МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ

Кафедра інформаційних технологій проектування

Пояснювальна записка

до кваліфікаційної роботи

бакалавра

(освітньо-кваліфікаційний рівень)

на тему: «Розробка PWA-застосунку для ведення власного бюджету»

Виконав: студент __2__ курсу, групи WebC-1811
Спеціальності 126 «Інформаційні системи та технології»
ОП «Web-технології, Web-дизайн»
Лохно О.В.
(прізвище та ініціали)

Керівник Рудницький С.В.
(прізвище та ініціали)

Рецензент Землянський О.М.
(прізвище та ініціали)

Черкаси 2020 року

ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ

(назва вузу)

Факультет ФІТІС Кафедра ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ПРОЕКТУВАННЯ

Освітній рівень бакалавр

Спеціальність 126 «Інформаційні системи та технології»

Освітня програма «Web-технології, Web-дизайн»

ЗАТВЕРДЖУЮ:

зав. Кафедри _____Прокопенко Т.О.

"_____" _____ 2020 р.

ЗАВДАННЯ

На кваліфікаційну роботу студенту

Лохно Олексію Вікторовичу

(прізвище, ім'я, по-батькові)

1. Тема проекту (роботи) Розробка PWA-додатку для ведення власного бюджету

Керівник проекту(роботи) РУДНИЦЬКИЙ С.В., К.Т.Н., СТ.В..

Затверджена наказом Черкаського державного технологічного університету № 71/01 від 19.02.2020

2. Термін здачі студентом закінченого роботи _____

3. Вихідні дані до проекту (роботи) Середовище розробки - IDE WebStorm;Мова програмування – TypeScript, JavaScript; Фреймворки: Angular; Бібліотеки: Angular Material, ngx-charts; WEB-додаток.

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)

ВСТУП; РОЗДІЛ 1. АНАЛІТИЧНА ЧАСТИНА

1.1 Цілі, принципи, переваги застосування PWA; 1.2 Базові поняття, принципи та технології створення Web-додатків; 1.3 Аналіз ринку рішень для ведення власного; бюджету; 1.4 Висновок;

РОЗДІЛ 2. ПРОЕКТУВАННЯ PWA

2.1 Постановка задачв; 2.2 Можливості додатку; 2.3 Архітектура 2.4 Стек технологій; 2.4 Висновок

РОЗДІЛ 3. РЕАЛІЗАЦІЯ PWA

3.1 Реалізація БД; 3.2 Реалізація UI; 3.3 Реалізація PWA-підтримки

ВИСНОВКИ. ДОДАТКИ. СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____

1) Таблиця порявняння аналогів; 2) Архітектура додатку 3) Реалізована БД. 4) Головна сторінка додатку

6. Консультанти з проекту (роботи) із зазначенням розділів проекту, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання *20 лютого 2020 р.*

Керівник *РУДНИЦЬКИЙ С.В*
(підпис)

Завдання прийняв до виконання _____
(підпис)

Календарний план

Пор. №	Назва етапів дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	*ПІДГОТОВЧА СТАДІЯ*		
1.1	*ПОСТАНОВКА ЗАДАЧІ*	*ЛЮТИЙ 2020*	*ВИКОНАНО*
1.2	*ПІДГОТОВКА ЗАВДАННЯ*	*ЛЮТИЙ 2020*	*ВИКОНАНО*
1.3	*ПОГОДЖЕННЯ ЗАВДАННЯ*	*ЛЮТИЙ 2020*	*ВИКОНАНО*
1.4	*ЗАТВЕРДЖЕННЯ ЗАВДАННЯ*	*ЛЮТИЙ 2020*	*ВИКОНАНО*
2	*ОСНОВНА СТАДІЯ*		
2.1	*ПІДБІР МАТЕРІАЛІВ*	*БЕРЕЗЕНЬ 2020*	*ВИКОНАНО*
2.2	*АНАЛІЗ ШЛЯХІВ РІШЕННЯ ПОСТАВЛЕНОЇ ЗАДАЧІ*	*БЕРЕЗЕНЬ 2020*	*ВИКОНАНО*
2.3	*РОЗРАХУНОК ОСНОВНИХ ПАРАМЕТРІВ РОБОТИ*	*БЕРЕЗЕНЬ 2020*	*ВИКОНАНО*
2.4	*ВИБІР КІНЦЕВОГО ВАРІАНТУ ПРОЕКТНОГО РІШЕННЯ*	*БЕРЕЗЕНЬ 2020*	*ВИКОНАНО*
2.5	*ОФОРМЛЕННЯ ПЕРВІСНОЇ РЕДАКЦІЇ РОБОТИ*	*КВІТЕНЬ-ТРАВЕНЬ 2020*	*ВИКОНАНО*
3	*ЗАКЛЮЧНА СТАДІЯ*		
3.1	*УЗГОДЖЕННЯ ПРИЙНЯТИХ ПРОЕКТНИХ РІШЕНЬ З КЕРІВНИКОМ*	*ТРАВЕНЬ 2020*	*ВИКОНАНО*
3.2	*ОФОРМЛЕННЯ ПОЯСНЮВАЛЬНОЇ ЗАПИСКИ РОБОТИ В КІНЦЕВІЙ РЕДАКЦІЇ*	*ДО 20 ТРАВНЯ 2020*	*ВИКОНАНО*
3.3	*ПОПЕРЕДНІЙ ЗАХИСТ РОБОТИ*	*20 ТРАВНЯ 2020*	*ВИКОНАНО*
3.4	*ЗАТВЕРДЖЕННЯ РОБОТИ*	*ДО 17 ЧЕРВНЯ 2020*	*ВИКОНАНО*
3.5	*РЕЦЕНЗУВАННЯ РОБОТИ*	*ДО 17 ЧЕРВНЯ 2020*	*ВИКОНАНО*
3.6	*ЗАХИСТ РОБОТИ*	*18 ЧЕРВНЯ 2020*	

Студент-дипломник _____
(підпис) (ПІБ)

Керівник проекту Рудницький С.В.
(підпис) (ПІБ)

АНОТАЦІЯ

В даній кваліфікаційній роботі розглянуто дослідження в області новітніх технологій розробки PWA. Мета роботи – розробка PWA із застосуванням новітніх технологій для створення Web-додатків.

Предмет дослідження – принципи, основні етапи створення, переваги застосування PWA.

В першому розділі проведений аналіз цілей, принципів та переваг застосування PWA перед звичайними додатками. Також проведений аналіз базових понять принципів та технологій створення сучасних Web-додатків. Проведений аналіз існуючих технологій та ринок рішень ведення власного бюджету. Аналіз допоміг сформувати вимоги та виділити ключові потреби для користувачів, що пояснює актуальність розробки PWA для ведення власного бюджету.

В другому розділі поставлена задача створення PWA для ведення власного бюджету, описані його головні особливості та можливості, представлена архітектура майбутнього додатку, стек технологій та інструментів, що використовувалися під час процесу розробки.

В третьому розділі відображено особливості реалізації клієнтського додатку, БД, та підтримки PWA.

Кваліфікаційна робота складається з: 57 сторінок, 1 таблиці, і 20 рисунків. Включає в себе такі частини: аналітична частина, проектування, реалізація, висновки та список використаних джерел.

АННОТАЦИЯ

В данной квалификационной работе рассмотрены исследования в области новейших технологий разработки PWA. Цель работы – разработка PWA с применением новейших технологий для создания Web-приложений.

Предмет исследования – принципы, основные этапы создания, преимущества применения PWA.

В первой главе проведен анализ целей, принципов и преимуществ применения PWA перед обычными приложениями. Также проведен анализ базовых понятий принципов и технологий создания современных Web-приложений. Проведен анализ существующих технологий и рынок решений ведения собственного бюджета. Анализ помог сформировать требования и выделить ключевые потребности для пользователей, что объясняет актуальность разработки PWA для ведения собственного бюджета.

Во втором разделе поставлена задача создания PWA для ведения собственного бюджета, описаны его основные особенности и возможности, представлена архитектура будущего приложения, стек технологий и инструментов, которые использовались во время процесса разработки.

В третьем разделе отражены особенности реализации клиентского приложения, БД, и поддержки PWA.

Квалификационная работа состоит из: 57 страниц, 1 таблицы, и 20 рисунков. Включает в себя следующие части: аналитическая часть, проектирование, реализация, выводы и список использованных источников.

ANNOTATION

This qualification work considers research in the field of the newest PWA development technologies. The purpose of the work is to develop PWA using the latest technologies to create Web-applications.

The subject of research - principles, main stages of creation, advantages of PWA.

The first section analyzes the goals, principles, and benefits of using PWA over conventional applications. The analysis of basic concepts of principles and technologies of creation of modern Web-applications was also carried out. An analysis of existing technologies and the market for their own budget solutions was conducted. The analysis helped to form requirements and identify key needs for users, which explains the relevance of developing PWA for budget management.

The second section set the task of creating a PWA for budget management, describes its main features and capabilities, presented the architecture of the future application, the stack of technologies and tools used during the development process.

The third section reflects the features of the client application, database, and PWA support.

Qualification work consists of 57 pages, 1 table, and 20 figures. Includes the following parts: analytical part, design, implementation, conclusions, and a list of sources used

ЗМІСТ

СПИСОК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

PWA – Progressive Web App;

БД – База даних;

UI – User Interface;

ПЗ – Програмне забезпечення;

ОС – Операційна система;

SPA – Single Page Application;

HTML – HyperText Markup Language;

URL – Uniform Resource Locator;

JS – JavaScript;

CSS – Cascading Style Sheets;

API – Application Programming Interface;

DOM – Document Object Model ;

JSX – JavaScript XML;

XML – eXtensible Markup Language;

CLI – Command Line Utility;

DI – Dependency Injection;

SEO – Search Engine Optimization;

UX – User eXperience;

CSV – Comma-Separated Values;

QIF – Quicken Interchange Format;

ООП – Об’єктно-орієнтоване програмування;

NOSQL – not only SQL;

SQL – Structured Query Language.

ВСТУП

Розробники вже давно звикли до того, що якщо планується створювати додаток, то необхідно розробляти його під кожну підтримувану платформу. Якщо потрібно підтримувати Windows, macOS, Linux, Android та IOS, то для створення та підтримки додатку необхідно залучати розробників для конкретних платформ. Виходить що вартість створення додатку буде збільшуватись пропорційно кількості підтримуваних платформ.

Але що якщо можна було б зробити один додаток, який би працював на всіх платформах, без необхідності реалізації під кожну з них? Тут і приходить на допомогу PWA.

PWA має змогу працювати на будь-якій платформі, яка підтримує сучасні браузери так як додаток побудований та доставлений через Web. PWA будується на одній кодовій базі для Web і тому більше не потрібно розробляти нативні додатки для кожної платформи, що значно економить ресурси розробки. Більш того, ці додатки можуть працювати швидше та надійніше ніж нативні додатки, розроблені окремо для платформи. І навіть це ще не все, PWA надає можливість поширювати свої додатки без необхідності реєстрації в Google Play, App Store і Microsoft Store та платити за розміщення. Цим підтверджується *актуальність теми кваліфікаційної роботи* «Розробка PWA-застосунку для ведення власного бюджету».

Метою кваліфікаційної роботи є розробка PWA із застосуванням новітніх технологій для створення Web-додатків.

Для реалізації поставленої мети слід вирішити наступні *завдання*:

- Розглянути цілі, принципи, переваги застосування PWA.
- Проаналізувати відомості про базові поняття, принципи та технології, які використовуються в процесі створення Web-додатків.
- Провести аналіз аналогів.
- Сформувати вимоги до додатку.
- Дослідити основні етапи створення та проаналізувати технології, які використовуються при розробці PWA.
- Дослідити та проаналізувати технології сучасної розробки Web-додатків.
- Спроектувати структуру PWA.
- Реалізувати БД для додатку.
- Реалізувати UI для додатку.
- Реалізувати підтримку PWA.

Об’єктом роботи є процес розробки окремих додатків для різних платформ.

Предметом роботи є принципи, основні етапи створення, переваги застосування PWA.

1 АНАЛІТИЧНА ЧАСТИНА

1.1 Цілі, принципи, переваги застосування PWA

Прогресивні Web-додатки надають можливість установки на ПК і мобільних пристроях у вигляді нативних додатків, які створюються і доставляються безпосередньо через Інтернет. Це швидкі та надійні Web-додатки. І найголовніше, це Web-додатки, які працюють в будь-якому браузері.

PWA створений з використанням певних технологій для досягнення заданих цілей (рис. 1.1).

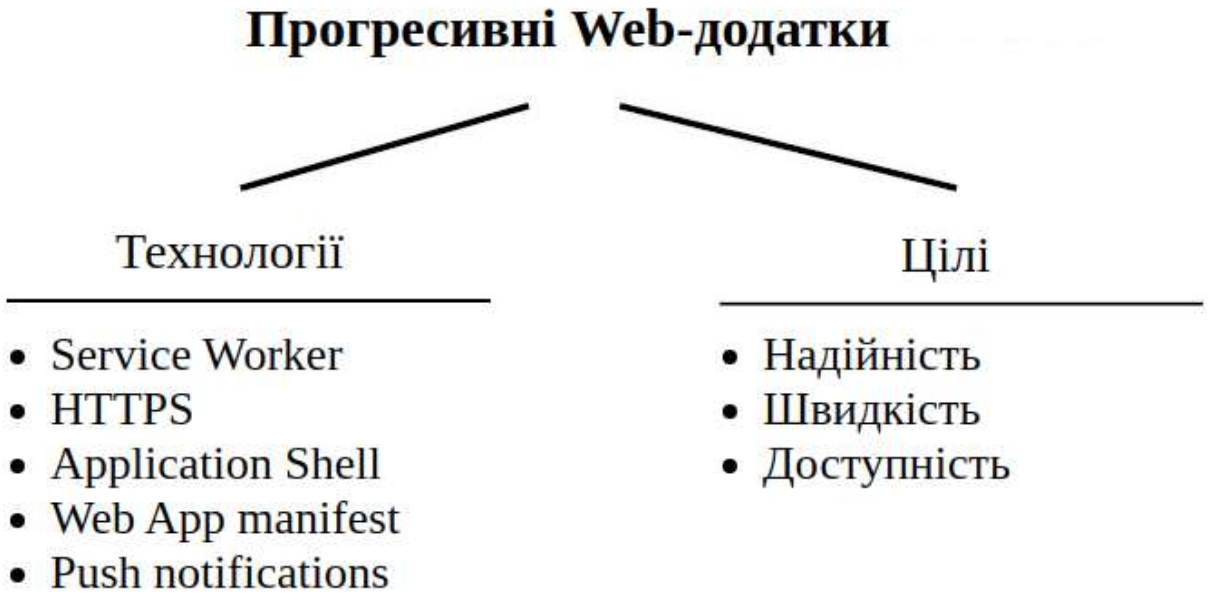

Рисунок 1.1 – PWA

Цілі PWA

Надійність – можливість працювати при будь-якому мережевому підключенні (Мобільний інтернет, оффлайн). Досягається за рахунок використання кешування на пристроях.

Швидкість – додаток надзвичайно швидко працює у порівнянні з Web-сайтами та нативними додатками, які працюють з динамічними даними з сервера. Досягається за допомогою Service Worker – технології, яка виступає як проксі між клієнтом і сервером.

Доступність – можливість встановлення на будь-яку платформу та функціонування як нативний додаток без необхідності реєстрації в Google Play або App Store, що дозволяє зекономити кошти за розміщення та спростити розповсюдження додатку [24].

Принципи PWA

Швидкість та надійність – перше завантаження буде довгим, так як і в нативному додатку, який встановлюється з Google Play або App Store. Під час першого завантаження PWA встановлюється статичний контент, наступні ж рази завантажуватись буде тільки динамічний контент.

Установка – PWA можна просто запускати у браузері, але їх також можна встановити. Встановлений PWA виглядає і поводиться як всі інші встановлені програми. Він запускається з того ж місця, що і інші додатки. Є можливість контролювати настройки екрану-заставки, значки та багато іншого. PWA запускається у вікні програми без адресного рядка або іншого призначеного для користувача інтерфейсу браузера.

Нативні додатки – для роботи PWA використовується одна кодова база для Web, яка використовується між платформами. Як Android є віртуальною машиною для Android-додатків, так і браузер Браузер виступає в ролі віртуальної машини для PWA. Як нативний додаток звертається через файлову систему до своїх ресурсів, так само і PWA звертається до своїх ресурсів – по HTTP, але ці ресурси зберігаються локально [17].

Переваги PWA:

- не потрібно писати різні версії додатку для різних платформ;
- не потрібно реєструвати додаток в Google Play або App Store і платити за це;
- можливість роботи з десктопом;
- всі основні ресурси можна зберігати на клієнті, по мережі буде передаватися тільки динамічний контент;
- можливість налаштування PUSH-нотифікацій;

– використання кешування та service worker;
– робота додатку оффлайн;
– перехід на PWA із звичайного додатку здійснюється поступово [24].

1.2 Базові поняття, принципи та технології створення Web-додатків

Web-додаток – це ПЗ, яке працює на Web-сервері, на відміну від програмних програм на базі комп'ютерів, які локально зберігаються на ОС пристрою. Доступ до Web-додатків користувач здійснює через Web-браузер із активним підключенням до Інтернету. Ці програми працюють за допомогою структури клієнт-сервер – користувачеві надаються послуги через сервер, розміщений третьою стороною. Прикладами широко використовуваних Web-додатків є: Web-пошта, Інтернет-магазини, Інтернет-банкінг та Інтернет-аукціони.

Загальна відмінність між динамічною Web-сторінкою будь-якого виду та Web-додатком часто незрозуміла. Web-сайти, які називають Web-додатками мають аналогічні функції нативних програм для ПК або смартфонів. HTML5 представив підтримку мови для створення програм, завантажених у вигляді Web-сторінок, які можуть зберігати дані локально та продовжувати функціонувати в режимі офлайн. Web-додатки часто реалізовані за принципом SPA.

SPA

SPA більше схожі на нативні програми, оскільки вони відкидають більш типову Web-парадигму переміщення між окремими сторінками за різними URL-адресами.

SPA взаємодіє з Web-браузером, динамічно переписуючи поточну Web-сторінку новими даними з Web-сервера, замість браузера за замовчуванням завантаження цілих нових сторінок. Метою є – швидші переходи, які дозволяють Web-сайту відчувати себе як нативний додаток.

У SPA весь необхідний HTML, JS і CSS-код браузер отримує із завантаженням однієї сторінки, або відповідні ресурси динамічно завантажуються та додаються на сторінку за необхідності, як правило, у

відповідь на дії користувача, наприклад при переході по елементах навігації. Сторінка не завантажується в будь-який момент процесу, а також не передає керування на іншу сторінку, хоча хеш розташування або API історії HTML5 можуть використовуватися для забезпечення сприйняття та навігації окремих логічних сторінок у додатку.

Переваги SPA:

- SPA швидкий, оскільки більшість ресурсів (HTML + CSS + JS) завантажуються лише один раз протягом усього життя програми. Вперед та назад передаються лише дані.
- Розробка спрощена. Немає необхідності писати код для візуалізації сторінок на сервері. Починати набагато простіше, тому що зазвичай можна розпочати розробку з файлового файлу: // URI, не використовуючи жодного сервера.
- SPA-адреси легко налагоджувати в Chrome, оскільки є можливість контролювати мережеві операції, досліджувати елементи сторінки та пов’язані з нею дані.
- Зробити мобільний додаток простіше, оскільки розробник може повторно використовувати один і той самий резервний код для Web-додатків та нативного мобільного додатка.
- SPA може ефективно кешувати будь-які локальні сховища. Додаток надсилає лише один запит, зберігає всі дані, тоді він може використовувати ці дані та працює навіть офлайн.

Мобільні додатки

Також існує кілька способів орієнтації на мобільні пристрої під час створення Web-додатку.

Responsive Web-дизайн можна використовувати для створення Web-додатку – будь то звичайний Web-сайт або SPA, з яким можна комфортно працювати на невеликих екранах, та який функціонує коректно на сенсорних екранах.

PWA – це Web-додатки, які завантажуються як звичайні Web-сторінки чи Web-сайти, але можуть запропонувати користувачеві функціональні можливості, такі як робота в автономному режимі та апаратний доступ до пристрою, традиційно доступний лише для нативних мобільних додатків.

Нативні додатки або мобільні додатки працюють безпосередньо на мобільному пристрої так само, як звичайне ПЗ працює безпосередньо на настільному комп'ютері, без Web-браузера (і, можливо, без необхідності підключення до Інтернету); вони, як правило, написані на Java або Kotlin (для пристроїв Android) або Objective-C або Swift (для пристроїв iOS). Останнім часом фреймворки дозволяють розробляти нативну програму для всіх платформ, що використовують інші мови, ніж стандартну рідну мову.

Гібридні програми вбудовують Web-сайт у нативну програму, можливо, використовуючи гібридні рамки. Це дозволяє розробляти за допомогою Web-технологій (і, можливо, безпосередньо скопіювати код з існуючого Web-сайту для мобільних пристроїв), зберігаючи при цьому певні переваги нативних програм (наприклад, прямий доступ до апаратних засобів пристрою, робота в автономному режимі, видимість магазину додатків).

Гібридні програми включають:

- Apache Cordova;
- Electron;
- Haxe;
- React Native;
- Xamarin.

JavaScript-фреймворки

JavaScript-фреймворк – це фреймворк, написаний на JavaScript. Він відрізняється від JavaScript бібліотек своїм потоком управління. Бібліотека пропонує функції, які можна викликати батьківському коді, тоді як фреймворк визначає весь дизайн програми. Розробник не викликає фреймворк; натомість фреймворк викликає і використовує код певним чином. Деякі фреймворки

JavaScript дотримуються парадигми MVC, розробленої для поділу Web-додатку на ортогональні одиниці для поліпшення якості коду та підтримуваності.

Розробка проекту подібна до будівництва будинку, і саме так працюють фреймворки JavaScript. Можна написати весь код з нуля для кожного аспекту Web-додатку. Але навіщо винаходити колесо, коли люди вже створили готові рішення для спільних функцій, які не сильно відрізняються від Web-додатку до Web-додатку? Що роблять фреймворки JavaScript? Вони надають ці будівельні блоки у вигляді написаного коду.

Навіщо використовувати фреймворк JavaScript?. Це економить ресурси на розробку. Це також приносить більше стимулів у процесі розробки, оскільки замість того, щоб постійно реалізовувати одній й ті ж рішення тривіальних проблем, можна зосередитись на вирішенні більш серйозних технічних та унікальних проблем [11].

Список фреймворків:

- Angular;
- React;
- Vue.js;
- Ember.js;
- ExtJS;
- Knockout.js;
- Meteor.js;
- Backbone.js.

Найбільшими та найпопулярнішими є Angular, React, Vue.js (рис. 1.2).

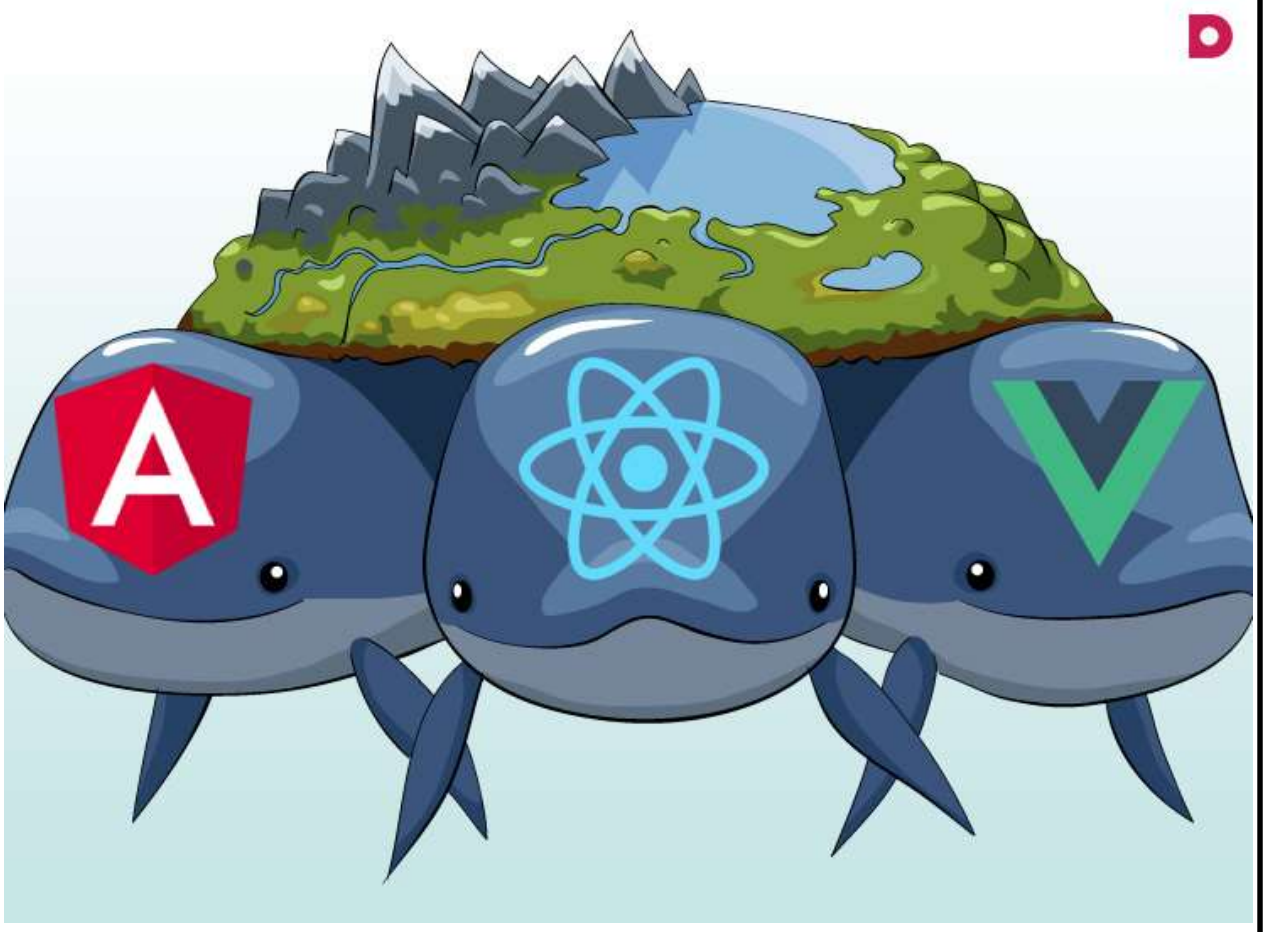

Рисунок 1.2 – Три кита JavaScript-фреймворків

React

React – це фреймворк JavaScript з відкритим кодом для створення інтерфейсів користувача. Він підтримується Facebook та спільнотою окремих розробників та компаній.

React можна використовувати як базу при розробці SPA або мобільних додатків. Однак React стосується лише надання даних у DOM, а тому створення програм React зазвичай вимагає використання додаткових бібліотек для управління станом та маршрутизації. Redux та React Router є відповідними прикладами таких бібліотек.

React – найвідоміший фреймворк Javascript з найбільшою спільнотою та екосистемою. Всі компоненти виражають свій інтерфейс користувача у функціях візуалізації, використовуючи JSX (декларативний XML-подібний синтаксис, який працює в JavaScript) для об'єднання HTML-структур із логікою. Він

фокусується на візуалізації вікон, але все ж дозволяє вашим проектам масштабувати, додаючи додаткові пакети [14].

Переваги:

Віртуальний DOM в React покращує швидкість UI та оновлення DOM роботи користувачів і DOM (об'єктна модель документа) – це логічна структура документів у форматах HTML, XHTML або XML (рис. 1.3). Характеризуючи це в простому розумінні, це угода про представлення даних та входів та даних, яка має форму дерева. Web-браузери використовують механізми компонування для перетворення або розбору HTML-синтаксису представлення в об'єктну модель документа.

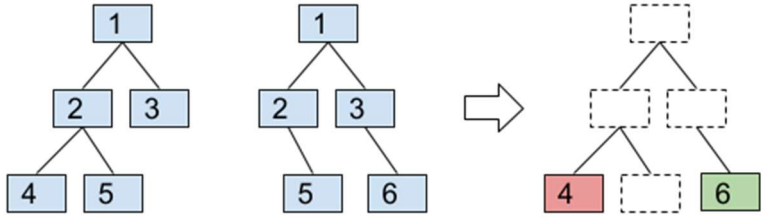

Рисунок 1.3 – Віртуальний DOM

Можливість повторного використання компонентів React
Ще одна перевага, яку Facebook представив разом з React – це можливість використовувати кодові компоненти іншого рівня в будь-який час, ще один значущий ефект економії часу.

Принцип роботи дизайнерів. Вони постійно використовують одні і ті ж набори шаблонів та готових рішень. Якщо б вони цього не зробили, їм би довелось малювати корпоративні логотипи, знову і знову. Це досить очевидно: повторне використання – це ефективність проектування. У програмуванні цей процес трохи складніше. Також покращується можливість оновлення системи, оскільки кожна зміна вже не може вплинути на роботу інших компонентів системи. Це дозволяє повторно використовувати компоненти, які самі по собі не

змінюють, зробити програмування більш точним, ергономічним та зручним для розробників.

Односторонній потік даних в ReactJS.

Дозволяє здійснювати безпосередню роботу з компонентами та використовує прив'язку даних вниз, щоб гарантувати, що зміни дочірніх структур не впливають на їх батьків. Це робить код стабільним. Більшість складних систем представлення мають істотний, але зрозумілий недолік – структуру потоку даних. У системі представлення-моделі дочірні елементи можуть впливати на батьків, якщо їх змінити. Facebook усунув ці проблеми в React JS, зробивши це просто системою view.

Redux: зручний контейнер стану

Redux спрощує зберігання та керування станом компонентів у великих додатках із багатьма динамічними елементами, де стає все важче. Redux зберігає стан програми в одному об’єкті і дозволяє кожному компоненту отримати доступ до стану програми, не маючи стосунку з дочірніми компонентами або використовуючи зворотні виклики. Redux реалізований за принципом Flux, що створює єдине джерело істини даних, що надає широкі можливості для зручного керування станом додатку.

І React, і Redux оснащені гідним набором відповідних інструментів, які полегшують життя розробника. Наприклад, розширення React Developer Tools для Chrome і подібне для Firefox дозволяють досліджувати ієрархії компонентів у віртуальному DOM та редагувати стани та властивості. Крім того, можна перевірити React Sight, який візуалізує стан та підтримує дерева.

Недоліки:

Відсутність встроєних інструментів та CLI.

Якщо працювати з DOM, роутингами, контролем стану – необхідно встановити бібліотеки для цього. Також для кожного проекту необхідно з нуля налаштовувати всі залежності та конфігурації.

Недостатня документація для нових функцій та змін.

Проблема з документацією пов'язана з постійними випусками нових інструментів. Різні та нові бібліотеки, такі як Redux та Reflux, обіцяють прискорити роботу бібліотеки або покращити всю екосистему React. Зрештою, розробники борються за інтеграцію цих інструментів з ReactJS. Деякі члени спільноти вважають, що технології React оновлюються та прискорюються настільки швидко, що немає часу написати належну інструкцію. Щоб вирішити це, розробники пишуть власну документацію для конкретних інструментів, якими вони користуються в поточних проектах.

HTML в JavaScript – JSX як бар'єр.

ReactJS використовує JSX. Це розширення синтаксису, яке дозволяє змішувати HTML з JavaScript. JSX має свої переваги (наприклад, захист коду від ін'єкцій), але деякі члени спільноти розробників вважають JSX серйозним недоліком. Розробники та дизайнери скаржаться на складність JSX та її круту криву навчання.

Відсутність єдиного формату побудови проекту.

Наприклад, якщо працювати з різними проектами на React, у них може бути зовсім різна структура, стиль написання, та спосіб використання React через відсутність якогось стандарту або рекомендацій.

Недостатня документація для нових функцій та змін.

Проблема з документацією пов'язана з постійними випусками нових інструментів. Різні та нові бібліотеки, такі як Redux та Reflux, обіцяють прискорити роботу бібліотеки або покращити всю екосистему React. Зрештою, розробники борються за інтеграцію цих інструментів з ReactJS. Деякі члени спільноти вважають, що технології React оновлюються та прискорюються настільки швидко, що немає часу написати належну інструкцію. Щоб вирішити це, розробники пишуть власну документацію для конкретних інструментів, якими вони користуються в поточних проектах.

Angular

Angular – це платформа та основа для побудови односторінкових клієнтських додатків за допомогою HTML та TypeScript. Angular пишеться в TypeScript. Він реалізує основну та додаткову функціональність як набір бібліотек TypeScript, які можна імпортувати у додатки.

Архітектура програми Angular спирається на певні фундаментальні концепції. Основними будівельними блоками є NgModules, які забезпечують контекст компіляції компонентів. NgModules збирають відповідний код у функціональні набори; Angular додаток визначається набором NgModules. У додатку завжди є принаймні кореневий модуль, який дозволяє завантажувати, і, як правило, має ще багато функціональних модулів.

Компоненти визначають представлення даних, що представляють собою набори елементів екрану, які можна вибирати та змінювати відповідно до логіки та даних вашої програми.

Компоненти використовують сервіси, які надають певні функціональні можливості, безпосередньо не пов'язані з представленнями. Постачальники послуг можуть бути введені в компоненти як залежності, що робить ваш код модульним, багаторазовим та ефективним.

І компоненти, і сервіси – це просто класи, в яких є декоратори, які позначають їх тип та надають метадані, які вказують Angular, як ними користуватися.

Метадані класу компонентів асоціюють його з шаблоном, який визначає представлення. Шаблон поєднує звичайний HTML з Angular директивами та розміткою, що дозволяють Angular змінювати HTML, перш ніж відображати його для користувача.

Метадані для класу надають інформацію, яку Angular потребує, щоб зробити її доступною для компонентів через введення залежності (DI).

Компоненти програми зазвичай визначають багато представлень,

розташованих ієрархічно. Angular надає послугу маршрутизатора, щоб допомогти визначити шляхи навігації серед представлень. Маршрутизатор забезпечує складні навігаційні можливості в браузері [13].

Переваги:

Висока продуктивність

Прискорення відбувається за рахунок декількох факторів. Основний приріст забезпечується за допомогою ієрархічної ін'єкції залежності та підтримки Angular Universal.

Введення ієрархічної залежності. Angular використовує покращену ієрархічну ін'єкційну залежність (рис. 1.4) порівняно з AngularJS. Метод відокремлює фактичні компоненти від їх залежностей, проводячи їх паралельно один одному. Angular будує окреме дерево залежностей, яке можна змінити без перенастроювання компонентів. Отже, класи не мають залежностей самі по собі, але споживають їх із зовнішнього джерела.

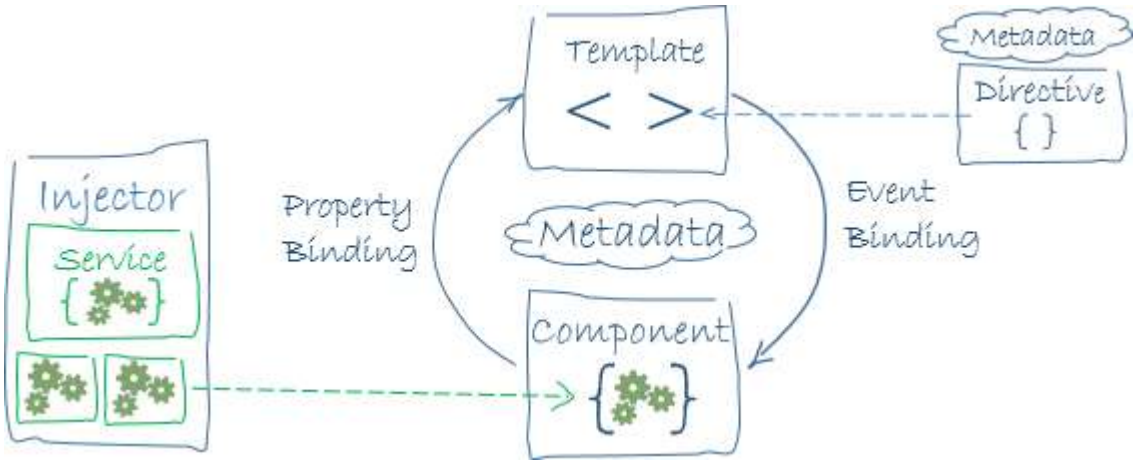

Рисунок 1.4 – Ін’єкція залежності у компонент

Angular Universal – це послуга, яка дозволяє візуалізувати програми на сервері замість браузера клієнтів. Google надає набір інструментів для попереднього відтворення вашої програми або для повторної рендерингу для кожного запиту користувача. В даний час набір інструментів підходить під рамки на сервері Node.JS і підтримує ASP.NET Core. Google заявляє, що збирається додати підтримку PHP, Python та Java.

Ivy rendering. Angular компоненти та шаблони записуються у TypeScript та HTML, але власне HTML не використовується безпосередньо у браузері. Це робить додатковий крок, коли HTML та TypeScript інтерпретуються в інструкції JavaScript. Renderer – це двигун, який перекладає шаблони та компоненти в JavaScript та HTML, які браузери можуть розуміти та відображати.

Диференціальне навантаження було додано в Angular 8 як інша методика оптимізації. Диференціальне завантаження – це спосіб завантаження вмісту та оптимізації розміру пакету. Насправді це дозволяє створити два різних пакети для застарілих браузерів та нових. Angular використовує останній синтаксис та поліфіли для нових браузерів, створюючи окремий пакет із стабільним синтаксисом для застарілих браузерів. Таким чином, диференціальне завантаження зменшує розмір пакету та швидкість завантаження для нових та старих браузерів, покращуючи загальну продуктивність [16].

TypeScript: кращий інструментарій, чистіший код та більша масштабованість.

У Angular компоненти написані за допомогою TypeScript, шаблони в HTML, доповнені за допомогою Angular. Так працює більшість фреймворків JS. Потім шаблони HTML будуть компільовані в інструкції JavaScript, тому TypeScript – це головний інструмент для роботи в Angular. TypeScript має кращі функції з навігації, автодоповнення та рефакторингу.

RxJS: ефективне, асинхронне програмування

RxJS – це бібліотека, яка зазвичай використовується з Angular для обробки асинхронних викликів даних. Вона дозволяє самостійно обробляти події паралельно та продовжувати виконання, не чекаючи того, що відбудеться подія, і не залишати Web-сторінку без відповіді. RxJS працює як конвеєр, де виконання розбивається на окремі та взаємозамінні частини, а не прив'язується до однієї людини [25].

Потужність Angular CLI

- Можливість створення кількох додатків та бібліотек за допомогою однієї команди.
- Підтримка створення шаблонів.
- Підтримка створення сутностей Angular (компонент, модуль, директива).
- Підтримка декількох мов від ng xi18n.
- Підтримка тестів.
- Підтримка правопису.
- Можливість легкої міграції версій.

Недоліки:

Надлишковість.

Архітектуру на основі компонентів є основною перевагою Angular, але спосіб управління компонентами є надто складним. Наприклад, може знадобитися до п'яти файлів для одного компонента в Angular, потрібно вводити залежності та оголошувати інтерфейси життєвого циклу компонентів.

Крута крива навчання

Для новачків знайомих з JavaScript, щоб вивчити та використовувати новий Angular, знадобиться більше часу порівняно з React або Vue. Масив тем та аспектів, які слід висвітлити, великий: модулі, введення залежності, компоненти, послуги, шаблони тощо. Також для багатьох розробників складним є синтаксис Angular, який схиляє до написання коду в ООП стилі.

Ще один бар’єр – RxJS, реактивна бібліотека програмування для асинхронного програмування. ЇЇ вивчення, принаймні на базовому рівні, є обов'язковим для використання Angular.

Vue.js

Vue – прогресивний фреймворк для побудови інтерфейсів користувача. На відміну від інших монолітних фреймворків, Vue розроблений з нуля, щоб бути поступово прийнятним. Основа фреймворку орієнтована лише на шар

представлення, і її легко підбирати та інтегрувати з іншими бібліотеками або існуючими проектами. З іншого боку, Vue також чудово здатний підтримувати складні SPA сторінки, коли їх використовують у поєднанні із сучасними бібліотеками інструментів та їх підтримуючих [15].

Переваги:

Крихітний розмір Завантажений zip із фреймворком важить 18KB. У порівнянні з React (98.81KB) та Angular (281.5KB). Фреймворк не тільки швидко завантажується та встановлюється, але й позитивно впливає на SEO та UX.

Віртуалізація DOM та продуктивність.

Продуктивність є одним з ключових факторів, який може впливати на вибір фреймворку. Фактичні орієнтири наведені на сторінці порівняння Vue. Наприклад, під час тестування DOM-компонентів, пов'язаних із оновленими даних, Vue.js є ефективнішим, ніж Angular та React. Звичайно, це далеко не лідируюча позиція, де Vanilla.js є лідером, але загальна картина здається оптимістичною.

Реактивне двостороннє прив'язування даних.

Ще одна перевага в маніпуляціях з DOM – це двостороння прив'язка даних, успадкована Vue від Angular. Двостороння прив'язка даних – це зв'язок між оновленнями даних моделі та представленням. Зв'язані компоненти містять дані, які можна час від часу оновлювати. За допомогою двостороннього прив'язки даних простіше оновлювати пов’язані компоненти та відстежувати дані оновлення.

Однофайлові компоненти та читабельність.

Кожен фрагмент вашої майбутнього додатку або веб-сторінки у Vue є компонентом. Компоненти представляють інкапсульовані елементи вашого інтерфейсу. У Vue.js компоненти можна записати в HTML, CSS та JavaScript, не поділяючи їх на окремі файли.

Інтеграційні можливості та гнучкість.

Важливим аспектом будь-якої нової технології є можливість інтеграції з існуючими програмами. З Vue.js це так просто, як пиріг, оскільки він покладається лише на JavaScript і не потребує інших інструментів для роботи.

Vue також дозволяє писати шаблони за своїм бажанням: використовуючи HTML, JS або JSX (розширення синтаксису JavaScript). Між своїми компонентами та легким характером Vue може використовуватися майже в будь-якому проекті. Це означає, що перехід від React або Angular не буде великою проблемою, оскільки внутрішня організація Vue – це сукупність цих двох.

Розщеплення коду програми – це фактично архітектурний підхід, який називається Компонентна архітектура, і він також використовується в Angular та React. Такий архітектурний підхід має багато переваг, які описані вище.

Гарна інструментальна екосистема.

За 5 років свого існування Vue.js отримав потужний набір інструментів для роботи. Майбутній випуск Vue CLI 3 – це повне перезапис, який забезпечить купу нових функцій. Vue CLI 3 підтримуватиме Babel та TypeScript поза коробкою, надаватиме Unit-тестування, інструменти тестування e2e та систему встановлення плагінів. Vue також має власні інструменти для налагодження браузера, візуалізацію сервера та менеджер стану.

Низька крива вивчення.

Щоб розпочати працювати з Vue немає необхідності у вивченні глибоких знань про бібліотеки, JSX та TypeScript, як це часто буває з іншими передовими технологіями. Все, що потрібно для початку, – це основні знання HTML, CSS та JavaScript.

Вичерпна документація.

Вона добре структурована і охоплює всі можливі теми, точно описуючи все, від встановлення до більш глибоких речей, таких як реактивність та масштабування програми. Що ще важливіше, є також розділ, який порівнює Vue з іншими структурами JS та називає речі, які мають спільне (наприклад, віртуальний DOM у Vue та React, синтаксис шаблонів у Vue та Angular).

Недоліки:

Мовний бар'єр.

Прийняття Vue на таких підприємствах, як Xiaomi та Alibaba, допомогло популяризувати рамки та створило попит на ринку праці. Завдяки популярності Vue.js у Китаї значна частина її змісту та обговорень ведеться китайською.

Китайський Великий Брандмауер відрізняється у цій країні, тому що зараз багато недоступних популярних ресурсів. Це ускладнює навчання та використання React або Angular. Vue є кращим варіантом.

Отже, під час пошуку відповідей на питання по Vue можна легко зіткнутися з обговореннями на форумі, описами плагінів та інструкціями китайською мовою. Це може бути проблемою для не китайськомовних інженерів.

Складність реактивності.

Додаток Vue.js складається з компонентів, з якими користувач може взаємодіяти. Кожен компонент має свій спостерігач, який відображає дані щоразу, коли користувач запускає компонент. Система реактивності видає лише викликані фрагменти даних. Ця система не така розумна і часто помиляється під час читання даних, тому вимагає «вирівнювання даних».

СУБД

Microsoft SQL Server

Microsoft SQL Server – це система управління реляційними базами даних, розроблена Microsoft. Як сервер бази даних, це програмний продукт з основною функцією зберігання та отримання даних, як цього вимагають інші програмні програми – який може працювати або на тому ж комп'ютері, або на іншому комп'ютері в мережі [24].

Переваги:

Безкоштовне видання для розробників. Видання для розробників SQL Server є абсолютно безкоштовним і включає всі функції корпоративного SQL Server.

Велика кількість можливостей для адміністрування, оптимізування запитів, дата-аналізу і т.д.

Корисна та детальна добре структурована онлайн-документація.

Недоліки:

Складна настройка продуктивності. Оптимізація запитів та налаштування ефективності можуть бути складними для спеціалістів із даних, які не мають глибоких спеціалізованих знань.

Відсутність власної підтримки для управління джерелом даних. Щоб зберегти всі зміни, внесені до об’єктів бази даних, потрібно використовувати сторонні інструменти.

Відсутність оффлайн-підтримки.

Cloud Firestore

Cloud Firestore – це гнучка, маштабована база даних для мобільних, Web та серверних розробок від Firebase та Google Cloud Platform. Дані синхронізуються між клієнтськими програмами через realtime listeners та пропонує автономну підтримку для мобільних пристроїв та Інтернету для створення додатків, які працюють незалежно від затримки мережі чи підключення до Інтернету. Cloud Firestore також пропонує безперешкодну інтеграцію з іншими продуктами Google Cloud Platform [18].

Переваги:

Cloud. Firestore відразу знаходиться у Cloud і тому не потребує додаткових маніпуляцій для завантаження та деплою.

Гнучкість. Модель даних Cloud Firestore підтримує гнучкі, ієрархічні структури даних. Документи впорядковані у колекції. Документи можуть містити складні вкладені об'єкти та підколекції.

Оновлення в реальному часі. Cloud Firestore використовує синхронізацію даних для оновлення даних на будь-якому підключеному пристрої. Однак він також розроблений для того, щоб робити прості одноразові запити ефективно.

Офлайн-підтримка. Cloud Firestore кешує дані, які активно використовує

додаток, тому програма може записувати, читати, слухати та запитувати дані, навіть якщо пристрій перебуває в режимі офлайн. Коли пристрій повернеться в Інтернет, Cloud Firestore синхронізує всі локальні зміни з Cloud Firestore.

Масштабованість. Cloud Firestore пропонує потужну інфраструктуру Google Cloud Platform: автоматичну реплікацію даних у багатьох регіонах, чіткі гарантії послідовності, атомні пакетні операції та реальну підтримку транзакцій. Cloud Firestore створений для обробки найважчих навантажень бази даних.

Недоліки:

Безкоштовна квота на використання ресурсів. При виході за цю квоту відбувається перехід, де йде оплата лише за використані ресурси. Ця функція зручна для невеликих, або тільки починаючих свій розвиток проектів.

1.3 Аналіз ринку рішень для ведення власного бюджету

Mint.com

Переваги:

- Безкоштовний;
- Простий UI;
- Зручний для базового бюджетування, цілей та кредитного стану;
- Двофакторна аутентифікація.

Недоліки:

- Не дуже якісна служба підтримки;
- Відсутність настільної версії;
- Відсутність працездатності офлайн;
- Проблеми із синхронізацією між різними девайсами;
- Тільки англомовна версія [21].

Moneymanagerex.org

Переваги:

- Настільні версії для Windows, macOS, Linux;
- Багатомовна;
- Імпорт даних з CSV, QIF;

– Інформація шифрується за допомогою AES.

Недоліки:

– Відсутня онлайн-версія;
– Відсутня синхронізація між платформами;
– Не дуже зручний UI та UX;
– Великі затрати часу на використання;
– Відсутність PUSH-нотифікацій [22].

BudgetTracker.com

Переваги:

– Простий інтерфейс;
– Багато бізнес-функцій (податкові відрахування, календарі, інструменти бюджетування для дітей, планування проектів);

Недоліки:

– Відсутність настільних версій;
– Відсутність працездатності офлайн;
– Відсутність версій для IOS та Android;
– Відсутність PUSH-нотифікацій;
– Занадто багато недороблених можливостей (Складається враження, що це кілька сирих функцій, зібраних разом. Жодна з цих функцій не є завершеною до кінця) [23].

Повний аналіз порівняння цих рішень представлений у таблиці (таб. 1.1).

Таблиця 1.1 – Порівняння існуючих рішень для ведення власного бюджету.

Порівняння	Вартість	Служба підтримки	Зручність	Інструменти та ресурси	Синхронізація	Доступність
Mint.com	10	4	9	8	7	8
Moneymanagerex.org	10	3	3	8	0	8
BudgetTracker.com	10	6	5	4	5	2

Висновок

Є додатки, які містять в собі потужні ідеї та хороший функціонал. Деякі з них хороші в одному, але відстають в іншому. Здається, що немає рішення, яке б закрило всі питання та зосередило всі найважливіші функції у одному додатку, який би водночас був доступним і зрозумілим для всіх. Саме тому, на мою думку, ринок потребує одного сучасного та універсального рішення, яке могло б працювати і як нативний додаток, і як онлайн ресурс з доступом від будь-якого девайсу з саме тими функціями, які потрібні користувачу.

1.4 Висновок до частини 1

Проведений аналіз цілей, принципів та переваг застосування PWA перед звичайними додатками. В ході аналізу виявлено переваги застосування PWA, а саме: кросс-платформенність з однією кодовою базою, прискорення швидкості роботи додатків за рахунок використання Service Worker та Cache Storage та підтримку оффлайн, що дає користувачу відчуття користування звичайним нативним додатком на настільних на мобільних пристроях.

Також проведений аналіз базових понять принципів та технологій створення сучасних Web-додатків. В ході аналізу встановлені базові поняття SPA, способи розробки для мобільних додатків та детально досліджені JavaScript-фреймворки, за допомогою яких можна значно покращити розробку Web-додатків. Досліджено переваги використання фреймворків, а саме: підвищення швидкості роботи та завантаження, наявність великої кількості готових рішень для тривіальних задач та багато іншого.

Проведений аналіз онлайн-СУБД рішень та дослідження їх переваг і недоліків.

Проведений аналіз ринку рішень для ведення власного бюджету.

2 ПРОЕКТУВАННЯ PWA

2.1 Постановка задачі

Необхідно розробити PWA, який даватиме користувачеві можливість ведення власного бюджету. Особливістю додатку є реалізація за допомогою технологій PWA та сучасних фреймворк-рішень, що дозволяє створити додаток, який працюватиме на всіх платформах, що підтримують сучасні браузери, на одній кодовій базі. Також повинна бути реалізована аутентифікація та реєстрація користувача через пошту та Google OAuth.

Інформація повинна зберігатися у БД. Користувач після реєстрації повинен мати можливість зручного і зрозумілого, за допомогою правильно реалізованого за сучасними стандартами та тенденціями UI і UX, працювати з власним бюджетом.

2.2 Можливості додатку

Менеджмент регулярних витрат за базовими категоріями та категоріями, які можуть створюватися індивідуально користувачем.

Менеджмент регулярних надходжень за базовими категоріями надходжень, які також можуть створюватися індивідуально користувачем.

Після введення даних додаток повинен розрахувати суму, яку кожний день можна буде виділяти на збереження.

Реалізувати можливість додавання “спонтанних” витрат та надходжень, які будуть впливати на поточний бюджет.

Реалізувати можливість відображення звітів у вигляді діаграм по витратам, доходам, категоріям за день, тиждень, місяць, рік.

2.3 Архітектура

Користувацький інтерфейс повинен взаємодіяти з БД через додатковий шар Service Worker, який являється технологією PWA (рис. 2.1). Також він повинен відсилати PUSH-нотифікації клієнтам

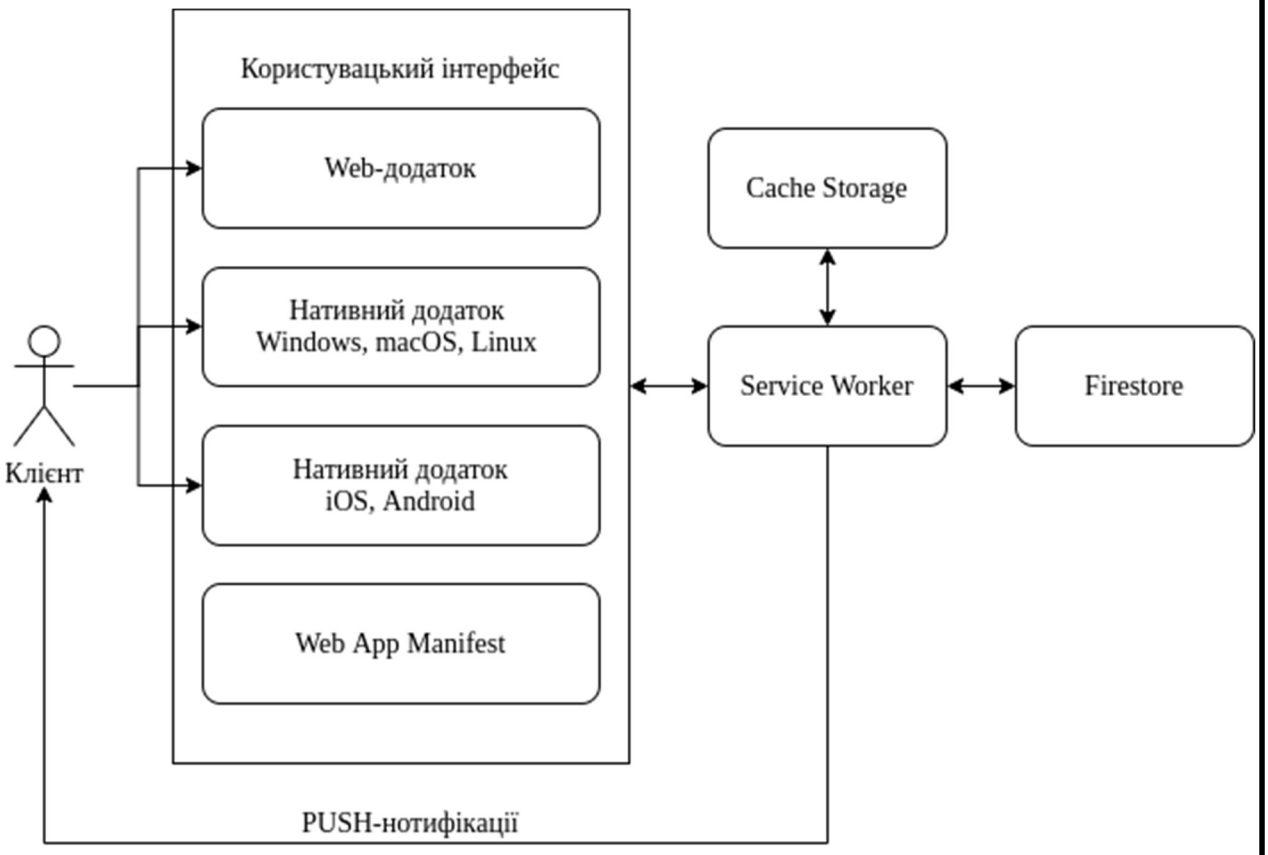

Рисунок 2.1 – Архітектура додатку

2.4 Стек технологій

Клієнтський інтерфейс вирішено розробляти на фреймворку Angular, так як він пропонує зручну, швидку розробку Web-додатків через Angular CLI та багато можливостей “з коробки”. Має високу продуктивність за рахунок ієрархічної залежності та Ivy renderer. Використовує TypeScript, що дозволяє писати більш структурований та типізований код в ООП-стилі. Також має потужний набір інструментів для роботи з асинхронним кодом RxJS та запитами. І найголовніше – гарну підтримку PWA, Service Worker та оффлайн.

Для СУБД вирішено використовувати Cloud Firestore через його гнучку

структуру і NOSQL які чудово підходить для специфіки проекту через відсутність складних та великих сутностей для зберігання у БД. Важливим є ще те, що Firestore знаходиться у Cloud, тож це дає можливість економити час та ресурси на деплой БД. Також через його можливість оновлення даних на всіх пристроях у реальному часі. Ще однією значною перевагою є наявність підтримки оффлайн.

Середовищем для розробки обрано WebStorm від JetBrains, через те, що на даний момент це найпотужніша IDE для розробки Web-додатків (рис. 2.2). Вона містить в собі безліч функцій та можливостей для розробників. Також особливістю є безкоштовні версії для навчання.

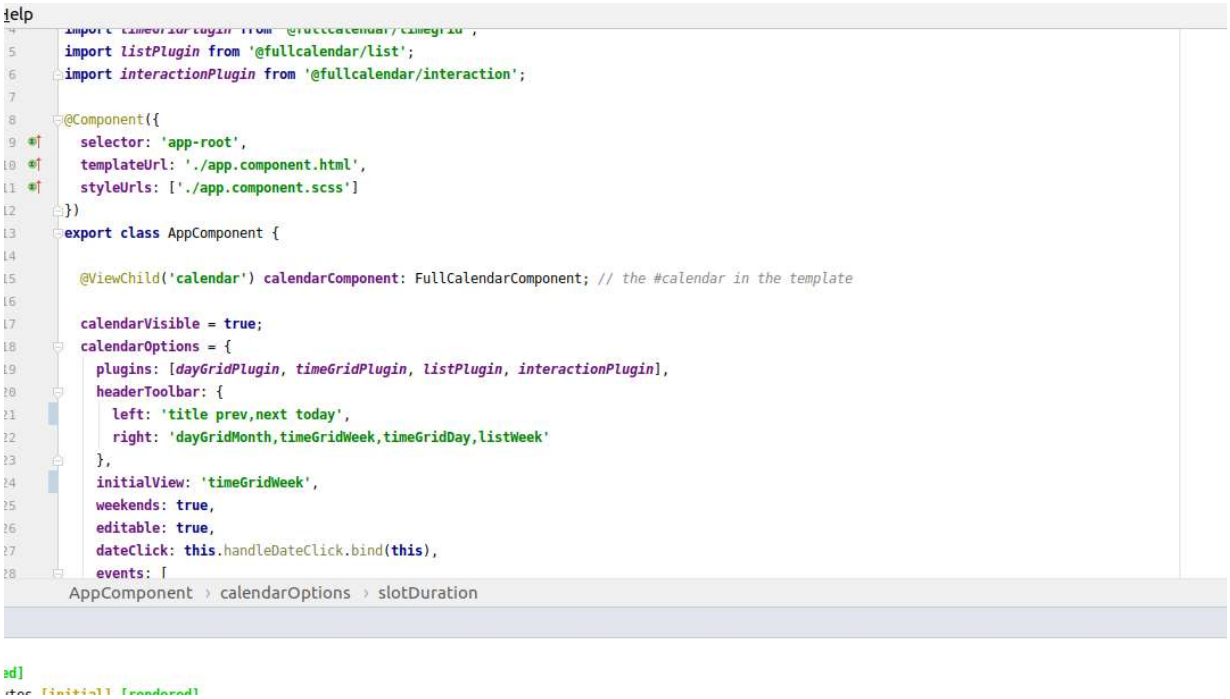

Рисунок 2.2 – Розробка у IDE WebStorm

Браузер – Google Chrome версій 79-81 через його потужність та зручні інструменти для розробників (мобільний перегляд, дебаггінг коду фреймворків) а також завдяки плагінам для фреймворків Augury для Angular та React and Redux Developer Tools (рис. 2.3).

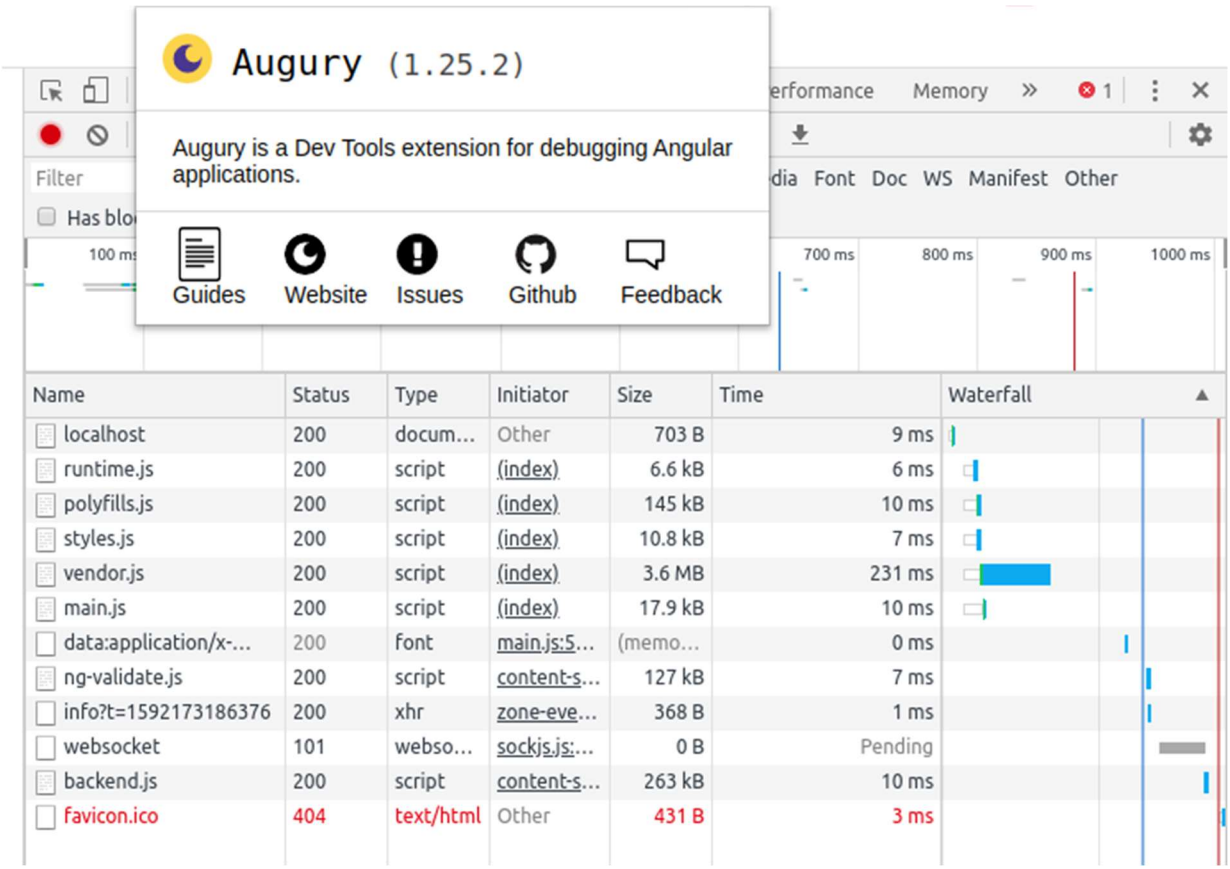

Рисунок 2.3 – Інструменти розробника Google Chrome

2.4 Висновок до частини 2

Поставлена задача створення PWA для ведення власного бюджету, описані його головні особливості та можливості, представлена архітектура майбутнього додатку, стек технологій та інструментів, для використання у процесі розробки.

3 РЕАЛІЗАЦІЯ PWA

3.1 Реалізація БД

БД додатку реалізована на основі NOSQL Cloud Firestore. Створені дві колекції (рис. 3.1). Колекція “users” потрібна для зберігання інформації про користувачів. Колекція “balances” зберігає у собі інформації про баланс користувачів, їхні регулярні та спонатні витрати та додатки.

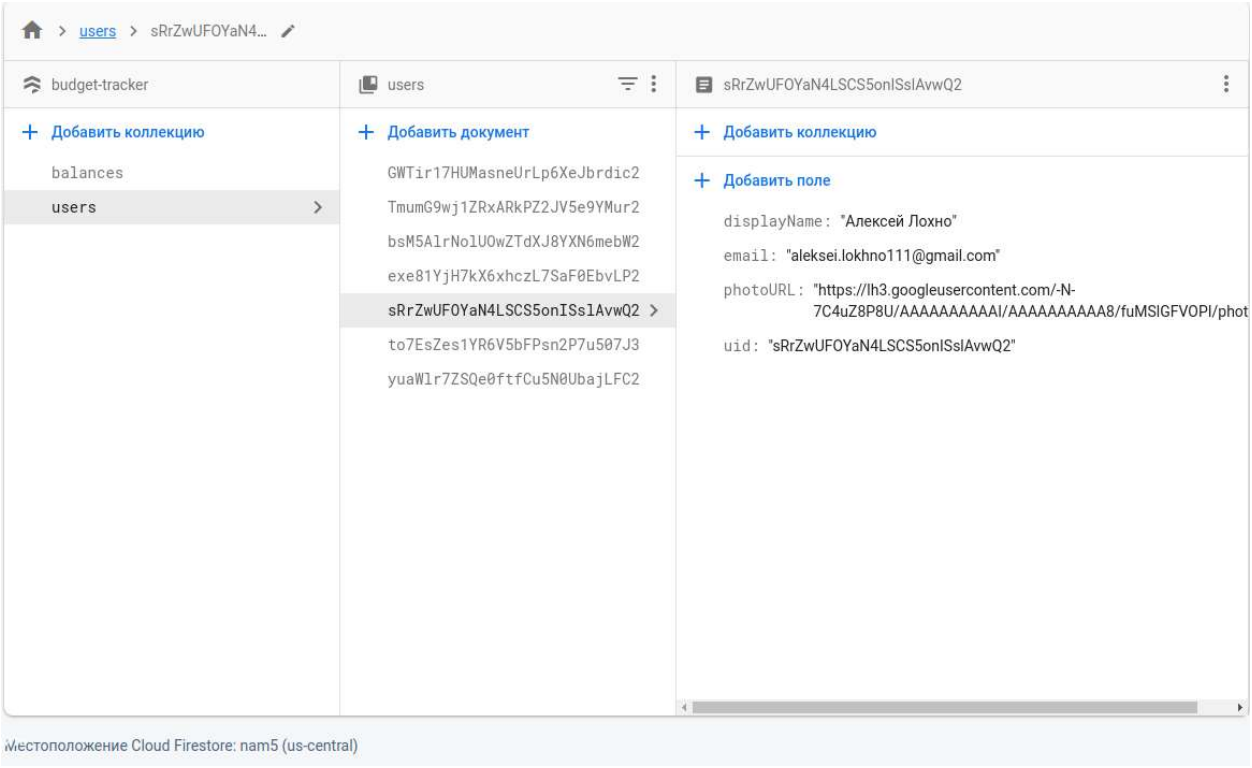

Рисунок 3.1 – Реалізована БД

Підключення до БД здійснюється за допомогою офіційної бібліотеки Angular Firestore та Firebase (рис. 3.2). Далі її можна використовувати для CRUD запитів у базу за допомогою RxJS.

```
import {Injectable} from '@angular/core';
import {AngularFirestore} from '@angular/fire/firestore';
import {AuthService} from './auth.service';
import {Router} from '@angular/router';
import {firestore} from 'firebase/app';
import {map, switchMap} from 'rxjs/operators';
import {Observable, combineLatest, of} from 'rxjs';

@Injectable({
        providedIn: 'root'
})
export class BalanceService {
        constructor(
                private afs: AngularFirestore,
                private auth: AuthService,
                private router: Router
        ) {
        }

        get(balanceId) {
                return this.afs
                        .collection<any>('balances')
                        .doc(balanceId)
                        .snapshotChanges()
                        .pipe(
                                map(doc => {
                                        return {id: doc.payload.id, ...doc.payload.data()};
                                })
                        );
        }
```

Рисунок 3.2 – Підключення Firestore

3.2 Реалізація UI

Головна сторінка додатку (рис. 3.3). На ній відображений поточний баланс користувача, його щоденний прибуток. Також відображена динаміка зростання коштів за допомогою діаграми. Внизу сторінки є кнопки, які відповідають за створення нових записів спонтанних витрат.

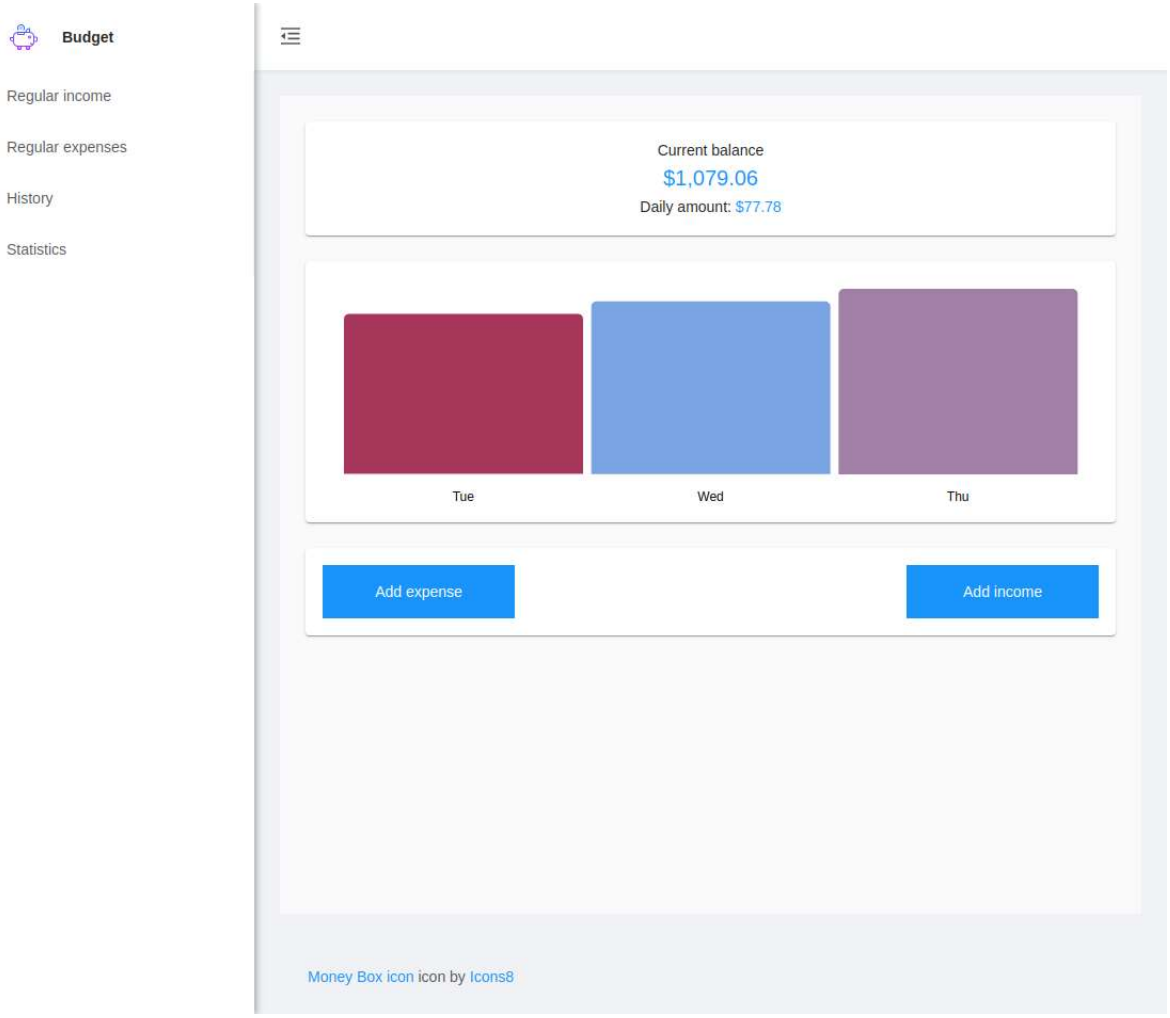

Рисунок 3.3 – Головна сторінка додатку

Сторінка реєстрації (рис. 3.4). На ній відображені поля для вводу користувачем, які необхідні для реєстрації у додатку. Також на ній міститься посилання на користувацькі погодження.

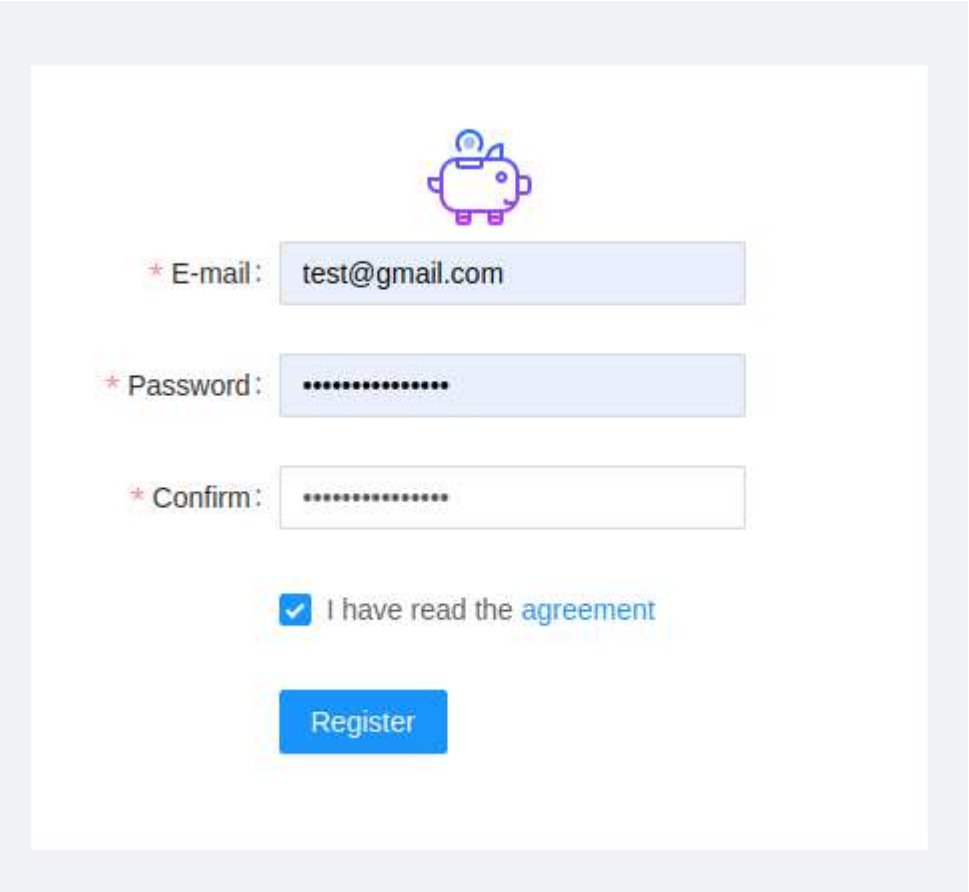

Рисунок 3.4 – Сторінка реєстрації

Сторінка авторизації (рис. 3.5). На ній відображені поля для вводу користувачем, які необхідні для авторизації у додатку. Також міститься посилання на сторінку реєстрації.

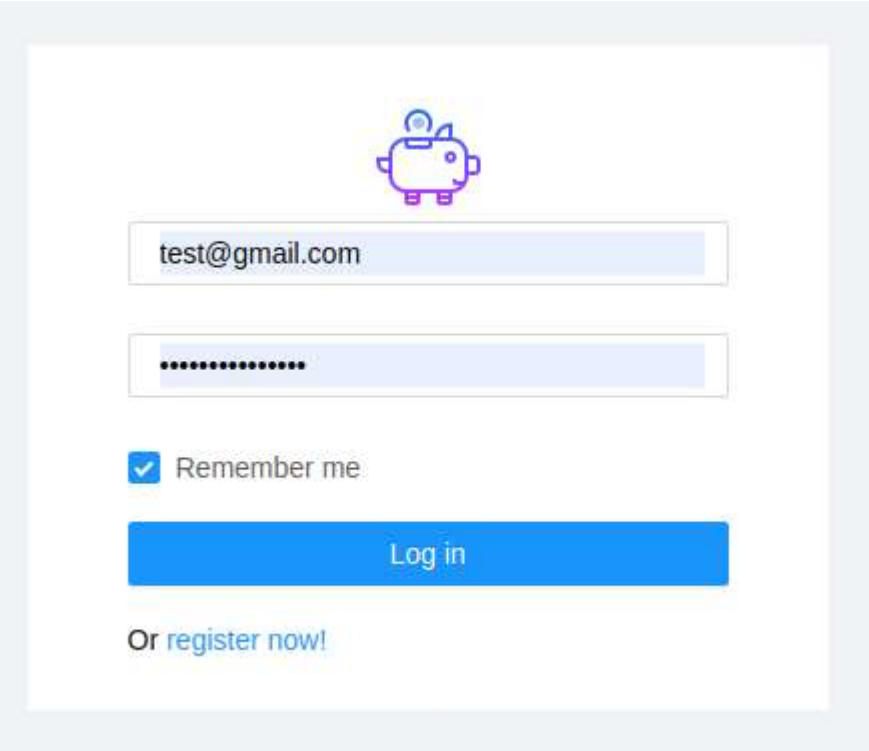

Рисунок 3.5 – Сторінка авторизації

Сторінка регулярних витрат (рис. 3.6). На цій сторінці відображені всі регулярні витрати, які заповнює користувач. Всі витрати містять в собі найменування, суму, період та категорію витрати. Додатково зверху сторінки відображені загальні витрати за певними періодами.

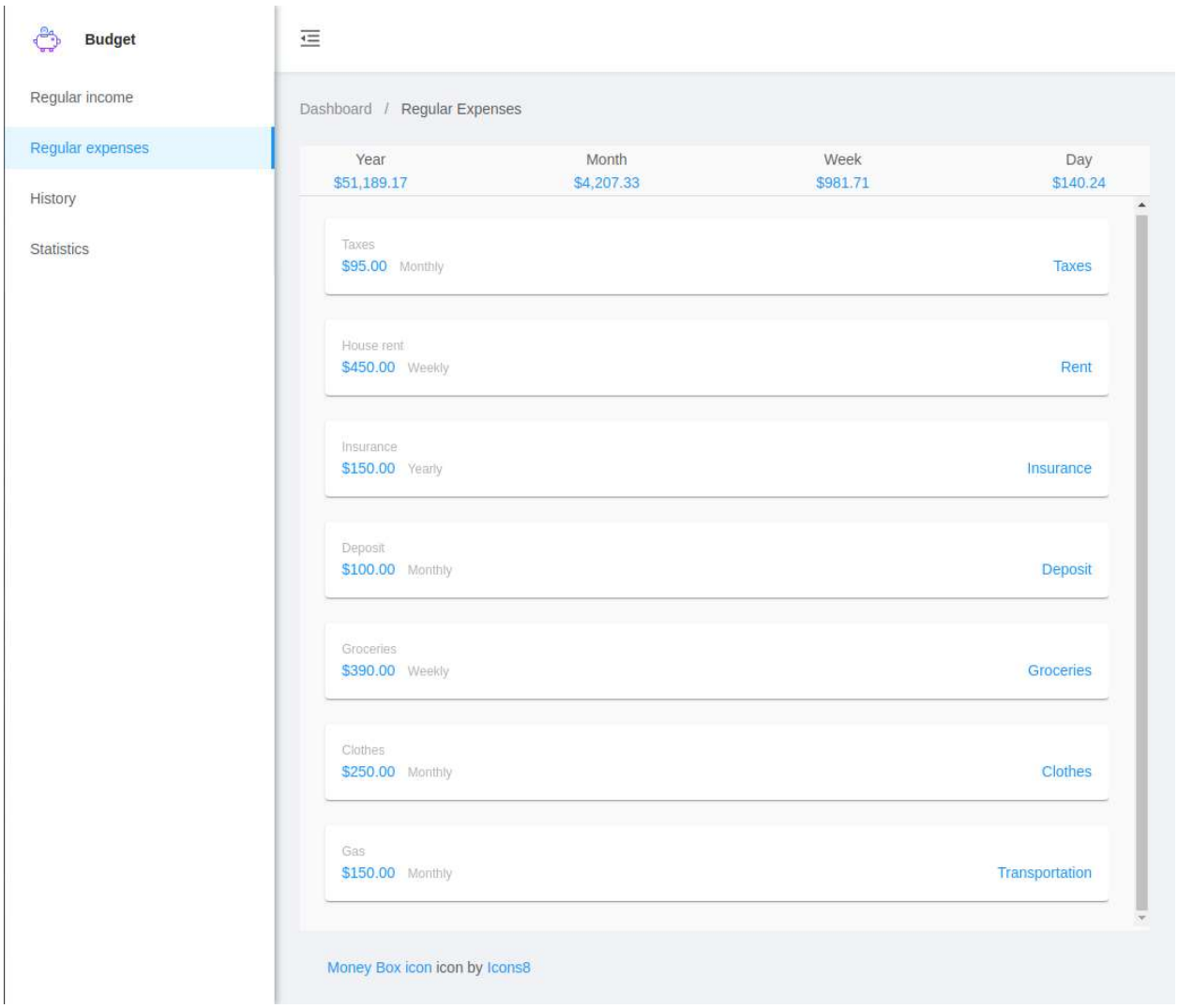

Рисунок 3.6 – Сторінка витрат

Сторінка регулярних доходів (рис. 3.7). Ця сторінка є такою самою як і сторінка витрат, тільки містисть в собі дані по доходам.

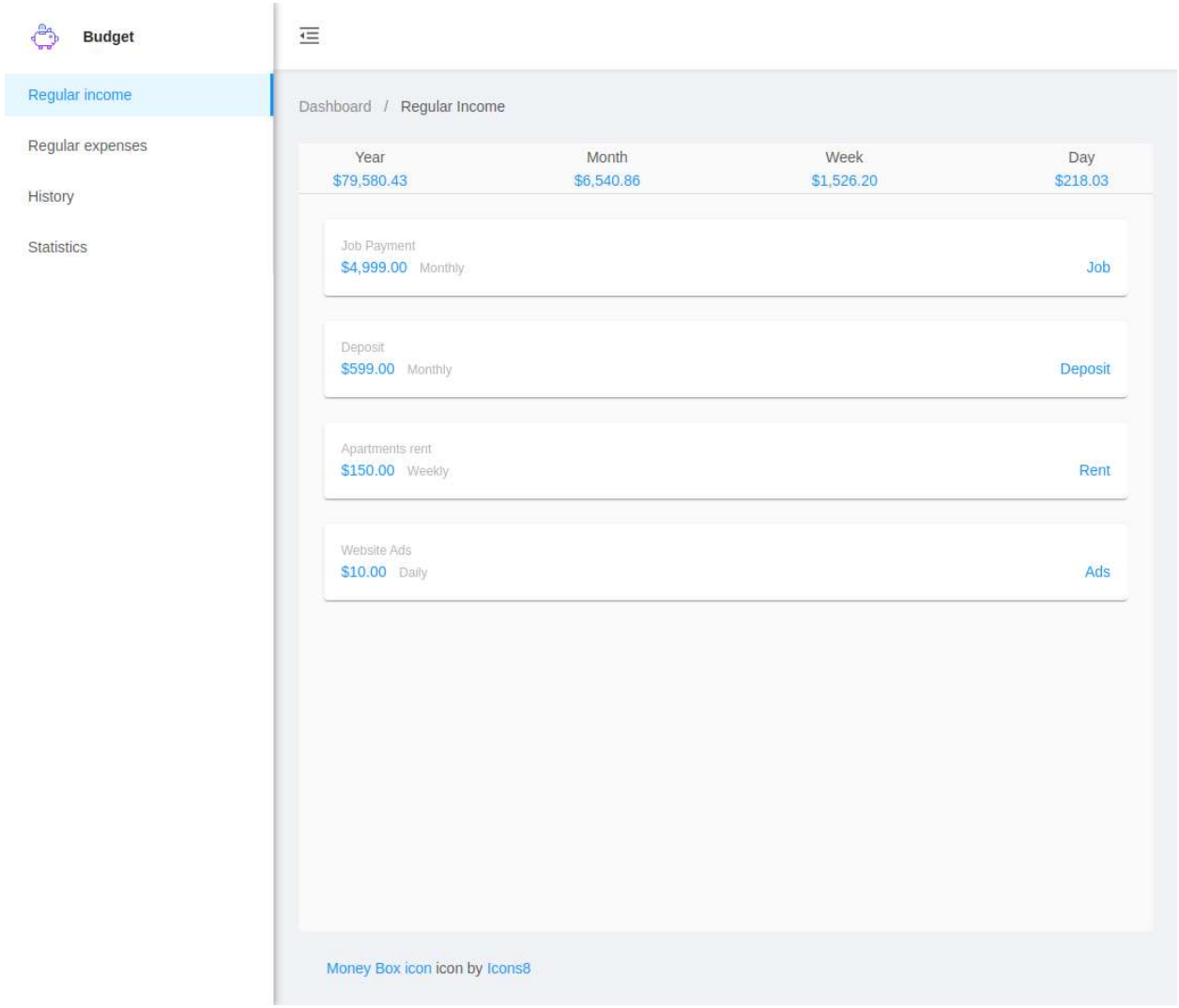

Рисунок 3.7 – Сторінка доходів

Сторінка історії спонтанних витрат та доходів (рис. 3.8). На цій сторінці відображена історія витрат, які здійснив користувач та отриманих доходів. Дані відображені на трьох вкладках. Вкладка “Both” відображає загальний стан, і містить в собі як витрати так і доходи. Вкладка “Expenses” містить всі витрати і аналогічно вкладка “Income” містить доходи. Також слід зазначити, що тип витрат відрізняється від регулярних і має додаткове поле дати, по якій і групуються записи історії. Всі записи відсортовані від найновіших до найстарших.

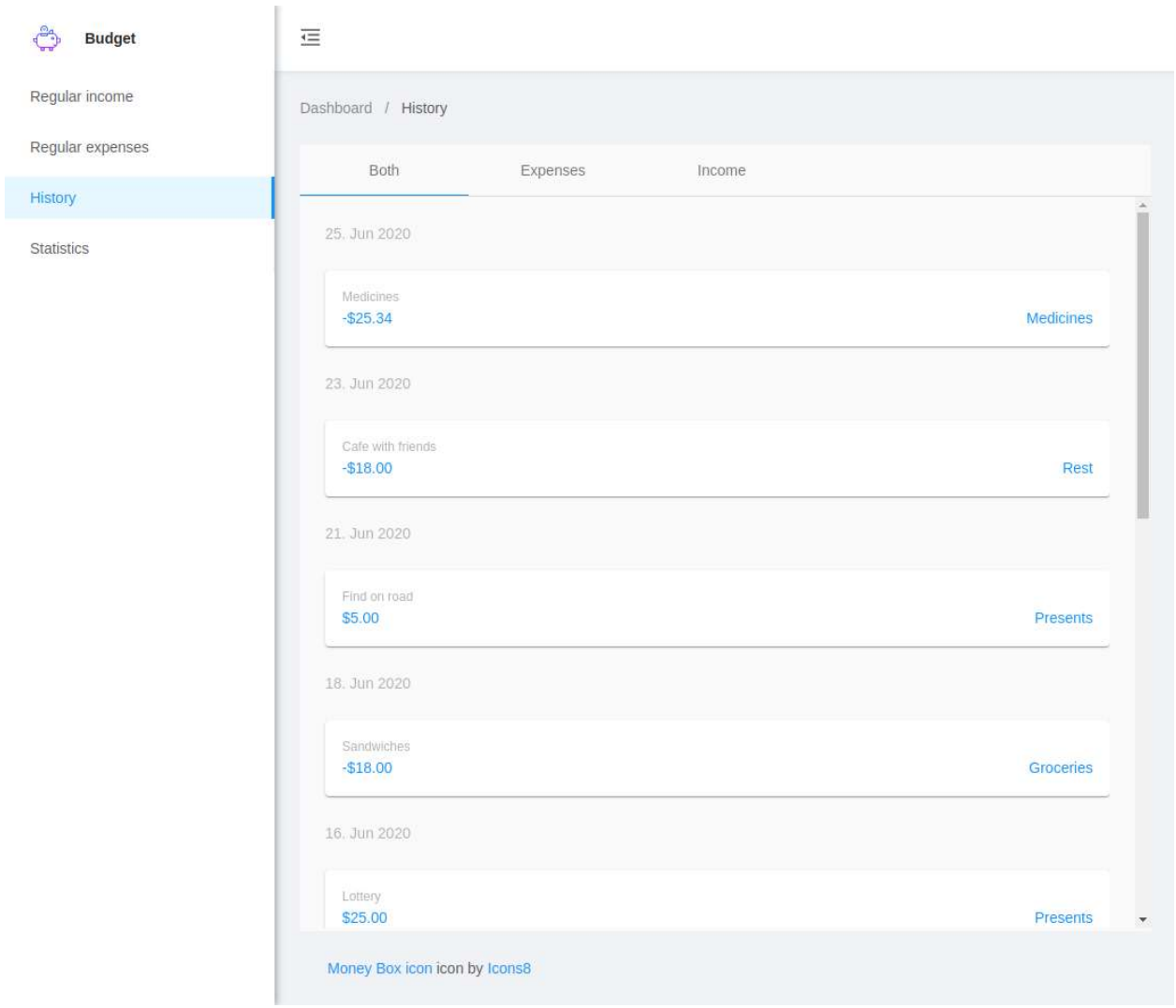

Рисунок 3.8 – Сторінка історії спонтанних витрат та доходів

Сторінка статистики (рис. 3.9). Сторінка містить у собі статистику по спонтанним витратам користувача. Статистика відображається у вигляді кругової та звичайної діаграми для зручності та наочності. Записи групуються по категоріям. Також на цій сторінці міститься загальна сума витрат або доходів в залежності від обраної користувачем вкладки.

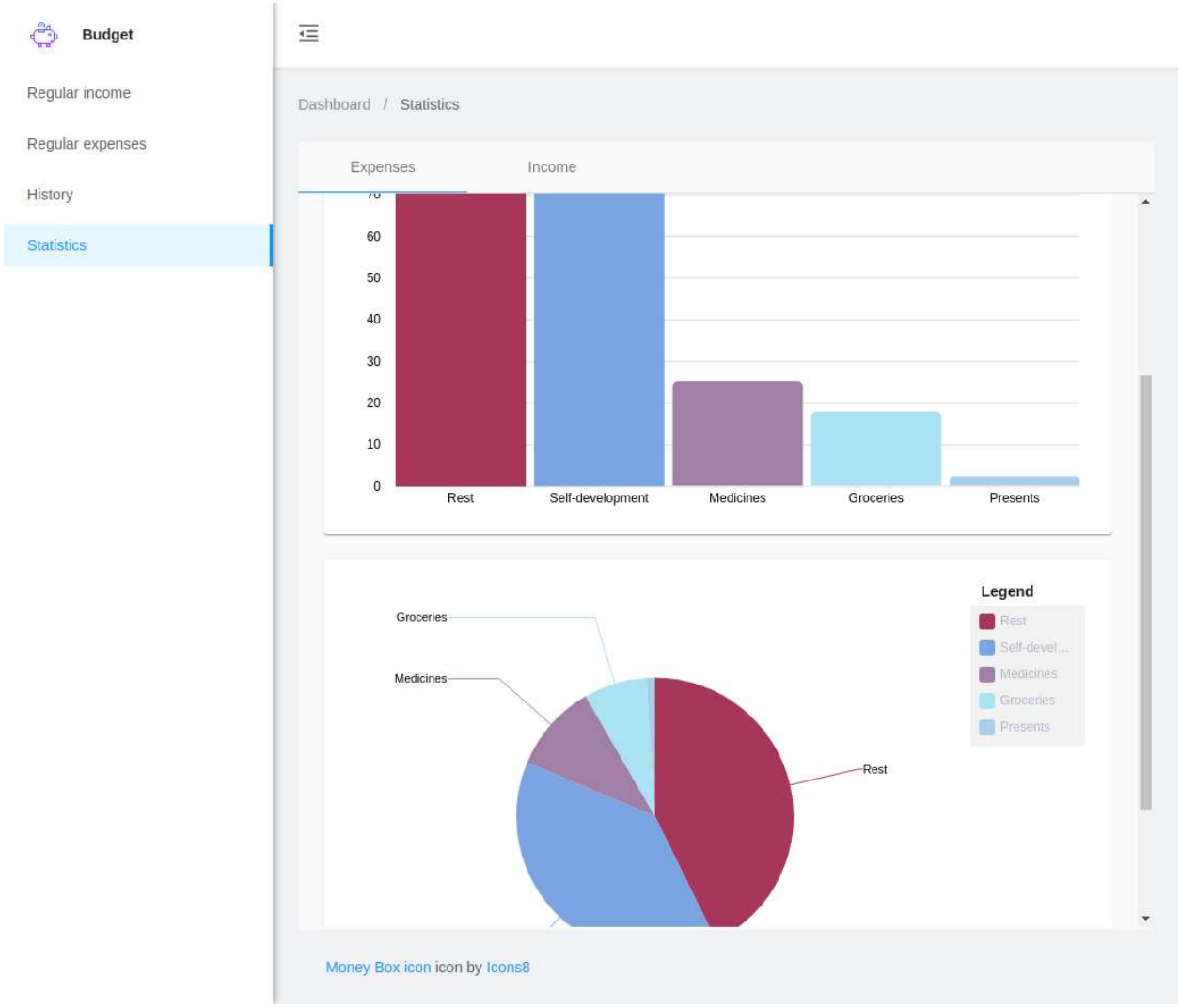

Рисунок 3.9 – Сторінка статистики з діаграмами

3.3 Реалізація PWA-підтримки

Для підтримки PWA у першу чергу необхідно створити та налаштувати Web App Manifest (рис. 3.10), у якому потрібно налаштувати параметри для PWA. Серед них обв'язково потрібно зазначити ім'я, колір теми, тип відображення та іконки, для робочого столу. Також необхідно підключити ServiceWorker.

Рисунок 3.10 – Web App Manifest

Після підключення необхідно перевірити коректність роботи та відповідність вимогам PWA. Це можна зробити за допомогою інструменту для аудиту LightHouse (рис. 3.11).

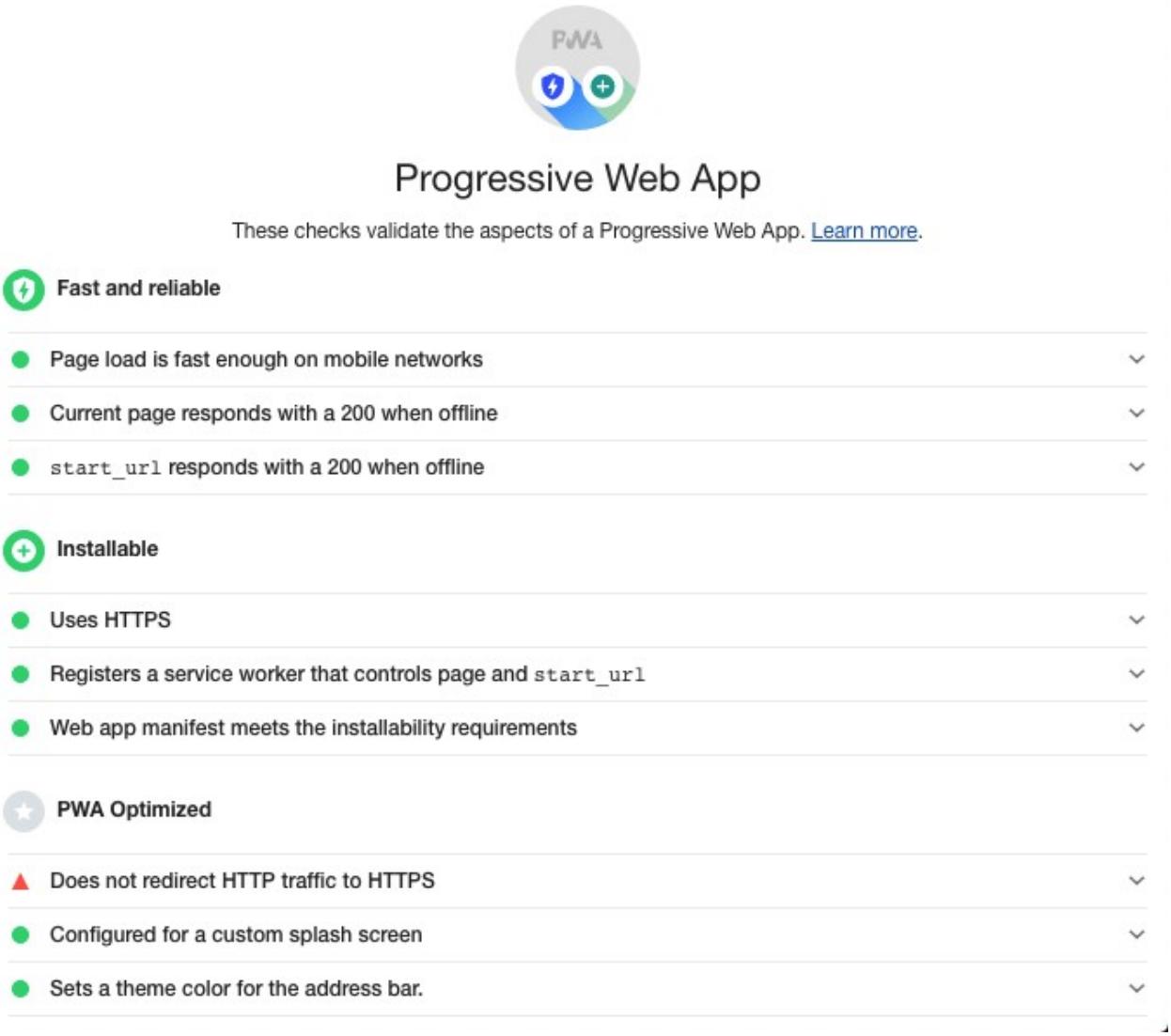

Рисунок 3.11 – Результат аудиту LightHouse

Якщо додаток відповідає вимогам, то з’являється можливість завантаження Web-додатку на девайс (рис. 3.12).

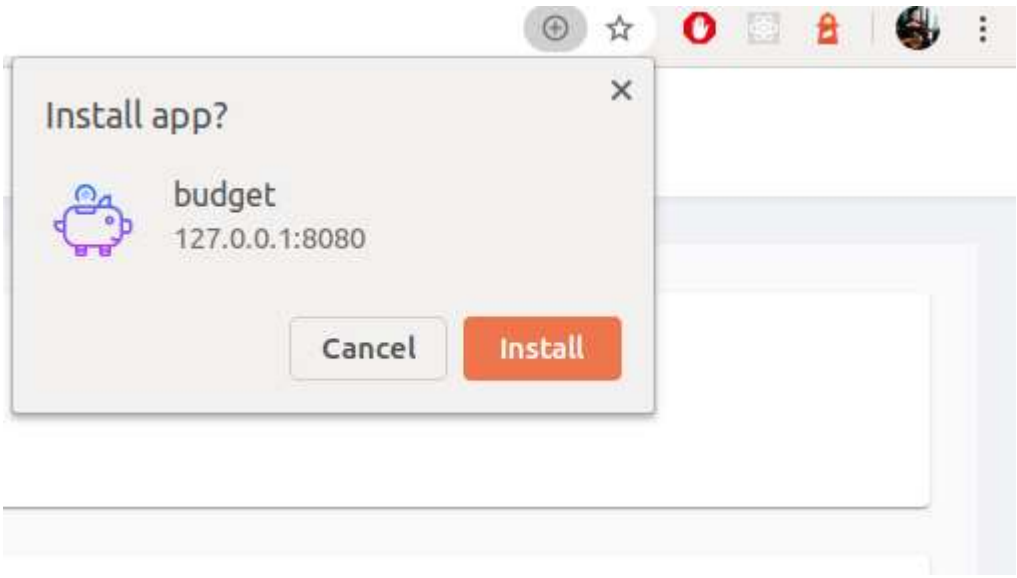

Рисунок 3.12 – Підтвердження установки на девайс

Після закінчення установки є можливість користуватися Web-додатком так само як і нативним (рис. 3.13).

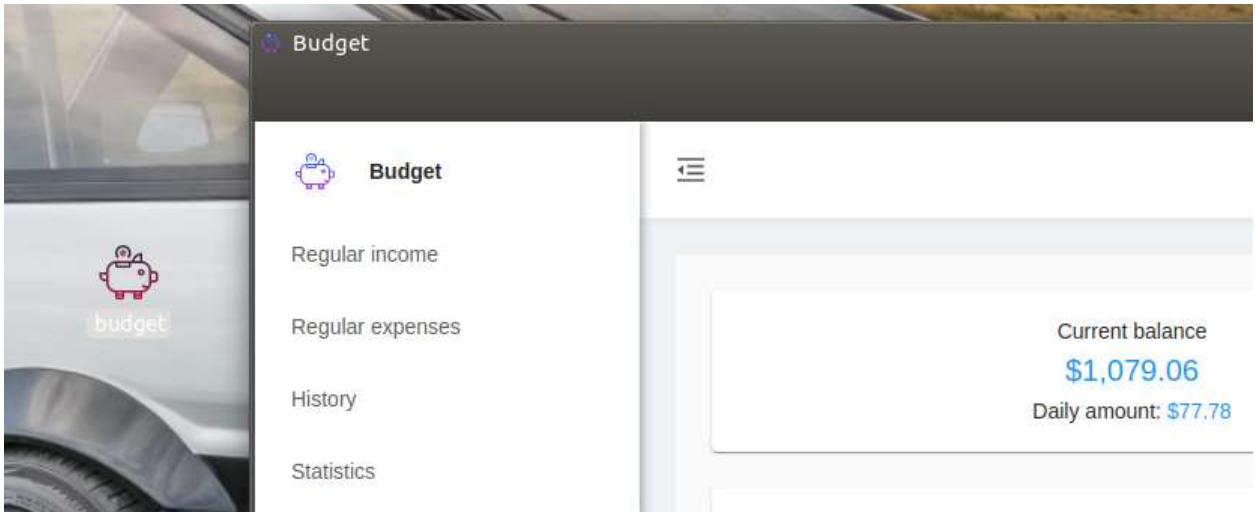

Рисунок 3.13 – Вигляд додатку запущеного з робочого столу

3.4 Висновок до частини 3

Реалізовано додаток згідно поставленого завдання і ТЗ проведення аналізу новітніх технологій розробки PWA та застосування цих технологій у створенні власного PWA. Представлений результат проведення підтримки PWA його можливості.

ВИСНОВКИ

Проведено аналіз цілей, принципів та переваг застосування PWA перед звичайними Web-додатками. Розглянуто різноманітні технології створення мобільних додатків. Проаналізовані найпопулярніші JavaScript-фреймворки та наведено їх позитивні та негативні якості.

Розглянуто різноманітні технології створення мобільних додатків. Проаналізовані найпопулярніші JavaScript-фреймворки, наведено їх позитивні та негативні якості.

Описано сферу застосування та головні функції додатку, архітектура додатку, стек технологій та інструментів, що використовувалися у процесі розробки для забезпечення гідного технічного рівня.

Додаток реалізований за допомогою Angular для клієнтського додатку та Cloud Firestore для БД. Також проведена інтеграція Web-додатку у PWA.

Створений PWA для ведення власного бюджету, який має змогу працювати на всіх настільних та мобільних пристроях, що підтримують сучасні браузери

Мета кваліфікаційної роботи досягнута. Вимоги виконані відповідно до поставленого на початку роботи технічного завдання.

ДОДАТОК А

ЗАТВЕРДЖЕНО
Зав. кафедри ІТП
_____ Прокопенко Т.О.
«____» _____ 2020 р.

Розробка PWA-застосунок для ведення власного бюджету

Специфікація
482 ЧДТУ 20846 – 01
Листів 2

Розробник	_____	Лохно О.В.
Керівник	_____	Рудницький С.В.
Н. контроль	_____	Колесніков К.В.

Н

Черкаси, 2020

Позначення	Найменування	Примітка
	Документація	
482 ЧДТУ 20846– 01 12 01	Текст програми	

Розробка PWA-застосунку для ведення власного бюджету

482 ЧДТУ 20846– 01 12 01

Текст програми

Листів 9

Розробник _____ Лохно О.В.

Н

Черкаси, 2020

Код app.module.ts:

```
import en from '@angular/common/locales/en';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule } from '@angular/common/http';
import { registerLocaleData } from '@angular/common';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';

import { ToastrModule } from 'ngx-toastr';
import { en_US, NZ_I18N } from 'ng-zorro-antd/i18n';

import { HttpRestService } from '@modules/http/http-rest.service';

import { AppComponent } from './app.component';
import { AppRoutingModule } from './app-routing.module';
import { IconsProviderModule } from './icons-provider.module';
import { NgxChartsModule } from '@swimlane/ngx-charts';
import { ServiceWorkerModule } from '@angular/service-worker';
import { environment } from '../environments/environment';

registerLocaleData(en);

@NgModule({
  imports: [
    FormsModule,
    BrowserModule,
    HttpClientModule,
    AppRoutingModule,
    IconsProviderModule,
    ToastrModule.forRoot(),
    BrowserAnimationsModule,
    NgxChartsModule,
    ServiceWorkerModule.register('ngsw-worker.js', { enabled: environment.production
})
  ],
  declarations: [ AppComponent ],
  providers: [
    HttpRestService,
    { provide: NZ_I18N, useValue: en_US }
  ],
```

```
  bootstrap: [ AppComponent ]
})
export class AppModule {}
```

Код шаблону portal.html:

```
<nz-layout
  class="app-layout">
  <nz-sider class="menu-sidebar"
            nzTheme="light"
            nzCollapsible
            nzWidth="256px"
            nzBreakpoint="md"
            [(nzCollapsed)]="isCollapsed"
            [nzTrigger]="null">
    <a routerLink="/dashboard">
      <div class="sidebar-logo">
        <img src="../../assets/icons8-money-box-64.png" alt="logo">
        <h1>Budget</h1>
      </div>
    </a>
    <ul nz-menu nzTheme="light" nzMode="inline" [nzInlineCollapsed]="isCollapsed">
      <li
        nz-menu-item
        nzMatchRouter>
        <a routerLink="/income">Regular income</a>
      </li>
      <li
        nz-menu-item
        nzMatchRouter>
        <a routerLink="/expenses">Regular expenses</a>
      </li>
      <li
        nz-menu-item
        nzMatchRouter>
        <a routerLink="/history">History</a>
      </li>
      <li
        nz-menu-item
        nzMatchRouter>
        <a routerLink="/statistics">Statistics</a>
```

```
    </li>
   </ul>
  </nz-sider>
  <nz-layout>
   <nz-header>
    <div class="app-header">
     <span class="header-trigger" (click)="isCollapsed = !isCollapsed">
       <i class="trigger"
         nz-icon
         [nzType]="isCollapsed ? 'menu-unfold' : 'menu-fold'"
       ></i>
     </span>
    </div>
   </nz-header>
   <nz-content>
    <router-outlet></router-outlet>
   </nz-content>
   <nz-footer>
    <a target="_blank" href="https://icons8.com/icons/set/money-box">Money Box
icon</a> icon by <a target="_blank"

href="https://icons8.com">Icons8</a>
   </nz-footer>
  </nz-layout>
</nz-layout>
```

Код файлу dashboard.html:

```
<div class="inner-content">
  <div class="card-row">
   <mat-card class="card">
    <div class="row-a">
     Current balance
    </div>
    <div class="row-a">
     <span class="title current">{{current | currency: 'CAD':'symbol-narrow': '1.2-
2'}}</span>
    </div>
    <div class="daily row-a">
     Daily amount: <span class="title">{{dailyAmount | currency: 'CAD':'symbol-
narrow': '1.2-2'}}</span>
    </div>
```

```
  </mat-card>
 </div>
 <div class="card-row">
  <mat-card class="card chart-card">
   <div class="chart">
    <ngx-charts-bar-vertical
     [results]="results"
     [gradient]="false"
     [xAxis]="true"
     [legend]="false">
    </ngx-charts-bar-vertical>
   </div>
  </mat-card>
 </div>
 <div class="card-row">
  <mat-card class="card actions">
   <button class="cta">
    <span class="button-text">Add expense</span>
   </button>
   <button class="cta">
    <span class="button-text">Add income</span>
   </button>
  </mat-card>
 </div>
</div>
```

Код файлу expenses.html:

```
<app-breadcrumbs-page>
 Regular Expenses
</app-breadcrumbs-page>
<div class="inner-content">
 <app-regular-total [data]="expenses"></app-regular-total>
 <cdk-virtual-scroll-viewport itemSize="73" class="demo-infinite-container">
  <div class="list">
   <app-card-list [data]="expenses"></app-card-list>
  </div>
 </cdk-virtual-scroll-viewport>
</div>
```

Код файлу statistics.html:

```
<app-breadcrumbs-page>
  Statistics
</app-breadcrumbs-page>
<div class="inner-content">
  <mat-tab-group mat-align-tabs="start">
    <mat-tab label="Expenses">
      <cdk-virtual-scroll-viewport itemSize="73" class="demo-infinite-container">
        <div class="card-row">
          <mat-card class="card">
            Total {{calculateTotalExpenses() | currency: 'CAD':'symbol-narrow': '1.2-2'}}
          </mat-card>
        </div>
        <div class="card-row">
          <mat-card class="card chart-card">
            <div class="chart">
              <ngx-charts-bar-vertical
                [results]="expenses"
                [gradient]="false"
                [xAxis]="true"
                [yAxis]="true"
                [legend]="false">
              </ngx-charts-bar-vertical>
            </div>
          </mat-card>
        </div>
        <div class="card-row">
          <mat-card class="card chart-card">
            <div class="chart chart-pie">
              <ngx-charts-pie-chart
                [results]="expenses"
                [gradient]="false"
                [legend]="true"
                [legendPosition]="'right'"
                [labels]="true"
                [doughnut]="false"
              >
              </ngx-charts-pie-chart>
            </div>
          </mat-card>
        </div>
```

```
    </cdk-virtual-scroll-viewport>
  </mat-tab>
  <mat-tab label="Income">
    <cdk-virtual-scroll-viewport itemSize="73" class="demo-infinite-container">
      <div class="card-row">
        <mat-card class="card">
          Total {{calculateTotalIncome() | currency: 'CAD':'symbol-narrow': '1.2-2'}}
        </mat-card>
      </div>
      <div class="card-row">
        <mat-card class="card chart-card">
          <div class="chart">
            <ngx-charts-bar-vertical
              [results]="income"
              [gradient]="false"
              [xAxis]="true"
              [yAxis]="true"
              [legend]="false">
            </ngx-charts-bar-vertical>
          </div>
        </mat-card>
      </div>
      <div class="card-row">
        <mat-card class="card chart-card">
          <div class="chart chart-pie">
            <ngx-charts-pie-chart
              [results]="income"
              [gradient]="false"
              [legend]="true"
              [legendPosition]="'right'"
              [labels]="true"
              [doughnut]="false">
            </ngx-charts-pie-chart>
          </div>
        </mat-card>
      </div>
    </cdk-virtual-scroll-viewport>
  </mat-tab>
 </mat-tab-group>
</div>
```

Код файлу total.service.ts:

```
import { Injectable } from '@angular/core';

import { RegularDto } from '@modules/common-dto/regular.dto';
import { RegularityType } from '@enums/regularity-type.enum';

@Injectable({
  providedIn: 'root'
})
export class TotalService {

  calculateDayTotal(values: RegularDto[]) {
    return this.calculateTotalByRegularity(values, RegularityType.aliasProperties.Daily);
  }
  calculateWeekTotal(values: RegularDto[]) {
    return this.calculateTotalByRegularity(values,
RegularityType.aliasProperties.Weekly);
  }
  calculateMonthTotal(values: RegularDto[]) {
    return this.calculateTotalByRegularity(values,
RegularityType.aliasProperties.Monthly);
  }
  calculateYearTotal(values: RegularDto[]) {
    return this.calculateTotalByRegularity(values,
RegularityType.aliasProperties.Yearly);
  }
  groupBy(values: RegularDto[], key: string) {
    return values.reduce((object, item) => {
      (object[ item[ key ] ] = object[ item[ key ] ] || []).push(item);
      return object;
    }, {});
  }
  calculateTotalNumbers(values: number[]) {
    return values.reduce((total, item) => total + item, 0);
  }
  private calculateTotalByRegularity(values: RegularDto[], multiplier: number): number
{
    const groupedValues = this.groupBy(values, 'regularity');
    const daily = Object.entries(groupedValues).map(([ key, group ]) =>
      this.calculateCurrency(group, RegularityType.getNumberFromType(key),
multiplier));
```

```
    return this.calculateTotalNumbers(daily);
  }

  private calculateCurrency(values, regularity, multiplier: number) {
    return values ? values.reduce((total, item) => total + item.amount, 0) /
Number(regularity) * multiplier : 0;
  }
}
```

Код файлу record.dto.ts:

```
import { RegularityType } from '@enums/regularity-type.enum';
import { Categories } from '@enums/category-type.enum';

export class RegularDto {
  name: string;
  amount: number;
  regularity?: RegularityType;
  category: Categories;

  constructor(data?: RegularDto) {
    if (data) {
      this.name = data.name;
      this.amount = data.amount;
      this.regularity = data.regularity || null;
      this.category = data.category;
    }
  }
}
```

Код файлу manifest.webmanifest:

```
{
  "name": "budget",
  "short_name": "budget",
  "theme_color": "#1976d1",
  "background_color": "#fafafa1",
  "display": "standalone",
  "scope": "./",
  "start_url": "./",
  "icons": [
    {
```

```
    "src": "assets/icons/icon-72x72.png",
    "sizes": "72x72",
    "type": "image/png",
    "purpose": "maskable any"
   }]
}
```

Код файлу service-worker.json:

```
{
 "$schema": "./node_modules/@angular/service-worker/config/schema.json",
 "index": "/index.html",
 "assetGroups": [
   {
    "name": "app",
    "installMode": "prefetch",
    "resources": {
     "files": [
       "/favicon.ico",
       "/index.html",
       "/manifest.webmanifest",
       "/*.css",
       "/*.js"
     ]
    }
   }, {
    "name": "assets",
    "installMode": "lazy",
    "updateMode": "prefetch",
    "resources": {
     "files": [
       "/assets/**",
       "/*.(eot|svg|cur|jpg|png|webp|gif|otf|ttf|woff|woff2|ani)"
     ]
    }
   }
 ]
}
```

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. M. Haverbeke, Eloquent JavaScript 2-d edition – No Starch Press , 2015. – 454 c.
2. S. Fulton, S. Fulton, HTML5 Canvas 2-d edition– O`Reilly, 2013. – 727c.
3. B. Lawson, R. Sharp, Introducing HTML5 2-d edition – New Riders, 2012. – 285c.
4. Bernd Jähne, Digital Image Processing – Springer, 2015. – 585c.
5. J. Cranford Teague, CSS3: Visual QuickStart Guide – Pitchpit Press, 2014. – 317c.
6. C. Cook, J. Garber, Foundation HTML5 with CSS3 – Print Press, 2015. – 417c.
7. D. McFarland, CSS3: The Missing Manual – Alpha Press, 2016. – 561c.
8. D. Crockford, JavaScript: The Good Parts: – O`Reilly, 2013. – 149c.
9. D. Flanagan, JavaScript: The Definitive Guide: – O`Reilly, 2011. – 857.c
10. D. Herman, Effective JavaScript: 68 Specific Ways to Harness the Power of JavaScript: – Addison Wesley, 2012. – 209c.
11. Основы JavaScript / learn.javascript.ru [Електронний ресурс] – 2020. – Режим доступу до ресурсу: https:// learn.javascript.ru.
12. JavaScript Online Training / tutorialspoint.com [Електронний ресурс] – 2020. – Режим доступу до ресурсу: https://tutorialspoint.com.
13. Angular Docs / angular.io [Електронний ресурс] – 2020. – Режим доступу до ресурсу: https://angular.io/docs.
14. React Docs / react.js.org [Електронний ресурс] – 2020. – Режим доступу до ресурсу: https://ru.reactjs.org/docs/getting-started.html.
15. Vue.js Docs / vuejs.org [Електронний ресурс] – 2020. – Режим доступу до ресурсу: https://vuejs.org/v2/guide/.
16. Angular Medium / angular.medium.com [Електронний ресурс] – 2020. – Режим доступу до ресурсу: https://blog.angular.io/.
17. PWA / developers.google.com [Електронний ресурс] – 2020. – Режим доступу до ресурсу: https://developers.google.com/web/ilt/pwa.

18. Cloud Firestore / firestore.google.com [Електронний ресурс] – 2020. – Режим доступу до ресурсу: https://firebase.google.com/docs/firestore.
19. WEB Storm: The Java IDE for Professional Developers by JetBrains / jetbrains.com [Електронний ресурс] – 2020. – Режим доступу до ресурсу: https://www.jetbrains.com/webstorm/.
20. Mint.com / mint.com [Електронний ресурс] – 2020. – Режим доступу до ресурсу: https://www.mint.com/how-mint-works.
21. Moneymanagerex.org / moneymanagerex.org [Електронний ресурс] – 2020. – Режим доступу до ресурсу: https://www.moneymanagerex.org/features/overview.
22. BudgetTracker.com / budgetracker.com [Електронний ресурс] – 2020. – Режим доступу до ресурсу: https://secure.budgettracker.com/.
23. Розроблення прогресивного Web-додатку Лохно О.В. – Збірник тез доповідей студентської науково-практичної конференції ЧДТУ 27-30 квітня 2020 р.
24. Microsoft SQL Server Management Studio / docs.microsoft.com [Електронний ресурс] – 2020. – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver15.
25. RxJS library / rxjs-dev.firebaseapp.com [Електронний ресурс] – 2020. – Режим доступу до ресурсу: https://rxjs-dev.firebaseapp.com/guide/overview.

ПРОГРАМНІ ЗАСОБИ

1. Office 365 для освіти © Корпорація Майкрософт, 1983-2020.
2. JetBrains Webstorm 2019.2.3 x64 © JetBrains, 2017-2020.
3. Adobe Photoshop CC6 x64 © Adobe, 2012-2020.

Copy protected with Online-PDF-No-Copy.com