Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра програмних систем і технологій

УДК 004.912 *На правах рукопису*

ВИПУСКНА КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

Тема: “ Веб-додаток персонального планування задач на основі архітектури Progressive Web Application ”

Спеціальність – 121 “Інженерія програмного забезпечення”

ПОЯСНЮВАЛЬНА ЗАПИСКА

БР.ІПЗ – 30.00.00.000

Студент
ІПЗ-41_____/Кирило БОВСУНОВСЬКИЙ/

Науковий керівник
д.т.н.,с.н.с._____/Геннадій ПОРЄВ/

Консультант
з питань нормоконтролю
фахівець_____/Тамара ЧАПОВСЬКА/

Допускається до захисту
Завідувач кафедри
д.т.н.,проф. _____ /Олексій БИЧКОВ/

Київ – 2021

Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра програмних систем і технологій
Освітньо-кваліфікаційний рівень бакалавр
Спеціальність 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖЕНО
Зав. кафедри програмних систем і технологій
_____ (Олексій БИЧКОВ)
(підпис) (прізвище та ініціали)

ЗАВДАННЯ
НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ
РОБОТУ СТУДЕНТУ

Бовсуновському Кирилу Олексійовичу

1. Тема бакалаврської роботи “ Веб-додаток персонального планування задач на основі архітектури Progressive Web Application” керівник проекту (роботи) Порєв Геннадій Володимирович
затверджені наказом вищого навчального закладу від “ 11 ” листопада 2020 р. № 6

2. Строк подання студентом роботи 30 травня 2021 р.

3. Вихідні дані до проекту (роботи) Теоретичні концепції побудови та функціонування інформаційних та програмних технологій направлених на побудову клієнт-серверного веб-додатку на базі технології PWA.

4. Зміст розрахунково - пояснювальної записки(перелік питань, які потрібно розробити)
1. Сучасні програми-конкуренти
2. Дослідження існуючих архітектур
3. Проектування та реалізація застосунку
4. Тестування працездатності застосунку

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
1. Клієнт-серверна модель (рис. 1.3, ст. 18)
2. Діаграма архітектури веб-застосунку(рис. 2.3, ст. 27)

3. Візуалізація роботи PWA(рис. 2.9, ст. 36)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Розділ 1	Порєв Г.В	07.03.2021	12.04.2021
Розділ 2	Порєв Г.В.	13.04.2021	14.04.2021
Розділ 3	Порєв Г.В.	18.04.2021	23.04.2021
Розділ 4	Порєв Г.В.	25.04.2021	27.05.2021

7. **Дата видачі завдання** 09 жовтня 2020 р.
Керівник _____ (Геннадій ПОРЄВ)
Завдання прийняв до виконання _____ (Кирило БОВСУНОВСЬКИЙ)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Формулювання мети та переліку конкретних завдань ДР	28.11.2020	виконано
2	Збір та аналіз інформаційних джерел за темою ДР	09.01.2021	виконано
3	Визначення програмних вимог та проектування застосунку	10.02.2021	виконано
4	Створення серверної частини проекту	20.03.2021	виконано
5	Розробка клієнтської частини проекту	25.03.2021	виконано
6	Тестування роботи розробленого веб-додатку	05.04.2021	виконано
7	Оформлення пояснювальної записки та презентації	01.05.2021	виконано
8	Затвердження пояснювальної записки роботи завідувачем кафедри.	29.05.2021	виконано

Студент – бакалавр _____(Кирило БОВСУНОВСЬКИЙ)
Керівник роботи _____(Геннадій ПОРЄВ)

АНОТАЦІЯ

Випускна кваліфікаційна бакалаврська робота: 71 с., 30 рис., 1 табл., 1 додат., 25 джерел.

Тема: Веб-додаток персонального планування задач на основі архітектури Progressive Web Application

Об'єкт дослідження: використання технології Progressive Web Application для реалізації веб-додатків.

Мета роботи: розробка веб-додатку, за допомогою якого користувач може завчасно планувати свій день і регулювати кількість витраченого на свої потреби часу.

Предмет дослідження: технологія Progressive Web Application, досліджується можливість і доцільність її використання для реалізації ефективних веб-додатків.

Результати дослідження: досліджено можливість і доцільність використання нової, ще погано вивченої, технології Progressive Web Application для створення сучасних веб-додатків.

Висновок

В результаті виконаної роботи було отримано веб-додаток на базі технології PWA, який дозволяє користувачам заздалегідь планувати свій день. Розроблений веб-додаток працює на будь-якій ОС та не потребує встановлення на мобільні пристрої чи персональні комп'ютери, що робить його гарною альтернативою мобільним додаткам. Є можливість вдосконалити додаток для повноцінної роботи в режимі відсутності доступу до мережі інтернет, з використанням протоколу передачі даних HTTPS.

ПРОГРЕСИВНИЙ ВЕБ-ДОДАТОК, PROGRESSIVE WEB APPLICATION, PWA, РОЗРОБКА ВЕБ-ЗАСТОСУНКУ, REACTJS, ВЕБ-ДОДАТОК, NODEJS, БАЗА ДАНИХ, MONGODB, JAVASCRIPT, CLIENT-SERVER

АННОТАЦИЯ

Выпускная квалификационная бакалаврская работа: 71 с., 30 рис., 1 табл., 1 додат., 25 источников.

Тема: Веб-приложение персонального планирования задач на основе архитектуры Progressive Web Application

Объект исследования: использование технологии Progressive Web Application для реализации веб-приложений.

Цель работы: разработка веб-приложения, с помощью которого пользователь может заблаговременно планировать свой день и регулировать количество затраченного на свои нужды времени.

Предмет исследования: технология Progressive Web Application, исследуется возможность и целесообразность ее использования для реализации эффективных веб-приложений.

Результаты исследования: исследована возможность и целесообразность использование новой, ещё плохо изученной, технологии Progressive Web Application для создания современных веб-приложений.

Вывод

В результате выполненой работы было получено веб-приложение на базе технологии PWA, которий позволяет пользователям заранее планировать свой день. Разработанное приложение работает на любой ОС и не требует установки на мобильные устройства, что делает его хорошей альтернативой мобильным приложениям. Есть возможность усовершенствовать приложение для полноценной работы в режиме отсутствия доступа к сети интернет, с использованием защищенного протокола передачи данных HTTPS.

ПРОГРЕССИВНОЕ ВЕБ-ПРИЛОЖЕНИЕ, PROGRESSIVE WEB APPLICATION, PWA, РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЯ, REACTJS, NODEJS, БАЗА ДАННЫХ, MONGODB, JAVASCRIPT, CLIENT-SERVER

ABSTRACT

The bachelor's thesis contains 71 pages, 30 images, 1 table, 1 application , 25 sources of information.

Subject: Personal task scheduling web application based on the Progressive Web Application architecture

Research object: use of Progressive Web Application technology to create web applications.

Purpose of work: Development of a web application with which the user can plan his day in advance and regulate the amount of time spent on his needs.

Subject of research: Progressive Web Application technology, the possibility and feasibility of its use for the implementation of efficient web applications is being investigated.

Research results: the possibility and feasibility of using a new, still poorly studied, Progressive Web Application technology for creating modern web applications was investigated.

Conclusion

As a result of the work performed, a web application based on PWA technology was obtained, which allows users to plan their day in advance. The developed web application runs on any operating system and does not require installation on mobile devices, which makes it a good alternative to mobile applications. It is possible to improve the application for full functionality in the absence of Internet access, using a secure HTTPS data transfer protocol.

PROGRESSIVE WEB APPLICATION, PWA, WEB-DEVELOPMENT, REACTJS, NODEJS, DATABASE, MONGODB, JAVASCRIPT, CLIENT-SERVER

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – База Даних

ОС – Операційна Система

ПЗ – Програмне Забезпечення

ПК – Персональний Комп’ютер

СКБД – Система Керування Базою Даних

ACID – Atomicity Consistency Isolation Durability

API – Application Programming Interface

CRUD – Create Read Update and Delete

CSS – Cascading Style Sheets

DOM – Document Object Model

HTML – HyperText Markup Language

HTTPS – HyperText Transfer Protocol Secure

HTTP – HyperText Transfer Protocol

MPA – Multi-Page Application

NFC – Near Field Communication

NoSQL – Non-Structured Query Language

NPM – Node Package Manager

PWA – Progressive Web Application

SPA – Single-Page Application

SQL – Structured Query Language

UML – Unified Modeling Language

URL – Uniform Resource Locator

XML – Extensible Markup Language

ВСТУП

Люди, що знаходяться у вічній погоні за кар’єрою та статусом зазвичай не здатні знайти часу на себе, що призводить до проблем зі здоров’ям та хронічної втоми. На допомогу часто приходять додатки по типу персонального планування задач, які здатні допомогти у питаннях відсутності або неспланованості часу. Люди, котрі побачили перспективу у таких рішеннях одразу почаии користування застосунками для планування свого вільного часу, що зіштовхнуло їх з різного роду проблемами при користуванні застосунками.

Головною з таких проблем став погано продуманий й не зручний користувацький інтерфейс, що позбавив користувачів будь-якого задоволення при користуванні продуктом. З цього виплаває те, що програма, яка спочатку здавалася ідеальним рішенням проблеми з часом, ніяк не змінила ситуацію й не вирішила проблему браку часу.

Іншим недоліком застосунків-планувальників була проблема швидкодії, через яку застосунки не могли нормально працювати й відповідати належному рівню надання сервісів та послуг.

Третім каменем спотикання стало те, що велика кількість користувачів не мала постійного зв’єднання з мережею інтернет та використовувала старі мобільні пристрої, на яких постійно бракувало вільної пам’яті та не виставало потужності.

Вище згадані проблеми давно, належним чином, не вирішувались, доки відносно нещодавно розробникам програм не було представлено нове архітектурне рішення, щодо виробництва веб-застосунків нового покоління. Цим рішенням стала програмна архітектура PWA, або іншими словами Прогресивний Веб Застосунок.

В порівнянні з архітектурними рішеннями минулого покоління PWA отримало вагомі переваги, серед яких є:

- можливість працювати в умовах відсутньої інтернет мережі, за допомогою кешування;
- легковажність застосунку, яка дозволяє миттєво встановлювати застосунок на будь-який мобільний пристрій без потреби у завантаженні додатку;
- використання протоколу захищеної передачі даних HTTPS;
- відсутність необхідності розробляти встановлювану програму для кожної ОС.

Дана технологія є погано розвиненою в Україні, що додає доцільності для дослідження можливостей використання технології на практиці. Технологія має широкий спектр використання, що робить її однією з найперспективніших у сфері створення веб-додатків. Технологія може стати однією з провідних серед фронтенд розробників веб-застосунків.

Реалізація PWA застосунку дасть можливість детально ознайомитися з її перевагами й недоліками на практиці й провести порівняльний аналіз з іншими архітектурними рішеннями.

Метою бакалаврської роботи є розробка веб-додатку, за допомогою якого користувач має змогу планувати свій день та регулювати кількість витраченого на свої потреби часу.

Досягнення мети роботи включає в себе вирішення наступних **задач**:

- аналіз сучасних програм-конкурентів;
- порівняльний аналіз нереляційного та реляційного підходів до організації сховища збереження даних;
- огляд загальної класифікації архітектур веб-застосунків;
- порівняльний аналіз актуальних архітектур побудови клієнтської частини веб-застосунків й у тому числі PWA;

- проектування застосунку;
- розробку веб-застосунку на базі PWA;
- ручне та автоматизоване види тестування й юніт-тестування дрібних частин додатку.

Предметом дослідження є технологія PWA, досліджується можливість та доцільність її використання для реалізації швидкодійних та ефективних веб-застосунків.

РОЗДІЛ 1
СУЧАСНІ ПРОГРАМИ-КОНКУРЕНТИ

Існує безліч різноманітних програм для планування своїх задач та цілей, котрі надають користувачеві спроможність записувати та структурувати свої справи, ідеї та прогрес виконання справи. Планувальник здатний підвищити ефективність і продуктивність роботи користувача. Планувальники задач безперервно вдосконалюються та розширяють свій функіконал.

Мною був проведен ретельний розбір найпопулярніших існуючих веб-застосунків, котрі надають змогу організовувати і планувати користувачеві розпорядок дня. Існує безліч планувальників відомих компаній-гігантів та додатки маленьких компаній. Проаналізувши існуючі застосунки, можна сказати що серед них є як дуже зручні для користування додатки, так і ті, які вимагають удосконалення.

1.1. Застосунок «RememberTheMilk»

RememberTheMilk – це розробка невеликої одноіменної фірми. На початку користувач опиняється на сторінці реєстрації, привертає увагу задній фон цього вікна, котрий кожні дві секунди блимає різними кольорами радуги[1]. Це може сподобатися зовсім не кожному користувачеві.

На сайті «RememberTheMilk» присутній приємний та унікальний помічник. Головна сторінка може показатися не зовсім зручною та простою, але тут приходить на допомогу помічник завдяки котрому користувач швидко адаптується і створення нових завдання не викликає ніяких труднощів. Візуально застосунок виглядає добре, але трохи гірший, аніж у конкурентних застосунків, котрі були проаналізовані в даній роботі. Присутньо багато проблем при користуванні застосунку через мобільний пристрій.

На рисунку 1.1 зображено головну сторінку застосунку «RememberTheMilk».

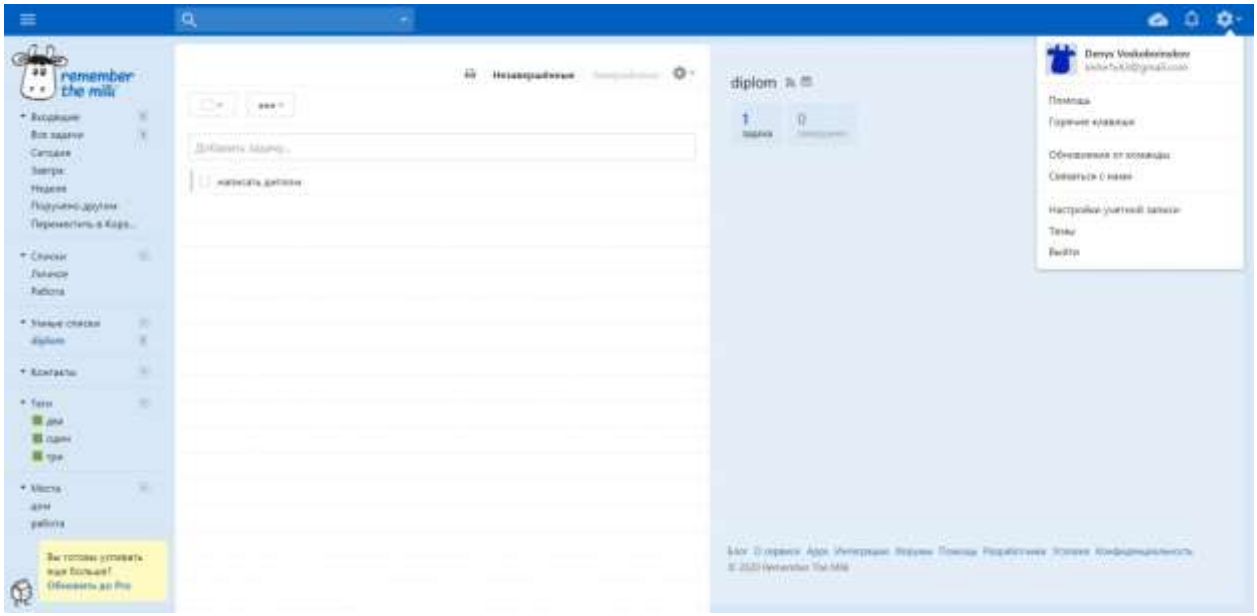

Рис. 1.1. Головна сторінка застосунку «RememberTheMilk»

На мою думку перевагою в даного сервісу - це система делегування, при роботі у великій команді буде дуже корисною. Користувач має можливість створювати задачі, списки задач та ділитися їми з іншими людьми, які зареєстровані в системі та входять у список контактів.

У данному веб-застосунку присутньо достатньо цікавих функцій, наприклад, персональна кастомізація, сортування, нагадувань, пов’язання з «Dropbox», «Google Drive» або «Microsoft Outlook». Але отримати ці функції можливо тільки придбавши повну версію.

1.2. Застосунок «TodoList»

ToDoList відносно безкоштовний сервіс для планування своїх задач. Пристуні незначні відмінності від попереднього представника. Застосунок повністю локалізований. Присутньо понад десяти різноманітних мов. Зовнішнє оформлення сайту значно привабливіше. Повністю адаптований. Повна пітримка мобільних, відсутні горизонтальні скроли.

Головна сторінка застосунку «TodoList» зображена на рисунку 1.2.

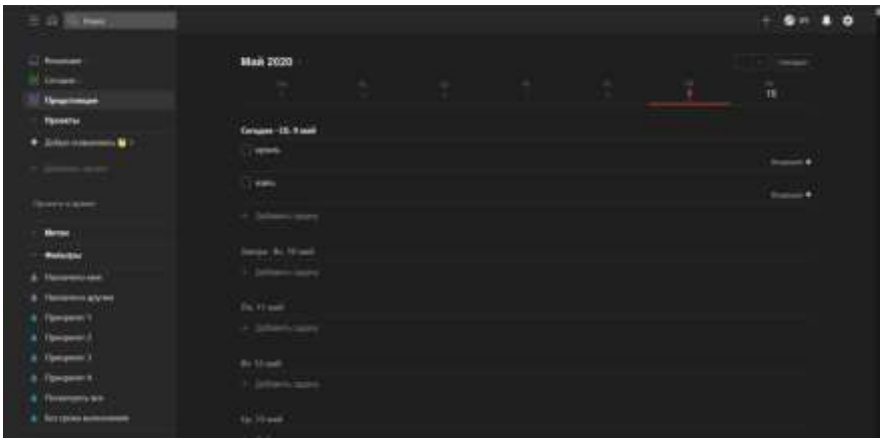

Рис. 1.2. Головна сторінка застосунку «TodoList»

Веб-застосунок має значно ширший функціонал відносно його попередника. Водночас, це і перевага і недолік. Найбільш зазнала головна сторінка. Присутньо безліч елементів навігації, це беспосередньо ускладнює користування сайтом. На мою думку управління задачами створенно не найкращим чином . А саме не є зручним в користуванні. В застосунку присутньо величезна кількість налаштувань, котра нараховує декілька рівнів вкладеності. Це робить додаток складнішим для використання.

Цікава функція у «TodoList» - це використання психологічних мотиваторів для виконання своїх завдань або цілей – система «карми»[2]. Після того як користувач досягає успіху в завданнях, він отримує поінти та новий рівень. Карма підвищується, коли користувач додає нові завдання, вчасно їх виконує, використовує присунуті функції застосунку: мітки, що повторюються, дедлайни, нагадування. Карма втрачається коли люда просрочує свої завдання або цілі.

Дана особливість веб-застосунку показана на рисунку 1.3.

Рис. 1.3. Степені карти користувачів у застосунку «TodoList»

1.3. Застосунок «Any.do»

Це представник десктопного застосунку[3]. Присутній простий, лаконічний та комфортний інтерфейс. Стартова сторінка не має нічого зайвого та інтуїтивно зрозуміла кожному . На мою думку, негативна сторона цього застасунку є вкладка налаштувань, рядовому користувачеві буде проблематично розібратися.

Веб-застосунок включив в себе ненайкращий UX, отже увесь інтерфейс додатку не вміщується на повний монітор персонального комп’ютера. Присутня прокрутка по горизонталі. Сайт не оптимізованний для мобільних пристроїв.

В «Anydo» класична модель монетизації. Базовий функціонал безкоштовший, але для певний функцій потрібна повна версія продутку. Я вважаю введення обмежень на планувальник задач неприпустимо.

Серед зручних і корисних можливостей сайта можна виділити якісну та зручну систему групування задач. Також є сортування по алфавіту та часу виконання. Присутня класична можливість входу до системи за допомогою Google аккаунту, також присутній зручний механізм, котрий надає можливість синхронізувати персональний аккаунт «Anydo» з календарем «Google», можливість імпорту задачі звідти. Для можливості кастомізувати головну сторінку потрібна платна версія.

На рисунку 1.4 зображена головна сторінка застосунку «Any.do».

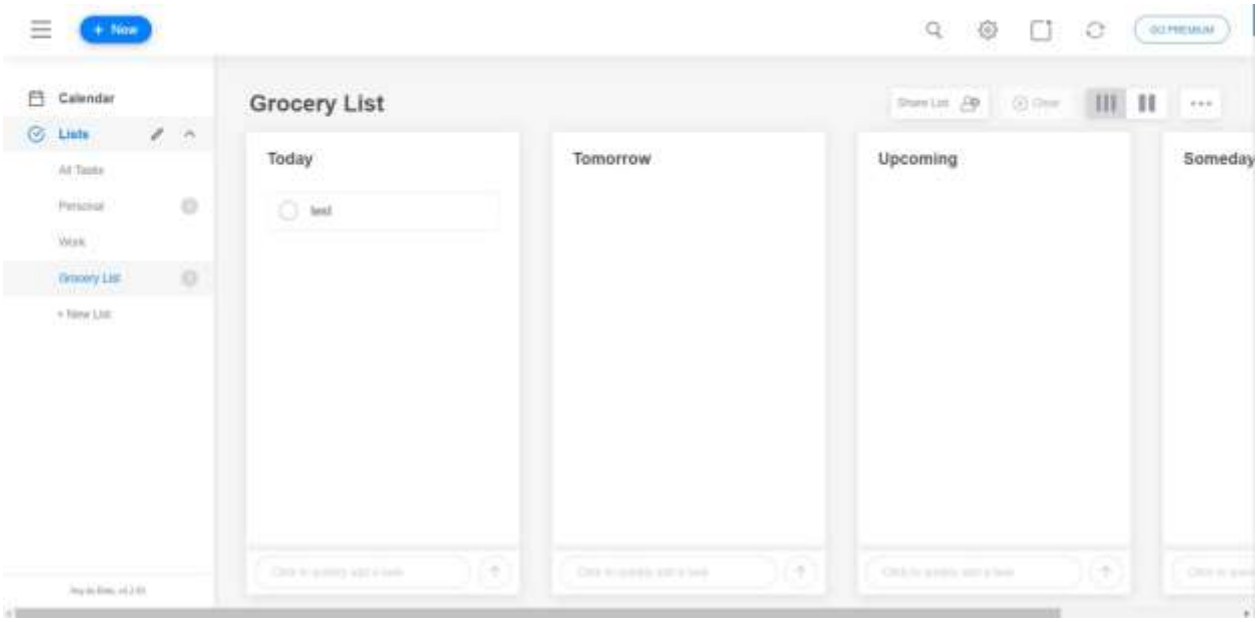

Рис. 1.4. Головна сторінка застосунку «Anydo»

1.4. Застосунок «To-do.Microsoft»

Головний конкурент компанії Google, компанія-гігант Microsoft випустила свій продукт під назвою «To-do.Microsoft»[4]. Повністю відсутня можливість реєстрації через аккаунт Google, можливість реєстрації через аккаунт Microsoft присутня.

Після аналізу інших програм, можу сказати, що «To-do.Microsoft» , на мою думку найкраща з представленних додатків. Перша причина, це проста,

лаконічна та інтуїтивно зрозуміла стартова сторінка. Не має жодних проблем при користуванні через мобільний пристрій, тому що сайт повністю адаптований.

Друга причина, нескладна система налаштувань, котра має с декілька зрозумілих чекбоксів. Присутня повна локалізація. Третя причина, веб-застосунок випускається повністю на безплатній основі. Можно зробити висновок, що всі корисні функції, які ми бачили в інших додатках, такі як система списків, система нагадувань, просунуте сортування та багато іншого доступно усім без потреби придбати повну версію продукту.

На рисунку 1.5 зображено головну сторінку застосунку «To-do.Microsoft».

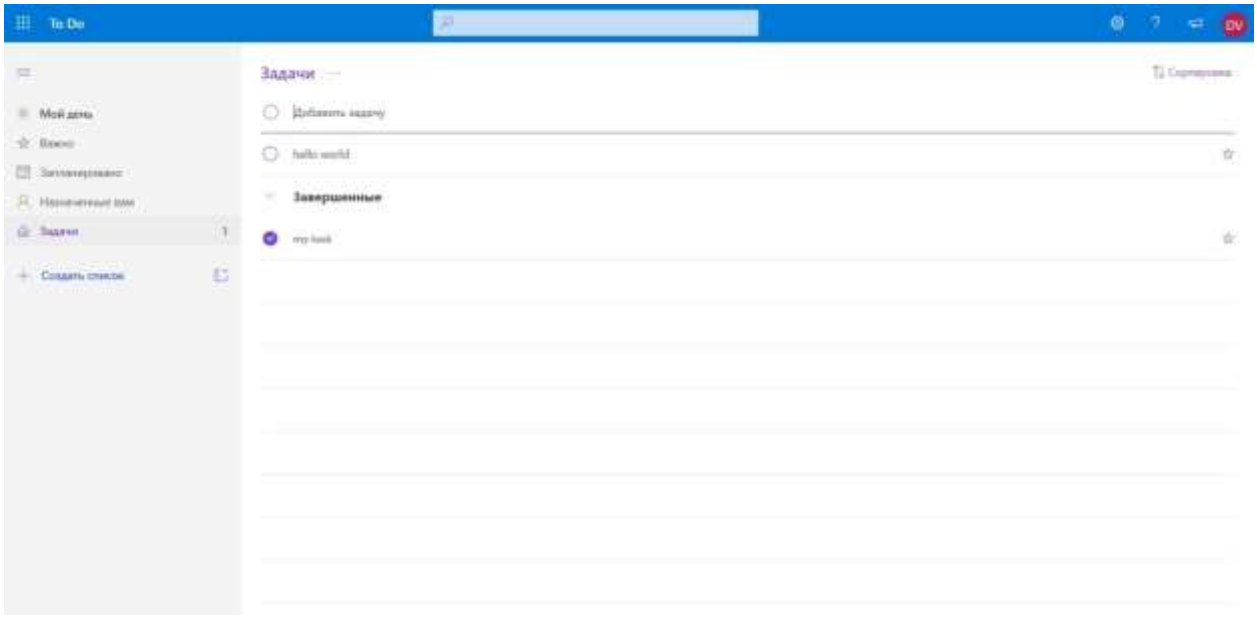

Рис. 1.5. Інтерфейс «To-do.Microsoft»

Але також присутні деякі незначні недоліки. Відсутня можливість спостерігати за системою прогресу, відсутність статистики, яка була у деяких попередніх рішеннях. Кастомізація відсутня. Немає нічного режиму.

1.5. Висновки з огляду існуючих програм-конкурентів

Зробивши ретельний розбір найпопулярніших існуючих програмних рішень, було зроблено такі висновки:

- у більшості веб-застосунків “Планувальник задач” перевантажений інтерфейс;
- велика кількість функцій стає доступною тільки після придбання повної версії;
- у багатьох планувальниках присутні проблеми з адаптацією під мобільні пристрої;
- схожа структура інтерфейсу у всіх планувальниках задач. У центральній частині екрану знаходяться замітки. У лівій частині та хедері знаходяться навігаційні елементи.

РОЗДІЛ 2
ДОСЛІДЖЕННЯ ІСНУЮЧИХ АРХІТЕКТУР

2.1. Огляд архітектурної моделі клієнт-сервер

Клієнт-серверна архітектура являє собою архітектуру комп'ютерної мережі, в основі якої лежить клієнт-серверна концепція, у якій віддалені процессори, запитують та отримують послуги від хост-машини[5]. Ця структура подрібнює повну функціональність застосунку на невеликі функції, після чого кожна така функція призначається певному компоненту або групі компонентів. Місце, у якому зберігаються ресурси називають сервером, а люди, котрі запитують сервіси або ресурси є клієнтами.

Клієнтські комп'ютери предоставляють собою інтерфейс, котрий дозволяє користувачу запитувати послуги від сервера й відображати результати, що були повернені сервером. Сервери чекають на запити від клієнтів, а потім відповідають на них. В ідеалі, сервер надає клієнтам стандартизований інтерфейс, щоб клієнтам не потрібно було знати особливості системи, котра надає послуги, тобто особливості апаратного й програмного забезпечення.

Клієнти часто знаходяться на робочих станціях або на ПК, у той час як сервери розташовуються в іншому місці мережі, на більш потужних машинах. Наприклад, при обробці даних лікарні, клієнтський комп'ютер здатний запускати програму для вводу інформації про пацієнта, у той час як сервер виконує іншу програму, котра керує базою даних, яка працює з інформацією. Велика кількість клієнтів може одночасно отримувати доступ до даних сервера, й у той же час клієнтські комп'ютери здатні займатися іншими задачами, тому велика кількість різних компаній, котрі працюють в мережі, використовують клієнт-серверну технологію.

Дана обчислювальна модель є особливо ефективною тоді, коли у кожного клієнта та сервера маються різні цілі й задачі, які вони вдало виконують. Зазвичай у застосунку реалізуються три головні функції:

збереження даних, обробка таких даних та відображення цих даних. У складних архітектурних рішеннях такі функції можуть бути розподілені між спеціальними виділеними ПК. Такими можуть бути клієнтський, проміжний й серверний комп'ютери. Серверний – береже, проміжний – обробляє, а клієнтський комп'ютер – відображає дані або представляє дані.

Оскільки і клієнтські і серверні комп'ютери вважаються незалежними пристроями, клієнт-серверна модель сильно відрізняється від старої моделі, моделі мейнфрейму, у якої централізований мейнфрейм виконував усі задачі для пов'язаних з ним мовчазних терміналів, котрі просто обмінювалися даними з головним мейнфреймом[6].

На рисунку 2.1 можна ознайомитися з загальним видом клієнт-серверної моделі архітектури.

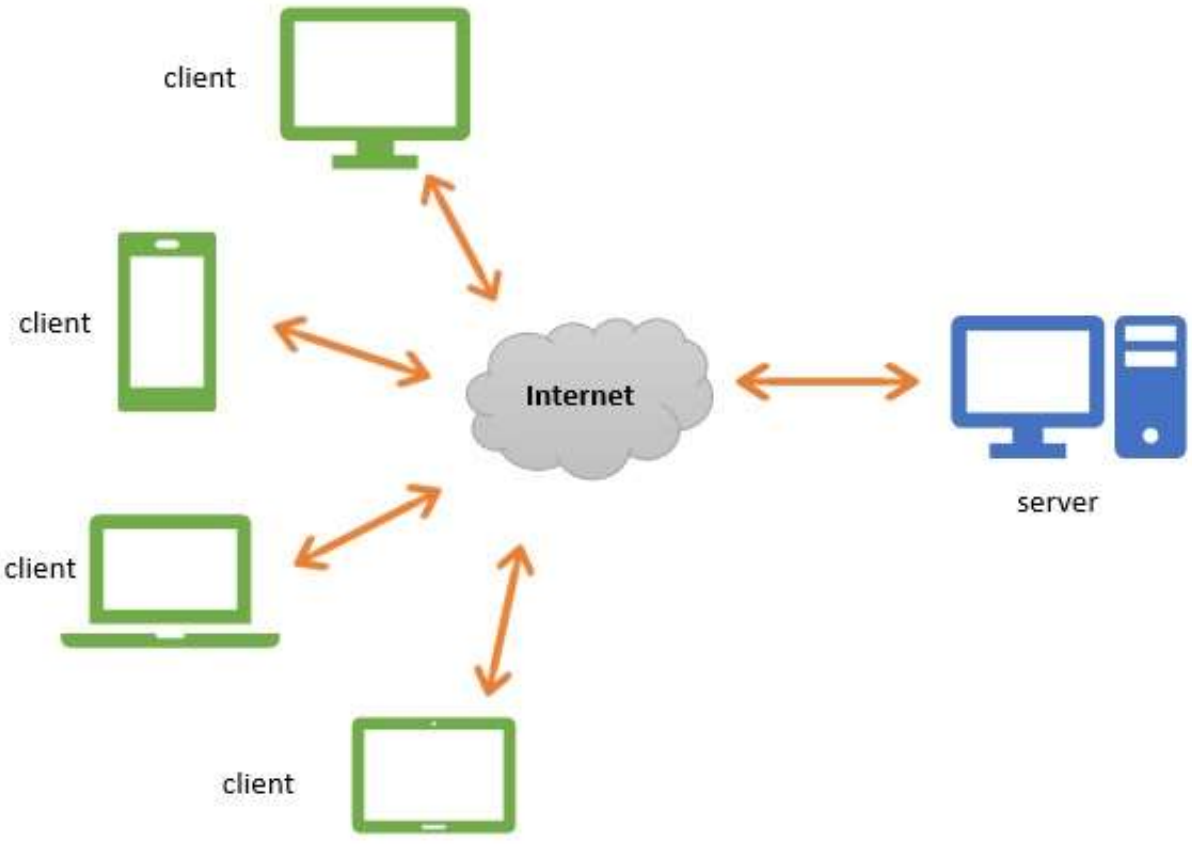

Рис. 2.1. Клієнт-серверна модель архітектури

2.1.1. Характеристики клієнт-серверної архітектури

У мережевій клієнт-серверній архітектурі користувачі, або іншими словами клієнти, та сервери, котрі надають доступ до сервісів та послуг застосунку, являють собою відокремлені логічні елементи, що здатні спілкуватися, здійснюючи обмін запитами, для вирішення певних визначених задач[7].

Загальними характеристиками клієнт-серверної архітектури веб-застосунку є:

- асиметричні протоколи. Між клієнтами та сервером існує відношення «багато до одного». Клієнт завжди розпочинає «розмову», коли запитує сервер про послугу чи сервіс, в той час коли сервер завжди працює, просто чекаючи на запит від клієнта;
- масштабованість. Існує можливість масштабувати клієнт-серверні системи як по вертикалі, так і по горизонталі. Масштабування по вертикалі являє собою додавання нових серверних машин або додавання більш потужної серверної машину. Горизонтальне ж, у свою чергу, являється додаванням або вилученням клієнтських станцій, під час чого відзначається невелике погіршення працездатності;
- модульна розширювана система. Модульна система архітектури клієнт-сервер дозволяє застосунку бути відмовостійким. У такому разі, програма має змогу автоматично й самостійно додавати деяку кількість серверів або служб, реагуючи на навантаження системи. У такій системі будь-які перебої можуть відбуватися без запинки роботи усього застосунку загалом. Коли один або декілька серверів виходять з ладу, система може продовжити свою роботу за умови наявності таких самих послуг на ще працюючих серверах;
- інкапсуляція послуг. Сервер визначає, як буде виконуватися робота, коли отримує повідомлення, у якому запитується послуга. Якщо

інтерфейс викладених повідомлень, який використовують і клієнт і сервер є незмінним, то існує змога оновити сервер, не торкаючись клієнта;

- спільні ресурси. Сервер має змогу налагоджувати одночасний доступ багатьох клієнтів до спільних ресурсів та надавати їм сервіси та послуги;
- платформна незалежність. Незалежність від платформи та ПЗ дозволяє поєднувати сервер та клієнтські платформи. На різних ОС можуть бути розроблені клієнти та сервери, що дозволяє оптимізувати роботу системи;
- цілісність. Серверні дані й інформація обслуговуються дешевше та захищеніше, через централізоване обслуговування. Також централізація сприяє захисту цілісності клієнт-серверних даних;
- розподіл клієнт-серверної функціональності. Клієнт та сервер забезпечують точний розподіл функцій. Сервер являється поставщиком послуг та сервісів, а клієнт – споживачем;
- спілкування шляхом відправки повідомлень. Оскількі сервер та клієнт являють собою слабко пов`язані процеси, спілкуватися вони можуть шляхом обміну запитами на доступ до сервісів та послуг;
- прозорість місцеположення. Сервер має змогу змогу перебувати на тій же самій машині, на якій перебуває клієнт або знаходитися на окремій машині. Клієнт-серверне ПЗ ховає місце розташування сервера від клієнтів, шляхом переадресація запитів на обслуговування;
- використовувати службові застосунки можна на декількох серверах.

2.1.2. Переваги та недоліки клієнт-серверної архітектури

Переваги й недоліки притаманні будь-якому типу архітектури. Клієнт-серверну архітектуру також не оминула ця доля. Далі будуть представлені плюси й мінуси архітектури клієнт-сервер[8].

Переваги клієнт-серверної архітектури:

- інформація, тобто системні дані, зберігається централізовано й у купі;
- модель являється досить ефективною у справах доставки ресурсів клієнту й потребує малих фінансових витрат на підтримку й обслуговування;
- архітектурою досить легко керувати й дані, тобто ресурси, можуть бути просто й швидко відправлені сервером та доставлені клієнту.
- система являється досить безпечною й забезпечує гарну безпеку даних, тому що дані є централізованими;
- якщо потрібно забезпечити більш кращу продуктивність й гнучкість, у сервер є змога вбудувати більшу кількість клієнтів чи серверів.

Недоліки клієнт-серверної архітектури:

- якщо на сервері були реалізовані й активовані різні роду шкідливі файли, то клієнти дуже легко можуть їх отримати;
- дані можуть бути підроблені під час передачі, тому має бути реалізовано додатковий захист, якщо є така можливість;
- коли сервер з якоїсь причини не працює, клієнт не має з’єднання та змоги отримати доступ до даних, що являє собою досить вагому проблему.

2.2. Нереляційні NoSQL та реляційні SQL бази даних

На сьогоднішній день на ринку інтернет послуг в наявності є велика кількість різних баз даних, з цієї причини обрати з них одну є дуже важкою задачею. З цієї причини необхідно спочатку отримати точне уявлення про різні аспекти й відмінності між технологіями й різними типами БД.

SQL являється мовою програмування, однак не в звичайному її значення, такому як Python або JavaScript. Замість цього SQL має на меті єдину ціль: отримувати доступ до даних, маніпулювати й керувати ними.

Якщо точніше, то SQL - це мова запитів, котра дозволяє видобувати дані з БД, і в цьому розумінні SQL створена й предназначена для роботи з даними та керування реляційними БД[9].

Реляційна БД являє собою такий тип БД, котрий зазвичай організований у виді таблиць та який дозволяє розпізнавати й отримувати доступ до даних, котрі відносяться до деякої іншої частини даних у цій же БД. Іншими словами, пов’язані дані зберігаються в декількох таблицях, що організовані у рядки й стовпчики, що дозволяє людині, яка користується БД, одночасно працювати з даними з різних таблиць. Реляційна БД наслідує реляційну модель даних. Для обслуговування та роботи с реляційною БД використовують систему керування реляційною БД - СКБД. Для роботи у цій системі велика кількість БД, зазвичай, використовують SQL для керування роботи з запитами до БД.

Сама по собі SQL не є системою БД, тому можна вважати, що порівнюючи NoSQL та SQL, ми порівнюємо безпосередньо нереляційні та реляційні БД. Також варто відзначити, що SQL є не єдиною мовою запитів, котра здатна забезпечувати роботу з реляційними БД, однак SQL є найбільш популярною. З цієї причини, часто SQL БД та реляційні БД виступають в якості синонімів. Одними з найвідоміших SQL СКБД є PostgreSQL й MySQL.

Коли ми говоримо про NoSQL, то мова йде про нереляційні та розподілені БД. NoSQL можна сприймати як «не лише SQL», тому що деякі

NoSQL системи все ж таки мають змогу працювати з мовою запитів SQL. NoSQL насамперед означає, що БД не являється реляційною СКБД.

У той час, коли звичайні реляційні СКБД покладаються на SQL, щоб працювати з даними й створювати запити, NoSQL системи користуються іншими інструментами та технологіями, котрі надають їм змогу зберігати структуровані або неструктуровані дані.

Згідно звіту про тенденції збереження даних, що був опублікований джерелом Dzone за 2021 рік[10], SQL БД являються найпопулярнішими СКБД. Однак, якщо об’єднати всі NoSQL БД, котрі мають відношення до усіх нереляційних СКБД, то ми побачимо, що об’єднані NoSQL БД є більш популярними, аніж SQL БД.

На рисунку 2.2 зображені найпопулярніші моделі СКБД станом на 2021 рік.

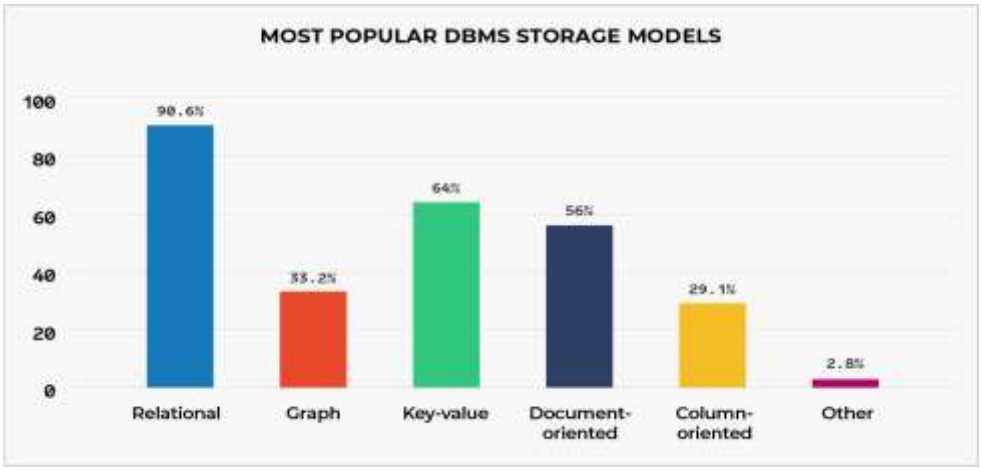

Рис. 2.2. Найпопулярніші моделі зберігання інформації

Загалом, вибір СКБД залежить, насамперед, від типу БД й вміння використовувати цим типом БД. Кожен тип БД краще всього відповідає якомусь відповідному застосунку чи додатку.

Підбиваючи підсумки можна сказати, що SQL не краща й не гірша за NoSQL, ало в той же час не є ідеальним рішенням без недоліків. Обидва рішення обираються на основі поставлених задач й використовуються для різних застосунків, опираючись на СКБД.

SQL БД зберігають й організовують дані за допомогою таблиць з фіксованими стовпцями і рядками. NoSQL БД має змогу зберігати дані різними способами: документом у JSON форматі, сховищем з широким стовпчиком, де таблиці організовані та використовуючи динамічні стовпці та рядки, парами ключ-значення та графовими БД, тобто за допомогою вузлів і ребер.

SQL бази даних потребують встановленої наперед визначеної схеми, такій структурі зобов’язані слідувати всі присутні дані. Потребується ретельна підготовка робочої системи. Будь-які можливі зміни в структурі, потенційно здатні зруйнувати цілістність системи. NoSQL бази даних не потребують визначеної структури наперед, вони представляють динамічні схеми для неструктурованих даних. NoSQL базам даних властиво мати більшу гнучкість, але це руйнує надійність.

Через процедуру JOIN є можливість залучати та передавати частину даних. Нерідко сервера нереляційних баз даних знаходяться на різних локаціях, але це не є проблемою для поєднання таблиць із різних серверів. Завдяки NoSQL можливо без проблем масштабувати за допомогою заточуванням даних. У NoSQL присутній рівень маршрутизації, він спроможний перенаправити запит, це все робить NoSQL масштабованим та чуйним на запити. Але цей підхід не відповідає ACID.

Відносно реляційних СКБД, маштабування там значно складніше втілити через ACID. Реляційна СКБД потребує швидкого внутрішнього каналу зв'язку. Такий канал повинен запобігати можливим перешкодам. Реляційні СКБД як правило, масштабуються тільки для того, щоб захистити цілісність даних.

SQL можливо описати, як декларативна, проста в реалізації та розумінні мова. Має повагу та популярна серед розробників. У NoSQL відсутня декларативна мова запитів, тому потребується додаткова обробка даних, на відмінно від SQL.

На ринку існує чимало варіантів NoSQL та SQL і вибір відповідної бази даних не є зваженим і сто відсотковим рішенням навіть для досвідченного програміста. Розглянено декілька варіантів, якщо дивитися в сторону величезної кількості неструктурованих даних, найпривабливішим варіантом буде CouchDB та MongoDB.

SQL бази даних пропонують нам величезну кількість переваг для транзакцій даних та загального збереження цілісності даних.

2.3. Архітектура веб-застосунків та її особливості

Архітектура веб-застосунків являє собою різновид програмної архітектури. Вона описує загальні процеси, котрі мають відношення до веб-застосунків й усіх програм, що використовуються та працюють з веб-браузером. Розробка архітектури будь-якого застосунку є першим й найголовнішим кроком при створенні продукту.

Термін архітектура веб-застосунку використовується розробниками ПЗ задля опису високорівневої структуру ПЗ й включає в себе особливості роботи сервера та методи зберігання даних. Правильний вибір архітектури веб-застосунку має змогу покращити безпеку, швидкість й надійність розроблюваного продукту. Окрім того, при розробці архітектури високого рівня враховуються різного роду проблеми та фактори, що касаються просування продукту[11].

Далі наведені основні параметри якості гарно продуманої архітектури веб-застосунку:

- стабільність;
- рівень безпеки;
- рівень автоматизації системи;
- чіткість щодо програмного коду;
- масштабованість компонентів продукту й самого продукту;
- швидкість обробки запитів;

- можливість аналізу кожного окремого взятого компоненту;
- повторне використання компонентів застосунку.

Веб-архітектура буквально є основою, на якій будуються всі програмні компоненти. Якщо цей фундамент міцний і стабільний, подальша робота над продуктом буде економічною за часом та витратами. Помилки на стадії проектування застосунку є дуже критичними, тому що на пізніх етапах розробки застосунку внесення змін до програмної архітектури продукту проходить дуже повільно й складно. Часто подібні зміни змушують розробника переробляти застосунок майже з нуля.

Архітектура веб-застосунку описує взаємозв'язок між користувацькими інтерфейсами, БД, системами проміжного ПЗ, тобто між компонентами застосунку, та способом їх взаємодії між собою. Іншими словами, архітектура містить набір компонентів, опис їх логіки роботи й забезпечує взаємозв’язок між клієнтами та сервером. Правильно розроблена архітектура веб-застосунків гарантує належну взаємодію всіх компонентів системи й являється надійною основою для розширення застосунку на більш пізніх етапах розробки програмного рішення.

В процесі розробки веб-застосунку розробник несе повну відповідальність за вибір інструментів та написання коду як на стороні сервера, так і з боку клієнта. Такі мови, як Python, Java й JavaScript користуються великим попитом, щодо написання коду на стороні сервера. Серверний код є відповідальним за зберігання даних різних видів й за створення веб-сторінок, за запитом користувача.

Клієнт, у свою чергу, не може напряму зчитувати файли на сервері і натомість спілкується з сервером HTTP-запитами. Клієнтський код парситься веб-браузером й відкликається на дії користувача. Користувач має можливість переглядати та модифікувати клієнтський код й у тому числі

HTML, CSS та JavaScript код, котрий використовується для створення веб-сторінок та наповнення їх контентом.

Взаємодію між користувачем та програмним сервером можна представити наступним чином[12]:

1) Користувач використовує веб-браузер та вводить URL.
2) Браузер транслює запит користувача на цифрову мову.
3) Браузер визначає шлях до сайту й надсилає запит, щоб отримати доступ до нього.
4) Сервер отримує запит, обробляє його, надсилаючи відповідь таким же чином назад клієнту.
5) Браузер користувача обробляє отриману відповідь та відображає певні отримані результати. Результатом може бути нова веб-сторінка, або оновлена версія вже відображеної веб-сторінки.

Незалежно від специфіки роботи веб-застосунку та навантажень, котрі застосунок повинен буде здатний витримати, архітектуру можна описати діаграмою, що приведена на рисунку 2.3. Діаграма описує основні принципи роботи веб-застосунків.

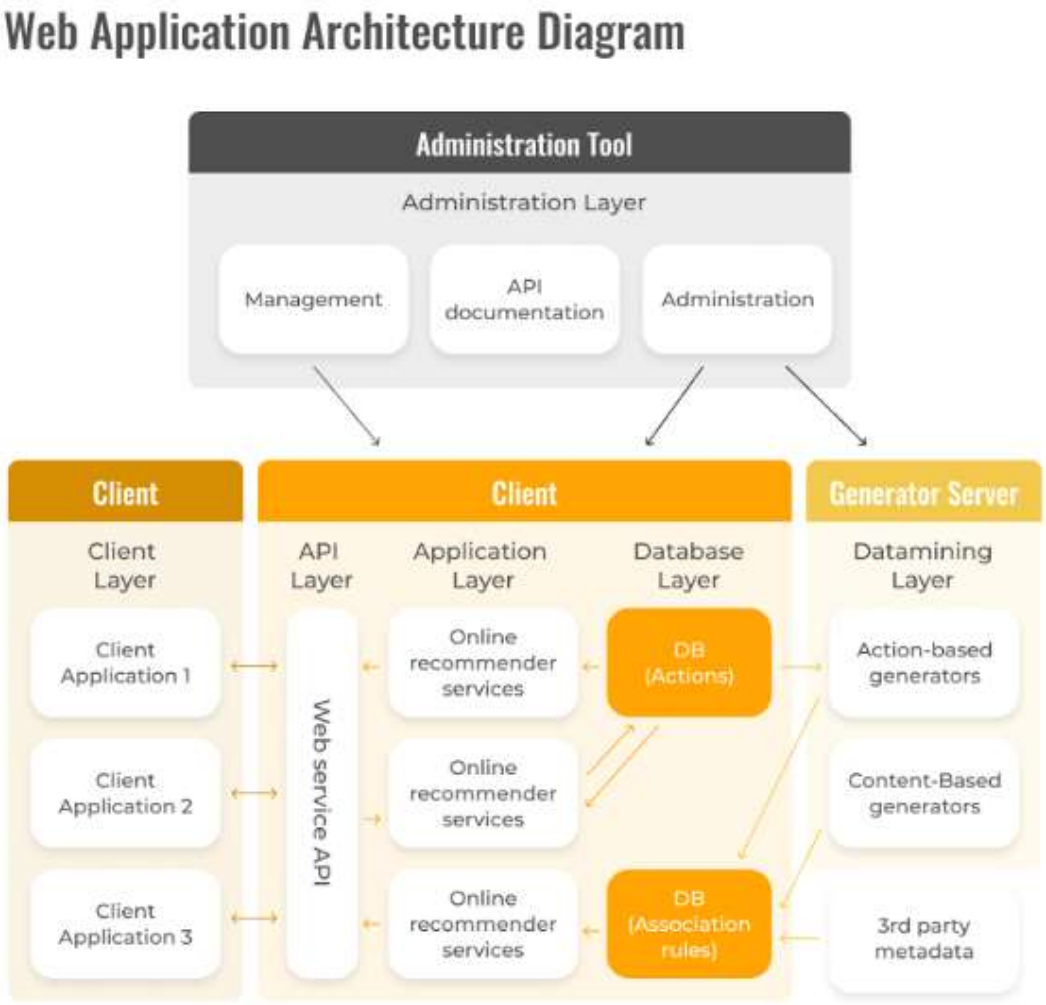

Рис. 2.3. Діаграма архітектури веб-застосунку

2.4. Рівні архітектури веб-застосунків

Компоненти архітектури веб-застосунків стосуються загальної конфігурації системи, не вдаючись у подробиці роботи рівнів архітектури. Рівні архітектури відносяться саме до вертикальної структури веб-застосунку. На рисунку 2.4 показана діаграма архітектурних рівнів, котра показує розподілення функцій між рівнями й дає змогу представити складність проекту, описаною рівнями[13].

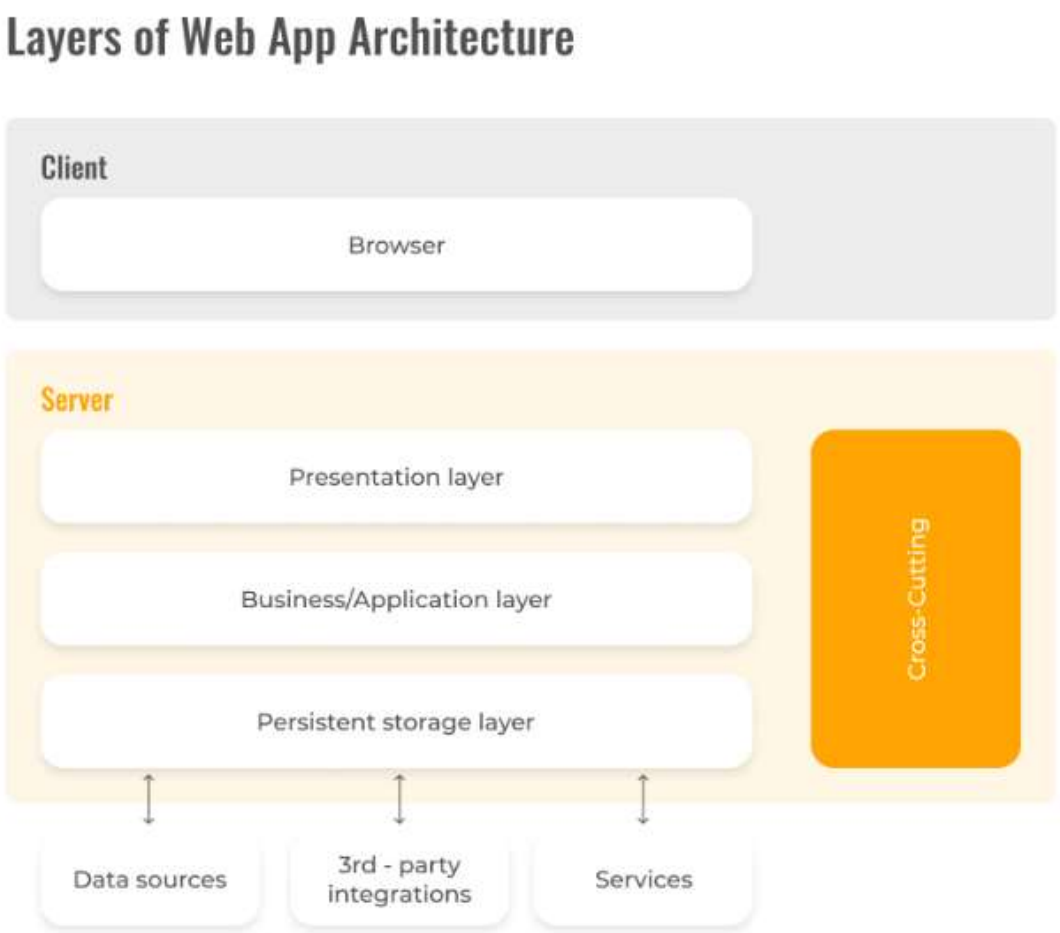

Рис. 2.4. Рівні архітектури веб-застосунку

Рівень доступу до даних(Data access layer) являє собою двері до даних, що сервер зберігає у серверній пам’яті. Даний рівень являється однією з характерних частин архітектури веб-застосунку. Рівень доступу до даних дає змогу маніпулювати даними, виконувати дії, котрі передбачає CRUD, тобто створювати, зчитувати, оновлювати та видаляти дані.

Рівень представлення(Presentation layer) відповідає за можливість сервером отримувати й працювати з вхідними даними й відображати кінцеві результати на клієнті. Рівень представлення охоплює увесь користувацький інтерфейс вцілому, тобто він має під собою усі елементи інтерфейсу, що бачить користувач й усі зв’язки міє користувачем і веб-сторінкою.

Рівень обслуговування даних(Data service layer) є відповідальним за безпеку даних клієнтів та БД. Завдяки даному рівню дані користувачів

передаються на рівень представлення системи, що дозволяє додати додатковий захист даних користувачів.

Рівень бізнес логіки(Business layer) – являє собою логіка ПЗ. Кожен тип архітектури веб-застосунку має певні особливості й властивості. Додавання їх у якості функцій на рівні бізнес логіки розроблюваної системи дозволяє зробити їх більш доступними користувачеві. Такими функціями є: обмін повідомленнями, вхід у систему чи інші форми взаємодії користувача зі змістом веб-сторінки.

2.5. Класифікація архітектур веб-застосунків

У сучасній розробці веб-застосунків існує низка популярних типів архітектур. Класифікувати й поділити на типи їх можна завдяки різному способу розподілу логіки роботи системи та комунікації між сервером та клієнтом. Також, у деяких моделей архітектур сервер може бути попросту відсутнім.

2.5.1. Single-Page Application

Головною ідеєю SPA архітектури являється те, що замість повного перезавантаження сторінки веб-сайту й витрачання на це великої кількості часу та ресурсів, у відповідь на запит користувача сервер перезавантажує лише конкретну область цієї сторінки. Такий підхід дозволяє заощадити ресурси веб-застосунку, тим самим зменшуючи показник відмов продукту. Даний тип архітектури, мабуть, є найпопулярнішим серед усіх інших на сьогоднішньому ринку інтернет послуг.

Веб-застосунки на базі SPA містять лише найнеобхіднішу інформацію та найбільш важливі елементи системи, що дозволяє їм бути інтуїтивно зрозумілими кожному користувачу. Завдяки роботі на мові JavaScript з використанням різних ефективних фреймворків, таких як React чи Angular, розроблені програмні рішення надають користувачам можливість

користуватися інтерактивним та зручним веб-застосунком. Для комунікації сторінок використовуються технології AJAX, XML й асинхронний JS[14].

На рисунку 2.5 зображена загальна схема моделі SPA.

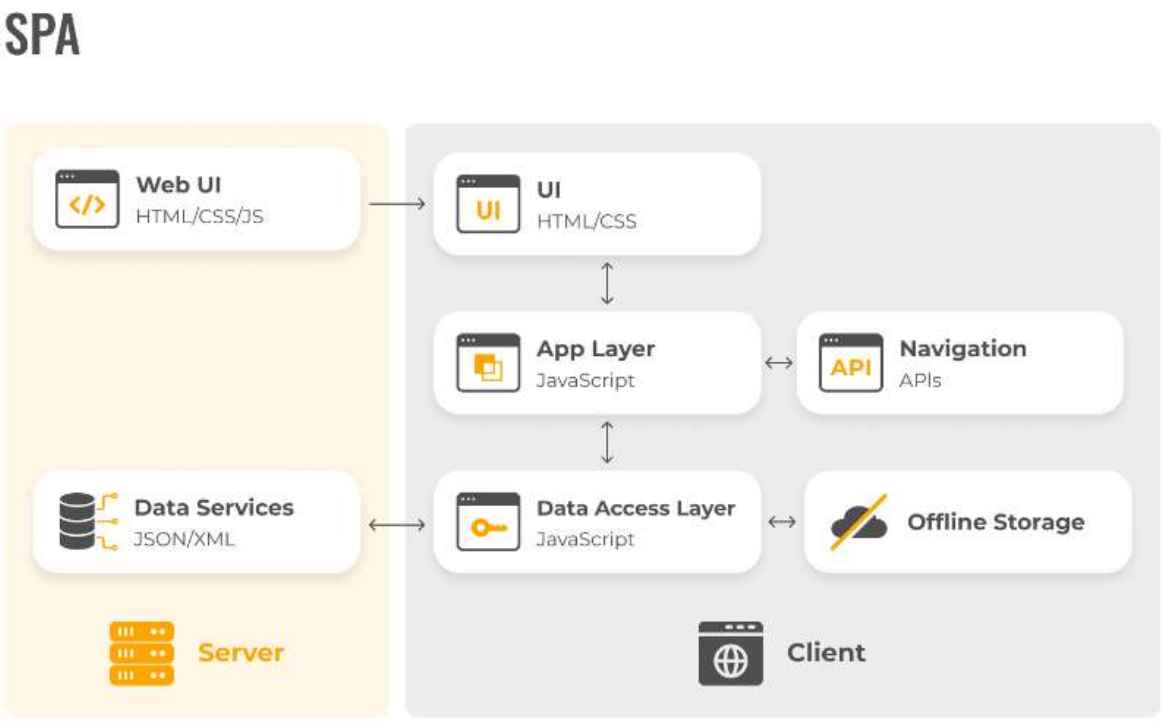

Рис. 2.5. Схема роботи Single-Page Application

На базі моделі SPA побудована велика кількість сучасних веб-сайтів. Гарними прикладами з використання технології Single-Page Application є Google Maps та Gmail.

2.5.2. Serverless

Головною особливістю безсерверної(Serverless) моделі архітектури є те, що певні операції виконуються, використовуючі різні типи існуючих хмарних служб й технологій, що є досить гнучким рішенням. Це дозволяє використовувати ресурси більш ефективно й витрачати менше часу та коштів на розробку продукту. Перевага цієї моделі полягає в тому, що вона дозволяє застосунку виконувати необхідну логіку, не турбуючись про інфраструктуру[15].

З іншого боку, безсерверні архітектурні рішення рідко використовують для розробки веб-застосунків, хоча вони й мають деякий попит при розробці продуктів у новітніх стартап компаній. Використання сторонніх служб й технологій, більшою мірою, дозволяє пришвидшити випуск продукту, що дозволяє не втратити довіру з боку користувачів й, навіть, підвищити його надійність. Сторонніми службами можуть бути: авторизація, подтримка фінансових операцій, розпізнавання голосу й зображень тощо.

На рисунку 2.6 зображена загальна схема Serverless архітектури.

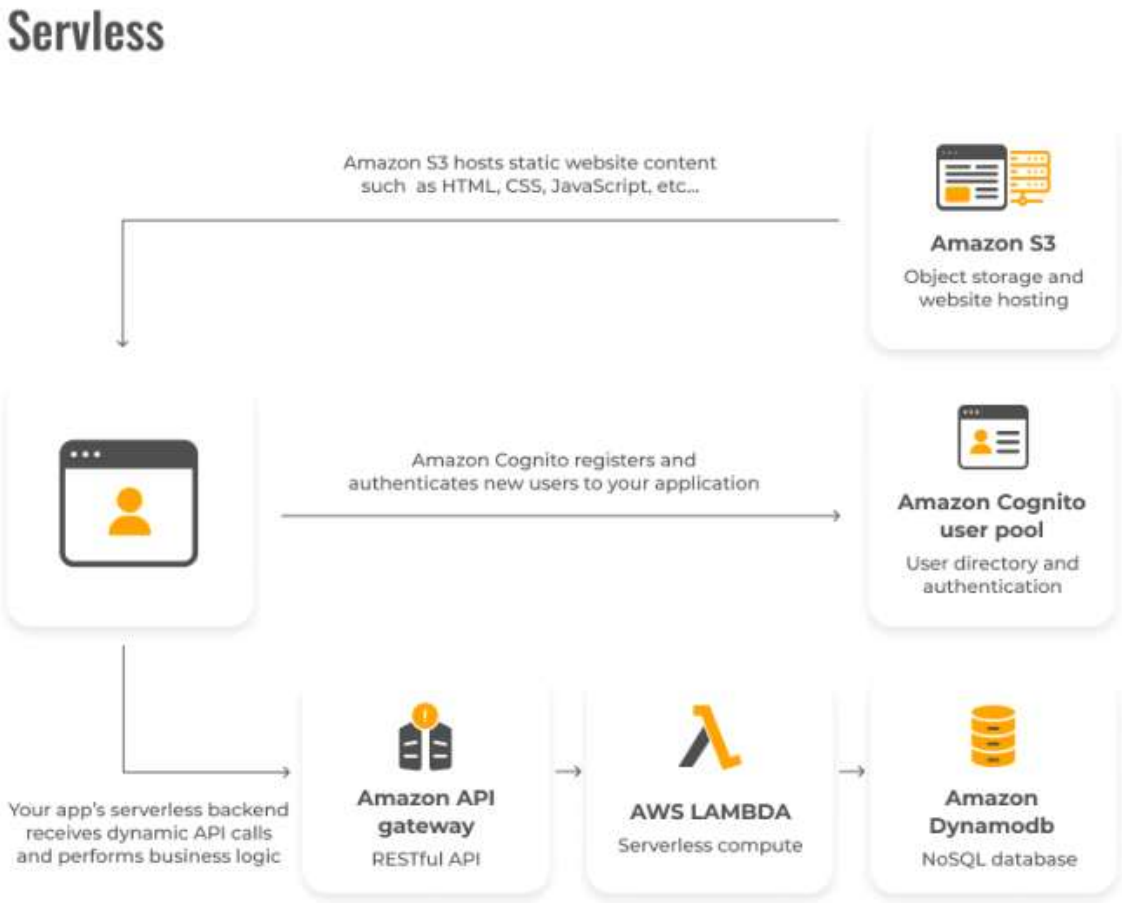

Рис. 2.6. Схема роботи Serverless

Такі компанії як Microsoft та Amazon, є монополістами, у випидках використання сторонніх хмарних технологій та сервісів.

2.5.3. Microservices

Microservices, або мікросервісна архітектура – це різнових сервісно-орієнтованої програмної архітектури, котрий представляє собою рішення, що використовуються для створення розподіленого ПЗ.

Головною перевагою мікросервісної архітектури є те, що дане архітектурне рішення допомагає легко й просто збільшувати масштаб застосунку. Мікросервісна архітектура надає змогу тестувати й аналізувати різні компоненти продукту значно простіше. Також, це впливає і на обслуговування системи. Завдяки такому стилю архітектури розробники використовують набір вільно пов’язаних між собою служб, котрі можуть працювати незалежно один від одного[16].

Компоненти системи можуть бути розроблені за допомогою різних мов програмування, що робить вибір мови програмування для створення системи більш гнучким, пришвидшує процес розробки й підвищує продуктивність розробки продукту. Дана архітектура отримала такі переваги через те, що кожний окремий сервер здатний постійно підтримувати взаємозв’язок з іншими серверами. Кожен мікросервер відповідає за кожну функцію у системі. Таке системне розподілення робить створення й розширення застосунків простішим.

На рисунку 2.7 показана загальна схема роботи мікросервісної архітектури.

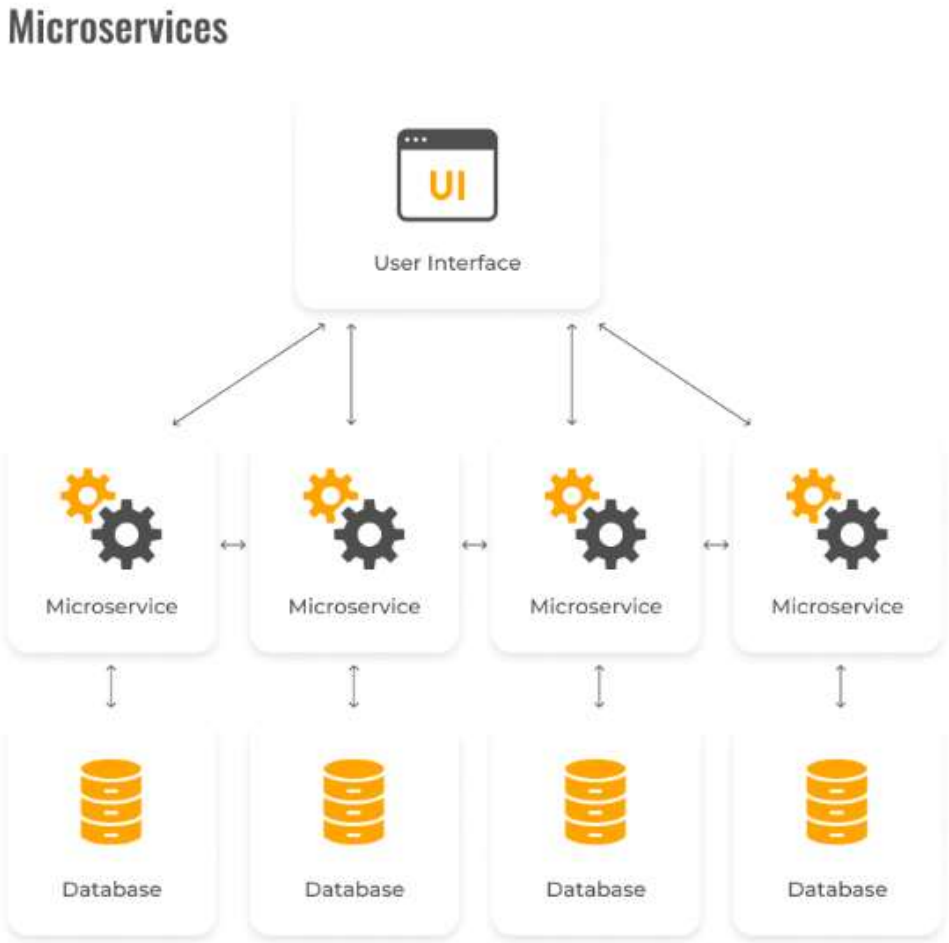

Рис. 2.7. Схема роботи Microservices

Найбільш відомими продуктами на базі даної архітектури є Netflix та PayPal.

2.5.4. Multi-Page Application

MPA технологія є досить поширеною серед існуючих веб-сайтів, оскільки, деякий час тому така архітектурна модель була актуальною на ринку інтернет послуг і тому всі веб-застосунки будувалися з використанням архітектури MPA. Головною особливістю MPA є те, що вона перезавантажує веб-сторінки з метою надіслати запит на сервер або з сервера, використовуючи користувацький інтерфейс[17]. Таку багатосторінкову архітектуру зазвичай використовують для створення великих веб-

застосунків. Найбільш відомими продуктами, що реалізовані на базі MPA є Ebay та Amazon.

На рисунку 2.8 зображено життєвий цикл MPA в порівнянні з SPA архітектурою.

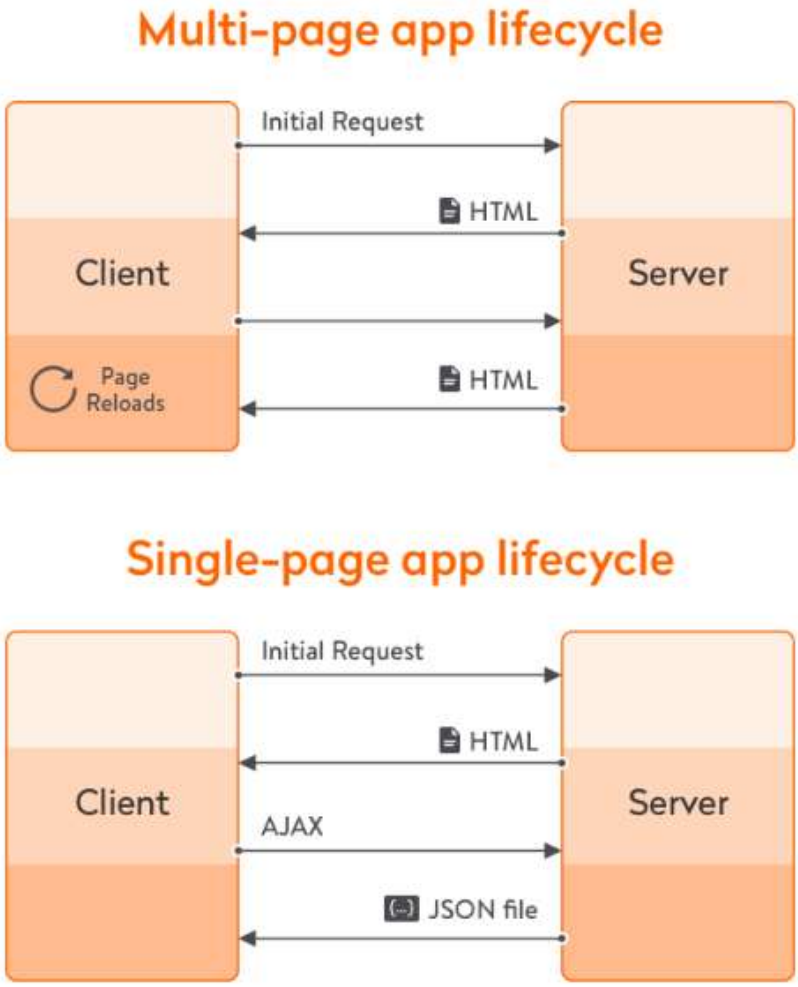

Рис. 2.8. Порівняння життєвих циклів MPA і SPA

2.5.5. Progressive Web Application

PWA архітектура являється найсучаснішим рішенням у питанні вибору архітектури веб-застосунку. Попри те, що такий підхід є досить новим та мало розвіданим, він вже користується колосальним попитом у провідних IT-компаній світу. Використовуючи PWA технологію користувач отримає безліч позитивних вражень та нових емоцій від гарного програмного інтерфесу, який тішить око.

Серед головних особливостей при розробці PWA застосунку можна виділити необхідність використання протоколу захищеної передачі даних HTTPS, одного чи декількох потоків Service Worker й Manifest-файлу. Дані особливості водночас являються й вимогами до програми[18].

Технологія PWA змушує веб-застосунок виглядати як мобільну програму, при всьому тому, що користувач не повинен завантажувати ніяких програм чи застосунків на свій мобільний пристрій чи ПК. Прогресивні веб-застосунки здатні працювати, навіть, у офлайн режимі, що вельми спрощує життя звичайним користувачам програм, які не мають постійного з'єднання з мережею інтернет.

Також за допомогою PWA можно надсилами пуш-сповіщення, задля сповіщення користувачів, й працювати, використовуючи API апаратного забезпечення. PWA програми поєднують у собі найкращі сторони мобільних застосунків та переваги звичайних веб-сайтів, що робить їх універсальними. Наприклад, використовуючи застосунок на базі PWA можна додати значок застосунку на робочий стіл персонального комп'ютера або на екран будь-якого мобільного пристрою., необхідність скачування застосунку при цьому відсутня.

Веб-застосунок обов'язково має працювати через HTTPS мережу. Додання захищеності дає користувачу можливість повністю довіряти застосунку й показує особливе піклування про безпеку користувацької інформації. Такий підхід грає особу роль, у разі виникнення потреби здійснювати надійні транзакції. HTTPS відкриває доступ до технології Service Worker.

Service Worker являє собою сценарій, котрий надає можливість маніпулювати тим, як браузер перехоплює свої мережеві запити й кешує ресурси. Саме завдяки використанню Service Worker застосунок має змогу працювати при слабкому або взагалі відсутньому зв'язку с мережею інтернет. Також, це додає надійності веб-сторінці.

Manifest-файл насправді є JSON файлом, котрий здатний управляти відображенням застосунку користувачеві й надає гарантію загальнодоступності PWA. Manifest визначає назву застосунку, URL посилання та інші параметри, котрі необхідня для подібності до справжньої комп’ютерної чи мобільної програми.

Застосунки на базі технології PWA здатні оновлюватися за допомогою серверного коду. Вони здатні адаптуватися до розміру дісплею юзера, що робить їх гарною заміною звичайних мобільних застосунків. Приємною особливістю є те, що PWA можуть працювати за умов поганого доступу до мережі інтернет або взагалі при відсутньому підключенні до інтернету. Це реалізується завдяки кешуванню.

Для розробки продуктних рішень на базі PWA розробники використовують JavaScript, HTML, CSS та інші технології. У разі необхідності є можливість використовувати додаткові API[19], такі як NFC, Bluetooth тощо.

Серед відомих продуктів, побудованих на базі архітектури PWA є Uber, Pinterest та Starbucks.

На рисунку 2.9 зображена загальна схема роботи технології PWA.

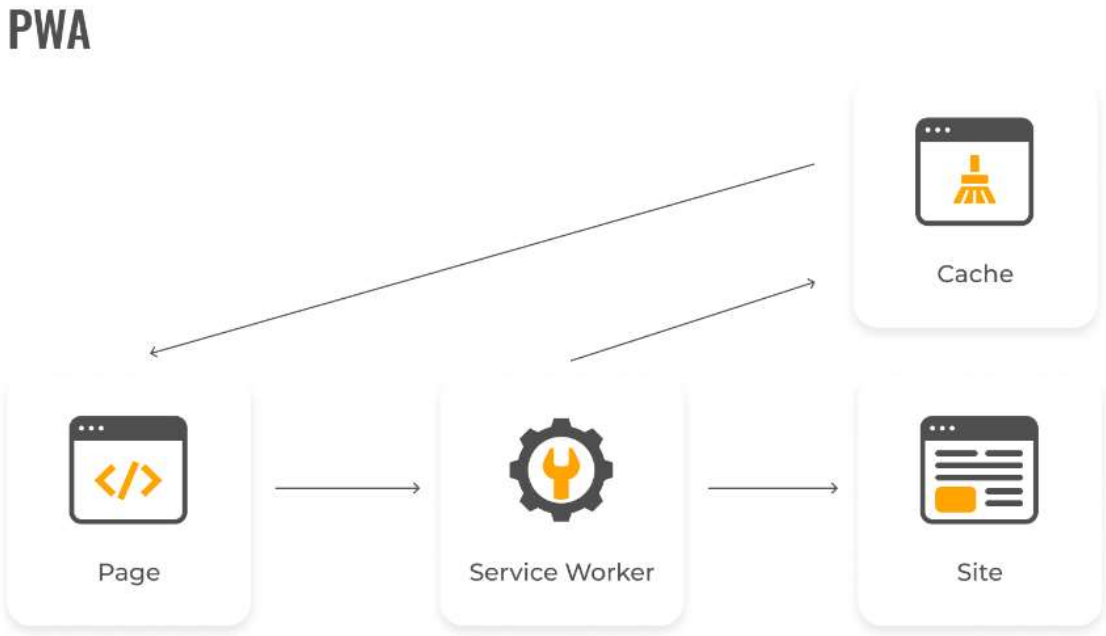

Рис. 2.9. Схема роботи PWA

Серед переваг PWA можна виділити:

- здатність до знаходження через пошукові системи. Головною задачею PWA є покращення роботи з пошуковими системами. Таким чином, застосунки простіше виставляти, каталогізувати й класифікувати, що надає їм особливі можливості;
- PWA дозволяє додати значок, без необхідності встановлення будь-яких програм;
- відсутність необхідності у використанні магазину веб-застосунків. Замість цього застосунком можна користуватися шляхом доступу через URL;
- можливість отримання доступу до веб-сайту та перегляду контенту без доступу до мережі інтернет;
- PWA може працювати на будь-якій ОС;
- одразу після випуску оновлення всі нові функції програми стають автоматично доступними юзеру;
- PWA можуть за допомогою медіа-запитів перевіряти відповідність інтерфейсу користувача до розміру екрану пристрою;
- PWA предоставляє безпечний механізм доставки, котрий запобігає перегляду й надає гарантію справжності змісту, завдяки HTTPS захисту;
- виглядає й працює як мобільний застосунок;
- займає малу частину пам’яті пристрою.

У таблиці 2.1 приведено порівняння PWA з мобільним застосунком.

Таблиця 2.1

Порівняння програмної архітектури PWA з класичним мобільним застосунком

Властивість	**Мобільний застосунок**	**PWA**
Опис	Для певної платформи чи ОС	Веб-зайт з інтерфейсом та подібними програмним можливостями
Установка	Завантаження з магазину мобільних застосунків на конкретний пристрій	Завантаження не потрібне, є можливість додати значок застосунку
Доступ до функцій пристрою	Повний доступ	Обмежений доступ
Робота у офлайн режимі	Можлива у більшості випадків	Можлива, завдяки кешуванню
Оновлення	Присутні	Автоматичні оновлення без участі користувача
Пошукова оптимізація	Не індексуються у пошуковій системі	Гарна пошукова оптимізація
Безпека	Рівні захисту на основі вбудованих у пристрій компонентів	HTTPS шифрування
Push-сповіщення	Присутні	Присутні всюди, крім IOS
Комунікація між програмами	Присутня	Відсутня

РОЗДІЛ 3
ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ

Першою циглиною на шляху до реалізації вдалого й функціонального веб-застосунку є проектування й визначення загальних вимог, а саме: функціональних вимог до застосунку, вимог до серверу, вимог до клієнта та інших.

3.1. Функціональні вимоги до застосунку

В процесі написання роботи та реалізації веб-застосунку, першим кроком стало визначення функціональних вимог до проекту. Варто пам’ятати, що основою веб-застосунку повинна була стати саме PWA технологія на базі клієнт-серверної моделі архітектури.

Функціональними вимогами до проектованого веб-застосунку є:

- має бути реалізована можливість створення облікового запису для користувача;
- має бути реалізована можливість увійти до свого облікового запису;
- має бути реалізована можливість виходу з поточного аккаунту користувача;
- має бути реалізована можливість авторизуватися з використанням Gmail;
- веб-застосунок повинен мати функцію створення нових задач;
- має бути реалізована можливість перегляду усіх існуючих задач;
- має бути реалізована можливість редагування задач;
- має бути реалізований перегляд статистики по обліковому запису юзера;
- у веб-застосунку має бути реалізована можливість сортування задач по категоріям;
- веб-застосунок має бути незалежним від операційної системи, на якій він буде працювати;

- веб-застосунок має підтримувати офлайн-режим;
- веб-застосунок має бути позбавлений архітектурних помилок і справно працювати;
- веб-застосунок має бути налагоджений для роботи з мобільними пристроями тощо.

3.2. Вимогу до серверу веб-застосунку

Проектуємий сервер має відповідати принципам Representational State Transfer(REST), які переводяться як Передача Репрезентативного Стану й забезпечувати безпеку використання, надійність, масштабованість та відмовостійкість. Сервер має передавати й постачати дані у тому форматі, котрий буде зручний для клієнта. Наприклад, такими форматами можуть бути XML або JSON.

Разом з сервером має бути реалізований і API, тобто програмний інтерфейс застосунку, який буде отримувати клієнтські HTTP запити й виконувати CRUD. Також, обов’язковою є наявність БД, для збереження, видалення або зміни інформації користувача. Далі приведені основні функції, котрі мають надаватися сервером:

- реєстрація акаунту для нового юзера;
- авторизація вже існуючого юзера;
- логаут з авторизованого облікового запису юзера;
- додавання нової задачі для юзера;
- видалення й вилучення вже існуючої задачі;
- отримання усіх створених задач;
- отримання задач згідно з обраним класом задачі;
- додавання нового класу задач;
- отримання всіх існуючих класів задач;
- редагування вже існуючого класу задач;
- видалення вже існуючого класу задач;

- зміна й маніпуляції вже існуючою задачею.

3.3. Вимоги до користувацького інтерфейсу веб-застосунку

Веб-застосунок має бути простим і легким у використанні, надаючи користувачеві кращий користувацький інтерфейс. Таким чином, використовуючи популярні UX/UI рішення, в ході виконання роботи були розроблені наступні вимоги до користувацького інтерфейсу:

- користистувацький інтерфейс має містити форму для реєстрації нового юзера, кнопку реєстрації разом з полем вводу логіну та полем вводу пароля;
- користувацький інтерфейс має містити форму для авторизації вже існуючого юзера, з полем вводу логіну, полем вводу пароля, кнопкою авторизації й кнопкою авторизації за допомогою Gmail;
- головна сторінка веб-застосунку має містити кнопку логауту;
- сторінка перегляду статистики має включати в себе представлення кількості виконаних задач тощо;
- головна сторінка веб-застосунку має включати кнопкудля переходу на сторінку перегляду статистику й назад;
- головна сторінка веб-застосунку має містити кнопку створення нової задачі. В момент натискання на кнопку має відкриватися форма створення нової задачі;
- на головній сторінці веб-застосунку мають знаходитися кнопки зміни класу задач, що там знаходяться.

3.4. Проектування застосунку та бази даних

У зв’язку з вимогами до застосунку, що були приведені раніше, в ходу проектування застосунку були розроблені відповідні діаграми, котрі дозволяють візуалізувати всю систему, її складові та краще представити принципи роботи програми та було опрацьовану вигляд схеми БД.

3.4.1. Схема бази даних

У контексті БД схема є деяким планом, котрий організовує дані у таблиці, кожна з якої має свої зв’язки, стовчкими та ключі. Якісно розроблена схема бази даних чітко показує дані у різних сутностях, атрибути сутностей й взаємозв’язки між сутностями, накладаючи обмеження до типів даних.

На рисунку 3.1 можна ознайомитися з розробленою схемою БД.

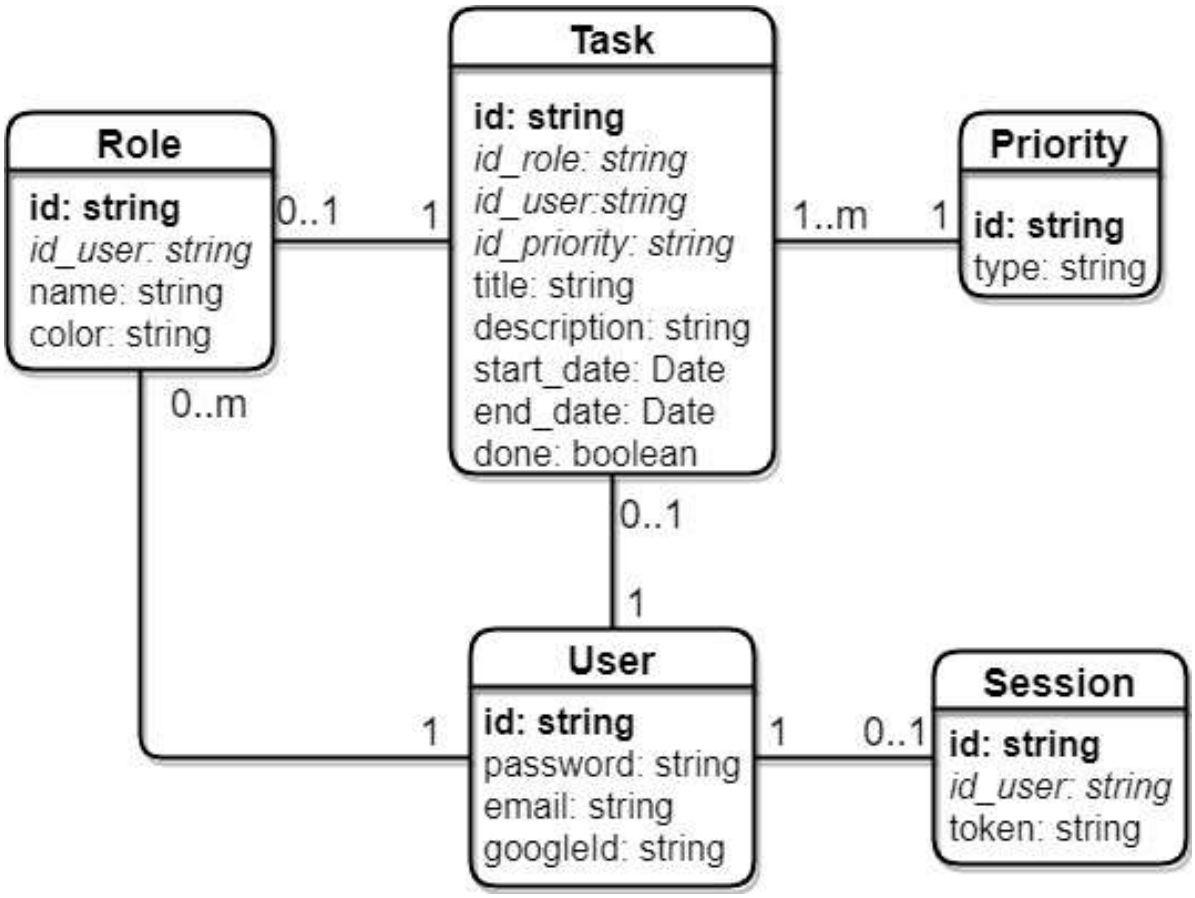

Рис. 3.1. Схема БД

3.4.2. Sequence діаграма

Sequence діаграма, або діаграма послідовності, являється однією з діаграм взаємодії та показує взаємодію між об’єктами системи в одному випадку використання системи. Вона моделює взаємодію різних компонентів системи, їх поведінку під час виконання деякої функції та ілюструє порядок, у якому під час виконання конкретного прецеденту, відбувається взаємодія

між компонентами[20]. Кожен об’єкт має свій окремий стовпець, стрілками показані повідомлення, котрими об’єкти обмінюються між собою. Система представляє собою деяку часову шкалу, яка починається зверху й поступово йде донизу.

На рисунку 3.2 зображена Sequence діаграма, що показує взаємодію між об’єктами у порядку послідовності зверху донизу.

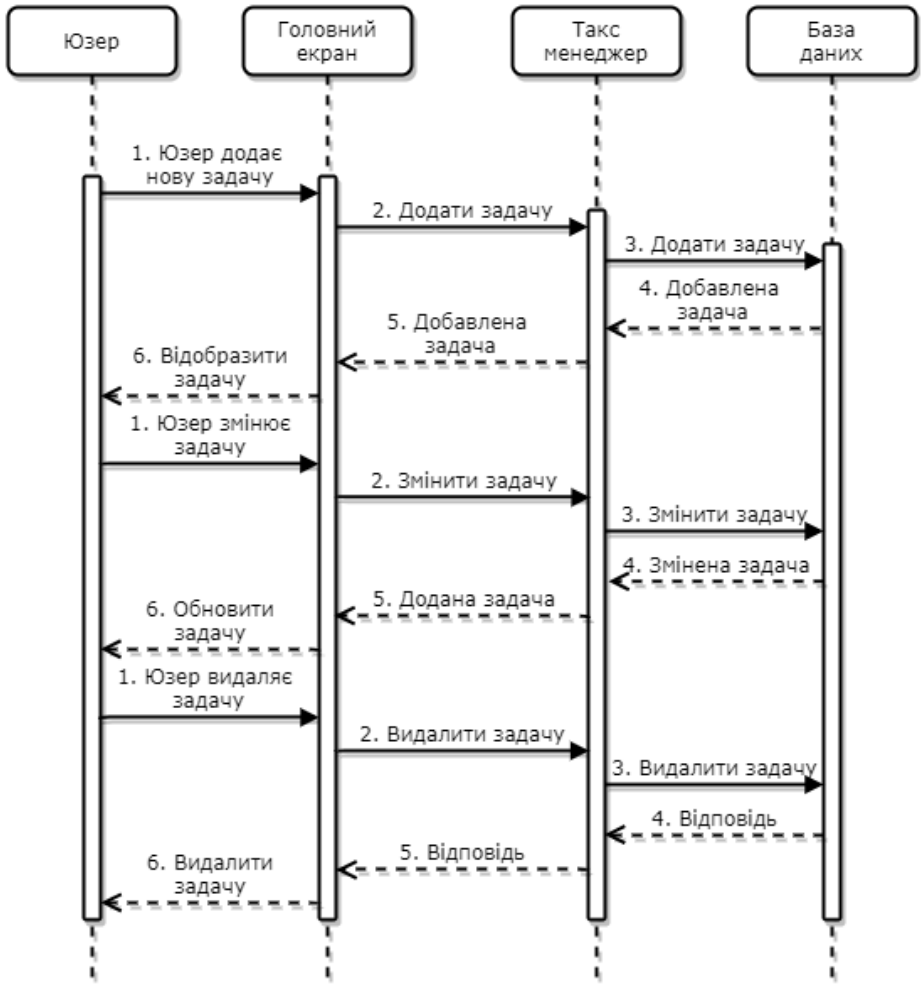

Рис. 3.2. Sequence діаграма

3.4.3. Use Case діаграма

Use Case діаграма моделює функції та основні задачі, котрі повині виконуватися системою та фіксує динамічну поведінку системи. Вона включає в себе основні концепції UML моделювання системи та, використовуючи випадки використання системи й акторів, описує вимоги та

функціональні можливості системи[21]. Показує, як зовнішня сутність(користувач) може поводитися з системою.

На рисунку 3.3 зображена Use Case діаграма, що показує загальні функції проектованого веб-застосунку.

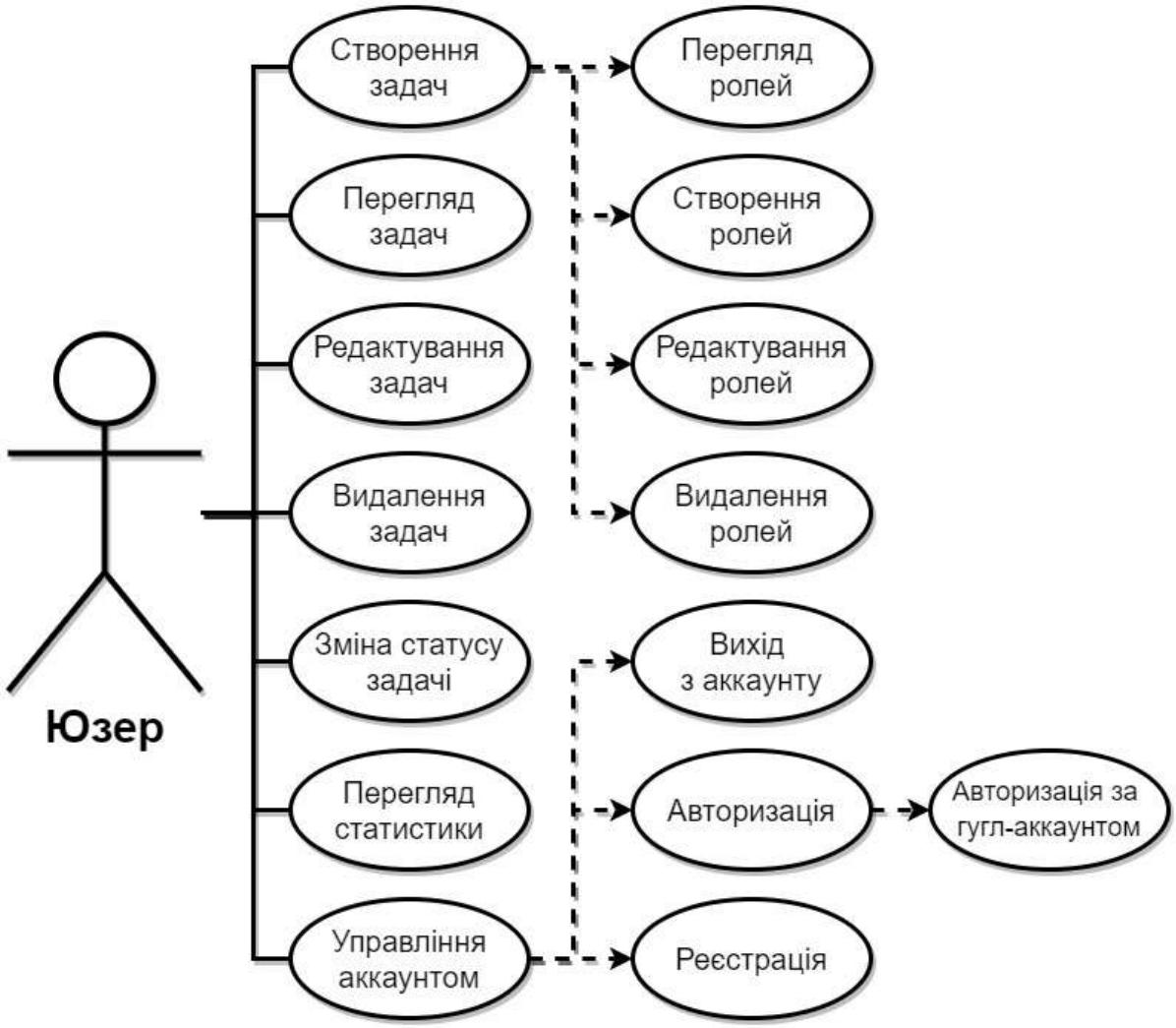

Рис. 3.3. Use Case діаграма

3.5. Інструменти розробки серверу веб-застосунку

В якості основного інструметру для розробки серверної частини застосунку було вирішено використовувати мову програмування JavaScript та технологію NodeJS.

NodeJS являє собою відкриту й безкоштовну неймовірно потужну платформу, котра працює на базі JavaScript й використовується в більшості

випадків для реалізації стрімінгових сервісів, SPA програм та інших веб-застосунків. Платформа побудована на базі двигуна JavaScript V8 від компанії Google та користується великим попитом серед розробників по всьому світу. Компанії Walmart і Paypal теж не зуміли обійти технології стороною. NodeJS має безліч переваг, які роблять її кращим вибором, у порівнянні з іншими технології серверної розробки.

На рисунку 3.4 зображено архітектуру NodeJS.

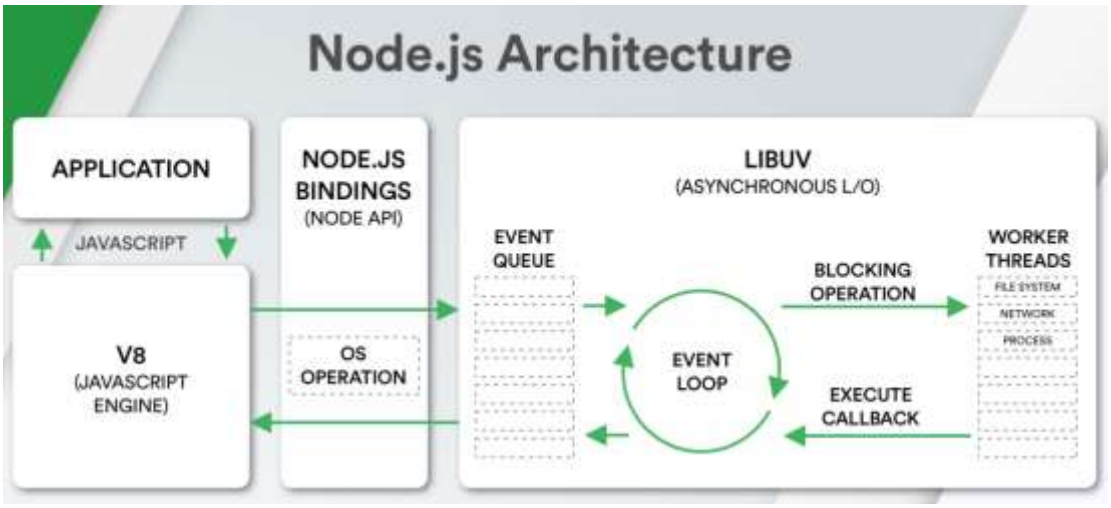

Рис. 3.4. Архітектура платформи NodeJS

Серед переваг NodeJS можна виділити наступні[22]:

- платформа є досить швидкою для виконання програмного коду;
- є багатопоточним, що дозволяє надавати послуги по набагато більшій кількості запитів;
- має необмежені потоки обробки запитів, що дозволяє легко масштабуватися;
- NodeJS використовує асинхронне програмування, що додає швидкості виконання, тому що серверу не треба чекати, доки API поверне дані;
- у платформи відсутня буферизація, що знову ж таки дозволяє зменшити час обротки при завантаженні відеофайлів чи аудіофайлів;

- NPM нараховує тисячі колекцій, які у разі необхідності можна імпортувати та використовувати, у тому числі HTTPS та HTTP модулі.

Також, гарним рішенням є використання фреймворку Express, котрий надає великий вибір інструментів та функцій для якісної розробки веб-застосунків. Фреймворк робить простішим керування сервером та маніпулювання маршрутами.

Серед особливостей фреймворку можна виділити:

- дозволяє використовувати динамічний рендеринг веб-сторінок, базуючись на передачі аргументів до шаблонів;
- має таблицю маршрутизації, котра працює на базі HTTP та URL;
- надає змогу використовувати мідлвари, щоб відповідати на HTTP запити, що були отримані від клієнта.

Під час пошуку інструменту для зберігання інформації користувача мій вибір зупинився на нереляційній NoSQL базі даних MongoDB, котра замість таблиць і рядків використовує колекції, які містять гнучкі документи з різною кількістю полів, щоб зберігати й обробляти різні формати даних. MongoDB надає змогу доволі просто зберігати багатовимірні типи даних. Документи MongoDB мають змогу розподілятися між кількома системами й відформатовані як двійковий JSON[23]. Модель даних БД надає змогу простіше представити ієрархічні відносини, щоб зберігати складні структури даних.

Головною перевагою перед іншими СКБД є те, що MongoDB здатна обробляти такі запити, котрі не вимагають схем, що були попередньо визначені, поля можна створювати на льоту. Також, приємною особливістю даної СКБД є те, що вона підтримує велику кількість мов програмування, що додає певної зручності у використанні.

Мова запитів, що використовується, надзвичайно схожа на SQL, що робить її доступною для недосвічених користувачів. СКБД MongoDB була обрана, тому що вона надає швидку й просту розробку, що є дуже важливою перевагою за умов обмеженого часу.

3.6. Інструменти розробки клієнту веб-застосунку

Для розробки клієнту веб-застосунку було вирішено використовувати мову програмування JavaScript. На сьогоднішній день таке рішення є найбільш популярним серед фронтенд розробників та признаним кращим у світу. JavaScript розвивається семимильними кроками, забезпечуючи найкращі UX/UI на всьому ринку розробки клієнтської частини веб-застосунків. JavaScript являється динамічною мовою програмування, здатною створювати динамічно оновлюваний зміст веб-сторінки, що дозволяє розробнику надати веб-сторінці більшої інтерактивності й динамічності, якої неможливо досягти використовуючи тільки HTML і CSS технології. JavaScript здатний працювати з усіма веб-браузерами та платформами, що широко використовуються користувачами.

Для більш швидкої й ефективної розробки користувацького інтерфейсу я використовую популярну компонентну інтерфейсну JavaScript бібліотеку з відкритим кодом React(ReactJS). Дана технологія була створена, щоб вирішити проблему часткового оновлення контенту на сторінці веб-браузера, з якою мають справу розробники веб-застосунків на базі архітектури SPA. React побудаваний на принципах перевикористання програмного коду. ReactJS веб-застосунки складаються з декількох React компонентів, які мають свою логіку й можуть бути перевикористані повторно. Технологія не є повноцінною платформою, тому реалізація маршрутизація й виклику веб-API тільки за допомогою React платформи є неможливою[24].

Приємною особливістю технології React являється використання DOM або, іншими словами, віртуальної моделі документу. Використання DOM має

багато переваг, основними з яких є: ефективність роботи з можливість оптимізації процесу оновлення окремих компонентів справжньої моделі DOM та можливість налагодження й тестування бібліотеки React й її взаємодії з DOM, не користуючить веб-браузером. Інтерфейс DOM дозволяє динамічно застосовувати стилі до веб-сторінки, та маніпулювати HTML й CSS. Окрім цього, React реалізує новий спосіб комунікації з HTML. React додає HTML напряму у JavaScript у виді JSX синтаксису, котрий має змогу компілюватися в чистий JavaScript код.

3.7. Розроблені системні компоненти

Використовуючи схему БД, що була розроблена на етапі проектування застосунку, були розроблені відповідні схеми БД серед якиї можна виділити наступні:

- схема користувача;
- схему ролей;
- схема сесії;
- схема задачі;
- схема приорітету.

В ході написання серверної частини застосунку було створено маршрутизатор, який здатний переадресосувати запити до необхідних контролерів. Ознайомитися з ним можна у «Лістингу А.1» і «Лістингу А.2».

Згідно з приведеними раніше вимогами до моделі архітектури веб-застосунку PWA, в ході написання сервера був розроблений manifest-файл й Service Worker, ознайомитися з якими можна у «Лістингу А.3» і «Лістингу А.4».

Далі буде описана загальна логіка роботи розробленого веб-застосунку.

Коли новий, ще не зареєстрований, юзер вперше потрапляє до веб-застосунку, то першим, що він бачить, є сторінка реєстрації нового

облікового запису. На сторінці реєстрації юзер вводить логін, тобто свою актуальну електронну пошту й бажаний пароль. Далі юзеру пропонується погодитися з умовами користування сервісом. Потім юзер натискає на кнопку реєстрації.

Був розроблений відповідний компонент сторінки реєстрації юзера.

На рисунку 3.5 можна ознайомитися зі сторінкою створення нового облікового запису юзера, сторінкою реєстрації.

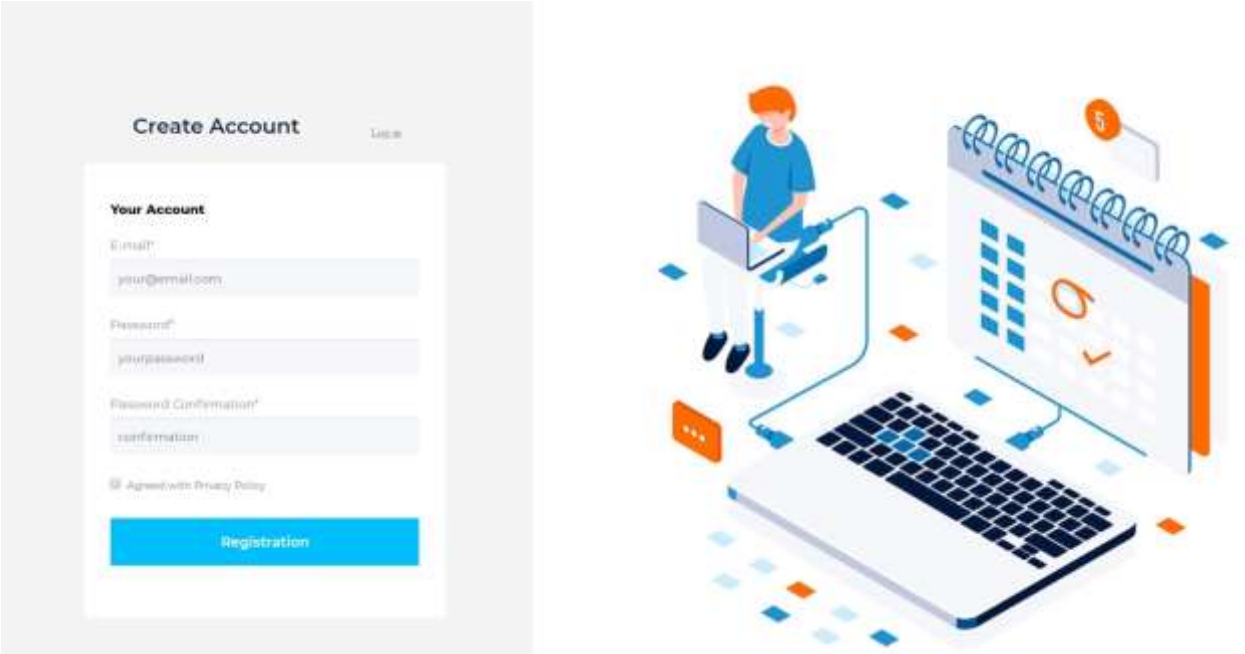

Рис. 3.5. Сторінка створення нового облікового запису юзера

Після проходження усіх перевірок новий обліковий запис юзера буде створено. У разі, якщо електронна пошта юзера вже занесена до системної БД, клієнт побачить помилку реєстрації.

У разі якщо юзер вдало пройшов реєстрацію нового облікового запису або якщо він вже має обліковий запис, то юзер може скористуватися сторінкою авторизації. На сторінці авторизації, для проходження процедури авторизації, юзер має ввести свої логін та пароль у відповідні поля форми авторизації. Якщо перевірку буде вдало пройдено, юзер потрапить прямісінько до системи. У разі неспівпадіння логіну чи паролю, юзер

отримає помилку. Також, на сторінкі авторизації знаходиться кнопку, натиснувши на яку юзер має змогу авторизуватися за допомогою Gmail.

Був розроблений відповідний компонент сторінки авторизації юзера.

На рисунку 3.6 зображено сторінку авторизації юзера.

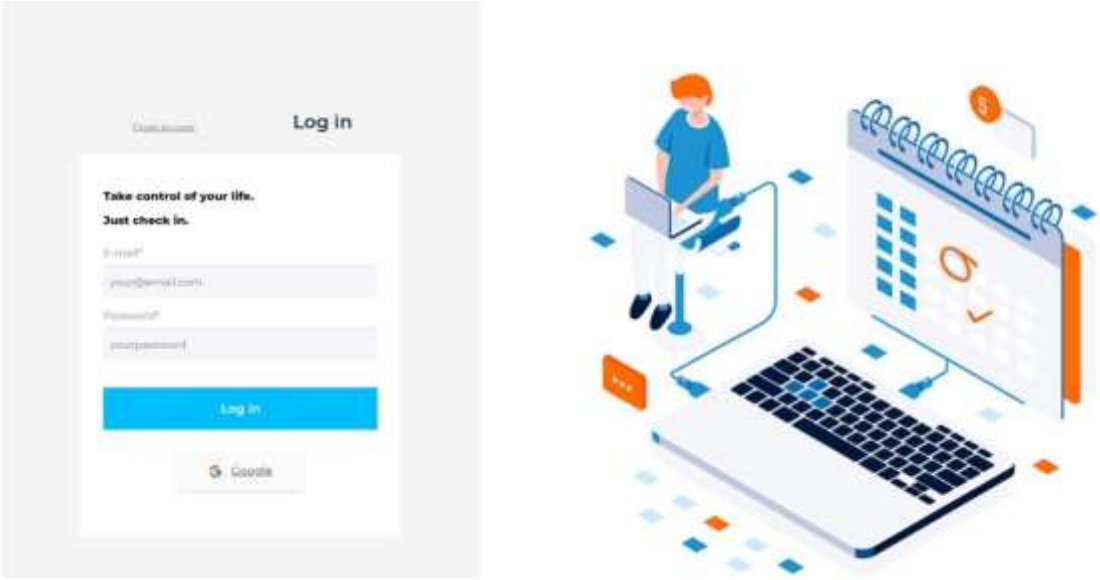

Рис. 3.6. Сторінка авторизації юзера

У разі вдалого проходження процедури авторизації, юзер опиняється на головній сторінкі застосунку, на якій він має змогу керувати своїми картками з цілями й створювати нові картки. Кнопка логауту, якою можна скористатися у разі необхідності завершити роботу з застосунком, знаходиться там же.

Був розроблений відповідний компонент головної сторінки застосунку.

На рисунку 3.7 зображено головну сторінку веб-застосунку.

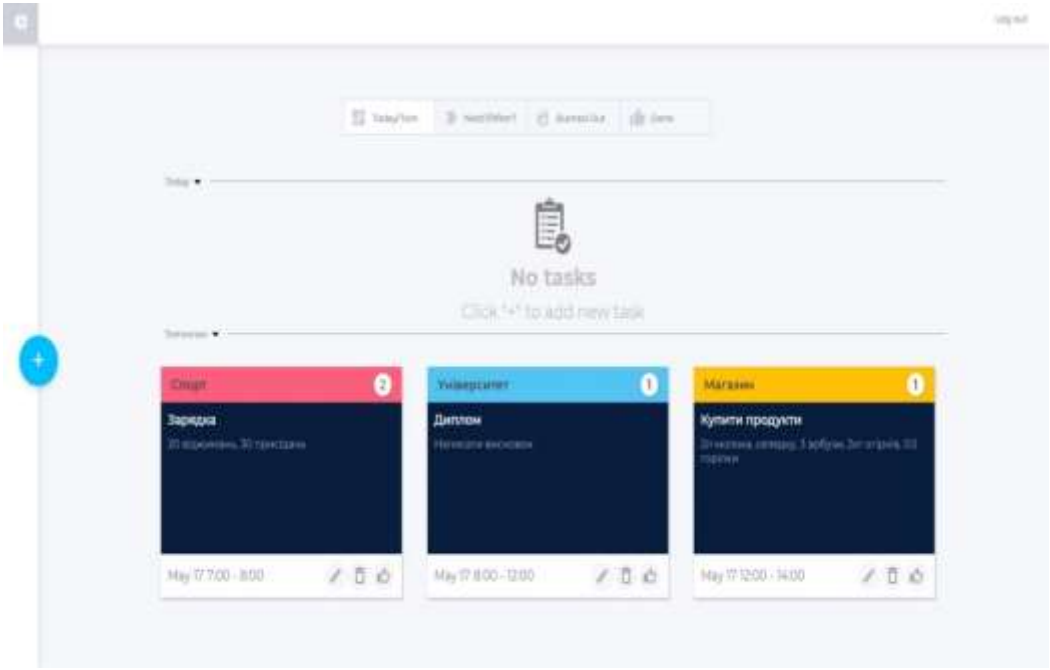

Рис. 3.7. Головна сторінка застосунку

Юзер має змогу відкривати цілі й задачі, задаючи їм деякі параметри, що показано на рисунку 3.8. На цій же сторінці юзер може керувати класами(ролями) задач, що показано на рисунку 3.9. Також є можливість видалити чи відредагувати існуючу задачу.

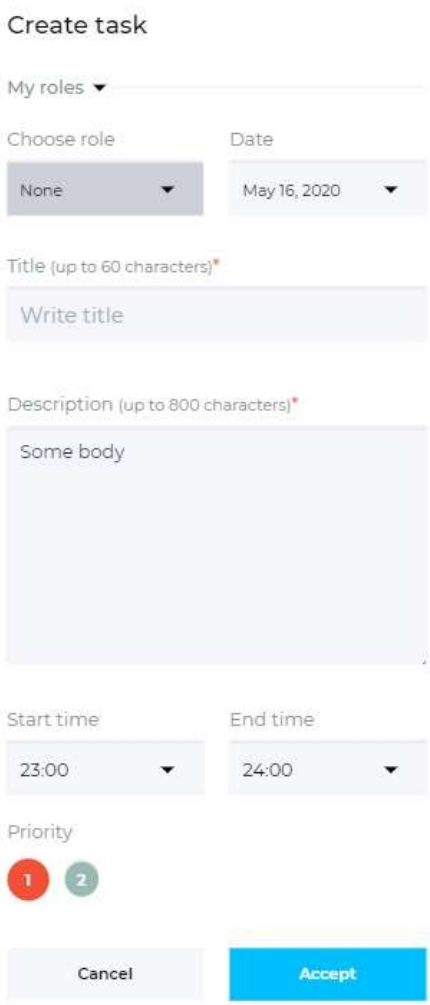

Рис. 3.8. Форма відкривання нового «таску»

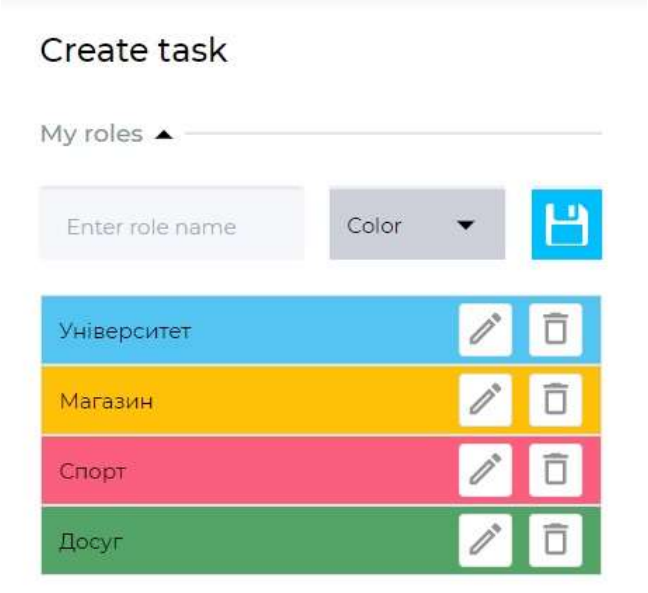

Рис. 3.9. Форма керування ролями

Також, з головної сторінки застосунку, юзер має змогу перейти на сторінку статистики й ознайомитися зі статистикою виконання своїх поставлених цілей і задач.

Був розроблений відповідний компонент сторінки статустики юзера, котра зображена на рисунку 3.10.

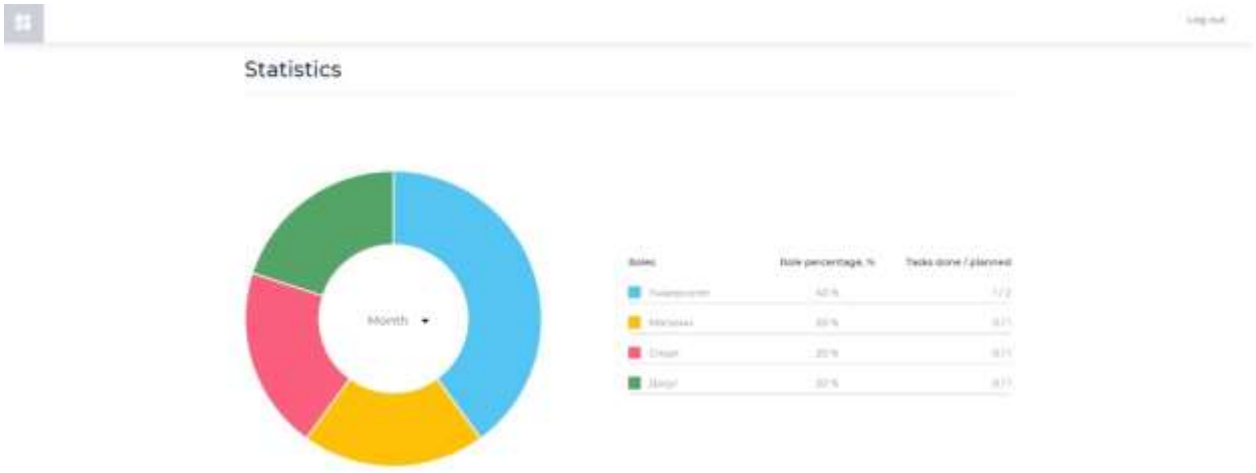

Рис. 3.10. Сторінка зі статистикою по обліковому запису юзера

РОЗДІЛ 4
ТЕСТУВАННЯ ПРАЦЕЗДАТНОСТІ ЗАСТОСУНКУ

У якості перевірки працездатності розробленого веб-застосунку було реалізовано й проведено декілька тестів, серед яких були unit, manual й automation види тестування.

Спершу, мною було проведено саме ручне(manual) тестування роботи веб-застосунку, метою якого стало виявлення проблем, помилок й дефектів у роботі системи та перевірка відповідності роботи застосунку до зазначених функціональних вимог. Ручне тестування є найбільш примітивним та простим типом тестування ПЗ та може бути проведене без навичок й умінь роботи з інструментами тестування, за умови наявності вільного часу тестувальника. Перш ніж проводити автоматизоване тестування, будь-яка програма обов'язково має бути протестована в ручному режимі для виявлення критичних помилок у роботі застосунку. Мануальне тестування виглядає гарним рішення у зв'язку з тим, що стовідсоткове тестування виключно в автоматичного режимі попросту неможливе[25].

Під час ручного тестування у різних популярних веб-браузерах, веб-застосунок працював безвідмовно, жодних помилок у роботі застосунку не було виявлено. Застосунок адаптований під роботу на ПК та різних мобільних пристроях.

На рисунку 3.11 та рисунку 3.12 можна побачити помилки, які показуються у разі не пройденої валідації правильності введених даних юзера під час процедур реєстрації та авторизації.

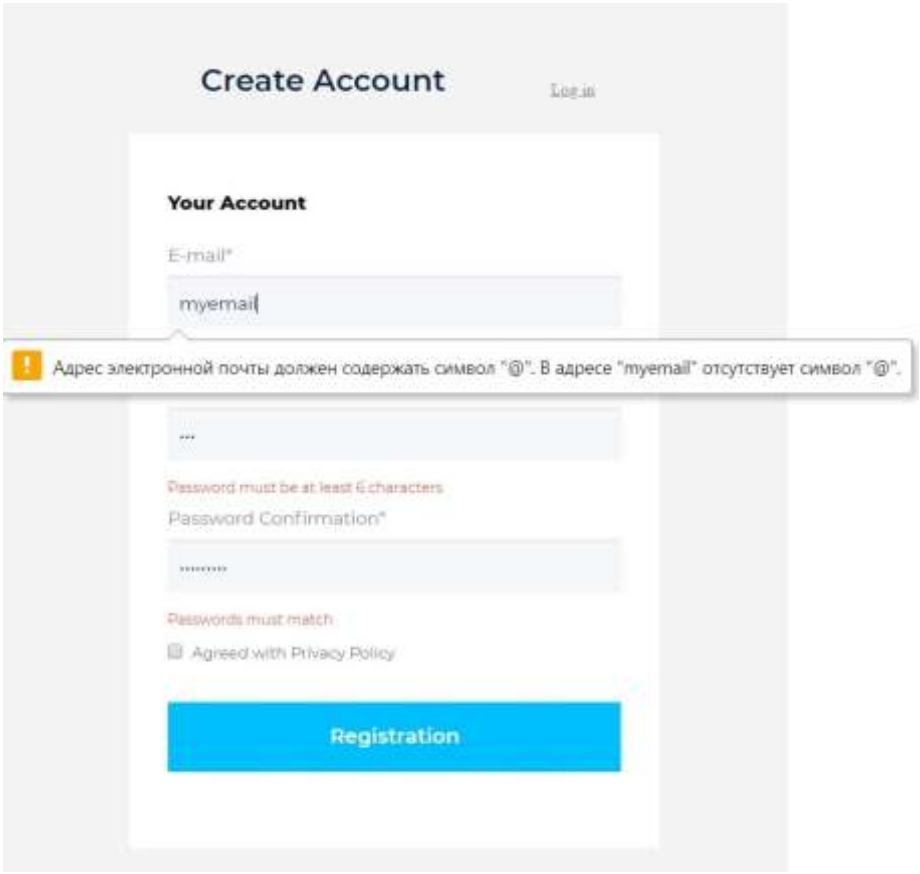

Рис. 3.11. Повідомлення про помилку при реєстрації нового юзера

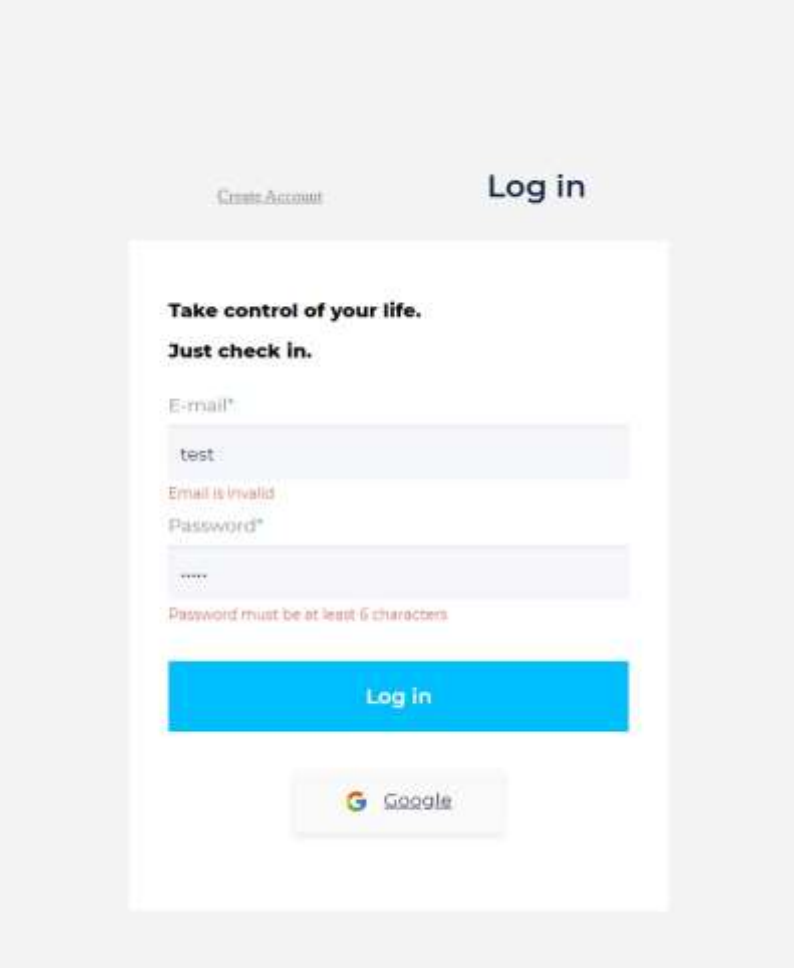

Рис. 3.12. Повідомлення про помилку при авторизації існуючого юзера

У разі вдалого проходження валідації даних юзера з боку клієнту, сервер отримує уведені дані й перевіряє їх відповідність даним у БД та, у разі якщо такого облікового запису не існує, створює новий обліковий запис. На рисунку 3.13 можно побачити помилку, котра показується юзеру, якщо було введено невірний логін або пароль.

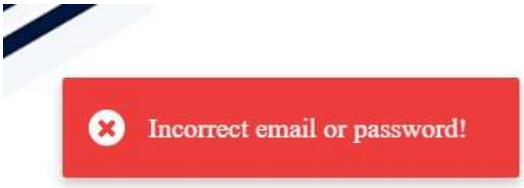

Рис. 3.13. Помилка при введенні невірного логіну чи паролю

Для створення нової цілі юзер має обов’язково задати назву й коротко описати створювану картку. У разі невиконання цього правилу, юзер побачить повідомлення, що зображено на рисунку 3.14.

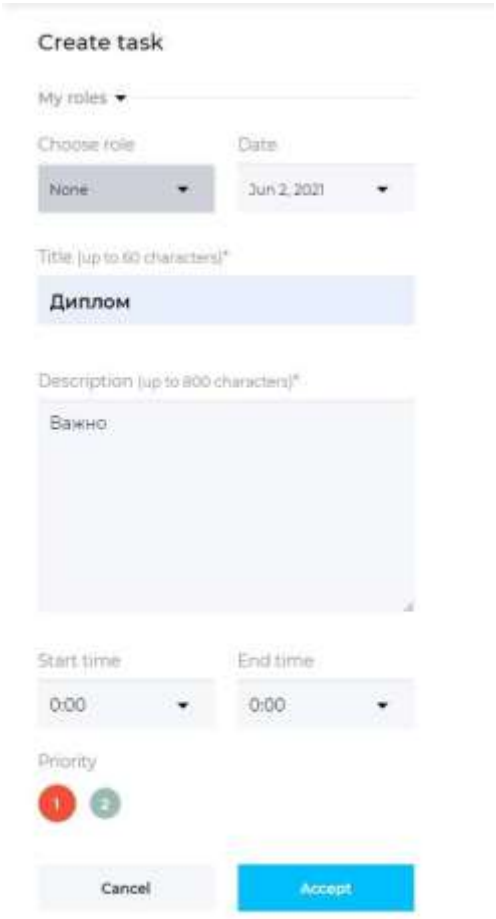

Рис. 3.14. Повідомлення при спробі створення нової задачі

Юзер має обов’язково створювати ролі з коректними й унікальними нахвами. На рисунку 3.15 зображено форму створення нової задачі з відповідним повідомленням.

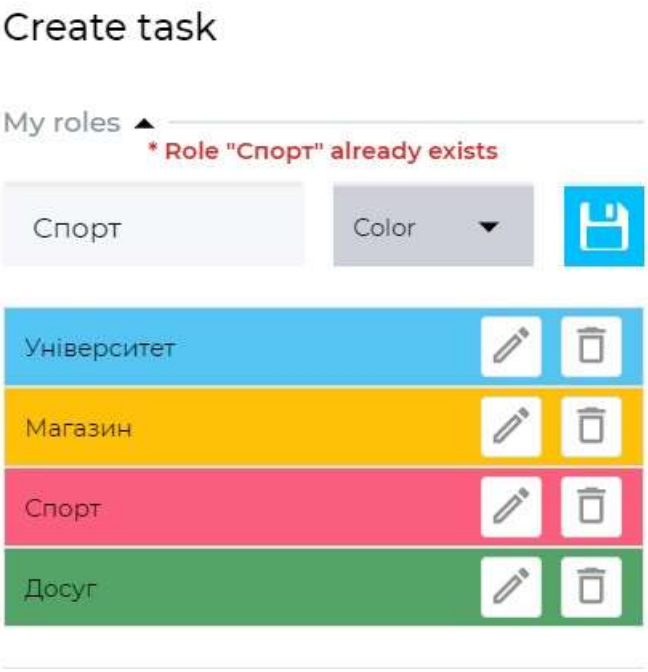

Рис. 3.15. Створення некоректної ролі задачі

Для того, щоб стовідсотково виявити усі нюанси, що пов’язані з помилками та сбоями у роботі веб-застосунку, зазвичай виконується автоматизоване тестування системи.

Автоматизоване(automation) тестування дозволяє зекономити купу часу та є більш ефективним й надійним, аніж ручне, попри те що обидва види грають важливу роль у перевірці працездатності застосунку. Воно забезпечує порівняння отриманого результату з очікуваним, що досягається шляхом написання тестових сценаріїв. Таке тестовування охуплює навіть ті рівні системної архітектури, яких зазвичай не досягає тестування руками. Зокрема, такими є: сховище даних, тести інтерфейсу чи серверу й контролері[26]. Такі програмні рівні швишле й простіше протестувати шляхом автоматизації тесті, ніж виконувати їх мануальним шляхом.

Також, для перевірки працездатності програми були реалізовані декілька unit-тестів, серед яких є: перевірка працездатності API розробленого застосунку, валідація задачі, контроль задачі й валідація юзера.

На рисунку 3.16 зображено вдале проходження тестування створеного веб-застосунку.

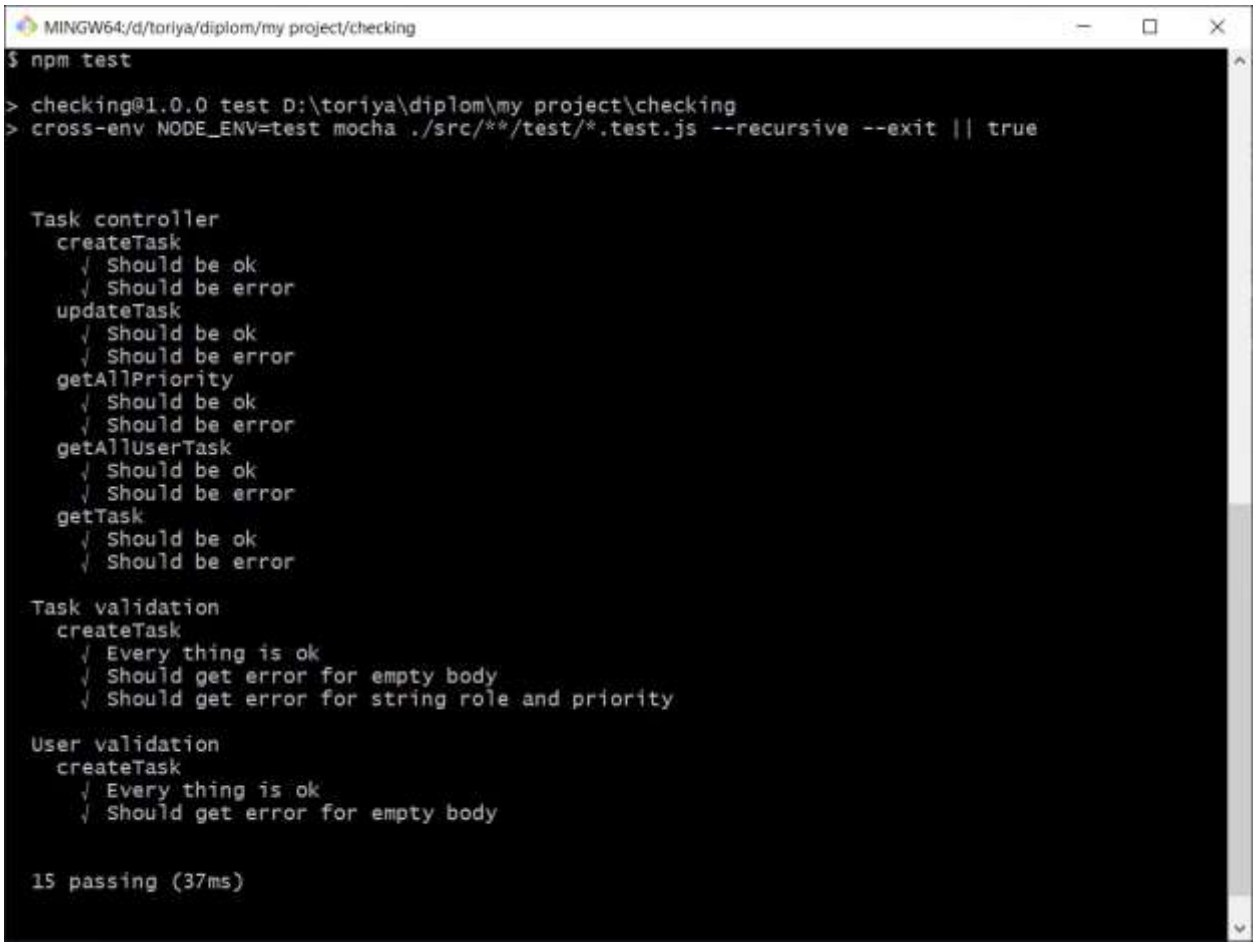

Рис. 3.16. Проходження тестування застосунку

ВИСНОВКИ

У ході виконання випускної бакалаврської роботи була розглянута доцільність використання надсучасної технології розробки веб-додатків PWA. У результаті проведених досліджень й аналізу стану сучасного ринку IT-послуг вдалося встановити ряд існуючих проблем, котрі можуть бути вирішені за допомогою використання технології PWA.

На основі розглянутих архітектурних моделей та інструментів реалізації веб-застосунків, було вирішено спроектувати застосунок на основі клієнт-серверної архітектури, з використанням нереляційної NoSQL бази даних. Важливою частиною роботи стало визначення функціональних вимог до проектованого застосунку, після чого було описано вимоги до клієнтської та серверної частин.

В ході виконання роботи було проведено:

- аналіз сучасних програм-конкурентів;
- дослідження й аналіз моделі клієнт-сервер;
- порівняльний аналіз нереляційного та реляційного підходів до організації сховища збереження даних;
- аналіз роботи рівнів клієнт-серверної архітектури;
- огляд загальної класифікації архітектур веб-застосунків;
- порівняльний аналіз актуальних архітектур побудови клієнтської частини веб-застосунків й у тому числі PWA;
- проектування застосунку й бази даних, кінцевим результатом чого стали UML діаграми та схема БД;
- розробку веб-застосунку на базі PWA;
- ручне та автоматизоване види тестування й юніт-тестування дрібних частин додатку.

Основою роботи програми став наступний стек технологій:

- JavaScript;
- Node.js;
- ReactJS;
- NoSQL БД MongoDB.

Кінцевим результатом виконання дипломної роботи являється веб-застосунок на базі технології PWA, котрий можна застосовувати в якості рішення для персонального планування задач. В порівнянні з аналогічними програмними рішеннями, застосунок має вагомі переваги, серед яких є:

- можливість частково працювати в умовах відсутньої інтернет мережі, за допомогою кешування;
- легковажність застосунку, яка дозволяє миттєво встановлювати застосунок на будь-який мобільний пристрій без потреби у завантаженні додатку;
- можливість використання протоколу захищеної передачі даних HTTPS.

Людський час є найдорожчим й неймовірно обмеженим серед усіх людських ресурсів, що змушує людей берегти його й користуватися ним раціонально для досягнення життєвих цілей. Тяга людей до економії й грамотного керування своїм часом робить застосунок актуальним. Щодо можливостей розвитку розробленого застосунку, то його можна легко масштабувати, що дозволяє додавати необхідний користувачу функціонал за мінімальний час.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Планувальник задач «RememberTheMilk». – [Електронний ресурс]. – Режим доступу: https://www.rememberthemilk.com/

2. Планувальник задач «TodoList». – [Електронний ресурс]. – Режим доступу: https://todoist.com/

3. Планувальник задач «Anydo». – [Електронний ресурс]. – Режим доступу: https://www.any.do/

4. Планувальник задач «To-do.Microsoft». – [Електронний ресурс]. – Режим доступу: https://to-do.microsoft.com/

5. Клієнт-серверна архітектура - GCSE. – [Електронний ресурс]. – Режим доступу: https://teachcomputerscience.com/client-server-architecture/

6. Архітектура клієнт-сервер – CIO Wiki. – [Електронний ресурс]. – Режим доступу: https://cio-wiki.org/wiki/Client_Server_Architecture

7. Клієнт-серверна модель – Oracle Center. – [Електронний ресурс]. – Режим доступу: https://docs.oracle.com/cd/E13203_01/tuxedo/tux71/html/intbas3.htm

8. Модель клієнт-сервер – Javatpoint. – [Електронний ресурс]. – Режим доступу: https://www.javatpoint.com/computer-network-client-and-server-model

9. SQL та NoSQL бази даних – Redgate Software. – [Електронний ресурс]. – Режим доступу: https://www.red-gate.com/simple-talk/databases/nosql/how-to-choose-between-sql-and-nosql-databases/

10. Різниця між SQL та NOSQL - Imaginary Cloud. – [Електронний ресурс]. – Режим доступу: https://www.imaginarycloud.com/blog/sql-vs-nosql/

11. Архітектура Веб-додатку - Web and Mobile App Development Blog. Technology News & Updates. – [Електронний ресурс]. – Режим доступу: https://www.mindinventory.com/blog/web-application-architecture/

12. Архітектура веб-застосунку – Lanars Company. – [Електронний ресурс]. – Режим доступу: https://lanars.com/blog/web-application-architecture-101

13. Архітектура додатку - GeeksforGeeks A computer science portal for geeks. – [Електронний ресурс]. – Режим доступу: https://www.geeksforgeeks.org/how-web-works-web-application-architecture-for-beginners/

14. Архітектури веб-застосунків - Software Development Company. – [Електронний ресурс]. – Режим доступу: Intellectsoft US https://www.intellectsoft.net/blog/web-application-architecture/

15. Типи веб-архітектур - Web, Mobile Application & Software Development Company IT Solutions Provider Blog. – [Електронний ресурс]. – Режим доступу: https://www.hiddenbrains.com/blog/types-of-web-applications-architecture-and-components.html

16. Архітектура веб-застосунку та її типи - Brainvire.com Website Development, Mobile Apps & Digital Marketing Blog. – [Електронний ресурс]. – Режим доступу: https://www.brainvire.com/blog/web-application-architecture-and-its-types/

17. Архітектура веб-додатку - Crowdbotics Blog. – [Електронний ресурс]. – Режим доступу: https://blog.crowdbotics.com/how-to-choose-the-best-architecture-for-your-web-application/

18. Progressive Web App – MDN Web Docs. – [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps

19. Порівняння PWA та Native App - Medium – Where good ideas find you. – [Електронний ресурс]. – Режим доступу: https://medium.com/the-react-native-log/progressive-web-apps-vs-native-apps-which-one-is-the-best-for-your-business-4786822b9ee3

20. Поняття діаграми послідовності - Online Diagram Software Lucidchart. – [Електронний ресурс]. – Режим доступу: https://www.lucidchart.com/pages/uml-sequence-diagram

21. Поняття діаграми прецедентів - Ideal Modeling & Diagramming Tool for Agile Team Collaboration. – [Електронний ресурс]. – Режим доступу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/

22. NodeJS та Express технології – MDN Web Docs. – [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction

23. Поняття MongoDB - Meet Guru99 - Free Training Tutorials & Video for IT Courses. – [Електронний ресурс]. – Режим доступу: https://www.guru99.com/what-is-mongodb.html

24. ReactDOM технологія - GeeksforGeeks A computer science portal for geeks. – [Електронний ресурс]. – Режим доступу: https://www.geeksforgeeks.org/reactjs-reactdom

25. Види тестування застосунку - Meet Guru99 - Free Training Tutorials & Video for IT Courses. – [Електронний ресурс]. – Режим доступу: https://www.guru99.com/difference-automated-vs-manual-testing.html

ДОДАТОК А

```
const {Router} = require('express');
const passport = require('passport');
const {
  loginUser,
  loginGoogle,
  registerUser,
  createRole,
  deleteRole,
  updateRole,
  getAllUserRoles,
  logout,
} = require('./controller');
const validation = require('./validation');
const auth = require('../../middlewares/auth');

const route = Router();

route.post('/login', validation.registration, loginUser);
route.post('/register', validation.registration, registerUser);
route.get('/logout', auth.checkAuth, logout);

// Google
route.get(
  '/google',
  passport.authenticate('google', {scope: ['profile', 'email',
'openid']}),
);
route.post('/roles', auth.checkAuth, validation.createRole,
createRole);
route.delete('/roles/:id', auth.checkAuth, deleteRole);
route.put('/roles/:id', auth.checkAuth, updateRole);
route.get('/roles', auth.checkAuth, getAllUserRoles);

module.exports = route;
```

Лістинг А.1 – Маршрутизація юзера

```
const {Router} = require('express');
const validate = require('./validation');

const route = Router();
const taskController = require('./controller');

route.get('/', taskController.getAllUserTask);
route.post('/', validate.createTask, taskController.createTask);
route.get('/priority', taskController.getAllPriority);
route.put('/:id', validate.updateTask,
taskController.updateTask);
route.get('/:id', taskController.getTask);
```

```
route.delete('/:id', taskController.deleteTask);

module.exports = route;
```

Лістинг А.2 – Маршрутизація завдань

```
{
  "short_name": "Checking",
  "name": "Checking",
  "icons": [
    {
      "src": "favicon.ico",
      "sizes": "64x64 32x32 24x24 16x16",
      "type": "image/x-icon"
    },
    {
      "src": "img192.png",
      "type": "image/png",
      "sizes": "192x192",
      "purpose": "any maskable"

    },
    {
      "src": "img512.png",
      "type": "image/png",
      "sizes": "512x512"
    }
  ],
  "start_url": ".",
  "scope": ".",
  "display": "standalone",
  "theme_color": "#000000",
  "background_color": "#ffffff"
}
```

Лістинг А.3 – Розроблений для PWA Manifest.json

```
let CACHE_NAME = 'checking-cache';
let urlsToCache = [
  '/',
  '/main',
  '/completed'
];

self.addEventListener('install', function(event) {
  var indexPage = new Request('index.html');
  event.waitUntil(
    fetch(indexPage).then(function(response) {
      return caches.open('pwabuilder-
offline').then(function(cache) {
        console.log(
          '[PWA Builder] Cached index page during Install' +
response.url,
```

```
        );
        return cache.put(indexPage, response);
      });
    }),
  );
});

self.addEventListener('fetch', async event => {
  if (event.request.url.includes('/api')) {
    event.waitUntil(
      (async function() {
        const cache = await caches.open('mygame-dynamic');
        await cache.add('/leaderboard.json');
      })(),
    );
  }
});

self.addEventListener('fetch', function(event) {
  var updateCache = function(request) {
    return caches.open('pwabuilder-
offline').then(function(cache) {
      return fetch(request).then(function(response) {
        console.log('[PWA Builder] add page to offline' +
response.url);
        return cache.put(request, response);
      });
    });
  };
  event.waitUntil(updateCache(event.request));
  event.respondWith(
    fetch(event.request).catch(function(error) {
      console.log(
        '[PWA Builder] Network request Failed. Serving content
from cache: ' +
          error,
      );
      return caches.open('pwabuilder-
offline').then(function(cache) {
        return
cache.match(event.request).then(function(matching) {
          var report =
            !matching || matching.status == 404
              ? Promise.reject('no-match')
              : matching;
          return report;
        });
      });
    }),
  );
});
```

Лістинг А.4 – Розроблений для PWA Service Worker