Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження архітектури та методів побудови PWA додатків на базі JavaScript
(тема)

Виконав:
Випускник 2 курсу, групи ІПЗм-19-2
Долгих К. І.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми Освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Керівник проф. Машталір В. П.
(посада, прізвище)

Допускається до захисту
Зав. кафедри _____ З.В. Дудар
(підпис) (прізвище, ініціали)

2021 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Програмної інженерії
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва)

Тип програми освітньо-наукова програма
(освітньо-професійна або освітньо-наукова)

Освітня програма Інженерія програмного забезпечення
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)
« 26 » березня 2021 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студента Долгих Костянтина Ігоровича
(прізвище, ім'я, по батькові)

1. Тема роботи «Дослідження архітектури та методів побудови PWA додатків на базі JavaScript»

затверджена наказом університету від 26.03.2021 № 385 Ст

2. Термін подання студентом роботи до екзаменаційної комісії «08» травня 2021 р.

3. Вихідні дані до роботи *методи побудови PWA додатків на JavaScript*

4. Перелік питань, що потрібно опрацювати в роботі *мета роботи, аналіз предметної галузі і постановка задачі, методи побудови PWA додатків на JavaScript, особливості керування станом та робота додатків без доступу до мережі*

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, ілюстрацій (слайдів) *мета завдання, обґрунтування доцільності розроблення, постановка задачі, методи і архітектурні підходи, структурно-логічна схема взаємодії даних, опис отриманих результатів, демонстраційні матеріали*

6 Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Спецчастина	Машталір В. П.		

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Аналіз предметної галузі	25.01.21 – 19.02.21	*Виконано*
2	Огляд існуючих методів та архітектур	20.02.21 – 10.03.21	*Виконано*
3	Огляд гібридних додатків	11.03.21 – 25.03.21	*Виконано*
4	Підготовка пояснювальної записки	26.03.21 – 15.04.21	*Виконано*
5	Спецчастина	15.04.21 – 17.04.21	*Виконано*
6	Підготовка презентації та доповіді	18.04.21 – 19.04.21	*Виконано*
7	Нормоконтроль, рецензування	12.05.21 – 13.05.21	*Виконано*
8	Занесення диплома в електронний архів	14.05.21	*Виконано*
9	Попередній захист	15.05.21	*Виконано*
10	Допуск до захисту у зав. кафедри	16.05.21	*Виконано*

Дата видачі завдання 25 січня 2021 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Машталір В.П.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ ABSTRACT

Кваліфікаційна робота магістра містить: 93 с., 29 рис., 12 джерел.

PWA, JAVASCRIPT, INDEXEDDB, TYPESCRIPT, REACT, АРХІТЕКТУРА, PUSH API, SERVICE WORKER, NOTIFICATION API, BROWSER, TYPESCRIPT.

Об’єктами дослідження є бібліотеки та фреймворки, які дозволяють будувати PWA додатки на базі JavaScript.

Метою роботи є дослідження архітектури та методів побудови PWA додатків на базі JavaScript та огляд існуючих пакетів і фреймворків, які дозволяють та спрощують розробку.

Результатом роботи є рекомендації щодо ефективного використання засобів та бібліотек JavaScript при побудові PWA додатків та програмна реалізація цих методів та підходів, яка може бути вбудована у будь-які PWA додатки для спрощення їх розробки.

PWA, JAVASCRIPT, INDEXEDDB, TYPESCRIPT, REACT, ARCHITECTURE, PUSH API, SERVICE WORKER, NOTIFICATION API, BROWSER, TYPESCRIPT.

The objects of research are libraries and frameworks that allow you to build PWA applications based on JavaScript.

The aim of the work is to study the architecture and methods of building PWA applications based on JavaScript and review existing packages and frameworks that allow and simplify development.

The result is recommendations for the effective use of JavaScript tools and libraries in the construction of PWA applications and software implementation of these methods and approaches, which can be built into any PWA application to simplify their development.

Я, *Долгих Костянтин Ігорович,* студент групи ІПЗм-19-2, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження архітектури та методів побудови PWA додатків на базі JavaScript», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений(а) з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

PWA – Progressive Web Application;

API – Application Programming Interface;

JS – JavaScript;

URL – Uniform Resource Locator;

ІБ – Інформаційна безпека;

ПЗ – Програмне забезпечення;

ПП – Програмний продукт.

ЗМІСТ

ВСТУП

В останні роки зростає кількість розроблюваних веб-сайтів та мобільних додатків, а разом з ними зростає кількість технологій, які швидко і зручно дозволяють розроблювати їх. На сьогодняшній день майже усі компанії намагаються мати свій власний сайт або мобільний додаток, щоб надати їх користувачам найкращий досвід роботи з сервісами компанії. З кожним днем зростають потреби бізнесу, а з ними в свою чергу і вимоги до розробки мобільних та веб додатків, вже не достатньо просто, щоб додатки працювали швидко і без збоїв, а ще на один з перших планів виходять такі аспекти: швидкість розробки, розширюваність, легкість інтеграції із сторонніми сервісами, простота інтеграції нових членів команди та багато інших. Ці потреби на сьогодні є невід'ємни, тому що за їх вісутністю скоріш за все додаток не зможе задовольняти потреби сучасних клієнтів та розробників, що може призвести до занепаду бізнесу. На ряду з таким прогресом в останні роки зросла потужність веб браузерів та мобільних телефонів, це в свою чергу призвело до зростання полулярності мови програмування JavaScript, бо саме ця мова програмування дозволяє легко маніпулювати веб-сторінками, забезпечує взаємодію користувача та сторінки, взаємодіє з API, надає можливості кешування данних на стороні клієнта та багато інших можливостей, саме ця мова розробки робить додатки більш живими та затребуваними.

JavaScript попри свої успіхи у браузерному середовищі не зупинився у розвитку і на даний момент є досить універсальним інструментом, що дозволяє писати код, який буде виконуватися на сервері, будувати мобільні та десктопні додатки, а також надає дуже велику кількість різноманітних бібліотек та фреймворків. У сучасномі світі зросла популярність використання мобільних пристроїв, популярність дійшла до того, що зараз люди витрачають удвічі більше часу на мобільних пристроях, ніж на ПК, а у деяких країнах телефони є єдиним пристроєм. З таким бумом користувачі вимагають незмінно чудового досвіду, в свою чергу це призводить до того, що розробники повинні мати можливість легко

підтримувати та розробляти програми для декількох ОС і дуже часто трапляється так, що користувачі отримують гарний досвід та емоції від використання сайту, але залишаються не в захваті від схожого мобільного додатку, або навпаки. В таких ситуаціях на допомогу прийшли PWA - progressive web applications (прогресивні веб додатки).

PWA - інструмент, а точніш підхід до розробки, який на сьогодняшній день надає такі переваги, як кросплатформеність, миттеві оновлення, підрімка роботи додатку в режимі без доступу до мережі інтернет, тобто офлайн, розротання та деплоймент без особих зусиль, у порівнянні з тим, як цей процес організований у звичайних нативних додатках. А також є такі переваги, як пуш-повідомлення, доступ до даних геолокації та жестів, доступ до камер та різноманітних мобільних датчиків, Bluetooth та інші особливості пристроїв.

Метою роботи є дослідження архітектури та методів побудови PWA додатків на базі JavaScript та огляд існуючих пакетів і фреймворків, які дозволяють та спрощують розробку.

Об’єктами дослідження є бібліотеки та фреймворки, які дозволяють будувати PWA додатки на базі JavaScript.

Предметом дослідження є методи та відомі архітектурні підходи до побудови PWA додатків, а також побудування та виявлення найкращих практик, які задовільняють сучасним стандартам розробки.

Практичним занченням можна вважати результати роботи різних підхідів та методів проектування PWA додатків.

В результаті дослідження формування практичних рекомендацій до розробки PWA додатків з використанням сучасних фреймворків та підходів на базі JavaScript.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз предметної галузі

PWA - веб-додатки, які користуються можливостями сучасних веб-браузерів та їх функціями, використовують нові API-інтерфейси та стратегії прогресивних вдосконалень. Такі прогресивні додатки на сьогодняшній день стали корисним шаблоном дизайну, хоча ще не встигли стати формалізованим стандартом. PWA в деяких моментах можна вважати подібним до AJAX або іншими подібними шаблонами, що охоплюють набір атрибутів програми, включаючи використання певних веб-технологій та методів. Наприклад, веб-програми можна виявити більше, ніж власні програми, набагато простіше і швидше відвідати веб-сайт, ніж встановити програму, і користувачі також можуть ділитися веб-програмами, надсилаючи посилання [1].

З іншого боку, власні програми краще інтегровані з операційною системою і, отже, пропонують користувачам безперебійну роботу. Можна встановити рідну програму, щоб вона працювала в автономному режимі, і користувачі люблять натискати на свої піктограми, щоб легко отримати доступ до своїх улюблених програм, а не переходити до неї за допомогою браузера.

PWA дають нам можливість створювати веб-програми, які можуть користуватися тими ж перевагами.

PWA не створюються з використання однієї технології, на даний момент вони представляють нову філософію створення веб-програм, що включають в себе специфічні шаблони. Не завжди кінцевий користувач зможе зрозуміти з першого погляду та після першого досвіду користування, що веб-програма використовує підходи та принципи, які властиві прогресивним веб-додаткам, а саме:

— працює в автономному режимі;

— встановлюється у якості додатку на пристрій;

— легко синхронізується;

— має можливість надсилати push-повідомлення.

Для того, щоб підвищувати якість виконання прогресивних веб-додатків існує ряд інструментів, одним із яких являється Google Lighthouse. Цей інструмент після ряду тестів створює звіт про те наскільки добре розреблена веб-сторінка, саме провалені тести дозволяють виявити слабкі місця, яким потрібно приділити більше часу, або виявити пункти, про які команда взагалі не здогадувалася. Отже були виявлені ключові принципи, яких повинна дотримуватися веб-програма, щоб визначатися саме як PWA [2]:

— виявляємість, тобто веб-програма може бути знайдена через пошукові системи;

— встанавлюємість, тобто веб-програма може бути доступна на головному екрані пристрою;

— посилання, тобто користувачі легко можуть ділитися додатком;

— незалежність від мережі, тобто має змогу працювати в автономному режимі;

— прогресивність, тобто може використовуватися в більш старих версіях браузерів;

— повторне залучення користувачів, тобто веб-програма здатна відправляти push-повідомлення;

— адаптивність, тобто можна використовувати на будь-якому пристрою з браузерос та екраном - планшети, мобільні телефони, телевізори, холодильники, ноутбуки та інші;

— безпечність, тобто зв’язок між користувачем, сервером та програмою є захищеними від третіх осіб та від доступу до приватної інформації.

Виконуючі усі ці вимоги, веб-програма може надавати найкращій та гнучкий користувацький досвід.

Базуючись на матеріалі, який був викладений вище можна виділити переваги таких PWA додатків:

— можливість оновлення конкретного змінюваного вмісту з оновленням програми, коли у звичаному нативному додатку навіть найменьші зміни можуть

призвести до того, що користувач буде змушений завантажувати увесь додаток з самого початку;

— зовнішній вигляд, який є інтегрованим з власною платформою, характерні значки та іконки на головному екрані або на панелі запуску програм;

— зменшення часу завантаження після встановлення завдяки кешування;

— присутність системних сповіщенн та push-повідомлень, які легше впроваджувати у поривнянні з розробкою під нативні пристрої і які в свою чергу дають кращі коефіцієнти конверсії та зацікавлених користувачів.

1.2 Нативні додатки

Нативні додатки залежать від платформи (Android, iOS, Windows), побудовані з використанням конкретної мови програмування, SDK. Розробники повинні дотримуватися Java для Android, C # для windows і swift для iOS. Власні програми мають повний доступ і користуються усіма перевагами пристрою такі функції, як доступ до акселерометра, камери, компаса, GPS, включають жести та API, завдяки чому забезпечують чудовий інтерфейс, UX, продуктивність та надійність. Нативна програма встановлюється на пристрій користувача через певні магазини програм (магазин програм Apple або Google play магазин) [3].

Власні програми залежать від платформи, тому не можуть бути розгорнуті на декількох платформах.

Вартість висока, а час розробки довший. Користувач повинен встановити програму через відповідний магазин на пристрої, щоб використовувати таким чином встановлені недоліки, якщо користувач хоче використовувати програму лише один раз або періодично. Крім того, оновлення рідної програми є нудним.

1.3 Гібридні додатки

Гібридні програми не залежать від платформи, побудовані з використанням веб-технологій HTML5, CSS 3 та JavaScript загорнутий у власний контейнер Cordova. Після того, як програма розроблена, вона може бути розгорнутим на декількох платформах. Гібридні програми мають доступ до більшості функцій пристрою, таких як нативних, але коли справа стосується 3D, плавної анімації, мультитач, графіки, переходу та ігор, їх продуктивність знижується. Однак вартість і час розробки є нижчий за нативний. На пристрої також повинні бути встановлені гібридні програми, які мають час від часу оновлюватися [4].

1.4 Постановка задачі

— дослідження існуючих засобів та методів побудови PWA додатків з використанням JavaScript та прилягаючих фреймворків і бібліотек;

— аналіз можливих проблем при розробці PWA додатків;

— аналіз існуючих архітектурних рішень при побудові веб-додатків за допомогою Angular;

— аналіз існуючих архітектурних рішень при побудові веб-додатків за допомогою React.js;

— розробка прототипу PWA додатку і порівняння його з нативною реалізацією;

— формування рекомендацій щодо використання методів та засобів побудови PWA додатків.

2 ОПИС ПРОВЕДЕНИХ ТЕОРИТИЧНИХ ДОСЛІДЖЕНЬ

2.1 Конфігурація PWA додатку, manifest файл

Створення прогресивних веб-програм та задоволення всіх вимог на основі продуктивності, доступності, найкращих практик та SEO є досить складним завданням. Щоб це сталося, всі компоненти PWA, тобто Service Worker, Web App Manifest, модель оболонки додатків та повідомлення Web Push, повинні бути реалізовані з великою обережністю. Нижче буде описаний PWA manifest файл.

Web App Manifest - це простий файл JSON, що містить інформацію: name, short_name, description, icons - іконки для девайсів різного розширення, start_url, display, theme - колір теми програми. Використання Web App Manifest встановлює веб-програму на головному екрані користувача між власними програмами. Як результат, користувач може отримати швидкий доступ і насолоджуватися повноекранним екраном, як із рідною програмою [5]. Файл manifest.json для веб-програми зображений на рисунку 2.1.

Як показано на рисунку 2.1, більшість термінів, що використовуються у файлі manifest.json, є зрозумілими на перший погляд: name - це те, що з’являється на екрані заставки, short_name - те, що з’являється на головному екрані, icons використовуються для зображення під різні розширення девайсів. Подібним чином, start_url - це URL-адреса, яка відкривається при натисканні на іконку; display керує режимом відображення, в якому запускається програма. Режим може бути автономним або повноекранним. Автономний режим відкриває програму без користувальницького інтерфейсу браузера, наприклад, панелі розташування.

```
{
  "short_name": "PWA app",
  "name": "PWA News App",
  "icons": [
    {
      "src": "favicon.png",
      "sizes": "192x192",
      "type": "image/png"
    },
    {
      "src": "./splash_images/splash-512.png",
      "sizes": "512x512",
      "type": "image/png"
    }
  ],
  "start_url": ".",
  "display": "standalone",
  "theme_color": "#000000",
  "background_color": "#ffffff",
  "gcm_sender_id": "482941778795"
}
```

Рисунок 2.1 - manifest.json файл

Після конфігурації manifest.json файлу його потрібно прив’язати до index.html файлу, на який будуть ссилатися браузери. Прив’язка manifest.json файлу до index.html здійснюється на малюнку 2.2.

```
<head>
<link rel="manifest" href="/manifest.json">
</head>
```

Рисунок 2.2 - Прив’язка manifest.json файлу до index.html файлу

Для того, щоб веб-програма могла відображатися на веб-сайтах як банер встановлення та забезпечувати подібну поведінку, веб-програма повинна відповідати таким критеріям:

— сайт повинен обслуговуватися через HTTPS;

— на сайті має бути service worker;

— файл маніфесту веб-програми сайту повинен мати принаймні чотири обов’язкові поля - name або short_name(бажано обидва), start_url, icons, display.

Коли вищезазначені критерії виконуються, браузер визначає веб додаток саме як PWA [6].

Файл маніфесту веб-програми можна перевірити вручну за допомогою вкладки «Маніфест» на панелі додатків Chrome DevTools, як показано на малюнку 2.3.

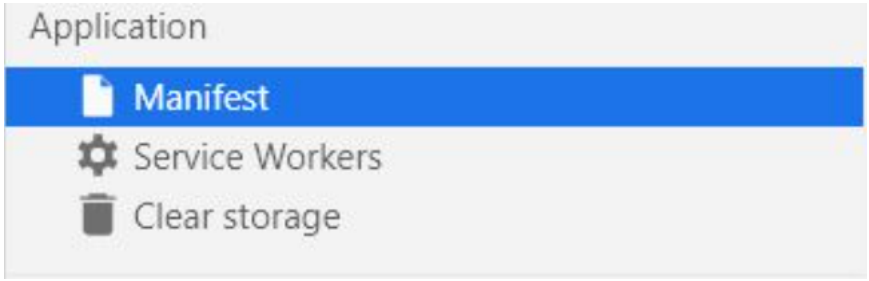

Рисунок 2.3 - manifest.json на панелі додатків Chrome DevTools

Маніфест веб-програми має проблему сумісності з браузерами. Не всі браузери підтримують файл маніфесту. Сумісність маніфесту веб-програми з різним браузерами можна перевірити за допомогою інструменту онлайн-перевірки, як показано на малюнку 2.4.

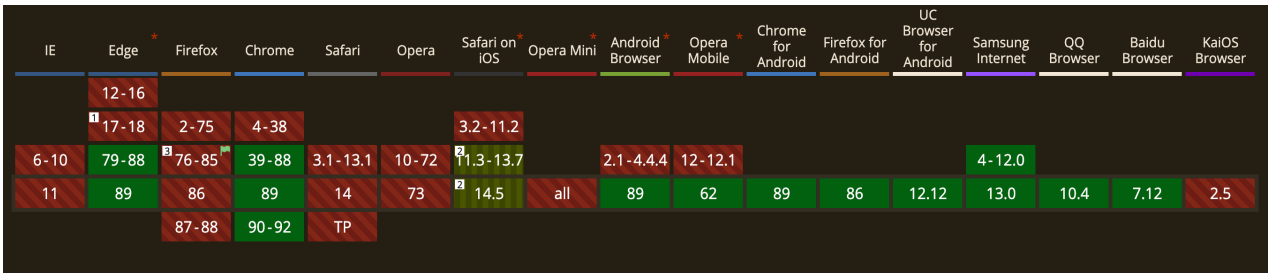

Рисунок 2.4 - Сумісність маніфесту веб-програми з різним браузерами

Як показано на рисунку 2.4, Web App Manifest найкраще працює для Chrome, Chrome для Android, Samsung Internet, Edge, UC браузер для Android, тоді як Firefox та IE досі не підтримують.

2.2 Service Workers у побудові PWA додатків

Сучасний інтернет, тим якім він є сьогодні - багатий, красивий і корисний, повинен надавати користувачам найкращий досвід, у більшості відвідувачей веб-додатків можуть з’явитися проблеми, коли зв’язок з мережею не є досить добрим або взагалі відсутній. Офлайн-стан веб-програми не може надати корисну інформацію для користувачів, але представлення Service Worker перетворило цю помилку на щось, з чим можна впоратися з витонченістю.

Service Worker - це сценарій, керований подіями, який працює у фоновому режимі окремо від веб-сторінки, реагує на події та перехоплює мережеві запити

програми чи веб-сайту за допомогою сервера та ресурсів. Він працює як проксі між мережею та браузером [7]. Він може працювати навіть тоді, коли додаток закрито, тому він служить для ініціювання подій, навіть коли сайт закритий. Такі функції, як push-сповіщення та фонова синхронізація, сьогодні можливі в Інтернеті завдяки Service Workers. Оскільки Service Worker може перехопити запит, змінити вміст або навіть повністю замінити новими відповідями, зареєструвати Service Worker можуть лише сторінки, що обслуговуються через захищені з'єднання (HTTPS). Це для захисту користувачів і запобігання атак. На рисунку 2.5 показана стратегія кешування Service Worker.

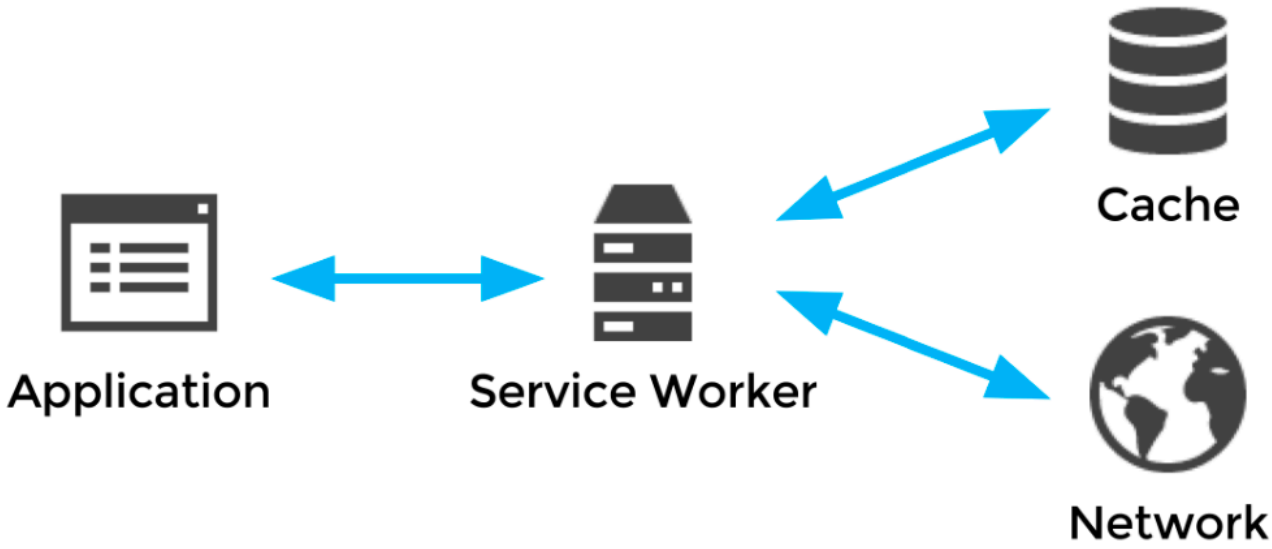

Рисунок 2.5 - Стратегія кешування Service Worker

Як показано на малюнку 2.5, Service Worker знаходиться між додатком та мережею та вирішує, коли йому слід отримати вміст із кешу та коли потрапити в мережу.

Однак програма, в якій запущений Service Worker, може відображати кешований графічний інтерфейс, запускати сценарії та навіть відображати кешований вміст користувача за попереднє відвідування. Добре написаний Service Worker може також реєструвати дані, що надсилаються на сервер, і

відправляти, коли підключення до Інтернету доступне через події синхронізації, та отримувати push-повідомлення від сервера.

Service Worker сприймається як поступове вдосконалення програми або веб-сторінки. Поступове вдосконалення покращує вміст та можливості програми. Таким чином, додаток повинен бути повнофункціональним на будь-якій платформі, перш ніж додавати Service Worker [8]. Кращий досвід може отримати користувач із сумісними браузерами з додатком. На рисунку 2.6 зображена сумісність браузерів для Service Worker.

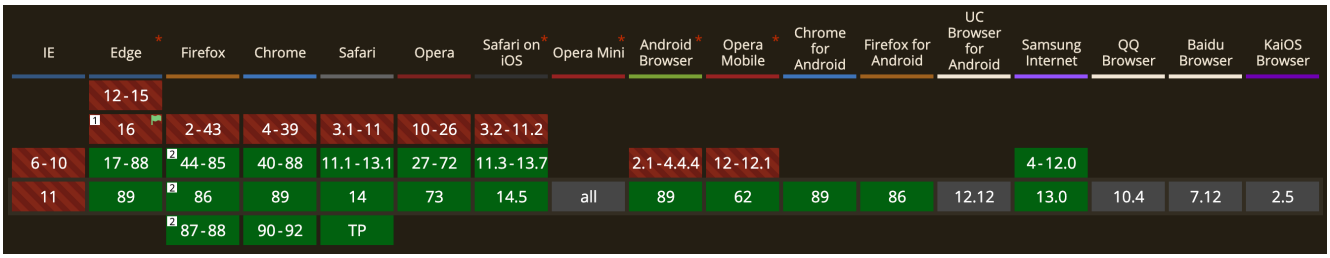

Рисунок 2.6. - Сумісність браузерів для Service Worker

Як показано на рисунку 2.6, більшість браузерів, таких як Chrome, Firefox, Opera, Chrome для Android, Safari, iOS Safari, Edge підтримують Service Worker. Opera Mini та Internet Explorer до сьогодні не підтримують Service Worker.

2.3 Веб пуш-повідомлення

Пуш-повідомлення - одна з ключових складових PWA. Він відсутній в Інтернеті, і він займав центральне місце в прогалині між нативною програмою та веб-програмою.

Пуш-повідомлення - це повідомлення, яке з’являється на пристрої користувача. Воно може бути ініційовано локально відкритою програмою або

коли програма перебуває в режимі очікування. Пуш-повідомлення забезпечує своєчасне оновлення вмісту та даних, які користувач обирає для входу. Це відіграє важливу роль у повторному залученні користувачів та повторному поверненні їх до програм. Пуш-повідомлення складається з: API-повідомлень та Push-API.

API сповіщень дозволяє веб-сторінці або Service Worker створювати та контролювати відображення системних сповіщень. Сповіщення з'являються в інтерфейсі пристрою (поза браузером) і, отже, існують поза контекстом будь-якої вкладки чи вікна браузера. Оскільки вони не залежать від будь-яких вкладок або вікон браузера, їх можна створити навіть після того, як користувач покине сайт. API сповіщень дозволяє відображати повідомлення користувачам. Для відображення сповіщень потрібен дозвіл користувача [9].

Push API дозволяє обробляти повідомлення, які надсилаються клієнту з сервера через push-службу, що використовується браузером. Коли Service Worker реєструється службою push, він видає події push у службі обслуговування. Під час цієї події Service Worker отримує дані, які потрібно відображати в сповіщенні, і використовує Notification API для відображення сповіщення. Активування Service Worker push-повідомлення може призвести до збільшення використання ресурсів, особливо акумулятора. Різні браузери мають різні схеми роботи з цим, в даний час не існує стандартного механізму. Firefox дозволяє надсилати обмежену кількість (квоту) push-повідомлень до програми, хоча Push-повідомлення, які генерують сповіщення, звільняються від цього обмеження. Ліміт оновлюється кожного разу, коли відвідується сайт. Для порівняння, Chrome не застосовує обмежень, але вимагає, щоб кожне push-повідомлення відображало сповіщення. Push API також має проблеми з підтримкою браузера (див. рис. 2.7) .

Як показано на малюнку 2.7, Chrome, Chrome для Android, Firefox, Opera та Edge мають підтримку push-сповіщень, тоді як Safari не підтримує Push API. Однак Safari має власний API для надсилання повідомлень користувачам Safari.

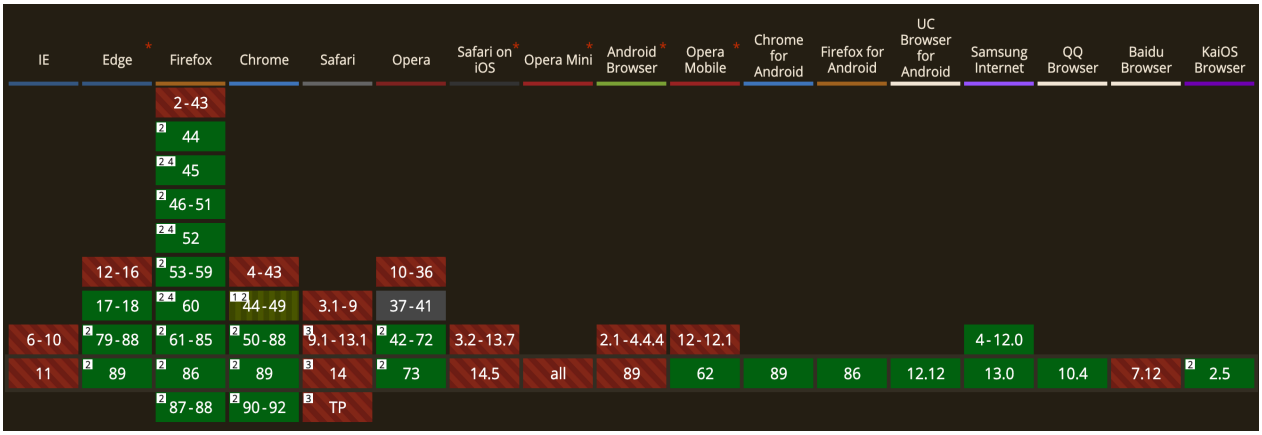

Рисунок 2.7 - Push API підтримка у різних браузерах

Таким чином, у данному розділі було розглянено основні складові сучасних PWA додатків і їх обмеження у сучасних браузерах, а саме було розглянено принципи роботи Service Worker та стратегія їх кешування, були затронуті головні файли конфігурації та сервіси пуш-повідомлень.

2.4 Підтримка роботи з нативними модулями. Android

Платформа Android надає повний доступ до таких нативних можливостей як:

— програвання звуку у фоновому режимі;

— доступ до GPS-кординатів;

— доступ до акселерометра прилада;

— програвання аудіо та відео без взаємодії користувача з пристроєм;

— перехоплення тексту, який вводиться користувачем;

— використання мікровону і камери;

— отримання доступу до бібліотеки фотографій;

— запуск одночасно декількох процесів;

— доступ до пуш-повідомлень;

— доступ до головного екрану прилада;

— доступ до жестів;

— доступ до типу та швидкості мережі;

— доступ до вібрації пристрою;

— доступ до орієнтації екрану;

— доступ до файлової системи;

— доступ до прав доступів;

— доступ до презентаційних;

— отримання доступу до геолокації користувача.

На сьогодняшній день підтримується не для усіх браузерів така функція як web-інтерфейс Bluetooth, таблиця підтримки зображена на рисунку 2.8.

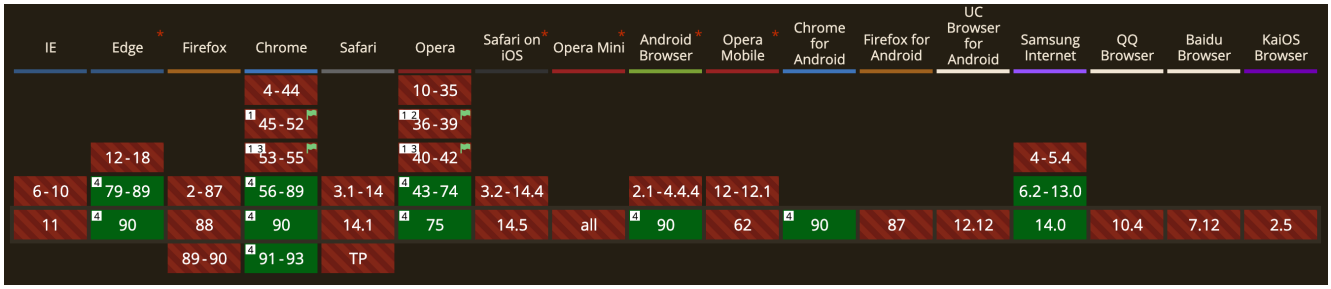

Рисунок 2.8 - Підтримка web-інтерфейсу Bluetooth у різних браузерах

NFC у свою чергу також не підтримується усіма сучасними браузерами, це ми можемо побачити на рисунку 2.9.

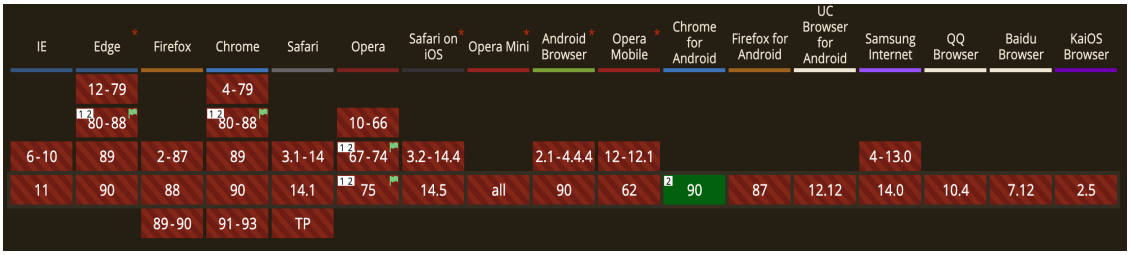

Рисунок 2.9 - Підтримка NFC у різних браузерах

Підтримку роботи детектора оточуючого освітлення можемо побачити на рисунку 2.10.

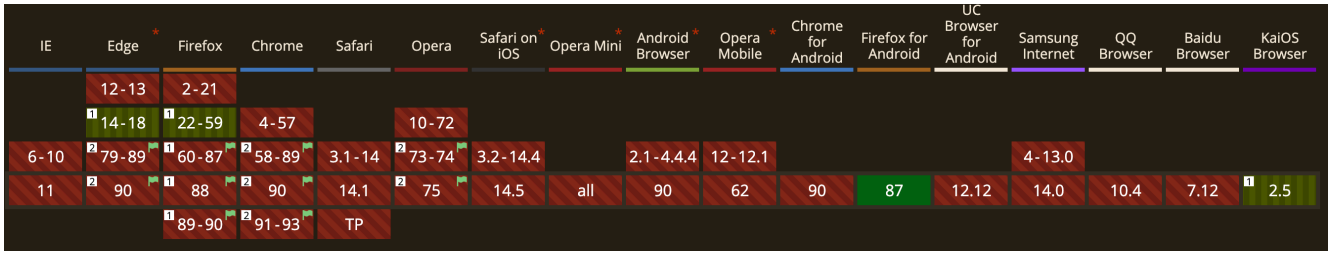

Рисунок 2.10 - Підтримка детектора оточуючого освітлення у різних браузерах

Детектор наближення підтримується браузерами, які зображені на рисунку 2.11.

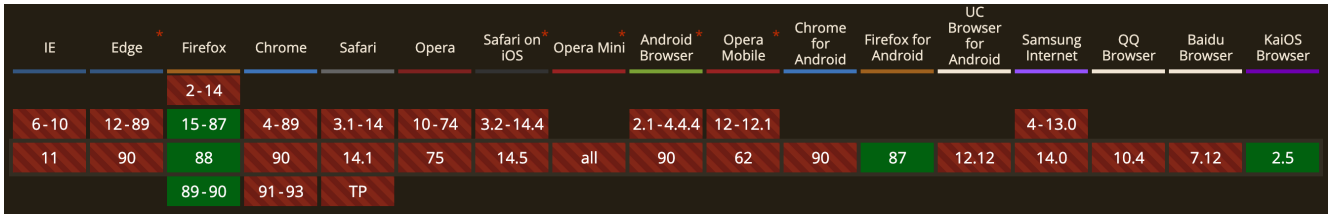

Рисунок 2.11 - Підтримка детектора наближення у різних браузерах

Підтримка акселерометру зображена на рисунку 2.12.

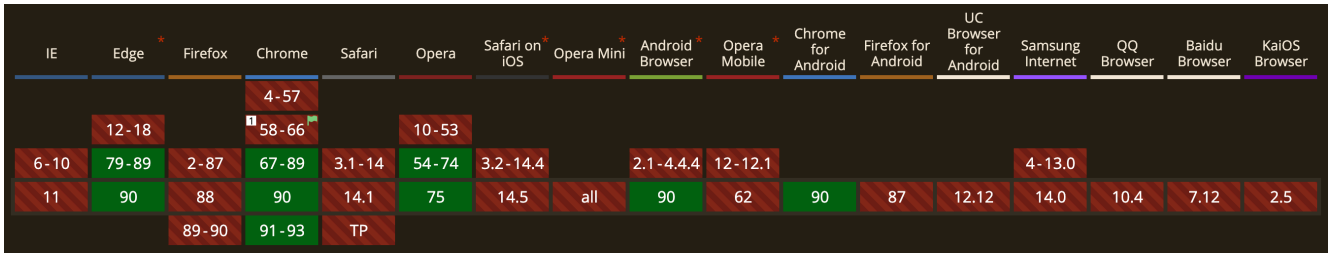

Рисунок 2.12 - Підтримка акселерометру у різних браузерах

Підтримка магнітометру зображена на рисунку 2.13.

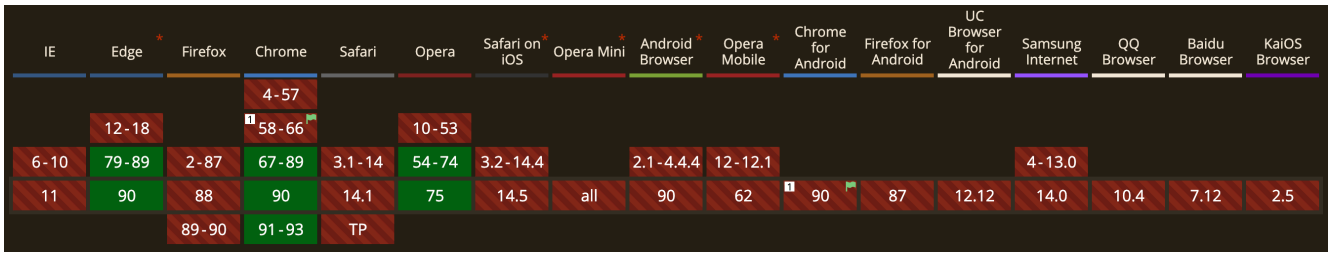

Рисунок 2.13 - Підтримка магнітометру у різних браузерах

Підтримка гіроскопу зображена на рисунку 2.14.

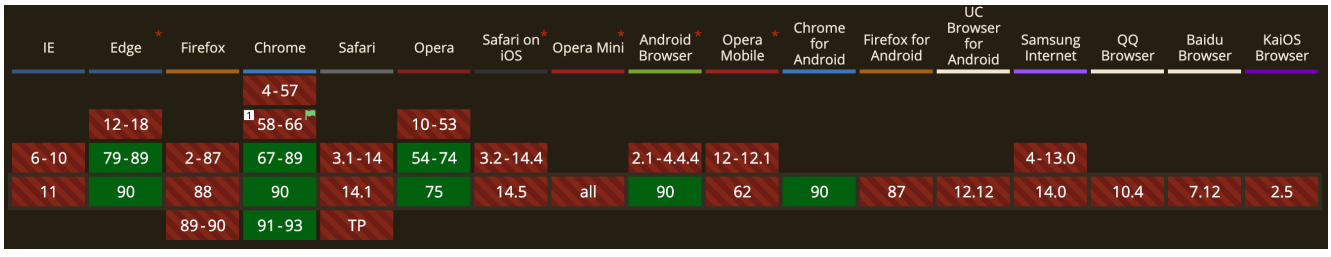

Рисунок 2.14 - Підтримка гіроскопу у різних браузерах

Не зважаючи на те, що на сьогодні сучасні браузери мають підтримку багатьох нативних функцій, все ще залишаються ті, які не підтримуються взагалі, нижче наведено список таких функцій:

— доступ до контактів;

— доступ до календарю;

— доступ до браузера;

— likability(доступ до будь-якої сторінки за прямим посиланням).

Таким чином, на данному етапі для розробки прогресивних додатків під платформу Android є достатня кількість підтримуємих web-API сервісів, що можуть значно облегшити роботу розробнику, а також спростити та пошвидшити розробку мобільних додатків на платформі Android.

2.5 Підтримка роботи з нативними модулями. iOS

У порівнянні з Android платформою iOS досить піздно дозволила своїм пристроям використовувати прогресивні додатки, хоча на одній із презентацій компанії Apple ще у 2007 році було закладено фундамент для розвитку цього напрямку, у подальшому більшість роботи було зроблено компанією Google.

Основні можливості PWA на iOS:

— доступ до геолокації;

— доступ до акселерометру;

— доступ до магнітометру;

— доступ до гіроскопу;

— доступ до синтезу мови(тільки через піключену гарнітуру);

— доступ до камери;

— доступ до аудіо виходу;

— доступ до Apple Pay;

— доступ до WebAssembly;

— доступ до WebRTC;

— доступ до WebGL;

Якщо порівнювати PWA додатки з нативними реалізаціями, то є деякі недоліки з якими, на данному етапі, розробникам прогресивних додатків під iOS доведеться миритися, а саме:

— PWA додаток може зберігати локальні файли і дані у обсязі, який не перевищує 50 Мб;

— якщо додаток не використовувався на протязі декількох тижнів, iOS відформатує усі дані і при подальших запусках додаток буде вантажитися наново.

— відсутній доступ до виконання коду у фоновому режимі;

— відсутність доступу до приватної інформації - контакти, локація, а також відсутній доступ до убудованих соціальних мереж;

— відсутній доступ до убудованого сервісу In App Payments;

— відсутня можливість роботи з Split Views;

— відсутні push-повідомлення та можливість інтеграції Siri.

У підсумку можно зауважити, що на сьогодні прогресивні додатки на базі iOS є досить обмеженими, з дебільшого це пов’язано з політикої компанії Apple.

3 ОГЛЯД АРХІТЕКТУРНИХ РІШЕНЬ

Під час проектування маштабуємого ПО дуже важливою частиною є архітектура. Якщо раніше розробники не приділяли багато часу на проетування та планування архітектури інтерфесних додаткіх, то на сьогодні цьому пункту варто приділяти час, тому що з’являються такі проблеми як: зростаюча складніть, присутніть багатої кількості бізнес-правил, зростаючий об’єм даних, кількість розробників одночасно працюючих над одним додатком. Для того, щоб справлятися з вищезгаданими критеріями інтерфейсні додатки повинні мати досить обосновану архітектуру, яка зможе забезпечети комфортну та швидку розробку.

На сьогодняшній день web-додатки не просто відображуюсть дані і забезпечують користувачські введення, сьогодні такі додатки надають їх користувачам більш широкі можливості, а серверна частина(back-end) використовується більше як рівень збереження даних. Саме таке твердження означає, що дуже велика відповідальність була перекладена на інтерфейсну частину. Не зважаючи на те, що у процесі розробки зростає кількість бізнес-правил та даних, ще з’являється проблема продуктивності додатків, найбільш ефективним рішенням для підвищення продуктивності, як показує досвід, є правильна архітектура, але з цим пунктом з’являється проблема інвестицій у проектування ефективної архітектури.

Зазвичай архітектурні рішення ліпше приймати на етапі, коли додаток ще не встиг наповнитися бізнес-правилами та великим обсягом коду, коли розробники ще не встигли зштовхнутися з етапом, коли єдиним рішенням є - переробити додаток. Якщо проекти поділити на ті, які проінвестували в розробку підходящої архітектуру і які обрали швидкість розробки на первинних етапах, як головний кретерій, то отримаємо приблизно таку діаграму, як зображено на рисунку 3.1.

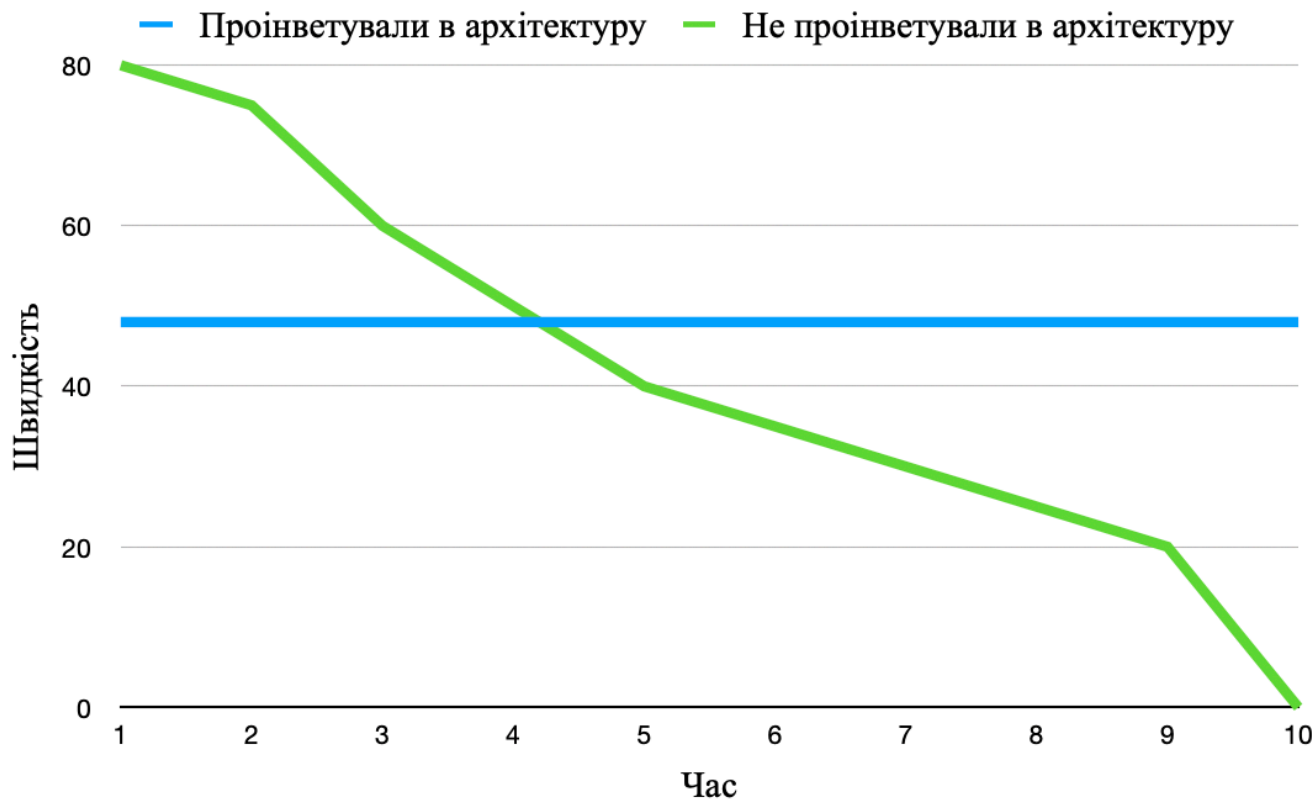

Рисунок 3.1 - Залежлість швидкості та часу розробки від інвестицій в архітектуру

З рисунку 3.1 можна зробити висновок, що правильна архітектура не тільки пришвидшує розробку у майбутньому, а ще й може зекономити кошти проекту.

3.1 Абстракції між прикладними рівнями

Абстракції між прикладними рівнями - один з перших підходів, який варто розглянути. На рисунку 3.2 зображено основна ідея даної концепції, яка полягає у тому, що потрібно помістити надлежну відповідальність в надлежний рівень системи.

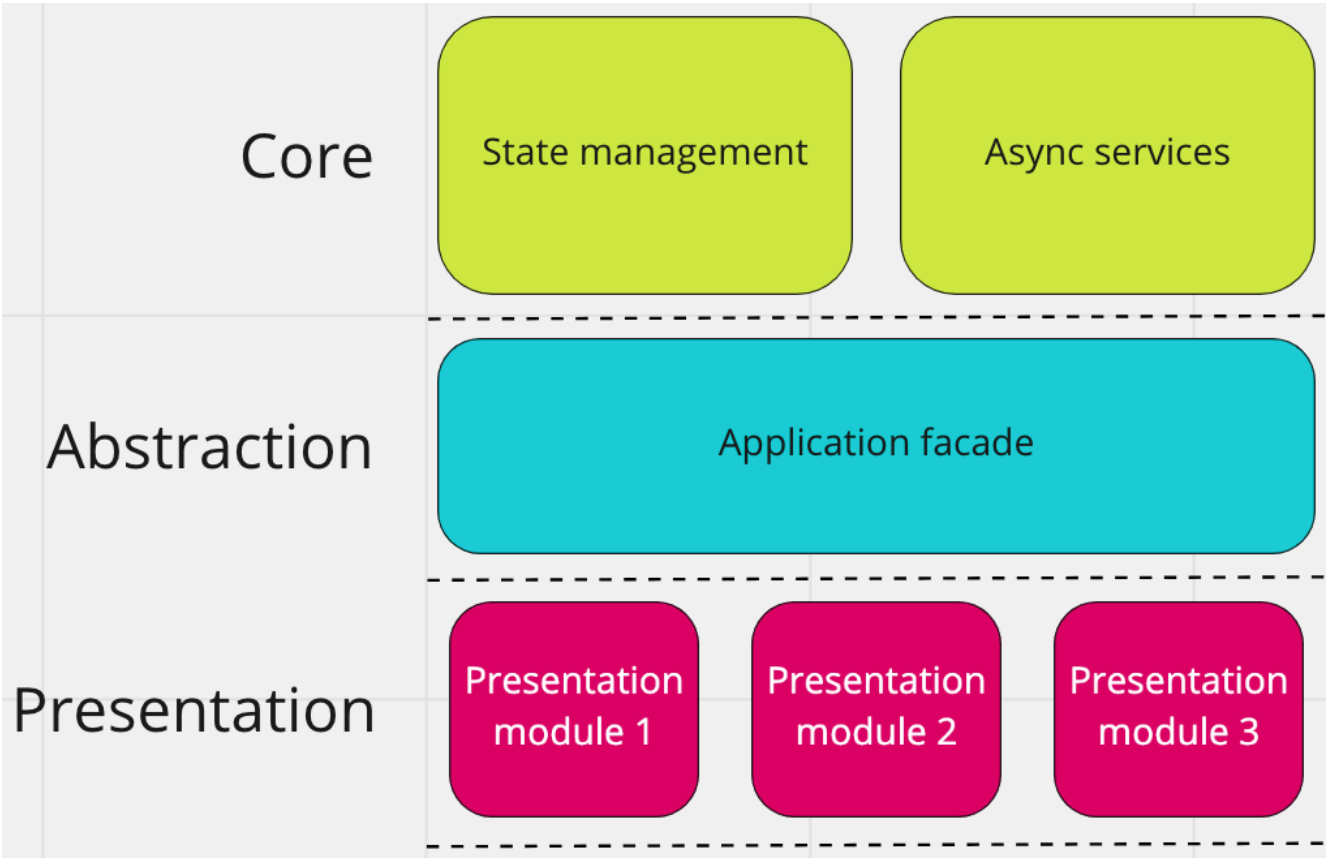

Рисунок 3.2 - Абстракції між прикладними рівнями

Такий підхід диктує правила «спілкування», це означає, що шар уявлення може спілкуватися з основним тільки через різноманітні рівні абстракції.

Рівень уявлення - це рівень, на якому будуть розташовуватися основні компоненти системи, не зважаючи на те, на якому саме фремворку буде вестися розробка(React, Angular та інші). Одні з головних обов'язків цього слою - презентація(користувацький інтерфейс) та делегування(перенаправлення дій користувача через слої абстракції), цей слой має інформацію про те як саме повинен відображатися компонент і що він повинен робити, но не має інформації про те як саме обробляти взаємодію з користувачем.

Рівень абстракції - це той рівень, який відділяє уявлення від основного рівня, саме на цьому рівні розташовуються інтерфейси та стан для рівня компонентів, які відображаються користувачу. Такий рівень виконує роль фасаду.

Інтерфейс абстракції - основні обов’язки даного слою вистявляти потоки стану і інтерфейсу компонетів. Різні методи для роботи зі станом додатку(у кожного фреймворку вони свої) абстрагують деталі керування і зовнішні визови API з компонентів уявлення. Крім того, зміни стану не мають значення для компонентів. Рівень представництва не повинен дбати про те, як все робиться, а компоненти повинні просто викликати методи з рівня абстракції, коли це необхідно. Вивчення відкритих методів на нашому рівні абстракції повинно дати нам швидке розуміння варіантів використання на високому рівні. Але треба пам’ятати, що рівень абстракції не є місцем реалізації ділової логіки. Тут просто пов’язується рівень уявлення з нашою бізнес-логікою, абстрагуючись від того, як він пов’язаний.

Стан - рівень абстракції, що робить компоненти уявлення незалежними від рішення по управлінню станом. Компонентам дають дані для відображення в шаблонах (зазвичай async), і їм повинно бути все одно, звідки ці дані. Для управління нашим станом ми можемо вибрати будь-яку бібліотеку управління станом, яку-саме буде залежати від фреймворку, на якому ведеться розробка, або просто можна створити свій власний механіз обробки стану компонетів уявлення. Цей рівень абстракції дає нам велику гнучкість і дозволяє змінити спосіб управління станом, навіть не впливаючи на рівень уявлення. Цей механізм дозволяє розробникам легко змінювати одні бібліотеки на інші при потребі.

Головний шар - це саме той шар, де реалізується основна логіка додатку. Усілякі маніпуляції з даними та зовнішнім світомвідбуваються саме тут. На рівні ядра ми також реалізуємо HTTP-запроси, сервіси API несуть тільки одну відповідальність - це зв'язок з кінцевими точками API і нічого більше. Ми повинні уникати кешування, логіки або маніпулювання даними на цьому рівні.

На цьому рівні також можна розмістити будь-які валідатори або більш складні власні випадки, які вимагають маніпуляції з багатьма фрагментами нашого стану інтерфейсу користувача.

Таким чином, кожен шар має свої чітко визначені межі та обов’язки. Також було визначено правила спілкування між шарами. Все це допомагає краще зрозуміти та міркувати про систему з часом, оскільки вона ускладнюється.

3.2 Архітектура Flux на основі React

Архітектура Flux - це не програмне забезпечення або пакет. Це набір архітектурних зразків, яким повинен керуватися розробник.

Flux архітектура, відповідальна за створення слоя даних у додатках JavaScript та розробці серверної сторони у веб-додатках. Flux доповнює складні компоненти вида View у React, використовуючи однонаправлений поток даних. Flux має 4 основні компоненти:

— сховище(store);

— диспетчер(dispatcher);

— уявлення(view);

— дія(action).

Це не схоже на звичний MVC, який часто використовується в інших фреймворках. Дійсно, там є Controller, але здебільшого це контролер, який відповідає за Views. Views знаходяться вгорі ієрархії, і вони передають функціонал і дані елементам-нащадкам.

Flux слід концепції односпрямованого потоку даних, що робить його простим для пошуку помилок. Дані проходять через прямий потік вашої програми. React і Flux - на даний момент два найпопулярніших фреймворка, які використовують принцип односпрямованого потоку даних.

У той час як React використовує віртуальний DOM для відображення змін, Flux робить це трохи інакше. У Flux, взаємодія з призначеним для користувача інтерфейсом викличе ряд дій, які можуть змінити дані програми.

Flux має відкритий вихідний код, і це скоріше шаблон проектування, а не фреймворк, тому його можна використовувати відразу. Те, що відрізняє його від інших фреймворків, то відрізняє його і від шаблону проектування MVC.

Flux допомагає зробити код більш передбачуваним, в порівнянні з MVC фреймворками. Розробники можуть створювати додатки, не турбуючись про складні взаємодії між джерелами даних.

Flux вигідно відрізняється більш організованим потоком даних - односпрямованим. Те, що він односпрямований, головна особливість Flux. Ці дії поширюються на нову систему щодо взаємодії з користувачем.

Також ви можете почати використовувати Flux, не використовуючи при цьому весь новий код.

На рисунку 3.3 зображено однонаправлений потік даних.

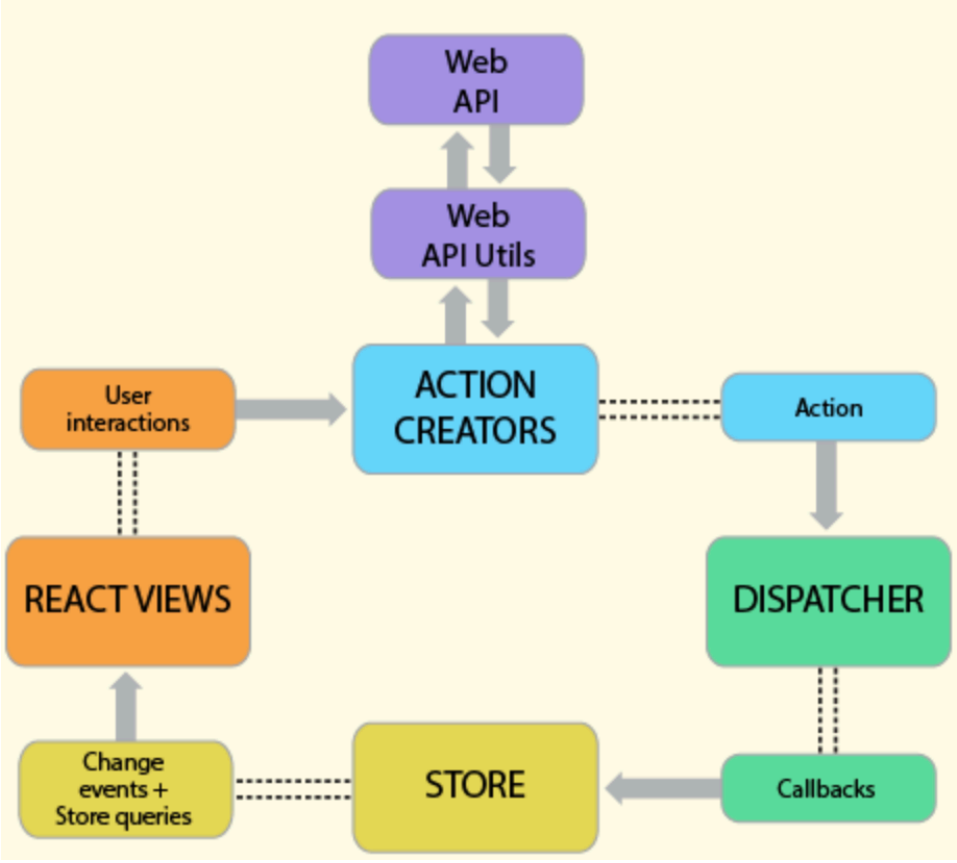

Рисунок 3.3 - Однонаправлений потік даних у Flux

Actions - помічники, які передають дані в Dispatcher.

Dispatcher - отримує Actions і передає зареєстрованим callback-ам.

Stores - діють як контейнери для стану програми та логіки. Реальна робота додатка відбувається в Stores. Stores, зареєстровані для прослуховування дій Dispatcher, будуть відповідно і оновлювати View.

Controller Views - React компоненти захоплюють стан з Stores, а потім передають дочірнім компонентам.

Контролери у звичному для розробників MVC і Flux розрізняються. Тут контролери є Controller-View і знаходяться на самій вершині ієрархії. View - це React компоненти.

Весь функціонал, як правило, знаходиться в Store. В Store виконується вся робота і повідомляється Dispatcher, які події / дії він прослуховує. Коли відбувається подія, Dispatcher відправляє "корисне навантаження" в Store, який зареєстрований для прослуховування конкретно цієї події. Тепер в Store необхідно оновити View, що в свою чергу викликає дію. Action, точно також як і ім'я, тип події та багато іншого відомі заздалегідь. View поширює Action через центральний Dispatcher, і це буде відправлено в різні Stores. Ці Stores будуть відображати бізнес-логіку програми та інші дані. Store оновлює все View. Найбільш добре це працює спільно зі стилем програмування React, тоді Store відправляє поновлення без необхідності детально описувати, як змінювати уявлення між станами. Це доводить, що шаблон проектування Flux слід за односпрямованим потоком даних. Action, Dispatcher, Store і View - незалежні вузли з конкретними вхідними та вихідними даними. Дані проходять через Dispatcher, центральний хаб, який в свою чергу управляє всіма даними. Dispatcher діє як реєстр із зареєстрованими callback-ами, на які відповідають Store. Store будуть давати зміни, які обрані Controller-Views.

Розповсюдження View у системі зображено на рисунку 3.4.

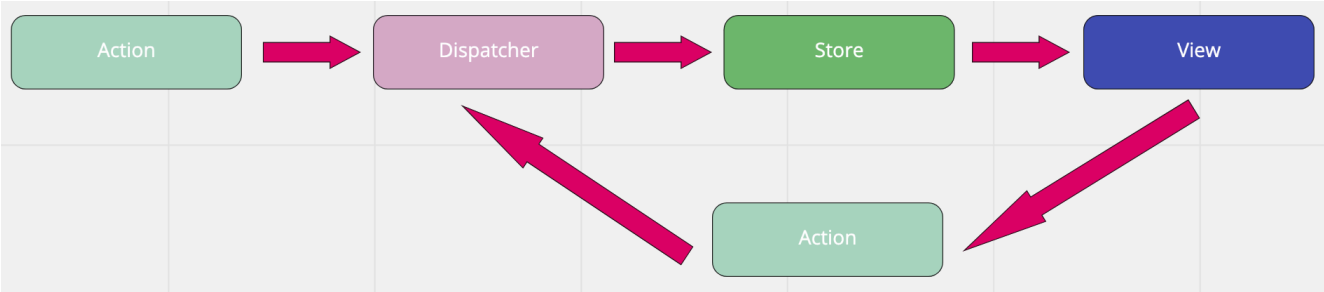

Рисунок 3.4 - Розповсюдження View у системі

Це доводить, що в Flux немає двосторонніх прив'язок (bind), що структура на кшталт функціональному відносного програмування, і ще щось на зразок потокового програмування.

Залежності, які відбуваються в Store, зберігаються в суворій ієрархії, тоді як Dispatcher обробляє оновлення. Ця структура також вирішує проблеми, які природним чином виникають при двосторонньої прив'язці.

3.3 Реалізація Flux. Redux

Redux - є передбачуваним контейнером стану для програм JavaScript. Це допомагає писати програми, які поводяться послідовно, працюють в різних середовищах (клієнтських, серверних та власних) і легко тестуються. Крім того, це забезпечує чудовий досвід для розробників, такий як редагування коду в реальному часі в поєднанні з налагоджувачем часу. Ви можете використовувати Redux разом з React або з будь-якою іншою бібліотекою перегляду. Він крихітний (2 кБ, включаючи залежності), але має велику екосистему додатків.

Основні переваги Redux:

— передбачуваність - Redux допомагає писати програми, які поводяться послідовно, працюють в різних середовищах (клієнтських, серверних та власних) і легко тестуються.

— централізація - централізація стану та логіки додатка забезпечує потужні можливості, такі як скасування / повтор, стійкість стану та багато іншого.

— налагодження - Redux дозволяє легко простежити, коли, де, чому та як змінився стан додатка. Архітектура Redux дозволяє реєструвати зміни, використовувати "налагодження подорожей у часі" і навіть надсилати на сервер повні звіти про помилки.

— гнучкий - Redux працює з будь-яким шаром інтерфейсу та має велику екосистему.

Весь глобальний стан додатка зберігається у дереві об’єктів в одному сховищі. Єдиний спосіб змінити дерево станів - це створити дію, об’єкт, що описує те, що сталося, і відправити його в сховище. Щоб вказати, як стан оновлюється у відповідь на дію, пишуться чисті функції редюсера, які обчислюють новий стан на основі старого стану та дії. На рисунку 3.5 зображено приклад редюсера Redux.

```
function counterReducer(state = { value: 0 }, action) {
  switch (action.type) {
    case 'counter/incremented':
      return { value: state.value + 1 }
    case 'counter/decremented':
      return { value: state.value - 1 }
    default:
      return state
  }
}
```

Рисунок 3.5 - Приклад редюсера Redux

Замість того, щоб безпосередньо змінювати стан, ви вказуєте мутації, які ви хочете мати з простими об’єктами, які називаються діями. Потім ви пишете спеціальну функцію, яка називається редуктором, щоб вирішити, як кожна дія трансформує весь стан програми.

У типовій програмі Redux існує лише один магазин з єдиною функцією зменшення кореневої функції. У міру зростання вашого додатка ви поділяєте кореневий редуктор на менші редуктори, які самостійно працюють на різних частинах дерева стану. Це точно так само, як у програмі React є лише один кореневий компонент, але він складається з безлічі дрібних компонентів.

Ця архітектура може здатися дуже схожою на програму-лічильник, але краса цього візерунка полягає в тому, наскільки добре він масштабується до великих та складних додатків. Це також дає можливість дуже потужних інструментів розробника, оскільки можна простежити кожну мутацію до дії, яка її спричинила. Ви можете записувати сеанси користувачів і відтворювати їх, просто відтворюючи кожну дію.

3.4 Кешоване управління станом. useSWR

SWR(stale-while-revalidate) - це стратегія спочатку повернути дані з кешу (stale), а потім надіслати запит на отримання (revalidate) і нарешті, надіслати із актуальними даними. На рисунку 3.6 зображено приклад отримання даних з стороннього API.

```
const Profile = () => {
  const fetcher = () => {}
  const { data, error } = useSWR('/api/user', fetcher)

  if (error) return <div>failed to load</div>
  if (!data) return <div>loading...</div>
  return <div>{data.name}</div>       You, seconds ago
}
```

Рисунок 3.6 - Приклад отримання даних з сторонньго API

У цьому прикладі хук useSWR приймає рядок ключів та функцію вибору. Ключ - це унікальний ідентифікатор даних (як правило, URL-адреса API), який передається завантажувачу. fetcher може бути будь-якою асинхронною функцією, яка повертає дані, ви можете використовувати власне отримання або такі інструменти, як Axios. У результаті виконання useSWR повертає дані та помилки на основі стану запиту.

Основні переваги useSWR:

— швидке, легке та багаторазове отримання даних;

— вбудований кеш;

— підтримка Typescript;

— використання на ReactNative;

— робота у реальному часі;

— швідка навігація;

— ревалідація на фокус;

— ревалідація при несправному підключенні;

— відновлення пагінації і розташування на сторінці;

— повторна відправка запросу при помилках.

При використанні useSWR можна досягти гарних показників продуктивності додатку, але стає неможливим використання Flux архітектури, тому що при використанні useSWR дані будуть знаходитися на рівні View, тобто на рівні компонентів, які буде бачити користувач.

Рішенням того, що дані не будуть зберігатися окремо від компонентів може стати використання HOC(high order components) - компоненти найвищого порядку - це означаає, що усе, що пов’язано з отриманням даних з сторонньго API та сховищем даних буде зверігатися в локальному стані компонентів вищого порядку. Приклад компонента найвищого порядку можна побачити на рисунку 3.7.

Рисунок 3.7 - Приклад HOC

3.5 Локальне сховище. LocalStorage

LocalStorage - властивість глобального об’єкту window лише для читання інтерфейсу вікна дозволяє отримати доступ до об'єкта Storage для походження документа, слід зауважити, що збережені дані зберігаються протягом сеансів браузера. LocalStorage подібний до sessionStorage, за винятком того, що хоча дані localStorage не мають часу закінчення, дані sessionStorage очищаються, коли сеанс сторінки закінчується - тобто, коли сторінку закрито.

LocalStorage зазвичай використовується для невеликих об’ємів даних, які не перевищують обсяг 5 Мб. На рисунку 3.8 зображено таблицю підтримки у різних браузерах.

	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	WebView Android	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet
localStorage	4	12	3.5	8	10.5	4	37	18	4	11	3.2	1.0

Рисунок 3.8 - Підтримка LocalStorage у різних браузерах

LocalStorage - це лише рядок, що зберігає ключове значення, із спрощеним синхронним API. Ця остання частина є ключовою. Хоча в специфікації немає нічого, що забороняє асинхронне веб-сховище, на даний момент всі реалізації є синхронними (тобто блокують запити).

3.6 Локальне сховище. IndexedDB

IndexedDB - це широкомасштабна система зберігання даних NoSQL. Це дозволяє зберігати в браузері користувача майже все, що завгодно. На додаток до звичайних дій пошуку, отримання та введення, IndexedDB також підтримує транзакції.

IndexedDB - API низького рівня для зберігання на стороні клієнта значних обсягів структурованих даних, включаючи файли / великі крапки. Цей API використовує індекси, щоб забезпечити високопродуктивний пошук цих даних. Хоча DOM Storage корисний для зберігання менших обсягів даних, це менш корисно для зберігання більших обсягів структурованих даних.

Кожна база даних IndexedDB унікальна для джерела (як правило, це домен сайту або субдомен), що означає, що він не може отримати доступ або отримати доступ до нього будь-яким іншим джерелом. Обмеження зберігання даних, як правило, досить великі, якщо вони взагалі існують, але різні браузери по-різному обробляють обмеження та виселення даних.

Так як IndexedDB є вбудованою у сучасні браузери, то існують деякі обмеження, на рисунку 3.9 зображено таблицю підтримки IndexedDB у різних браузерах

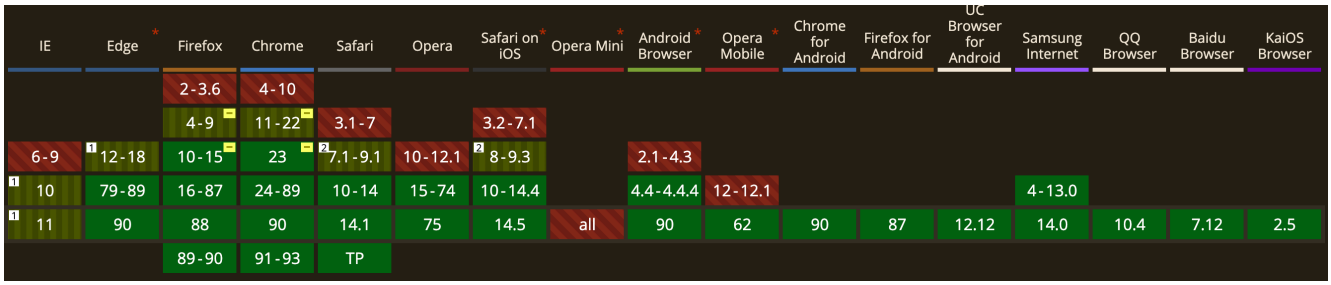

Рисунок 3.9 - Підтримка IndexedDB у різних браузерах

3.7 Використання контексту компонентів React. useContext

Контекст React ініціалізується за допомогою API верхнього рівня createContext. В залежності від ситуації та розроблюваного ПО контекст може використовутатися для передачі даних між різноманітними компонетами у додатку, на рисунку 3.10 зображено приклад створення контексту у React:

```
const ThemeContext = createContext(null);
```

Рисунок 3.10 - Приклад створення контексту у React

Для подальшого використання потрібно зрозуміти на яку саме галузь додатку буде розповсюджуватися контекст, якщо це з самого початку не є досить прозорим, то контекст можна помістити у файл входу React, зазвичай це ‘./src/app.tsx’. На рисунку 3.11 зображено приклад огортання провайдером контексту головної точки входу.

```
const App = ({ Component, pageProps }: AppProps) => {
  useRemoteConfig()
  useBranchInit()

  return (
    <>
      <ThemeContext.Provider value="green">
        <GlobalStyle />
        <Component {...pageProps} />
        <LoginPhoneNumberElements />
      </ThemeContext.Provider>
    </>
  )
}
```

Рисунок 3.11 - Приклад огортання провайдером контексту

Надалі, якщо є потреба використати значення, які були передані до провайдеру контексту, можна використати функцію-хук useContext у компоненті, де нам потрібний певний набір даних, на рисунку 3.12 зображено приклад використання функції-хука useContext.

```
const ChildComponent = () => {
  const color = React.useContext(ThemeContext);

  return (<div>{color}</div>
  );
};
```

Рисунок 3.12 - Приклад використання функції-хука useContext

Таким чином контекст у React може стати альтернативним методом організції коду та архітектури додатку.

3.8 Керування сторонніми ефектами. Redux-Saga

Redux-Saga - це бібліотека, яка створена для керування сторонніми ефектами при розробці додатків з дебільшого на JavaScript.

На цьому етапі потрібно ввести поняття сайд-ефекту або сторонньго ефекту - будь-який код, який працює асинхронно або звертається до зовнішнього джерела програми, наприклад:

— взаємодія з API;

— логування;

— доступ до локальних сховищ браузера;

— доступ до Redux, зручно у ситуації, коли потрібно модифікувати Redux сховище;

— можливість викликаті різні події Redux у межах однієї саги;

— можливість керування сторонніми ефектами(створення, відміна, зміна та інші);

— можливість використовувати саги по декілька разів;

— відкладення виклику сторонніх ефектів до тих пір, поки не пройде кілька мілісекунд після останнього виклику;

— регулювання швидкості, з якою запускаються сторонні ефекти додатка, тобто запобігати запуску сторонніх ефектів більше одного разу на кожні X мілісекунд.

— race - відстеження декількох сторонніх ефектів і у ситуації, коли один із запущенних ефектів завершується відміняється виконання усіх інших;

— запуск декількох ефектів паралельно.

Саги реалізовані як функції генератори, які передають об’єкти проміжному програмному забезпеченню.

Генератори - це основна концепція Redux-Saga. Функція генератора визначається оголошенням function* (ключове слово функції, після якого зірочка).

Функції генератора можуть бути закриті, а згодом повторно введені. Їхній контекст (прив'язки змінних) буде збережено при повторних входах. Функція генератора повертає об'єкт генератора, який є своєрідним ітератором.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Порівняння PWA додатків та гібридних додатків

Під гібридними додатками слід розуміти додатки, які були зроблені за допомогою таких засобів розробки як: ReactNative, Apache Cordova, Flutter та інші. Нижче у таблиці 4.1 було розглянуто та порівняно сильні та слабкі сторони PWA додатків у порівнянні з гібридними.

Таблиця 4.1 - Порівняння сильних та слабкіх сторін PWA додатків з гібридними

Метод розробки	Сильні сторони	Слабі сторони
PWA додатки	• Офлайн-режим • Потрібен HTTPS • Незалежно від App Store • Простота впровадження • Час розробки	• Слабкий доступ до функцій • Гірша продуктивність • Не повна підтримка браузерами / платформами
Гібридні додатки	• Уніфікована кодова база • Офлайн режим • Доступ до декількох платформ • Широкі можливості	• Продуктивність • API платформи • Проблеми з UX при розробці під різні платформи • Схвалення у різних магазинах програм

З таблиці 4.1 можна зробити висновок, що підхід PWA швидко наздоганяє гібридний стиль розвитку менші відмінності в плані доступу до обладнання та платформи при одночасному розмежуванні з точки зору базової технології, оскільки PWA використовують найкращі практики - Інтернет та функціональність браузера, в той час як гібридний підхід використовує перевага веб-технологій, але замість цього обгортає їх у рідну обгортку для нативних можливостей.

Також слід звернути увагу на споживання батареї переважно на пристроях під платформою Android. Порівняльна таблиця 4.2 використання батареї PWA додатками та гібридними на пристроях Android зображена нижче.

Таблиця 4.2 - Порівняння використання батареї пристрою при використанні PWA додатку та гібридного додатку

	Найвищий %	Найнижчий %	Середина %	Стандартне відхилення %
PWA додатки	0.17	0.09	0.12	0.021
Гібридні додатки	0.14	0.08	0.11	0.017

З таблиці 4.2 можна зробити висновки, що хоч PWA додатки і витрачають більше енергії Android пристрою у процентах, але відхилення є незначним для кінцевого користувача.

4.2 Керування станом та сторонніми ефектами компонентів

На сьогодні існує декілька бібліотек, які допомогають розробникам керувати станом розроблюваного додатку, а саме: redux-saga, redux-thunk, redux-observable, redux-promise. Було обрано такі критерії для порівняння: відмітка розробника на gitHub, використання прогрманого коду додатку для побудови власного, кількість багів, остання дата оновлення, розмір. На таблиці 4.3 зображено порівняння обраних бібліотек.

Таблиця 4.3 - Порівняння бібліотек для керування станом та сторонніми ефектами у програмних додатках

	Відмітка розробника на gitHub	Використання прогрманого коду додатку для побудови власного	Кількість багів	Остання дата оновлення	Розмір
Redux-saga	19 912	1771	160	Лютий, 2015	5,3 KB
Redux-thunk	14 106	792	2	Липень, 2015	236 B
Redux-observable	7084	459	91	Квітень, 2016	1,5 KB
Redux-promise	2590	127	29	Липень, 2015	1,1 KB

Таким чином з вищеуказаної таблиці 4.3 було прийнято рішення, що оптимальним рішенням для побудови PWA додатку є redux-saga, навіть не зважаючи на найбіший розмір бібліотеки. Усі переваги даного вибору указані у розділі 3.8 кваліфікаційної роботи.

4.3 Сховище даних PWA додатку

У ході дослідження було розглянуто декілька варіантів кешування на стороні браузеру: localStorage, indexedDB.

При створенні класичного web-додатку зазвичай використовують localStorage, тому що простий у використанні і має дуже низький порог старту.

IndexedDB - це система транзакційних баз даних, подібно до СУБД на базі SQL. Однак, на відміну від СУБД на базі SQL, які використовують таблиці з фіксованими стовпцями, IndexedDB є об'єктно-орієнтованою базою даних на основі JavaScript. IndexedDB дозволяє зберігати та отримувати об'єкти, проіндексовані за допомогою ключа; будь-які об'єкти, що підтримуються алгоритмом структурованого клонування, можуть зберігатися. Вам потрібно вказати схему бази даних, відкрити підключення до бази даних, а потім отримати та оновити дані в рамках серії транзакцій.

Зрозуміло, що IndexedDB дуже потужний, але це, звичайно, звучить не дуже просто.

Перевагою є доступ до даних в автономному режимі, але простим способом - подібно до того, як можна було б це зробити з localStorage, який має надзвичайно простий API. Для порівняння було обрано такі критерії: об’єм даних, асинхронність, підтримка тразакцій, можливість індексувати таблиці, можливість використання у ServiceWorker, рівень абстракції для роботи з сховищем.

Таблиця 4.4 - Порівняння localStorage та indexedDB

	localStorage	indexedDB
Об’єм даних	5 Mb	Залежить від сховища пристрою
Асинхронність	-	+
Підтримка тразакцій	-	+
Можливість індексувати таблиці	-	+

Продовження таблиці 4.4

	localStorage	indexedDB
Можливість використання у ServiceWorker	-	+
Рівень абстракції для роботи з сховищем	+	-

З вищевказаної таблиці можна зрозуміта, що indexedDB є найкращим та єдиним локальним сховищем при розробці PWA додатків на сьогодняшній день, тому що один з найважливіших критеріїв, який пов’язаний з доступом до сховища у рамках ServiceWorker.

4.4 Монорепозиторій

Монорепозиторій - це принцип побудови проекту, коли різні частини, які на перший погляд не повинні знаходитися в одному місці та запускатися однією командою, знаходяться поруч. Монорепозиторій при побудові PWA додатку зручно викоистовувати, тому що можна відділити, якісь загальні компоненти від основної логіки та зручно використовувати. Для порівняння було обрано такі бібліотеки: yarn-workflows, lerna.

Існує декілька варіантів побудови монорепозиторія, а саме:

— lerna з пакетним менеджером npm;

— lerna з пакетним менеджером yarn;

— yarn workspaces;

— lerna у поєднанні з yarn workspaces.

Надалі було проведено порівняння швидкості встановлення пакетів та бібліотек при усіх вищезгаданих комбінаціях. Для порівняння було вирішення взяти час установки та час начальної загрузки модулів. Встановлення проводились

на версії nodeJs v12.22.1. На таблиці 4.5 результати lerna з пакетним менеджером npm.

Таблиця 4.5 - lerna з пакетним менеджером npm

№	Час встановлення	Час загрузки	Загальний час
1	39.1	64.7	103.8
2	23.9	34.8	58.8
3	29.4	35.8	65.2

Надалі було проведено порівняння для lerna з пакетним менеджером yarn. Результати наведено у таблиці 4.6.

Таблиця 4.6 - lerna з пакетним менеджером yarn

№	Час встановлення	Час загрузки	Загальний час
1	36.5	58.5	95
2	24.4	36.8	61.2
3	31.5	40.5	72

Надалі було проведено порівняння для yarn workspaces. Результати наведено у таблиці 4.7.

Таблиця 4.7 - yarn workspaces

№	Час встановлення	Час загрузки	Загальний час
1	34.9	0	34.9
2	17.7	0	17.7
3	13.4	0	13.4

Надалі було проведено порівняння для lerna у поєднанні з yarn workspaces. Результати наведено у таблиці 4.8.

Таблиця 4.8 - lerna у поєднанні з yarn workspaces

№	Час встановлення	Час загрузки	Загальний час
1	60.4	0	60.4
2	32.7	0	32.7
3	17.8	0	17.8

Таким чином, після проведених досліджень, можна зробити висновок, що yarn workspaces є найкращим рішенням для побудови монорепозиторія, бо саме цей модуль поміщає час загрузки в час встановлення, та витрачає при цьюму меньше всього часу. Також така перевага є досить важливою при запуску різних тестів у репозиторії, бо саме вона зможе не марнувати час на великій кількості встановлюваємих модулів.

4.5 Базова архітектура PWA

Архітектура з якої починається розробка програмного додатку є дуже важливою, тому що саме вона може вирішувати на скільки довго проект буде легко підтримувати різноманітним розробникам. Для побудови PWA додатку було обрано у порівняння два підходи, а саме MVC та Flux(переваги даного підходу описані вище у розділі 3.2).

Такий архітектурний підхід як MVC існує вже досить давно і до недавнього часу був одним із найактуальніших підходів у побудові web-додатків. Головний принцип підходу зображений на рисунку 4.1.

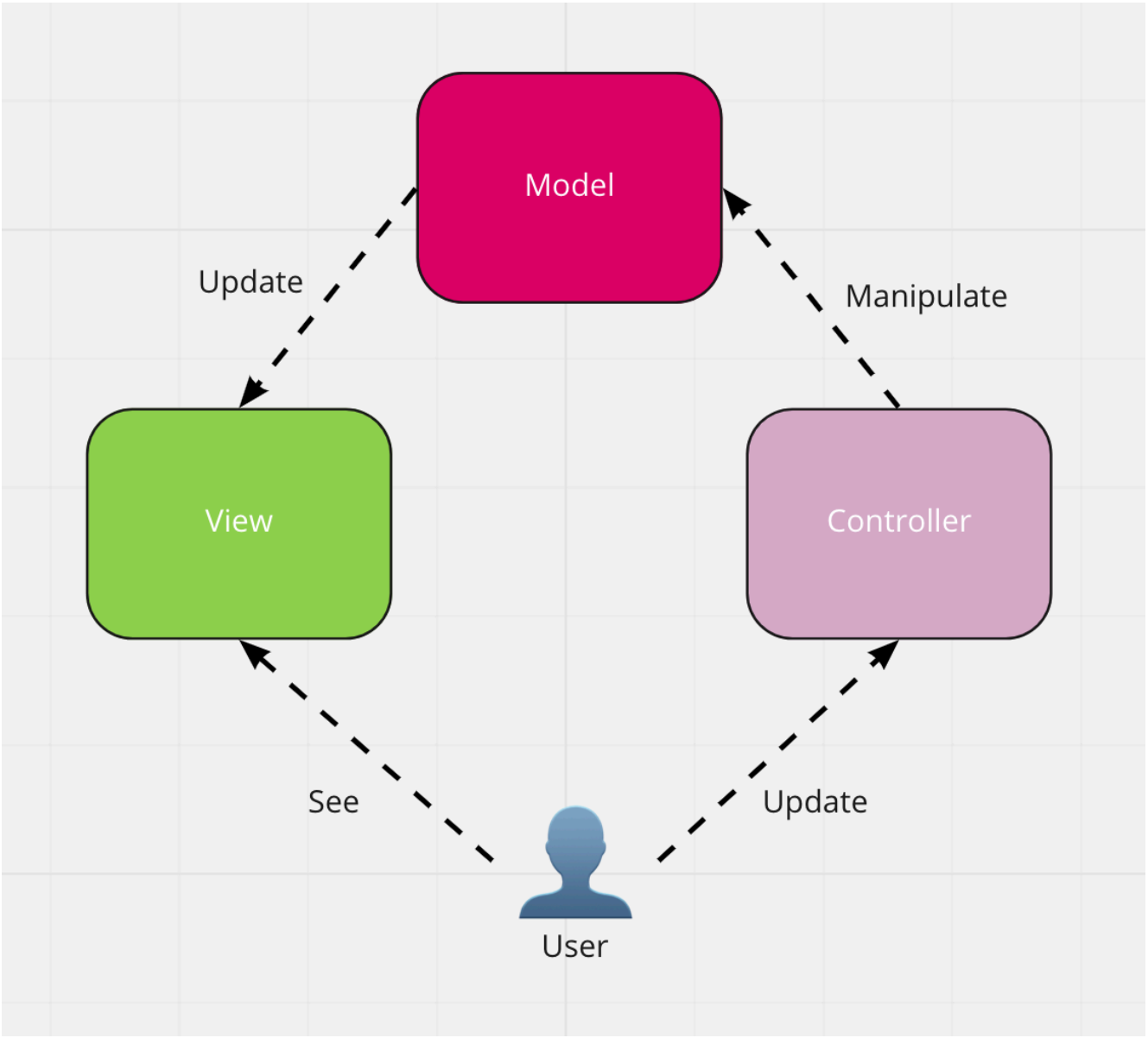

Рисунок 4.1 - MVC

Для пояснення принципу, модель керує поведінкою та даними додатку, view відповідає за відображення користувачу результатів обробки та UX, контролер в свою чергу приймає користувацькі результати, маніпулює моделью та викликає обновлення зовнішнього вигляду додатка.

Основними перевагами такого підходу є зручність тестування відділених одне від одного презентаційного слою від моделі та відділення презентації від контролера є дуже корисним при побудові web-додатків.

З часом стало зрозуміло, що такий підхід з часом збільшення програмного коду та збільшенням команди є не дуже ефективним(це стосується здебільшого клієнтської розробки), а саме те, що стає важко відстежувати різні зв'язки у поведінці програмного додатку. На рисунку 4.2 зображено ситуацію при зростанні кодової бази web-додатка.

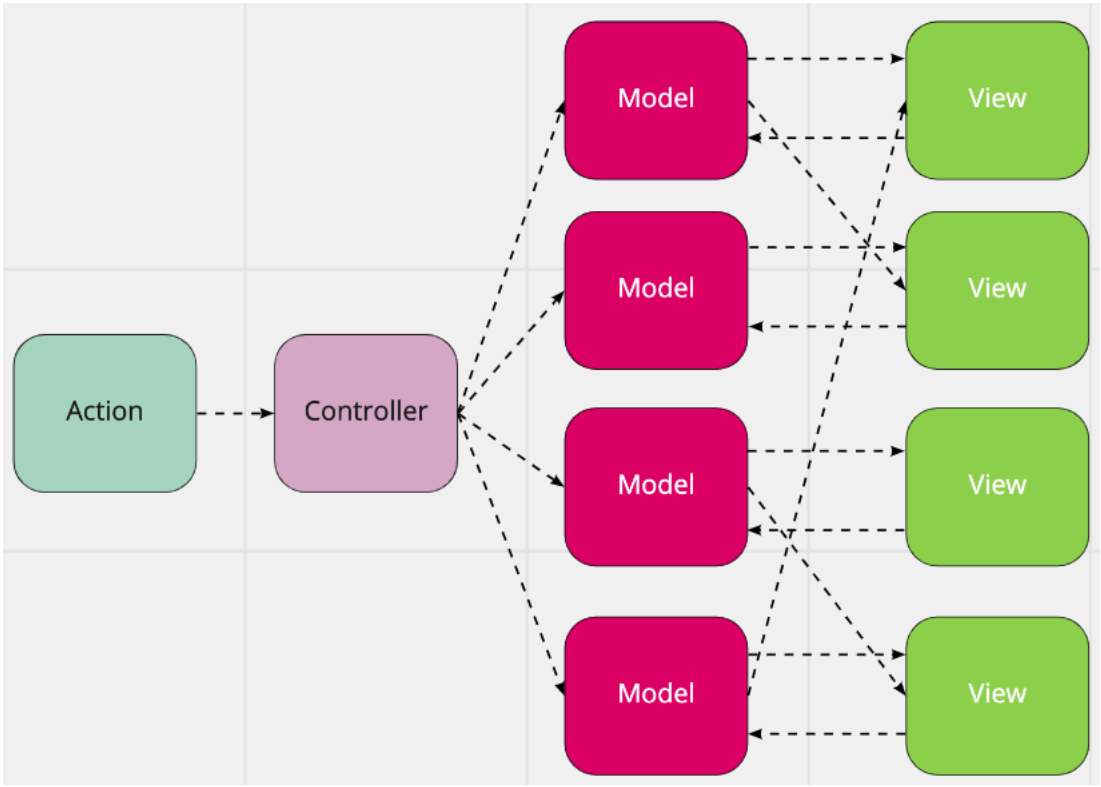

Рисунок 4.2 - MVC, збільшена кодова база web-додатка

При збільшенні та зростанні проекту та при використанні такого підходу з часом буде дуже важко відстежувати бізнес-логіку додатка та вносити корективи, а особливо шукати помилки, тому більш підходящим рішенням для побудови web-додатків є архітектура Flux(детальніший опис підходу див. У розділі 3.2), яка є односпрямованою.

5 РЕАЛІЗАЦІЯ АРХІТЕКТУРИ ТА МЕТОДІВ PWA ДОДАТКУ

5.1 Монорепозиторій

Для побудови архітектури PWA додатку було обрано фреймворк на базі JavaScript - React, тому що саме в цьому фреймворку вперше було впроваджено архітектурний підхід Flux для web-додатків, те, що саме цей підхід є найбільш вдалим було розглянуто у розділі 4.5.

Для того, щоб реалізувати монорепозиторій було обрано yarn workflows, яка є найбільш правільним рішенням, це було розлянуто у розділі 4.4.

У результаті вищеприйнятих рішень було отримано наступну структуру, яку зображено на рисунку 5.1.

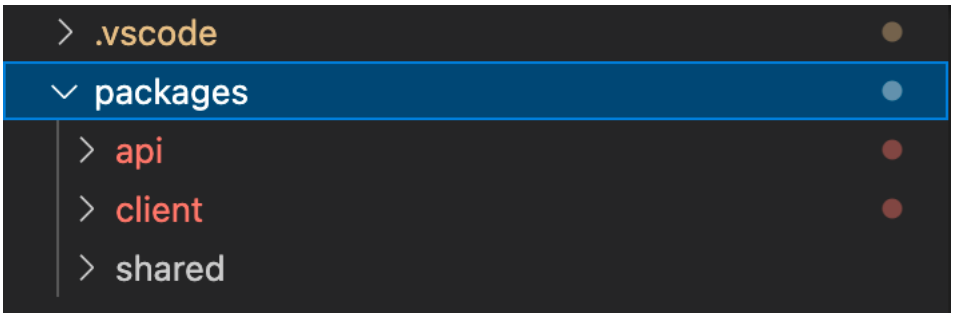

Рисунок 5.1 - Реалізація монорепозиторія

Таким чином було отримано 3 репозиторії, які повністю відповідають за коректну роботу програмного додатку.

Репозиторій api - відповідає за серверну частину додатку.

Репозиторій client - відповідає за клієнтську частину додатку.

Репозиторій shared - відповідає за зберігання реалізації по декільку разів використовуваних компонентів для клієнтського репозиторію. Таке рішення було прийнято, щоб зменьшити розмір файлу, який отримується при збірці додатку та

щоб пришвидшити швидке перевантеження додатку у реальному часі під час розробки.

5.2 Функції-помічники

Надалі було реалізовано архітектуру Flux у PWA додатку та для більш комфортного використання такого підходу у PWA було реалізовано декілька функцій-помічників для обробки API-запитів та для більш зручної обробки результатів цих запитів у контексті Redux та Redux-Saga. Нижче приведено код допоміжної функції makeCollectionFetcher, яка виконує перевірку стану додатку, визначає, коли в останнє були оновлені дані, робить запит до API і у разі успішної відповіді оновлює локальне сховище:

```
export function makeCollectionFetcher<
  ApiSuccessResponse extends PaginatedResponse<T>,
  // eslint-disable-next-line @typescript-eslint/no-explicit-any
  ApiFailResponse extends FailResponse<any, any>,
  // eslint-disable-next-line @typescript-eslint/no-explicit-any
  T extends AbstractEntry = any,
  SuccessPayload extends T[] = ApiSuccessResponse['payload']['list'],
  FailPayload = ApiFailResponse['payload']
>(
  // eslint-disable-next-line @typescript-eslint/no-explicit-any
  collection: Dexie.Table<T, any>,
  uri: string,
  // eslint-disable-next-line @typescript-eslint/no-explicit-any
  successAction: ActionCreator<Action<any, SuccessPayload>>,
  // eslint-disable-next-line @typescript-eslint/no-explicit-any
  failAction: ActionCreator<Action<any, FailPayload>>,
) {
  const name = collection.name
  const keyName = `${name}-state`
  return function*() {
    const settings = localStorage.getItem(keyName)
```

```
  ? JSON.parse(localStorage.getItem(keyName) || '')
  : {
      lastUpdate: new Date(),
    }

let hasMore = true
let page = 0
// eslint-disable-next-line @typescript-eslint/ban-ts-ignore
// @ts-ignore
let data: SuccessPayload = [] as T[]
const isOnline = yield select(isTransportAvailable)
if (!isOnline) {
  return
}
yield call(() => collection.clear())
do {
  page++
  try {
    const response: AxiosResponse<ApiSuccessResponse> = yield call(
      // eslint-disable-next-line @typescript-eslint/ban-ts-ignore
      // @ts-ignore Because these methods exist
      client.get,
      uri,
      {
        params: {
          page,
        },
      },
    )
    const {
      data: {
        payload: { list, pageCount },
      },
    } = response
    list.forEach(item => {
      // eslint-disable-next-line @typescript-eslint/ban-ts-ignore
      // @ts-ignore
      item.createdAt = new Date(Date.parse(item.createdAt))
      // eslint-disable-next-line @typescript-eslint/ban-ts-ignore
```

```
            // @ts-ignore
            item.modifiedAt = new Date(Date.parse(item.modifiedAt))
          })
          hasMore = pageCount > page
          yield call(() => collection.bulkAdd(list))
          // eslint-disable-next-line @typescript-eslint/ban-ts-ignore
          // @ts-ignore
          data = data.concat(list)
        } catch (err) {
          const { response } = err as AxiosError<ApiFailResponse>
          if (!response) {
            throw new Error('Unhandled error')
          }
          hasMore = false
          yield put(failAction(response.data.payload))
        }
      } while (hasMore)
      settings.lastUpdate = new Date()
      localStorage.setItem(keyName, JSON.stringify(settings))
      yield put(successAction(data))
    }
  }
```

Також було реалізовано функцію-помічник makeApiCaller, яка відповідала за виконання запитів до стороннього API, здебільшого з перевірками на методи POST та PUT і маніпулювання їх виконання у режимі доступу до інтернету та у режимі без інтернету, нижче приведено код реалізації данної функції:

```
export function makeApiCaller<
  RequestParams,
  ApiSuccessResponse extends SuccessResponse<any>,
  ApiFailResponse extends FailResponse<any, any>,
  SuccessPayload = ApiSuccessResponse['payload'],
  FailPayload = ApiFailResponse['payload']
>(
  uri: string,
  successAction: ActionCreator<Action<any, SuccessPayload>>,
  failAction: ActionCreator<Action<any, FailPayload>>,
  method: 'GET' | 'POST' | 'DELETE' | 'PATCH' | 'PUT' = 'POST',
  storeCollection?: Dexie.Table<RequestParams, any>,
```

```
) {
  const params = uri.match(/:([\w]+)/g) || []
  return function*(
    action: Action<
      any,
      RequestParams,
      { resolve?: Function; reject?: Function }
    >,
  ) {
    const isOnline = yield select(isTransportAvailable)
    if (!isOnline && storeCollection) {
      switch (method) {
        case 'POST': {
          const lastObj = yield call(() =>
            storeCollection
              .orderBy('id')
              .reverse()
              .limit(1)
              .first(),
          )
          const id = yield call(() =>
            storeCollection.add({
              ...action.payload,
              id: lastObj.id + 1,
              _id: (lastObj.id + 1).toString(),
              createdAt: new Date(),
              modifiedAt: new Date(),
              transport: 'pending',
            }),
          )
          if (action.meta && action.meta.resolve) {
            action.meta.resolve()
          }
          yield put(
            // eslint-disable-next-line @typescript-eslint/ban-ts-ignore
            // @ts-ignore
            successAction({
              id,
              _id: id,
```

```
            ...action.payload,
          }),
        )
        return
      }
      case 'PUT': {
        const origin = yield call(() =>
          storeCollection.get(
            // eslint-disable-next-line @typescript-eslint/ban-ts-ignore
            // @ts-ignore
            action.payload.id
              ? /**
                // @ts-ignore */
                action.payload.id
              : {
                  // eslint-disable-next-line @typescript-eslint/ban-ts-
ignore
                  // @ts-ignore
                  _id: action.payload._id,
                },
          ),
        )
        yield call(() =>
          storeCollection.update(origin.id, {
            ...action.payload,
            id: origin.id,
            // eslint-disable-next-line @typescript-eslint/ban-ts-ignore
            // @ts-ignore
            createdAt: new Date(action.payload.createdAt),
            modifiedAt: new Date(),
            transport: 'pending',
          }),
        )
        if (action.meta && action.meta.resolve) {
          action.meta.resolve()
        }
        // eslint-disable-next-line @typescript-eslint/ban-ts-ignore
        // @ts-ignore
        yield put(successAction(action.payload))
```

```
          return
        }
        default:
          throw new Error(
            `${method} is not implemented for offline mode for
${action.type}`,
          )
      }
    }
    try {
      const finalUri = params.length
        ? params.reduce((target, key) => {
            const item = key.replace(':', '')
            if (!Object.prototype.hasOwnProperty.call(action.payload,
item)) {
      const item =
        response.data.payload.createdAt && response.data.payload.modifiedAt
          ? {
              ...response.data.payload,
              // eslint-disable-next-line @typescript-eslint/ban-ts-ignore
              // @ts-ignore
              createdAt: new
Date(Date.parse(response.data.payload.createdAt)),
              // eslint-disable-next-line @typescript-eslint/ban-ts-ignore
              // @ts-ignore
              modifiedAt: new Date(
                Date.parse(response.data.payload.modifiedAt),
              ),
            }
          : response.data.payload
      if (action.meta && action.meta.resolve) {
        action.meta.resolve()
      }
      yield put(successAction(item))
    } catch (err) {
      const { response } = err as AxiosError<ApiFailResponse>
      if (!response) {
        console.error(err)
        throw new Error(err.message)
```

```
      }
      if (action.meta && action.meta.reject) {
        action.meta.reject(response)
      }
      yield put(failAction(response.data.payload))
    }
  }
}
```

Присутність таких помічних функцій домомагає тримати стан коду у сагах більш чистим та зрозумілим для розробника, нижче наведено приклад коду саги з використанням такої реалізації:

```
export const onCreateRequest = makeApiCaller<
  Collection,
  SuccessResponse<Collection>,
  ValidationResponse<Collection>
>(
  '/secure/someRoute',
  createAction(SomeAction.SUCCESS),
  createAction(SomeAction.FAIL),
  'POST',
  SomeCollection,
)
```

При такому підході розробнику стає легче орієнтуватися у власному коді, саги по усьому додатку залишаються досить чистими від постійних перевірок на належність певних даних у локальному сховищі, завжди зберігаються та підтягуються у користувацький інтерфейс тільки актуальні дані.

ВИСНОВКИ

Головною метою роботи було оцінити методи та аріхектуру побудови PWA додатків. Значну кількість часу було вкладено для отримання розуміння теми та її компонентів. Плюси і мінуси були проаналізовані шляхом порівняння PWA з іншими програмами та нативними додатками.

PWA не існує для заміни нативних програм. Це просто новий підхід зробити веб-програми більш схожими на нативні. Він швидко набирає популярність завдяки своїм особливостям, легкому процесу розробки, створює та застосовує стратегію в будь-якому місці. Однак PWA стикається з проблемами підтримки браузерів різних функціональних можливостей. Firefox та Safari мають деякі проблеми з App Manifest та Service Worker. Однак Chrome, Chrome для Android, UC Browser для Android і Samsung Internet, схоже, мають хорошу підтримку всіх функціональних можливостей.

На основі досліджень та практичного впровадження можна зробити висновок, що PWA може виглядати як загальнодоступне слово навколо вдосконаленого веб-стандарту, але позитивний вплив, який він може принести для бізнесу, так і для користувачів, не можна заперечувати. Він запропонував функції та вдосконалення для мобільного Інтернету, які до цього часу були ексклюзивним доменом нативних програм. Таким чином, такі компанії, як Flipkart, Housing.com, AliExpress, Financial Times, Google I / O, Twitter Lite, Forbes та BookMyShow впровадили PWA та зазнали зростання свого бізнесу. Крім того, Google надає стимули та докладає великих зусиль, надаючи навчальні посібники та підтримку PWA.

За результатами проведеного аналізу та порівняння PWA з нативними додатками було виявлено, що PWA потрібно використовувати для покращення користувацького досвіду, підвищення конверсії та якщо користувачі у своїй переважній більшості полюбляють досвід використання нативних додатків.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Gabor Cselle, “Tales of Creation”. URL: http://blog.gaborcselle.com/2012/10/every-step-costs-you-20-of-users.html (дата звернення: 02.03.2021).

2. Markov Danny, “Everything You Should Know About Progressive Web Apps”. Документація. URL: http://tutorialzine.com/2016/09/everything-you-should-know-about-progressive- web-apps/ (дата звернення: 06.03.2021).

3. Farrugia Kevin, “A Beginner’s Guide To Progressive Web Apps”. URL: https://www.smashingmagazine.com/2016/08/a-beginners-guide-to-progressive-web-apps/ (дата звернення: 04.03.2021).

4. Reshetilo Kateryna and Opanasenko Sergey, “Technology Org”. URL: https://www.technology.org/2017/07/28/progressive-web-apps-vs-native-which-is-better-for-your-business/ (дата звернення: 20.03.2021).

5. Osmani Addy, “The App Shell Model”. URL: https://developers.google.com/web/fundamentals/architecture/app-shell (дата звернення: 21.03.2021).

6. Ater Tal, Building Progressive Web Apps, O’Reilly.

7. Burtoft Jeff, “ThisHereWeb”. URL: https://thishereweb.com/understanding-the-manifest-for-web-app-3f6cd2b853d6 (дата звернення: 22.03.2021).

8. Gaunt Matt, “The Web App Manifest”. URL: https://developers.google.com/web/fundamentals/web-app-manifest (дата звернення: 23.03.2021).

9. Gaunt Matt, “Service Workers: an Introduction”. URL: https://developers.google.com/web/fundamentals/getting-started/primers/service-workers (дата звернення: 24.03.2021).

10. Mashtalir, S., Mashtalir, V., Stolbovyi, M.(2018). Proceedings of the 2018 IEEE 2nd International Conference on Data Stream Mining and Processing, DSMP 2018, 2018, pp. 545–548, 8478493(дата звернення: 28.03.2021).

11. Mashtalir, V.P., Shlyakhov, V.V., Yakovlev, S.V.Group structures on quotient sets in classification problems. Cybernetics and Systems Analysis, 2014, 50(4), pp. 507–518 (дата звернення: 02.04.2021).

12. Kinoshenko, D., Mashtalir, V., Shlyakhov, V., Yegorova, E. Metrical properties of nested partitions for image retrieval. Machine Learning Techniques for Adaptive Multimedia Retrieval: Technologies, Applications, and Perspectives, 2010, pp. 18–49 (дата звернення: 08.04.2021).