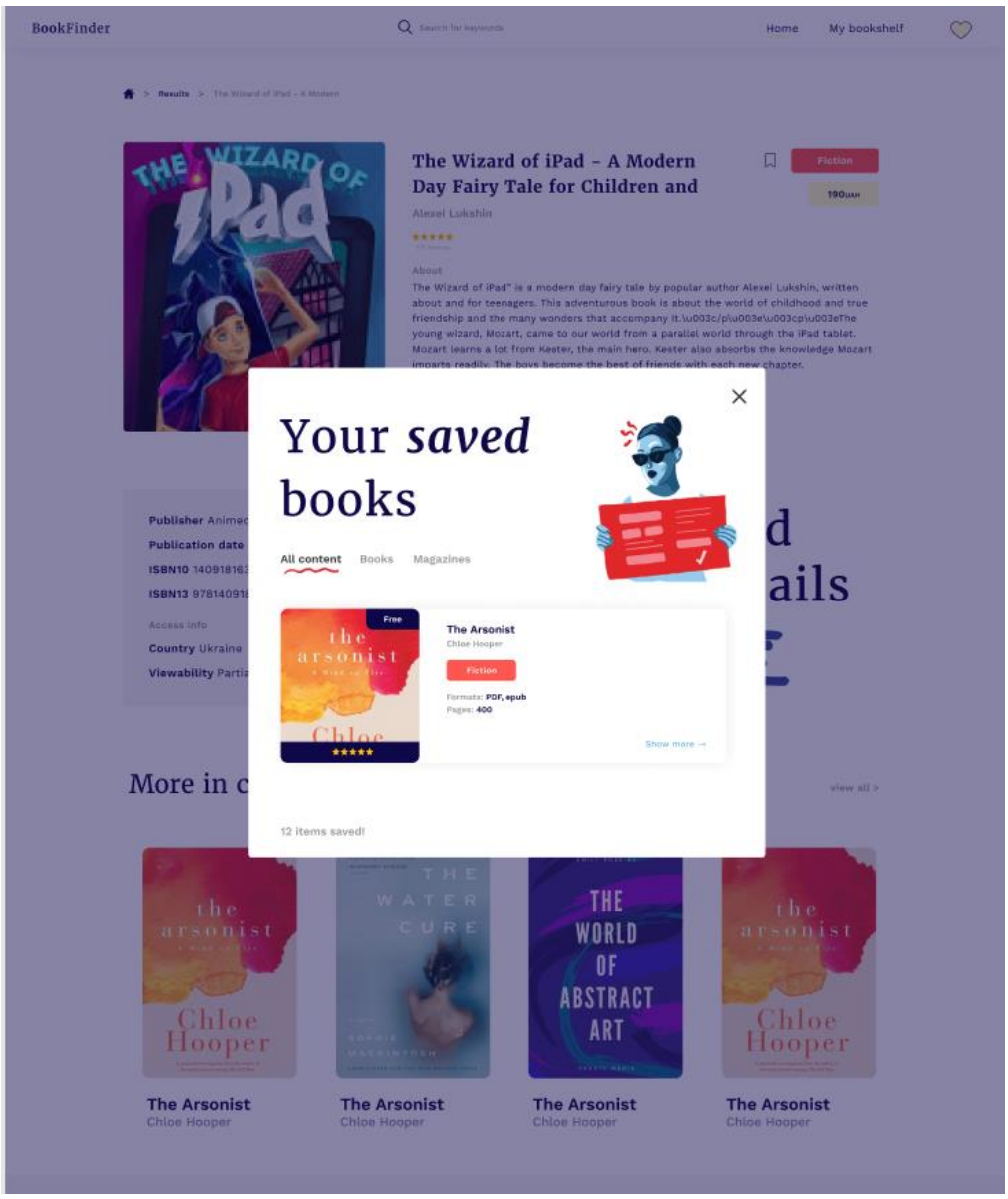

Рис. 3.6. Макет компонента з обраними книгами.

Елементи поділені на категорії в залежності від типу контента, це може бути книга або журнал. В компоненті також присутня пагінація, максимальні кількість елементів на сторінці - 10.

Висновок до розділу 3

В третьому розділі було створено проект та архітектуру додатку. Визначено етапи розробки, тестування та підтримки додатку. Зпроектовано користувацький інтерфейс за допомогою графічних інструментів.

РОЗДІЛ 4. РОЗРОБКА ВЕБ ДОДАТКУ

4.1 Архітектура проекту

Архітектура програмного забезпечення це процес перетворення таких характеристик програмного забезпечення, як гнучкість, масштабованість, виконуваність, повторне використання та безпека у структуроване рішення, яке відповідає технічним та діловим очікуванням. Архітектура програмного забезпечення системи зображує організацію або структуру системи і надає пояснення щодо її поведінки. Система являє собою набір компонентів, які виконують певну функцію або набір функцій. Іншими словами, архітектура програмного забезпечення забезпечує міцний фундамент, на якому можна будувати програмне забезпечення. Серія архітектурних рішень і компромісів впливає на якість, продуктивність, ремонтопридатність і загальний успіх системи. Визначення загальних проблем і довгострокових наслідків може поставити систему під загрозу. Існують численні структури та принципи архітектури високого рівня, які широко використовуються в сучасних системах. Вони часто називають архітектурними стилями. Архітектура системи програмного забезпечення рідко обмежується єдиним архітектурним стилем. Замість цього, комбінація стилів часто складають повну систему.

Принципи гарної архітектури

Принцип відкриття-закриття (Open Close Principle або OCP). Програмні суті такі як класи, модулі та функції повинні бути відкриті для розширення, але закриті для змін.

Принцип Заміщення Ліскоу (Liskov's Substitution Principle). Похідні типи повинні бути здатні повністю заміщатися їх базовими типами.

Принцип Єдиної Відповідальності (Single Responsibility Principle). Клас повинен мати тільки одну причину для зміни.

Принцип Відділення Інтерфейсу (Interface Segregation Principle). Клієнти не повинні бути залежними від інтерфейсів, які вони не використовують.

Принцип інверсії залежностей. (Dependency Inversion Principle) - залежності всередині системи будуються на основі абстракцій. Модулі верхнього рівня не залежать від модулів нижнього рівня. Абстракції не залежать від подробиць.

Принцип інверсії залежностей в деталях. У протиставленні поганому дизайну, гарний дизайн архітектури повинен бути гнучким, стійким, і пристосованим до повторного використання. Чим нижче взаємозв'язок компонентів додатка один з одним, тим вища гнучкість і мобільність всієї програми в цілому. Програми, що характеризуються високим коефіцієнтом мобільності, дозволяють застосовувати свої компоненти знову і знову для вирішення однотипних задач. Це веде до зниження дублювання коду. Такі програми складаються з великого набору досить дрібних компонентів, кожен з яких виконує малу частину роботи, але виконує її якісно. Дрібні компоненти набагато простіше тестувати, реалізовувати і супроводжувати.

Model-Viewer-Controller

У шаблоні MVC, як випливає з назви, є три основних компоненти: Модель, Представлення, і Контролер.

Представлення відповідає за відображення інформації, що надходить із системи або в систему.

Модель є «суттю» системи і відповідає за безпосередні алгоритми, розрахунки тощо внутрішній устрій системи.

Контролер є сполучною ланкою між «поданням» і «моделлю» системи, за допомогою якого і існує можливість зробити поділ між ними. Контролер отримує дані від користувача і передає їх в «модель». Крім того, він отримує повідомлення від моделі, і передає їх в «подання».

Представлення. Модуль виведення інформації. Це може бути шаблонизатор або що-небудь подібне, мета якого є тільки в поданні інформації у вигляді HTML на основі будь-яких готових даних.

Контролер. Модуль управління введенням і виведенням даних. Даний модуль повинен стежити за переданими в систему даними (через форму, рядок запиту, cookie або будь-яким іншим способом) і на основі введених даних вирішити:

• Передавати чи їх у модель

• Вивести повідомлення про помилку і запросити повторне введення (змусити модуль уявлення оновити сторінку з урахуванням умов)

Крім того, контролер зобов'язаний визначати тип даних, отриманих від моделі (чи є це готовий результат, відсутність повідомлення про помилку) і передавати інформацію в модуль уявлення.

Модель. Модуль, що відповідає за безпосередній розрахунок чого-небудь на основі отриманих від користувача даних. Результат, отриманий цим модулем, повинен бути переданий в контролер, і не повинен містити нічого, що відноситься до безпосереднього висновку (тобто має бути представлений у внутрішньому форматі програми).

Досить складно з першого разу розібратися і зрозуміти. На це потрібен час і відповідний проект.

Але насправді нічого надскладного в цьому немає.

На рисунку 10 зображено Стандартна схема архітектури «Модель-Вид-Контролер».

Рис. 4.1. Стандартна схема архітектури «Модель-Вид-Контролер».

4.2 Структура проекту

В розробці веб сервісу з використанням бібліотеки React, було обрано компонентний підхід у створенні структури проекту, що означає відокремлювати кожну частину інтерфейсу в окремий компонент, та зберігати в окремій папці. Кожен компонент складається з файлу .tsx у якому описана логіка компоненту та html шаблон, файл .scss зі стилями які підключаються до компонента та файл з юніт тестами. .test.ts

Всі компоненті незалежні один від одного. Використовуються на сторінках до яких вони підключаються.

Всі сторінки які є на сайті знаходяться у папці Pages. Спільним корінним файлом до якого підключаються всі сторінки та реалізовано роутинг додатку є файл App.tsx.

Всі спільни стилі, графіку, шрифти, іконки та ассети відокремлено в окрему папку assets.

Так як додаток використовує бібліотеку для зберігання даних Redux, всю логіку з модифікаціями даних винесено в окремі папки, store, reducer, actions.

Всі спільні утиліти та константи які використовуються на проекті та поширюються між усіма файлами винесено в окремі папки. У папці constants знаходяться спільні константи, у папці utils - утиліти.

На рисунку 11 зображено структуру папок у проекті.

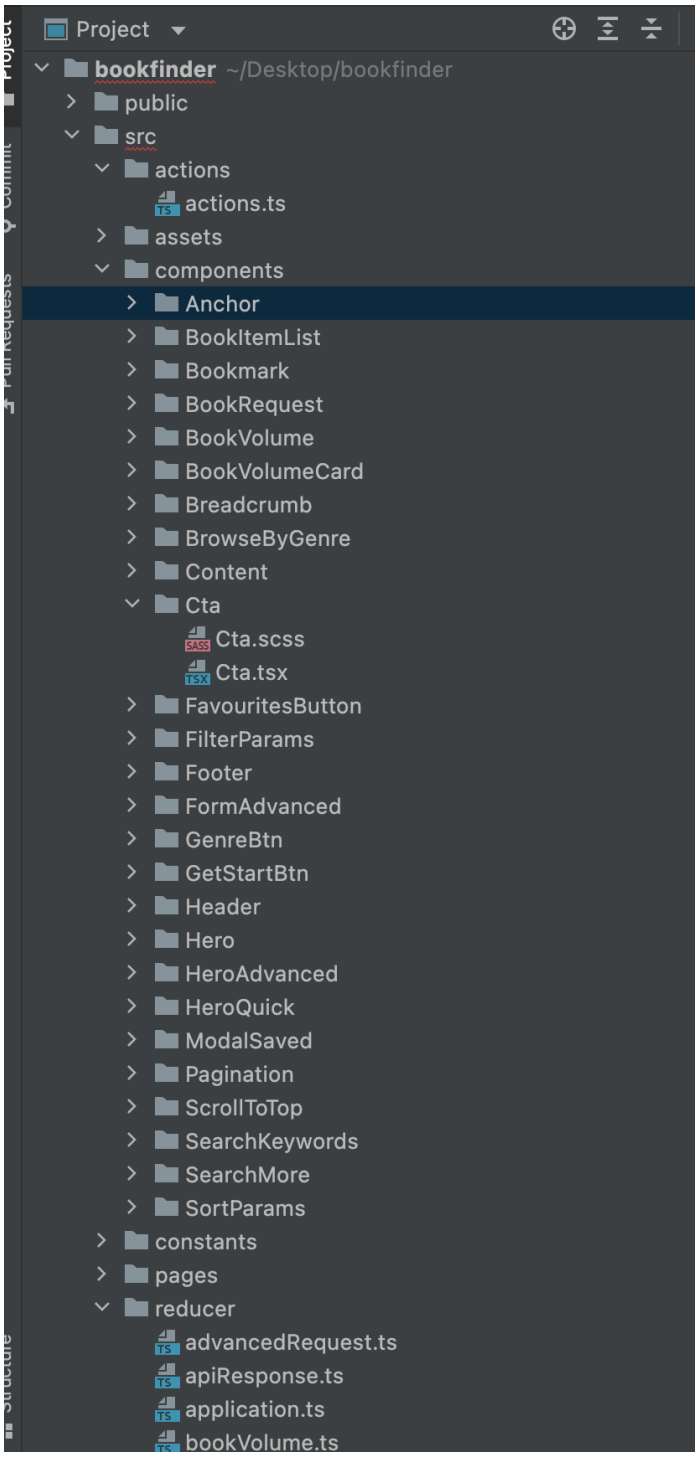

Рис. 4.2. Структура папок.

4.3 Розробка функціоналу

Основна частина створення веб додатку або іншого додатку є розробка його функціоналу, що повинна проводитись з урахуванням результатів з попередніх розділів.

Розробка додатку з використанням бібліотеки React починається з розробки компонентів, адже вони є складової сторінок з яких складається додаток. Компоненти дозволяють розділити інтерфейс користувача на незалежні частини, придатні до повторного використання, і сприймати їх як такі, що функціонують окремо один від одного. Концептуально компоненти є подібними до функцій JavaScript. Вони приймають довільні вхідні дані (так звані “пропси”) і повертають React-елементи, що описують те, що повинно з’явитися на екрані.

Функціональні компоненти

Найпростішим способом визначення компонента є написання функції JavaScript:

```
function Welcome(props) {
  return <h1>Привіт, {props.name}</h1>;
}
```

Рис. 4.3. Приклад функціонального компоненту.

Ця функція є валідним React-компонентом, оскільки вона приймає єдиний аргумент “пропс” (скорочено від properties - властивості), який є об’єктом з даними і повертає React-елемент. Такі компоненти ми називаємо “функціональними компонентами”, оскільки вони буквально є JavaScript функціями.

Классові компоненти

Також можно використовувати ES6 класи, щоб визначити компонент:

```
class Welcome extends React.Component {
  render() {
    return <h1>Привіт, {this.props.name}</h1>;
  }
}
```

Рис. 4.4. Приклад классового компоненту.

Композиція компонентів

Компоненти можуть посилатися на інші компоненти під час виведення. Це дозволяє нам використовувати одну і ту ж абстракцію компонентів для будь-якого рівня деталізації. Кнопка, форма, діалогове вікно, екран: у React-додатках всі вони зазвичай виражаються як компоненти.

Наприклад у нашому додатку існує компонент App.tsx який є функціональним та рендерить усі сторінки.

```
function App({isSidebarOpen}) {
  console.log(isSidebarOpen);
  return (
    <div className={`page ${isSidebarOpen && 'scroll-disabled'}`}>
      <Header/>
      <ModalSaved/>
      <div className="page-content">
        <Switch>
          <Route exact path={HOME_PAGE} component={MainPage}/>
          <Route exact path={SEARCH_PAGE} component={SearchPage}/>
          <Route exact path={DETAILS_PAGE} component={DetailsPage} />
          <Route component={Page404}/>
        </Switch>
      </div>
      <Footer/>
    </div>
  );
}
```

Рис. 4.5. Приклад композиції компонентів.

Життєвий цикл компонента

Кожен компонент, який ми створюємо, буде мати власні життєві події, які можуть стати у пригоді за певних обставин. Наприклад, якщо ми хочемо зробити ajax запит на першопочатковий рендеринг і отримати деякі дані, то де ми маємо його викликати? Або, якщо б ми забажали використовувати будь-яку логіку, у той час як наші реквізити змінились, як це зробимо? Різноманітні події життєвого циклу — відповіді на обидва цих питання.

componentDidMount — викликається одноразово, після первинного рендеринга. Оскільки компонент вже викликається при виклику цього методу, у вас є доступ до віртуальної DOM, ви можете викликати його через this.getDOMNode(). Таким чином ця подія життєвого циклу, в якій ви робитимете запити AJAX для здобуття певних даних.

componentWillUnmount — метод життєвого циклу викликається безпосередньо перед видаленням компонента з DOM його використовують для звільнення пам'яті та очищення змінних.

getDerivedStateFromProps — інколи вам потрібно відновити стан вашого компонента на основі отриманих props. Цей метод життєвого циклу допоможе вам це зробити. Ви можете передати props і state, а об'єкт, що повертається, буде об'єднано з поточним станом.

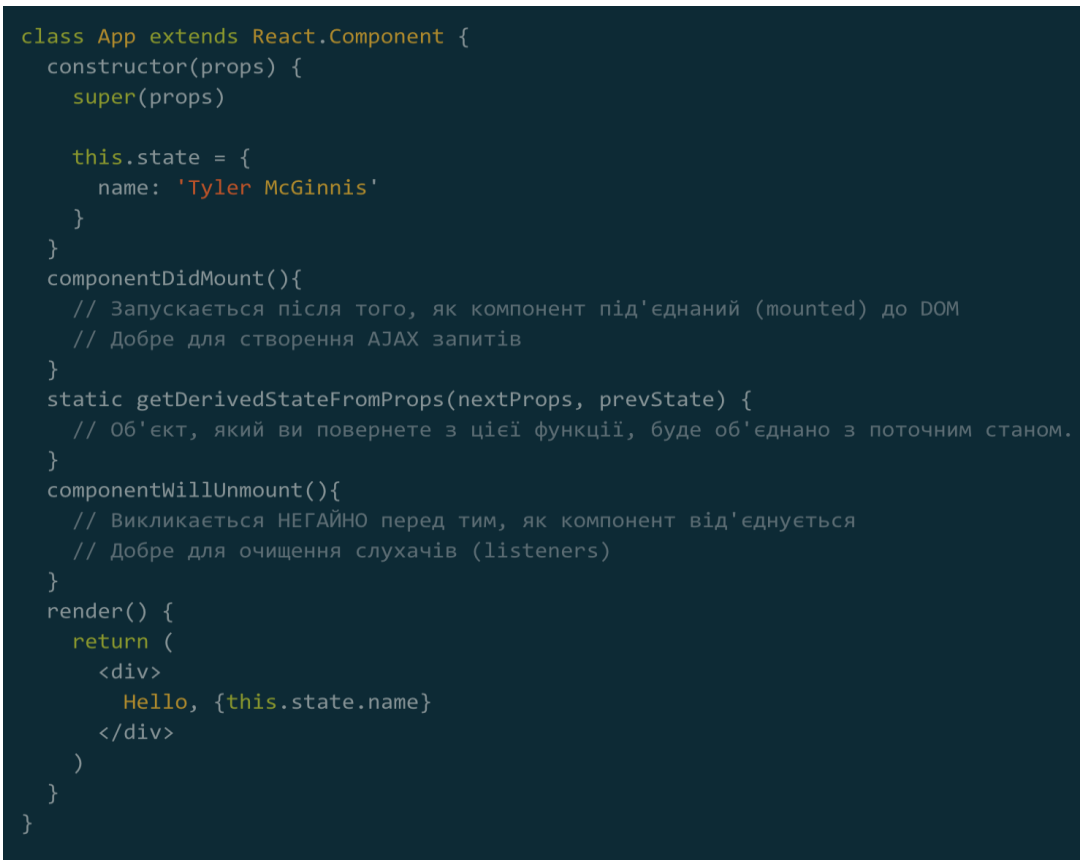

Рис. 4.5. Життєвий цикл компонента.

4.4 Розробка веб доступності додатку

Скрінрідери (screen reader) — найпоширеніша технологія, з якою й асоціюють веб-доступність. Перетворює текст з екрана на аудіопотік синтезованої мови або набору звуків різної частоти. Крім того, скрінрідери дають змогу швидко вивести список елементів (заголовки, посилання, кнопки, секції тощо) і переходити між ними, що значно спрощує життя. Найпопулярніші скрінрідери — VoiceOver (macOS, iOS), NVDA (Windows), Google TalkBack (Android).

Дисплей Брайля — пристрій, що перетворює візуальну інформацію на набір знаків, побудованих із шістьох точок, які можна відрізнити на дотик пальцями рук. Для роботи з ним потрібен спеціальний софт, що виводитиме інформацію на дисплей.

Розпізнавання мовлення й керування голосом. Цим зараз нікого не здивуєш, усі вміють «Ok, Google» і «Hey, Siri». Проте для людей з ушкодженнями опорно-рухового апарату розпізнавання голосу для роботи з пристроями завжди залишатиметься актуальним, зокрема зараз, коли якість розпізнавання надзвичайно висока. Проморолик від Apple показує, наскільки розпізнавання мовлення важливе.

Перемикачі — одна або кілька великих кнопок в одному корпусі, на які користувач може призначити різні дії, як-от скрол або вибір елемента. Його також використовують люди з ушкодженнями опорно-рухового апарату.

Клавіатура й фокус

Якщо ви працюєте над веб-сайтом або вже маєте такий, користувач повинен мати змогу користуватись інтерфейсом лише за допомогою клавіатури.

Семантика й контент

Для того щоб відобразити сторінку, браузери парсять HTML, CSS, JS, будують DOM-, CSSOM-дерева, а також дерево Accessibility. A11y-дерево — це спрощене структуроване подання документа, яке використовують асистивні технології:

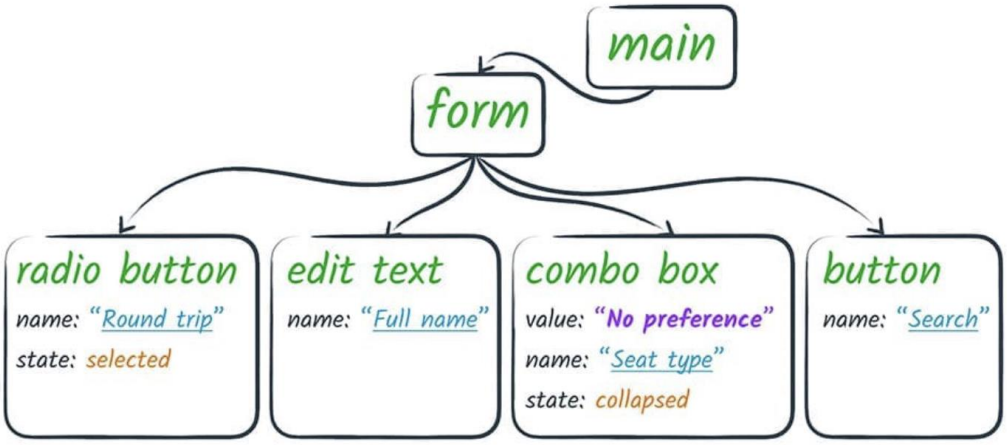

Рис. 4.6. Структурна схема документа.

Під час побудови цих дерев браузери ігнорують помилки, невідповідності до специфікацій і намагаються вгадати, що все-таки мав на увазі автор. І це добре працює з HTML, проте це не варіант для асистивних технологій.

Також семантика стосується інших HTML-елементів.

Альтернативний текст зображень. Атрибут alt існує не просто так. Якщо зображення не завантажилося або користувач не може його побачити, то альтернативний текст — це єдиний спосіб передати зміст. Написання цього тексту — ціле мистецтво, таке саме, як і назви класів та змінних. Якщо зображення ілюстративне, опишіть його; якщо це комікс — опишіть діалог. Проте якщо це іконка або прев'ю в каруселі, то, може, варто взагалі його приховати? Ось усі можливі комбінації зображення та alt тексту:

Правильне використання заголовків. Коли люди читають книги або документи, вони час від часу переглядають зміст — набір заголовків і підзаголовків з відповідною нумерацією сторінок. Заголовки в HTML створили з такою самою метою — для структурування документа й спрощення навігації. Користувачі асистивних технологій використовують заголовки для швидкої навігації. Я часто зустрічаю заголовки, які насправді — просто текст із font-size 30, або навпаки, усі заголовки — це H2. Пропоную користуватися простим правилом: теги H — для структури, CSS — для краси.

Посилання й кнопки. Увесь веб живе завдяки посиланням. Без них це був би просто набір поодиноких сторінок, які відвідували б лише автори та їхні мами. Проте, як і з заголовками, є кілька правил для ефективного використання посилань. Якщо користувач знає, що він шукає саме посилання (до прикладу, це вікіпедія та статті про мандарин і апельсин точно мають посилання одна на одну), то він викличе окремий список посилань за допомогою скрінрідера.

HTML-орієнтири (landmark). Усе можна зверстати div-ами. Проте у W3C не гають часу й намагаються всім спростити життя. HTML5 Landmarks — це не просто хіпстерські трюки, а спосіб полегшити життя користувачам асистивних технологій, пошукових систем і девелоперів.

ARIA

Технології не стоять на місці, і в нас уже є десятки HTML-тегів, проте багато звичних нам UI-патернів, таких як модальне вікно, дерево чи тултип, доводиться реалізовувати самим. І весь набір елементів, з яких ми будуємо ці патерни, потрібно

представити користувачеві асистивних технологій. Саме для цього створили WAI-ARIA (Web Accessibility Initiative — Accessible Rich Internet Applications), набір атрибутів, що дають змогу авторові розмітки змінювати accessibility tree, щоб асистивні технології інакше відображали контент. Це ініціатива, призначена оживити веб для користувачів асистивних технологій — дати їм можливість не просто читати сторінки, а й користуватися веб-застосунками.

Основний атрибут в ARIA — role. За допомогою нього можна змінити тип елемента для асистивних технологій. До прикладу, якщо ви використовуєте іконковий шрифт і певний символ важливий у вашому контексті, то варто озвучити значення цього символу.

Стилі

CSS — невід'ємна частина доступності. Стилі — це не лише про дизайн і яскраві кольори. Якісне візуальне оформлення й зручність для користувачів дає свій результат, і вони частіше повертаються до зручних інтерфейсів.

Responsive styles. Нікого не здивувати responsive-стилями. Проте не варто забувати, що вони стають у пригоді не лише для користувачів смартфонів, але й для користувачів з поганим зором. Набагато зручніше переглядати сторінку в одну колонку на 30-дюймовому моніторі, ніж «бігати» нею, використовуючи, окрім вертикального, ще й горизонтальний скрол. Не блокуйте зум і використовуйте більший font-size у media queries.

Контраст. Згідно з WCAG 2.0, текст і зображення зобов'язані мати контраст щонайменше 4.5:1.

4.5 Аналіз метрик продуктивності додатку

Тестування та аналіз метрик було проведено за допомогою інструмента Lighthouse. Google Lighthouse — це автоматизований інструмент із відкритим вихідним кодом для вимірювання якості веб-сторінок. Його можна запускати на будь-якій веб-сторінці, загальнодоступній чи з вимогою аутентифікації.

Google Lighthouse перевіряє продуктивність, доступність і пошукову оптимізацію веб-сторінок. Він також включає можливість тестування прогресивних веб-додатків на відповідність стандартам і передовим практикам.

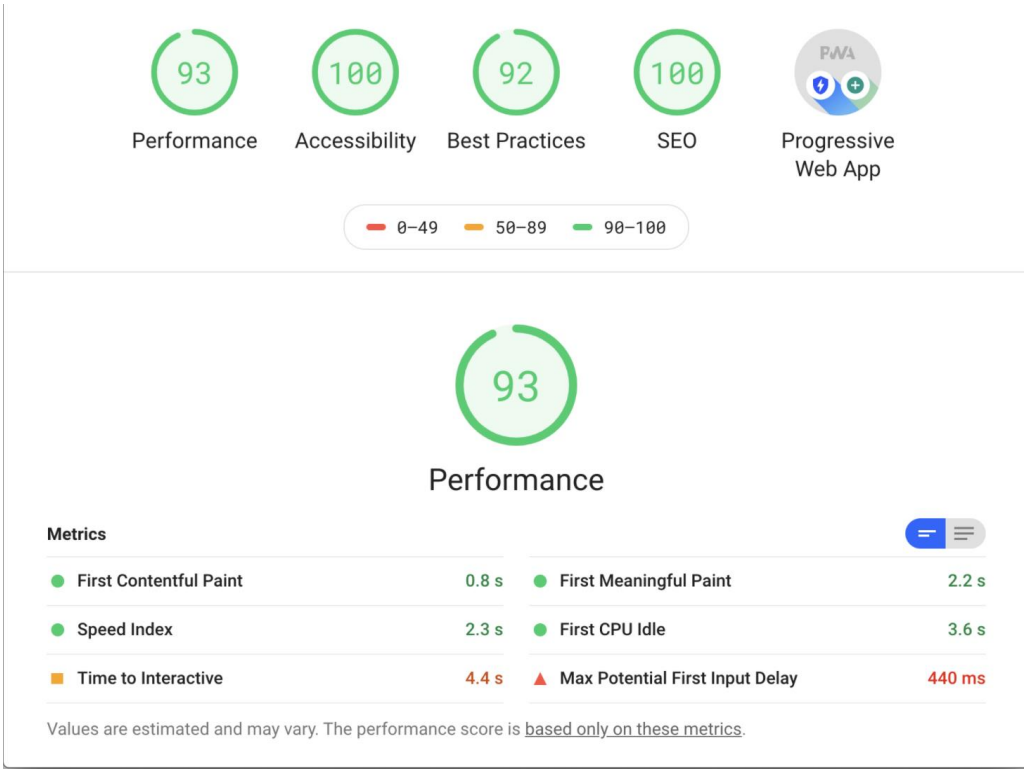

Рис. 4.7. Згенерований репорт по продуктивності додатку.

4.6 Тестування компонентів

Для тестування додатку - написання unit тестів було використано бібліотеку Jest та React testing library.

Основні переваги автоматизованого тестування:

1. Тестування дає впевненість в тому, що навіть один змінений рядок коду не зламає весь застосунок;
2. Замінює ручне тестування, яке з часом стає обтяжливим;

   допомагає задокументувати граничні випадки;

Існує безліч видів тестів, однак загалом їх можна поділити на модульні, інтеграційні та end-to-end. Разом вони утворюють таку ієрархію.

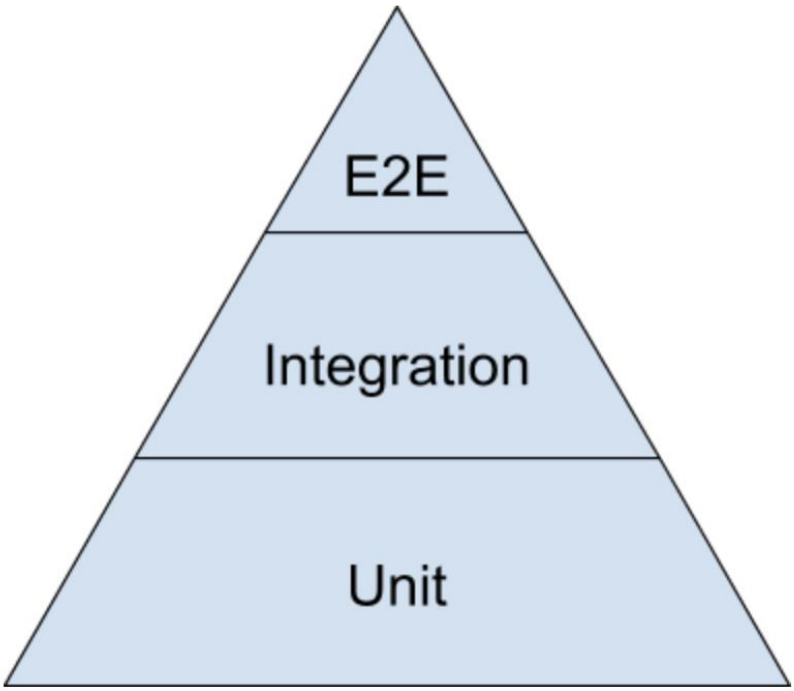

Рис. 4.8. Ієрархія тестів.

Тести на нижчих рівнях потребують менше ресурсів для написання та підтримки, а також швидше запускаються. То чому б нам тоді не обмежитись лише юніт-тестами? Суть в тому, що саме тести на вищих рівнях дають більше впевненості у роботі системи та взаємодії компонентів між собою.

Принципи тестування React-застосунків

Мета тестування — переконатися, що застосунок працює коректно. Якщо всі важливі частини функціоналу застосунку мають тести, ви одразу дізнаєтесь, коли щось зламалось.

Поширеними інструментами автоматизованого тестування застосунків на React є Jest у поєднанні з @testing-library/react (або Testing Library). Існують і їхні аналоги. Наприклад, аналогами Jest є Mocha, Jasmine та AVA. Testing Library — альтернатива для Enzyme, яку багато розробників досі використовують.

Testing Library розглядає тестування застосунку з погляду користувача. Наслідком такого підходу стає написання інтеграційних тестів, декілька компонентів системи взаємодіють між собою.

Демонстрація компонентів

Перейдемо до інтеграційних тестів. Чому в ієрархії вони стоять вище за модульні тести? Усе тому, що тепер ми перевіряємо не просто JavaScript-код — а, швидше, взаємодію між елементами DOM та відповідну логіку роботи компонента.

Перший компонент, який ми перевіримо, демонструє свій стан у певному вигляді. Коли ми клікаємо на кнопку, стан компонента змінюється.

```
import React from 'react';
import { render } from '@testing-library/react';
import { Footer } from './Footer.js';

describe('Footer', () => {
  it('should render component', () => {
    const { getByTestId } = render(<Footer />);

    const element = getByTestId('info');

    expect(element).toHaveTextContent('Click to modify');
    expect(element).toContainHTML('<p class="info" data-testid="info">Click to modify</p>');
    expect(element).toHaveClass('info');
    expect(element).toBeInstanceOf(HTMLParagraphElement);
  });
});
```

Рис. 4.9. Приклад написання тестів до компонента.

Взаємодія батьківських-дочірніх елементів

У React батьківські компоненти спілкуються з дочірніми через props, а дочірні компоненти з батьківськими — за допомогою колбеків. У тесті нам потрібно передати props та перевірити, чи викликає компонент метод onModify. Компоненту ми передаємо prop info, а також prop-функцію onModify. Коли відбувається подія кліку на кнопку, метод onModify модифікує змінну callArgument. Assert-функція наприкінці перевіряє, чи був аргумент callArgument змінений дочірнім компонентом.

```
it('should handle interactions', () => {
  const info = 'Click to modify';
  let callArgument = null;
  const onModify = arg => callArgument = arg;
  const { getByTestId } = render(<Footer info={info} onModify={onModify} />);

  const button = getByTestId('button');
  fireEvent.click(button);

  expect(callArgument).toEqual('Modified by click');
});
```

Рис. 4.10. Приклад написання тестів до компонента.

Інтеграція сховища

У попередніх прикладах стан завжди був у компонентах. У складних застосунках необхідно мати доступ та змінювати той самий стан в декількох місцях. Тоді на допомогу приходить Redux: бібліотека управління станом, яка допомагає зберігати стан в одному місці, а його зміни робить передбачуваними. У сховищі ми зберігаємо єдиний стан, той самий, що ми бачили в компоненті. Ми можемо модифікувати його за допомогою функції onModify, яка передає вхідні параметри функції-редьюсеру, що змінює стан.

```
it('should modify state', () => {
  store.dispatch(onModify('Modified by click'));

  expect(store.getState()).toEqual('Modified by click');
});
```

Рис. 4.11. Тестування роботи сховища даних.

Роутинг

Найпростіший спосіб протестувати роутинг у React-застосунку — створити компонент, що показуватиме поточний шлях. Компонент Footer огортаємо компонентом вищого порядку withRouter, який передає додаткові props в дочірній

компонент. Нам потрібен ще компонент App, який огортає Footer та прив'язує роути. В тесті ми перевіряємо контент елементу Footer. Наш компонент відловлює всі роути, адже ми не визначили prop path в компоненті Route. Всередині тесту не бажано модифікувати History API браузера, але ми можемо скористатися реалізацією in-memory та передати її як prop history-компоненту Router.

```
import { Router } from 'react-router-dom';
import { createMemoryHistory } from 'history';
import { render } from '@testing-library/react';

describe('Routing', () => {
  it('should display route', () => {
    const history = createMemoryHistory();
    history.push('/modify');

    const { getByTestId } = render(
      <Router history={history}>
        <App/>
      </Router>
    );

    expect(getByTestId('location-display')).toHaveTextContent('/modify');
  });
});
```

Рис. 4.12. Тестування роутингу додатку.

HTTP-запити

Найчастіше стан компонента змінюється саме після HTTP-запитів. Хоч як спокусливо б не було виконувати всі запити, цього робити не варто, адже тест буде залежати від сторонніх ефектів, а отже не буде надійним. Аби уникнути такого антипатерну, ми можемо замінити реалізацію запиту під час виконання. Такий процес називається mocking. В нас є функція: її вхідні параметри передаються методу post, а результат обробляє метод commit. Код стає асинхронним і приймає Axios як сторонню залежність. Тестова функція стає асинхронною одразу, як ми додаємо до неї модифікатор async: Jest очікує повернене значення за допомогою await. Наприкінці ми робимо ствердження, що імітований нами метод commit було викликано з параметрами, поверненими post.

```
it('should set info coming from endpoint', async () => {
  const commit = jest.fn();
  jest.spyOn(axios, 'post').mockImplementation(() => ({
    body: 'Modified by post'
  }));

  await onModify({ commit }, 'Modified by click');

  expect(commit).toHaveBeenCalledWith('modify', { info: 'Modified by post' });
});
```

Рис. 4.13. Тестування роутингу додатку.

Браузер

Create React App з коробки не підтримує рішень для E2E-тестування, тож нам потрібно налаштувати все самостійно: запустити застосунок, запустити Cypress тести в браузері, зупинити застосунок. Організовуються E2E-тести подібно до модульних: блок describe відповідає за групування, it — за конкретний тестовий випадок. Глобальна змінна cy — це екземпляр Cypress, який відповідає за запуск тестів. Ми також можемо контролювати дії Cypress у браузері.

Після переходу на основну сторінку (за допомогою visit) ми можемо отримати доступ до її html за допомогою селекторів. Так відбувається перевірка вмісту елементів. Усі види взаємодій відбуваються за аналогією: обираємо елемент (get) і виконуємо над ним певну дію (click). Наприкінці тесту перевіряємо — змінився вміст елемента чи ні.

```
describe('New todo', () => {
  it('it should change info', () => {
    cy.visit('/');

    cy.contains('.info', 'Click to modify');

    cy.get('button').click();

    cy.contains('.info', 'Modified by click');
  });
});
```

Рис. 4.14. Тестування додатку у браузері.

Висновок до розділу 4

В четвертому розділі було описано імплементацію додатку з використанням засобів та інструментів які було обрана на фазі планування.

Протестовано компоненти та користувацький інтерфейс. Задеплоєно додаток на хостинг та підготовлено до користування.

ВИСНОВКИ

Таким чином, можна стверджувати, що електронна бібліотека не антагоніст і не конкурент традиційній бібліотеці, а нове явище в бібліотечно-бібліографічному обслуговуванні. Скоріше за все електронні бібліотеки можна розглядати як форму реалізації функцій традиційної бібліотеки в сучасних умовах (у одних випадках більш вдалу, а в інших – менш вдалу), тобто на принципово новій технологічній основі. Електронний простір дає змогу реалізовувати різноманітні зв’язки між елементами записів бібліотечних каталогів, що безумовно підвищує можливості використання бібліотечної інформації. З огляду на цифрову природу електронних бібліотек вони, крім першочергового свого завдання – забезпечення тематичного доступу читачів до повнотекстових документів, цілком здатні реалізувати вищезазначений інтегративний потенціал електронного простору. Розроблена структура дає змогу створити формалізовану модель, виокремити й узагальнити точки доступу, за допомогою яких встановлюються зв’язки між різними предметними областями та атрибутованими ними документами. У підсумку це робить можливим організацію та представлення тематичних електронних бібліотек, здатних забезпечити релевантний інформаційний пошук для користувачів і бути повноцінно інтегрованими в сучасне інформаційне середовище.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Head First. Патерни проєктування / Ерік Фрімен , Елізабет Робсон. – К.:НІСД, 2010. – 203 с.
2. Чистий код. Створення і рефакторинг за допомогою Agile. 2005. – 348с.
3. JavaScript. Полное руководство, 7-е издание. Флэнаган Дэвид. 2003. – 67с.
4. The World’s Technological Capacity to Store, Communicate, and Compute Information [Електронний ресурс] : http://science.sciencemag.org/content/332/6025/60 – Дата звернення: 01.10.18. – Назва с екрану.
5. Abour Radar [Електронний ресурс] : https://radar.io/about– Дата звернення: 01.10.18. – Назва с екрану.
6. Top 100 UX Design Tips from a User Experience Master [Електронний ресурс]
7. Sensors chemical and drug detection [Електронний ресурс]. URL:http://www.prosecurityzone.com/News/Detection/Sensors__chemical_and_drug_detection/Chemical_warfare_agent_detection_integrated_to_surveillance_vms_15154.asp#ixzz2AtaXXzdU (дата звернення 14.06.2021);
8. Матриця відповідальності [Електронний ресурс]: https://studopedia.com.ua/1_64379_ponyattya-matritsi-vidpovidalnosti.html/ – Дата звернення: 01.10.18. – Назва с екрану.
9. Understanding the Risk Breakdown Structure (RBS) [Електронний ресурс]: https://project-management.com/understanding-the-risk-breakdown-structure-rbs/ – Дата звернення: 01.10.18. – Назва с екрану.
10. Beyond create React App [Електронний ресурс] : https://medium.com/@auth0/beyond-create-react-app-react-router-redux-saga-and-more-3e80503c66ab?source=search_post---------6 – Дата звернення: 01.04.20. – Назва з екрану.