ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

Пояснювальна записка
до магістерської роботи
на ступінь вищої освіти магістр
на тему: «**УДОСКОНАЛЕННЯ МЕТОДИКИ WEB-РОЗРОБКИ ЗА ДОПОМОГОЮ ТЕХНОЛОГІЙ PWA ТА БАГАТОПОТОКОВОСТІ**»

Виконав: студент 6 курсу, групи ПДМ-61
спеціальності:
121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

Петрухно Т. О.
(прізвище та ініціали)

Керівник Жебка В. В.
(прізвище та ініціали)

Рецензент
(прізвище та ініціали)

Нормоконтроль Трінтіна Н.А.
(прізвище та ініціали)

Київ – 2021

1. ТЕОРЕТИЧНА ЧАСТИНА

1.1 Проблематика

Проблематика web-розробки, полягає в ресурсах які необхідно витратити на розробку, від вибору конкретних практик та технологій залежить багато показників. Тому удосконаливши методику web-розробки, можна оптимізувати витрати на виробництво та подальшу підтримку програмного продукту в майбутньому.

Найбільшим замовником IT, послуг в сучасному світі є бізнес, тобто підприємницька, комерційна або будь яка інша діяльність, спрямована на отримання прибутку. Практика показує що, зазвичай бізнес реалізує вже добре відпрацьований стандарт. Створюється офіційний web-сайт, який є представництвом, джерелом реклами, і каналом комунікації з конкретним бізнесом.

Але сайт має ряд обмежень, це відсутність можливості взаємодіяти з девайсом і операційною системою клієнта, такими можливостями характеризується нативний застосунок. Другим важливою вимогою є наявність інтернету, часто поза межами міст та населених пунктів, підключення до глобальної мережі інтернет відсутнє, або дуже слабе з'єднання, що унеможливить, користуватись web-сайтом. Крім того, для нових відвідувачів сайт, це не завжди зручно. Новому відвідувачу треба з першого разу запам'ятати його назву, що б пізніше, ввести його назву в пошук, щоб знову його знайти. Як показують дослідження якщо ресурс завантажуються більше 3 секунд, то більше 50% користувачів покидають цей сайт не дочекавшись повного завантаження.

Вихід з цієї ситуації, на перший погляд очевидний, треба створити застосунок, точніше кілька під різні платформи. Тепер треба визначити основні популярні операційних системі, це IOS, Android, Windows. Вони матимуть можливість працювати офлайн, і будуть зручно розміщені під рукою, в постійному полі зору, на робочому столі.

Виникає наступна проблема, це складність впровадження рішення, так як під IOS застосунки створюються мовою SWIFT, під Android як правило Java, під Windows це може бути C#, C++, Pyton або та ж Java і на додаток створення web-сайту на JavaScript. Таке рішення виливається в збільшення штату, тому що під кожну мову програмування потрібне конкретний кваліфікований спеціаліст, якщо місто невелике можна зіткнутися з проблемою дифіциту кадрів.

Щоб розробити web-сайт, і потрібну кількість застосунків під основні популярні платформи (Window, Android, IOS, браузери Chrome, FireFox, Mozila), треба витратити значно більше грошей, ніж це передбачалось спочатку на web-сайт понад те розробка займе більше часу, враховуючи таку кількість платформ.[13]

Стосовно мобільних застосунків, а точніше їх розповсюдження. Для того, щоб мобільний застосунок почати широко розповсюджувати його треба розмістити на майданчику продажів мобільних застосунків. Розроблений застосунок повинен відповідати всім умовам та притримуватись правил і політик, які встановлюють і модерують ці майданчики продажів мобільних застосунків. Це створює додаткову

залежність, що для власника застосунка, не є добре. Для прикладу в 2020 році в світі спостерігається технологічні перегони маж Китаєм та Сполученими Штатами Америки, і в Сполучених Штатах Америки був заборонений найпопулярніший застосунок соціальна мережа для молоді «TikTok» китайського виробництва.[14][15]

Розроблений застосунок можуть просто відхилити від розміщення на майданчику продажів застосунків, або пізніше в будь який момент видалити його з каталогу продажів, якщо порушиться хоча б одна з вимог політик цього майданчику продажів застосунків. Більшість мобільних пристроїв в світі, це пристрої, що працюють під операційною системою «Androind OS», найбільший виробник таких пристроїв корейська компанія «Samsung», на цих пристроях майданчик продажів застосунків називається «Play Market», китайські виробники, наприклад Xiaomi, створюють власні майданчики продажів застосунків, хоч пристрої і працюють під управлінням операційної системи «Androind OS», але майданчик продажів застосунків носить назву «Mi Store». Для пристроїв, які використовують операційну систему «IOS», пристрої американського виробництва компанії «Apple» цей майданчик продажів застосунків, носить назву «App Store».

Майданчики продажів застосунків працюють в різних країнах з різним державним устроєм та законодавством, тому не важко, зробити помилку. Що в одній країні може бути дозволено, в іншій країні може бути під забороною, тим самим легко порушити одну з умов та політик користування конкретним майданчиком продажів застосунків.

При розміщені застосунку, в «App Store», він натрапляє знову на витрати, тому, що за можливість розміщувати застосунки на майданчику продажів застосунків «App Store» також встановлена плата, яку доведеться здійснити, плата становить дев’яносто дев’ять американських доларів за рік.

Підбивши ґрунтовні підсумки з всього вище зазначеного, можна зробити висновки, що традиційний метод web-розробки не дає очікуваного результату, навіть якщо додатково розробити застосунки на основні платформи. Це приводить до значного збільшення зусиль, витрат коштів, витрат часу, та призводить до додаткової небажаної залежності.

1.2 Постановка завдання дослідження

Основним завданням дипломної роботи, є удосконалення методики web-розробки за допомогою поєднання потрібних інноваційних Web API, , що в сумі, дозволяють створити web-застосунок, який буде гібридом між web-сайтом та застосунком, В процесі дослідження були відібрані необхідні технології та практики а саме технології pwa та багатопотоковість.

З викладеної проблематики, в частині «Проблематика», випливає те, що звичайний сайт, і окремо розроблені застосунки під кожну платформу це, довгий, дорогий, і важкий шлях. Крім того залишаються невирішені проблеми залежності. Web-сайт приводить до прямої залежності, від інтернету, а точніше, його наявності, а при наявності, його швидкості. Застосунок приводить до безпосередньої

залежності від політики майданчику продажів застосунків, який має розповсюджувати цей застосунок.

1.3 Сутність «Удосконалення методики web-розробки за допомогою технологій PWA та багатопотоковості».

Спираючись на дану проблематику, і її всеосяжне дослідження, запропоновано удосконалити методику web-розробки. Що має на меті створити гібрида між web-сайтом і застосунком, поєднавши плюси кожного варіанту за для створення прогресивного web-застосунку з відсутністю цих проблем. Технічно це є web-сайт і залишає всі свої найкращі риси, але в той самий час, набуває можливостей притаманних для нативного застосунку. Тобто коли вигідно або зручно працювати з web-сайтом, прогресивний web-застосунок працює в режимі web-сайту, а коли виникає потреба працювати, з застосунком то тоді прогресивний web-застосунок працює в режимі нативного застосунку.

Удосконалена методика web-розробки за допомогою pwa та багатопотоковості, це набір стратегій мислення, використовуючи найсучасніші браузерні WEB API, які допомагають створювати можливості, які в web раніше ніколи не було. Впровадження методики відбувається на стороні front-end, тобто на клієнтській частині. А тепер більш ґрунтовно.

Створений прогресивний web-застосунок, отримує можливість працювати як web-сайт, в тому числі і в режимі офлайн, при повній втраті зв'язку з всесвітньою мережею інтернет. А також за потреби і в режимі нативного застосунку, тобто завантажуватись на пристрій, з своєю особистою назвою та іконкою на робочому столі, і відкриватись в своєму особистому вікні. Мати при цьому високу швидкодію за рахунок багатопоточності та стратегій кешування.

Для реалізації задуму першим етапом треба використати Service Worker. Service worker це скрипт написаний мовою JS який виконується браузером в фоновому процесі. Він представляє з себе proxy сервер між web-застосунком і мережею інтернет.

Його робота можлива лише при використанні захищеного протоколу HTTPS, або звичайний HTTP у випадку коли web-застосунок запускається на локальному сервері localhost. Так само service worker, має і обмеження, він не може працювати з DOM(document object model) деревом, ним ми керуємо з основного потоку. Використання service worker-а, дозволяє перехоплювати інтернет запити з клієнта до серверу, і обробляти їх потрібним, для виконання цілі чином. Запит з клієнтської частини, не встигнувши потрапити в мережу інтернет перехоплюється service worker-ом, а далі в залежності від запрограмованої логіки реалізується його обробка, він може бути відправлений в мережу інтернет, або дістати відповідь для очікуючого запиту з кешу, або зробити ці дії одночасно, в залежності від реалізованої стратегії кешування.

Існує п'ять основних стратегій кешування, далі ми їх розглянемо детальніше:

Перша стратегія кешування під назвою «cache-first» або «спершу кеш», суть якої полягає в тому, що service worker перехоплює запит з клієнта на сервер, і спочатку перевіряє сховище кешу «Cache Storage», якщо за цим запитом є

закешовані дані, то в такому випадку service worker відповідає цими даними з сховища кешу «Cache Storage» на запит з клієнта, не надсилаючи при цьому запит мережею інтернет до серверу. Якщо в сховищі кешу «Cache Storage» дані за надісланим запитом відсутні, тоді service worker відправляє запит мережею інтернет до серверу, що б отримати потрібні дані, отримавши дані, кешує їх в сховищі кешу «Cache Storage» і відправляє відповідь на запит з клієнта. Якщо сховище кешу «Cache Storage» порожнє, і підключення до інтернету відсутнє, і не передбачено ніяких fallback заглушок, то service worker не зможе ніяк обробити запит. Слід використовувати цю стратегію для таких ресурсів, як зображення, відео, CSS і та інших файлів, яких не є критичними для web-застосунку. Стратегія кешування представлена на рис. 1.1

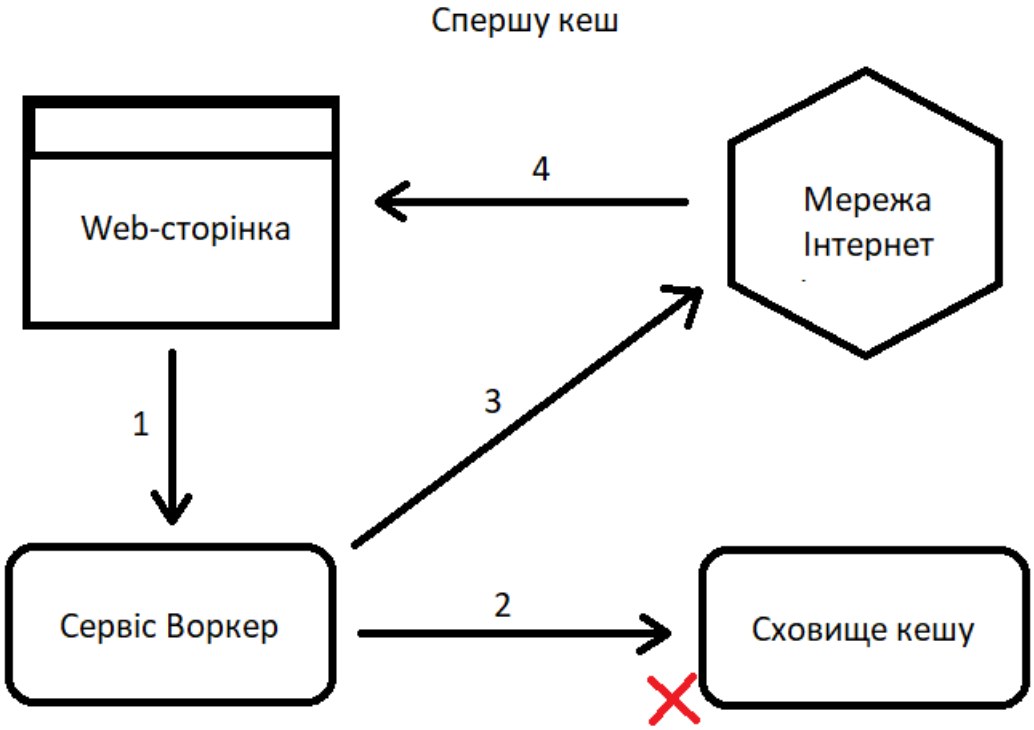

Рисунок 1.1 – Стратегія кешування «спершу кеш»

Друга стратегія кешування під назвою «network-first» або «спершу мережа інтернет», суть якої полягає в тому, що service worker перехоплює запит з клієнта на сервер, і зразу відправляє цей запит в мережу інтернет, намагаючи отримати відповідь від віддаленого сервера, в разі наявності інтернету і успішної відповіді сервера, данні відповіді на запит кешуються в сховищі кешу «Cache Storage» і паралельно відправляються, відповіддю на запит який виходив з клієнта. Якщо ж інтернет відсутній, чи не вдається отримати дані від сервера з якоїсь причини, service worker іде в сховище кешу «Cache Storage» і якщо за цим запитом є закешовані дані, то в такому випадку service worker відповідає цими даними з сховища кешу «Cache Storage» на запит з клієнта. Використовують цю стратегію

кешування для запитів, які часто оновлюються, особливо для запитів POST. Користувачі в мережі отримують найбільш актуальний контент, а користувачі в офлайновому режимі отримують кешовану версію. Стратегія кешування представлена на рис. 1.2

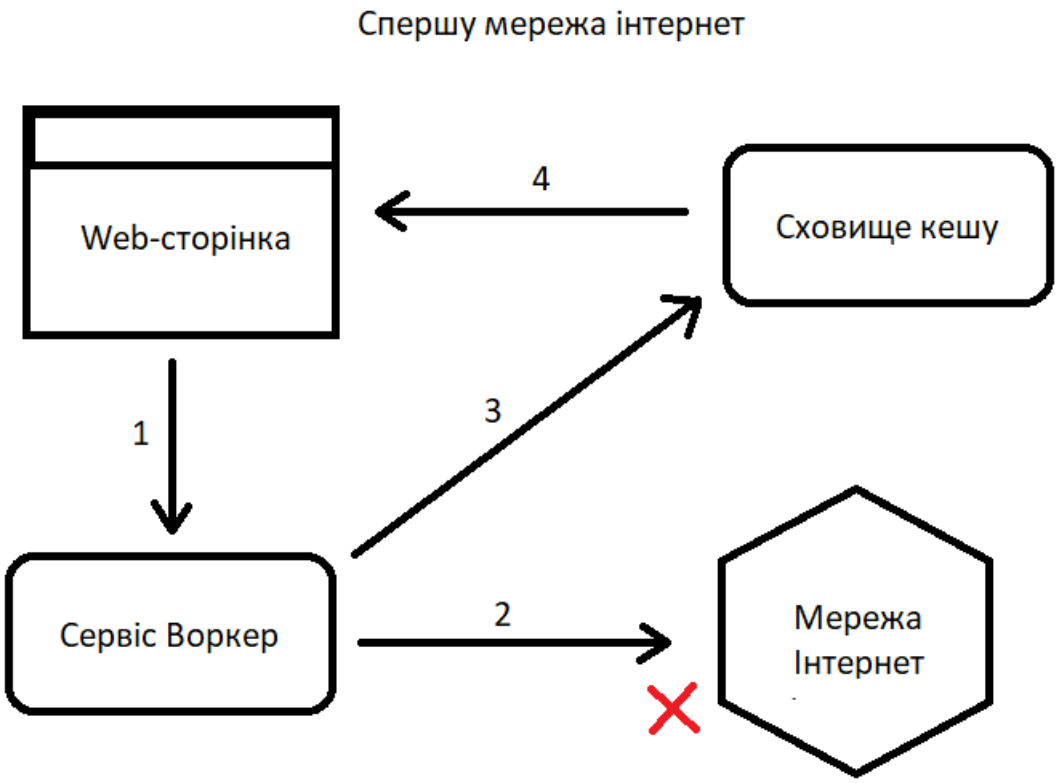

Рисунок 1.2 – Стратегія кешування «спершу мережа інтернет»

Третя стратегія кешування під назвою «cache-only» або «тільки кеш», суть якої полягає в тому, що service worker перехоплює запит з клієнта на сервер, і зразу перевіряє сховище кешу «Cache Storage». Якщо за цим запитом є закешовані дані, то в такому випадку service worker відповідає цими даними з сховища кешу «Cache Storage» на запит з клієнта, не надсилаючи при цьому запит мережею інтернет до серверу. Ця стратегія кешування зовсім не використовує мережу інтернет для обробки запиту з клієнта, а завжди відповідає даними з сховища кешу «Cache Storage», якщо вони там присутні. Використовується для даних які не змінюються з часом, тому вони кешуються всього один раз з самого початку. Якщо сховище кешу «Cache Storage» порожнє, service worker не зможе відповісти на цей запит. Стратегія кешування представлена на рис. 1.3

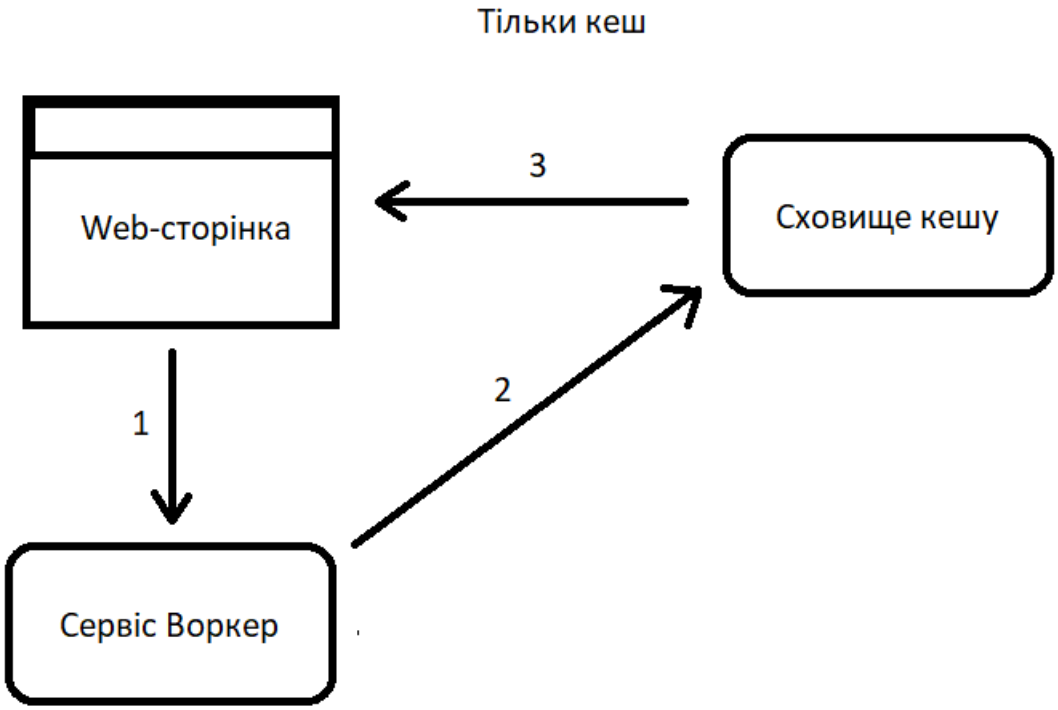

Рисунок 1.3 – Стратегія кешування «тільки кеш»

Четверта стратегія кешування під назвою «network-only» або «тільки мережа інтернет», суть якої полягає в тому, що service worker перехоплює запит з клієнта на сервер, і зразу відправляє його мережею інтернет на сервер, не звертаючись при цьому до сховище кешу «Cache Storage». Таким чином service worker працює в прозорому режимі. Ця стратегія використовується для файлів і ресурсів, які часто змінюються. Стратегія кешування представлена на рис. 1.4

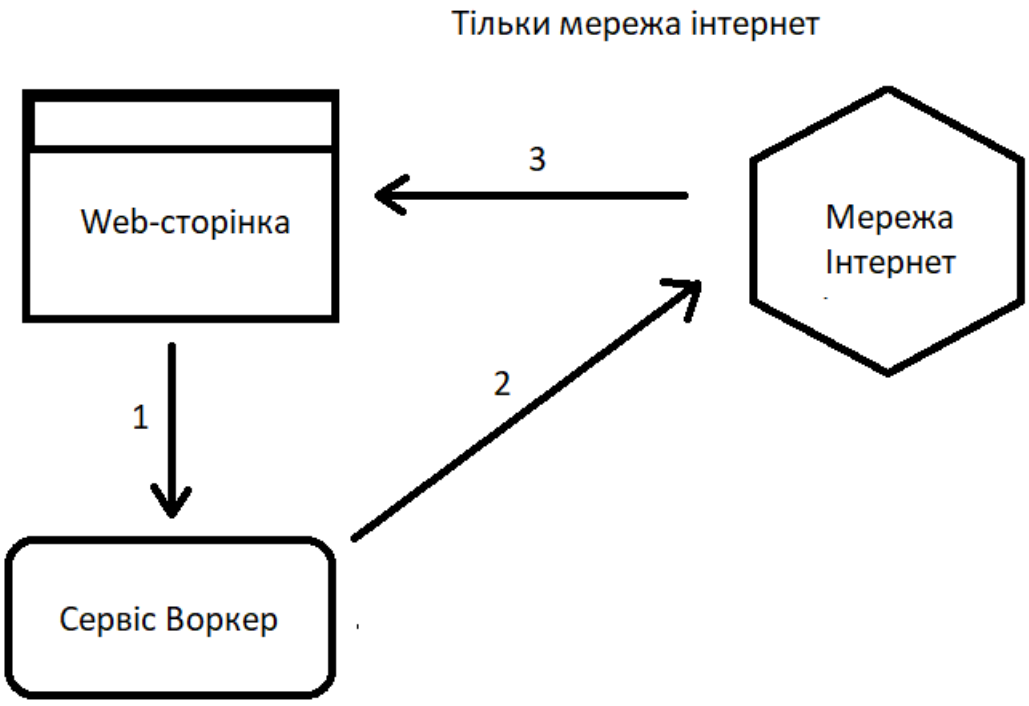

Рисунок 1.4 – Стратегія кешування «тільки мережа інтернет»

П’ята стратегія кешування під назвою «stale-while-revalidate» або «попередні дані під час оновлення», суть якої полягає в тому, що service worker перехоплює запит з клієнта на сервер, і спочатку перевіряє сховище кешу «Cache Storage», якщо за цим запитом є закешовані дані, то в такому випадку service worker відповідає цими даними з сховища кешу «Cache Storage» на запит з клієнта, паралельно надсилаючи при цьому запит мережею інтернет до сервера, що б отриману відповідь помістити в сховище кешу «Cache Storage», там самим оновивши його для наступного запиту. Тобто, кожен раз ми отримуємо відповідь з миттєвою швидкістю тільки кешовані від попереднього запиту «несвіжі» і паралельно підготовлюємо останні актуальні дані для наступної видачі. Якщо в сховищі кешу «Cache Storage» дані за надісланим запитом відсутні, тоді service worker відправляє запит мережею інтернет до серверу, що б отримати потрібні дані, отримавши дані, кешує їх в сховищі кешу «Cache Storage» і відправляє відповідь на запит з клієнта. Якщо сховище кешу «Cache Storage» порожнє, і підключення до інтернету відсутнє, і не передбачено ніяких fallback заглушок, то service worker не зможе ніяк обробити запит. Слід використовують цю стратегію для високопріоритетних і критичних файлів. А також, для (не GET запитів), таких як POST запити. Стратегія кешування представлена на рис. 1.5

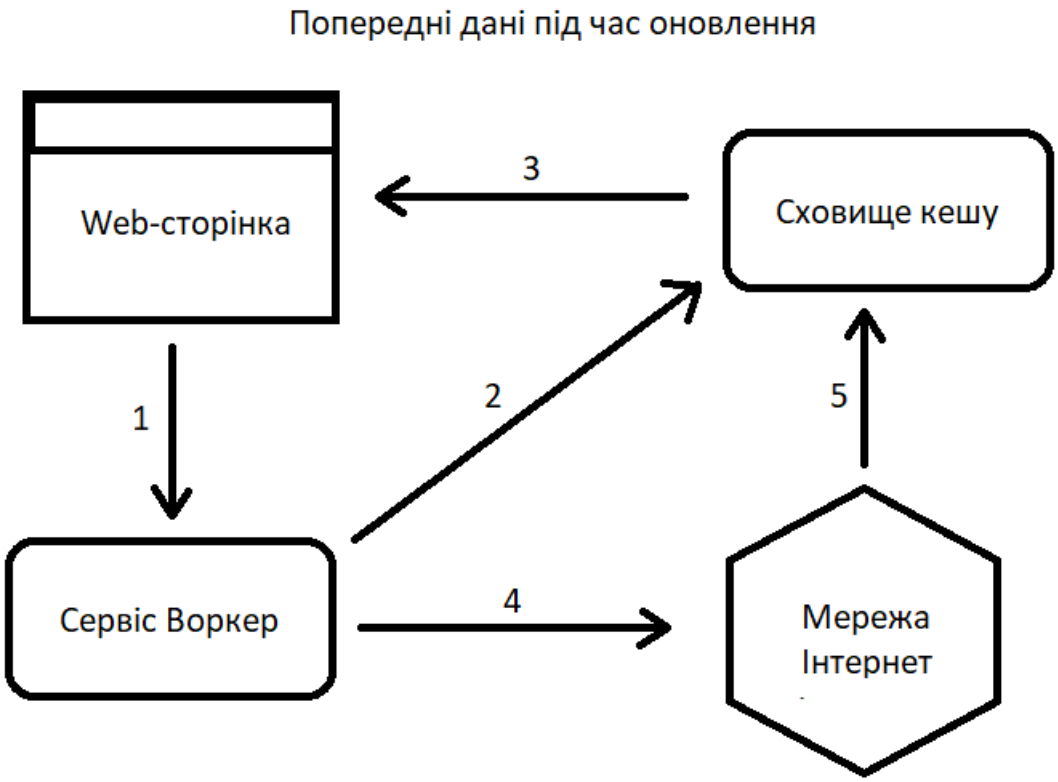

Рисунок 1.5 – Стратегія кешування «попередні дані під час оновлення»

Описані вище п’ять основних стратегії кешування даних, використовуються для збереження даних відповіді з сервера в сховище кешу на клієнті «Cache Storage», передбачені для запитів з клієнта, на віддалений сервер з методом запиту «GET».

Для запитів з клієнта, на віддалений сервер з іншими методами запиту («POST», «PUT», «DELETE», «UPDATE»), збереження даних відповіді з сервера в сховище кешу на клієнті «Cache Storage» не передбачено, для таких випадків реалізуються наступні описані нижче сценарії.

Якщо з’єднання з мережею інтернет відсутнє, і відбуваються запити з клієнта, на віддалений сервер з будь яким із перелічених методів запиту («POST», «PUT», «DELETE», «UPDATE»), то в першу чергу, користувач може бути повідомлений про те, що робота відбувається в режимі відсутності підключення до інтернету, і його запит, буде обов’язково відправлений на віддалений сервер, як тільки відновиться з’єднання з мережею інтернет, Інформування може відбуватись стандартними методами JavaScript BOM(Browser Object Model) методом alert(), що є погано з точки зору користувацького інтерфейсу, кращою практикою буде використовувати сам інтерфейс web-застосунку. Далі треба зберегти дані запиту в сховище браузера з унікальними ідентифікаторами, для того, щоб при відновлені інтернету ми могли цей запит надіслати, цим сховищем може слугувати IndexedDB або LocalStorage. Збереження даних запиту відбулось, і як тільки користувач повернеться до мережі, ці дані поміщаються до прихованої форми на HTML

сторінці, і форма надсилається а, записи щодо цих запитів в сховищі IndexedDB або LocalStorage мають бути видалені.

Таким чином реалізується підтримка запитів («POST», «PUT», «DELETE», «UPDATE») в режимі відсутності підключення до інтернет мережі.

Якщо ж час відправлення запиту з одним з методів запитів («POST», «PUT», «DELETE», «UPDATE») не може бути відкладений, тоді service worker віддає на клієнт звичайну fallback заглушку, що інформує клієнта про те, що такий запит не може бути відкладеним, що б клієнт здійснив його при відновленні інтернет з'єднання самостійно.

За таких умов наш web-застосунок не падає, як повела б себе типова web-сторінка, при роз'єднанні з мережею інтернет, таким чином забезпечується безперебійність роботи web-застосунок, в без залежності від наявності підключення до мережі інтернет.

Наступним кроком прописується web manifest, який являє собою JS файл, який містить об'єкт JSON всередині. Цей файл вміщає основні параметри, які характеризують web-застосунок, як нативний застосунок, такі параметри як: ім'я застосунку, спосіб відображення застосунку, орієнтація застосунку, загальний опис застосунку, зазначаються опис іконок застосунку і їх розміри під різні роздільні здатності, системний колір застосунку та інше.

При впроваджені Service Worker-и та web manifest, реалізуємий web-застосунок отримує нову можливість, бути завантаженим на настільний чи мобільний пристрій при чому минаючи майданчики продажів застосунків по типу «Play Market», «Mi Store», «App Store».

Таким чином реалізуємий застосунок позбавляється від проблеми залежності та контролю зі сторони майданчиків продажів застосунків, одночасно уникаючи витрати на розміщення цього web-застосунку на цих майданчиках розповсюдження застосунків.

Для розширення можливостей web-сайту, який поступово перетворюється з web-сайту в web-застосунок використовуємо Web API, які дають змогу взаємодіяти з девайсом, а точніше з його апаратним та/або програмним забезпеченням. Наприклад для взаємодії з контактною книжкою на пристрої застосовується Contact API, для створення спливаючих push повідомлень застосовується Push API, для взаємодії з апаратним модулем bluetooth на пристрої, застосовується Bluetooth API, для взаємодії з геолокацією і апаратним модулем GPS, застосовується Geolocation API[1].

Тобто для різних специфічних задач, пов'язаних з операціями над операційною системою пристрою, або його програмним забезпеченням використовуються відповідні Web API, через які ми можемо реалізувати потрібний для web-застосунку функціонал.

Для того що б пошукова системи, сприймали web-застосунок як progressive web application, коли він працює в режимі web-сайту. Для цього останнім штрихом, необхідно отримати сертифікат криптографії TSL або SSL, який допоможе встановити безпечне з'єднання між клієнтом і сервером, для формування захищеного HTTPS зв'язку. Отримати такий сертифікат, можна безкоштовно, або за символічну суму коштів.

Користувацький інтерфейс створюється традиційними засобами web-розробки, тобто для розмітки сторінки мова розмітки HTML, для створення стилістики, каскадні таблиці стилів CSS. Можуть бути використані збірники проекту по типу (webpack, gulp), препроцесори, постпроцесори, бібліотеки та фреймворки(React, Vue, Angular), що значно спрощує проектування і реалізацію користувацького інтерфейсу. Це є можливим, так як в основі всього лежить web-сайт, який створений за допомогою удосконаленої методики web-розробки з застосуванням технологій PWA та багатопотоковості. В результаті цей web-сайт отримує можливості завантажуватись, та працювати в режимі нативного застосунку, перетворюючись в web-застосунок.

Web-застосунок створюється мовою програмування JavaScript, ця мова по своїй природі є однопоточною. Це значить, що виконання «важких» операцій, для виконання яких потрібен деякий час, блокує потік, так як він всього один. В наслідок цього програма не може виконувати в цей момент інші операції, реагувати на події(наприклад клік або прокрутка), що призводить до недієздатності програми на цей час. Використовуючи Worker’s з Web Worker API, створюється можливість створювати додаткові потокі за рахунок внутрішнього устрою браузера, але з певними обмеженнями. З створених потоків не може бути ініційоване змінення DOM(Document Object Model) дерева, змінювати DOM деремо, можливо лише з основного потоку коду, це обмеження покликане зупинити виникнення конфліктів, коли один елемент DOM дерева, гіпотетично могли одночасно змінити декілька паралельних потоків, що неодмінно спричинило б конфлікт.

Використовуючи ці додаткові потоки для розподілення «важкого» коду, вирішується проблема з блокуванням основного потоку, що беззаперечно позитивно впливає на досвід використання користувачем web-застосунку.

Впроваджуючи вище описані кроки під час web-розробки, розроблений web-застосунок позбавляється зайвих залежностей і надмірних ускладнень процесу розробки та підтримки цього web-застосунку і є удосконаленням методики web-розробки, використовуючи набір технологій PWA та багатопотоковості.

1.4 Вимоги для впровадження підходу

1.4.1 TSL (Transport Layer Security) / SSL (Secure Socket Layer)

SSL «рівень захищених сокетів». Криптографічний протокол, що забезпечує безпечне встановлення з’єднання між клієнтом та сервером. Частіше всього використовують як рівень безпеки в протоколі HTTPS.

Для криптографії використовується асиметричний алгоритм шифрування з відкритим ключем. Тобто за допомогою криптографічного алгоритму генерується приватний і публічний ключ, які математично зв’язані між собою. На обох сторонах з’єднання. Приватний ключ з публічного вирахувати неможливо, хоч вони і математично зв’язані, але використана одностороння функція шифрування.

Наступним кроком відбувається обмін публічних ключів, між один одним, які призначені, для шифрування повідомлень, тобто кожна сторона отримує публічний відкритий ключ іншої. Представлено на рис. 1.6

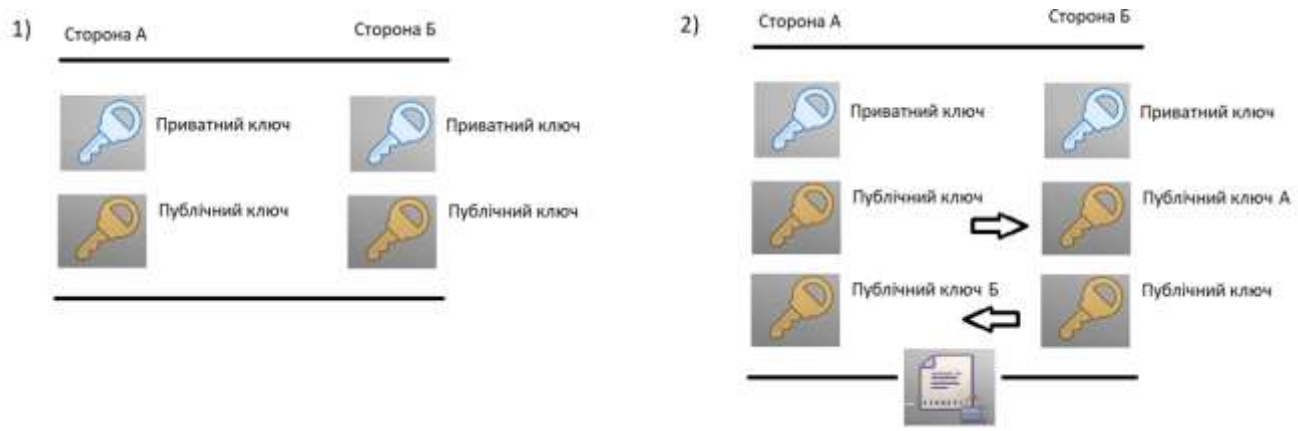

Рисунок 1.6 – Генерація та обмін ключами

Тепер сторона А може зашифрувати своє повідомлення для сторони Б, отриманим публічним ключом від сторони Б, і відправити стороні Б, сторона Б в свою чергу розшифрує це повідомлення своїм приватним ключем. Так само сторона Б може зашифрувати повідомлення відкритим ключем А, отриманим від сторони А на стадії обміну ключів відкритих публічних ключів, зашифрувати ним повідомлення і відправити стороні А, яка використавши свій приватний ключ зможе розшифрувати це повідомлення.

Пісня шифрування чужим відкритим ключем, сам відправник вже не зможе подивитись вміст послання, так як для дешифрування потрібен математично зв'язаний приватний ключ, який надійно зберігається на іншій стороні з'єднання.

Обмін ключами між сторонами може здійснюватися різними методами:

1. Анонімний обмін ключами.
2. Обмін ключами при використанні RSA і аутентифікації.
3. Обмін ключами при використанні протоколу Діффі-Геллмана і аутентифікація.

Також кожне повідомлення перевіряється на цілісність, На кожному рівні повідомлення включають поля для довжини, опису і перевірки. Протокол запису приймає повідомлення, які потрібно передати, фрагментує дані в керовані блоки, розумно стискає дані, застосовуючи MAC (message authentication code), шифрує і передає результат. Отримані дані він розшифровує, перевіряє, розпаковує, збирає і доставляє верхнім рівням клієнта.

Існує чотири протоколи запису

1. Протокол рукостискання (Handshake Protocol).
2. Протокол тривоги (Alert Protocol).
3. Протокол зміни шифру (The Change Cipher Spec Protocol), протокол даних застосунку (Application Data Protocol). Якщо SSL реалізація отримує тип запису, який їй невідомий, то цей запис просто ігнорується.
4. TSL захист на транспортному рівні, є наступником технології SSL, використовується за замовченням в браузерах FireFox і Google Chrome.

Кроки відкриття захищеного сеансу:

1. Клієнт відправляє вітання (client hello) в якому повідомляє серверу про підтримувану ним версію протоколу та список підтримуваних алгоритмів шифрування в порядку їх бажаності. Також це повідомлення має рядок випадкових байтів. Протокол дозволяє зазначати підтримувані клієнтом методи стиснення даних.
2. У відповідь сервер відправляє власне вітання (server hello), в якому називає вибраний ним шифр (CipherSuite) із запропонованого клієнтом переліку, ідентифікатор сеансу (session ID) та рядок випадкових байтів. Також сервер виправляє свій цифровий сертифікат. Якщо сервер потребує цифровий сертифікат клієнта для його автентифікації, то до відповіді додається відповідний запит (. client certificate request) та перелік підтримуваних типів сертифікатів та унікальні імена (Distinguished Name; DN) центрів сертифікації (Certification Authority; CA).
3. Клієнт перевіряє (верифікує) отриманий цифровий сертифікат сервера.
4. Клієнт відправляє у відповідь рядок випадкових байтів зашифровану відкритим ключем сервера. Цей рядок необхідний для шифрування даних при подальшому обміні.
5. Якщо сервер запитав сертифікат клієнта, то до відповіді додається рядок з випадковими байтами, який клієнт шифрує своїм приватним ключем, та цифровий сертифікат клієнта. Якщо клієнт не має сертифіката, то у відповідь відправляється відповідне попередження (no digital certificate alert). Якщо сервер налаштований на обов'язкову автентифікацію клієнтів, то на цьому процедура рукостискання може перерватись.
6. Сервер перевіряє (верифікує) сертифікат клієнта.
7. Клієнт відправляє повідомлення про успішне завершення рукостискання (finished).
8. Сервер відправляє повідомлення про успішне завершення рукостискання, яке зашифроване таємним ключем.
9. Тепер протягом сеансу клієнт та сервер можуть обмінюватись даними, зашифрованими спільним таємним ключем.

На даний час актуальними вважається TSL 1.3 – 2018, RFC 8846, або попередня TSL 1.2 – 2008, RFC 5246. Всі попередні версії TSL 1.1, TSL 1.0, та всі версії SSL вважаються застарілими і ненадійними.[2]

SSL\TSL надає наступні переваги:

1. Підвищуються позиції ранжування web-сайту в пошукових системах;
2. Захищеність даних, під час руху мережею;
3. Розширює можливості, так як наприклад реалізація оплати на web-сайті, можливо реалізувати тільки з захищеним каналом зв’язку;
4. Збільшення довіри до web-сайту.

1.4.2 Service Worker

Найважливішою проблемою як і раніше є відсутність гарного механізму для управління кешем ресурсів і налаштованим мережевими запитами. Технологія Service Worker, це заміна попередньої технології AppCache, яка використовувалась для кешування, технологія дійсно дозволяла просто вказувати ресурси для кешування. Але, технологія AppCache допускає багато припущень про те, що програміст намагається зробити, і потім з гуркотом ламається, коли застосунок працює не в точності з цими припущеннями.

Тому на заміну прийшла технологія Service Worker. Програмно це скрипт, який браузер запускає в фоновому режимі, окремо від web-сторінки. Додає підтримку функції push-повідомлень, функції фонової синхронізації, та функції перехоплення мережевих запитів.

За допомогою Service Workers, можна спроектувати застосунок, який в першу чергу буде використовувати кешовані ресурси, таким чином працюючи за замовченням в автономному режимі, до того моменту як з мережі не будуть отримані нові дані. В такий спосіб зазвичай працюють нативні застосунки, що є критерієм вибору в їх користь, але технологія Service Worker, дає змогу модифікувати web-сайт до рівня нативного застосунку.

Сам Service Worker виконує роль проксі-сервера, що знаходиться між клієнтською частиною і серверною частиною, зі сторони клієнтської частини. Service Worker, частина специфікації Web Worker’s API, тобто є представником фонового завдання, яке виконується в потоці worker, тому не має доступу до DOM(Document Object Model) і працює в окремому власному потоці, окремо від основного потоку JavaScript застосунку, а також є подієвим(реагує на певні події). Так як Service Worker покликаний бути асинхронним, прямий доступ з нього до сховища LocalStorage та сховища SessionStorage, використовувати не вийде, запит поверне відповідь «undefined», тому дані потрібно повернути з worker-а до основного потоку.

Service Worker представляє JavaScript файл, в якому реєструються шляхи та джерела мережевих запитів з клієнта на сервер. Таким чином Service Worker в фоновому режимі контролює сторінку, до якої він застосований, а саме перехоплює та модифікує мережеві запити навігації та ресурсів, що дає широкий контроль над web-сторінкою, в ситуаціях коли наприклад відсутній зв’язок з мережею інтернет. Схему роботи сервіс worker-а представлено на рис. 1.7

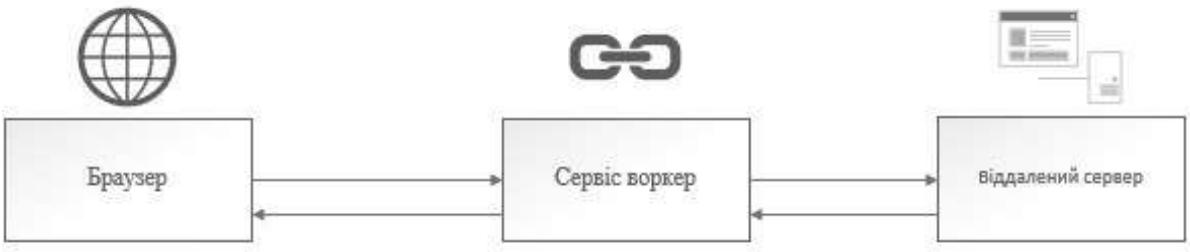

Рисунок 1.7 – Схема роботи сервіс worker-а

ServiceWorker’s мають працювати за захищеним з’єднанням HTTPS, така вимога передбачена з метою забезпечення безпеки. Так як модифікація

мережевого запиту, в процесі руху інформації мережею зловмисником, є надто високою небезпекою. Також ServiceWorker’s можуть працювати на localhost, така можливість передбачена в цілях можливості тестування ServiceWorker’s під час розробки web-застосунку.

Реєстрація ServiceWorker це перший крок, він реєструється за допомогою методу *ServiceWorkerContainer.register()*. Після цього, якщо реєстрація пройшла успішно, Service Worker буде завантажений на клієнт, як тільки користувач вперше наддасть запит до web-сайту, якому належить цей Service Worker. Після цього він спробує встановитись, встановлення відбувається у випадку якщо завантажений Service Worker завантажений вперше, або є новим, тобто має відмінності від попереднього. Останнім кроком життєвого циклу є активація, для вказаних для нього URL, або для всіх адресів ресурсів.

Service Worker має оновлюватись кожні двадцять чотири години, це максимальна кількість годин, він може завантажуватись і частіше, що б запобігти використанню старої версії коду клієнтом.

У випадку коли Service Worker вже існує, нова версія Service Worker встановлюється у фоновому режимі, але не активується. Тоді Service Worker переходить в стан в очікування. Нова версія Service Worker активується тільки тоді, коли більше не залишиться завантажених сторінок, що використовують старий Service Worker. Як тільки це станеться, новий Service Worker активується.

При використанні методу ServiceWorkerGlobalScope.skipWaiting(), очікування закриття завантажених сторінок може і не здійснюватися, в такому випадку треба передати контроль над новими сторінками новому worker-у методом *Clients.claim().*

При використанні Service Worker, для підготовки його для використання, використовуються його події *oninstall* та/або *onactivate.* Які спрацьовують на відповідних етапах його життєвого циклу: завантаження, встановлення, активація.

Так як Service Worker опрацьовує мережеві шляхи, один Service Worker може працювати одразу на кілька сторінок. Отримуємо можливість централізованого оновлення, для даних використовують важкі обчислення, таких як геолокація або гіроскоп. Тоді кілька відкритих сторінок змогли б використовувати одні й ті самі дані, не надсилаючи багато, схожих запитів серверу тим самим його навантажуючи, не навантажуючи пристрій зайвими операціями.

За допомогою Service Worker також можна, поліпшити продуктивність, за допомогою попереднього завантаження ресурсів, які знадобляться користувачу в скорому майбутньому, наприклад кілька наступних постів в стрічці.

Також Service Worker можна застосувати фонової синхронізації (оновлювати кеш в сховищі кешу CacheStorage), push-повідомлень(запускати service worker для відправки повідомлень користувачам сповіщень), реакції на певний час і дату, введення обмежень на геопозицію(обмеження для певних міст, країн).[3]

Опис інтерфейсів

Service Worker:

1. *Cache* – представляє сховище для пар Request/Response об'єктів, які кешуються як частина ServiceWorker життєвого циклу.

2. *CacheStorage* – представляє сховище для Cache об'єктів. Він забезпечує головний каталог усіх названих кешів, до яких ServiceWorkerможе отримати доступ, і підтримує відображення імен рядків до відповідних Cache об'єктів.

3. *Client* – представляє область дії клієнта служби обслуговування. Клієнт працівника служби - це або документ у контексті браузера, або a SharedWorker, який контролюється активним працівником.

4. *Clients* – представляє контейнер для списку Client об'єктів; основний спосіб доступу до активних клієнтів службових служб у поточному джерелі.

5. *ExtendableEvent* – продовжує термін служби «install» та «activate» події, що надсилаються ServiceWorkerGlobalScope, як частину життєвого циклу працівника сфери послуг. Це гарантує, що будь-які функціональні події (наприклад FetchEvent) не надсилатимуться до ServiceWorker, поки він не оновить схеми бази даних і не видалить застарілі записи кешу тощо.

6. *ExtendableMessageEvent* – об'єкт ServiceWorkerGlobalScope/message_event події, запущеної на службовця (коли повідомлення каналу надходить ServiceWorkerGlobalScopeз іншого контексту) - продовжує час таких подій.

7. *FetchEvent* – параметр, переданий в ServiceWorkerGlobalScope.onfetch обробник, FetchEvent являє собою дію вибору, яка надсилається на ServiceWorkerGlobalScopea ServiceWorker. Він містить інформацію про запит та отриману відповідь і забезпечує FetchEvent.respondWith() метод, який дозволяє нам надати довільну відповідь назад на контрольовану сторінку.

8. *InstallEvent* – параметр, переданий в oninstall обробник, InstallEvent інтерфейс представляє дію встановлення, яка відправляється на ServiceWorkerGlobalScopea ServiceWorker. Як похідна від ExtendableEvent, вона гарантує, що такі функціональні події, як FetchEventне передаються під час встановлення.

9. *NavigationPreloadManager* – надає методи управління попереднім завантаженням ресурсів із працівником служби.

10. *Navigator.serviceWorker* – повертає ServiceWorkerContainer об'єкт, який забезпечує доступ до реєстрації, видалення, оновлення та зв'язку з ServiceWorker об'єктами для пов'язаного документа .

11. *NotificationEvent* – параметр, переданий в onnotificationclick обробник, NotificationEvent інтерфейс являє собою подію клацання сповіщення, яка відправляється в ServiceWorkerGlobalScopea ServiceWorker.

12. *ServiceWorker* – представляє працівника служби. Багато контекстів перегляду (наприклад, сторінки, працівники тощо) можуть бути пов'язані з одним і тим же ServiceWorker об'єктом.

13. *ServiceWorkerContainer* – надає об’єкт, що представляє працівника сфери послуг як загальний підрозділ в екосистемі мережі, включаючи засоби для реєстрації, скасування реєстрації та оновлення

службовців, а також доступу до стану працівників сфери обслуговування та їх реєстрацій.

14. *ServiceWorkerGlobalScope* – представляє глобальний контекст виконання працівника служби.

15. *ServiceWorkerMessageEvent* – представляє повідомлення, надіслане на ServiceWorkerGlobalScope. Зверніть увагу, що цей інтерфейс застарілий у сучасних браузерах. Повідомлення службовця тепер використовуватимуть MessageEvent інтерфейс для узгодження з іншими функціями обміну повідомленнями.

16. *ServiceWorkerRegistration* – представляє реєстрацію працівника сфери послуг.

17. *SyncEvent* – інтерфейс SyncEvent представляє дію синхронізації, яка надсилається ServiceWorkerGlobalScopeв службі ServiceWorker.

18. *SyncManager* – забезпечує інтерфейс для реєстрації та переліку реєстрацій синхронізації.

19. *WindowClient* – представляє сферу клієнта службовця, який є документом у контексті браузера, керованим активним працівником. Це особливий тип Client об’єкта, з деякими додатковими методами та властивостями.

1.4.3 Протокол HTTPS

HTTPS (Hypertext Transfer Protocol Secure) – це протокол передачі даних прикладного рівня, що є розширенням протоколу HTTP, за рахунок захисту рівня захищених сокетів SSL, або більш сучасний TSL, за рахунок захисту транспортного рівня. Тому протокол також називають HTTP через SSL, або HTTP через TLS. Відповідає за захист конфіденційності, цілісності даних, якими обмінюються під час передачі, та аутентифікації. Захищає безпечну передачу даних, в небезпечних мережах, наприклад громадський «Wi-Fi» Він захищає від атак типу "Посередник", де зловмисник пасивно атакує, читаючи відправлені дані(паролі, банківські дані, та інші конфіденційні дані) порушуючи конфіденційність. Або активна атака, коли зловмисник змінює дані які ідуть по мережі, наприклад суму переводу коштів та одержувача, що порушує цілісність даних. Схему представлено на рис. 1.8.

Рисунок 1.8 – Атака типу «Посередник»

Двонаправлене шифрування комунікацій між клієнтом і сервером захищає комунікації від прослуховування та втручання.

Базою для HTTPS, використовується протокол HTTP, остання літера «S» в абревіатурі значить «secure», тобто захищений(за допомогою SSL або TSL). Деталі роботи SSL застарілий та TSL, розібрано вище в пункті «TSL (Transport Layer Security) / SSL (Secure Socket Layer)». На даний час актуальними вважається TSL 1.3 – 2018, RFC 8846, або попередня TSL 1.2 – 2008, RFC 5246. Всі попередні версії TSL 1.1, TSL 1.0, та всі версії SSL вважаються застарілими і ненадійними.[3]

Використання https: URL вказує, що протокол HTTP має використовуватися, але з іншим портом за замовчуванням (443) і додатковим шаром шифрування/автентифікації між HTTP і TCP. Аспект автентифікації HTTPS вимагає від третьої сторони, якій довіряють, підписувати цифрові сертифікати на стороні сервера.

Інфраструктура відкритих ключів є інтегрованим комплексом методів та засобів (набір служб), призначених забезпечити впровадження та експлуатацію криптографічних систем із відкритими ключами і передбачає, що сторони в обміні даними покладаються на деякі центри сертифікації (Certificate Authority, CA; також довірчий центр) для перевірки ідентичності сервісів та web-сайтів. На центри сертифікації покладена надзвичайно велика відповідальність.

Інфраструктура відкритих ключів передбачає, що мережа складається з багатьох вузлів, для спрощення демонстрації наприклад з двох клієнта і сервера, але вони не довіряють друг другу, бо можливо це атака типу «посередник». Тому вони довіряють вузлу, який називається «Довірчим центрам». Зображено на Рис. 1.9.

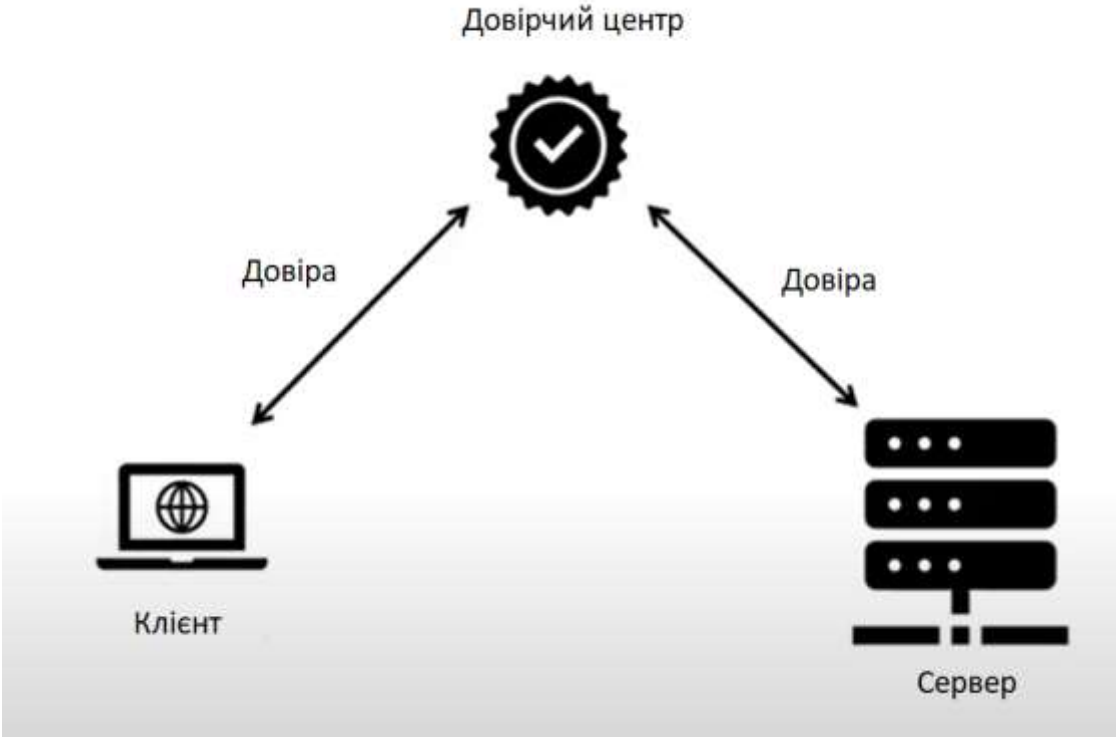

Рисунок 1.8 – Інфраструктура мережі відкритих ключів

Центр сертифікації (довірчий центр) - об'єкт, уповноважений створювати, підписувати та публікувати сертифікати. Центр має також повноваження ідентифікувати користувачів.

Основними операціями, що виконує довірчий центр, є видання, відновлення та анулювання сертифіката. Атрибутами сертифіката є ім'я та ідентифікатор суб'єкта, інформація про відкритий ключ суб'єкта, ім'я, ідентифікатор і цифровий підпис уповноваженого з видачі сертифікатів, серійний номер, версія і термін дії сертифіката, інформація про алгоритм підпису тощо.

Важливо, що цифровий сертифікат, який видає вузлам мережі довірчий центр містить цифровий підпис на основі секретного ключа довірчого центру. А основним компонентом цього сертифіката є відкритий ключ цього вузла якому «довірчий центр» видає сертифікат.

Якщо клієнт довіряє довірчому центру то у нього є відкритий ключ довірчого центру. Зображено на рисунку 1.9

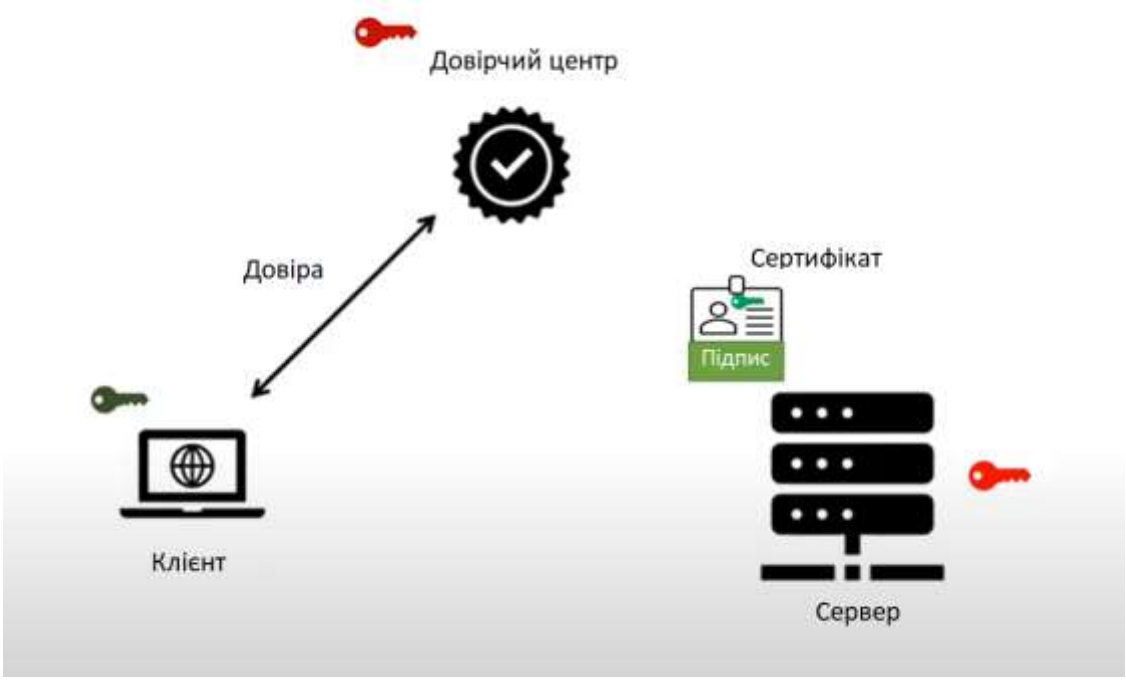

Рисунок 1.8 – Інфраструктура мережі відкритих ключів

Зображені на рисунку зелені ключі, це публічні ключі(відкриті), а червоні ключі, це приватні(захищені), пара ключів(захищений і приватний) з темним відтінком, це ключі довірчого центру, пара ключів з світлішим відтінком ключі серверу.

В нашому випадку сервер отримав сертифікат від довірчого центру, цей сертифікат підписаний приватним ключем довірчого центру і в тілі містить публічний ключ серверу, який цей сертифікат отримав для встановлення захищеного зв’язку з клієнтом. Так як клієнт довіряє довірчому центру він має публічний ключ довірчого центру і коли сервер передає клієнту сертифікат. То клієнт зможе розшифрувати цифровий підпис, який «довірчий центр» підписав своїм захищеним ключем, бо клієнт довіряє довірчому центру і має його публічний ключ, після розшифрування сертифіката клієнт, в тілі сертифіката знайде

публічний ключ сервера, і може бути впевнений, що цей ключ дійсно сервера. Таким чином відбувся обмін ключами між клієнтом і сервером, за допомогою інфраструктури відкритих ключів.

Мережа інтернет дуже масштабна, і один довірчий центр не зможе видати всім вузлам інтернету сертифікатів, тому використовується інфраструктура відкритих ключів, яка складається з довірчих центрів об'єднаних у ієрархію, таким чином є корінний довірчий центр, який сам напряму не видає сертифікати, з корінним довірчим центром працюють довірчі центри яких може бути багато, і клієнти звертаються вже до цих довірчих центрів. Дивитись рис 1.9

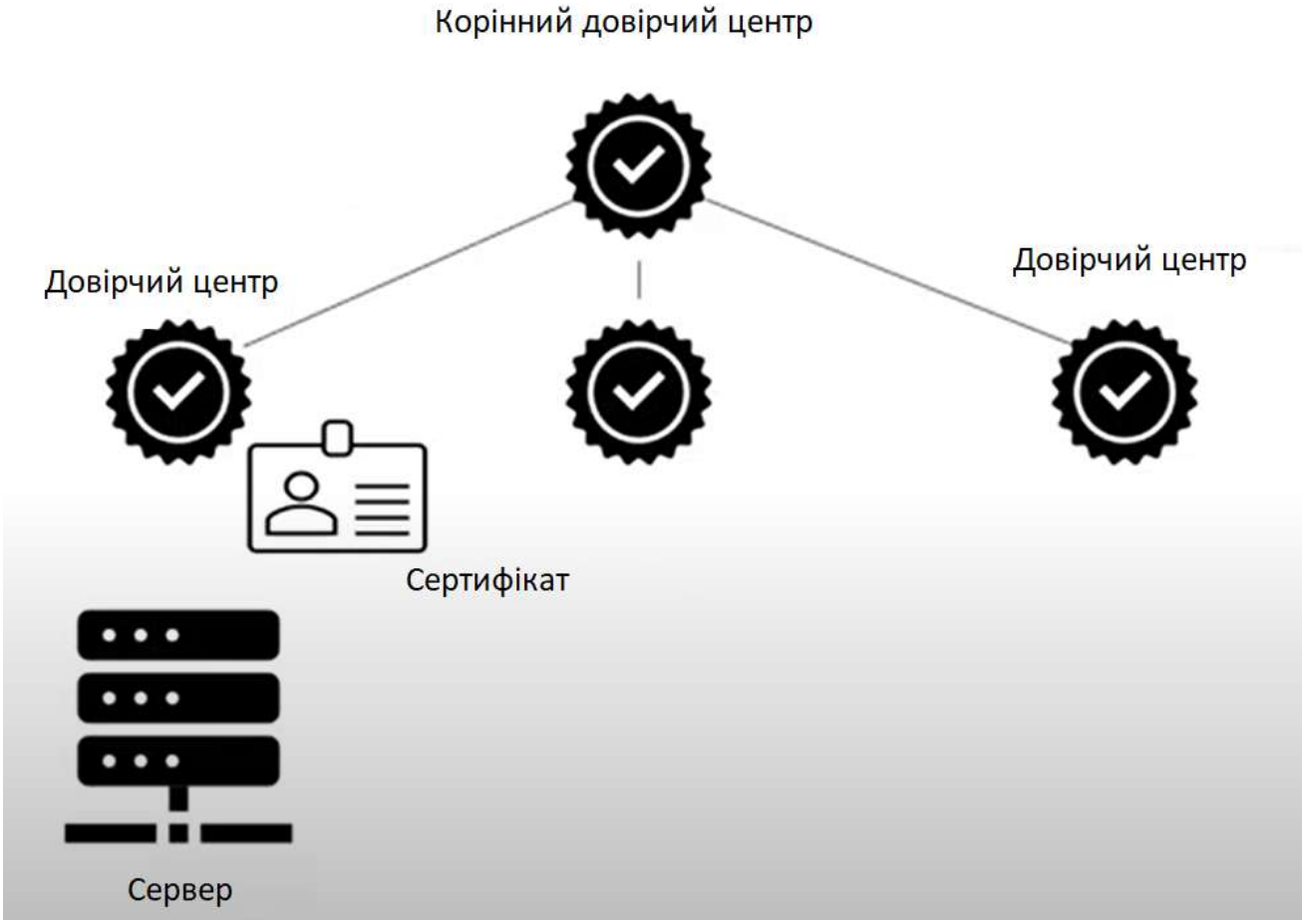

Рисунок 1.9 – Інфраструктура відкритих ключів

Ієрархія довірчих центрів можу бути більше одного рівня, і для підпису сертифікатів використовується ланцюжок довіри, або путь сертифікації. Сервер отримує сертифікат, сертифікат підписує довірчий центр, а сертифікат цього довірчого центру підписує, вище по ієрархії довірчий центр і так до корінного довірчого центру. Для того щоб перевірити, надійність сертифікату треба пройтись по цьому ланцюжку до корінного довірчого центру, інакше сертифікат може бути самопідписаний зловмисником.

І для того щоб атака типу посередник, не трапилась на рівні корінного довірчого центру, якому немає на кого довіритись, який є верхівкою ієрархії, в

операційній системах існують сховища центрів сертифікатів, які містять сертифікати корінних довірчих центрів, для вирішення цієї проблеми. І операційна система, сповіщає, що цей корінний довірчий центр не є зловмисником, а справжній і записаний в реєстрі.

Таким чином HTTPS створює захищений канал через незахищену мережу. Це забезпечує розумний захист від підслуховування та атак "людина посеред" , за умови використання відповідних наборів шифрів та перевірки та довіри сертифіката сервера.

В основі протокола лежить протокол HTTP — протокол прикладного рівня, створений для обміну даними в мережі інтернет. Обмін повідомленнями йде за звичайною схемою «запит-відповідь». На відміну від багатьох інших протоколів, HTTP не зберігає свого стану. Це означає відсутність збереження проміжного стану між парами «запит-відповідь».

Кожен запит/відповідь складається з трьох частин:

1. Стартовий рядок;
2. Заголовки;
3. Тіло повідомлення, що містить дані запиту, запитаний ресурс або опис проблеми, якщо запит не виконано.

Обов'язковим мінімумом запиту є стартовий рядок, виглядає він так *‹Метод› ‹URI› HTTP/‹Версія›*.

Обов’язковий заголовок Host (щоб розрізнити кілька доменів, які мають одну й ту ж IP-адресу).

Методи запиту:

1. Метод GET вимагає представлення зазначеного ресурсу. Запити з використанням GET повинні отримувати лише дані та не повинні мати ніякого іншого ефекту;
2. Метод HEAD запитує відповідь, ідентичну відповіді запиту GET, але без тіла відповіді. Це корисно для отримання метаінформації, записаної у заголовках відповідей, без необхідності транспортувати весь вміст;
3. Метод POST вимагає від сервера прийняти об'єкт з даними, вкладений у запит. Як новий запис. Надіслані дані можуть бути, наприклад, анотацією для існуючих ресурсів; групи новин, списку розсилки або списку коментарів; блок даних, який є результатом подання веб-форми на процес обробки даних; або елемент для додавання до бази даних;
4. Метод PUT вимагає, щоб вкладений об'єкт з даними зберігався під наданим URI. Якщо URI посилається на вже існуючий ресурс, він модифікується; якщо URI не вказує на існуючий ресурс, тоді сервер може створити ресурс із цим URI;
5. Метод DELETE видаляє вказаний ресурс;
6. Метод TRACE повторює отриманий запит, щоб клієнт міг побачити, які (якщо такі є) зміни або доповнення були внесені проміжними серверами;

7. Метод OPTIONS повертає методи HTTP, які сервер підтримує для вказаної URL-адреси. Це може бути використано для перевірки функціональності веб-сервера, запитуючи "*" замість певного ресурсу;

8. Метод CONNECT перетворює з'єднання запиту на прозорий тунель TCP / IP, як правило, для полегшення зв'язку, зашифрованого TSL (HTTPS), через незашифрований HTTP-проксі;

9. Метод PATCH застосовує часткові зміни до ресурсу.

Заголовки, набір рядків, параметри і його значення розділені двокрапкою. Містять головну інформацію про браузер і пристрій клієнта серверу, і навпаки, інформацію про сервер, браузеру клієнта. Між заголовками і тілом запиту має обов'язково бути пустий рядок.

Відповідь серверу, а точніше перший рядок відповіді виглядає так

HTTP/‹Версія› ‹Код статусу› ‹Опис статусу›

Код стану HTTP в основному поділяється на п'ять груп для кращого пояснення запиту та відповідей між клієнтом та сервером, як названо:

10. Інформаційний 1XX;
11. Успішні 2XX;
12. Перенаправлення 3XX;
13. Помилка клієнта 4XX;
14. Помилка серверу 5XX.

Приклад інтернет запиту, на web-сайт Державного Університету Телекомунікацій, деталі запиту на Рис.1.10

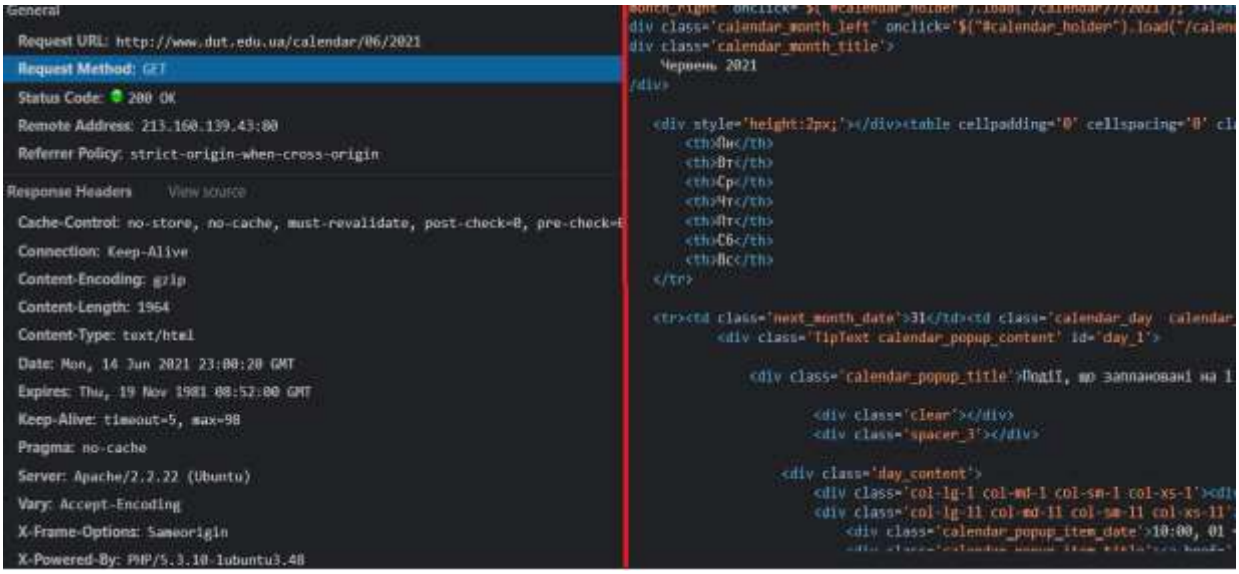

Рисунок 1.10 – Діалог клієнта з сервером на рівні HTTP/HTTPS

Можна спостерігати що, запит успішний зі статус кодом 200, методом GET, сервер відповів, HTML сторінкою, код якої зображено правіше на слайді, і вказав в заголовку, що сервер працює на Ubuntu, створений на PHP.

1.4.4 Application shell

Скорочено App shell model, архітектура, що дозволяє побудувати web-застосунок, що миттєво завантажується на екран пристрою, подібно до нативних застосунків, і що є відмінністю від web-сайтів, які не в змозі завантажитись миттєво по причині того, що треба отримати сторінку з віддаленого сервера, мережею інтернет, а на це потрібно певний час.

Application shell це оболонка, що містить основні риси користувацького інтерфейсу, тільки без вмісту. Тобто App shell є частиною web-сторінки, мінімально необхідний набор файлів JavaScript, CSS та HTML, який є незмінним впродовж довгого часу, і необхідний для кожного відвідування web-сторінки.

Для web-сайтів з архітектурою SPA(односторінковою архітектурою), Application shell використовується за замовчуванням.

Суть підходу базується на тому що, оболонка Application shell кешується в сховище кешу AppCache, за допомогою Service Worker, що забезпечує миттєвий, надійний доступ, і більшу продуктивність при повторному відвідуванні цього web-сайту. В перший раз відбувається кешування, але в наступний, статична оболонка Application shell дістається миттєво з сховища кешу, а динамічний вміст завантажується в цю оболонку за допомогою AJAX, при чому, об’єм завантажений даних, вже менше, тому що завантажується лише частина сторінки а не, вся сторінка. Таким чином користувач миттєво отримує сторінку, не чекаючи її на білому екрані, і завантаження даних для неї пришвидшується. Якщо мережа інтернет відсутня то web-застосунок покаже каркас користувацького інтерфейсу, і зможе про це проінформувати користувача в стилі інтерфейсу web-застосунку, на місцях для даних можуть бути передбачені заглушки, що б все виглядало органічно. Web-застосунок при цьому не впаде, як звичайний web-сайт, який втратив підключення до мережі інтернет.

Для web-сайтів, які мають тільки статичні дані використання Application shell теж можливе, просто web-сайт буде на сто відсотків Application shell.

Для web-сайтів, які мають і статичний і динамічний вміст, статичні дані завантажиться локально зі сховища, а динамічні довантажуються через серверне API. Продемонстровано на рис 1.11

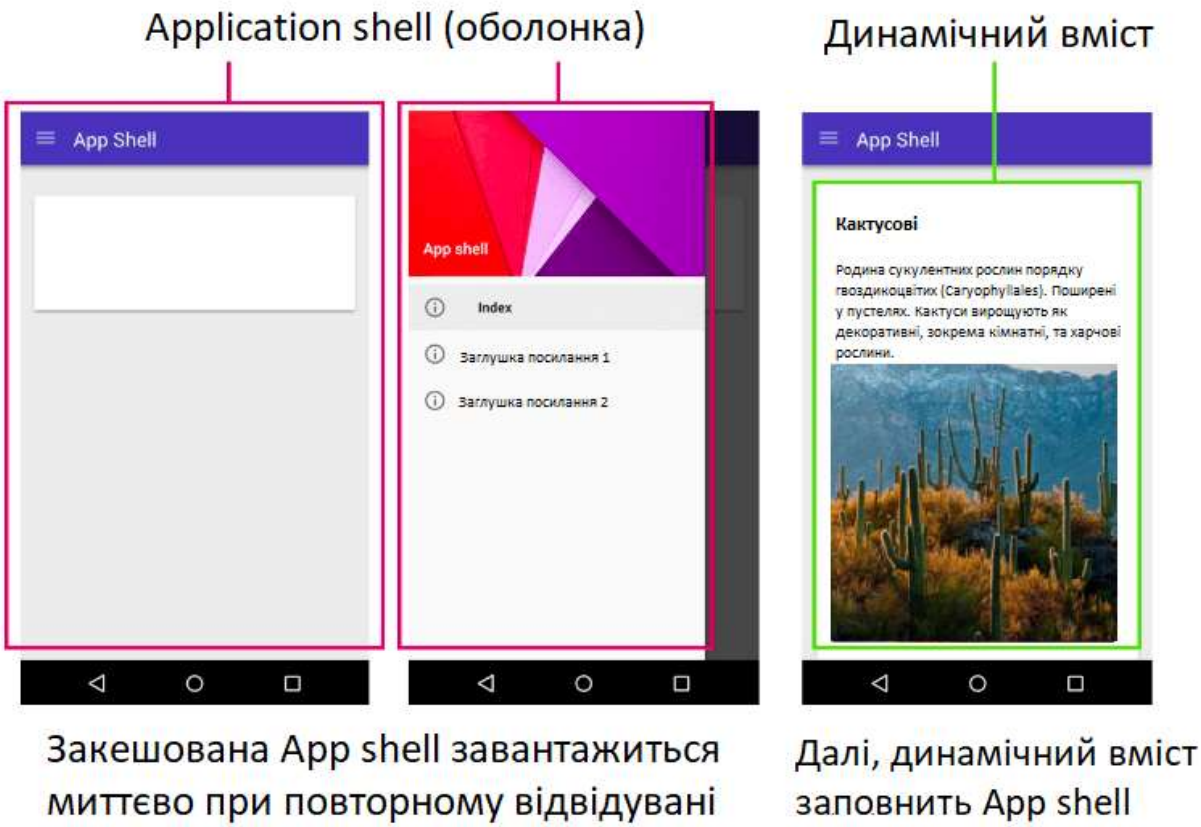

Рисунок 1.11 – Робота Applicatio shell

Application Shell, має використовувати статичні ресурси із сховища кешу, швидко завантажуватись, якнайменше використовувати данні, відділяти контент від навігації, бажано опційно кешувати динамічний контент.

В процесі проектування web-застосунку з архітектурою Application Shell варто, пам’ятати що динамічні і статичні дані мають бути чітка межа, що б пізніше розділити статичні дані в сховище кешу CacheStorage, а динамічні дані окремо за першої потреби підвантажувати.

Для роботи в режимі відсутності підключення до інтернету треба підключити до HTML файлу створений Web manifest.[4] Він надає інформацію про програму в текстовому файлі JSON, який необхідний для того, щоб web-застосунок було завантажено і відображалося користувачеві аналогічно нативному застосунку (встановлення на домашній екран пристрою, що надає користувачам більш швидкий доступ і більше можливостей). Приклад створеного маніфесту рис 1.12

Рисунок 1.12 – Web Manifest

Проектувати варто так, що б Application Shell була якомога простішою, наскільки це можливо, що б показати web-застосунок одразу як тільки він відкрився. Весь користувацький інтерфейс і інфраструктура кешуються локально в сховище кешу CacheStorage за допомогою Service Worker так, що при наступних завантаженнях повертаються не всі, а тільки нові або змінені дані.

App shell, в поєднанні з Service worker та CacheStorage створює потужний шаблон для оффлайн кешуванн. Крім цього він також пропонує значні переваги продуктивності, у вигляді миттєвого завантаження при повторних відвідинах web-застосунку. Потрібно закешувати статичні ресурси і UI (наприклад HTML, JavaScript, зображення і CSS) так, щоб вони працювала в режимі відсутності підключення до мережі інтернет і заповнити його вмістом(або даними або заглушками), використовуючи JavaScript. Контент може бути також закешований при першому відвідуванні, але зазвичай він завантажується в міру необхідності.

Створюється відчуття взаємодія, як з нативним застосунком. Прийнявши модель Application shell model, можна створювати користувацькі інтерфейси зі швидкістю нативного застосунку, економічне використання даних з сервера і в додачу підтримка роботи поза мережею.

1.4.5 Push Notification

Реалізується шляхом поєднання двох Web API, а точніше Nitification API, а також Push API. Реалізація Push Notification є бажаним, але не обов'язковим. Notification API для відображення сповіщень, а Push API для обробки цих повідомлень, які відправлені з сервера на клієнтську частину за допомогою push service браузера.

На стороні клієнта знаходиться web-сторінка, Service Worker, який взаємодіє з web-сторінкою, та User Agent, також відомий як браузер. Тож web-сторінка взаємодіючі з Service Worker, отримує повідомлення від сервера, завдяки push-подіям браузера, який на них реагує і спрацьовує.

На сервері надсилається повідомлення з серверного API до push service, який вже далі доставить повідомлення правильному клієнту. Архітуктура Push Notification зображена на рис. 1.13

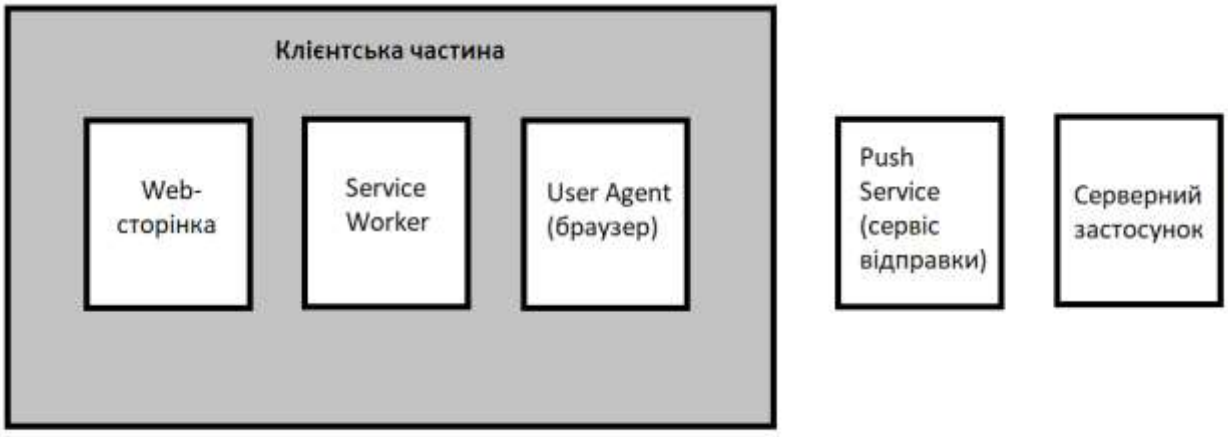

Рисунок 1.13 – Архітуктура Push Notification

Тепер детальніше, Notification API, є інструментом, і дає можливість розробнику відображати сповіщення користувачеві web-сторінки. І перед тим як надсилати користувачу сповіщення, спочатку треба отримати згоду від користувача на отримання таких сповіщень, дізнатись дозволу, можна за допомогою методу Notification.requestPermission(), який отримує функцію колбек з аргументом статусу, який покаже чи дозволене сповіщення, чи ні. Але можна обходитись і без цього методу, тому що браузер автоматично запитає про це, і в разі згоди браузер підписує користувача на push service.

Для відображення повідомлення, спочатку в Service Worker відбувається перевірка, чи надано дозвіл, затим ми викликаємо показ метод показ сповіщення, а саме метод «showNotification» на об'єкті реєстрації «reg» в Service Worker, і передаєм в нього заголовок повідомлення першим аргументом. Більш детальні налаштування сповіщення здійснюються через об'єкт параметрів, який передається другим аргументом після заголовку повідомлення, в метод

«showNotification» на об’єкті реєстрації «reg» в Service Worker. В цьому об’єкті можна вказати, «body» тіло повідомлення, тобто основний текст, «icon» іконку для повідомлення, «vibration» довжину вібрації від повідомлень для телефонів, і «data» дані, ключ в яких дасть змогу визначити, яке повідомлення було натиснуте, що б в повідомлені створити кнопки в об’єкті налаштунків існує поле «actions», в яке передається масив, з об’єктами кнопок, в кожні зазначається назва, якщо необхідно іконка, та обробник для кліку.

Подія «notificationclose», тобто закриття сповіщення спрацює лише тоді, коли сповіщення відхиляється за прямого закриття на цього повідомленні, якщо користувач натисне закрити всі пов’язані сповіщення, то ця подія не спрацює.

Подія «notificationclick», якщо користувач натискає на повідомлення, або на кнопку дії в повідомлені, то подія спрацьовує. Після спрацьовування однієї з подій, ми можемо визначити, на якому сповіщенні спрацювала подія, і до якої дії це має призвести.

Push API, дозволяє підписуватись на повідомлення, надісланих з сервера через push service, що використовується браузером, Шляхом створення спеціального об’єкт підписки, з допомогою методу getSubscription на об’єкті reg.pushManager, який пізніше надсилається на сервер, якщо користувач погодився на сповіщення з сервера, і там на сервері зберігається цей об’єкт підписки.

Для того, що клієнт отримав повідомлення, сервер спочатку повинен їх надіслати на клієнт, а він має їх обробити. Кожен браузер керує push-повідомленнями через власну систему, яка називається push-сервісом. Коли користувач надає дозвіл на отримання повідомлень, браузер підписує користувача на отримання сповіщень. Push Service створює об’єкт, з відкритим публічним ключем, відміткою згоди на повідомлення та URL адресою для Push Service браузера, який є унікальним для кожного користувача, на цю адресу і відправляються повідомлення з сервера, зашифровані публічним ключем, і відправляє цей об’єкт на сервер, сервер його зберігає. Приклад JSON об’єкту зображено на рис. 1.14

```
{
  "endpoint":
  "https://android.googleapis.com/gcm/send/
  f1LsxkKp...",
  "keys": {
    "p256dh": "BLc4xRzKlKORKWlbOQRv-1n...",
    "auth": "5I2Bu2oKdyy9CwL8QVF0NQ=="
  }
}
```

Рисунок 1.14 – Об’єкт підписки, згенерований Push API

Service Worker відреагує на нове сповіщення, так як спрацює push-подія, і це дозволить програмі реагувати на сповіщення, відображаючи їх.

Для того, що б отримувати повідомлення, браузер навіть не повинен бути відкритим, і це заощаджує потужність процесору та економить заряд акумулятора на портативних пристроях.

Послідовність дій:

1. Перевірка чи користувач вже підписаний
2. Якщо не підписаний пропонуємо підписатись і надсилаємо об’єкт підписки, якщо вже підписаний надсилаємо, оновлення, останній об’єкт підписки.
3. Зберігаємо на сервері об’єкт підписки.

Демонстрація на рис 1.15

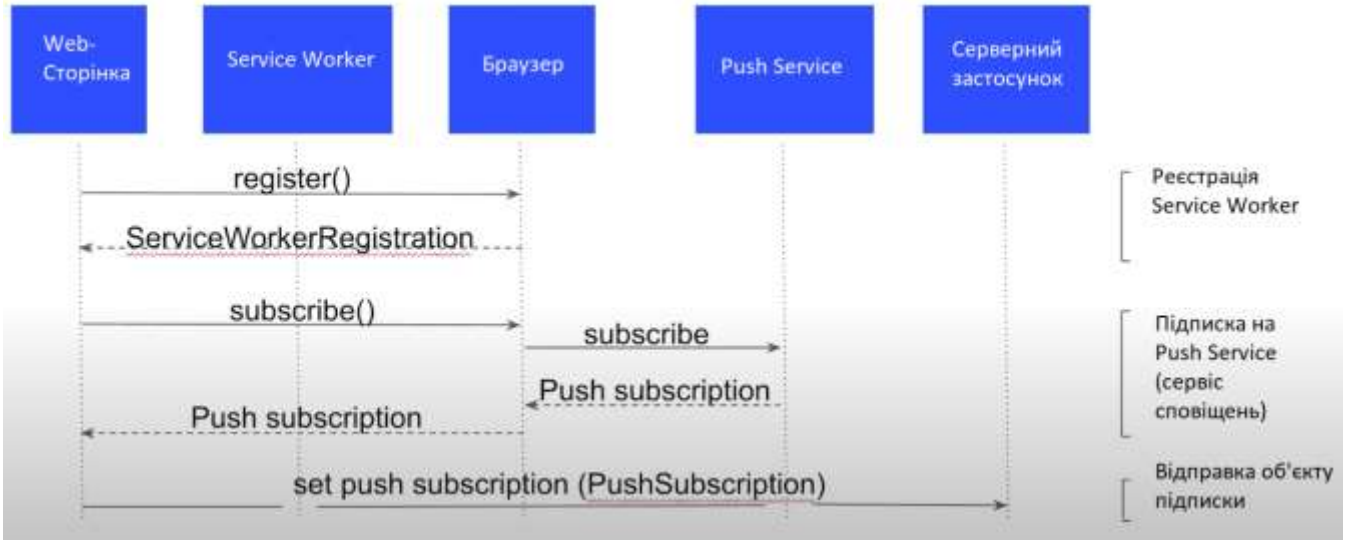

Рисунок 1.15 – Реєстрація на сповіщення

Сервер генерує повідомлення, шифрує публічним ключем, відправляє на URL адресу, яку видав Push Service, в об’єкті підписки збереженому на сервері, далі браузер отримує повідомлення. Для відправки повідомлень з сервера існує пакет бібліотека «web-push», для платформи NodeJS.

Відправка push-повідомлень з сервера на клієнтську частину, зображено на рис. 1.16

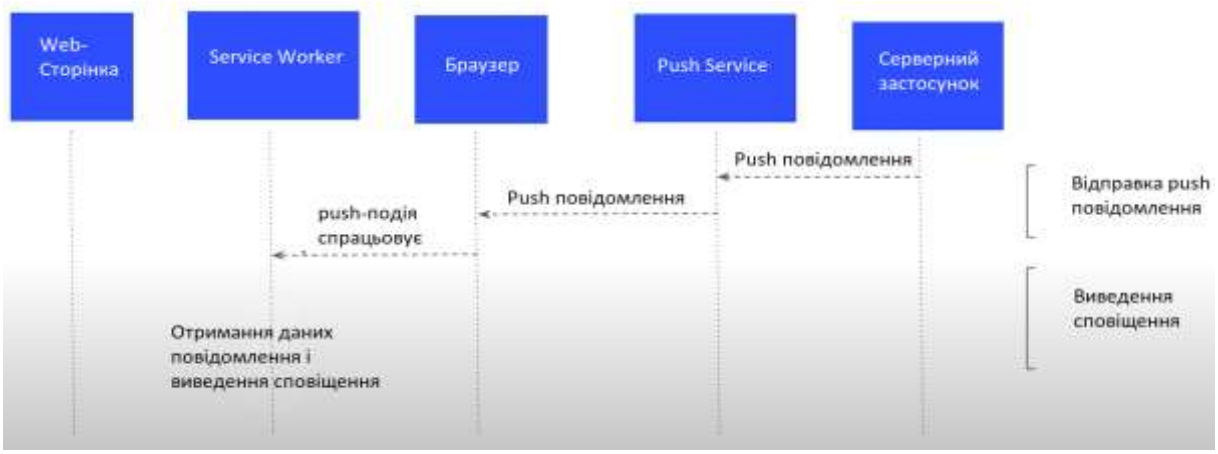

Рисунок 1.16 – Відправка push-повідомлення з сервера

Отримання повідомлення відбувається в наступній послідовності:

1. Повідомлення надходить з серверу.
2. Service Worker реагує на push-повідомлення, push-подією.
3. Повідомлення обробляється, потрібним чином.
4. Сповіщення виводиться на екран.

1.5 Потоки виконання операцій та Web Worker’s API

Перш ніж розібрати принцип багатопоточності, варто звернути увагу не те, що мова JavaScript – це однопотокова мова, яка підтримує можливості асинхронності.

За допомогою асинхронності, система «планує» навантаження на цикл подій таким чином, щоб в першу чергу виконувалися операції, пов'язані з призначеним для користувача інтерфейсом, намагаючись не заблокувати основний потік виконання програми. За це відповідає концепт під назвою «Event loop» або цикл подій. Суть якого полягає в тому, що програма біжить по рядкам коду, коли інтерпретатор доходить функції, він бере її і переміщає цей код в стек «Call Stack», з якого цей код йде на виконання. Якщо доходить черга до асинхронного коду, наприклад запит на сервер за даними, то браузер запускає виконання цієї функції переміщаючи її в стек «Call Stack», і не чекаючи повного виконання(поки сервер, опрацює запит і наддасть відповідь), виймає її з стуку «Call Stack», що б не блокувати потік. Але браузерне «Web API» реєструє, що функція має довиконатись, і біжить далі по рядкам виконуючи наступні функції, поміщаючи їх в стек «Call Stack». Коли, як в наведеному прикладі надходить відповідь з сервера, «Web API» розуміє, що почата функцію слід виконатись до кінця, тому що дані вже отримані, і ця функція потрапляє в чергу колбеків «Callback Queue», по якій постійно бігає цикл і з метою звільнити чергу, переміщуючи код в стек «Call Stack», з якого код вже виймається і виконується. За таким принципом працює асинхронність в JavaScript, що дозволяє не блокуючи потік працювати з ним.

Але асинхронність обробляє запити Web API браузера, події, setTimeOut() і інше. Проблеми виникають коли необхідно опрацювати довільний

код, який буде інтенсивно використовувати ресурси процесору, до прикладу не асинхронний мережевий HTTP запит, а величезний цикл for, з складним сортуванням. В такому випадку, так як JavaScript однопотоковий, потік блокується на строк виконання цього коду і вивільнити цикл подій для деблокування програми не вийде, користувач буде змушений чекати поки програма відновить працездатність.

З вище зазначеного можна зробити висновок, що асинхронні функції, знімають лише частину обмежень, зв’язаних з однопотоковістю JavaScript.

В деяких випадках використовується підхід, коли за допомогою функції setTimeout(), розбивають «важкий» код, на більш легкі фрагменти, виконуючи кожен в окремому setTimeout(). Таким чином навантаження розподіляється по циклу подій, не блокуючи потік програми, а з ним і користувацький інтерфейс. Але такий підхід, роздуває об’єм коду та ускладнює його візуальне сприйняття програмістом.

Для вирішення вище описаних проблем, чудово підійде Web Worker, які дають можливість перемістити важкий JavaScript код в окремі потоки, які будуть належати браузеру. Створюючи додаткові потоки, в яких в фоновому режимі можуть виконуватись важкі обчислення, які навантажують процесор, але вже без блокування основного потоку і користувацького інтерфейсу.[12]

JavaScript і надалі залишається однопоточним, варто чітко розуміти, що JavaScript не визначає модель потоків, Web Worker являють собою частину браузера, до якої можна отримати доступ через JavaScript. Web Worker надають інструмент для запуску скриптів у фоновому потоці, що допомагає вивільнити основний потік, див рис 1.17

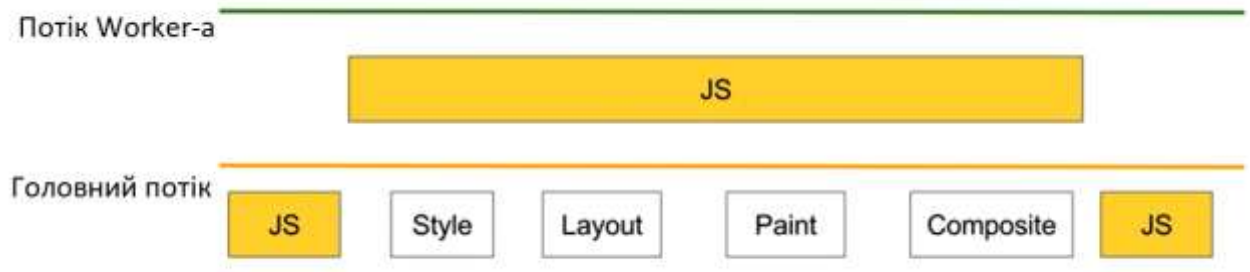

Рисунок 1.17 – Потоки в браузері

Існують три типи worker-ів в специфікації Web Worker, Service Worker, Shared Worker, Dedicated Worker.

Service Worker це worker або worker-и, керовані подіями, зареєстровані з використанням джерела їх походження та шляху. Вони можуть контролювати web-сторінку, з якою пов'язані, перехоплюючи та модифікуючи команди навігації та запити ресурсів, виконуючи хешування даних, якими можна дуже чітко керувати. [5][11]

Dedicated Worker, тобто виділений worker, доступний лише із батьківського скрипта, скрипта, який його викликав, тоді як спільні worker-и Shared Worker можуть бути доступними з кількох сценаріїв.[6]

Дає можливість запустити будь-який код всередині окремого потоку Dedicated Worker-а, з кількома обмеженнями, які стосуються, обмеженнями для всіх worker-ів.

Всі три worker-а: Service Worker, Dedicated Worker, Shared Worker мають програмні обмеження, не можна прямо маніпулювати DOM всередині Worker-а, що б не створювати колізій, коли кілька потоків намагаються одночасно змінити один елемент DOM. Не вийде використати окремі методи і властивості з об’єкта window, так як там немає об’єкту window, замість нього об’єкт self. Але цілком важливо використовувати великий набір опцій, доступний під Window, включаючи WebSockets, Fetch api, Location, setTimeout(), JSON та механізми зберігання даних, таких як IndexedDB, але не local Storage і session storage.

Дані передаються між робочими worker-ами та головним потоком через систему повідомлень - сторони передають свої повідомлення, використовуючи метод postMessage() та відповідають на повідомлення за допомоги обробника подій onmessage (повідомлення зберігається в атрибуті даних про події повідомлення). Данні при цьому копіюються, а не пересилаються. Об'єкти worker можуть, у свою чергу, створити нові об'єкти worker і так далі, якщо вони всі працюють в рамках поточної сторінки. А також об’єкти worker можуть використовувати XMLHttpRequest для мережевого вводу / виводу. Але з обмеженням – атрибути responseXML і об'єкт каналу XMLHttpRequest завжди повертають null.

Shared Worker розділений worker, він має кілька відмінностей від Dedicated Worker. Shared Worker є особливим видом worker до якого можна отримати доступ з декількох контекстів браузера, наприклад, з декількох вкладень браузера, iframe або інших worker.[7]

Доступ до Shared Worker може отримати будь-який процес, що має те саме джерело, що і цей worker. А Dedicated Worker може обмінюватись даними лише з батьківським скриптом, в чому і є основна відмінність між Shared Worker і Dedicated Worker. Не варто створювати десятками Dedicated Worker, тому що операція не є дешевою з точки зору ресурсів, і при створені великої кількості worker-ів, процесор пристрою буде навантажуватись, тому в деяких ситуаціях буде вигідним перевикористовувати один й той самий worker за допомогою Shared Worker.

1.6 Основні Web API, для реалізації.

1.6.1 Cache API

Cache API це інтерфейс, який створювався для збереження пари об’єктів Request / Response, які кешуються, тобто зберігаються в спеціальному сховищі

кешу Cache Storage. Cache API доступний, як в області видимості вікна, так і в області видимості worker-ів. Для одного скрипта, може бути створена множина іменованих об’єктів Cache.

Для відкриття іменованого об’єкту Cache використовується метод об’єкту сховища кешу CacheStorage.open(cacheName), після цього можна визивати любі методи для керування станом кешу.

Записи в Cache не будуть оновлені, поки не буде виконано явний запит; строк існування не закінчиться до моменту видалення.

Браузер робить все, для того щоб оптимізувати дисковий простір, тому кожний браузер має визначені обмеження, щодо об’єму сховища кешу, доступного для збереження коду. Як правило при переповнені сховища кешу браузер видалить всі дані із кешу, або залишить все як є, не чіпаючи жодного файлу. Тому варто враховувати періодичне видалення кешу. Для кешування треба встановлювати версії кешу в імені, що б використовувати найбільш актуальну версію кешу.

Сховище кешу допускає збереження файлів мережевого запиту з методом GET, тому слід використовувати Cache.add, Cache.put, Cache.deleteAll з GET запитами, також в нових версія браузера починаючи з Chrome 46 Cache API будуть зберігати запити лише з безпечних джерел, що використовують безпечне з’єднання через HTTPS.[8]

Методи об’єкту Cache:

1. Cache.match (request, options)

   Повертає Promise, який успішно завершується з виявленням першого збігу для даного запиту в об'єкті Cache;
2. Cache.matchAll (request, options)

   Повертає Promise, який успішно завершується і повертає масив всіх запропонованих варіантів для даного запиту в об'єкті Cache;
3. Cache.add (request)

   Приймає як параметр URL, отримує дані по ньому і додає отриманий об'єкт відповіді в заданий кеш. Функціональний еквівалент виклику fetch () з подальшим викликом Cache.put () для додавання результату в кеш;
4. Cache.delete (request, options)

   Знаходить запис Cache, чий ключ є запитом, і, в разі знаходження, видаляє запис Cache і повертає Promise, успішно завершується зі значенням true. Якщо ж запис Cache не знайдено, повертається false;
5. Cache.addAll (requests)

   Приймає масив URL як параметр, отримує дані по ним, додає отримані об'єкти відповідей в заданий кеш;
6. Cache.put (request, response)

   Приймає запит і відповідь на нього і додає їх в заданий кеш;
7. Cache.keys (request, options)

   Повертає Promise, який віддає масив ключів Cache.

1.6.2 LocalStorage, SessionStorage та IndexedDB

Session Storage інтерфейс з Web Storage API, локальне сховище браузера на стороні клієнта, що викликається з об’єкта window.sessionStorage або

sessionStorage. Дозволяє отримати поточний об’єкт сесії. Всі збережені дані в ньому мають відведений програмістом темін існування. Відкриття тієї ж сторінки в новому вікні браузера або новій вкладці призводить до створення нової сесії сторінки, що відрізняється від поведінки session cookies.

SessionStorage підтримує окрему область зберігання для кожного даного джерела, яка доступна на час сеансу сторінки (якщо браузер відкритий, включаючи перезавантаження та відновлення сторінки).

Зберігає дані лише для сеансу, тобто дані зберігаються до закриття браузера або його вкладки. Дані ніколи не передаються самостійно на сервер.

Обмеження зберігання більше, ніж файлу cookie (не більше 5 мегабайт).

Методи window.sessionStorage:

1. sessionStorage.getItem(key)
   Повертає значення для даного ключа або нуль, якщо ключ не існує;
2. sessionStorage.setItem(key, value)
   Встановлює значення для даного ключа;
3. sessionStorage.removeItem(key)
   Видаляє видаляє ключ та його значення;
4. sessionStorage.key(index)
   Повертає ключ для значення в заданому числовому положенні;
5. sessionStorage.clear()
   Видаляє всі пари ключ-значення.

Local Storage також інтерфейс з Web Storage API, локальне сховище браузера на стороні клієнта, що викликається з об’єкта window.localStorage або localStorage.

Локальне сховище LocalStorage робить те саме що і SessionStorage, але дані зберігаються навіть тоді, коли браузер закрито та знову відкрито. Тобто сховище зберігає дані без терміну придатності та очищається лише за допомогою JavaScript або очищення кешу браузера / локально збережених даних.

Обмеження зберігання більше, ніж сховища sessionStorage (не більше 10 мегабайт). Дані ніколи не передаються самостійно на сервер. Доступ до сховища може здійснюватися з любого вікна, на відміну від SessionStorage, де сховище працює тільки для одного вікна.

Методи window.localStorage

1. localStorage.getItem(key)

Повертає значення для даного ключа або нуль, якщо ключ не існує.

2. localStorage.setItem(key, value);

Встановлює значення для даного ключа.

3. localStorage.removeItem(key);

Видаляє видаляє ключ та його значення.

4. localStorage.key(index)

Повертає ключ для значення в заданому числовому положенні.

5. localStorage.clear()

Видаляє всі пари ключ-значення.

Також існує сховище Cookie, яке теж використовуватиметься в даній роботі, воно буде розглянуто для повноти картини, див рис 1.18

	Cookies	sessionStorage	localStorage
Розмір	4 Кбайт	5 Мбайт	до 10 Мбайт
Підтримка браузером	HTML4/HTML5	HTML5	HTML5
Відправка з запитом	Так	Ні	Ні
Строк життя	Налаштовується вручну	При закриті вкладення браузера	Не видаляється
Місце зберігання	Браузер і сервер	Тільки в браузері	Тільки в браузері
Доступність	Із будь якого вікна	Із того самого вікна	Із будь якого вікна

Рисунок 1.18 – Порівняння характеристик сховищ

Основною відмінністю сховища Cookie є те, що воно має надмалий об’єм, всього 4 кілобайта, і дані цього сховища, прикріплюються до кожного запиту з клієнта на сервер автоматично.[9]

Найпотужніше сховище IndexedDB, взаємодіяти з ним можна через низькорівневе IndexedDB API. Створено для клієнтського сховища великого обсягу структурованих даних, включаючи файли / blobs. IndexedDB API використовують індекси для забезпечення високо-продуктивного пошуку даних.

Якщо localStorage та SessionStorage корисний для зберігання невеликої кількості даних, по суті, це лише рядкове сховище ключових значень із спрощеним синхронним (тобто блокують запити) API.

IndexedDB був розроблений для роботи зі значно більшими обсягами даних. він забезпечує як синхронний, так і асинхронний API. На практиці, майже всі поточні реалізації є асинхронними, і запити не блокують завантаження інтерфейсу користувача. Крім того, indexedDB, як видно з назви, забезпечує індекси. Створює можливість запускати елементарні запити у своїй базі даних і отримувати записи, шукаючи їх ключі в певних діапазонах ключів. IndexedDB також підтримує транзакції та надає прості типи (наприклад, Date).

IndexedDB – дає можливості зберігання даних у браузері користувача. Оскільки це дозволяє створювати веб-програми з розширеними можливостями запитів незалежно від доступності мережі, ці програми можуть працювати як в Інтернеті, так і в режимі офлайн.

Бази даних IndexedDB зберігають пари ключ-значення. Значення можуть бути складними структурованими об'єктами, а ключі можуть бути властивостями цих об'єктів. Є можливість створювати індекси, які використовують будь-які властивості об’єктів для швидкого пошуку, а також відсортовані переліки. Ключі можуть бути двійковими об'єктами.

IndexedDB побудований на моделі транзакційної бази даних. Все, що відбувається в IndexedDB, завжди відбувається в контексті транзакції. API IndexedDB надає безліч об'єктів, що представляють індекси, таблиці, курсори тощо,

але кожен з них прив'язаний до певної транзакції. Таким чином, немає можливості виконувати команди або відкривати курсори поза транзакцією. Транзакції мають чітко визначений термін роботи, тому спроба використовувати транзакцію після її завершення видає «exeption». Крім того, транзакції здійснюються автоматично, і їх неможливо здійснити вручну.

Ця модель транзакцій дійсно корисна, якщо врахувати, що може статися, якщо користувач одночасно відкрив два екземпляри веб-застосунка на двох різних вкладках. Без транзакційних операцій обидва екземпляри можуть заважати модифікаціям один одного.

API IndexedDB в основному асинхронний. API не надає даних, повертаючи значення; замість цього потрібно передати функцію зворотного виклику.

API IndexedDB не "зберігає" значення в базі даних, або "отримує" значення з бази даних синхронно. Натомість користувач надає запит про те, що відбувається операція з базою даних. Користувач отримуєте повідомлення про подію DOM, коли операція закінчується, а тип події, яка отримується, дозволяє дізнатися, чи вдалася операція, чи не вдалася.

IndexedDB використовує багато запитів. Запити - це об’єкти, які отримують події успіху чи невдачі DOM, про які згадувалося вище. Вони мають «onsuccess» і «onerror» події, і можна зареєструвати за допомогою addEventListener() і видалити реєстрацію події removeEventListener() за ними. Вони також мають «readyState», «result» і «errorCode» властивості, які повідомляють про стан запиту. result Властивість особливо корисна, це може бути різна інформація, в залежності від того, як був згенерований запит (наприклад, IDBCursor екземпляр, або ключ до значення , який було щойно встановлено в базу даних).

IndexedDB використовує події DOM, щоб сповіщати вас про доступність результатів. Події DOM завжди мають властивість «type» (у IndexedDB вона найчастіше встановлюється в "success" або "error"). Події DOM також мають властивість «target», яка вказує, де спрацювала подія. У більшості випадків «target» подією є IDBRequest об’єкт, який був створений в результаті виконання певної операції з базою даних. Події успіху не спливають, і їх не можна скасувати. Натомість події помилок спливають і можуть бути скасовані. Це досить важливо, оскільки події помилок переривають будь-які транзакції, в яких вони виконуються, якщо вони не скасовані.[10]

IndexedDB є об’єктно-орієнтованим. IndexedDB не є реляційною базою даних з таблицями, що представляють колекції рядків і стовпців. Ця важлива і принципова різниця впливає на спосіб проектування та побудови застосунків.

У традиційному реляційному сховищі даних у вас буде таблиця, що зберігає колекцію рядків даних і стовпців названих типів даних. З іншого боку, IndexedDB вимагає створити сховище об’єктів для типу даних і зберігати об’єкти JavaScript у цьому сховищі. Кожне сховище об’єктів може мати колекцію індексів, що робить його ефективним запитом та ітерацією.

IndexedDB не використовує структуровану мову запитів (SQL). Він використовує запити в індексі, який створює курсор, який використовується для ітерації по набору результатів.

IndexedDB дотримується політики того самого походження CORS. Походження - це домен, протокол рівня програми та порт URL-адреси документа, де виконується сценарій. Кожне джерело має свій власний набір баз даних. Кожна база даних має ім'я, яке ідентифікує її в межах походження.

Межа безпеки, накладена на IndexedDB, заважає програмам отримувати доступ до даних з іншим походженням. Наприклад, хоча програма або сторінка в https://www.example.ua/cat/ може отримувати дані з https://www.example.ua/folder/ , оскільки вони мають однакове походження, вона не може отримати дані з https://www.example.ua:3030/cat/ (інший порт) або http://www.example.com/cat/ (інший протокол), оскільки вони мають різне походження.

Інтерфейс IndexedDb

Щоб отримати доступ до бази даних, потрібно викликати метод open() у атрибута indexedDB об'єкта window . Цей метод повертає об'єкт IDBRequest; асинхронні операції зв'язуються із викликаючим застосунком, викликаючи події об'єкта IDBRequest .

Підключення до бази даних:

1. IDBEnvironment

   Надає доступ до функцій IndexedDB. Реалізовано об'єктами windowі worker;

2. IDBFactory

   Надає доступ до бази даних. Цей інтерфейс представлений глобальним об'єктом indexedDB. Він є точкою входу для API;

3. IDBOpenDBRequest

   Представляє запит на відкриття бази даних;

4. IDBDatabase

   Являє собою підключення до бази даних. Це єдиний спосіб отримати транзакцію в базі даних;

Отримання і зміна даних:

1. IDBTransaction

   Являє транзакцію. Дає можливість створити транзакцію в базі даних, вказавши область дії (наприклад, до яких сховищ об'єктів до яких потрібно отримати доступ) далі визначається тип доступу (тільки читання або читання / запис), який потрібен;

2. IDBRequest

   Загальний інтерфейс, який обробляє запити до бази даних і забезпечує доступ до результатів;

3. IDBObjectStore

   Універсальний інтерфейс, який обробляє запити до бази даних і надає доступ до результатів;

4. IDBIndex

   Дозволяє отримати доступ до підмножини даних в IndexedDB, але замість первинного ключа використовує індекс для витягання запису / записів. Іноді це швидше, ніж використання IDBObjectStore;

5. IDBCursor

   Ітерує по сховищам об'єктів і індексам;

6. IDBCursorWithValue

Ітерує по сховищам об'єктів і індексам і повертає поточне значення курсора;

7. IDBKeyRange

Визначає діапазон ключів, який можна використовувати для отримання даних з бази даних в певному діапазоні.

1.6.3 API взаємодії з ОС та апаратною частиною.

Оскільки базою для web-застосунку, є web-сайт, який завантажується за допомогою браузера, а в технічному завданні застосунку, може стояти вимога взаємодіяти з операційною системою або з апаратною частиною пристрою, наприклад модулем GPS або Bluetooth.

Для рішення подібного технічного завдання, потрібно використовувати специфічний браузерний Web API. Сучасні браузери, такі як Google Chrome, Mozila FireFox, Sefari підтримують сучасні браузерні API.

Браузерний API(інтерфейс прикладного програмування) – це заздалегідь готові програмні конструкції, що дозволяють, силами браузера створювати важкий функціонал з меншими зусиллями. Браузерні інтерфейси прикладного програмування вбудовані в web-браузер і здатні використовувати дані браузера і комп'ютерного середовища для здійснення більш складних дій з цими даними. Наприклад, API Геолокації (Geolocation API) надає прості у використанні конструкції JavaScript для роботи з даними розташування, для того, що б визначити розташування об’єкту на карті. Всередині, в браузері виконується складний низькорівневий код (наприклад C# або C++) для підключення до модулю GPS або іншого модулю геолокації, для отримання даних і передачі їх браузеру для подальшої обробки створюваною програмою. Браузерна API ці складні взаємодії ховає всередині, даючи розробнику зручний інтерфейс для взаємодії.

На даний момент існують безліч, до розгляду представлено декілька цікавих:

Web Bluetooth API Дозволяє веб-сайтам обмінюватися даними через GATT із вибраними користувачем пристроями Bluetooth безпечним та не порушуючи конфіденційність способом.

WebUSB API веб-платформа для підтримки програмування USB-пристроїв.

Clipboard API надає можливість реагувати на команди буфера обміну (вирізати, копіювати та вставляти), а також асинхронно читати та записувати в системний буфер обміну. Доступ до вмісту буфера обміну здійснюється за API дозволів: Дозвіл на запис у буфер обміну надається сторінкам автоматично, коли вони знаходяться на активній вкладці. Потрібно запитати дозвіл на читання буфера обміну, що можна зробити, спробувавши прочитати дані з буфера обміну.

Vibration API дозволяє програмному коду забезпечувати фізичний зворотний зв'язок з користувачем, викликаючи вібрацію пристрою. API вібрації пропонує web доступ до цього апаратного модулю вібрації, якщо він існує, або нічого не вдіє, якщо в пристрої він відсутній.

API Network Information дозволяє визначити тип інтернет підключення системи (Wi-Fi, стільникова мережа). Інформацію можна використовувати, на приклад, для того, щоб надіслати оптимізувати, наприклад якість відео або зображень, під той чи інший тип інтернет підключення.

Таким чином браузерний API – це програмні конструкції, вбудовані в браузер, які всередені виконують складний, як правило низькорівневий програмний код, для взаємодії з операційною або апаратною частиною пристрою, взаємодія з браузерним інтерфейсом прикладного програмування здійснюється за допомогою мови JavaScript. Браузерний API призначений для полегшення web-розробки і розширення функціональності.

1.7 Web-застосунок в порівнянні з нативним застосунком.

Перевага гібридного web-застосунку над звичайним застосунком.

Основною перевагою є те що, web-застосунок одночасно являється web-сайтом, з цього випливають основні переваги.

Перша перевага стосується просування застосунку для потенційного користувача, для звичайного застосунку, що б його просунути на ринок, треба платити кошти за рекламу, web-застосунок має ж можливість просувати сам себе, за умови, що його SEO налаштований, тоді пошуковий робот піднімає його вище, в списку web-сайтів.

Друга перевага стосується того, вага web-застосунку і звичайного застосунку, для прикладу якщо взяти середню вагу web-сайтів це біля 2 мегабайт, і в порівняння з середньою вагою застосунка в App Store, ця вага складає 23 мегабайти. З цього видно що web-застосунок 10 разів легший, за звичайний застосунок, і це не максимум, існують випадки коли цей показник сягає десятків разів.

Третя перевага, стосується незалежності від майданчику розповсюдження застосунків. Розроблений застосунок можуть відхилити від розміщення на майданчику продажів застосунків, або пізніше видалити його з каталогу продажів, якщо порушиться хоча б одна з вимог політик цього майданчику продажів застосунків. Крім цього за розміщення власного застосунку може зніматись плата. Майданчики продажів застосунків працюють в різних країнах з різним державним устроєм та законодавством, тому не важко, зробити помилку, бо що одній країні дозволено, в іншій країні може бути під забороною, тим самим легко порушити хоча б одну з умов користування конкретним майданчиком продажів застосунків, що призведе до штрафу і\або видалення застосунок. З web-застосунком такого трапитись не може, так як він має власне представництво, свою web версію, з якої він і розповсюджується (завантажується).

Четверта перевага багатоплатформність web-застосунку, це значить що він буде працювати одночасно на всіх популярних операційних система, це IOS, Android, Windows. Що є великим плюсом, тому що це розширює потенційну аудиторію користувачів web-застосунком. В той час як звичайний застосунок, треба розробляти окремо під кожну платформу.

Як підсумок з меншими витратами, часу і коштів, розробляється багатоплатформний застосунок, при цьому це не вимагає залучення додаткових спеціалістів для розробки кожної версії застосунку окремо, так як вся розробка ведеться засобами web-розробки мовою JavaScript.

Крім цього залишається за потреби можливість використовувати web-застосунок в режимі web-сайту, що може бути корисно для певних ситуацій.

Зведена порівняльна табличка web-застосунку і нативного застосунку зображено в таблиці 1.1.

Таблиця 1.1 – Порівняльна таблиця web-застосунку і нативного застосунку

Характеристика	Web-застосунок	Нативний застосунок
Здатність застосунку до самопросування	+	–
Швидкість роботи	–	+
Швидкість завантаження	+	+
Залежність застосунку від посередників	+	–
Вага застосунку	+	–
Багатоплатформність рішення	+	–
Вартість і час розробки	+	–
Режим роботи в браузері	+	–
Можливість працювати в режимі відсутності підключення до мережі інтернет	+	+
Працювати в режимі застосунку	+	+
Зменшене навантаження на сервер	+	+

Підбивши підсумки порівняння можна зробити висновок, що удосконалення web-розкобки за рахунок PWA технологій та багатопотоковості, в результаті перетворює web-сайт на web-застосунок, що має цілий ряд переваг над звичайним типовим нитивним застосунком. Створюється багатоплатформна версія застосунку, але без залучення додаткових спеціалістів, під кожну окрему платформу, при цьому заощаджується час та кошти при розробці, що робить розробку web-застосунку більш вигідною і доступною.

1.8 Web-затосунок в порівнянні з web-сайтом. Текстом а внизу табличкою.

Переваги web-застосунку в порівнянні з звичайним web-сайтом очевидні, web-застосунок реалізує впровадження багатопотоковості, чим покращує UX

досвід використання, за рахунок більш високої швидкості виконання програмного коду в кількох потоках одночасно, що не спричиняє до блокування основного потоку виконання програми, а з ним і користувацького інтерфейсу.

Можливість працювати в режимі відсутності підключення до мережи інтернет, є наступною перевагою web-застосуноку наб звичайним web-сайтом, за рахунок різних стратегій кешування даних, і перехоплення мережевих запитів, вдається забезпечити безперебійну роботу web-застосуноку, в умовах поганої доступності мережі інтернет або повної відсутності мережі інтернет, що спричинило би втрату працездатності для web-сайту, або нестерпні умови для користування.

Вища швидкість завантаження даних, за рахунок розумних підходів кешування і оновлення даних, вдається забезпечити миттєве отримання даних з сервера, що є вирішальним для багатьох користувачів. Як показують дослідження якщо ресурс завантажуються більше трьох секунд, то більше 50% користувачів покидають цей web-сайт не дочекавшись повного завантаження.

За рахунок підвищення швидкості і роботи і завантаження web-застосуноку, пошуковий робот пошукових систем, додає вищу пріоритетність web-застосуноку над такими самими web-сайту, і піднімає його в результатах пошуку, що позитивно впливає на приплив нових користувачів.

Додається можливість завантажити цей web-сайт як web-застосунок, для швидкого доступу до нього, з робочого столу пристрою, що є зручно, користувач тепер не має пам’ятати назву ресурсу і постійно її вводити в пошук браузера, що б потрапити до нього.

Зведена порівняльна табличка web-застосунку і web-сайту зображено в таблиці 1.2.

Таблиця 1.2 – Порівняльна таблиця web-застосунку і web-сайту web-сайту

Характеристика	Web-застосунок	Web-сайту
Здатність застосунку до самопросування	++	+
Швидкість роботи	+	–
Швидкість завантаження	+	–
Залежність застосунку від посередників	+	+
Вага застосунку	+	+
Багатоплатформність	+	+
Вартість і час розробки	+	+
Режим роботи в браузері	+	+
Можливість працювати в режимі	+	–

відсутності підключення до мережі інтернет		
Працювати в режимі застосунку	+	–
Зменшене навантаження на сервер	+	–

Підбивши підсумки порівняння можна зробити висновок, що удосконалення web-розкобки за рахунок PWA технологій та багатопотоковості, в результаті створює web-застосунок, що має цілий ряд переваг над звичайним типовим web-сайтом. З режимом роботи в умовах відсутності з’єднання з інтернетом, і миттєвим завантаженям сторінки, що значно покращує досвід використання web-застосунку користувачем, при чому web-застосунок краще ранжування браузером. Що робить впровадження технологій PWA та багатопотоковості вигідним, та раціональним кроком, як для полегшення розробки виконавцю програмісту, так і для здешевлення кінцевого програмного продукту для замовника.

2. ПРОЕКТНА ЧАСТИНА

2.1 Постановка задачі

Основним завданням технічної частини дипломного проекту є розробка web-застосунку з використанням поєднання технологій PWA та багатопотоковості, що зможе собою заміщувати звичайний настільний застосунок, а також web-сайт, реалізовуючи основні їх можливості. Тема для web-застосунку — щоденний помічник.

2.1.1 Опис розділів web-застосунку

Сторінка входу в систему

Сторінка є основною точкою входу, яку в першу чергу побачить користувач при першому візиті.

Призначення сторінки входу в систему:

1. дати можливість аутентифікації та авторизації користувача;
2. дати можливість перейти на сторінку реєстрації нових користувачів.

Сторінка реєстрації в системі

Сторінка забезпечує можливість створити профіль в системі, для можливості повноцінного користування веб-застосунком.

Призначення сторінки входу в систему:

1. дати можливість реєстрації в системі нового користувача з підтвердженням електронної пошти;
2. дати можливість перейти на сторінку входу в систему.

Внутрішні сторінки (загальні вимоги)

Розділ «Погода»

призначення розділу

1. інформування користувача стосовно погодних умов на поточні шість діб, з сьогоднішньою включно;
2. Для кожної доби відображати температурний режим погодинно;
3. Опис
4. В основній частині розділу міститься інформація про погоду на 6 діб, кожна доба анімована в карточку, що припіднімається при наведені курсору.

«Панель навігації»

На сторінці повинні бути присутніми посилання на наступні розділи 2-го рівня:

1. «Погода»;
2. «Мотивація»;
3. «Задачі на день»;

4. «Здоров’я»;
5. «Профіль»

А також функціональні кнопки:

1. Кнопка «Вихід» для виходу з системи.

«Інформаційне вікно»

Спливаюче анімоване інформаційне вікно, для діалогу системи з користувачем.

Розділ «Мотивація»

призначення розділу

Розділ web-застосунку, призначений для мотивування та заохочення користувача ефективніше використовувати час, та робити нові досягнення.

опис

В основній частині розділу міститься побажання на день, та мотиваційне зображення. На кожен день система підбирає декілька унікальних побажань та мотивацій.

«Панель навігації»

На сторінці повинні бути присутніми посилання на наступні розділи 2-го рівня:

1. «Погода»;
2. «Мотивація»;
3. «Задачі на день»;
4. «Здоров’я»;
5. «Профіль»

А також функціональні кнопки:

1. Кнопка «Вихід» для виходу з системи.

«Інформаційне вікно»

Спливаюче анімоване інформаційне вікно, для діалогу системи з користувачем.

Розділ «Задачі на день»

призначення розділу

Дає можливість користувачу ефективно планувати свій день та контролювати виконання поставлених цілей.

опис

В головній частині розділу, розміщений анімований список задач, з реалізованим функціоналом: додати нову справу, помітити справу як виконану, або видалити зі списку справ.

«Панель навігації»

На сторінці повинні бути присутніми посилання на наступні розділи 2-го рівня:

1. «Погода»;
2. «Мотивація»;
3. «Задачі на день»;

4. «Здоров’я»;
5. «Профіль».

А також функціональні кнопки:

1. Кнопка «Вихід» для виходу з системи.

«Інформаційне вікно»

Спливаюче анімоване інформаційне вікно, для діалогу системи з користувачем.

*Розділ «*Здоров’я»

призначення розділу

1. дає можливість вносити показники стосовно здоров’я;
2. дає можливість переглядати динаміку змін показників;
3. дає можливість отримати характеристику стану тіла.

опис

В головній частині розділу, в стилі поштового листа, розміщені поля для вводу поточної ваги, кількість випитої рідини, час в який користувач прокинувся та час в якому користувач пішов спати.

Нище можна розрахувати стан свого тіла, і отримати коментар з рекомендацією.

Ще нижче знаходяться показники в діаграмах, де можна бачити динаміку зміни показників поденно. Діаграми по трьох основних показниках: вага, вода, сон.

«Панель навігації»

На сторінці повинні бути присутніми посилання на наступні розділи 2-го рівня:

1. «Погода»;
2. «Мотивація»;
3. «Задачі на день»;
4. «Здоров’я»;
5. «Профіль»

А також функціональні кнопки:

6. Кнопка «Вихід» для виходу з системи.

«Інформаційне вікно»

Спливаюче анімоване інформаційне вікно, для діалогу системи з користувачем.

*Розділ «*Профіль»

призначення розділу

1. дає можливість вносити основну інформацію про користувача;

опис

В головній частині розділу, в стилі поштового листа, з лівої частини розміщено актуальні дані про користувача, в правій частині розміщена форма полів для вводу/редагування даних: ім’я користувача, зріст, місто, рік народження.

«Панель навігації»

На сторінці повинні бути присутніми посилання на наступні розділи 2-го рівня:

1. «Погода»;
2. «Мотивація»;
3. «Задачі на день»;
4. «Здоров'я»;
5. «Профіль»

А також функціональні кнопки:

6. Кнопка «Вихід» для виходу з системи.

«Інформаційне вікно»

Спливаюче анімоване інформаційне вікно, для діалогу системи з користувачем.

2.1.2 Структура web-застосунку

Загальна структура web-застосунку представлена на рис. 2.1.

Рисунок 2.1 – Загальна структура web-застосунку

Web-застосунок використовує архітектура SPA(Single Page Aplication), тобто, технічно це односторінковий web-застосунок, що завантажується на одній сторінці. Завдяки динамічному оновленню за допомогою JavaScript, під час використання не потрібно перезавантажувати або довантажувати додаткові сторінки. Тобто застосовується підхід AJAX до побудови, користувацьких інтерфейсів, що полягає в «фоновому» обміні даними застосунка з web-сервером. В результаті при оновленні даних web-застосунок не перезавантажується повністю, і web-застосунки стають швидше і зручніше. На практиці це означає, що користувач бачить в браузері весь основний контент, а під час переміщення або переходах на інші сторінки, замість повного перезавантаження потрібні елементи просто завантажуються.

В процесі роботи користувачу може здатися, що він запустив не web-застосунок, а нативний застосунок, так як він миттєво реагує на всі його дії, без затримок і «підвисань». Такий ефект отримується з допомогою використання клієнтської JavaScript бібліотеки React.

Серед переваг архітектури:

1. Висока швидкість - все ресурси завантажуються за одну сесію, а під час дій на сторінці дані просто змінюються, що дуже економить час;
2. Гнучкість користувацького інтерфейсу - за рахунок того, що web-застосунок це одна сторінка, простіше побудувати насичений інтерфейс, зберігати відомості про сеанс, управляти станами застосунку і анімацією;
3. Спрощена розробка - код можна починати писати з файлу file: // URL, не використовуючи сервер, не потрібен окремий код для рендера сторінки на стороні сервера;
4. Кешування даних - застосунок відправляє всього один запит, збирає дані, а після цього може функціонувати в офлайн-режимі.

2.1.3 Структура серверної частини

Серверне API має бути побудоване за REST архітектурою, що передбачає:

1. Клієнт-серверну модель. Розділення відповідальності між сервером(оновлення і збереження даних), і клієнтом(відображення даних через користувацький інтерфейс). Таке розділення дозволяє, цим двом сторонам розвиватись незалежно від один одного.
2. Відсутність стану, тобто кожен запит з клієнта на сервер містить всю необхідну інформацію для його обробки, і не покладається на те, що сервер знає щось з попереднього запиту клієнта. Тобто дані про сесію зберігаються на клієнті, а не на сервері, і передаються на сервер з кожним запитом, чим знімає навантаження з сервера.
3. Кешування, тобто дані які надходять з сервера на клієнт, повинні мати інформацію про те чи можна їх закешувати, та на який термін.
4. Однорідний інтерфейс, тобто клієнт, має доступ до всього функціоналу через гіперпосилання на REST API, який має структуру маршрутизації, але також треба додати до посилання всі необхідні данні, щоб запит міг бути опрацьований.

Маршрутизація для сервера:

2.1 '/api/auth/' – набір посилань для реєстрації та аутентифікації і авторизації.

2.2 '/api/auth/registration' – серверний інтерфейс, що приймає від клієнта JSON з електронною поштою, паролем з двох полів, потрібен для створення нового користувача в системі.

2.3 '/api/auth/login' – серверний інтерфейс, що приймає від клієнта JSON з даними електронної пошти та паролем, для аутентифікації та авторизації в системі, якщо користувач існує.

2.4 '/api/auth/logout' – серверний інтерфейс для виходу з системи користувача.

2.5 '/api/auth/activate/:link' – серверний інтерфейс для підтвердження електроної пошти користувачем.

2.6 '/api/auth/refresh' – серверний інтерфейс для оновлення пари токенів, доступу та перезаряджального токенів.

2.7 '/api/task' – набір посилань для роботою з списком справ.

2.8 '/api/task/daily' – серверний інтерфейс, що повертає список існуючих задач користувача для даного профілю.

2.9 '/api/task/add' – серверний інтерфейс, отримує від клієнта JSON об’єкт з даними для нової задачі, і зберігає в базі даних для даного профілю.

2.10 '/api/task/toggle' – серверний інтерфейс, що встановлює для задачі позначку «виконано» або знімає її, якщо вона існує, зберігає зміни в базу даних для даного профілю.

2.11 '/api/task/remove' – серверний інтерфейс, що видаляє обрану користувачем задачу, з бази даних для даного профілю.

2.12 '/api/motivation' – набір посилань для роботи з мотиваційною системою.

2.13 '/api/motivation/daily' – серверний інтерфейс, що повертає на клієнт одне побажання з кількох визначених на день, а також мотиваційне зображення.

2.14 '/api/forecast' – набір посилань для роботи з інформацію щодо погоди

2.15 '/api/forecast/week' – серверний інтерфейс, що повертає прогноз погоди на тиждень, для конкретного міста, з широким набором погодних показників.

2.16 '/api/account' – набір посилань для налаштування профілю користувача

2.17 '/api/account/setName' – серверний інтерфейс, що зберігає дані про ім’я користувача в базі даних для даного профілю.

2.18 '/api/account/setBirth' – серверний інтерфейс, що зберігає дані про рік народження користувача в базі даних для даного профілю.

2.19 '/api/account/setCity' – серверний інтерфейс, що зберігає дані про місто користувача в базі даних для даного профілю.

2.20 '/api/account/setHeight' – серверний інтерфейс, що зберігає дані про зріст користувача в базі даних для даного профілю.

2.21 '/api/account/getAccountData' – серверний інтерфейс, що відправляє на клієнт дані профіля даного користувача в системі.

2.22 '/api/health' – набір посилань для роботи з системою моніторингу здоров’я користувача.

2.23 '/api/health/info' – серверний інтерфейс, що відправляє на клієнт дані користувача ствосовно здоров’я, для побудови графіків динаміки показників здоров’я користувача, для даного профілю.

2.24 '/api/health/today' – серверний інтерфейс, що відправляє на клієнт поточні дані користувача на сьогоднішній день стосовно здоров’я, для даного профілю.

2.25 '/api/health/update-wb' – серверний інтерфейс, для реєстрації кожного прийому води за день, для відстежування водного балансу, дані зберігаються в базі даних.

2.26 '/api/health/set-weight' – серверний інтерфейс, для реєстрації ваги користувача щоденно, для відстежування динаміки змін ваги, дані зберігаються в базі даних.

2.27 '/api/health/set-gut' – серверний інтерфейс, для реєстрації часу о котрій прокинувся користувач щоденно, для відстежування динаміки показників сну, дані зберігаються в базі даних.

2.28 '/api/health/set-st' – серверний інтерфейс, для реєстрації часу о котрій користувач ліг спати, щоденно, для відстежування динаміки показників сну, дані зберігаються в базі даних.

2.1.4 Сутності бази даних

База даних має всього чотири сутності «health», «motivation», «task», «user» що забезпечують даними весь web-застосунок.

1. Сутність «health» відповідає за дані пов’язані з функціоналом моніторингу стану здоров’я користувача:

1.1 'id' – унікальний ідентифікаційний номер;

1.2 'user_id' – посилання на унікальний ідентифікаційний номер користувача;

1.3 'date' – дата запису;

1.4 'weight' – вага користувача;

1.5 'wake up' – час о котрому користувач прокидається;

1.6 'go to sleep' – час о котрому користувач лягає спати;

1.7 'water' – кількість випитої води користувачем за день.

2. Сутність «motivation» відповідає за дані пов’язані з функціоналом для піднесення психологічного стану користувача:

2.1 'id' – унікальний ідентифікаційний номер;

2.2 'wish' – побажання;

2.3 'motivation image' –посилання на мотиваційне зображення.

3. Сутність «task» відповідає за дані пов’язані з функціоналом структурування завдань користувача:

3.1 'id' – унікальний ідентифікаційний номер;

3.2 'user id' – посилання на унікальний ідентифікаційний номер користувача;

3.3 'title' – текст задачі;

3.4 'completed' – так чи ні, відмітка чи виконано завдання.

4. Сутність «user» відповідає за дані пов’язані з функціоналом доступу до системи і безпеки користувача, а також налаштування профіля:

1 'id' – унікальний ідентифікаційний номер;

2 'name' – ім’я користувача;

3 'birth day' – день народження користувача;

4 'height' – зріст користувача;
5 'city' – місто перебування користувача;
6 'email' – електрона пошта користувача;
7 'password' – пароль користувача в зашифрованому виді;
8 'refresh token' – перезаряджальний токен для безпеки даних;
9 'activation link' – посилання на активацію;
10 'activated' – так чи ні, позначка чи активований профіль (підтверджена пошта);
11 'role' – роль користувача в системі.

2.1.5 Стадії і етапи розробки

На таблиці 2.1 зображується стадії і етапи розробки, зміст роботи і виконавці.

Таблиця 2.1 – Результати розрахунків

№	зміст роботи	термін	Виконавець етапу розробки
1	Дослідження різних концепцій web-розробки	1-2 тижні	Петрухно Т.О
2	Вироблення власної методології, шляхом поєднання технологій	3-й тиждень	Петрухно Т.О
3	Розробка технічного завдання для демонстрації нової методології.	4-й тиждень	Петрухно Т.О
4	Розробка візуального оформлення для web-застосунку.	5-7 тижні	Петрухно Т.О
5	Розробка серверної частини web-застосунку REST API.	5-7 тижні	Петрухно Т.О
6	Створення клієнтської частини з впровадженням напрацьованої методології.	8-10 тижні	Петрухно Т.О
7	Тестування взаємодії серверу і клієнта.	8-10 тижні	Петрухно Т.О
8	Тестування web-застосунку створеного по напрацьованій методології на виконання, передбачених нею переваг.	11 тиждень	Петрухно Т.О
9	Здача і захист проекту.	12 тиждень	Петрухно Т.О

2.1.6 Призначення і цілі створення web-застосунку

Цільова аудиторія web-застосунку

Цільова аудиторія сайту представлена наступними групами користувачів:

1. громадські організації;
2. люди 14-55 років;
3. бізнес.

Цілю створення web-застосунку є заміна ним класичного підходу, створення web-сайту, звичайного настільного застосунку і мобільного застосунку, з метою оптимізації часу написання коду, грошових ресурсів, та загальної складності проекту.

Розроблений web-застосунок виконує роль щоденного помічника, допомагає ефективно та передбачувано планувати час. Web-застосунок щоденно надихає на роботу та нові досягнення. Також допомагає відслідковувати кількість годин сну, та години коли користувач лягає спати та прокидається, відображуючи ці дані зручним графіком, також допомагає відслідковувати кількість випитої води за день і щоденну вагу. При потребі web-застосунок може оцінити стан тіла та дати свою рекомендацію користувачу, згідно формул «Всесвітньої організації охорони здоров'я» Він підтримує роботу в режимі відсутності підключення до мережі інтернет. Може бути завантажений як настільний застосунок, а також може працювати в режимі інтернет сайту.

За рахунок багатопотоковості може виконувати важкі підрахунки в фоні, не блокуючи головний потік застосунку, що позитивно впливає, на досвід використання користувачем програмного забезпечення.

2.1.7 Вимоги до web-застосунку

Основні вимоги:

1. Замінювати собою web-сайт;
2. Працювати в режимі відсутності підключення до мережі інтернет;
3. Завантажуватись на пристрій, аналогічно настільним застосункам;
4. Не мати залежності від майданчиків розповсюдження застосунків;
5. Працювати багатопотоково;
6. Багатобраузерність;
7. Багатоплатформність;
8. Легкість подальшої підтримки програмного продукту;
9. Низька ціна розробки та підтримки порівняно з класичним підходом розробки.

Вимоги до компонування сторінок web-застосунку

1. компонування сторінок web-застосунку повинна забезпечувати автоматичне масштабування сторінок в залежності від ширини робочого поля браузера користувача.

2.2 Вимоги до програмного та апаратного забезпечення

1. Операційна система: Windows XP, Windows, Windows Vista, Windows 7, Windows 8, Windows 10, Windows 11;

2. Один з браузерів Google Chrome, Android Browser, Edge, Opera Mobile, FireFox for Android, Chrome for Android, Samsung Internet, QQ Browser, Baidu Browser.
3. ОЗУ 512 МБ;
4. Жорсткий диск 350 МБ;
5. Відео пам’ять 64 МБ;
6. Наявність контролерів клавіатури та миші;
7. наявність мережевої карти;
8. наявність інтернет з’єднання.

2.3 Програмні засоби і середовище розробки

2.3.1 Середовище розробки

Visual Studio Code — засіб для створення, редагування та підтримки сучасних web-застосунків і програм для хмарних систем. Visual Studio Code розповсюджується безкоштовно і доступний у версіях для платформ Windows, Linux і OS X.

2.3.2 Інтерфейс web-застосунку

Інтерфейс web-застосунку, виконується стандартними інструментами web-розробки, використовується бібліотека React:

Javascript - це сценарій/мова програмування, що дозволяє реалізовувати складні функції на веб-сторінках, кожен раз, коли веб -сторінка робить більше, ніж просто відображає статичну інформацію. Робить сторінку динамічною, відображає своєчасно оновлення вмісту сторінки, інтерактивні можливості, анімовані 2D/ 3D - графіки, робота з відео та/або музичними матеріалами, бізнес-логікою тощо.

CSS(каскадні таблиці стилів) - це мова розмітки, яка пропонує додаткові можливості для форматування тексту та блоків для мови розмітки HTML. Дає можливість задавати, шрифти висоту, ширину, кольори виконання, відступи, анімації і багато чого іншого, що стосується стилізації.

HTML(мова розмітки гіпертексту) - це стандартна мова розмітки, виконує функцію створення структури сторінки, і використовується для створення web-сторінок. Остання версія мови розмітки HTML5.

2.3.3 Мова програмування

Мовою програмування для web-застосунків виступає мова JavaScript, мова реалізації всієї бізнес-логіки, мова реалізації динамічного користувацького інтерфейсу, мова реалізації серверної частини, на платформі NodeJS, а також можливий варіанти, коли може реалізувати взаємодію з базою даних при використанні NoSQL баз даних. JavaScript використовується для реалізації функціоналу кешування, багатопотоковості, та завантажуваності на пристрій.

2.3.4 Мова програмування баз даних

Мовою запитів баз даних використовуються SQL(мова структурованих запитів) – використовується для створення і модифікації схем баз даних, створення,

оновлення, видалення, індексації і оновлення полями реляційних баз даних. SQL не є самостійним, для керування реляційними базами даних, потрібно використовувати систему керування базами даних, в проекті використовується система керування базами даних MySQL.

2.3.5 Платформа сервера

Платформою серверною частини є NodeJS.

NodeJS – реалізований на двигуні V8, і виконує JavaScript код в своєму середовищі, що дає можливість запускати JavaScript код поза web-браузерами. Таким чином може бути використаний для написання серверної частини, маршрутизації на сервері, обробкою мережевих запитів, і видачі актуального контенту на клієнт, для створення web-застосунків, з динамічним контентом.

Стосовно хостинг-платформи сервера використовується платформа Heroku.

Heroku – хмарна платформа, що підтримує платформу NodeJS та багато інших мов програмування, до прикладу PHP, Java, Pyton т інші. Дає можливість розмістити на хостинг, клієнтську частину, серверну частину та базу даних.

3 ПРАКТИЧНА ЧАСТИНА

3.1 Кешування серверних запитів

Для того, що б система могла працювати в режимі відсутності підключення до мережі інтернет, реалізовано кешування за допомогою Web API Service Worker. Для створення сервісного працівника Service Worker в проекті використовується набір бібліотек WorkBox, вони можуть бути завантаженні через пакетний менеджер NodeJS npm або використовуватись через посилання cdn.

Для налаштування WorkBox спочатку створюється конфігураційний файл workbox-build-inject.js, в якому описується налаштування для генерації Service Worker.

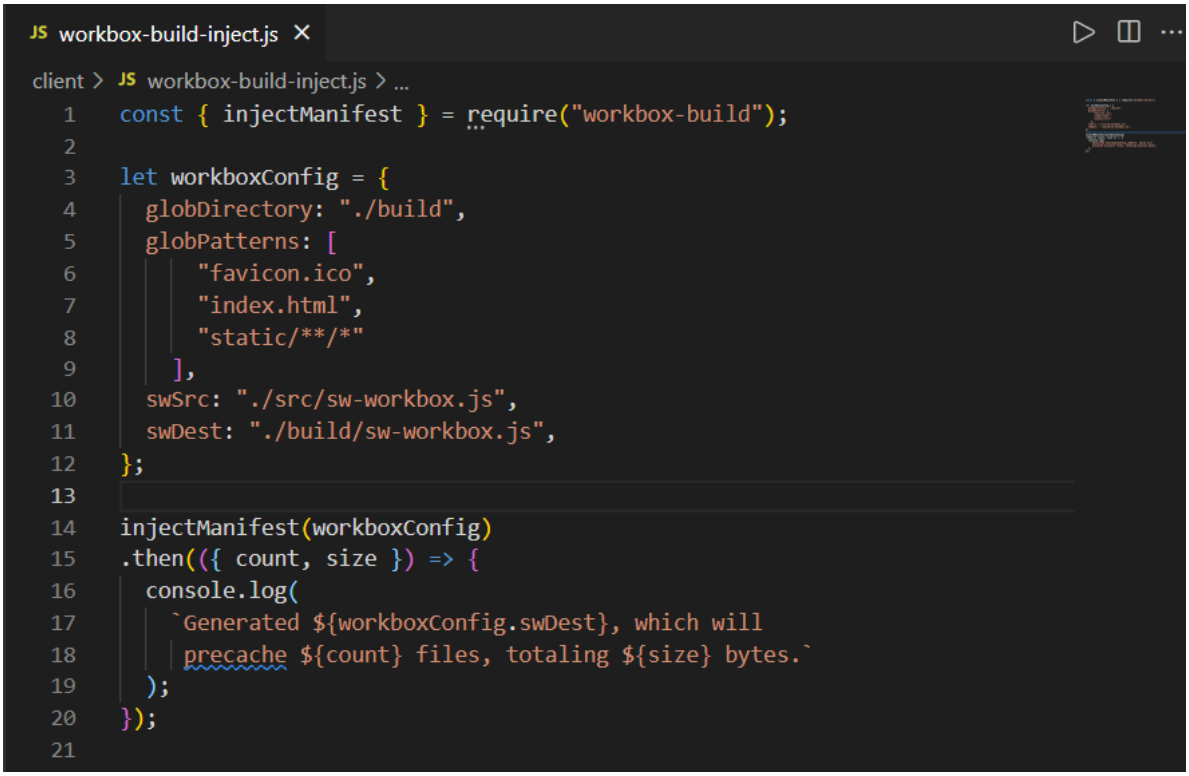

Рисунок 3.1 – Конфігураційний файл для генерації Service Workera

В поле об’єкту конфігурацій «globPatterns» передається масив з назвами статичних файлів, що потрібно закешувати, а в «globPatterns» відносний шлях до статичних файлів на клієнті, які мають бути закешовані і модифікації перекешовані знову. В «swSrc» вказано файл з кодом для workbox під назвою sw-workbox.js (зображено на рисунку 3.2), в якому описуються деталі реалізації Service Worker. А в «swDest» куди має згенеруватись Service Worker. Генерація відбувається за допомоги функції injectManifest, модуля «workbox-build», що встановлений як npm модуль на NodeJS.

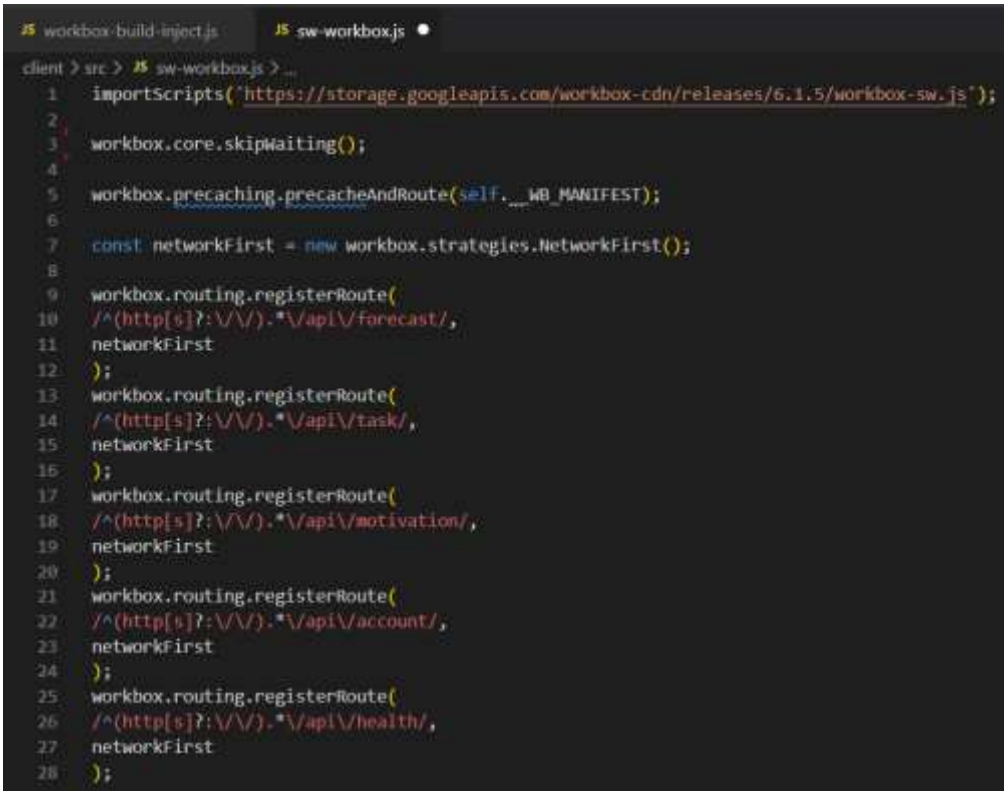

Рисунок 3.2 – Налаштунок Service Worker

В файлі sw-workbox.js відбувається прекешинг статичних ресурсів на 5 рядку коду, це відбувається легко, тому що всі файли і шляхи до файлів вже налаштовані в конфігураційному файлі.

Прекешинг статичних ресурсів використовує стратегію кешування під назвою «stale-while-revalidate» в перекладі «попередні дані під час оновлення», суть якої полягає в тому, що service worker перехоплює запит з клієнта на сервер, і спочатку перевіряє сховище кешу «Cache Storage», якщо за цим запитом є закешовані дані, то в такому випадку service worker відповідає цими даними з сховища кешу «Cache Storage» на запит з клієнта, паралельно надсилаючи при цьому запит мережею інтернет до сервера, що б отриману відповідь помістити в сховище кешу «Cache Storage», там самим оновивши його для наступного запиту. Тобто, кожен раз ми отримуємо відповідь з миттєвою швидкістю, але кешовані дані від попереднього запиту «несвіжі», і паралельно готуємо останні актуальні дані на цей момент для наступної видачі.

Також на рисунку зображені зареєстровані маршрути, і назва стратегії кешування, в дипломному проекті для реєстрації маршрутів використана стратегія кешування network-first, тобто спочатку з інтернету, а якщо інтернет з'єднання відсутнє то тоді з кешу, і при кожному запиті сховище кешу «Cache Storage» оновлюється новими даними.

Після цього за допомогою цього файлу sw-workbox.js і конфігураційного файлу workbox-build-inject.js генерується вже готовий Service Worker, який залишається зареєструвати.

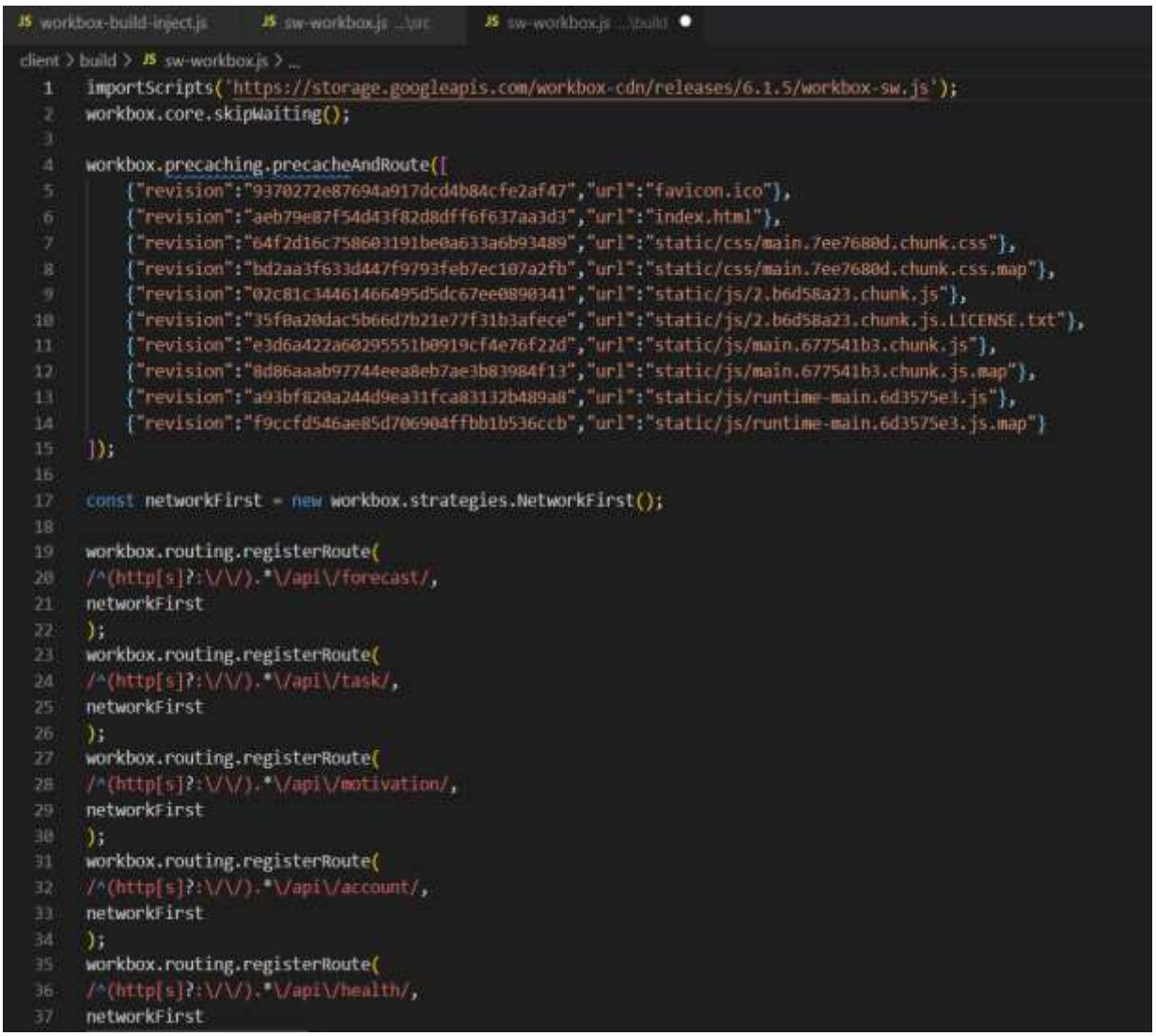

Рисунок 3.3 – Сгенерований Service Worker

Після цього масив статичних файлів наповнюється, об’єктами, що містять посилання на статичний ресурс «url», та хеш сумами цих файлів «revision», хеш суми потрібні, вони зберігаються в браузері користувача web-застосунку, і коли з сервера приходять нові модифіковані файли, клієнт побачить різницю між хеш сумами старих файлів і нових файлів, а якщо хеш-суми файлів змінились, це сигнал для Service Worker треба оновити кеш, і Service Worker в фоні оновить кеш. Так як стратегія кешування, прекешингу статичних файлів «stale-while-revalidate» або «попередні дані під час оновлення» то нові файли користувач побачить тільки при наступному оновлені сторінки, такий підхід використовується для критично важливих ресурсів.

Після того як Service Worker згенерований його треба зареєструвати в HTML файлі, але в момент коли сторінка вже завантажена, для цього є подія «onload» викликається через глобальний об’єкт window window.addEventListener('load', func). В func його активувати через функцію «register» аргументом якій передається шлях до згенерованого Service Worker, функція належить «window.navigator.serviceWorker». Тобто реєстрація буде виглядати так navigator.serviceWorker.register('./sw-workbox.js').

```
export const register =  () => {
    if ('serviceWorker' in navigator) {
      window.addEventListener('load', () => {
      navigator.serviceWorker.register('./sw-workbox.js')
    });
  }
}
```

Рисунок 3.4 – Реєстрація Service Worker

Після цього Service Worker якщо все пройшло успішно Service Worker активується і приступить до роботи, визначені статичні і динамічні ресурси зможуть бути закешовані і працювати в режимі відсутності підключення до мережі інтернет, поставлена ціль досягнута.

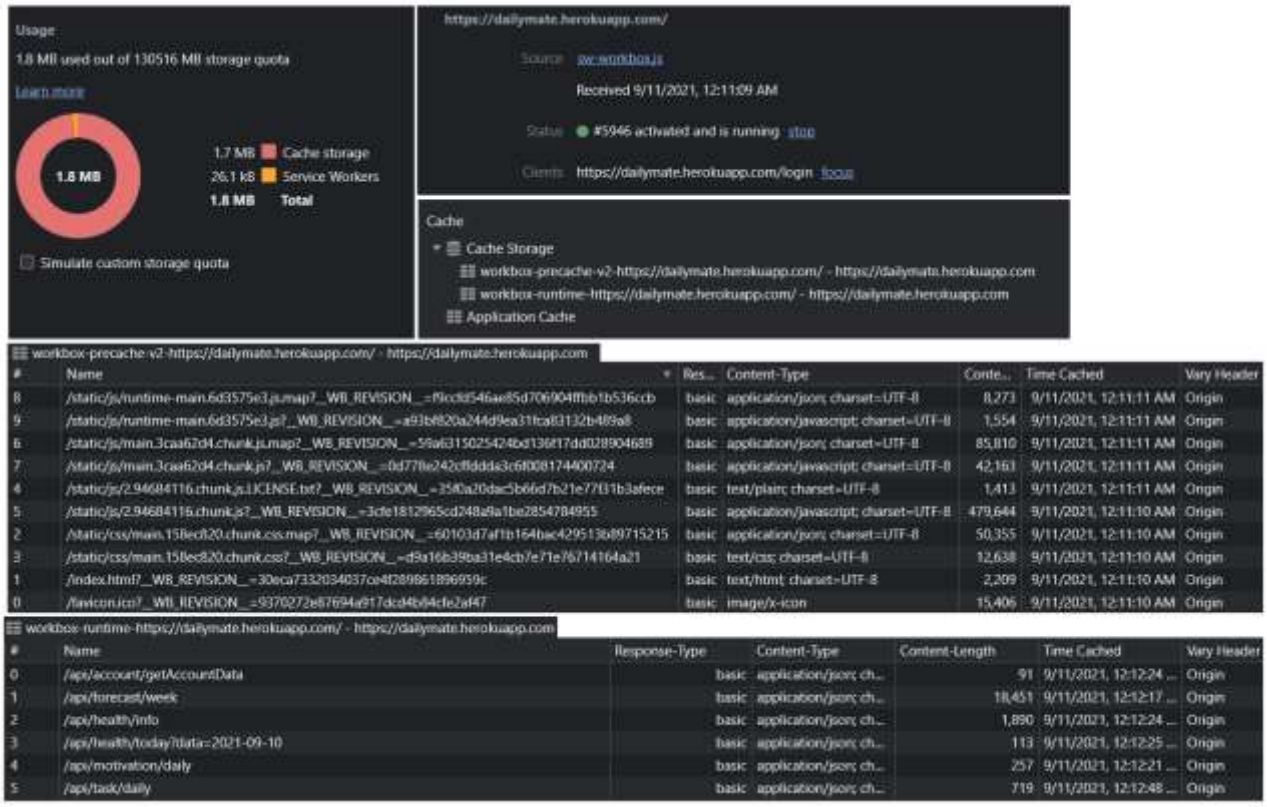

Рисунок 3.5 – Результати роботи Service Worker

Service Worker разом з усім закешованим даними в сховищі «Cache Storage» зайняли всього 1.8МБ, з яких сам Service Worker займає 26.1 кБ. Service Worker був успішно активований і запущений.

В сховище кешу «Cache Storage» видно, що створені два нових розділи:

1. Workbox-precache-v2-https://dailymate.herokuapp.com/ - https://dailymate.herokuapp.com/ (статичні дані App Shell каркас web-застосунку, в які увійшли 9 файлів);
2. Workbox-runtime-https://dailymate.herokuapp.com/ - https://dailymate.herokuapp.com/ (динамічні дані, в які увійшли 6

мережевих запитів, які були викликані при тестуванні і встигли закешуватись).

Сумарний об’єм закешованих даних дипломного проекту у сховищі кешу «Cache Storage» замає 1.7 МБ.

Для масштабних проектів місця сховища кешу «Cache Storage» теж вистачить, так як максимальний розмір сховища кешу «Cache Storage» становить 2147483647 байтів, або 2147 МБ або трохи менше ніж 2 ГБ.

3.2 Встановлюваність web-застосунку

Для того що б web-застосунок працювати в режимі настільного застосунку треба, реалізувати можливість виклику по іконці на робочому столі, в окремому власному вікні, як працюють настільні застосунки. Для реалізації цього, потрібно розробити іконки під різні пристрої, з різною роздільною здатністю, а також розробити спеціальний JSON файл з налаштуваннями для завантаження під спеціальною назвою manifest.json. Нижче наведений приклад маніфесту, з реалізованого дипломного проекту на рисунку 3.6

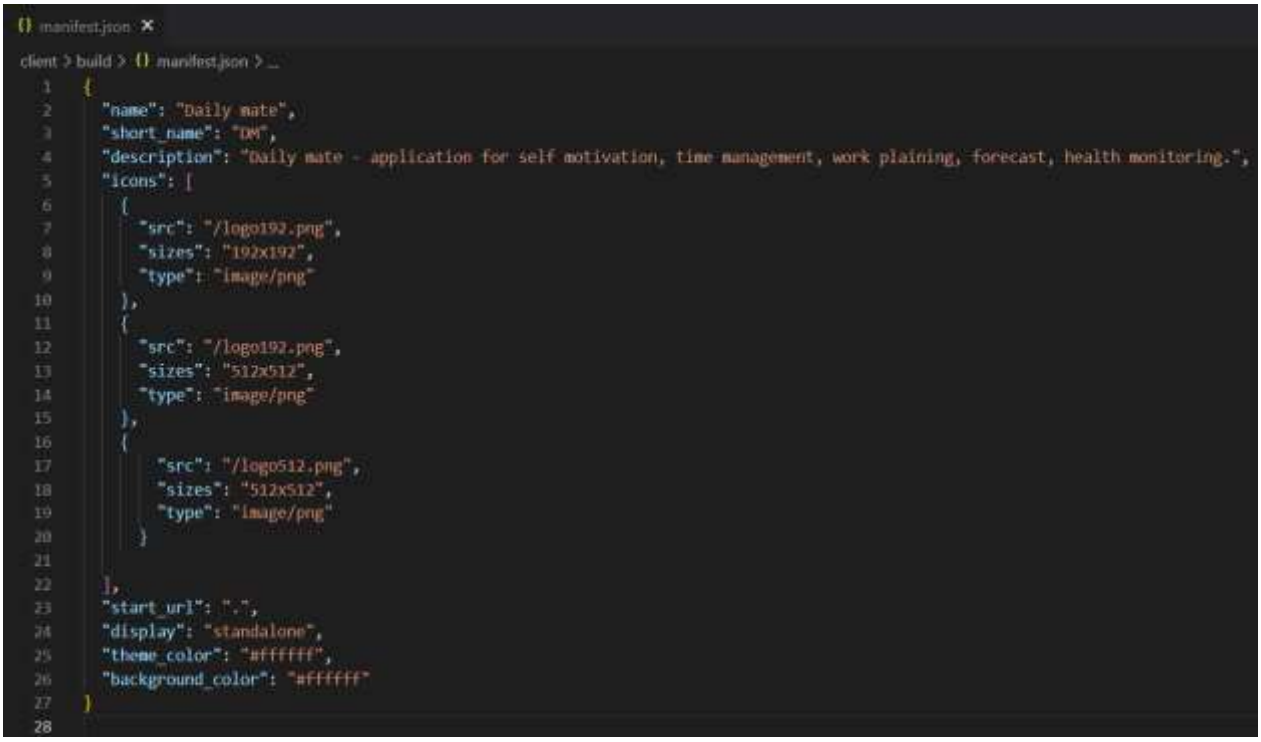

Рисунок 3.6 – Реалізація Web Manifest

Маніфест містить всі необхідні налаштування, підключається до HTML файлу

<link rel="manifest" href="./manifest.json" />. Дає можливість web-застосунку бути встановленим на домашній екран пристрою без магазину розповсюдження застосунків.

Властивості manifest.json:

1. «name» — повне ім’я web-застосунку;

2. «short_name» — визначає назву web-застосунку для демонстрації користувачу, якщо для відображення «name» мало місця (наприклад, в якості написи під іконкою застосунку на екрані телефону).

3. «description» — це рядок, в якому розробники можуть описати, призначення засторонку, тобто що він робить.

4. «icons» — визначає масив об'єктів зображень, які можуть використані як іконки програми в різних контекстах. Наприклад, вони можуть бути використані для подання застосунку серед списку інших застосунків або для інтеграції його з перемикачем завдань ОС і / або налаштувань системи;

4.1 «src» — шлях до файлу зображення. Якщо src є відносним URL, основним URL буде URL маніфесту;

4.2 «sizes» — рядок, що містить розміри зображення.

4.3 «type» — служить для визначення медіа-типу зображення. Мета властивості дозволити агенту користувача проігнорувати зображення медіа-типу, який воно не підтримуване.

5. «start_url» — є рядком, що представляє початкову URL-адресу web- застосунку – оптимальну URL-адресу, яка повинна бути завантажена під час запуску користувачем web-застосунку (наприклад, коли користувач натискає на значок web-застосунку в меню застосунків або на домашньому екрані);

6. «display» — визначає бажаний розробником режим відображення для web-застосунку. Режим відображення змінює тип відображення для користувача інтерфейсу і може варіюватися від "browser" (коли відображається в повне вікно браузера) до "fullscreen" (коли застосунок повноекранний);

7. «theme_color» — визначає колір теми за замовчуванням для програми;

9. «background_color» — визначає очікуваний колір фону для web- застосунку;

Якщо зазначені властивості вказані, іконки додані, і manifest.json правильно підключений до HTML файлу, web-застосунок отримує можливість бути завантаженим. Для цього необхідно в правому куточку в рядку пошуку браузера натиснути на відповідну іконку, що виглядає як монітор комп’ютера в який вписана стрілочка направлена вниз. Після натискання браузер, ще раз перепитає, чи дійсно є намір встановити web-застосунок, після підтвердження web-застосунок буде успішно встановлений, і на робочому екрані з’явиться його іконка, дивитись рисунок 3.7

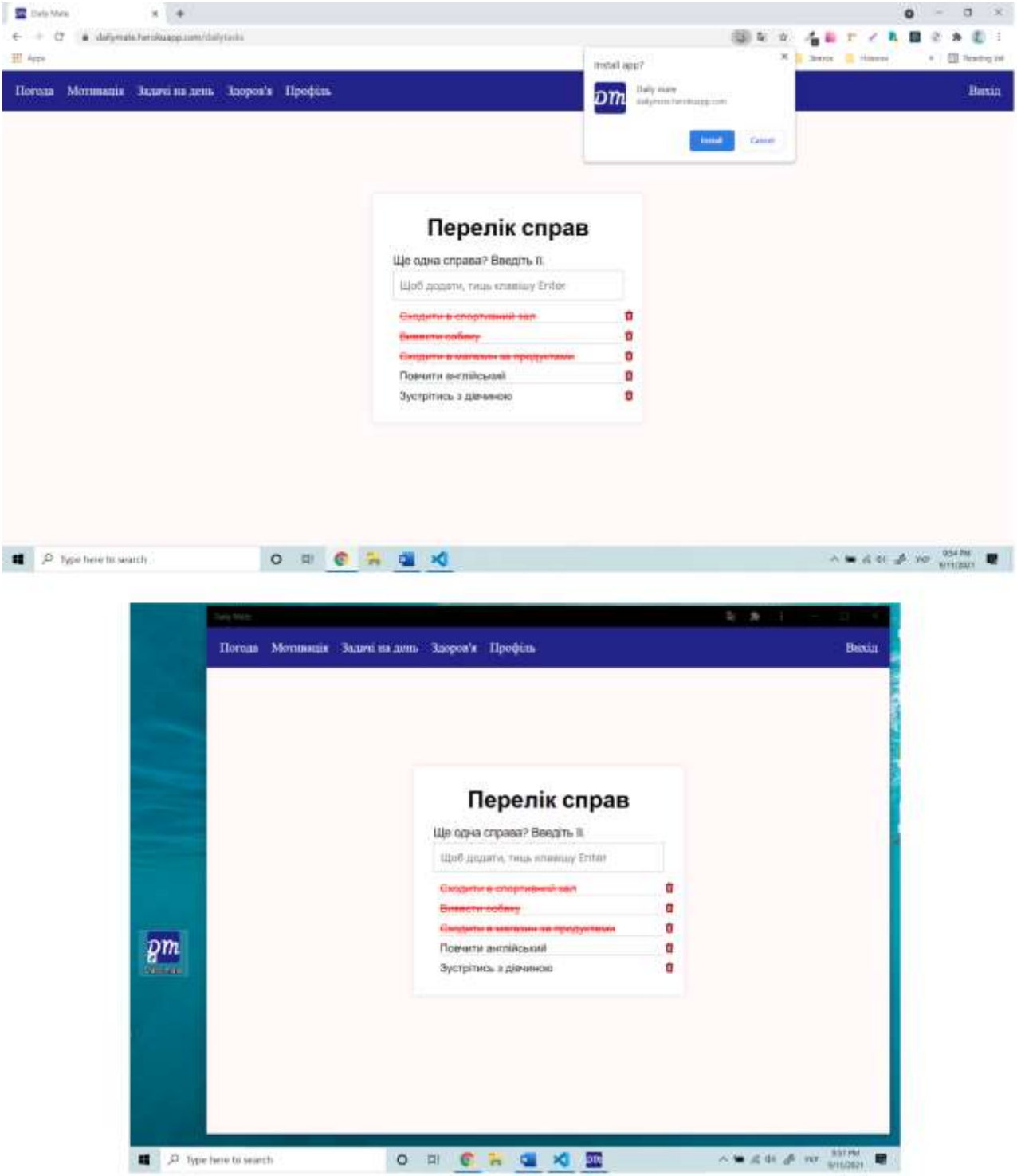

Рисунок 3.7 – Завантаження на пристрій web-застосунку

Коли користувач знаходиться в режимі застосунку в налаштуваннях є пункт відкрити web-застосунок в браузері, і за потреби можна здійснити легкий перехід в режим web-сайту, без вводу адреси в поле пошуку, або ж якщо користувач знаходиться на в режимі web-сайту, на місці де розташований ярличок завантажити web-застосунок, якщо він був вже завантажений буде інший ярличок, який відкриє web-застосунок в режимі настільного застосунку, що є дуже зручним переходом.

3.3 Багатопотоковість для ресурсоємних функцій

Для того, що б система могла працювати не втрачаючи працездатності, в момент коли опрацьовується ресурсоємна функція, наприклад з важким сортуванням, або факторіалами, або величезними циклами, що блокують потік виконання і поки ця функція не виконається система не буде реагувати ні на які дії користувача, що звісно є дуже погано з точки зору досвіду використання користувацького інтерфейсу. Так трапляється із за того, що мова JavaScript сама по собі є однопотоковою з можливістю асинхронних викликів. Тому використання багатопотоковості за допомогою Worker-ів браузера є рішенням цієї проблеми.

Процес створення Worker-а можна побачити на рисунку 3.8

use-webworker.js | HealthInfo.tsx

client > src > hooks > use-webworker.js > useWebWorker > run

```
1   import { useState, useEffect, useRef } from "react"
2
3   // worker inner
4   const workerHandler = (fn) => {
5       onmessage = (event) => {
6           console.log(event.data)
7           postMessage( fn(event.data) )
8       }
9   }
10
11  //  in main tread
12  export const useWebWorker = (fn) => {
13
14      const [result, setResult] = useState();
15
16      const run = (value) => {
17          const worker = new Worker(
18              URL.createObjectURL(new Blob([`(${workerHandler})(${fn})`]))
19          )
20
21          worker.onmessage = (event) => {
22              setResult(event.data)
23              worker.terminate();
24          }
25          worker.postMessage(value);
26      }
27
28      return {
29          result,
30          run,
31      }
32  }
```

Рисунок 3.8 – React хук для web worker

Функція useWebWorker створює новий одноразовий Worker, передає в неї код для Worker-а і вказує, що цей Worker має прийняти ресурсоємну функцію, що має бути запущено в своєму потоці, і зразу після виконання передати відповідь назад в головний потік коду і worker має знищитись. Функцію «postMassege» передається значення для ресурсоємної функції в Worker-і, і в середині Worker-а він приймає це значення подією «onmesssage» виконує обчислення і повертає результат виконання в потік функцію «postMassege», і в головному потоці спрацьовує подія «onmesssage» приймає результат розрахунків, worker свою задачу виконав і викликається функція «terminate» для знищення Worker-а. Тобто мій головним потоком відбувається двосторонній почерговий обмін даними, що дуже нагадує роботу web socket. Демонстрація на UML Sequence diagram діаграмі послідовності рисунок 3.9

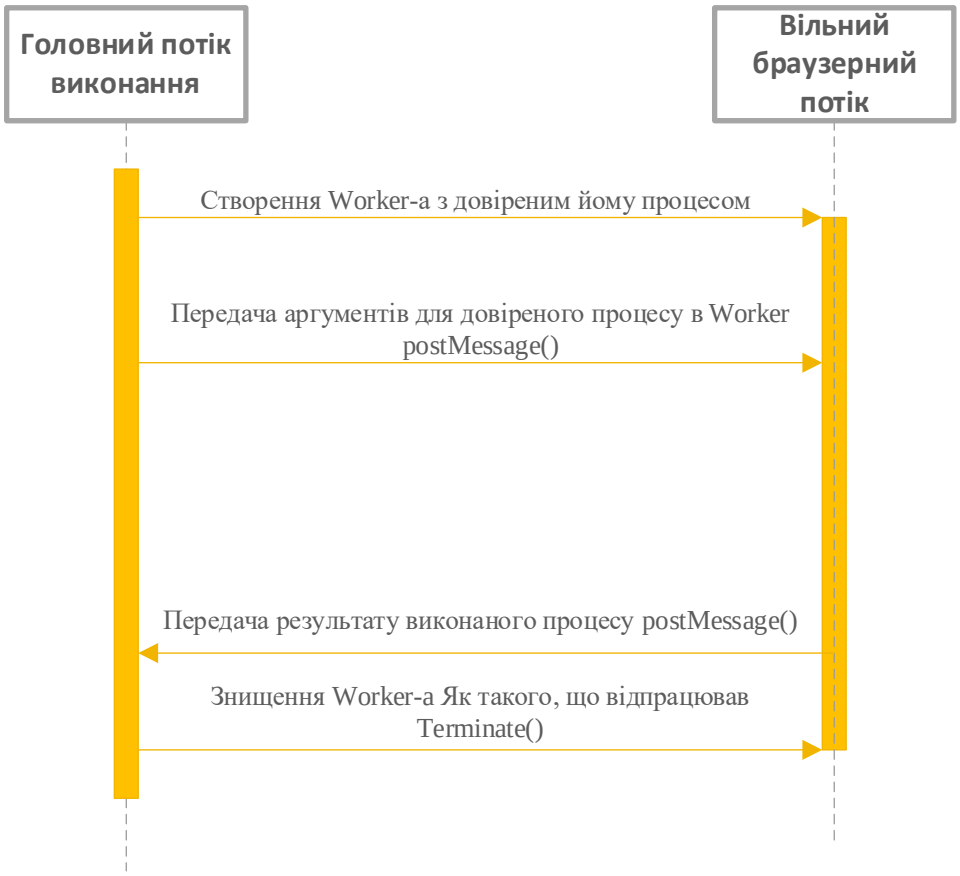

Рисунок 3.8 – UML діаграмі послідовності

4 ДОСЛІДЖЕННЯ ПЕРЕВАГ ПІДХОДУ

4.1 Оцінка часу планування і розробки при традиційній розробці і з використанням вдосконаленої методології.

Покращення показників і оптимізація процесу, залежить від способу організації структури процесу. Для чіткого виконання правил і високої швидкості впроваджується автоматизація процесів. Це дає прогрес в швидкості і якості. І в наш час це широко застосовується на виробництві, при надані послуг, розвитку науки або навчанні в приватних і державних навчальних закладах.

Цифровізація процесів вимагає певних інтелектуальних і трудових витрат, а також передбачає використання обчислювальної техніки, що визначає особливості розрахунку собівартості програмного продукту.

Оцінка вартості програмування розглядається як оцінка затрат коштів і часу, необхідних для виконання етапів проекту.

Вартість програмного продукту включає:

1. собівартість програмного продукту;
2. плановий прибуток.

Для порівняння цін для розробки багатоплатформного мобільного застосунку, web-сайту, та настільного застосунку в порівнянні з web-застосунком, будуть проведені наступні розрахунки.

4.1.1 Розрахунок часу

Загальний час на створення програми складається з різних компонентів. Структура загального часу на створення програмного продукту представлена в таблиці 4.1

Таблиця 4.1 – Структура загального часу на створення програмного продукту.

№ етапу	Позначення часу, для конкретного етапу	Зміст етапу
1	$T_{ПО}$	Підготовка опису завдання
2	$T_{О}$	Опис завдання
3	$T_{А}$	Розробка алгоритмів
4	$T_{БС}$	Розробка блок-схем для алгоритмів
5	$T_{ПІ}$	Проектування користувацького інтерфейсу
6	$T_{Н}$	Написання коду програми
7	$T_{ВТ}$	Тестування і відладка програми
8	$T_{Д}$	Оформлення технічної документації

Час розраховується в людино-годинах, причому $T_{ПО}$ та $T_Д$ береться по фактично відпрацьованому часу, а час останніх етапів визначається розрахунковий по умовному числу команд Q.

Умовне число команд Q визначається за формулою

$$Q= q \cdot з, \quad (4.1)$$

де q - коефіцієнт, що враховує умовне число команд залежно від типу завдання. Значення коефіцієнта q вибирається з таблиці 4.2.

Значення коефіцієнта з вибирається з таблиці 4.3.

Таблиця 4.2 – Значення коефіцієнта q.

Тип завдання	Значення коефіцієнта q
Завдання обліку	1400…1500
Завдання оперативного управління	1500…1700
Завдання планування	3000…3500
Багатоваріантні завдання	4500…5000
Комплексні завдання	5000…5500

1. Для web-застосунку приймається коефіцієнт q = 3500.
2. Для традиційної методики розробки приймається:
   2.1 Багатоплатформного мобільного застосунку q = 2500;
   2.2 Для web-сайту q = 3000;
   2.3 Для настільного застосунку q = 2000;

Загальне q = 3000 + 2500 + 2000 = 7500.

Програмні продукти за мірою новизни можуть бути віднесені до однієї з чотирьох груп:

1. група А - розробка принципово нових завдань;
2. група Б - розробка оригінальних програм;
3. група В - розробка програм з використанням типових рішень;
4. група Г - разове типове завдання.

Для даного завдання міра новизни: В

За складністю програмні продукти можуть бути віднесені до однієї з трьох груп:

1. алгоритми оптимізації і моделювання систем;
2. завдання обліку, звітності і статистики;
3. стандартні алгоритми.

Дане завдання може бути віднесено до третьої групи складності високого рівня.

Коефіцієнт **(** з **)** визначається з таблиці 4.3 в залежності від складності і міри новизни.

Таблиця 4.3 – Значення коефіцієнта з.

Мова програмування	Група складності	Міра новизни			
		А	Б	В	Г
Високого рівня	1	1,38	1,26	1,15	0,69
	2	1,30	1,19	1,08	0,65
	3	1,20	1,10	1,00	0,60
Низького рівня	1	1,58	1,45	1,32	0,79
	2	1,49	1,37	1,24	0,74
	3	1,38	1,26	1,15	0,69

Для даного завдання коефіцієнт з = 1,00.

За формулою 4.1 визначається умовне число команд Q.

1. Вдосконалена методологія розробки:

$$Q = 3500 \cdot 1{,}00 = 3500 \text{ люд / годин.}$$

2. Для традиційного рішення:

$$Q = 7500 \cdot 1{,}00 = 7500 \text{ люд / годин.}$$

Визначається час, витрачений на кожен етап створення програмного продукту:

1 $T_{ПО}$ (час на підготовку опису завдання), береться по факту і для даного завдання

$$T_{ПО} = 45 \text{ люд / годин.}$$

2 $T_О$ (час на опис завдання) визначається за формулою

$$T_О = Q \cdot B / (50 \cdot K), \quad (4.2)$$

де В - коефіцієнт обліку змін завдання. Коефіцієнт В залежно від складності завдання і числа змін вибирається в інтервалі від 1,2 до 1,5. Для даного завдання В = 1,3

К - коефіцієнт, що враховує кваліфікацію програміста. Значення коефіцієнта вибирається з таблиці 4.4.

Таблиця 4.4 – Значення коефіцієнта К.

Стаж програміста	Значення коефіцієнта К
до 2-х років	0,8
від 2 до 3 років	1,0
від 3 до 5 років	1,1…1,2
від 5 до 10 років	1,2…1,3
понад 10 років	1,3…1,5

В даному випадку коефіцієнт К = 0,8.

За формулою 4.2 підраховується час на опис завдання.

2.1 Вдосконалена методологія розробки:

$$T_О = 3500 \cdot 1{,}3 / (50 \cdot 0{,}8) = 113{,}75 \text{ [люд. / годин].}$$

2.2 Для традиційного рішення:

$T_O = 7500 \oplus 1{,}3 / (50 \oplus 0{,}8) = 243{,}75$ [люд. / годин].

3 T_A (час на розробку алгоритмів) розраховується за формулою

$$T_A = Q / (50 * K), \quad (4.3)$$

За формулою 4.3 підраховується час на розробку алгоритмів.

3.1 Вдосконалена методологія розробки:

$T_A = 3500 / (50 \cdot 0{,}8) = 87{,}5$ [люд. / годин].

3.2 Для традиційного рішення:

$T_A = 7500 / (50 \cdot 0{,}8) = 187{,}5$ [люд. / годин].

4 $T_{БС}$ (час на розробку блок-схеми) визначається аналогічно ТА за формулою 4.3 і складає

4.1 Вдосконалена методологія розробки:

$T_{БС} = 3500 / (50 \cdot 0{,}8) = 87{,}5$ [люд. / годин].

4.2 Для традиційного рішення:

$T_{БС} = 7500 / (50 \cdot 0{,}8) = 187{,}5$ [люд. / годин].

5 T_H (час написання програми мовою програмування) визначається за формулою

$$T_H = Q \cdot 1{,}5 / (50 \cdot K), \quad (4.4)$$

За формулою 4.4 підраховується час написання програми на мові програмування.

5.1 Вдосконалена методологія розробки:

$T_H = 3500 \cdot 1{,}5 / (50 \cdot 0{,}8) = 131{,}25$ [люд. / годин].

5.2 Для традиційного рішення:

$T_H = 7500 \oplus 1{,}5 / (50 \oplus 0{,}8) = 281{,}25$ [люд. / годин].

6 $T_{ПІ}$ (час на проектування інтерфейсу) визначається за формулою

$$T_{ПІ} = Q / 50, \quad (4.5)$$

За формулою 4.5 підраховується час на проектування інтерфейсу програми.

6.1 Вдосконалена методологія розробки:

$T_{ПІ} = 3500 / 50 = 70$ [люд. / годин].

6.2 Для традиційного рішення:

$T_{ПІ} = 7500 / 50 = 150$ [люд. / годин].

7 $T_{ВТ}$ (час відладки і тестування програми) визначається за формулою

$$T_{ВТ} = Q \cdot 4{,}2 / (50 \cdot K), \quad (4.6)$$

За формулою 4.6 підраховується час відладки і тестування програми.

7.1 Вдосконалена методологія розробки:

$T_{ВТ} = 3500 \cdot 4{,}2 / (50 \cdot 0{,}8) = 367{,}5$ [люд. / годин].

7.2 Для традиційного рішення:

$T_{ВТ} = 7500 \oplus 4{,}2 / (50 \oplus 0{,}8) = 787{,}5$ [люд. / годин].

8 $T_Д$ (час на оформлення документації), береться по факту і для даного завдання

8.1 Вдосконалена методологія розробки:

$T_Д = 70$ [люд / годин].

8.2 Для традиційного рішення:

$T_Д = 150$ [люд / годин].

9 Підраховується загальний час на створення програмного продукту за формулою:

$$T = T_{ПО} + T_{О} + T_{А} + T_{БС} + T_{Н} + T_{ПІ} + T_{ВТ} + T_{Д}, \quad (4.7)$$

9.1 Вдосконалена методологія розробки

$$T_{вмз} = 45 + 113,75 + 87,5 + 87,5 + 131,25 + 70 + 386,5 + 70 = 991,5$$

[люд. / годин].

9.2 Для традиційного рішення:

$$T_{тр} = 45 + 243,75 + 187,5 + 187,5 + 281,25 + 150 + 787,5 + 150 = 2032,5$$

[люд. / годин].

10 $О_{ЧВ}$ (Оптимізація часу в відсотках) Різниця в часі при розробці трандиційним методом і з використанням вдосконаленої методології.

$$О_{ЧВ} = 100 – (T_{вмз} / (T_{тр} / 100)), \quad (4.8)$$

$$О_{ЧВ} = 100 – (911,5 / (2032,5 / 100)) = 55,1 [\%].$$

4.1.2 Розрахунок заробітної плати виконавця робіт із створення програмного продукту

Основна ЗП визначається за формулою:

$$ЗПосн.=(ЗП\ 1р \cdot Кт \cdot Т)/(Чр \cdot tр.д.)\cdot(1 + П/100), \quad (4.9)$$

де ЗП 1р - місячна зарплата 1-го розряду, грн.;

Кт - тарифний коефіцієнт, відповідний розряду тарифної сітки, за яким працює виконавець;

Т - загальний час на створення програмного продукту, люд. / годин;

Чр - кількість робочих днів в місяць;

tр.д. - тривалість робочого дня в годинах;

П - відсоток премії.

Для даного завдання:

1. Заробітна плата 1 розряду = 2670,00 грн станом на 2021;
2. виконавець має 7 розряд;
3. для 7 розряду тарифний коефіцієнт Кт = 1.54;
4. загальний час створення програмного продукту Твмз = 991,5 [люд. / годин], Ттр = 2032,5 [люд. / годин]
5. кількість робочих днів Чр = 20;
6. тривалість робочого дня tр.д. = 8 годин;
7. відсоток премії П=15%.

1. Таким чином, визначаємо основну заробітну плату виконавця робіт із створення програмного продукту.

1.1 Вдосконалена методика розробки:

$$ЗП\ осн. = (2670 \cdot 1,54 \cdot 991.5) / (20 \cdot 8) \cdot (1 + 15 / 100) = 29302,30 [грн].$$

1.2 Для традиційного рішення:

ЗП осн. = (2670 · 1,54 · 2032,5) / (20 · 8) · (1 + 15 / 100) = 60067,60 [грн].

2. Додаткова заробітна плата береться у розмірі 15 % від основної і розраховується за формулою:

$$ЗП\ дод. = ЗПосн. \cdot 15 / 100, \quad (4.10)$$

2.1 Вдосконалена методика розробки:

ЗП дод. = 16913,30 · 15 / 100 = 4395 [грн].

2.2 Для традиційного рішення:

ЗП дод. = 60067,60 · 15 / 100 = 9010,10 [грн].

3. Загальна заробітна плата розраховується за формулою:

$$ЗПзагальна = ЗПосн + ЗПдод., \quad (4.11)$$

3.1 Вдосконалена методика розробки

$ЗПзагальна_{вмз}$ = 29302,30 + 4395= 33697,30 [грн].

3.2 Для традиційного рішення:

$ЗПзагальна_{тр}$ = 60067,60 + 9010,10 = 69077,70 [грн].

4.1.3 Розрахунок нарахувань на заробітну плату (єдиного соціального внеску)

Єдиний соціальний внесок нараховується на заробітну плату за формулою 4.12 і складає 22 % від загальної заробітної плати

$$ЄСВ = ЗПзагальна \cdot 22 / 100, \quad (4.12)$$

1. Вдосконалена методика розробки

ЄСВ = 33697,30 · 22 / 100 = 7413.40 [грн].

2. Для традиційного рішення:

ЄСВ = 69077,70 ⊕ 22 / 100 = 15197.10 [грн].

4.1.4 Розрахунок витрат на збереження та експлуатацію ПЕОМ, що відносяться до даного програмного продукту

Витрати на збереження та експлуатацію ПЕОМ визначаються у розмірі 130% від основної заробітної плати працівників, що забезпечують функціонування ПЕОМ.

До таких витрат відносяться витрати на:

1. основну заробітну плату працівників, що забезпечують функціонування ПЕОМ (інженер-електронник; системний програміст; оператор);
2. основна ЗП адміністративного і допоміжного персоналу;
3. амортизаційні відрахування;
4. витрати на електричну енергію;

5. витрати на матеріали (до їх числа входять диски, картриджі і папір для принтерів та інше);
6. витрати на профілактику ПЕОМ з периферією;
7. витрати на опалювання виробничих площ;
8. витрати на обслуговування виробничих площ;
9. інші виробничі витрати.

Розрахунок витрат на збереження та експлуатацію ПЕОМ, що відносяться до даного програмного продукту за формулою

$$В\ зб.\ експл. = ЗП\ осн. \cdot 130 / 100, \quad (4.13)$$

1. В досконалена методика розробки

$$В\ зб.\ експл. = 33697{,}30 \cdot 130 / 100 = 43806{,}50\ [грн].$$

2. Для традиційного рішення:

$$В\ зб.\ експл. = 69077{,}70 \oplus 130 / 100 = 89\ 801{,}01\ [грн].$$

4.2 Розрахунок собівартості при традиційній розробці і з використанням вдосконаленої методології.

У собівартість програмного продукту входять наступні елементи:

1. основна заробітна плата виконавця робіт із створення програмного продукту;
2. додаткова заробітна плата виконавця робіт із створення програмного продукту;
3. нарахування на заробітну плату (єдиний соціальний внесок);
4. витрати на збереження та експлуатацію ПЕОМ, що відносяться до даного програмного продукту;
5. інші витрати.

Інші витрати складають 10% від суми перших чотирьох елементів і розраховуються за формулою

$$І.вит.=(ЗП\ осн. + ЗП\ дод. + ЄСВ + В\ зб.\ експл.) \cdot 10 / 100 \quad (4.14)$$

1. В досконалена методика розробки

$$І.вит_{вмр}= (29302{,}30 + 4395 + 7413 + 43806{,}50) \oplus 10 / 100 = 8481{,}68\ [грн].$$

2. Для традиційного рішення:

$$І.вит_{тр}= (60067{,}69 + 9010{,}10 + 15197.10 + 89801) \oplus 10 / 100 = 17407{,}60\ [грн].$$

Визначається собівартість програмного продукту за формулою

$$Сп.п. = ЗП\ осн. + ЗП\ дод. + ЄСВ + В\ зб.експл. + І.вит. \quad (4.15)$$

1. В досконалена методика розробки

$$Сп.п._{вмт} = 29302{,}30 + 4395 + 7413 + 43806{,}50 + 8481{,}68 = 93398{,}50\ [грн].$$

2. Для традиційного рішення:

$Сп.п._{тр}$ = 60067,69 + 9010,10 + 15197.10 + 89801 + 17407,60 = 191483.49[грн].

4.3 Розрахунок ціни програмного продукту при традиційній розробці і з використанням вдосконаленої методології

Ціна програмного продукту складається з декількох компонентів і розраховуються за формулою

$$Ц = Сп.п. + П + ПДВ, \quad (4.16)$$

де Сп.п.- собівартість програмного продукту, грн.;

П - плановий прибуток, грн.;

ПДВ - податок на додану вартість, грн.

П – прибуток складає 40% від повної собівартості програмного продукту і розраховуються за формулою

$$П = Сп.п. \cdot 40 / 100 \quad (4.17)$$

1. В

досконалена методика розробки

$$П = 93398,50 \cdot 40 / 100 = 37359 \text{ [грн]}.$$

2. Для традиційного рішення:

$$П = 191483.49 \oplus 40 / 100 = 76593,40 \text{ [грн]}.$$

ПДВ - податок на додану вартість, який береться у розмірі 20% від суми собівартості і прибутку розраховуються за формулою

$$ПДВ = (Сп.п. + П) \cdot 20 / 100, \quad (4.18)$$

1. Вдосконалена методика розробки

$$ПДВ = (93398,50 + 37359191483.49) \cdot 20 /100 = 26\ 151.5 \text{ [грн]}.$$

2. Для традиційного рішення:

$$ПДВ = (191483.49 + 76593,40) \oplus 20 /100 = 53616.30 \text{ [грн]}.$$

За формулою 3.15 визначається ціна програмного продукту.

1. Вдосконалена методика розробки

$$Ц_{вмр} = 93398,50 + 37359 + 26\ 151.5 = 156\ 908 \text{ [грн]}.$$

2. Для традиційного рішення:

$$Ц_{тр} = 191483.49 + 76593,40 + 53616.30 = 321\ 692 \text{[грн]}.$$

$О_{КВ}$ (Оптимізація коштів в відсотках) Різниця в часі при розробці традиційним методом і з використанням вдосконаленої методології.

$$О_{КВ} = 100 – (Ц_{вмр} / (Ц_{тр} / 100)) , \quad (4.19)$$

$$О_{КВ} = 100 – (156\ 908 / (321\ 692 / 100)) = 51,2 \text{ [%]}.$$

4.4 Зведена таблиця показників

Результати розрахунків відображаються в підсумкової таблиці 4.5.

Таблиця 4.5 – Результати розрахунків

Найменування показника	$Сума_{вмр,}$ грн.	$Сума_{тр,}$ грн.
Собівартість програмного продукту	93 398,50	191 483.49
Прибуток	37 359	76 593,40
ПДВ	26 151.5	53 616.30
Ціна програмного продукту	156 908	321 692
Оптимізація грошових ресурсів в відсотках	51,2%	

Підбивши підсумки розрахунків можна бачити, що собівартість програмного рішення за удосконаленою методико розробки становить 93 398,50 грн, при розрахованому прибутку в 40 відсотків, становитиме 37 359, і з урахуванням 20% ПДВ, що становитиме 26 151.5 грн, загальна ціна програмного продукту становитиме 156 908 грн.

Тим часом використавши традиційний підхід розробки, настільного застосунку, багатоплатформного мобільного застосунку, і web-сайту. По собівартості коштує 191 483.49 грн, очікуваний прибуток в 40% становитиме 76 593,40 грн і з урахуванням 20% ПДВ, що становить 53 616.30 грн, загальна сума продукту коштує 321 692 грн.

4.5 Дослідження, порівняння, і аналіз часу розробки

В ході проведення черги розрахунків, пов’язаних з визначенням часу для виконання поставленого технічного завдання багатоплатформного програмного рішення, було окремо розраховано час виконання завдання виконуючи традиційний підхід:

1. Розробка настільного застосунку;
2. Розробка багатоплатформного мобільного застосунку для IOS/Android;
3. Розробка web-сайту.

А також запропоновану методику розробку, кінцевим результатом якої створюється гібридний web-застосунок, що покриває всі необхідній платформи. Працюючи і як web-сайт, і як мобільний застосунок, і як настільний застосунок.

В процесі розрахунків було визначено, що для схожих за розміром програм(кількість виконаних команд) проходячи всі етапи розробки цього програмного рішення, а саме:

1. Підготовка опису завдання;
2. Опис завдання;
3. Розробка алгоритмів;
4. Розробка блок-схем для алгоритмів;
5. Проектування користувацького інтерфейсу;
6. Написання коду програми;
7. Оформлення технічної документації.

Для розробки традиційною методикою, було витрачено 991,5 людино-годин, а застосовуючи вдосконалену методику 2032,5 людино-годин. Таким чином оптимізація часу на проходження всіх етапів розробки скорочується на 55.1%. Що є значним показником, так як скорочення часу більше ніж в половину може бути вирішальним, для замовника програмного продукту, наприклад, у випадку виходу на ринок з стартапом. Тому перевага в часі тут суттєва і очевидна.

4.6 Дослідження, порівняння і аналіз вартості

В ході проведення черги розрахунків стосовно ціни виконання поставленого технічного завдання багатоплатформного програмного рішення, було окремо розрахована собівартість програмного рішення, прибутку, та сплати податків.

При розробці використовуючи виконуючи традиційний підхід:

1. Розробка настільного застосунку;
2. Розробка багатоплатформного мобільного застосунку для IOS/Android;
3. Розробка web-сайту.

А також запропоновану покращену методику розробку, кінцевим результатом якої створюється гібридний web-застосунок, що покриває всі необхідні платформи. Працюючи на всіх необхідних платформах перерахованих вище.

Розрахунки велись з врахуванням, що розробники, які виконують проект за різними методиками підібрані з однаковим стажем роботи. Таким чином отримуємо показники, структуровані в з таблиці 4.6:

Таблиця 4.6 – Показники розрахунків вартості.

Показники	Вдосконалена методика розробки	Традиційна методика розробки
Витрати на заробітну плату працівнику/працівникам	29 302,30 грн	60 067,60 грн
Витрати на премії працівнику/працівникам 15%	4 395 грн	9 010,10 грн
Податок ЄСВ 22%	7 413,40 грн	15 197,10 грн
Витрати на збереження та експлуатацію	43 806,50 грн	89 801,01 грн
Прибуток 40%	37 359 грн	76 593,40 грн
ПДВ 20%	26 151,5 грн	53 616,30 грн
Підсумкова вартість	156 908 грн	321 692 грн
Оптимізація витрат	51.2%	

Проаналізувавши витрати, можна зробити висновок, що при застосуванні вдосконаленої методики розробки спостерігається зниження вартості розробки програмного рішення, трохи більше ніж в половину, на 51.2%, що буде вирішальним для замовника. За оптимізовані кошти можна розробити, ще один проект з схожою складністю розробки і проектування. Тому перевага вдосконаленої методики розробки тут очевидна.

А якщо врахувати витрати на подальшу підтримку проекту, тенденція розриву ціни буде збільшуватись, в користь розробленого web-застосунку. Це відбуватиметься за рахунок того, що обслуговувати доведеться всього одну програму, а не три на різних платформах. Web-застосунок використовує, одну й ту саму мову програмування, тому штат працівників буде меншим, що є також оптимізацією витрат, при експлуатації програмного продукту.

4.7 Дослідження і аналіз і порівняння ключових характеристик

В процесі додаткових дослідження, було виявлено, що побудова web-застосунку використовуючи удосконалений метод web-розробки, можна зменшити вагу застосунку, в рази. Якщо взяти середню вагу сайту, це біля 2 мегабайт, і в порівняння з середньою вагою застосунку в App Store, ця вага складає 23 мегабайти. А це значить, в 10 разів web-застосунок легший, і це не максимум, існують випадки коли цей показник сягає десятків разів.

Додаткові дослідження показують, що двигун JS погано справляється з об'ємними обрахунками. Двигун JS є одно поточним, і коли починається виконання «важкої» функції, він блокує потік, а з ним і весь застосунок чекає поки цей важкий код виконається, в цей момент застосунок не реагує ні на кліки, зупиняються анімації. Для виходу з цієї незручної ситуації, важкий код розбивають на багато легших операцій, і виконують їх асинхронно, в порядку черги, і цей варіант, не дає заблокувати потік. Є й інша сторона, код стає важчим, та більш об'ємним, тому більш практичним рішенням буде використовувати web worker`s. Браузер, його внутрішнє Web API містить це налаштування під назвою web worker над двигуном JS, і дає можливість створити паралельні потоки. Що є чудовим рішенням для побудови web-застосунку.

При застосуванні удосконаленого методу web-розробки, використавши утиліту Lighthouse в Google DevTools, можна отримати звіти по сторінці, і вони демонструють, що сторінка починає краще індексуватись пошуковим роботом, що стовідсотково впливає на відвідуваність цього ресурсу.

Наступна перевага випливає з попередньої, за рахунок того, що web-застосунок в рази легше, швидкість завантажування застосунку підвищується в рази, що позитивно впливає на кількість завантажень.

За рахунок того що, web-застосунок завантажується напряму як web-застосунок, він не потребує, майданчиків розповсюдження застосунків як Apple Store, Play Market та інше. А це значить, що за розміщення не майданчику не потрібно витрачати кошти. І web-застосунок не буде модеруватись політикою цього майданчика розповсюдження застосунків. Так як ці майданчики належать

різним країнам, з різними правовими нормами. І тому дуже просто порушити одну з політик цих майданчиків, і застосунок буде обмежений або заблокований.

Використання web-застосунку знімає ці обмеження, що є ще одною перевагою.

Проведені вище аналізи та дослідження, показують, що дана методика, має безліч переваг, над традиційною розробкою сайту, і окремо застосунків під кожну платформу.

ВИСНОВКИ

В ході написання магістерської дипломної роботи, було проаналізовано проблематику розробки сучасних програмних рішень, виявлено недоліки з якими стикаються замовники і програмні інженери. Для вирішення цих недоліків, була запропонована вдосконалена методика web-розробки, що базується на застосуванні технологій PWA та багатопотоковості.

В першому розділі дипломної роботи було детально описано механізм роботи і головну ідею застосування вдосконаленої методики web-розробки і побудови багатоплатформних рішень. Проблеми які існують при web-розробці, і шляхи їх вирішення.

В другому розділі була виставлена умовна задача, яка може бути розробленим традиційним методом web-розробки, або ж удосконаленим методом web-розробки. Ця задача, це застосунок «Щоденний помічник». Потрібно реалізувати web-версію, мобільну версію, і настільну версію. В цьому розділі записані технічні вимоги до застосунку.

В результаті було розроблене програмне рішення, застосовуючи вдосконалену методику web-розробки, що дало можливість в подальшому провести аналіз переваг підходу. І дослідити чи задовольняє поставлені вимоги, вдосконалений метод розробки.

В третьому розділі було детально описано застосування вдосконаленої методики web-розробки на практиці. З застосуванням теорії з першого розділу вже на конкретному прикладі розробки.

Четвертий розділ підсумковий, в ньому проводиться аналітична робота і дослідження, кількісних переваг вдосконаленого методу web-розробки, над традиційним методом.

Було доведено вищу ефективність розробки, використовуючи вдосконалений метод web-розробки для побудови багатоплатформних програмних рішень. Переваги виявились очевидні, час розробки скорочується майже в половину, вартість розробки скорочується трошки більше ніж в половину. Вага застосунку зменшується в рази, паралельно зростає швидкість завантаження. Що позитивно впливає на кількість завантажень. Покращується індексація браузерами. Застосунок позбувається небажаних залежностей, а його подальша підтримка в експлуатації стає простішою та дешевшою.

Вдосконалення методики web-розробки є інноваційним рішенням, що дає велику кількість переваг над традиційними методами web-розробки і побудови багатоплатформних рішень. Тому зможе бути дуже актуальним для бізнесу.

Коли мова йде про невеликий бізнес, або новий стартап, підприємець виходить на ринок з новим продуктом. Вартість і швидкість розробки, стають ключовими вимогами. Застосовуючи вдосконалену методику web-розробки для побудови багатоплатформних рішень, значно оптимізує ці витрати.

З вище зазначеного випливає висновок, що вдосконалена методика web-розробки на базі багатопотоковості і PWA є високоефективною і має великий потенціал застосування.