МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет
імені Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ
В.о. завідувача кафедри інтелектуальних інформаційних систем, канд.техн.наук, доц.
_____ Є. В. Сіденко
«____»_____2023 року

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

ОПТИМІЗАЦІЯ ПРОДУКТИВНОСТІ ПРОГРЕСИВНОГО WEB-ЗАСТОСУНКУ БІБЛІОТЕКИ НА ОСНОВІ МОДЕЛІ RAIL

Спеціальність 122 «Комп'ютерні науки»

122 – МКР – 601.21710120

Виконав студент 6-го курсу, групи 601
_____ М. Г. Олійник
«16» лютого 2023 р.

Керівник: канд. пед. наук, доцент
_____Н. М. Болюбаш
«16» лютого 2023 р.

Миколаїв – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет ім. Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

Освітньо-кваліфікаційний рівень **магістр**
Галузь знань **12 «Інформаційні технології»**
(шифр і назва)
Спеціальність **122 «Комп'ютерні науки»**
(шифр і назва)

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри інтелектуальних інформаційних систем, канд.техн.наук, доц.
_____ Є. В. Сіденко
«___» _____ 20__ р.

З А В Д А Н Н Я
на магістерську кваліфікаційну роботу

Олійнику Михайлу Геннадійовичу

1. Тема магістерської кваліфікаційної роботи «Оптимізація продуктивності прогресивного web-застосунку бібліотеки на основі моделі RAIL».
Керівник роботи Болюбаш Надія Миколаївна, канд. пед. наук, доцент.
Затв. наказом Ректора ЧНУ ім. Петра Могили від «03» листопада 2022 р. № 199
2. Строк подання студентом роботи 16 лютого 2023 р.
3. Вхідні (початкові) дані до роботи: загальні відомості про мережеві сервіси у сфері діяльності бібліотек, електронні ресурси сервісу Google Books.
Очікуваний результат роботи: прогресивний вебзастосунок бібліотеки, оптимізаці якого здійснюється у процесі його розробки за моделлю RAIL.
4. Перелік питань, що підлягають розробці (зміст пояснювальної записки):
- дослідження теоретичних засад створення прогресивних web-засунків та здійснення аналізу мережевих сервісів у сфері діяльності бібліотек;
- обґрунтуваня вибору технологій і засобів розробки прогресивного web-застосунку;

- розробка та здійснення програмної реалізації прогресивного web-застосунку бібліотеки з реалізацією оптимізації його продуктивності на основі моделі RAIL.

5. Перелік графічного матеріалу: презентація, рисунки, таблиці.

6. Завдання до спеціальної частини: охорона праці та безпека у приміщенні комп'ютерної лабораторії закладу вищої освіти.

7. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис
Спеціальна частина з охорони праці	докт.біол.н.,професор Л. І. Григор'єва	
Методична частина	канд.пед.н., доцент Н.М. Болюбаш	

Керівник роботи __канд. пед. наук, доц. Болюбаш Н. М._____
(наук. ступінь, вчене звання, прізвище та ініціали)

_____
(підпис)

Завдання прийнято до виконання ___Олійник М. Г._____
(прізвище та ініціали)

_____
(підпис)

Дата видачі завдання «__07__» _____листопада___ 2022 р.

КАЛЕНДАРНИЙ ПЛАН

виконання магістерської кваліфікаційної роботи

Тема: «Оптимізація продуктивності прогресивного web-застосунку бібліотеки на основі моделі RAIL»

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Визначення керівника і теми МКР. Подання заяви на затвердження теми МКР	01.09.2022	20.10.2022	Виконано
2.	Отримання завдання на виконання МКР	21.10.2022	10.11.2022	Виконано
3.	Складання календарного плану	11.11.2022	15.11.2022	Виконано
4.	Огляд літератури за темою дослідження. Аналіз існуючих мережевих сервісів у сфері діяльності електронних бібліотек, методів оптимізації прогресивних web-застосунків	16.11.2022	27.11.2022	Виконано
5.	Проходження переддипломної практики, збір та аналіз матеріалів до МКР	28.112022	18.12.2022	Виконано
6.	Аналіз предметної області та розробка технічного завдання	19.12.2022	22.12.2022	Виконано
7.	Проектування та програмна реалізація прогресивного web-застосунку бібліотеки з аналізом отриманих результатів	23.12.2022	15.01.2023	Виконано
8.	Робота над розділами фахової частини МКР	16.01.2023	24.12.2023	Виконано
9.	Розробка методичної частини МКР та спеціальної частини з охорони праці	25.01.2023	01.02.2023	Виконано
10.	Обговорення отриманих результатів з керівником та попередній захист МКР	02.02.2023	3.02.2023	Виконано
11.	Корегування роботи за результатами попереднього захисту	4.02.2023	6.02.2023	Виконано
12.	Остаточне оформлення пояснювальної записки та слайдів доповіді до зихисту	7.02.2023	9.02.2023	Виконано
13.	Подання рецензенту та рецензування МКР	9.02.2023	12.02.2023	Виконано
14.	Подання МКР, її електронної копії та інших документів (відгуку, рецензії) до захисту	15.02.2023	16.02.2023	Виконано
15.	Захист МКР перед ЕК	23.02.2023	23.02.2023	Виконано

Розробив студент _____Олійник М.Г. _____ _____

(прізвище та ініціали) (*підпис*)

Керівник роботи __канд.пед.н., доц. Болюбаш Н.М. _____ _____

(наук. ступінь, вчене звання, прізвище та ініціали) *(підпис)*

«_12_» ___листопада__ 2022 р.

АНОТАЦІЯ

до магістерської кваліфікаційної роботи
студента групи 601 ЧНУ ім. Петра Могили

Олійника Михайла Геннадійовича

на тему: «**ОПТИМІЗАЦІЯ ПРОДУКТИВНОСТІ ПРОГРЕСИВНОГО WEB-ЗАСТОСУНКУ БІБЛІОТЕКИ НА ОСНОВІ МОДЕЛІ RAIL**»

Керівник: канд.пед.н. доцент Болюбаш Надія Миколаївна

Магістерська кваліфікаційна робота присвячена розробці та здійсненню програмної реалізації прогресивного web-застосунку бібліотеки, оптимізацію продуктивності якого реалізовано з використанням моделі RAIL. Що є актуальним в умовах підвищення вимог до якості та продуктивності електроних бібліотек, оскільки модель RAIL забезпечує оптимізацію показників продуктивності у процесі розробки.

Об'єкт дослідження – оптимізація продуктивності web-застосунків.

Предмет дослідження – прогресивні web-застосунки для підтримки діяльності бібліотек із оптимізацією їх продуктивності на основі моделі RAIL.

Мета дослідження – підвищення ефективності роботи електронної бібліотеки шляхом створення прогресивного web-застосунку, реалізованого на основі орієнтованої на користувача моделі оптимізації продуктивності RAIL.

Магістерська кваліфікаційна робота складається з фахової, методичної і спеціальної частини з охорони праці. Пояснювальна записка кваліфікаційної роботи складається зі вступу, трьох розділів, висновків та додатків. У першому розділі розкрито теоретичні засади створення прогресивних web-застосунків та основні підходи до оптимізації їх продуктивності, проаналізувано сучасний стан мережевих сервісів у сфері діяльності електронних бібліотек. У другому розділі обґрунтовано вибір технологій і засобів розробки прогресивного веб-застосунку. У третьому розділі описано проєктування та програмну реалізацію прогресивного web-застосунку бібліотеки на основі моделі RAIL. У спеціальній частині з охорони праці розглядаються питання охорони праці та безпеки у надзвичайних ситуаціях.

Дипломна робота містить ___ сторінку (без додатків), ___ рисунків, ___ таблиці, ___ джерел, ___ додатки.

Ключові слова: прогресивний web-застосунок, модель RAIL, технологія Service Workers, відгук, анімація, очікування, завантаження.

ABSTRACT

to the master's qualification work
by the student of the group 601 of Petro Mohyla Black Sea National University

Oliinyka Mykhaila Hennadiiovycha

on the subject: «**OPTIMIZING THE PRODUCTIVITY OF THE LIBRARY'S PROGRESSIVE WEB APPLICATION BASED ON THE RAIL MODEL**»

Leader: Ph.D., associate professor Bolyubash Nadiya Mikolaivna

The master's thesis is devoted to the development and implementation of the software implementation of a progressive web application of the library, the optimization of the performance of which is implemented using the RAIL model. What is relevant in the conditions of increasing requirements for the quality and productivity of electronic libraries, since the RAIL model provides optimization of productivity indicators in the development process.

Object of research – optimization of the performance of web applications.

Subject of research – progressive web applications to support the activities of libraries with optimization of their performance based on the RAIL model.

The purpose of the study is to improving the efficiency of the electronic library by creating a progressive web application implemented on the basis of the RAIL user-oriented performance optimization model.

The master's qualification work consists of a professional, methodical and special part on labor protection. The explanatory note of the qualification work consists of an introduction, three sections, conclusions and appendices. In the first chapter, the theoretical foundations of the creation of progressive web-applications and the main approaches to optimizing their productivity are disclosed, the current state of network services in the field of electronic libraries is analyzed. The second section substantiates the choice of technologies and tools for developing a progressive web application. The third section describes the design and software implementation of the library's progressive web application based on the RAIL model. The special part on labor protection deals with issues of labor protection and safety in emergency situations.

Thesis contains ___ page (without appendices), ___ figures, ___ tables, ___ sources, ___ appendics.

Key words: progressive web app, model RAIL, technology Service Workers, response, animation, idle, load.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

БД – база даних

FCP – First Contentful Paint

FID – First Contentful Paint

HTML – Hyper Text Markup Language

JS – JavaScript

LCP – Largest Contentful Paint

PWA – Progressive Web App

PWM - Progressive Web Metrics

RAIL – Response Animation Idle Load

SW - Service Workers

Пояснювальна записка

до магістерської кваліфікаційної роботи

на тему:

«ОПТИМІЗАЦІЯ ПРОДУКТИВНОСТІ ПРОГРЕСИВНОГО WEB-ЗАСТОСУНКУ БІБЛІОТЕКИ НА ОСНОВІ МОДЕЛІ RAIL»

Спеціальність 122 «Системний аналіз»

122 – МКР – 601.21710120

Виконав студент 6-го курсу, групи 601
_____ М. Г. Олійник
«16» лютого 2023 р.

Керівник: канд. техн. наук, доцент
_____Н. М. Болюбаш
«16» лютого 2023 р.

Миколаїв – 2023

ВСТУП

Актуальність. В умовах інформатизації суспільства зростає кількість додатків, які взаємодіють із користувачем через web-інтерфейс та підвищуються вимоги до їх якості, надійності і продуктивності у різних галузях оточуючі дійсності та у бібліотечній сфері зокрема. Це обумовило появу нових трендів та ідей у сфері розробки програмного забезпечення, націлених на оптимізацію продуктивності web-застосунків. Одним із шляхів такого вдосконалення є створення прогресивних web-застосунків (англ. Progressive Web App, PWA) на основі моделі RAIL (англ. Response Animation Idle Load), яка забезпечує оптимізацію показників продуктивності, орієнтованих на користувача, із забезпеченням розширених можливостей надійності та доступності у процесі їх розробки.

Прогресивні web-застосунки є новим поколінням web-застосунків, які можуть адаптивно переналаштовуватися під параметри пристрою, на якому їх відкрито (смартфоні, планшеті, десктопному ПК, ноутбукові, нетбуку), працюючи на усіх ОС та браузерах таким чином, що робота з сайтом схожа на роботу з нативним web-застосунком. За рахунок завчасного кешування даних за технологією Service Workers PWA дозволяє взаємодіяти з ресурсом у режимі offline незалежно від з’єднання з Інтернетом. Існує багато метрик, які дозволяють виявити проблемні місця продуктивності web-застосунків, проте значна частина їх націлена на оптимізацію продуктивності уже розробленого застосунку. Більш ефективним є підхід, реалізований у моделі RAIL, де оптимізація продуктивності впроваджена у процес розробки.

Дослідження сучасного стану мережевих та мобільних бібліотечних сервісів дозволило виявити, що їх використання зростає швидкими темпами. Однак підходи до їх оптимізації не завжди є задовільними, оскільки інколи завантаження сторінки на екрані необхідно чекати 15-20 секун, що негативно впливає на попит користування бібіліотечними послугами у електронному форматі[12].

Мета дослідження – підвищення ефективності роботи електронної бібліотеки шляхом створення прогресивного web-застосунку, реалізованого на основі орієнтованої на користувача моделі оптимізації продуктивності RAIL.

Досягнення поставленої мети обумовлює необхідність вирішення наступних **завдань**:

1) дослідити теоретичні засади створення прогресивних web-засунків та здійснити аналіз мережевих сервісів у сфері діяльності бібліотек;
2) обґрунтуваня вибір технологій і засобів розробки прогресивного web-застосунку;
3) розробити та здійснити програмну реалізацію прогресивного web-застосунку із забезпеченням оптимізації його продуктивності на основі моделі RAIL.

Об’єктом дослідження є оптимізація продуктивності web-застосунків.

Предметом дослідження є прогресивні web-застосунки для підтримки діяльності бібліотек із оптимізацією їх продуктивності на основі моделі RAIL.

Методологічною основою дослідження є загальнонаукові аналітичні методи та методи оптимізації web-застосунків, які дозволили вивчити предмет та об’єкт дослідження, дослідити розвиток науково-методичних засад, напрямів та шляхів підвищення ефективності роботи електронної бібліотеки шляхом створення прогресивного web-застосунку, реалізованого на основі орієнтованої на користувача моделі оптимізації продуктивності RAIL.

Наукова новизна одержаних результатів дослідження полягає у тому, що автором: запропоновано та обґрунтовано напрями вдосконалення оптимізації web-застсоунків; одержали подальший розвиток підходи до підвищення ефективності роботи електронних бібліотек; узагальнено теоретичні засади створення прогресивних web-застсоунків.

Результати дослідження обговорювалися на Всеукраїнській науково-практичній конференції молодих вчених, аспірантів і студентів «Інформаційні технології та інженерія» (7-10 лютого 2023 року) та отримали схвалення.

Практичне значення отриманих результатів полягає в тому, що сформульовані теоретичні положення та практичні рекомендації щодо оптимізації продуктивності прогресивних web-застосунків можна застосувати у діяльності електронних бібліотек з метою розширення читацької аудиторії.

Структура магістерської роботи. Відповідно до мети, завдань і предмета дослідження, магістерська робота містить основну, методичну та спеціальну частини. Основна частина магістерської роботи складається із вступу, трьох розділів, висновку, списку використаних джерел та __ додатків. Загальний обсяг магістерської роботи – __ сторінок, із них основного тексту основної частини – __ сторінок, методичної частини – __ сторінок, спеціальної – __ сторінок. Кількість використаних джерел – __.

1 ТЕОРЕТИЧНІ ЗАСАДИ ОПТИМІЗАЦІЇ WEB-ЗАСТОСУНКІВ У СФЕРІ ДІЯЛЬНОСТІ ЕЛЕКТРОННИХ БІБЛІОТЕК

1.1 Прогресивні web-застосунки

Прогресивні web-застосунки є новим поколінням web-застосунків, які працюють як самостійні мобільні або нативні застосунки, використовуючи push-повідомлення та offline доступ. Прогресивний вебзастосунок підлаштовується під можливості браузера користувача таким чином, що робота з сайтом схожа на роботу з нативним веб-застосунком.

Перевагами PWA є вирішення таких проблем, як низька швидкість Інтернету, довге завантаження сайту й інтерактивність. До популярних PWA відносять сайти Alibaba, Forbes, The Weather Channel й MakeMyTrip, Uber – додаток для виклику таксі, Pinterest – web-сервіс для обміну малюнками та фото, Starbucks – всесвітня мережа кав’ярень. Twitter запустив сайт mobile.twitter.com, який працює як PWA. Основні переваги PWA, які роблять їх ефективними для користувачів електронної бібліотеки, полягають у їх прогресивності, адаптивності та незалежність від з’єднання з Інтернетом [41].

Основні переваги PWA, які роблять їх продуктивними полягають у наступному (табл. 1.1).

1. Прогресивність: вони не мають обмежень традиційних застосунків, підлаштовуючись під програмне налаштування користувача – працюють на усіх ОС та браузерах.

2. Адаптивність: дизайн прогресивного вебзастосунку є адаптивним і переналаштовується під розміри та параметри пристрою, на якому він відкритий – смартфоні, планшеті, десктопному ПК, ноутбукові, нетбуку і може отримувати доступ до функцій обладнання.

3. Незалежність від з'єднання з Інтернетом: застосунок за рахунок завчасного кешування даних за технологією Service Workers дозволяє взаємодіяти з ресурсом у режимі offline.

4. Подібність до застосунків: незважаючи на те, що PWA виходять за рамки звичайних застосунків, вони підтримують структуру, подібну застосункам. У цьому полягає основна відмінність PWA від вебзастосунку – він містить інтерактивні функції, які викликаються користувачем і дозволяє взаємодіяти з ним як з нативним застосунком.

Таблиця. 1.1 – Характеристики PWA

Харакеристика	Опис
Виявлення	Застосунок можна знайти з результатів пошуку в Інтернеті
Встановлюванність	Застосунок можна встановити на робочому столі
Повторно задіюванні	Програма може отримувати push-сповіщення, навіть якщо вона неактивна.
Незалежність від мереж	Застосунок працює без доступу до мережі Інтернет
Адаптивний	Інтерефейс може масштабуватися згідно з екраном користувача
Безпека	Застосунок може працювати по протоколу https та використовувати інші міри безпеки

Нативні застосунки і прогресивні застосунки пропонують користувачеві практично однакову взаємодію, але вони різні насамперед з технічного боку. Річ у тому, що нативні застосунки розробляються під кожну платформу окремо (наприклад, iOS або Android), встановлюються на гаджет, тому весь застосунок зберігається на пристрої користувача [20]. Прогресивні веб-застосунки можуть використовуватися в браузері, на телефон або планшет встановлюється лише оболонка, яка забезпечує миттєвий доступ та завантаження застосунку. При цьому

вони можуть бути встановлені й однаково добре функціонують будь-якому пристрої з будь-якою операційною системою [3].

У результаті проведеного дослідження установлено, що до базових компонент PWA відносять:

1) маніфест застосунку (англ. Web App manifest) – для надання нативних функцій, таких як іконка застосунку на робочому столі тощо;

2) технологію Service Workers – для фонових завдань і роботи в offline-режимі;

3) архітектуру Application Shell (оболонка застосунку): для швидкого завантаження з Service Workers.

Service Workers є основою PWA застосунку. Це проксируючий шар між фронтендом та бекендом, що знаходиться у браузері. Всі запити браузера проходять через нього. Даний поділ на два незалежні шари дозволив зробити перехід звичайного веб сайту в PWA максимально простим. Зі сховищ у Service Worker'a є доступ до Cache Storage для web ресурсів, та IndexDB для даних.

PWA вимагає, щоб усі ресурси сайту передавалися за протоколом HTTPS. SSL сертифікат можна отримати безкоштовно, деякі хостери роблять за вас. Але критично, щоб на сайті не було посилань на незахищені ресурси - деякі браузери просто не відображатимуть сайт у цьому випадку.

Архітектура Application Shell – це шаблон графічного інтерфейсу. Наприклад, візьмемо середній сайт з хедером, футером та секціями посередині. Умовно кажучи, виріжемо з нього контент поточної сторінки і всю динамічну інформацію, статика, що залишилася – App Shell. Суть у тому, що App Shell зберігається на клієнті та завантажується при запуску програми, а потім уже в нього вантажиться з мережі динамічна інформація [2].

Web App manifest. JSON файл, декларативно визначальний для браузера назву програми, іконку, як виглядатиме PWA (fullscreen, standalone та ін) та деякі інші параметри. Дозволяє встановити PWA як окремий додаток на домашній екран смартфона.

Ключовою перевагою даної технології є те, PWA працює для всіх користувачів у всіх браузерах і має можливість своєчасного відновлення, надаючи користувачам завжди актуальний функціонал і релевантну інформацію. Новий опублікований контент приходить користувачеві у вигляді відновлення, як тільки користувач підключається до Інтернету. Оболонка й уміст застосунку після кэшування завжди завантажуються з локального сховища. Також крім системи відновлень є Push-повідомлення, які працюють за рахунок інтерфейсів Notifications Push API й Notifications API. Дані інтерфейси підвищують імовірність повторного відвідування PWA користувачем.

Користувач може одержувати дані, відправлені із сервера, які можуть відображатися як повідомлення. Повідомлення прогресивних web-застосунків повністю природні й схожі на повідомлення нативних додатків.

Крім цього, у плані доступності PWA варто відзначити збережену структуру web-сайтів, що підходить для всіх форм, факторів і розмірів пристроїв: мобільних, настільних і планшетних. Адаптивна функція досягається за рахунок дизайну, концепції гнучкої сітки й медіа-запитів CSS3. Доступність даних додатків також обумовлюється маніфестом web-застосунку, за рахунок чого пошукові системи ідентифікують PWA як застосунок. Це збільшує ймовірність відображення в пошукових системах у порівнянні з нативным застосунками.

Впровадження Service Workers дозволяє PWA працювати в автономному режимі й забезпечувати гарну продуктивність навіть у локальній мережі. Додаток розглядає втрату підключення до Інтернет не як помилку, а як випадок, у якому можна запланувати й обробити без зволікання відкладені завдання.

Таким чином, прогресивний веб застосунок – це технологія у web-розробці, яка візуально і функціонально трансформує сайт у застосунок (мобільний застосунок в браузері). Статистика говорить про те, що 66 % користувачів не скачують жодного застосунку, а більшу частину свого часу – приблизно 85%, користувач проводить у застосунках. Як правило, це месенджери, соцмережі, відеохостинг. При цьому мобільний браузер також у багатьох випадках не є

пріоритетною формою виходу в Інтернет. За даними comScore, у 2017 році користувачі смартфонів і планшетів втратили 87% свого часу на роботу у застосунках порівняно з 13% у браузері. З того часу цей відсоток змінився, збільшивши частку мобільних додатків, на «ринку» уваги користувачів. PWA дозволяє користувачам установлювати улюблений сайт на смартфон, на прикладі того як це працює зі звичайними мобільними чи нативними додатками [22].

PWA застосунки набувають все більшої популярності оскільки можуть допомогти значно збільшити охоплену аудиторію, а також час який вона проводить в застосунку. А отже збільшити дохід для комерційних проектів і популярність для некомерційних. Компанії, що запустили прогресивні веб-додатки, досягли вражаючих результатів. Наприклад, у Twitter кількість сторінок, що переглядаються за сеанс, збільшилася на 65%, кількість твітів — на 75%, а показник відмов знизився на 20%, при цьому розмір програми зменшився більш ніж на 97%. Після переходу на PWA трафік Nikkei збільшився у 2,3 рази, кількість підписок зросла на 58%, а кількість активних користувачів за день – на 49%. Hulu перейшла з платформо-залежної десктопної програми на прогресивний веб-додаток, і кількість повторних відвідувань збільшилася на 27%. За рахунок підвищення продуктивності завантаження сторінок можна стверджувати, що PWA є новим способом компаній привернути та зберегти увагу користувачів до тих товарів та послуг, які вони надають [36].

1.2 Розвиток підходів до оцінки продуктивності web-застонків – прогресивні web-метрики

Під продуктивність web-застосунків розуміють ресурси, які досить швидко за запитами та діями користувача відображать свій вміст на екран. Швидкість завантаження та відгуку сайту є дуже важливою для приємного досвіду використання клієнтом, і взаємодії з ним. Якщо сайт працюватиме занадто повільно є ризик втратити не тільки відвідувачів, а й потенційних клієнтів.

Користувачі не люблять чекати, очікування на сайті довше 3 секунд збільшує ризик відмов на 32% [36].

Дослідження підходів до оптимізацій web-астосунків дозволило виявити загальні поради для покращення швидкості роботи сайтів. Google надає базові рекомендації для покращення швидкості завантаження сайту на своєму порталі:

1) уникати редиректів цільової сторінки;
2) увімкнути стиск;
3) поліпшити час відгуку сервера;
4) використовувати кешування браузера;
5) мінімізувати ресурси;
6) оптимізувати зображення;
7) оптимізувати доставку CSS;
8) віддавати перевагу видимому контенту;
9) видалити JavaScript, що блокує рендеринг.

Але ці рекомендації зосереджується лише на часі, необхідному для завантаження сайтів.

Доволі популярними на просторах інтернету є Технічне CEO. Основою технічного SEO є надійна архітектура веб-сайту. Не можна просто опублікувати випадкову колекцію сторінок і публікацій. Оптимізована для пошукових систем архітектура сайту допоможе користувачам у всьому вашому сайті та полегшить Google сканування та індексування ваших сторінок [37].

UX також враховує бізнес-цілі та завдання. Найкращі практики UX зосереджені на покращенні якості взаємодії з користувачем. За словами Пітера Морвілла, фактори, що впливають на UX, включають такі положення.

1. *Корисно*: вміст сторінок має бути унікальним і задовольняти потреби користувачів.
2. *Зручність використання*: веб-сайт має бути простим у використанні та навігації.
3. *Бажано*: елементи дизайну та бренд мають викликати емоції та вдячність у

користувача.

4. *Зручне розташування елементів*: об’єднайте елементи дизайну та навігації, щоб користувачі могли легко знаходити те, що їм потрібно.
5. *Доступність*: вміст має бути доступним для всіх, у тому числі для 12,7% людей з обмеженими можливостями.
6. *Надійність*: сайт має бути надійним, аби він не викликав сумнівів у своїй надійності.
7. *Цінність*: ваш сайт має бути цінним для користувача з точки зору досвіду та для компанії з точки зору позитивної рентабельності інвестицій.

Понятя продуктивності web-застосунку є відносним з наступних причин.

1. Сайт може одночасно бути швидким для одного користувача і повільним для іншого, за умови, що перший користувач працює на потужному пристрої в швидкій мережі, а другий - на пристрої з нижнього цінового сегмента в повільній мережі.

2. Два сайти можуть завантажитися одночасно, але може здатися, що один завантажується швидше (якщо він завантажує контент поступово, а не чекає до останнього, щоб відобразити хоч щось).

3. Може здатися, що сайт спочатку завантажується швидко, але потім відповідає дуже повільно (або взагалі не реагує) на дії користувача.

Тому продуктивність слід розглядати з погляду об'єктивних критеріїв, які можна виміряти кількісно з достатньою мірою точності. Ці критерії відомі як метрики.

Вимірювання продуктивності web-застосунку є складним завданням, тому розробники разом із спільнотою працюють над розробкою прогресивних веб-метрик (англ. Progressive Web Metrics, PWM). Якийсь час тому було дві основні події для вимірювання продуктивності [39].

1. DOMContentLoaded викликалося, коли сторінку було завантажено, але скрипти тільки почали виконуватися.

2. Load подія викликалася, коли сторінка була повністю завантажена і відпрацювали всі скрипти, тому користувач міг повноцінно взаємодіяти зі сторінкою.

Однак вказані метрики не завжди є оптимальними. Проблема DOMContentLoaded полягає у тому, що час парсингу та виконання скриптів може бути занадто великим, якщо скрипти дуже великі, наприклад, на мобільних пристроях. Скажімо, таймлайн, який був виміряний з імітацією роботи через мережу 3G, показує час приблизно 10 секунд до повного завантаження.

Незважаючи на те, що load чітко визначений момент життєвого циклу сторінки, він не обов'язково відповідає тому, що хвилює користувача. Наприклад, сервер може миттєво завантажити мінімальну сторінку, а потім відкласти вибірку та відображення контенту на сторінці на кілька секунд вже після спрацювання події load. Хоча технічно така сторінка може мати швидкий час завантаження, вона не відповідатиме часу завантаження сторінки для користувача.

Іншими словами, завантаження йде надто довго, щоб якось аналізувати проблеми продуктивності. Спиратися на ці метрики сьогодні не є оптимальним, тому що за проведеними дослідженнями не усі користувачі будуть очікувати сторінку, якщо вона завантажуєтсья більше, наж за 2 секунди. Тому доцільним є розробка метрик, які орієнтовані на користувача.

Протягом останніх кількох років розробники Chrome у співпраці з W3C Web Performance Working Group працювали над стандартизацією набору нових API та показників, які точніше вимірюють те, як користувачі сприймають продуктивність веб-сторінки. Ці метрики створені на підставі важливих для користувача наступних питань.

1. Чи відбувається завантаження: навігація розпочалася успішно, сервер відповів?
2. Сторінка придатна для використання: чи достатньо відображеного контенту для того, щоб користувачі могли з ним взаємодіяти?

3. Сторінка придатна для використання: чи можна взаємодіяти зі сторінкою: чи можуть користувачі взаємодіяти зі сторінкою чи вона зайнята?
4. Сторінка чудова: взаємодії з елементами сторінки проходять плавно та природно, без затримок та ривків?

Є кілька інших типів метрик, які стосуються того, як користувачі сприймають продуктивність.

1. Сприйнята швидкість завантаження: як швидко сторінка може завантажуватися та відображати всі візуальні елементи на екрані.
2. Чутливість під час завантаження: як швидко сторінка може завантажуватися та виконувати будь-який необхідний код JavaScript для того, щоб компоненти вчасно реагували на взаємодію з користувачем.
3. Чутливість у середовищі виконання: як швидко після завантаження сторінка може реагувати на взаємодію користувача.
4. Візуальна стабільність: чи змінюються елементи на сторінці несподіваним користувачам, що потенційно заважає взаємодії?
5. Плавність: чи відображаються переходи та анімація з постійною частотою кадрів і чи плавно відбувається перехід з одного стану до іншого?

Серед великої кількості метрик для оцінювання характеристик продуктивності web-застосунку головними для відслідковувати є наступні.

1. *Перше відображення контенту (FCP)*: Вимірює час від початку завантаження сторінки до відображення на екрані будь-якої частини вмісту сторінки (лабораторна, польова).

2. *Швидкість завантаження основного контенту (LCP)*: вимірює час від початку завантаження сторінки до моменту, коли на екрані з'являється найбільший текстовий блок або елемент зображення (лабораторна, польова).

3. *Час очікування до першої взаємодії з контентом (FID)*: вимірює час від моменту, коли користувач вперше взаємодіє з сайтом (тобто, коли він клацає

посилання, натискає кнопку або використовує елемент управління на основі JavaScript, що настроюється) до моменту, коли браузер дійсно зможе відповісти на це взаємодія (польова).

4. *Час до інтерактивності (TTI)*: вимірює час від початку завантаження сторінки до моменту її візуального відображення, завантаження початкових сценаріїв (якщо є) і здатності сторінки швидко і надійно реагувати на введення користувача (лабораторна).

5. *Загальний час блокування (TBT)*: вимірює загальну кількість часу між FCP та TTI, коли основний потік був заблокований на досить довгий час, щоб запобігти реакції на введення (лабораторна).

6. *Сукупне зміщення макета (CLS)*: вимірює сукупну оцінку всіх несподіваних зрушень макета, що відбуваються між моментом початку завантаження сторінки та зміною стану життєвого циклу на приховане. (лабораторна, польова) [40].

У цей перелік увійшли далеко ще не всі метрики продуктивності, орієнтовані користувача (наприклад, час відгуку і плавність виконання).

Наведені вище метрики продуктивності дозволяють отримати загальне уявлення про характеристики продуктивності більшості сайтів в Інтернеті. Загальний набір метрик дає можливість порівнювати отримані результати з результатами конкурентів. Однак буває, що конкретний сайт до певної міри унікальний. Отже, щоб одержати повну картину продуктивності потрібно використовувати додаткові метрики. Наприклад, метрика LCP вимірює швидкість завантаження основного контенту, але іноді найбільший елемент не є частиною основного вмісту сторінки, і, отже, LCP може бути марною для аналізу продуктивності.

Для вирішення таких випадків було стандартизовано API-інтерфейси нижчого рівня, які можуть бути корисні для реалізації власних показників користувача: User Timing API, Long Tasks API, Element Timing API, Navigation Timing API, Resource Timing API, Server timing.

У деяких випадках вводять нові показники для охоплення областей, що відсутні серед переліку загальних метрик. Найбільш цінні аналітичні дані нададуть показники, спеціально розроблені для конкретного сайту.

Відслідковуваня продуктивності зазвичай відбувається уже після розробки застосунку під час його тестування та впровадження. Це не самий оптимальний підхід, оскільки він передбачає оптимізацію застосунку уже після його розробки. Тому доцільніше застосувати підхід, який дозволяє забезпечувати продуктивність web-застосунку у процесі його розробки [26].

1.3 Модель RAIL

Застосування моделі RAIL при розробці прогресивного web-застосунку дозволяє задачу підвищення продуктивності впровадити у процес розробки, оцінюючи ті чи інші архітектурні рішення з точки зору їх впливу на швидкість роботи застосунку. У результаті проведеного дослідження установлено, що модель RAIL передбачає наступний набір інструкцій для вимірювання продуктивності web-застосунку, виділяючи основні аспекти у його життєвому циклі (рис. 1.1):

1) відповідь (англ. Response) на виконання дій користувача повинна надходити у час, не більший ніж 100 мілісекунд;

2) анімація (англ. Animation) повинна мати затримку не більшу, ніж 60 кадрів у секунду;

3) завантаження (англ. Load) повинно становити не більше 1-2 секунд, тому при звернені користувача до застосунку завантажується тільки його базова версія;

4) очікування (англ. Idle) у роботі застосунку – його простій між діями користувача повинен бути використаний для виконання відкладеної роботи, зокрема завантаження тих частин програми, які ще не були завантажені.

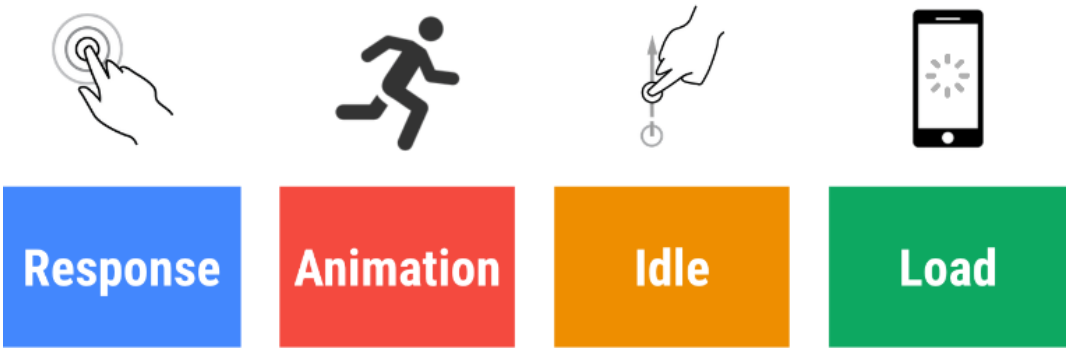

Рисунок 1.1 – Основні компоненти моделі RAIL

Модель оптимізації web-застсоунку RAIL була розроблена працівниками Google, які вирішили розробити модель, притримуючись якої можна вивести оптимізацію сайта на вищий рівень, тим самим збільшивши кількість задоволених користувачів[39].

RAIL – це абревіатура від англійських слів response (відгук), animation (анімація), idle (очікування) та load (завантаження), в якій досвід користувача розбивається на кілька ключових дій. Наприклад, клік, свайп (рух із натиснутою клавішею миші або тачпада), скролінг, завантаження. RAIL встановлює цілі щодо продуктивності для цих дій. Наприклад, від кліка до відтворення дій по цьому кліку має пройти не більше 100 мілісекунд.

RAIL забезпечує структуру для планування робіт щодо покращення продуктивності. Дизайнери та розробники можуть визначити ті моменти, поліпшення яких може дати найбільший вплив, та працювати над ними.

Основна проблема з якою бореться кожна модель по оптимізації web-застосунків – збільшення швидкості відгуку застосунку. Але оскільки неможливо постійно намагатися покращити оптимізацію сайта, необхідно визначити показники до яких необхідно прагнути. Таке дослідження провів Якоб Нільсен, виклавши свою роботу «Response Times: The 3 Important Limits». На базі цих досліджень і була побудована модель Rail. А саме:

100 мілісекунд: необхідно відреагувати на дію користувача за цей час, і він сприйматиме, що реакція була негайною. Все, що довше за це створює відчуття затримки між дією і реакцією.

1 секунда: той період часу, протягом якого користувач відчуває природним виконання «завдання». Завданням може бути, наприклад, завантаження сторінки або зміна списку товарів при зміні фільтра.

16 мілісекунд: враховуючи, що екран оновлюється 60 разів на секунду на більшості пристроїв, за такий час на екрані повинен з’явитися кожен наступний кадр (1000/60 = 16). Люди дуже добре відстежують рух очима, і все, що повільніше за 60 кадрів на секунду, порушує їхні очікування. Хоча останній параметр найближчим часом може сильно змінитися, з поширенням екранів з частотою 90 та 120 герців.

RAIL передбачає оптимізацію усіх чотирьох параметрів: відгуку, анімації, очікування та завантаження, з урахуванням часу реакцій які очікують користувачі.

Відгук (англ. Response). Коли користувач натискає кнопку, необхідно «відгукнутися» на його дію так швидко, щоб він не помітив жодного лага (не більше 100 мілісекунд). Це стосується будь-якого елементу введення користувача, не важливо, чи натискає користувач на перемикач у формі або на звичайну кнопку. Якщо користувач не побачить відгук системи швидко, чи це вмикання або вимкнення перемикача або анімація натискання кнопки, у нього буде відчуватися розрив між дією та реакцією системи. Тоді у нього з’явиться відчуття, що система працює із затримкою.

Анімація (англ. Animation). Анімація є у всіх видах сучасних додатків. Під анімацією мається на увазі, звичайно, не елементи дизайну, накшталт обертаючихся елементів , а такі операції як скролінг, меню що висувається та інші подібні ефекти, пов'язані з тим, що вміст екрана повинен постійно змінюватися протягом якогось часу. Анімацію можна поділити за видами настуупним чином.

1. Візуальна або звичайна: включає традиційні анімовані заставки, індикатори завантаження, індикатори зміни стану тощо, тобто те, що виглядає як анімований фрагмент на екрані.

2. Анімація скролінгу: хоча зазвичай це не здається анімацією, технічно вона нею є. Користувач веде пальцем по екрану або скролює за допомогою миші або тачпада на звичайному комп'ютері, і зображення постійно змінюється.

3. Анімація перетягування (drag): коли користувач використовує якісь функції програми або сайту для зміни розташування елементу в середині розмітки сайту.

Щоб анімація виглядала безперервною, кожен кадр анімації повинен з’являтися на екрані менш ніж за 16 мілісекунд, тобто зі швидкістю 60 FPS (1 секунда / 60 кадрів = 16,6 мілісекунд на кадр).

Очікування (англ. idle). Усередині кожного додатку відбувається безліч процесів, але далеко не всі вони повинні працювати в такі критичні моменти, коли програма відпрацьовує взаємодію з користувачам типу «відгук» або «анімація».

Завантаження **(**англ. Load**).** Насамперед ми хочемо якнайшвидше показати користувачеві перший екран, на якому він повинен побачити досить корисну для себе інформацію. Просто показати йому шапку меню і потім чекати на появу іншої інформації – не годиться. Як тільки перший екран показаний користувачеві, програма або сайт повинні зберегти здатність відгукуватися на дії користувача, навіть якщо у фоні відбувається дозавантаження інших частин сторінки. Але користувач не повинен навіть у цей момент мати проблеми зі скролінгом, натисканнями або анімацією [37].

Впровадження моделі, орієнтованої на користувача, яка спрямована на підвищення продуктивності розролюваного web-застосунку за моделлю RAIL дозволяє оптимально підійти до підвищення йогго продуктивності у процесі його розробки. Це вигідно виділяє модель на фоні інших підходів до підвищення продуктивності web-застосунків.

1.4 Мережеві сервіси у діяльності електронних бібліотек

Електронні бібліотеки виникли нещодавно, проте вже за цей невеликий період існування вони встигли стати одним з найбільш перспективних каналів книгорозповсюдження. Здійснюється цифрова трнасформація бібліотечної сфери, перехід до електронних бібліотек. Якщо в невеликій публічній бібліотеці на одну книгу на рік припадає близько трьох видач (це вважається нормою), то в електронних бібліотеках із сучасним фондом набагато більше. Причому з точки зору працевитрат в електронній бібліотеці зусилля людини витрачаються тільки на завантаження, опис та підтримку роботи ПЗ, а на видачу, здачу та інші техпроцеси – не витрачаються і, отже, цифрова книговидача набагато дешевша за друковану.

На даний момент електронних бібліотек налічується величезна кількість з офіційним або піратським контентом.

Сайти з піратським контентом як правило йдуть по одному з двох шляхів. Або надають змогу користувачеві завантажувати книгу на власний пристрій. Звідки користувач може сам обирати через який додаток читати книгу. Один з плюсів такого способу – відсутня необхідність в постійному доступі до мережі інтернет. Прикладом такого сайту може бути електронна бібіліотека RoyalLib (RoyalLib.com). З іншого боку, існують сайти, які дозволяють читати книги онлайн, тим самим зберігаючи аудиторію на сайті якомога довше. До плюсів таких сайтів можна віднести можливість швидко перейти на іншу книгу, та відсутність необхідності засоряти пам’ять пристроїв.

За схожею моделлю працюють і сайти з офіційним контентом. Деякі з них дозволяють купувати та завантажувати книги на пристрій, деякі пропонують читати книги онлайн по підписці. Наприклад Google Books, та Yakoboo. І хоча моделі збуту в цих сайтів різні, більшість онлайн-бібліотек мають однакову структуру, яка є наступною.

1. Головна сторінка. На головній сторінці сайтів-бібліотек як правило можна

побачити декілька збірок книг, які є найпопулярнішими в своїх жанрах або нові книги (див. рис. 1.2, рис. 1.3)

2. Сторінка катологу. В каталогах електронної бібілотеки як правило реалізується можливість пошуку за фільтрами (жанри, автори, теги, тощо) (див. рис. 1.4).
3. Сторінка продукту. Ця сторінка містить детальну інформацію про обрану книгу та рекомендації по подальшим діям користувача з нею.

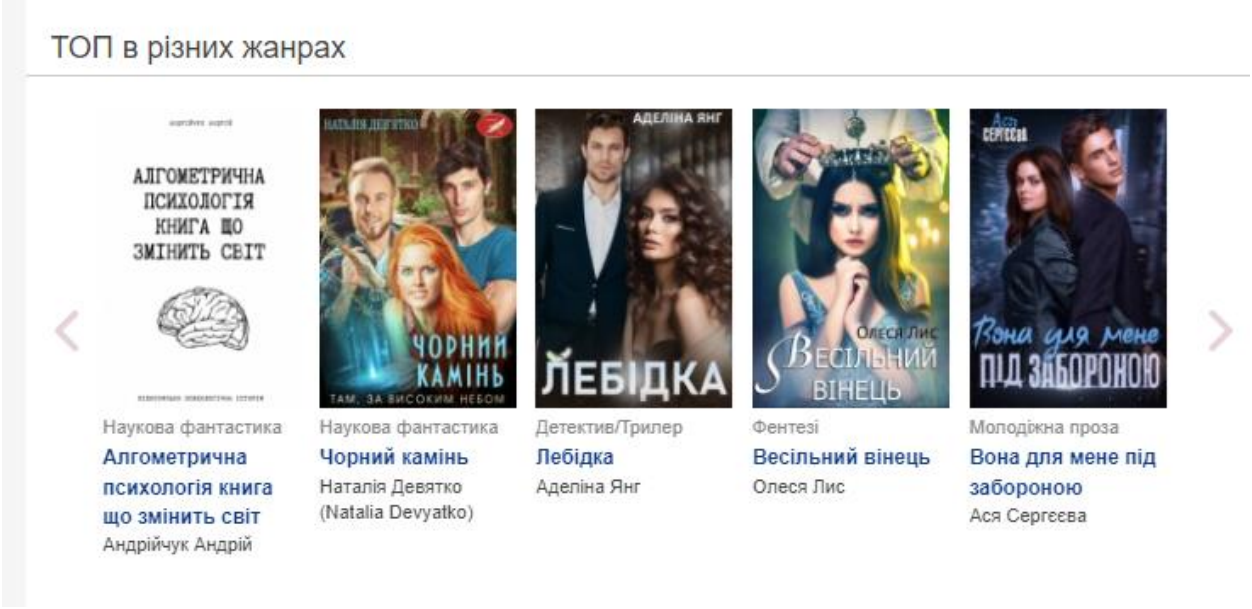

Рисунок 1.2 – Найпопулярніші книги в жанрах

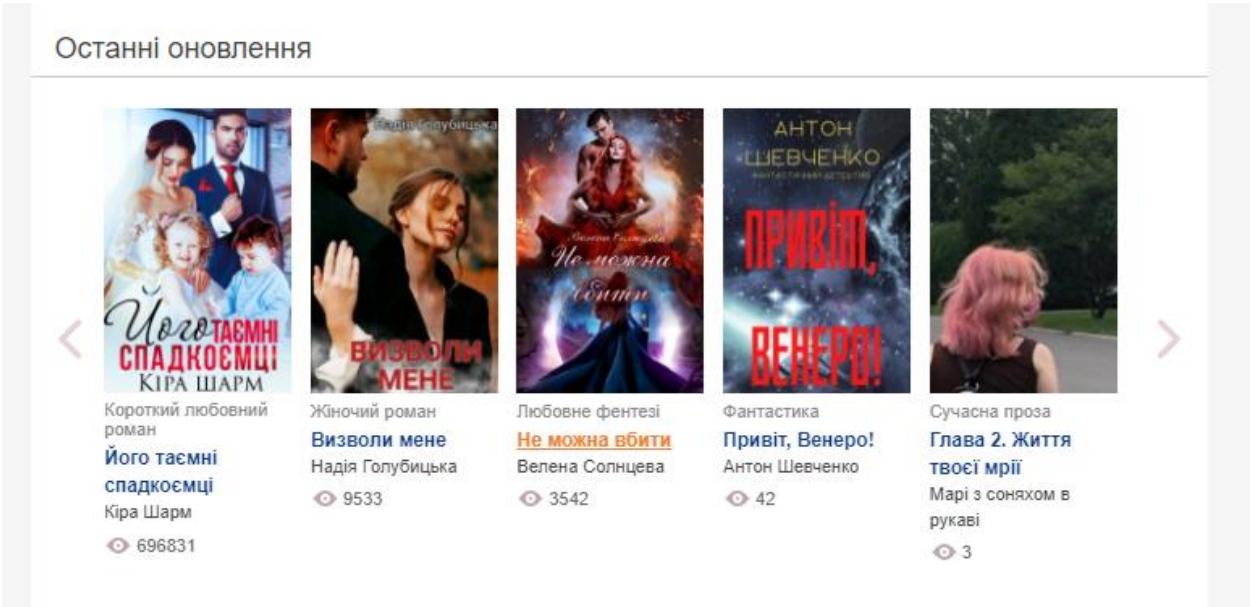

Рисунок 1.3 – Останні оновлення

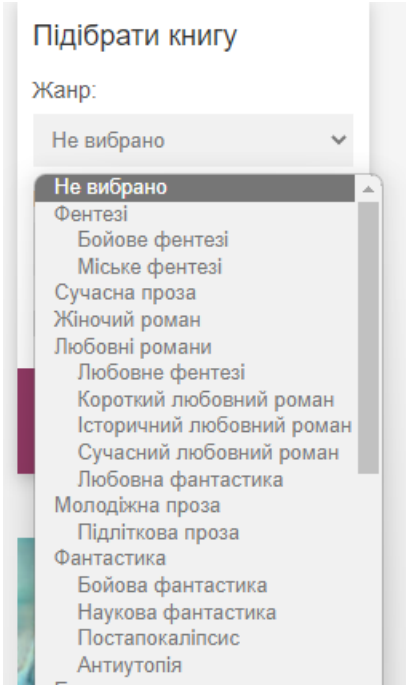

Рисунок 1.4 – Пошук по категоріям на сторінці каталогу

Проведенні дослідження продуктивності електронних ресурсів, які впроваджено у діяльність електронних бібліотек дозволило виявити, що тільки деякі з них андаптовані до роботи на мобільних пристроях та мають задовільні показники продуктивності. Швидкість роботи на десктопних пристроях є допоміжним параметром, оскільки на нього слід орієнтуватися тільки при доступі до бібліотеки з невисокою швидкістю Інтерент. Результати показали, що твільки невеликий процент електронних бібліотек має прийнятну швидкість роботи. Що потребує більш прогресивних підходів до розробки ресурсів електронних бібліотек.

1.5 Постановка задачі

Провівши аналіз підходів до оптимізації продуктивності web-застосунків було зроблено висновок про розробку прогресивного web-застосунку бібліотеки

на основі моделі RAIL, яка орієнтована на користувача та дозволяє оптимізувати продуктивність у процесі розробки застосунку.

Об’єктом дослідження є оптимізація продуктивності web-застосунків.

Предметом дослідження є прогресивні web-застосунки для підтримки діяльності бібліотек із оптимізацією їх продуктивності на основі моделі RAIL.

Метою дослідження є підвищення ефективності роботи електронної бібліотеки шляхом створення прогресивного web-застосунку, реалізованого на основі орієнтованої на користувача моделі оптимізації продуктивності RAIL.

Досягнення поставленої мети обумовлює необхідність вирішення наступних **завдань**:

1) дослідити теоретичні засади створення прогресивних web-засунків та здійснити аналіз мережевих сервісів у сфері діяльності бібліотек;
2) обґрунтуваня вибір технологій і засобів розробки прогресивного web-застосунку;
3) розробити та здійснити програмну реалізацію прогресивного web-застосунку із забезпеченням оптимізації його продуктивності на основі моделі RAIL.

Висновки до розділу 1

Установлено, що сьогодні цільова споживацька аудиторія все більше використовує Інтернет для отримання інформації та задаволення потреб у товарах та послугах. Це обумовило різке зростання вимог до продуктивності web-ресурсів та їх кросплатформенності та виникненню нового напряму – розробки прогресивних web-застоунків PWA. До базових компонент PWA віднесено: 1) маніфест застосунку – для надання нативних функцій, таких як іконка застосунку на робочому столі тощо; 2) технологію Service Workers – для фонових завдань і роботи в offline-режимі; 3) архітектуру Application Shell (оболонка застосунку): для швидкого завантаження з Service Workers.

Результати проведеного дослдіження показали, що тільки невеликий процент електронних бібліотек має прийнятну швидкість роботи. Що потребує більш прогресивних підходів до розробки ресурсів електронних бібліотек. У результаті проведеного дослідження установлено, що найбільш оптимальною моделлю оптимізації продуктивності прогресивних застосунків є орієнтована на користувача модель RAIL, яка оцінює архітектурні рішення з точки зору їх впливу на швидкість роботи застосунку: 1) відповідь (англ. Response) на виконання дій користувача повинна надходити у час, не більший ніж 100 мілісекунд; 2) анімація (англ. Animation) повинна мати затримку не більшу, ніж 60 кадрів у секунду; 3) очікування (англ. Idle) – простій між діями користувача повинен бути використаний для виконання відкладеної роботи; 4) завантаження (англ. Load) повинно становити не більше 1-2 секунд.

2 ТЕХНОЛОГІЇ ТА ІНСТРУМЕНТАЛЬНІ ЗАСОБИ РОЗРОБКИ ПРОГРЕСИВНОГО WEB-ЗАСТОСУНКУ

2.1 Засоби програмування дизайну: HTML, CSS

HTML – (hyper text markup language), стандартизована мова гіпертекстової розмітки документів для перегляду веб-сторінок у браузері. Веб-браузери отримують HTML документ від сервера за протоколами HTTP/HTTPS або відкривають з локального диска, далі інтерпретують код в інтерфейс, який відображатиметься на екрані монітора [28].

Мова гіпертекстової розмітки HTML була розроблена британським вченим Тімом Бернерсом-Лі приблизно в 1986-1991 роках у стінах ЦЕРНу в Женеві у Швейцарії [3]. HTML створювався як мова для обміну науковою та технічною документацією, придатний для використання людьми, які не є фахівцями в галузі верстки. HTML успішно справлявся із проблемою складності SGML шляхом визначення невеликого набору структурних та семантичних елементів – дескрипторів. Дескриптори також часто називають "тегами". За допомогою HTML можна просто створити відносно простий, але красиво оформлений документ. Крім спрощення структури документа, HTML внесена підтримка гіпертексту. Мультимедійні можливості було додано пізніше.

Першим загальнодоступним описом HTML був документ "Теги HTML", вперше згаданий в Інтернеті Тімом Бернерсом-Лі наприкінці 1991. У ньому описуються 18 елементів, що становлять початковий, відносно простий дизайн HTML. За винятком тега гіперпосилання, на них сильно вплинув SGMLguid, внутрішній формат документації, що базується на стандартній узагальненій мові розмітки (SGML), в CERN. Одинадцять із цих елементів усе ще існують у HTML4 [31].

Спочатку мова HTML була задумана і створена як засіб структурування та форматування документів без їх прив'язки до засобів відтворення (відображення).

В ідеалі, текст з розміткою HTML повинен був без стилістичних та структурних спотворень відтворюватися на устаткуванні з різним технічним оснащенням (кольоровий екран сучасного комп'ютера, монохромний екран органайзера, обмежений за розмірами екран мобільного телефону або пристрою та програми голосового відтворення текстів).

Однак сучасне застосування HTML дуже далеке від його початкового завдання. Наприклад, тег <table> призначений для створення в документах таблиць, але іноді використовується для оформлення розміщення елементів на сторінці. З часом основна ідея платформонезалежності мови HTML була принесена в жертву сучасним потребам у мультимедійному та графічному оформленні.

Елементи HTML є будівельними блоками сторінок HTML. За допомогою HTML різні конструкції, зображення та інші об'єкти, такі як інтерактивна веб-форма, можуть бути вбудовані в сторінку, що відображається. HTML пропонує засоби для створення заголовків, абзаців, списків, посилань, цитат та інших елементів. Елементи HTML виділяються тегами, записаними з використанням кутових дужок. Такі теги, як <img /> і <input />, безпосередньо вводять контент на сторінку. Інші теги, такі як <p>, оточують та оформляють текст у собі і можуть включати інші теги як поделементи. Браузери не відображають HTML-теги, але використовують їх для інтерпретації вмісту сторінки.

CSS (Cascading Style Sheets) — каскадні таблиці стилів. Це код, який ви використовуєте для стилізації веб-сторінки. Основи CSS допоможуть вам зрозуміти, що потрібно для початку роботи.

До появи CSS оформлення веб-сторінок здійснювалося виключно засобами HTML, безпосередньо всередині документа. Однак з появою CSS став можливим принциповий поділ змісту та подання документа.

За рахунок цього нововведення стало можливим легке застосування єдиного стилю оформлення для подібних документів, а також швидка зміна цього оформлення.

Переваги CSS є наступними.

1. Декілька дизайнів сторінки для різних пристроїв перегляду. Наприклад, на екрані дизайн буде розрахований на велику ширину, під час друку меню не виводитиметься, а на КПК та стільниковому телефоні меню йтиме за вмістом.
2. Зменшення часу завантаження сторінок сайту за рахунок перенесення правил подання даних до окремого файлу CSS. У цьому випадку браузер завантажує лише структуру документа та дані, які зберігаються на сторінці, а подання цих даних завантажується браузером лише один раз і може бути закешоване.
3. Простота подальшої зміни дизайну. Не потрібно редагувати кожну сторінку, а достатньо лише змінити CSS-файл.
4. Додаткові можливості оформлення. Наприклад, за допомогою CSS-верстки можна зробити блок тексту, який решта тексту буде обтікати (наприклад для меню) або зробити так, щоб меню було завжди видно при прокручуванні сторінки.

Недоліки CSS:

1) різне відображення верстки в різних браузерах (особливо застарілих), які по-різному інтерпретують ті самі дані CSS;
2) часто зустрічається необхідність на практиці виправляти не тільки один CSS-файл, але і теги HTML, які складним і ненаглядним способом пов'язані з селекторами CSS, що іноді зводить нанівець простоту застосування єдиних файлів стилів і значно збільшує час редагування та тестування.

SCSS, SASS, LESS. SASS – розширення CSS, яке надає потужності та елегантності цій простій мові. Sass дасть вам можливість використовувати змінні, вкладені правила, міксини, інлайнові імпорти та багато іншого, все з повністю сумісним із CSS синтаксисом. Sass допомагає зберігати великі таблиці стилів добре організованими, а невеликим стилям працювати швидко [32].

Для SASS є два синтаксиси. Перший, відомий як SCSS (Sassy CSS) - це розширений синтаксис CSS. Це означає, що кожна валідна таблиця стилів CSS це валідний SCSS файл, що несе в собі ту ж саму логіку. Більш того, SCSS розуміє більшість хаків у CSS. Файли, які використовують цей синтаксис мають розширення .scss [32].

Другий і старіший синтаксис, також відомий як краєний синтаксис або іноді просто Sass, дає більш стислу можливість роботи з CSS. Він використовує відступи замість дужок, що відокремлює вкладення селекторів і нові рядки замість точок з комою для поділу властивостей. Іноді люди знаходять такий спосіб простіше для розуміння та швидше для написання, ніж SCSS. За фактом такий синтаксис має той же функціонал, хоча деякі з них мають трохи інший підхід. Файли, що використовуються цей синтаксис мають розширення .sass.

Будь-який синтаксис може імпортувати файли, написані в іншому.

2.2. JavaScript-фреймворк AngularJS

AngularJS – JavaScript-фреймворк із відкритим програмним кодом, який розробляє Google. Призначений для розробки односторінкових додатків, що складаються з одної HTML сторінки з CSS і JavaScript. Його мета — розширення браузерних застосунків на основі шаблону Модель-вид-контролер (MVC), а також спрощення їх тестування та розробки [24].

Фреймворк працює зі сторінкою HTML, що містить додаткові атрибути і пов'язує області вводу або виводу сторінки з моделлю, яка є звичайними змінними JavaScript. Значення цих змінних задаються вручну або отримуються зі статичних або динамічних JSON-даних [10].

За даними служби аналізу JavaScript для Libscore, AngularJS використовується на вебсайтах Wolfram Alpha, NBC, Walgreens, Intel, Sprint, ABC News та близько 12,000 інших сайтів з 1 мільйона протестованих у жовтні 2016

року AngularJS наразі входить до трійки проектів, що набрали найбільшу кількість зірок на GitHub.

Як платформа Angular включає:

1) заснований на компонентах фреймворк для створення веб-додатків, що масштабуються;
2) набір добре інтегрованих бібліотек, що охоплюють широкий спектр функцій: маршрутизація, керування формами, клієнт-серверна взаємодія тощо;
3) набір інструментів розробника, які допоможуть вам розробляти, збирати, тестувати та оновлювати ваш код.

Створення за допомогою Angular застосунок, дає переваги платформи, яка може масштабуватися від проекту, який розробляє одна особа, до програм корпоративного рівня. Angular розроблено, щоб максимально спростити оновлення, тому ви можете використовувати останні розробки з мінімумом зусиль. А чудово те, що екосистема Angular складається з величезної спільноти, що включає більш ніж 1.7 мільйона розробників, авторів бібліотек і творців контенту [13].

Розберемо, що являє собою архітектура Angular програми. Сам фреймворк складається з декількох бібліотек (або модулів), кожна з яких містить певний функціонал, а кожен модуль складається з сукупності класів та їх властивостей і методів. Кожен клас має своє функціональне призначення.

Розберемо модулі, саме з них починається проектування архітектури Angular програми. Кожен із них має власний набір структурних елементів:

1) *component* - відповідає за частину web-сторінки і включає HTML-шаблон, CSS-стилі і логіку поведінки;
2) *service* - постачальник даних для component;
3) *directive* – перетворює певну частину DOM заданим чином.
4) Все перераховане вище збирається в кореневий модуль, який загальноприйнято називається AppModule.

Кореневий модуль може бути лише один, але може використовувати функціонал інших модулів, оголошених в об'єкті декоратора @NgModule() у властивості imports.

Компонент – це частина інтерфейсу додатку із власною логікою. Вся видима частина Angular App реалізується за допомогою компонентів, тому можна почути, що архітектура Angular компонентна.

Сервіси потрібні для надання даних компонентам. Це можуть бути не тільки запити до сервера, а й функції, що перетворюють вихідні дані заданого алгоритму. Вони дозволяють архітектурі Angular програми бути більш гнучкою та масштабованою. Завдання сервісу має бути вузьким і строго визначеним.

Директиви. Структурні директиви в Angular відповідають за маніпулювання елементами, їх зміну та видалення усередині шаблону компонента. Структурна директива застосовується до основного елемента, який змінюється та оновлюється разом зі своїми дочірніми елементами відповідно до поведінки структурної директиви. Angular має декілька вбудованих структурних директив, таких як ngFor, ngSwitch і ngIf.

Angular CLI є офіційним інструментом для ініціалізації та роботи з проектами Angular. Це позбавить вас від клопоту складних конфігурацій і інструментів для створення, таких як TypeScript, Webpack тощо [10].

Після встановлення Angular CLI вам потрібно буде запустити одну команду, щоб створити проект, і іншу, щоб обслуговувати його за допомогою локального сервера розробки, щоб грати з вашою програмою.

Як і більшість сучасних зовнішніх інструментів, Angular CLI побудовано на основі Node.js. Node.js – це серверна технологія, яка дозволяє запускати JavaScript на сервері та створювати серверні веб-додатки. Однак Angular – це інтерфейсна технологія, тому навіть якщо вам потрібно встановити Node.js на вашій машині розробки, це лише для запуску CLI.

Після того, як буде створена програма, не знадобиться Node.js, оскільки остаточні пакети – це лише статичні HTML, CSS і JavaScript, які можуть обслуговуватися будь-яким сервером або CDN.

З огляду на це, якщо створюєтсья web-застосунок із повним стеком за допомогою Angular, може знадобитися Node.js для створення back-end частини, якщо є потреба використовувати JavaScript для зовнішньої та задньої частини.

Перевірте стек MEAN — це архітектура, яка включає MongoDB, Express (веб-сервер і фреймворк REST API, створений на основі Node.js) і Angular.

У цьому випадку Node.js використовується для створення back-end частини вашої програми та може бути замінений будь-якою серверною технологією, як-от PHP, Ruby або Python. Але Angular не залежить від Node.js, за винятком інструменту CLI та встановлення пакетів із npm.

Аби користуватися Angular CLI для початку його треба встановити. Для цього необхідно встановити Node та npm, бажано останніх версій, оскільки зі старими можуть виникати конфлікти. Далі необхідно запустити команду для встановлення Angular CLI: $ npm install @angular/cli [5].

CLI надає такі команди:

1) build (b): компілює додаток Angular у вихідний каталог під назвою dist/ за заданим вихідним шляхом. Має бути виконано з каталогу робочої області;
2) config: отримує або встановлює значення конфігурації Angular;
3) generate (g): генерує та/або змінює файли на основі схеми;
4) help: перелік доступних команд та їх короткий опис;
5) new (n): створює новий робочий простір і початкову програму Angular;
6) serve(s): будує та обслуговує вашу програму, перебудовуючи зміни у файлі;
7) test (t): запускає модульні тести в проекті.

Можна використовувати Angular CLI для швидкого створення проекту Angular, виконавши таку команду в інтерфейсі командного рядка:

$ ng new frontend

Як згадувалося раніше, CLI запитає, чи необхідно додати маршрутизацію (Angular routing). Можна відповісти, ввівши Y (Так) або N (Ні), що є параметром за замовчуванням. Він також запитає про формат таблиці стилів, який необхідно використовувати (наприклад, CSS). Необхідно обрати параметри та натиснути Enter, щоб продовжити. Після цього у вас буде створений проект зі структурою каталогів і купою файлів конфігурацій і коду (рис. 2.1). Це буде переважно у форматах TypeScript і JSON.

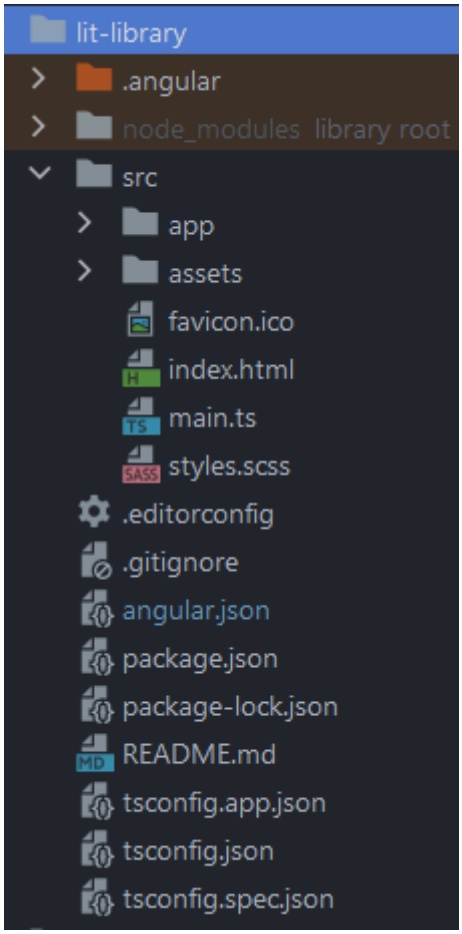

Рисунок 2.1 – Структура проєкту

Розглянемо роль кожного файлу:

1) /node_modules/: усі сторонні бібліотеки встановлюються в цю папку за допомогою npm install;
2) /src/: містить вихідний код програми, найбільше роботи буде зроблено тут;

3) /app/: містить модулі та компоненти;
4) /assets/: містить статичні ресурси, такі як зображення, значки та стилі;
5) /environments/: містить файли конфігурації середовища (виробництва та розробки);
6) browserslist: потрібен автопрефіксу для підтримки CSS;
7) favicon.ico: значок сайту;
8) index.html: основний файл HTML;
9) main.ts: основний початковий файл, з якого завантажується AppModule;
10) styles.css: файл глобальної таблиці стилів для проекту;
11) angular.json: містить конфігурації для CLI;
12) package.json: містить основну інформацію про проект (назва, опис і залежності);
13) README.md: файл розмітки, який містить опис проекту;
14) tsconfig.json: файл конфігурації для TypeScript.

Angular CLI надає повний набір інструментів для розробки інтерфейсних програм на локальній машині. Таким чином, не потрібно встановлювати локальний сервер для обслуговування проекту — можна просто використати команду ng serve із свого терміналу, щоб обслуговувати проект локально [7].

Тепер можна перейти за адресою http://localhost:4200/, щоб почати працювати з інтерфейсною програмою. Сторінка автоматично оновиться, якщо буде змінено будь-який вихідний файл.

Angular CLI надає команду ng generate, яка допомагає розробникам генерувати базові артефакти Angular, такі як модулі, компоненти, директиви, канали та служби:

Ng generate component <name>

Angular CLI автоматично додасть посилання на компоненти, директиви та канали у файлі src/app.module.ts.

Робота з Angular застосунком багато в чому зводиться до роботи з компонентами, тому давайте детально розберемо що це таке.

Компонент (Angular component) - особлива частина функціоналу зі своєю логікою, HTML-шаблоном і CSS-стилями.

Компонента сворюється у файлі <name>.component.ts, наприклад app.component.ts (див. рис. 2.2).

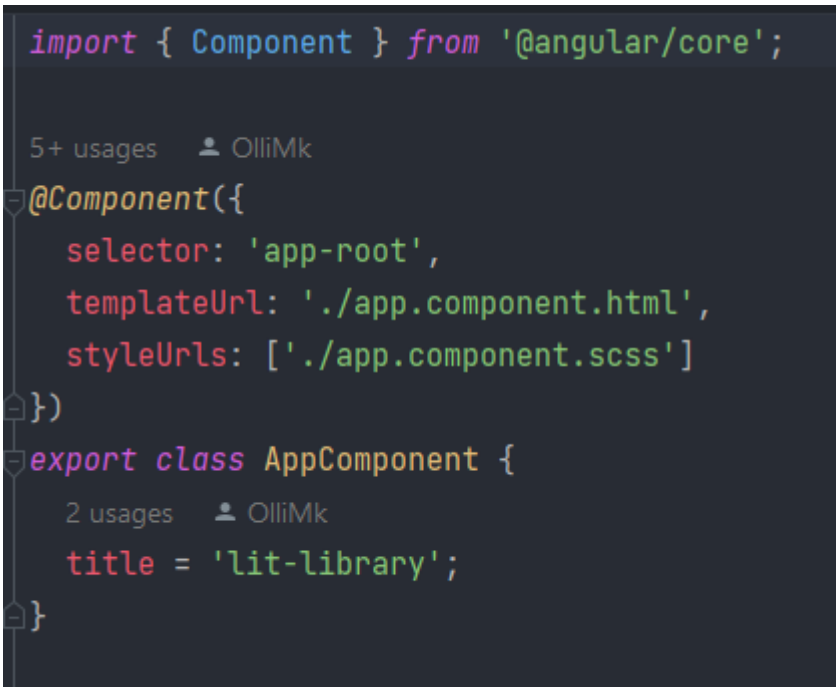

Рисунок 2.2 – Створення компоненти

Клас стає компонентом Angular, якщо його об'явленню передує декоратор @Component() з конфігурацією об'єкта. Для об'явлення @Component(), спочатку треба імпортувати цю директиву з бібліотеки ангуляр [6].

Щоб клас міг використовуватись в інших модулях, він визначається ключовим словом export. У самому ж класі визначено лише одну змінну, яка як значення зберігає певний рядок.

Для створення компонента необхідно імпортувати функцію декоратора @Component із бібліотеки @angular/core. Декоратор @Component дає змогу ідентифікувати клас як компонент. При створені компоненти за допомогою Angular CLI усі необхідні залежності будуть імпортовані автоматично.

Якби ми не застосували декоратор @Component до класу AppComponent, то клас AppComponent компонентом не вважався б. Декоратор @Component як параметр приймає об'єкт із конфігурацією, яка вказує фреймворку, як працювати з компонентом та його поданням.

Кожен компонент повинен мати один шаблон. Однак необов'язково визначати шаблон безпосередньо за допомогою властивості template. Можна винести шаблон у зовнішній файл із розміткою html, а для його підключення використовувати властивість templateUrl (див. рис. 2.3).

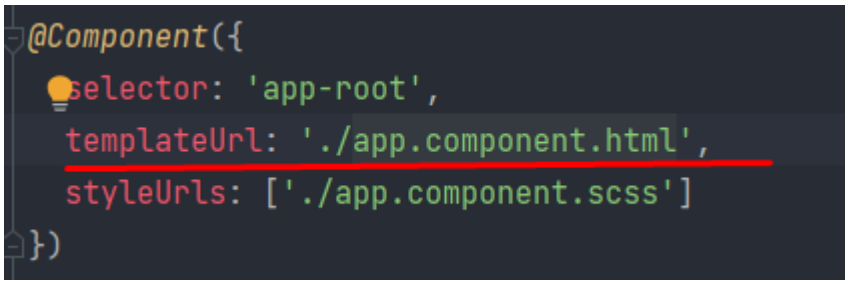

Рисунок 2.3 – Властивість компоненти templateUrl

Шаблон може бути однорядковим або багаторядковим. Якщо шаблон багаторядковий, то він полягає в косі лапки (`), які варто відрізняти від стандартних простих лапок (') [9].

Також у прикладі вище встановлюється властивість selector, що визначає селектор CSS. В елемент з цим селектором Angular додаватиме уявлення компонента. Наприклад, у прикладі вище, селектор має значення app-root. Відповідно, якщо html-сторінка містить елемент <app-root></app-root>, наприклад, як на рисунку 2.4.

Сервіси в Angular представляють досить широкий спектр класів, які виконують деякі специфічні завдання, наприклад логування, роботу з даними і тощо. На відміну від компонентів і директив, сервіси не працюють з уявленнями,

тобто з розміткою html, не надають на неї прямого впливу. Вони виконують строго певне і досить вузьке завдання [13].

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>LitLibrary</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root></app-root>
```

Рисунок 2.4 – Використання основної компоненти

Стандартні завдання сервісів є наступними.

1. Надання даних додатку. Сервіс може сам зберігати дані в пам'яті або для отримання даних може звертатися до якогось джерела даних, наприклад, до сервера.
2. Сервіс може представляти канал взаємодії між окремими компонентами програми
3. Сервіс може інкапсулювати бізнес-логіку, різноманітні обчислювальні завдання, завдання з логування, які краще виносити з компонентів. Таким чином, код компонентів буде зосереджений безпосередньо на роботі з поданням. Крім того, тим самим ми також можемо вирішити проблему повторення коду, якщо нам потрібно виконати те саме завдання в різних компонентах і класах [15].

2.3 Google Books API

API (Application Programming Interface) – це механізми, які дозволяють двом програмним компонентам взаємодіяти один з одним, використовуючи набір визначень та протоколів. Архітектура API зазвичай пояснюється з погляду клієнта та сервера. Програма, що надсилає запит, називається клієнтом, а програма, що надсилає відповідь, називається сервером. Отже, у прикладі з погодою база даних служби – це сервер, а мобільний додаток – це клієнт.

Існує чотири різні способи роботи API залежно від того, коли та чому вони були створені.

SOAP - Simple Object Access Protocol, тобто простий протокол доступу до об'єктів. Клієнт та сервер обмінюються повідомленнями за допомогою XML. Це менш гнучкий API, який був більш популярним у минулому.

RPC API такі API називаються системою віддаленого виклику процедур. Клієнт виконує функцію (або процедуру) на сервері, і сервер надсилає результат назад клієнту.

Websocket API – це ще одна сучасна розробка web API, яка використовує об'єкти JSON для передачі даних. WebSocket API підтримує двосторонній зв'язок між клієнтськими програмами та сервером. Сервер може надсилати повідомлення зворотного дзвінка підключеним клієнтам, що робить його ефективнішим, ніж REST API [26].

REST API. На сьогоднішній день це найпопулярніші та гнучкіші API-інтерфейси в Інтернеті. Клієнт надсилає запити на сервер у вигляді даних. Сервер використовує це введення клієнта для запуску внутрішніх функцій і повертає вихідні дані назад клієнтові. Давайте розглянемо API REST докладніше нижче.

REST – це Representational State Transfer, тобто передача репрезентативного стану. REST визначає набір функцій, таких як GET, PUT, DELETE тощо, які клієнти можуть використовуватиме доступу до даних сервера. Клієнти та сервери обмінюються даними за протоколом HTTP.

Головною особливістю REST API і те, що така передача виконується без збереження стану. Без збереження стану означає, що сервери не зберігають дані клієнта між запитами. Клієнтські запити до сервера подібні до URL-адрес, які ви вводите в браузері для відвідування веб-сайту. Відповідь від сервера є простими даними без типового графічного відображення веб-сторінки.

REST API має чотири основні переваги.

1. Інтеграція. API використовуються для інтеграції нових програм із існуючими програмними системами. Це підвищує швидкість розробки, тому що кожну функцію не потрібно писати з нуля. API можна використовувати для посилення наявного коду.

2. Інновації. Цілі галузі можуть змінитися з появою нової програми. Компанії повинні швидко реагувати та підтримувати швидке розгортання інноваційних послуг. Вони можуть це зробити, вносячи зміни на рівні API без необхідності переписувати весь код.

3. Розширення. API-інтерфейси надають компаніям унікальну можливість задовольняти потреби своїх клієнтів на різних платформах. Наприклад, карти API дозволяє інтегрувати інформацію про карти через веб-сайти, Android, iOS тощо. Будь-яка компанія може надати аналогічний доступ до своїх внутрішніх баз даних, використовуючи безкоштовні або платні API.

4. Простота обслуговування. API діє як шлюз між двома системами. Кожна система зобов'язана вносити внутрішні зміни, щоб це не вплинуло на API. Таким чином, будь-які майбутні зміни коду однією стороною не вплинуть на іншу сторону.

API класифікуються як за архітектурою, так і за сферою застосування. Ми вже розглянули основні типи архітектур API, тому пропонуємо розглянути сфери застосування.

Приватні API – це внутрішні організації API, які використовуються тільки для з'єднання систем і даних всередині бізнесу.

Загальнодоступні API – це API із загальним доступом і можуть бути використані будь-ким. З цими типами API може бути (але не обов'язково) пов'язана певна авторизація та вартість.

Партнерські API – це API, які доступні лише авторизованим стороннім розробникам для сприяння партнерським відносинам між підприємствами.

Складові API – це API, що об'єднують два або більше API для вирішення складних системних вимог або поведінки.

Google Books API. API-інтерфейси сімейства API Книг Google дозволяють використовувати функції Книг Google на сайті або в додатку. Новий Google Books API дозволяє програмно виконувати більшість операцій, які можна виконувати в інтерактивному режимі на веб-сайті Google Books [42].

Embedded Viewer API дозволяє вбудовувати контент безпосередньо на ваш сайт. Google надає такий набір зумовлених книжкових полиць для кожного користувача.

1. Список побажань: Мобільна книжкова полиця.
2. Куплено: заповнюється томами, які придбав користувач. Користувач не може вручну додавати або видаляти томи.
3. Читати: Книжкова полиця, що змінюється.
4. Читаю зараз: Книжкова полиця, що змінюється.
5. Прочитав: Змінна книжкова полиця.
6. Переглянуто: заповнено томами, переглянутими користувачем. Користувач не може вручну додавати або видаляти томи.
7. Нещодавно переглянуті: містить томи, які нещодавно відкриті користувачем у веб-рідері. Користувач не може вручну додавати томи.
8. Мої електронні книги: Книжкова полиця, що змінюється. Куплені книги додаються автоматично, але їх можна видалити вручну.
9. Книги для вас: заповнені персоналізованими рекомендаціями щодо обсягу. Якщо у нас немає рекомендацій для користувача, то цієї книжкової полиці не існує.

Є кілька способів викликати API. Безпосереднє використання REST. Використання REST з JavaScript (код на стороні сервера не потрібен). У системі RESTful ресурси зберігаються в сховищі даних; клієнт надсилає запит, щоб сервер виконав певну дію (наприклад, створення, отримання, оновлення або видалення ресурсу), а сервер виконує дію та надсилає відповідь, часто у формі представлення зазначеного ресурсу.

В RESTful API від Google клієнт визначає дію за допомогою дієслова, наприклад POST, GET, PUT або DELETE. Оскільки всі ресурси API мають унікальні URI, доступні через HTTP, REST дозволяє кешувати дані та оптимізований для роботи з розподіленою веб-інфраструктурою. Підтримувані операції Books зіставляються безпосередньо з дієсловами REST HTTP, як описано в операціях API Books.

Конкретний формат URI для Books API:

https://www.googleapis.com/books/v1/{collectionName}/resourceID?parameters.

Наприклад наступне посилання видасть нам список книг з колекції «бізнес» (див. рис. 2.5): https://www.googleapis.com/books/v1/volumes?q=business+subject, де resourceID — це ідентифікатор тома або ресурсу книжкової полиці, а параметри — будь-які параметри, які застосовуються до запиту.

Запити до Books API щодо непублічних даних користувача мають бути авторизовані автентифікованим користувачем.

Деталі процесу авторизації або «потоку» для OAuth 2.0 дещо відрізняються залежно від типу програми, яку ви пишете. Наступний загальний процес застосовується до всіх типів програм.

1. Коли ви створюєте свою програму, ви реєструєте її за допомогою Google API Console. Потім Google надає інформацію, яка знадобиться вам пізніше, наприклад ідентифікатор клієнта та секрет клієнта.
2. Активуйте Books API на Google API Console. (Якщо API немає в списку на консолі API, пропустіть цей крок.)

3. Коли вашій програмі потрібен доступ до даних користувача, вона запитує у Google певний обсяг доступу.
4. Google показує користувачеві екран згоди з проханням дозволити вашій програмі запитувати деякі їхні дані.
5. Якщо користувач схвалює, Google надає вашій програмі короткочасний маркер доступу.
6. Програма запитує дані користувача, долучаючи до запиту маркер доступу.
7. Якщо Google визначить, що запит і маркер дійсні, він повертає запитані дані.


```
{
  "kind": "books#volumes",
  "totalItems": 502,
  "items": [
    {
      "kind": "books#volume",
      "id": "HBYYAQAAMAAJ",
      "etag": "gF7Mx7M4yis",
      "selfLink": "https://www.googleapis.com/books/v1/volumes/HBYYAQAAMAAJ",
      "volumeInfo": {
        "title": "Baylor Business Studies",
        "publishedDate": "1981",
        "industryIdentifiers": [
          {
            "type": "OTHER",
            "identifier": "MINN:31951001223550T"
          }
        ],
        "readingModes": {
          "text": false,
          "image": false
        },
        "printType": "BOOK",
        "categories": [
          "Business"
        ],
```


Рисунок 2.5 – Об’єкт колекції книг

Деякі потоки включають додаткові кроки, наприклад використання маркерів оновлення для отримання нових маркерів доступу[42].

Висновки до розділу 2

Для розробки PWA було використано фреймворк Angular та мову TypeScript, мову розмітки HTML, CSS як засіб стилізації, Rx.js для написання

запитів і back-end частини. Для створення електронної бібліотеки використано Google Books API, який дозволяє вбудовувати сервіс повтотекстового пошуку по книгах, які оцифровані компанією Google й містять більше 10 млн книг, безпосередньо у web-застосунок бібліотеки.

3 ПРОЄКТУВАННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОГРЕСИВНОГО WEB-ЗАСТОСУНКУ НА ОСНОВІ МОДЕЛІ RAIL

3.1 Створення прогресивного веб застосунку в WebStorm

Сторення прогресивного веб-застосунку як і будь-якого іншого починається з вибору середовища розробки. Через описані в другому розділі причини було обрано середовище розробки web-storm від JetBrains. Для початку роботи з застосунком потрібно створити новий проект (рис. 3.1).

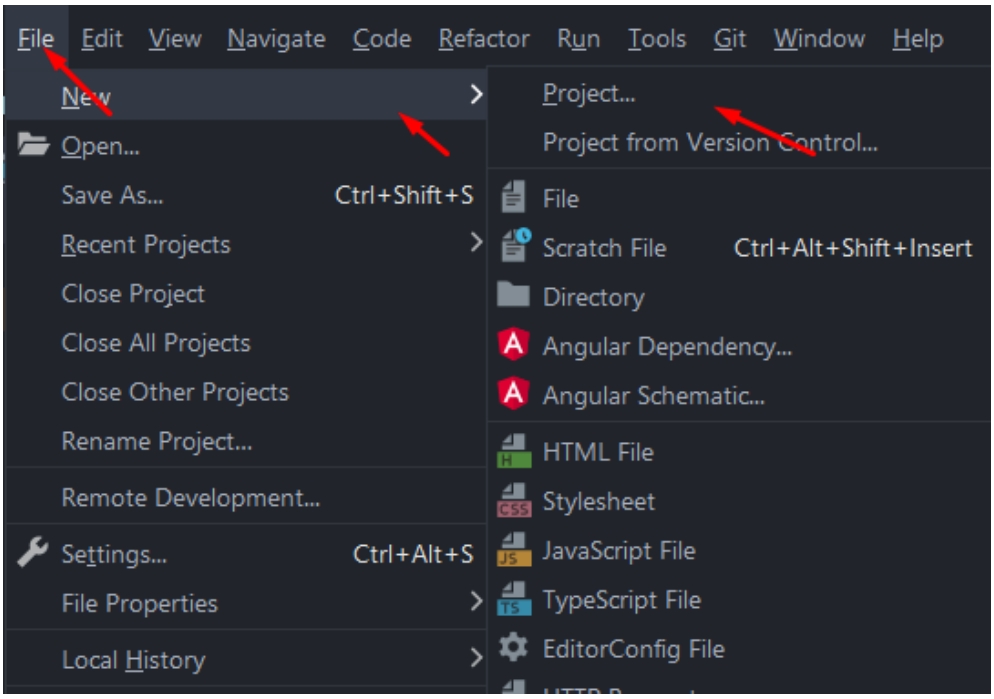

Рисунок 3.1 – Створення нового проекту у WebStorm

Після вибору створення нового проєкту буде доступний вибір передустановлених структур проєктів. Було обрано проєкт для Ангуляру, директорії для розміщення проекту, а також версії Node.js та AngularCLI для подальшої розробки (дис. рис. 3.2).

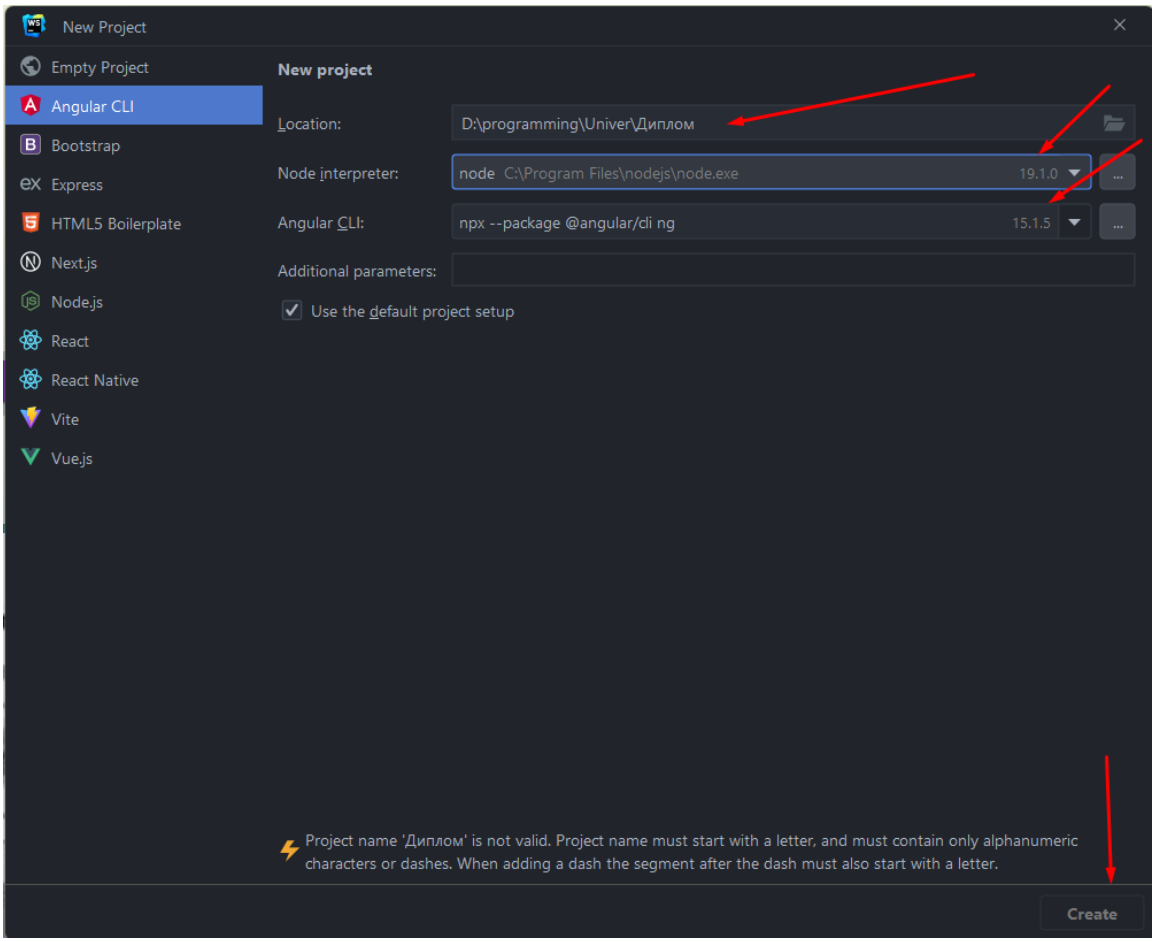

Рисунок 3.2 – Налаштування проєкту

Після створення проекту автоматично запуститься встановлення пакету npm та angular (рис. 3.3). Також будуть створені усі необхідні для початку розробки файли.

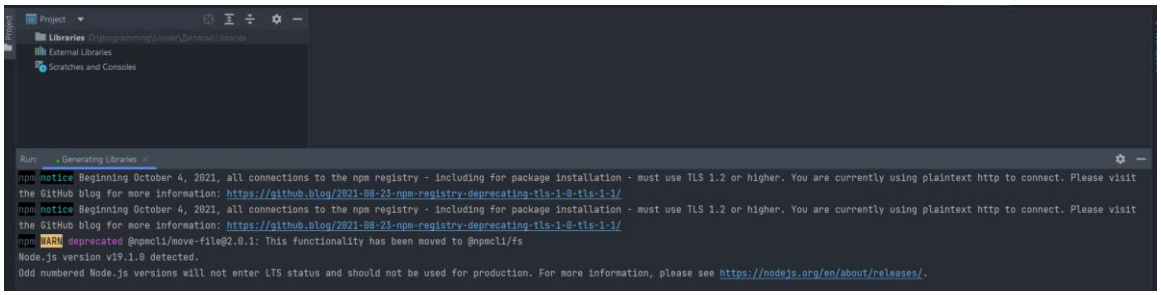

Рисунок 3.3 – Налаштування проекту: встановлення пакету npm та angular

Було створено компоненту home в папці pages, для відображення контенту на головні сторінці застосунку (див. рис. 3.4).

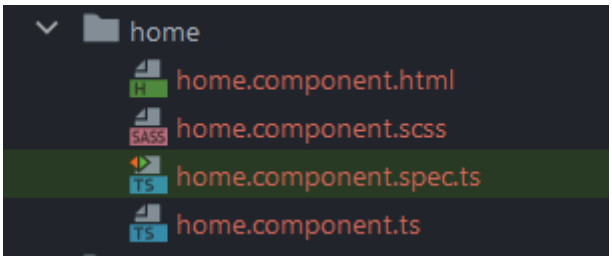

Рисунок 3.4 – Компонента Home

У середині компоненти було прописано верстку для головної сторінки, а також вставлено компоненту book-card яка відображає карточку з книгою, та компонента search-input, яка відповідає за пошук всередині сторінки. Також компонента home містить й інші елементи, такі як елементи пошуку та кнопки категорій, завдяки яким можна обирати книги по категоріям (див. рис. 3.5).

```
<app-search-input (searchValue)="getSearchValue($event)"></app-search-input>
<div class="toggle-book">
  <button mat-raised-button class="toggle-btn"  (click)="toggleCards( button: 'All')">All</button>
  <button mat-raised-button class="toggle-btn"  (click)="toggleCards( button: 'History')">History</button>
  <button mat-raised-button class="toggle-btn"  (click)="toggleCards( button: 'Business')">Business</button>
  <button mat-raised-button class="toggle-btn" (click)="toggleCards( button: 'Detective')">Detective</button>
  <button mat-raised-button class="toggle-btn" (click)="toggleCards( button: 'Psychology')">Psychology</button>
  <button mat-raised-button class="toggle-btn" (click)="toggleCards( button: 'Fantasy')">Fantasy</button>
</div>
<div class="books-grid-count">
  {{titleCount}} <span>({{gridDataCount}})</span>
</div>
</section>
<section class="books-grid">
  <book-card *ngFor="let book of gridDataBooks | async | search: searchValue"
            [book]="book"></book-card>
</section>
```

Рисунок 3.5 – Верстка компоненти Home

Елемент books-grid-count виводить загальну кількість книг які відображені на сторінці (рис. 3.6). А також показує в якому жанрі ці книги. Для виводу інформації в цьому елементі була використана інтерполяція [30].

Інтерполяція – це вбудовування виразів у розмічений текст. За замовчанням при інтерполяції як роздільники використовуються подвійні фігурні дужки {{ }}. Кнопки відповідають за зміни категорій відображених книг. Для цього було використано eventListener вбудований в ангуляр «(click)». Цей eventListener викликає метод toggleCards із певним параметром, характеризуючим обрану категорію.

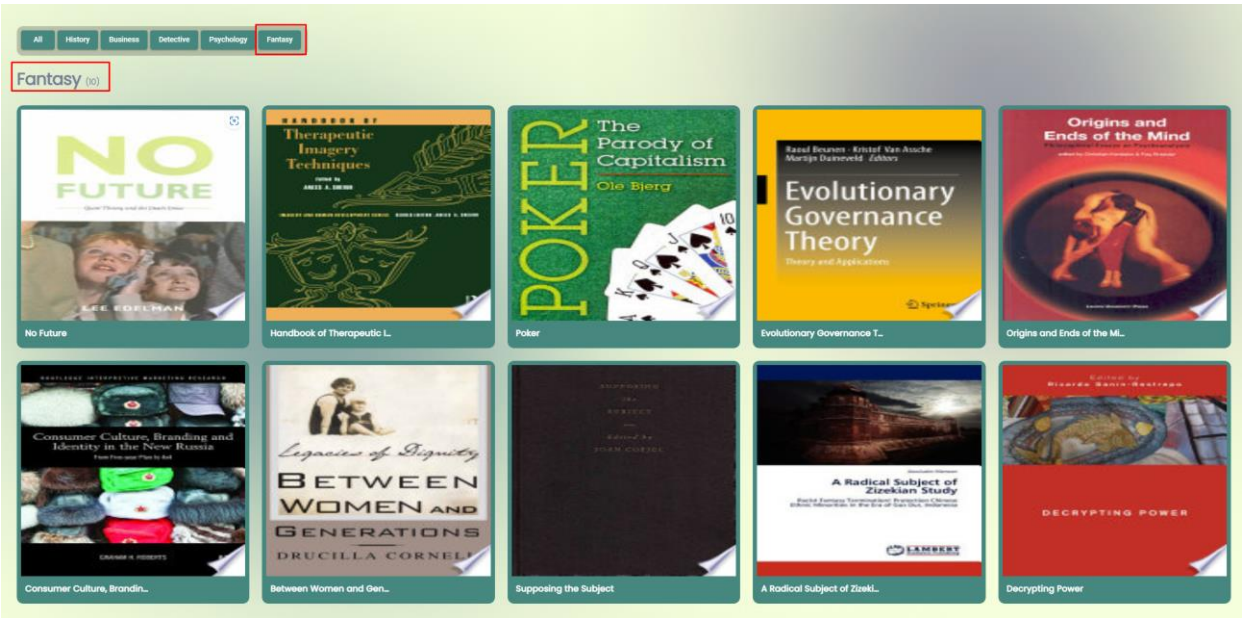

Рисунок 3.6 – Робота елементу books-grid-count

Для пошуку книг всередині сторінки було створено компоненту пошуку search-input (рис. 3.7).

```
<form class="example-form">
  <mat-form-field class="example-full-width"
    <mat-label>Search books</mat-label>
    <img width="20" height="21" matPrefix [s
    <input [type]="type ? type : 'text'" mat
  </mat-form-field>
</form>
```

Рисунок 3.7 – Компонента пошуку книг

Компонента починає шукати книгу одразу після вводу символу (рис. 3.8).

Рисунок 3.7 – Пошук книг

Також було створено сторінку для відображення контенту по кожній конкретній книзі. На сторінці сінгл продукт відображено хедер всього сайту, бредкрамбс для розуміння користувача де саме він зараз знаходиться, а також секція з інормацією про книгу (див. рис. 3.8).

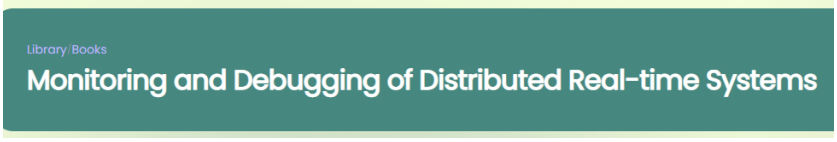

Рисунок 3.8 – Бредкрамбс

На сторінці відображено титульну обкладинку книги, її автора, дату публікації та кількість сторінок (рис. 3.9). Також є інформація про категорію

книги і її публікацію. Є посилання для переходу на сторінку Google Boks, для подальшого ознайомлення з екземпляром (додаток А).

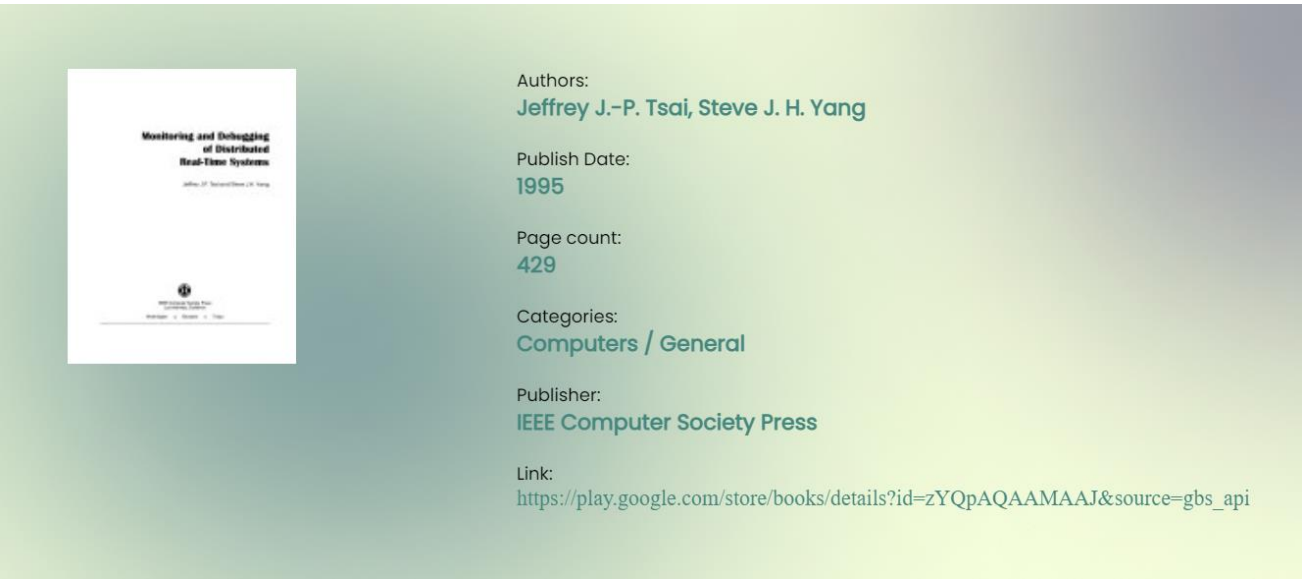

Рисунок 3.9 – Секція інформації про книгу

На прикладі коду для виведення сторінок ми можемо побачити що задля недопущення нукопичення пустих тегів в разі відсутності необхідно контенту на стороні серверу, за допомогою можливостей фреймворку Angular, в самій верстці була прописана умова для відображення блоку з контентом (рис. 3.10). У разі якщо book.volumeInfo.pageCount буде дорівнювати 0, тобто нестиме пусту строку, цей блок вивидено не буде [14].

```
<div class="details-container__block page-block"
*ngIf="book.volumeInfo.pageCount">
  <p class="label">Page count: </p>
  <p class="value">{{book.volumeInfo.pageCount}}</p>
</div>
```

Рисунок 3.10 – Код виведення даних по кількості сторінок

3.2 Налаштування роботи PWA

PWA – це веб-технологія, яка трансформує сайт на застосунок. При відкритті, програма запускається в обгортці браузера, через що PWA дозволяє використовувати програму на будь-якій платформі, яка використовує браузер, що відповідає стандартам [1].

Всередині PWA використовує Service worker, який взаємодіє з браузером, для забезпечення доступу до деяких вбудованих функцій. Service worker'a має доступ до Cache Storage для web ресурсів, і IndexDB для даних. Завдяки Service worker'у можлива реалізація кешування, що дозволяє PWA додатку працювати в режимі "офлайн".

AngularCLI має власний пакет для налаштування роботи прогресивного веб застосунку: @angular/pwa. Для його встановлленя необхідно в консолі в середині проекту ввести команду ng add @angular/pwa [38]. Після вдалого встановлення пакету додадутся нові файли та оновляться декілька вже існуючих (рис. 3.11) [18].

```
The package @angular/pwa@next will be installed and executed.
Would you like to proceed? Yes
√ Packages successfully installed.
CREATE ngsw-config.json (631 bytes)
CREATE src/assets/icons/icon-144x144.png (1394 bytes)
CREATE src/assets/icons/icon-152x152.png (1427 bytes)
CREATE src/assets/icons/icon-192x192.png (1790 bytes)
CREATE src/assets/icons/icon-384x384.png (3557 bytes)
CREATE src/assets/icons/icon-512x512.png (5008 bytes)
CREATE src/assets/icons/icon-72x72.png (792 bytes)
CREATE src/assets/icons/icon-96x96.png (958 bytes)
UPDATE angular.json (4132 bytes)
UPDATE src/index.html (1345 bytes)
```

Рисунок 3.11 – Нові файли для pwa

Короткий опис файлів, що додалися в проект.

1. *Ngsw-config.json* – конфіг, що відповідає за створення ngsw-worker.js (serviceworker.js). Містить базову структуру нашого Service worker's.

2. *Manifest.webmanifest* – файл маніфесту, що визначає як застосунок PWA буде виглядати при відкритті. Дозволяє конфігурувати такі параметри, як значки, різні бекграунди, назва тощо (додаток Б).

Відвідавши файл src/index.html, ми побачимо, що наш маніфест прописався в head частині, нашого html файлу (див. рис. 3.12). Файл manifest.webmanifest також піідключається в angular.json. При встановлені пакету також були завантажені стандартні іконки для логотипу прогресивного веб застосунку (рис. 3.13) [45].

```
<head>
  <meta charset="utf-8">
  <title>Books Shell</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
  <link rel="preconnect" href="https://fonts.gstatic.com">
  <link href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500&amp;display=swap" rel="stylesheet">
  <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
  <link rel="manifest" href="manifest.webmanifest">
  <meta name="theme-color" content="#1976d2">
</head>
```

Рисунок 3.12 – Підключення файлу manifest.webmanifest

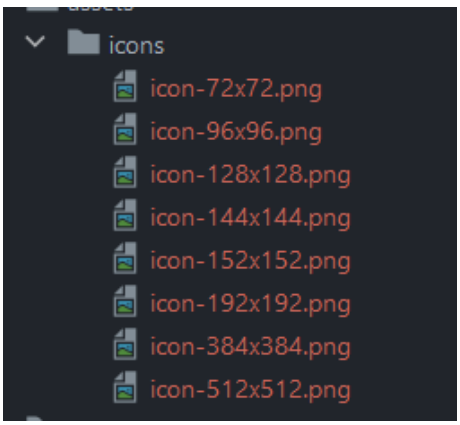

Рисунок 3.13 – Іконки для логотипу застосунку

У app.module.ts додався модуль Service worker для реєстрації файлу ngsw-worker.js (даний файл генерується при білді програми) (рис. 3.14).

Рисунок 3.14 – Підключення Service Workers

Тепер можемо збилдити наш застосунок і запустити його. Але оскільки PWA працює тільки з https і localhost і ng serve не працює з Service worker'ами, нам необхідний окремий HTTP-сервер, щоб локально перевірити наш проект. Потрібно створити складання та розмістити її окремо, використовуючи http-сервер. Для цього встановимо пакет npm http-server командою:

npm i -g http-server

Після цього перебуваючи в кореневій папці нашої програми, створимо продакшен білд командою:

ng build

Після закінчення білда в папці dist з'явиться наш зібраний білд проекту (рис. 3.15). Також у ньому буде згенерований файл нашого Service worker'a ngsw-worker.js.

Тепер, після успішного генерування проекту знаходячись в папці dist/app, в нашому випадку dist/book-shell-app необхідно запустити сервер командою:

http-server -p 8080

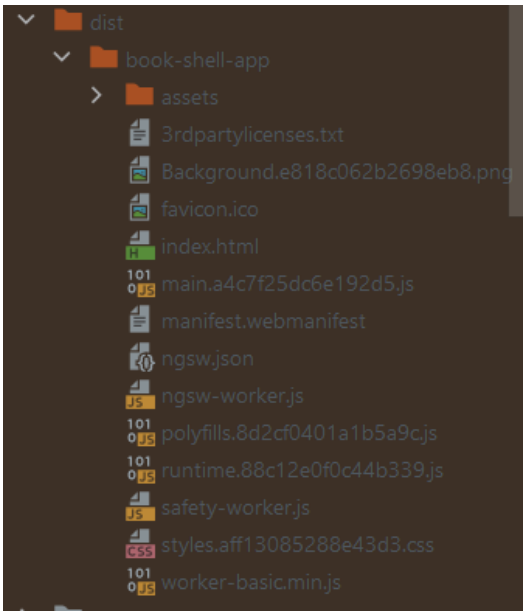

Рисунок 3.15 – Папка build проекту

Після відкриття сторінки сайту можна встановити прогресивний web-застосунок на свій пристрій не залежно від операційної системи та запускати його на виконання із використанням іконки на робочому столі (рис. 3.16, рис. 3.17) [16].

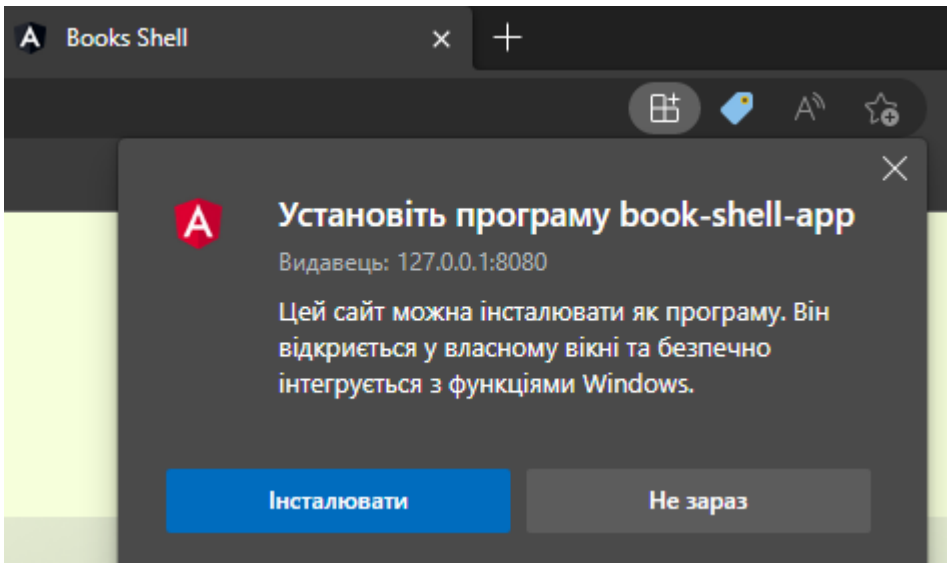

Рисунок 3.16 – Встановлення застосунку комп’ютер

Рисунок 3.17 – Іконка застосунку на робочому столі

Таким чином було створено та налаштовано повноцінний прогресивний веб-застосунок бібліотеки.

3.3 Оптимізація прогресивного web-застосунку бібліотеки

Швидкість веб-застосунку створює перше враження про ваш бізнес. Важливо розуміти, що ви не отримаєте другого шансу, коли справа доходить до взаємодії з користувачем. Низька швидкість веб-сайту є однією з найбільш неприємних речей, яка відштовхне людей від вашого ресурсу.

Високоефективні веб-застосунки забезпечують високий рівень повторних відвідувань, низькі показники відмов, вищі конверсії, залучення, вищі рейтинги в звичайному пошуку та кращий досвід користувача. Повільні веб-застосунки коштуватимуть зайвих витрат і погіршать репутацію.

1. *Налаштування часу завантаження Load.* Перш ніж почати оптимізацію швидкості веб-застосунку, необхідно визначити поточний час завантаження та визначити, що сповільнює застосунок. Слід встановити цілі ефективності веб-застосунку. Згідно моделі RAIL, Load тобто час завантаження сторінки має бути не довше 2-х секунд.

Використовуючи засіб оцінки продуктивності Google PageSpeed Insights ми визначили, що час завантаження сторінки займає 3,1 секунди (рис. 3.18, рис. 3.19). При цьому LCP (largest contentful paint) – швидкість завантаження основного контенту, повідомляє час рендеринга самого великого зображення або текстового

блоку, видимого в області перегляду, перевіреного з моменту початку завантаження сторінки [24].

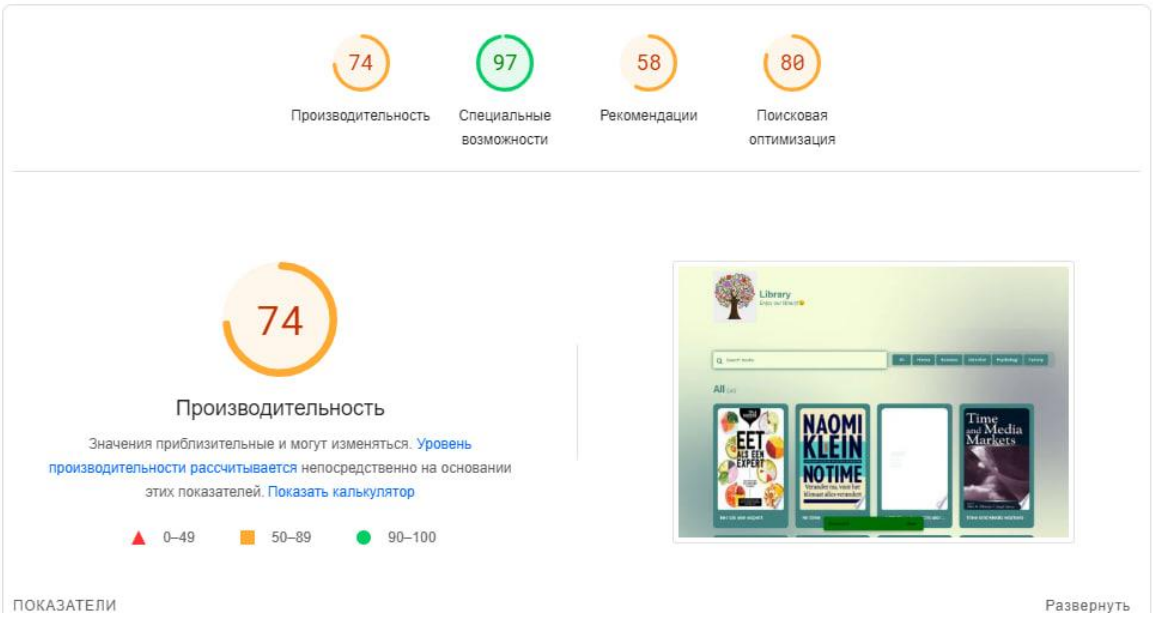

Рисунок 3.18 – Оцінка продуктивності веб-застосунку

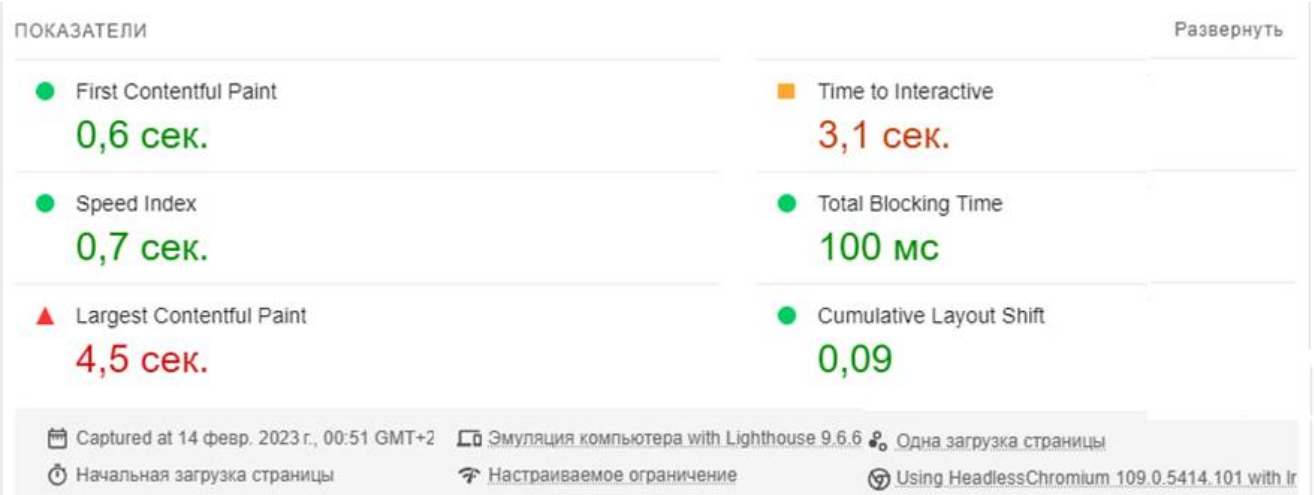

Рисунок 3.19 – Показники продуктивності веб-застосунку

Google PageSpeed Insights також надає поради по оптимізації швидкості сторінки веб-застосунку. Основний час завантаження сторінки прийшовся на гарфічні зображення, в особливості – фотографію фону (див. рис. 3.20).

Аби пришвидшити завантаження сторінки необхідно зменшити розмір зображень. Для цього було використано безкоштовний сервіс tinypng.

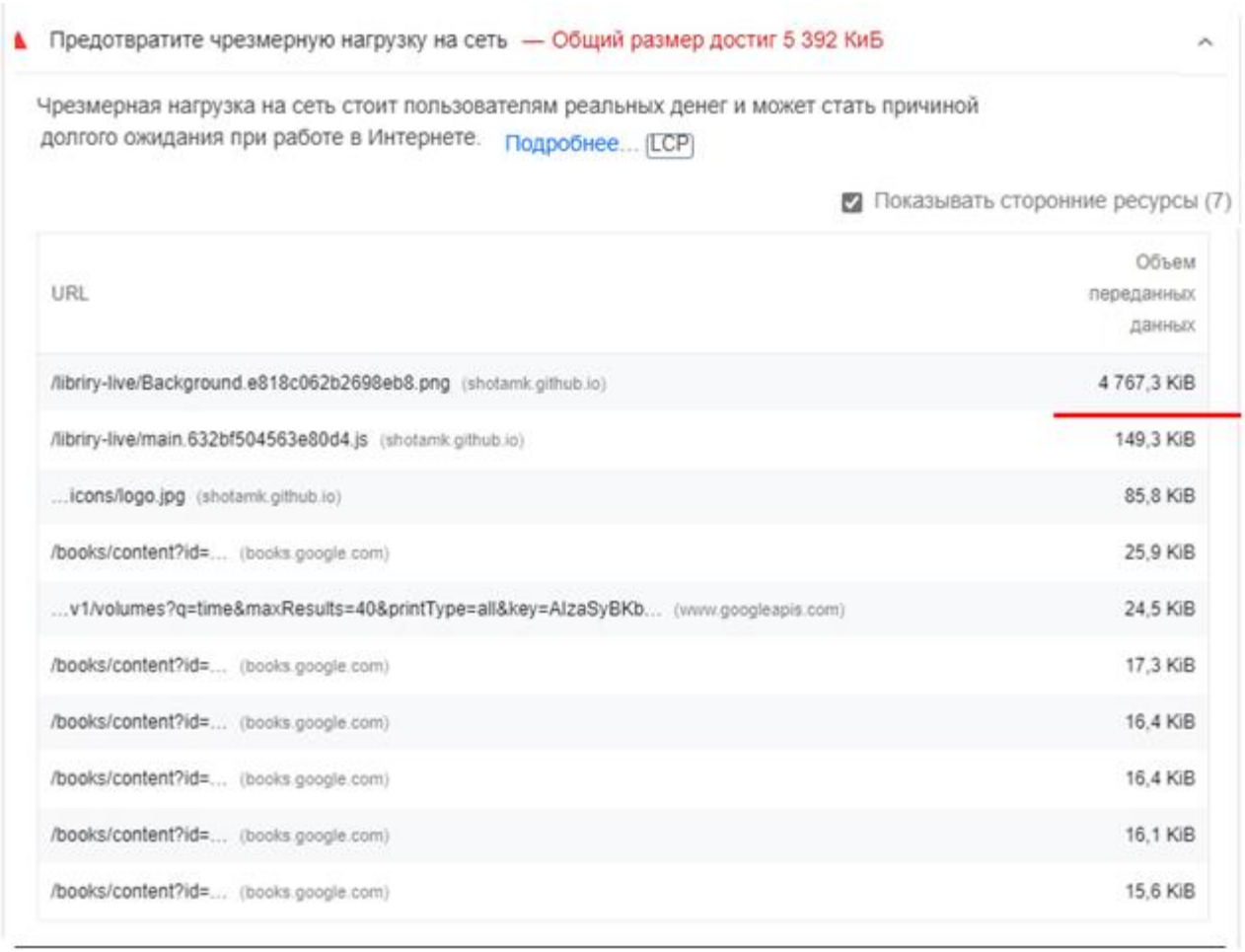

Рисунок 3.20 – Погане завантаження графічних зображень

Після оптимізації зображенн їх розмір зменшився. Розмір фонового зображення яке займало найбільше місця вдалося зменшити майже на 80%, тим самим значно прискоривши завантаження сторінки. Також було дотримано й інших порад, таких як завдання розмірів зображень явним способом через атрибути. Таким чином нові результати завантаження сторінки цілком задовольняють критерії моделі RAIL (рис. 3.21).

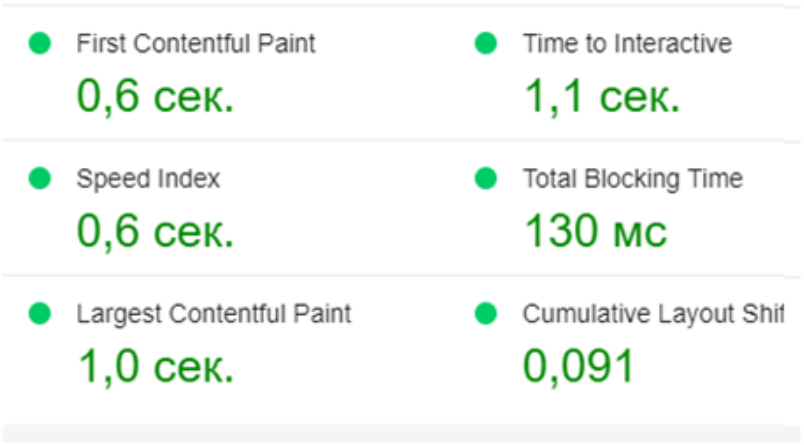

Рисунок 3.21 – Нові показання завантаження сторінки.

2. *Оптимізація відповіді Response*. Взаємодія з користувачем відображає реальний досвід користувача після однієї взаємодії з інтерфейсом застосунку. Взаємодія починається з події, створеної реальним користувачем через інтерфейс користувача в браузері. Типовими типами таких подій є клацання мишею, дотики та події клавіатури.

Релевантними вважаються лише взаємодії, що призводять до будь-яких запитів на стороні сервера. Усі інші взаємодії ігноруються та ніколи не надсилаються на сервер для звітування. У результаті цього прокручування статичних сторінок або клацання порожніх областей ніколи не реєструються як дії користувача.

Кожна зафіксована взаємодія пов’язана з будь-якими HTTP-запитами, які виникають через цю взаємодію. Згідно з критеріями моделі RAIL час відгуку сайту не має перевищувати 100 мілісекунд.

Для оцінки швидкості відгуку сайту було використано сервіс chrome DevTools. Ми можемо записати нашу роботу в електронній бібліотеці, після чого подивитися данні які надав DevTools, в зокрема Interections, тобто взаємодію користувача і вебзастосунку, або по іншому response.

Для цього необхідно відкрити засоби розробника, клацнувши по екрану правою кнопкою миші і обрати Inspect (рис.3.22). Після чого перейти до вкладки Perfomance (рис.3.23), і обирати там новий запис (рис.3.24)

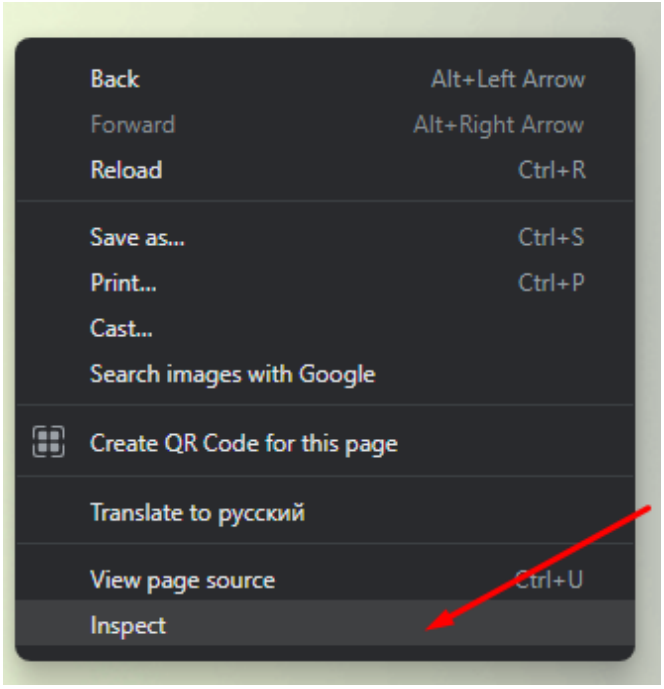

Рисунок 3.22 – Відкриття засобів розробника

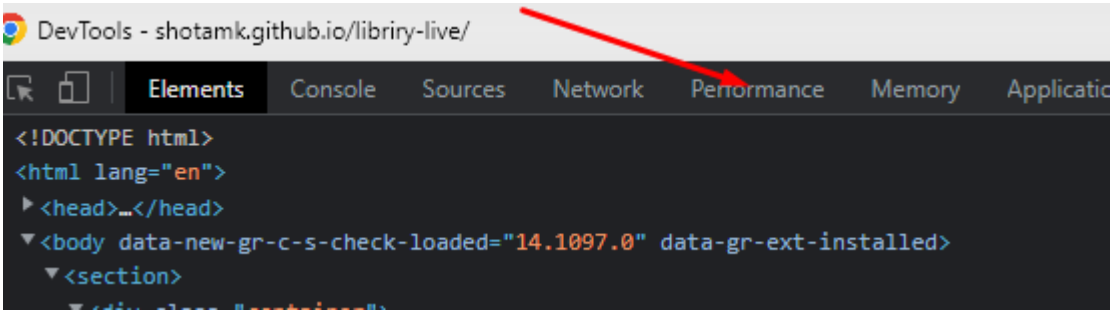

Рисунок 3.23 – Вкладка Perfomance

З приведених далі рисунків видно що швидкість відгуку на вебзастосунку електронної бібліотеки становить від 72 до 128 мілісекунд (рис. 3.25).

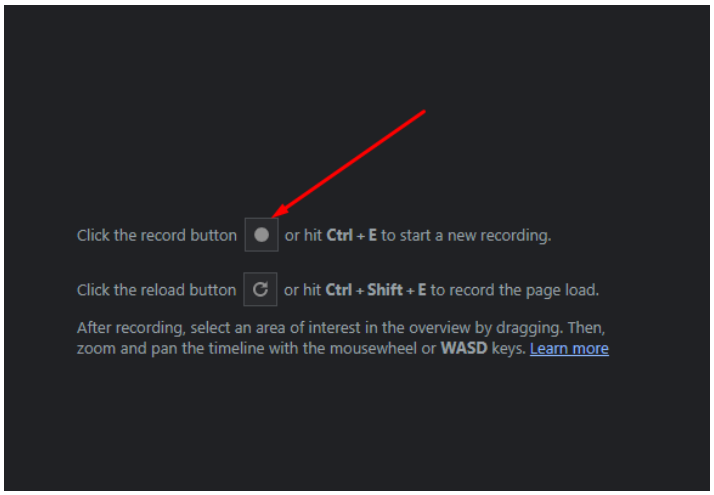

Рисунок 3.24 – Запуск запис подій

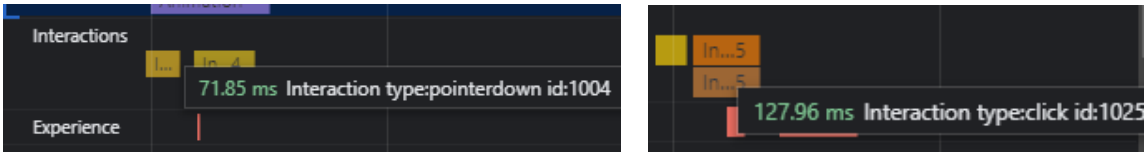

Рисунок 3.25 – Взаємодія користувача з сайтом

Час відгуку в 128 мілісекунд нас не влаштовує, для прискорення цього процесу є два найбільш дієвих метода, перший це зменшення джаваскріпт коду на сторінці, тобто видалення усіх зайвих функцій, а також розділення довгих задач шляхом впровадження асинхронного коду.

Обмеження кількості JavaScript-коду на сторінці скорочує час, який браузер повинен витрачати на виконання цього коду. Це прискорює реакцію браузера на будь-які дії користувача. Щоб зменшити кількість JavaScript, що виконується на сторінці відкладіть завантаження JavaScript, що не використовується [23].

За замовчуванням, весь JavaScript блокує рендеринг. Коли браузер зустрічає тег скрипта, який посилається на зовнішній файл JavaScript, він повинен зупинити свої дії та завантажити, проаналізувати, скомпілювати та виконати цей JavaScript. Тому слід завантажувати лише той код, який потрібний для сторінки або відповіді

на введення користувача. Після того, як код було зменшено, можна спробувати розбити JS код, що довго виконується, на дрібніші асинхронні завдання.

Тривалі завдання - це періоди виконання JavaScript, коли користувачі можуть помітити, що інтерфейс користувача не відповідає (рис. 3.26). Будь-який фрагмент коду, який блокує основний потік на 50 мс або більше, можна розглядати як тривале завдання. Тривалі завдання ознаки потенційного роздування JavaScript-коду (завантаження та виконання більшої кількості завдань, ніж може знадобитися користувачеві прямо зараз). Поділ тривалих завдань може зменшити затримку на сайті [4, 40].

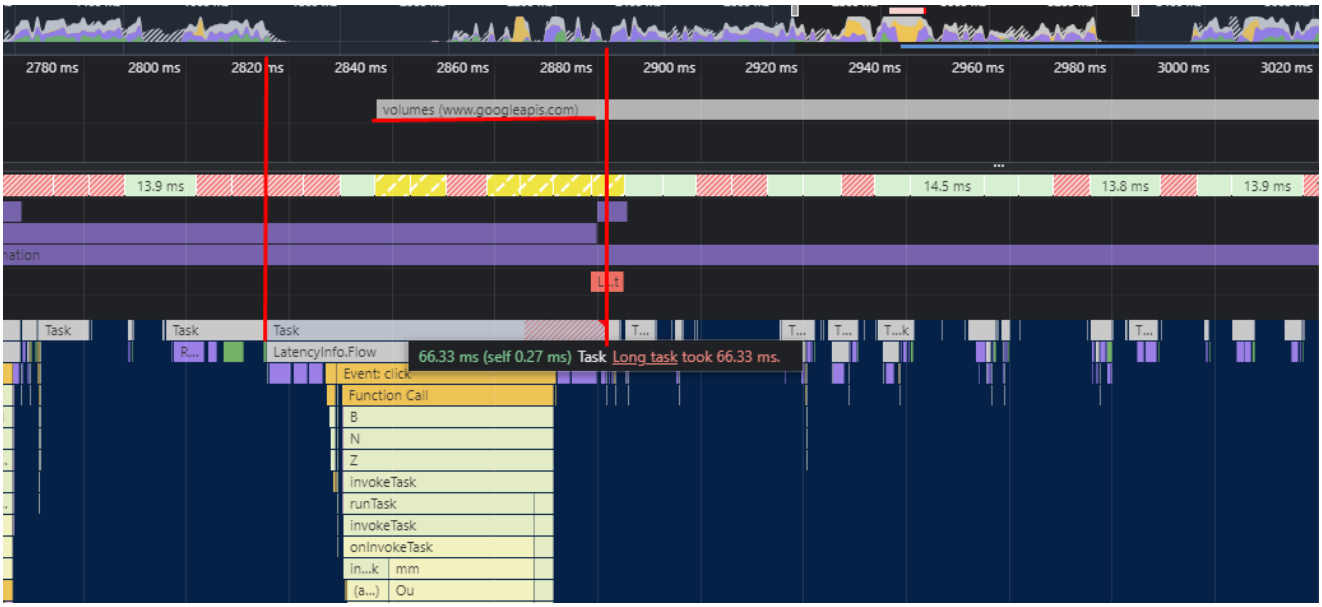

Рисунок 3.26 – Тривале завдання

В продовж роботи прогресивного застосунку було одне js завдання яке тривало довше 50 мс. Якщо проаналізувати графу main де ми бачимо час виконання джаваскріпт блоків, та графу нетворк, можна помітити що через 20 мс після початку виконнаня завдання JS кодом почали завантажуватися данні з Google Books API.

Ця частина коду як й інші запити на сайті виконуються асинхронно, тому доволі складно зменшити час виконання ще більше. Але оскільки ліміти привищенні не критично, що дає нам можливість вкластися в необхідний проміжок часу 100 мс, (можемо спостерігати, що та ж анімація продовжує працювати стабільно, і її час не перевищує 8 мс), було прийнято рішення залишити цей блок коду нетронутим [27].

3. *Оптимізація анімації Animation*. Анімація є у всіх видах сучасних додатків. Під анімацією мається на увазі, звичайно, не інтерактивні елементи зроблені розробником, а такі операції як скролінг, що висувається збоку меню та інші подібні ефекти, пов'язані з тим, що вміст екрана повинен постійно змінюватися протягом якогось часу.

Анімація перетягування (drag): коли користувач використовує якісь функції програми або сайту, які мають на увазі, що він натискає на якусь область на екрані і потім «тягне» убік у натиснутому стані.

Це може бути масштабування екрану, може бути перетягування об'єктів і так далі. Щоб анімація виглядала безперервною, кожен кадр анімації повинен з'являтися на екрані менш ніж за 16 мілісекунд, тобто зі швидкістю 60 FPS (1 секунда / 60 кадрів = 16,6 мілісекунд на кадр).

Для перевірки часу анімації програмного застосунку скористаємося можливосями chrome DevTools. У вкладці перфоманс знайдемо графу під назвою frames (рис. 3.27, рис. 3.28). У цій графі ми можемо бачити скільки часу був активним той чи інший кадр.

Наприклад на малюнку 3.27 перший фрейм займає доволі велику частину, ми також можемо бачити що він був активний більше 500 мс, приблизно стільки часу в нас зайняло відкрити інспектор і запустити запис екрану для аналізу продуктивності.

При перевірки роботи анімації не було виявлено жодних проблем, «картинка» працювала плавно навіть на моніторі з частотою оновлення 120 Гц.

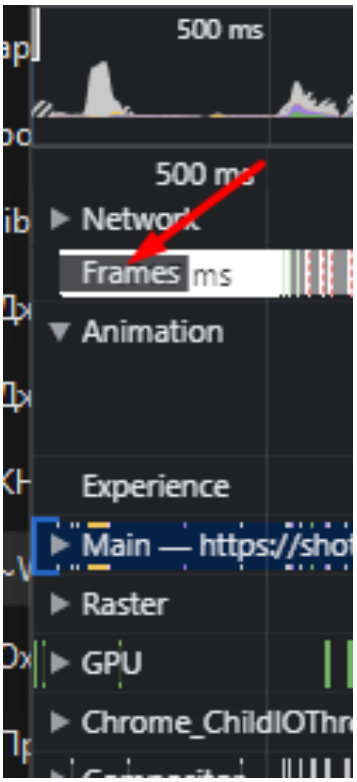

Рисунок 3.27 – Графа Frames

Рисунок 3.28 – Час виконання анімації

4. *Оптимізація очікування Idle*. Всередині кожного додатку відбувається безліч процесів, але далеко не всі вони повинні працювати в такі критичні моменти, коли програма відпрацьовує взаємодію з користувачами типу «відгук» або «анімація» (рис. 3.29). Ініціалізація різних компонентів, пошук і сортування даних, відправлення даних до сервісу аналітики: все це можна робити в той момент, коли програма або браузер знаходяться в режимі очікування.

Щоб використовувати час очікування правильно, потрібно групувати завдання, які виконуються в цей час, у блоки не більше 50 мілісекунд. Чому так? Тому що якщо користувач почне взаємодію, необхідно встигнути зробити відгук за 100 мілісекунд, а не змусити його чекати дві секунди, поки програма щось робить і не може відгукнутися на його дії. Як бачимо на рисунку 3.29 під час простоювання сторінки, за необхідністю може бути виконано відкладений код.

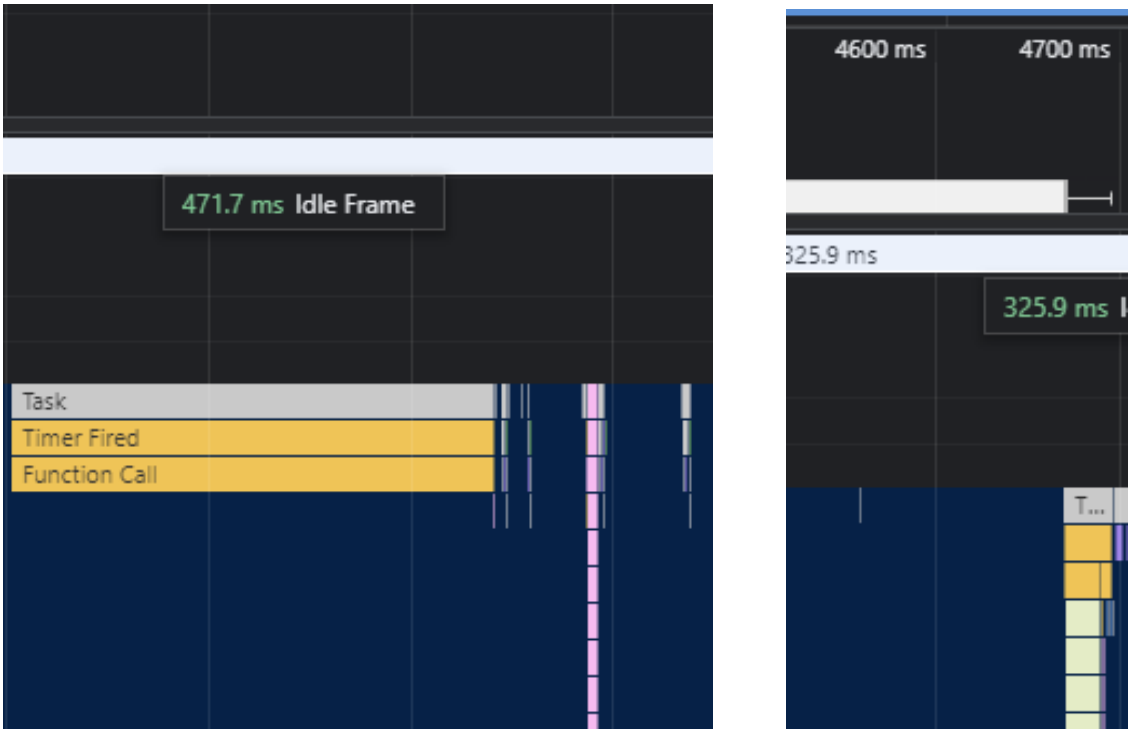

Рисунок 3.29 – Відкладена робота

Висновки до розділу 3

Здійснено розробку, програмну реалізацію та тестування прогресивного web-застосунку бібліотеки. Розроблений PWA має розширену функціональність у порівнянні зі звичайними web-застосунками з доступом до пристроїв користувача. Дозволяє працювати в автономному режимі, забезпечує легкість установки,

простоту налаштувань інтерфейсу, більшу ефективність, продуктивність та надійність навіть у локальній мережі.

Для оцінювання продуктивності вебзастосунку бібліотеки з метою його оптимізації було використано сервіси Google PageSpeed Insights, chrome DevTools. Аби пришвидшити завантаження сторінки було зменшено розмір зображень із використанням сервісу tinypng і здійснено завдання розмірів зображень явним способом через атрибути. Для прискорення відгуку PWA бібліотеки на релевантні події було зменшено JavaScript коду на сторінці (видалення усіх зайвих функцій), а також розділення довгих задач шляхом впровадження асинхронного коду.

Продуктивніть web-застосунку бібліотеки було оптимізовано у процесі його розробки відповідно до моделі RAIL: відповідь на виконання дій користувача надходить у час, не більший ніж 100 мілісекунд, анімація має затримку не більшу, ніж 60 кадрів у секунду, завантаження становить 1,1 сек, очікування у роботі застосунку використовується для виконання відкладеної роботи.

ВИСНОВКИ

У результаті проведеного дослідження виявлено, що сьогодні цільова споживацька аудиторія все більше використовує Інтернет для отримання інформації та задаволення потреб у товарах та послугах. Це обумовило різке зростання вимог до продуктивності web-ресурсів та їх кросплатформенності та виникненню нового напряму – розробки прогресивних web-застоунків PWA.

Установлено, що до базових компонент PWA віднесено: 1) маніфест застосунку – для надання нативних функцій, таких як іконка застосунку на робочому столі тощо; 2) технологію Service Workers – для фонових завдань і роботи в offline-режимі; 3) архітектуру Application Shell: для швидкого завантаження з Service Workers.

Дослідження сучасного стану мережевих та мобільних бібліотечних сервісів дозволило виявити, що їх використання зростає швидкими темпами. Однак підходи до їх оптимізації не завжди є задовільними, оскільки інколи завантаження сторінки на екрані необхідно чекати 15-20 с, що негативно впливає на попит користування бібіліотечними послугами у електронному форматі. Результати проведеного дослдіження показали, що тільки невеликий процент електронних бібліотек має прийнятну швидкість роботи. Що потребує більш прогресивних підходів до розробки ресурсів електронних бібліотек.

У результаті проведеного дослідження установлено, що найбільш оптимальною моделлю оптимізації продуктивності прогресивних застосунків є орієнтована на користувача модель RAIL, яка оцінює архітектурні рішення з точки зору їх впливу на швидкість роботи застосунку: 1) відповідь (Response) на виконання дій користувача повинна надходити у час, не більший ніж 100 мілісекунд; 2) анімація Animation) повинна мати затримку не більшу, ніж 60 кадрів у секунду; 3) очікування (Idle) – простій між діями повинен бути використаний для виконання відкладеної роботи; 4) завантаження (Load) повинно становити не більше 1-2 секунд.

Для розробки PWA було використано фреймворк Angular та мову TypeScript, мову розмітки HTML, CSS як засіб стилізації, Rx.js для написання

запитів і back-end частини. Для створення електронної бібліотеки використано Google Books API, який дозволяє вбудовувати сервіс повтотекстового пошуку по книгах, які оцифровані компанією Google й містять більше 10 млн книг, безпосередньо у web-застосунок бібліотеки.

Здійснено розробку, програмну реалізацію та тестування прогресивного web-застосунку бібліотеки. Розроблений PWA має розширену функціональність у порівнянні зі звичайними web-застосунками з доступом до пристроїв користувача. Дозволяє працювати в автономному режимі, забезпечує легкість установки, простоту налаштувань інтерфейсу, більшу ефективність, продуктивність та надійність навіть у локальній мережі.

Для оцінювання продуктивності вебзастосунку бібліотеки з метою його оптимізації було використано сервіси Google PageSpeed Insights, chrome DevTools. Аби пришвидшити завантаження сторінки було зменшено розмір зображень із використанням сервісу tinypng і здійснено завдання розмірів зображень явним способом через атрибути. Для прискорення відгуку PWA бібліотеки на релевантні події було зменшено JavaScript коду на сторінці (видалення усіх зайвих функцій), а також розділення довгих задач шляхом впровадження асинхронного коду.

Продуктивніть web-застосунку бібліотеки було оптимізовано у процесі його розробки відповідно до моделі RAIL: відповідь на виконання дій користувача надходить у час, не більший ніж 100 мілісекунд, анімація має затримку не більшу, ніж 60 кадрів у секунду, завантаження становить 1,1 сек, очікування у роботі застосунку використовується для виконання відкладеної роботи.

Поставлені завдання виконано повністю, однак функціонал розробленого застосунку може бути розширений у подальшому шляхом надання можливості ствоерння власного аккаунту в застосунку і власних підбірок книг.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Sheppard D. Beginning Progressive Web App Development: Creating a Native App Experience on the Web. CA: Pub. Apress Bercel, 2017. – 266 p. doi: 10.1007/978-1-4842-3090-9
2. Bowlin G. Building Progressive Web Apps, Createspace Independent Pub. 2017. 178 c.
3. Dormann A. Ionic 6: Create awesome apps for iOS, Android, Desktop and Web, D&D Verlag Bonn. 2022. 540 c.
4. Love C. Progressive Web Application Development by Example: Develop fast, reliable, and engaging user experiences for the web, Packt Publishing. 2018. 354 c.
5. Munro J. 50 Recipes for Programming Angular: Volume 1, Independently published. 2017. 455 c.
6. Tarasiewicz P., Böhm R. angularJS, Brainy Software. 2014. 348 c.
7. Bates J. AngularJS Mastery: A Code Like A Pro Guide For AngularJS Beginners, CreateSpace Independent Publishing Platform. 2016. 56 c.
8. Hajian M. Progressive Web Apps with Angular: Create Responsive, Fast and Reliable PWAs Using Angular, Springer. 2019. 397 c.
9. Shyam Seshadri Angular: Up and Running: Learning Angular, Step by Step; O'Reilly Media, 2018. -300 с.
10. Rungta K. Learn AngularJS in 1 Day: Complete Angular JS Guide with Examples, Independently published. 2018. 245 c.
11. ng-book: The Complete Guide to Angular 4 / Nathan Murray, Ari Lerner, Felipe Coury, Carlos Taborda; CreateSpace Independent Publishing Platform,. 2017. - 622 с.
12. Hitwise. UK Mobile Search: Topics and Themes; CONSUMER INSIGHTS REPORT, 2016. -14 с.
13. Freeman A. Pro Angular, Apress. 2017. 820 c.

14. Despoudis T. TypeScript 4 Design Patterns and Best Practices: Discover effective techniques and design patterns for every programming task, Packt Publishing. 2021. 350 с.
15. The TypeScript Workshop : A practical guide to confident, effective TypeScript programming, Gryngaus B. та інш. Packt Publishing. 2021. 714 с.
16. Tal A. Building Progressive Web Apps: Bringing the Power of Native to the Browser 1st Edition. O'Reilly Media, 2017. 288 с.
17. Blokdyk G. Progressive web app A Clear and Concise Reference Paperback, 5STARCooks. 2022.308 с.
18. Dean A.H. Progressive Web Apps. Addy Osmani, 2017. 200 с.
19. Flanagan D. JavaScript: The Definitive Guide: Master the World's Most-Used Programming Language. O'Reilly Media, 2020. 706 с.
20. Marla S. A Journey to Angular Development: Learn Angular Fundamentals, TypeScript, Webpack, Routing, Directives, Components, Forms, and Modules with Practical Examples (English Edition), BPB Publications. 2021 674 с.
21. Freeman A. Essential TypeScript 4: From Beginner to Pro, Apress. 2021. 581 с.
22. Goswami B. Progressive Web Apps for Social Development, Independently published. 2019. 96 с.
23. Frisbie M. AngularJS Web Application Development Cookbook, Packt Publishing. 2014. 346 с.
24. Kyle Simpson. You Don't Know JS: Async & Performance; O'Reilly Media. 2015. 296 с.
25. Blokdyk G. Progressive Web Apps Standard Requirements, 5STARCooks. 2022.310 с.
26. Amir S., Brenda J., Saurabh S. Designing Web APIs: Building APIs That Developers Love 1st Edition. O'Reilly Media, 2018. 232 с.
27. Learning TypeScript. Enhance Your Web Development Skills Using Type-Safe JavaScript. O'Reilly Media, 2022. 318 с.
28. А.Ю. Гаевский 100% самоучитель. Создание Web-страниц и Web-сайтов.

HTML и JavaScript / А.Ю. Гаевский, В.А. Романовский. - М.: Триумф, 2015. 464 с.

29. В.А. Зеньковский 47 готовых решений для создания Web-сайта (+ DVD-ROM) / А.Г. Богданов и др. - Москва: СПб. [и др.] : Питер, 2016. 272 с.
30. Файн Я., Моисеев А. Angular и TypeScript. Сайтостроение для профессионалов; Питер Прессб 2018. 464 с.
31. Фрейен Бен HTML5 и CSS3.Разработка сайтов для любых браузеров и устройств; Питер - Москва, 2014. 304 с.
32. Дакетт Джон HTML и CSS. Разработка и дизайн веб-сайтов (+ CD-ROM); Эксмо - М., 2013. 480 с.
33. Дино Эспозито, Разработка современных веб-приложений: анализ предметных областей и технологий; Уільямс, 2017. 464 с.
34. Хантер Т., Инглиш Б. Многопоточный JavaScript; O'Reilly 2022. 188 с.
35. Мартин Фаулер. Рефакторинг кода на JavaScript. Улучшение проекта существующего кода; Діалектика, 2020. -464 с.
36. Every PWA Statistics for Better eCommerce Insights in 2022 : Вебсайт. URL: https://www.simicart.com/blog/pwa-statistics/#1
37. Оптимізація сайту: як збільшити швидкість завантаження сайту? : Вебсайт. URL: https://webstudio2u.net/ua/optimization/734-kak-uvelichit-skorost-zagruzki-saita.html
38. Angular PWA, install and configure : Вебсайт. URL: https://medium.com/ngconf/angular-pwa-install-and-configure-858dd8e9fb07
39. Introducing RAIL: A User-Centric Model For Performance : Вебсайт. URL: https://www.smashingmagazine.com/2015/10/rail-user-centric-model-performance/
40. Analyze runtime performance : Вебсайт. URL: https://developer.chrome.com/docs/devtools/performance/
41. DEFINITION OF PROGRESSIVE WEB APP : Вебсайт. URL: https://techradar.softwareag.com/technology/progressive-web-apps/

42. Using the API : Вебсайт. URL: https://developers.google.com/books/docs/v1/using
43. A Pinterest Progressive Web App Performance Case Study : Вебсайт. URL: https://medium.com/dev-channel/a-pinterest-progressive-web-app-performance-case-study-3bd6ed2e6154
44. Why Progressive Web Apps Will Replace Native Mobile Apps : Вебсайт. URL: https://www.forbes.com/sites/forbestechcouncil/2018/03/09/why-progressive-web-apps-will-replace-native-mobile-apps/?sh=276466ba2112
45. Web app manifests : Вебсайт. URL: https://developer.mozilla.org/en-US/docs/Web/Manifest
46. Progressive Web Apps: Escaping Tabs Without Losing Our Soul : Вебсайт. URL: https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/

ДОДАТОК А

Лістинг коду для організації роботи із книгами

Сторінка Книги

```
constructor(private route: ActivatedRoute, private apiService: ApiService) { }
destroy$: Subject<boolean> = new Subject<boolean>();
book: BooksModel | undefined;
authors!: string;
categories!: string;
id: string = '';
ngOnInit(): void {
  this.route.params.pipe(
    takeUntil(this.destroy$),
    tap(params => this.id = params['id']),
    switchMap(() => this.apiService.getBookByTitle(this.id)),
    tap(data => {
      this.book = data;
      if(data.volumeInfo.authors) {
        data.volumeInfo.authors.forEach(item => {
          this.authors = this.authors ? this.authors + ', ' + item : item;
        })
      }
      if(data.volumeInfo.categories) {
        data.volumeInfo.categories.forEach(item => {
          this.categories = this.categories ? this.categories + ', ' + item : item;
        })
      }

    })
  ).subscribe()
}

ngOnDestroy() {
  this.destroy$.next(true);
  this.destroy$.unsubscribe();
}
```

Карта Книги

```
destroy$: Subject<boolean> = new Subject<boolean>();
@Input() book!: BooksModel;

getBookDetails(id: string) {
  if(id) {
    this.router.navigate(['/books', id]);
  }
}
constructor(private router: Router) { }
```

```
ngOnInit(): void {
}
ngOnDestroy() {
  this.destroy$.next(true);
  this.destroy$.unsubscribe();
}
```

Запити

```
private baseUrlBooks =
'https://www.googleapis.com/books/v1/volumes?q=time&maxResults=40&printType=all&key=AIzaSyBKbqRhuwEeQ2RxsBUyD7UxOPeTJZEgZPU'
constructor(private http: HttpClient) { }
getAllBooks(): Observable<BooksModel[]> {
  return this.http.get<BooksModelApi>(`${this.baseUrlBooks}`).pipe(
    pluck('items')
  );
}
getHistory(): Observable<BooksModel[]> {
  return
this.http.get<BooksModelApi>(`https://www.googleapis.com/books/v1/volumes?q=history+subject`).
pipe(
    pluck('items')
  );
}
getBusiness(): Observable<BooksModel[]> {
  return
this.http.get<BooksModelApi>(`https://www.googleapis.com/books/v1/volumes?q=business+subject`)
.pipe(
    pluck('items')
  );
}

getDetectives(): Observable<BooksModel[]> {
  return
this.http.get<BooksModelApi>(`https://www.googleapis.com/books/v1/volumes?q=detective+subject`
).pipe(
    pluck('items')
  );
}
getPsychology(): Observable<BooksModel[]> {
  return
this.http.get<BooksModelApi>(`https://www.googleapis.com/books/v1/volumes?q=psychology+subje
ct`).pipe(
    pluck('items')
  );
}
getFantasy(): Observable<BooksModel[]> {
  return
this.http.get<BooksModelApi>(`https://www.googleapis.com/books/v1/volumes?q=fantasy+subject`).
pipe(
```

```
    pluck('items')
  );
}
getBookByTitle(id: string): Observable<BooksModel> {
  return this.http.get<BooksModel>(`https://www.googleapis.com/books/v1/volumes/${id}`);
}
```

ДОДАТОК Б

Лістинг коду маніфесту

Manifest

```
{
  "name": "book-shell-app",
  "short_name": "book-shell-app",
  "theme_color": "#1976d2",
  "background_color": "#fafafa",
  "display": "standalone",
  "scope": "./",
  "start_url": "./",
  "icons": [
    {
      "src": "assets/icons/logo-72x72.png",
      "sizes": "72x72",
      "type": "image/png",
      "purpose": "maskable any"
    },
    {
      "src": "assets/icons/logo-96x96.png",
      "sizes": "96x96",
      "type": "image/png",
      "purpose": "maskable any"
    },
    {
      "src": "assets/icons/logo-128x128.png",
      "sizes": "128x128",
      "type": "image/png",
      "purpose": "maskable any"
    },
    {
      "src": "assets/icons/logo-144x144.png",
      "sizes": "144x144",
      "type": "image/png",
      "purpose": "maskable any"
    },
    {
      "src": "assets/icons/logo-152x152.png",
      "sizes": "152x152",
      "type": "image/png",
      "purpose": "maskable any"
    },
    {
      "src": "assets/icons/logo-192x192.png",
      "sizes": "192x192",
      "type": "image/png",
      "purpose": "maskable any"
```

```
    },
    {
      "src": "assets/icons/logo-384x384.png",
      "sizes": "384x384",
      "type": "image/png",
      "purpose": "maskable any"
    },
    {
      "src": "assets/icons/logo-512x512.png",
      "sizes": "512x512",
      "type": "image/png",
      "purpose": "maskable any"
    }
  ]
}
```