НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

Створення веб-застосунків на основі технології Progressive Web Application

Текстова частина до курсової роботи

за спеціальністю «Комп'ютерні науки»

Керівник курсової роботи
Олецький О. В.

_____
(підпис)
"___" _____ 2022 р.

Виконав студент
Козодой М. О.
"___" _____ 2022 р.

Київ 2022

Зміст

Анотація

У цій курсовій роботі було досліджено технологію PWA, розглянуто її основні особливості та переваги у порівнянні зі звичайними веб та нативними застосунками. Також, як приклад для ілюстрації практичних застосувань розглянутої технології, було розроблено повноцінний PWA додаток “PWA Marketplace” на основі технологічного стеку MERN. “PWA Marketplace” – повноцінний marketplace (торгівельна платформа), що дозволяє зареєструватися, купувати продукти, для продавців – додавати власні товари, редагувати їх чи видаляти.

Вступ

Інтернет докорінно змінив світ починаючи від самого свого зародження на початку 80-х років XX століття. Люди отримали можливість знаходити інформацію, на пошук якої раніше витрачалися години, а то і дні, проведені в бібліотеці навпроти книжкових полиць, навіть не виходячи з дому, спілкуватися з друзями і рідними, що проживають у найвіддаленіших куточках планети, не купуючи конвертів, марок, не витрачаючи чорнил та без необхідності відвідувати найближче відділення пошти, буквально в декілька натискань кнопок клавіатури чи комп'ютерної мишки. Ці та багато інших благ, які принесла з собою глобальна мережа, спочатку були доступні виключно власникам персональних комп'ютерів.

Однак, все змінилося у 2007 році з представленням світові першої моделі IPhone, що знаменувало революцію у світі мобільних пристроїв, назавжди закріпивши у свідомості людства термін "smartphone", телефон, який позиціонується як повноцінний багатофункціональний пристрій на рівні з комп'ютерами, а не лише засіб зв'язку. З тих пір більшість інтернет користувачів надають перевагу власним телефонам та планшетам як засобам для взаємодії з їх улюбленими веб-ресурсами. Результати досліджень німецької компанії Statista стверджують, що кількість користувачів смартфонами зростає приблизно на 100 мільйонів за рік і досягне 6.8 мільярдів у 2023 році. Це яскраво ілюструє, наскільки важливими для екосистеми інтернету є користувачі смартфонів, а тому будь-яка компанія має робити все можливе, щоб надати їм якомога кращий досвід користування своїм додатком. Багато компаній обирають в якості рішення створення платформо-орієнтованих або, як їх ще називають, нативних застосунків.

Платформо-орієнтовані застосунки відомі тим, що вони надзвичайно багатофункціональні та надійні. Вони завжди присутні на робочому столі та на панелі задач, працюють незалежно від наявності з'єднання з інтернетом,

запускаються автономно, а не, скажімо, за допомогою браузера, здатні читати файли з локальної файлової системи та записувати їх туди ж, отримувати доступ до приладів, підключених через USB або Bluetooth, ба навіть взаємодіяти з різноманітними даними, які зберігаються на пристрої користувача, як-от: контакти, події в календарі. У цих додатках є можливість робити фотографії, керувати відтворенням пісень на фоні і багато іншого. Такі додатки відчуваються як частина пристрою, на якому вони запущені.

Однак, розробка нативного застосунку, наприклад мобільного, може бути досить виснажливим процесом, який потребує чимало коштів, часу, а інколи і пошуку нової додаткової команди розробників, якщо наявні працівники не спеціалізуються на мобільних додатках. Ба більше, найскладнішим же процесом є забезпечення ідентичного або хоча б максимально схожого користувацького досвіду усім своїм клієнтам на всіх платформах: будь то веб-версія додатку, повноцінна версія для комп’ютера чи нативний мобільний застосунок.

Для вирішення цих проблем у 2015 році компанією Google був представлений концепт, який стер кордон між нативними, настільними і веб-застосунками – PWA. Progressive Web Application – веб-застосунок, який забезпечує функціонал та користувацький досвід на рівні повноцінного нативного застосунку. Такий функціонал як спливаючі повідомлення, можливість працювати без з’єднання з інтернетом, завантажити саму програму з браузера, встановити на робочий стіл без необхідності навіть заходити в Apple Store чи Play Market та багато іншого роблять PWA надзвичайною та корисною технологією, що дозволяє розробляти повноцінні кросплатформенні застосунки, які ні в чому не поступаються нативним.

Основа цієї курсової роботи – дослідження технології PWA, її переваг. Також буде розроблено PWA застосунок, який називатиметься “PWA Marketplace”, що матиме в собі всі переваги веб і нативного застосунку.

1 Огляд основних типів застосунків

Намагаючись розробити кросплатформенний застосунок, який би об’єднував в собі найкращі риси усіх основних типів застосунків, які активно працюють із всесвітньою мережею, необхідно розглянути їх, виділивши ключові особливості.

1.1 Веб-застосунок

Веб-застосунок – це програмне забезпечення, яке служить для виконання задач через мережу інтернет, використовуючи веб-браузер та веб-технології. Яскравими прикладами є: онлайн магазини, електронні поштові скриньки, сайти банків, блоги, форуми. Більшість веб-застосунків використовують клієнт-серверну архітектуру, де сервер надає клієнту інформацію і сервіси: сервер обробляє запити клієнта та зберігає, модифікує чи видаляє дані. Веб-браузер використовується як проміжна ланка для реалізації їх взаємодії. Це дає можливість веб-застосункам працювати на різних платформах і незалежно від операційної системи. Небувала популярність, розвиток і поширення веб-застосунків почалися з еволюцією таких технологій як JavaScript, HTML (особливо після виходу HTML5), значну роль також відіграв Adobe Flash, який, втім, нещодавно перестав підтримуватися. Сильний вплив, як це не очевидно, мало і покращення власне інтернет технологій. Колись веб-сайти складалися із набору статичних сторінок з дуже обмеженими можливостями для інтерактивності, однак зараз їх імерсивність, функціонал, зручність і багатогранність нічим не поступаються повноцінним настільним застосункам. Навіть такі задачі як редагування фото, відео, текстових файлів тощо, які вимагають додаткових ресурсів і спеціального додаткового програмного забезпечення, можуть бути здійснені у веб-застосунку. Прихід респонсивного і адаптивного дизайну дав можливість відійти від стандартного “однакового”

вигляду застосунку на всіх дисплеях на користь різноманітних і цікавих підходів, що видозмінюють дизайн під кожен конкретний випадок.

1.2 Мобільний застосунок

Мобільний застосунок – це програмне забезпечення, розроблене спеціально для використання на малих портативних обчислювальних пристроях, як-от: смартфони, планшети. Мобільний застосунок зазвичай автономний, компактний, вузько направлений на виконання чітко визначених функцій. Спочатку такі застосунки були призначені для полегшення користування пристроєм, надаючи більш зрозумілий і зручний інтерфейс для використання можливостей смартфону, підтримки продуктивності користувача, наприклад, наявність доступу до своєї електронної поштової скриньки, створення подій і нагадувань в календарі, керування контактами тощо. Однак, апетити людей росли експоненційно, спричинивши виникнення і стрімкий розвиток індустрії мобільних ігор, GPS сервісів, доставок, таксі та багато інших, чисельність яких вимірюється тисячами екземплярів. Встановлюються вони зазвичай за допомогою спеціальних платформ-дистриб'юторів, якими керує власник мобільної операційної системи. Play Market від Google та App Store від Apple – дві основні платформи такого типу. Існують й інші незалежні платформи-дистриб'ютори мобільних застосунків, однак їх потужності й об'єми значно менші, ніж в двох вище зазначених представників.

В травні 2022 року Play Market налічував близько 2.6 мільйонів мобільних застосунків, роблячи його платформою-лідером, у той же час як у найближчого конкурента від Apple - 1.96 мільйонів. Більшість застосунків на цих платформах – безкоштовні, однак, трапляються і такі, які розповсюджуються шляхом одноразової або щомісячної плати. Більшість розробників таких мобільних програм заробляють шляхом розміщення всередині їх продукту рекламних інтеграцій. Враховуючи

шалений приріст мобільної аудиторії, кількість завантажень подібних застосунків росте надзвичайно швидко. У наш час саме мобільні застосунки є найбільш популярними у порівняння з іншими типами. Одними з причин є краща швидкодія, зручність, інтерактивність й продуктивність у порівнянні з, наприклад, веб-застосунками. Одними з ключових переваг мобільних застосунків є:

1) Спливаючі повідомлення. Мобільні застосунки здатні надсилати спливаючі повідомлення та оповіщення всередині додатку, що підсилює увагу та залученість користувача.
2) Спеціально оптимізований інтерфейс. Розробляючи інтерфейс першочергово беручи до уваги апаратні параметри і можливості чітко визначеної групи пристроїв, можливо створити простіший, зручніший та зрозуміліший інтерфейс.
3) Можливість використовувати системні ресурси.
4) Швидкодія. Цьому сприяє те, що мобільні застосунки мають можливість зберігати дані локально на пристрої.
5) Здатність працювати без інтернету. Хоча більшість основних функцій застосунку будуть недоступними у цьому режимі, він все ще працюватиме і збереже деякі можливості для взаємодії.
6) Безпека і надійність. Так як в основному мобільні застосунки поширюються через перевірені сервіси-дистриб’ютори з серйозною репутацією, вони є значно безпечнішими і надійнішими у порівнянні з веб-застосунками.

1.2.1 Нативний застосунок

Нативний застосунок – це застосунок, який створений під конкретний пристрій або платформу, що робить його несумісним з іншими платформами. Як

результат, додаток, написаний під операційну систему Android, не працюватиме на iOS і навпаки. Також нативні застосунки зазвичай розробляються на одній з конкретних мов програмування, їх різноманіття значно менше, ніж при розробці інших видів застосунків. Так для прикладу додатки для Android частіше всього розробляються на Java або Kotlin (який фактично є надбудовою над Java), а для iOS – на Objective-C. Нативні застосунки надають найкращий користувацький досвід, оскільки максимально використовують усі можливості пристрою, наприклад: мають доступ до камери, геолокації, компасу, дзвінків тощо. Однак, такий тип застосунків з точки зору бізнесу є не дуже ефективним і вигідним, адже є платформо залежним, а значить, що необхідно буде створити ще як мінімум декілька версій одного і того ж продукту для різних платформ, що вимагатиме додаткових вкладень, як матеріальних, так і часових.

Отож підсумовуючи, можна сказати, що нативні мобільні застосунки хоч і пропонують якісний користувацький досвід, покращену швидкодію й продуктивність, мають деякі значні мінуси:

1) Вони є доволі дорогими з точки зору розробки і підтримки. Це може стати великою перепоною для малого бізнесу.
2) Так як їх кількість на основних платформах-дистриб’юторах є надзвичайно великою, доведеться вкладатися в маркетинг, що понесе великі матеріальні витрати.
3) Часто вимагають багато місця в пам’яті, а також нерідко є доволі вибагливими щодо інших апаратних ресурсів. Це може стати перепоною для багатьох користувачів з недостатньо потужними пристроями.

2 Огляд технології PWA

Progressive Web Applications – це веб-застосунки, які надають користувацький досвід на рівні нативних додатків за допомогою стратегії прогресивного вдосконалення разом з API веб-браузера, що постійно покращується та розвивається. Такі додатки створюються на основі стандартних підходів до архітектури й вже існуючих основних веб-технологій, таких як: JavaScript, HTML, CSS.

Кожен PWA застосунок починається як набір звичайних сторінок, стаючи в процесі розробки захоплюючими високоякісними програмами , що забезпечують неповторний імерсивний користувацький досвід, зберігаючи при цьому всі переваги вебу, як наприклад доступність та низька вимогливість до системних ресурсів. PWA не потрібно завантажувати з платформ-дистриб'юторів, таких як Play Market чи App Store, адже вони можуть бути завантажені напряму з браузерів в один клік, як для персональних комп'ютерів, так і для смартфонів, планшетів тощо. Це величезна перевага для користувачів, яким більше до вподоби нативні застосунки, але які не бажають переходити на платформу-дистриб'ютор. Таким чином відбувається безшовний і швидкий перехід клієнта в ту версію додатку, взаємодія з яким йому більш комфортна, при цьому з точки зору функціоналу і користувацького досвіду він не втратить ні в чому. Більше того, наявність у PWA можливостей присилати спливаючі повідомлення, працювати в офлайн режимі, а також доступатися до функціоналу пристрою на рівні з нативними додатками, робить з PWA абсолютно унікальний і неповторний інструмент, який неодмінно покращить враження користувачів від взаємодії з продуктом.

PWA пропонує істотно якісніший за рівнем користувацький досвід ніж у звичайного традиційного веб-застосунку, одночасно будучи економічно вигідним, ефективним та зручним у розробці, що робить його ідеальним варіантом для

компаній. Варто лише сказати, що такі гіганти як Starbucks, Tinder, BMW, Uber, Debenhams та багато інших вже успішно використовують PWA у своїх проектах.

PWA не розробляються за допомогою однієї технології, мови чи фреймворку. PWA швидше репрезентує сучасну філософію побудови веб-застосунків, тягнучи за собою певні протестовані роками ефективні шаблони проектування, API та функції. Застосунок може називатися PWA, якщо він задовольняє вимоги та відповідає характеристикам наведеним нижче:

1) Прогресивний – застосунок функціонує на всіх пристроях, використовуючи техніки прогресивного вдосконалення, не зважаючи на конкретний браузер.
2) Респонсивний – інтерфейс адаптується до всіх без виключення пристроїв з максимально різними розмірами екранів, будь то персональний комп’ютер, телефон чи планшет.
3) Є можливість завантаження – має бути змога завантажити застосунок при чому без необхідності користуватися платформами-дистриб’юторами.
4) Незалежний від підключення до мережі – застосунок має працювати і у випадку повної відсутності з’єднання з інтернетом, і у випадку його нестабільності.
5) Безпечний – захищений за допомогою HTTPS та TLS задля запобігання втручання в контент чи викрадення інформації.
6) Відкритий – додаток легко знаходиться за допомогою пошукових рушіїв.
7) Актуальний – постійно й автоматично оновлюється для підтримки актуальної версії за допомогою service worker’а.
8) Подібний до нативного застосунку – емулює таку ж поведінку і забезпечує ідентичний з ним користувацький досвід.

Варто зазначити, що для оцінки більшості з цих параметрів можна скористатися інструментом з відкритим вихідним кодом від Google, який носить назву

Lighthouse. В нових версіях браузеру Google Chrome він доступний на одній із вкладок в режимі розробника.

2.1 Ключові технології при розробці PWA

Структура PWA застосунку хоч і мало відрізняється від звичайного веб-додатку, але все ж містить в собі декілька ключових відмінностей у вигляді різноманітних сервісних компонентів. Саме вони і дають змогу PWA істотно відрізнятися у своїй роботі від інших традиційних типів додатків. Основу таких компонент складають service worker'и та web app manifest. Нижче детально розглядається кожна із них.

2.1.1 Service Worker

Service worker – один із найважливіших і фундаментальних елементів, який забезпечує основні необхідні функції для PWA застосунку, такі як спливаючі повідомлення, швидке завантаження, робота в офлайн режимі, фонову синхронізацію та багато іншого. Service worker представляє собою звичайний клієнтський JavaScript код, який виконується у фоновому режимі, відділений від основного потоку і відповідає на взаємодію користувача із застосунком, а також здатний перехоплювати мережеві запити. Оскільки service worker працює не в основному потоці для виконання JavaScript коду сторінки, він не має прямого доступу до DOM, але все одно здатний взаємодіяти зі сторінкою шляхом використання інтерфейсу postMessage. Більше того, для виконання задач service worker використовує Promise-based підхід, що робить всі його процеси повністю

асинхронними. Це дозволяє не переривати й не призупиняти робочий процес, працюючи з різними контекстами.

Service worker багато в чому схожий на proxy сервер, що знаходиться між веб-застосунком, браузером і мережею (якщо вона доступна). Він здатний перехоплювати мережеві запити, записувати дані в кеш або діставати їх звідти.

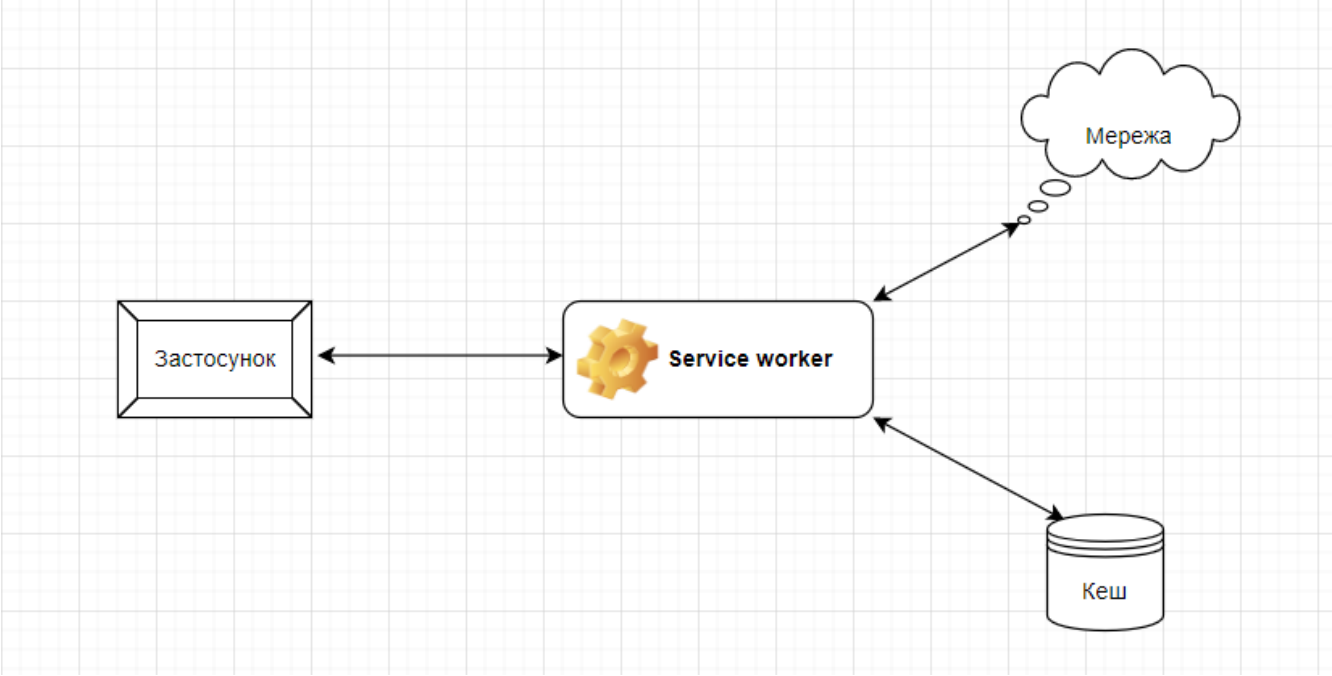


Рисунок 3.1 Service worker у якості proxy

Перехоплюючи запит, надісланий з клієнта, service worker використовує підхід “offline first” або, як його ще називають, “cache first”. Це означає, що вищим пріоритетом є аналіз закешованих даних і видобування їх звідти, якщо необхідна інформація, за якою робився запит, там присутня. Якщо ж в кеші необхідних даних не знайшлося, запит буде спрямовано вже до сервера, а отримана у відповідь інформація буде успішно записана до кешу для можливого використання у майбутньому. Отже, саме за допомогою підходу “cache first” service worker дозволяє PWA застосунку працювати офлайн, а також значно пришвидшити завантаження, оскільки більшість необхідних ресурсів будуть відразу отримані з кешу.

Важливий момент полягає в тому, що service worker’и можна застосовувати тільки у веб-додатках, які використовують безпечне з’єднання, тобто працюють на HTTPS. Це зроблено для того, щоб уникнути атак типу “man in the middle” (людина посередині), оскільки без цих запобіжних заходів зловмисник міг би легко перехопити запити, сфабрикувати і змінити інформацію, яку вони надають, нанісши колосальних збитків і шкоди.

Service worker розглядається як прогресивне покращення, яке додається до веб-застосунку у випадку, коли всі необхідні умови для його роботи виконані. Застосування service worker’ів значно покращує користувацький досвід і загальну швидкодію застосунків. Оскільки service worker’и дотримуються принципів прогресивних покращень, то тільки сучасні і сумісні з ними браузери можуть використовувати повний спектр їх функцій та можливостей.

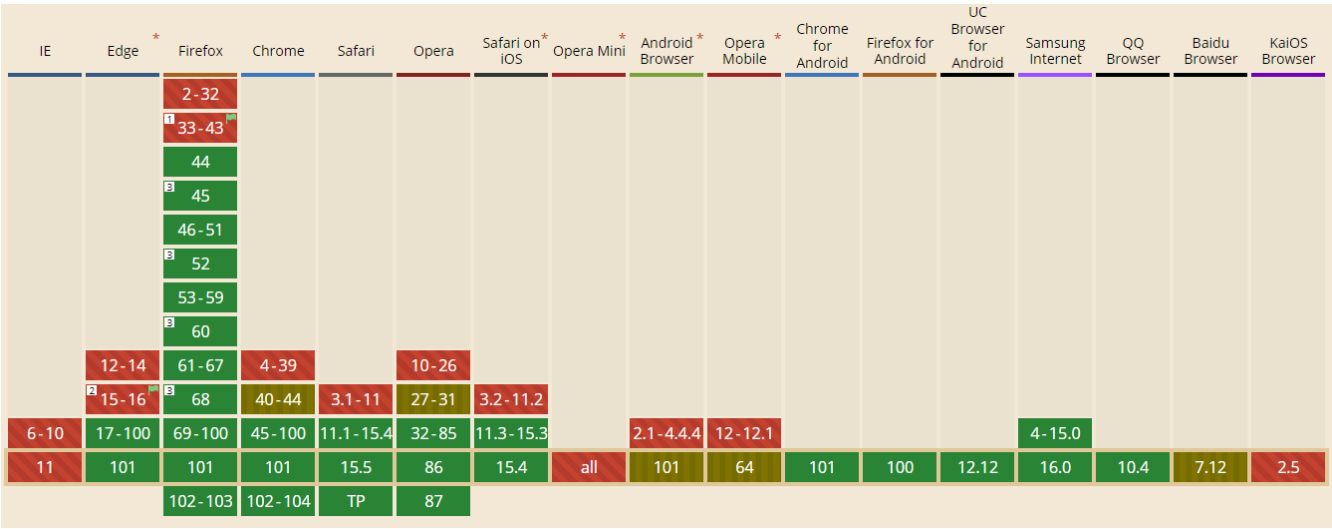


Рисунок 3.2 Підтримка service worker різними браузерами

Можна побачити, що service worker’и підтримують практично всі сучасні версії браузерів.

2.1.1.1 Життєвий цикл service worker'а

Service worker – це event-driven (оснований на подіях) скрипт, життєвий цикл якого сильно відрізняється від життєвого циклу веб-сторінки. Його життєвий цикл можна розділити на три основних етапи: реєстрація, встановлення, активація.

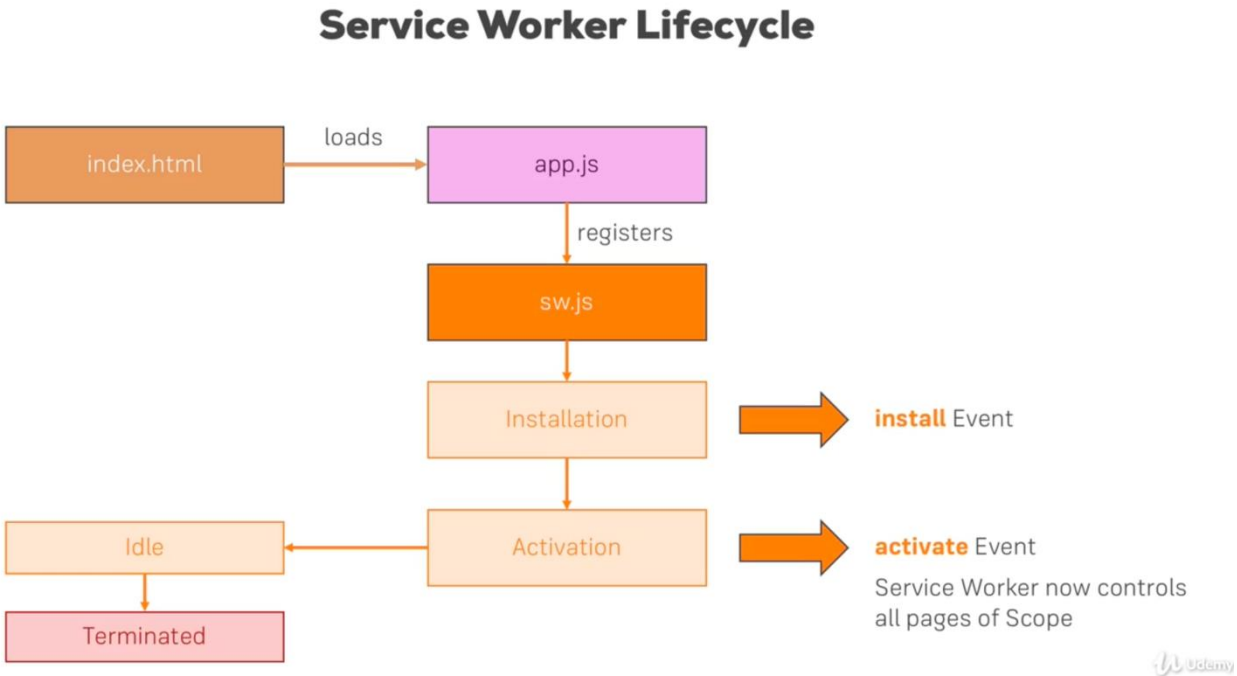


Рисунок 3.3 Життєвий цикл service worker'а

2.1.1.1.1 Реєстрація

Перед тим, як service worker почне контролювати сторінку, його необхідно зареєструвати. Це означає, що під час першого запуску всі мережеві запити будуть надсилатися напряму до сервера, оскільки service worker ще не почав їх контролювати. Реєстрація service worker'а дозволить браузеру знайти необхідний service worker у файлах проекту та почати його встановлення фоновим процесом.

```
22      if ("serviceWorker" in navigator) {
23        window.addEventListener("load", () => {
24          navigator.serviceWorker
25            .register("./service-worker.js")
26            .then((reg) => {
27              console.log("Registration was successful: ", reg.scope);
28            })
29            .catch((err) => {
30              console.log("Something went wrong: ", err);
31            });
32        });
33      }
```

Рисунок 3.4 Реєстрація service worker'a

Після перевірки, чи підтримує браузер Service Worker API, PWA може зареєструвати цей service worker, код якого знаходиться за вказаним шляхом. Помилка виконання якоїсь із частин наведеного вище коду призведе до повної відміни всього процесу реєстрації і виведення повідомлення з причиною помилки в термінал. Якщо ж процес реєстрації відбувся успішно, про це теж буде повідомлено в терміналі, додатково буде вказана область видимості цього service worker’а, а також запуститься процес його встановлення. Перевірити успішність реєстрації можна і за допомогою інструментів розробника у браузері, достатньо лише перейти до вкладки “Застосунок” і у панелі зліва обрати розділ “Service Workers”.

Важливо завжди брати до уваги область видимості service worker’а, яка визначається директорією, у якій знаходиться його код. Так service worker, який знаходиться у директорії *pwa-marketplace/views/service-worker.js* зможе контролювати компоненти з директорії views і нижче, наприклад *pwa-marketplace/views/components*. Дозволяється мати лише один service worker для певної області видимості.

2.1.1.1.2 Встановлення

У разі, якщо процес реєстрації був успішним, починається встановлення service worker’a. Варто зазначити, що встановлення відбувається лише один раз. Встановлення відбувається фоном, без необхідності отримати дозвіл користувача. При цьому це відбувається просто під час відкриття сторінки у браузері, тобто не потрібно навіть завантажувати застосунок на власний пристрій. Service worker API підтримується навіть на тих браузерах, які не підтримують опцію встановлення PWA на пристрій.

```
const CACHE_NAME = "version-1";
const urlsToCache =
    [
        'index.html',
        'offline.html',
        "icons/android-chrome-192x192.png",
        "icons/favicon.ico"
    ]

const self = this;

//---Install SW
self.addEventListener('install', (event) => {
    event.waitUntil(caches.open(CACHE_NAME).then((cache) => {
        console.log("Opened cache");
        return cache.addAll(urlsToCache)
    }))
})
```

Рисунок 3.5 Встановлення service worker'a

Реєстрація і встановлення service worker’a хоча і взаємопов’язані, але представляють собою абсолютно різні події. Реєстрація відбувається, коли застосунок звертається до service worker’a, викликаючи метод register(), як було продемонстровано вище. Встановлення відбувається, коли зареєстрований service

worker існує, може бути розпаршеним як JavaScript, а також були відсутні будь-які помилки під час його першого виконання. Під час встановлення завантажуються та кешуються всі необхідні статичні ресурси веб-застосунку. Коли всі статичні ресурси закешовані, а процес встановлення успішно завершився, відбувається активація service worker'а. Якщо під час кешування виникає помилка, встановлення переривається і service worker не зможе бути активованим. У такому випадку процес встановлення буде перезапущено.

2.1.1.1.3 Активація

Після успішного встановлення новий service worker буде активований тільки тоді, коли існуючі старі service worker'и перестануть контролювати будь-яких клієнтів. Такий стан називається станом очікування, саме завдяки ньому браузер гарантує існування тільки однієї версії service worker'а. Користувачу необхідно закрити всі вікна і вкладки застосунку, які використовують поточний service worker. Тоді старі service worker'и будуть видалені, і тільки після цього нові вступлять в силу.

```
//---Activate the SW
self.addEventListener('activate', (event) => {
    const cacheWhitelist = [];
    cacheWhitelist.push(CACHE_NAME);

    event.waitUntil(
        caches.keys().then((cacheNames) => Promise.all(
            cacheNames.map((cacheName) => {
                if (!cacheWhitelist.includes(cacheName)) {
                    return caches.delete(cacheName);
                }
            })
        ))

    )
});
```

Рисунок 3.6 Активація service worker'a

Активований service worker починає відповідати на взаємодію користувача із інтерфейсом веб-застосунку, керуючи подіями типів fetch, push, sync та іншими.

```
//---Listen for requests
self.addEventListener('fetch', (event) => {
    event.respondWith(
        caches.match(event.request)
            .then(() => {
                return fetch(event.request)
                    .catch(() => caches.match('offline.html'))
            })
    )
});
```

Рисунок 3.7 Обробка service worker'ом події типу fetch

Також service worker'и здатні виконувати такі речі:

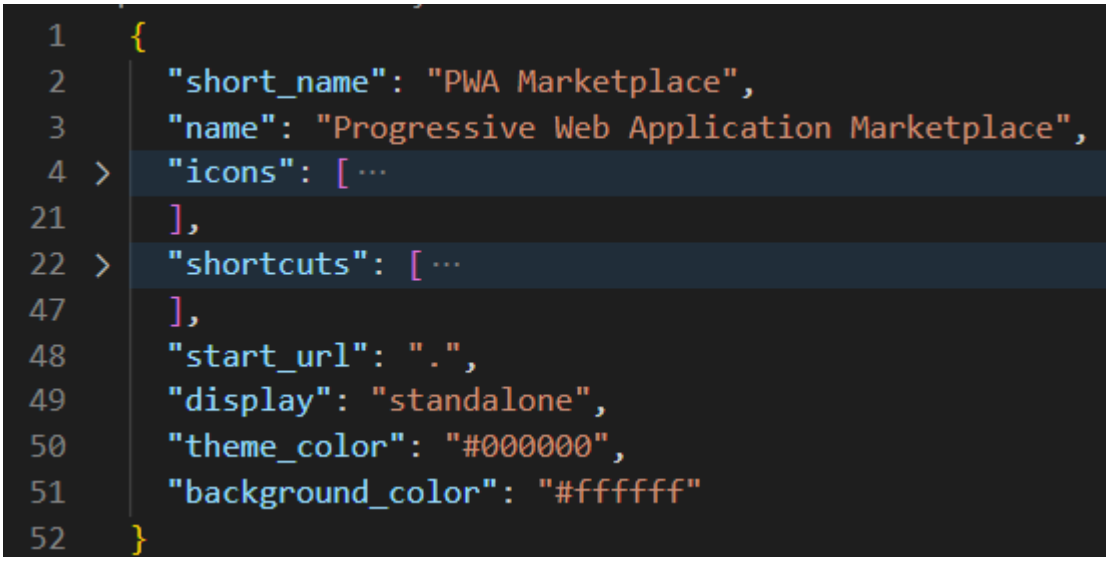
1) Фонова синхронізація даних.
2) Відповідати на запити з різних джерел
3) Централізовано зберігати, оновлювати й надавати усім своїм клієнтам великі й дорогі для обробки дані, наприклад геолокацію та гіроскоп. Тобто усі клієнти можуть спільно користуватися подібними ресурсами.
4) Компіляція та керування залежностями на боці клієнта для таких технологій як CoffeeScript, less, CJS/AMD тощо.
5) Гачки (hooks) для фонових сервісів.
6) Покращення швидкодії та продуктивності, наприклад, наперед завантажувати ресурси, які теоретично знадобляться користувачу у майбутньому, скажімо, наступні три фото у галереї, яку він прямо зараз листає.

Ці та безліч інших можливостей і функцій уже наявні у service worker'ів, але вони все ще продовжують надзвичайно активно розвиватися і немає причин сумніватися, що у майбутньому з'являтиметься ще більше нових і дивовижних речей, за допомогою яких вони дозволятимуть ще сильніше стирати кордон між нативними і веб-застосунками.

2.1.2 Web App Manifest

Web App Manifest – це ще одна із фундаментальних технологій для створення PWA.

Web App Manifest являє собою JSON документ, в якому визначається зовнішній вигляд і деякий функціонал PWA. У ньому зберігається інформація така як ім'я застосунку, початковий URL, опис, іконки, тематичні кольори, скорочення тощо. Цей JSON надає інформацію браузеру про те, як має поводитися застосунок, а також допомагає під час завантаження його на пристрій.



```
{
  "short_name": "PWA Marketplace",
  "name": "Progressive Web Application Marketplace",
  "icons": [...
  ],
  "shortcuts": [...
  ],
  "start_url": ".",
  "display": "standalone",
  "theme_color": "#000000",
  "background_color": "#ffffff"
}
```

Рисунок 3.8 Приклад Web App Manifest'у

Файл маніфесту складається із інформації про ім'я застосунку, початковий URL, який має відкриватися при його запуску, іконки, які мають використовуватися (у тому числі для створення ярлика застосунку у випадку завантаження його на пристрій), основні тематичні і фонові кольори і багато іншого.

Кожна частинка маніфесту надає спеціальну інформацію для конфігурування зовнішнього вигляду і спеціальних аспектів поведінки PWA застосунку. Браузер розпізнає Web Appl Manifest, підключений в head як

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

і повідомляє користувачів про можливість завантажити застосунок на свій пристрій.

Рисунок 3.9 Браузер повідомляє про наявність можливості завантажити застосунок

Об’єднання Web App Manifest і Add To Home Screen (додати на домашній екран) функціоналу, який наявний в сучасних браузерах, дозволяє повноцінно завантажити PWA на пристрій, повністю реплікуючи поведінку і функціонал нативного застосунку. Однак, щоб можливість завантаження з’явилася, необхідно, щоб додаток відповідав наступним критеріям:

1) Застосунок має працювати на HTTPS.
2) У файлах застосунку має бути наявний коректний Web App Manifest, підключений в head сторінки.
3) Застосунок повинен мати активний зареєстрований service worker.

4) Обов’язково мають бути іконки для створення ярлика на домашньому екрані пристрою.

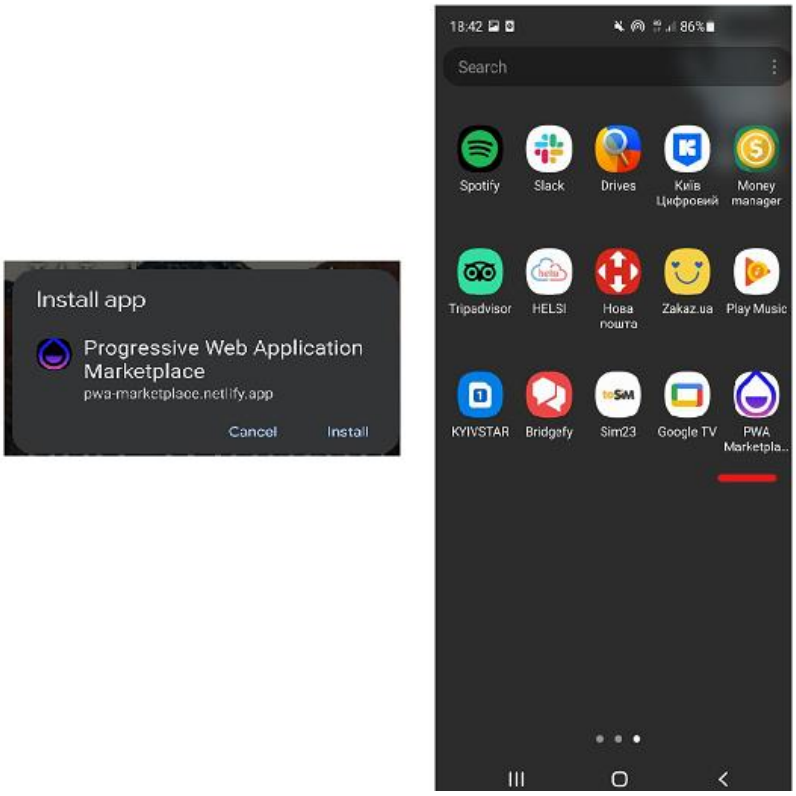


Рисунок 3.10 Встановлений на мобільний пристрій PWA

При натисканні на повідомлення з пропозицією від браузера додати застосунок на головний екран, з’явиться можливість завантажити його на свій пристрій.

2.2 Переваги PWA

Повноцінний PWA використовує одні із найпотужніших веб-технологій, доступних на сьогоднішній день, маючи виняткові переваги у порівнянні з іншими типами застосунків як для розробників, так і для користувачів. PWA стають все популярнішими, оскільки вони не просто забезпечують функціонал і

користувацький досвід на рівні з нативними застосунками, а в дечому навіть показують себе краще за них.

2.2.1 Незалежність від мережі

За допомогою service worker'ів PWA можуть кешувати більшість необхідних їм даних і використовувати їх навіть за умови нестабільності або навіть повної відсутності інтернет з'єднання.

2.2.2 Покращена швидкодія та продуктивність

Безліч досліджень говорять, що більша частина користувачів покине сайт, якщо він буде вантажитися більше трьох секунд. Завдяки service worker'ам, які кешують більшість статичних ресурсів, та створенню app shell (оболонки застосунку, яка складається з мінімального набору HTML, CSS та JavaScript, необхідних для побудови номінального користувацького інтерфейсу) вдається домогтися неймовірного покращення швидкодії і продуктивності.

2.2.3 Залучення користувачів

У традиційних веб-застосунків надзвичайна мала кількість засобів по збереженню користувацької уваги й повторного його залучення у порівнянні з PWA. З використанням service worker'ів для PWA доступні Push і Notifications API, що дозволяють надсилати оповіщення користувачам, повторно залучаючи і тримаючи їх увагу на високому рівні. PWA може надсилати новини про оновлення застосунку, доступ на порталі нового контенту і загалом вміст повідомлень може

бути який завгодно, а що найголовніше – користувач їх обов'язково отримає, адже для цього навіть не потрібно, щоб застосунок був у цей момент запущений.

2.2.4 Легше встановлення та підтримка

PWA є незалежними від платформ-дистриб'юторів, це означає, що вони можуть бути завантажені без необхідності користуватися Google Play Market чи Apple App Store. Можливість завантаження за допомогою браузера буквально в один клік робить цей процес значно зручнішим для користувачів, які надають перевагу нативним застосункам, якщо в продукту є такі версії. Ба більше, порівнюючи з нативними, PWA застосунки займають значно менше місця в пам'яті, використовують менше системних ресурсів та сильно економлять кошти і час компанії, які інакше були б витрачені на розробку і підтримку різних версій свого продукту для декількох платформ.

3 Розробка власного застосунку

У цій секції будуть описані технології і деталі розробки застосунку "PWA Marketplace", який був створений як приклад повноцінного PWA у рамках цієї курсової роботи.

3.1 Огляд використаних технологій та інструментів

Застосунок "PWA Marketplace" розроблявся на основі технологічного стеку MERN, що розшифровується як MongoDB, ExpressJS, ReactJS, NodeJS.

3.1.1 MongoDB

Будь-який full-stack застосунок потребує бази даних для збереження і пізнішого доступу до певної інформації, яка використовується додатком. База даних – це структурована колекція даних, до якої можна доступатися, підтримувати її та модифікувати. MongoDB – це кросплатформенна система управління нереляційними базами даних з відкритим вихідним кодом, яка використовує гнучкі документи замість таблиць і рядків для обробки і зберігання різних форм даних. Розробляється і підтримується компанією MongoDB Inc. MongoDB зберігає дані у вигляді моделі на основі документів BSON (Binary JSON), що робить процеси отримання і маніпулювання даними легкими і ефективними. Документ у MongoDB – це структура даних, що представляє собою набір пар поле-значення, де значеннями полів окрім звичайних традиційних типів даних можуть виступати інші документи, масиви та масиви документів. Група документів називається колекцією. Такий тип моделі даних дозволяє легко представляти складні структури та ієрархічні зв’язки.

Mongoose – це ODM (Object Document Mapper – об’єктно-документний віддзеркалювач) для MongoDB. Він керує зв’язками між даними і дає можливість визначити об’єкти зі строго-типізованою схемою, яка відповідає документам в MongoDB. Існують й інші ODM для MongoDB, наприклад Doctrine, MongoLink, Mandango. Mongoose додає ще один шар абстракції над MongoDB.

Mongoose виступає посередником між MongoDB і сервером. Він полегшує взаємодію з MongoDB та додає багато корисних речей, як-от валідацію, побудову запитів і приведення типів.

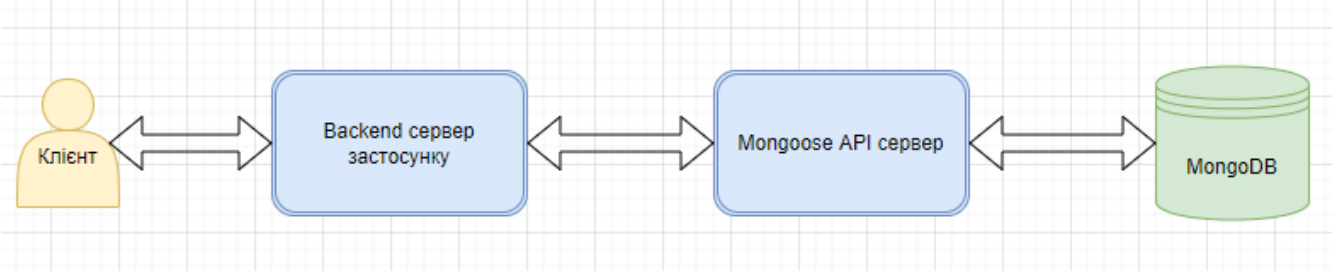


Рисунок 4.1 Схема застосунку з використанням Mongoose

3.1.2 Node.js

Back-end – ще одна невід'ємна частина кожного веб-застосунку. Він є так званим data-access (з доступом до даних) рівнем, який відповідає за бізнес логіку застосунку та збереження даних.

Node.js – це кросплатформенне середовище з відкритим вихідним кодом для виконання високопродуктивних мережевих застосунків, які написані мовою JavaScript. Головна особливість цього середовище полягає в тому, що воно дозволяє виконувати JavaScript код не в браузері, а на окремому сервері, а потім відправляти користувачеві результати його роботи. Node.js – це серверна платформа, заснована на рушії Google Chrome's V8 JavaScript. Головна парадигма Node.js звучить як "JavaScript всюди", що має на увазі використання мови JavaScript як на серверній, так і на клієнтській частині застосунку, що дозволяє значно пришвидшити розробку продукту та потребує меншої кількості вузько направлених спеціалістів.

Більше того, Node.js є повністю асинхронним, event-driven (заснованим на подіях) та легко розширюваним, що дозволяє обробляти декілька вхідних/вихідних запитів одночасно. Також Node.js надає npm (node package manager) - один із найзручніших і найкращих пакетних менеджерів на ринку. Він ефективно керує залежностями між сторонніми модулями та бібліотеками, використання яких значно прискорює та полегшує розробку будь-якого застосунку.

3.1.3 Express.js

Express.js – це зручний, швидкий і мінімалістичний фреймворк для Node.js, який використовується для розробки API. Express.js є дуже гнучким і легким модулем для середовища Node.js. Загальноприйнято вважати його стандартним вибором поміж серверних фреймворків для побудови back-end'у на Node.js через його величезну популярність і широке використання.

Ще декілька інших сторонніх бібліотек і модулів були використані для побудови back-end'у застосунку "PWA Marketplace", основними з них є:

1) bcrypt.js – дозволяє безпечно зберігати паролі у базі даних шляхом їх хешування і перетворення у нерозбірливий рядок.
2) dotenv – для визначення environmental variables (змінних середовища), які необхідні для деплою застосунку.
3) Jsonwebtoken – для реалізації автентифікації

3.1.4 React.js

Front-end – це презентаційний шар застосунку, який відповідає за користувацький досвід. Так як саме з цим шаром прямо взаємодіють користувачі, він є чи не найважливішим у всьому застосунку.

React.js – це гнучка, декларативна та ефективна JavaScript бібліотека для створення інтерактивного користувацького інтерфейсу. Це бібліотека з відкритим вихідним кодом, яка була випущена компанією Facebook у 2013 році та продовжує підтримуватися нею. React.js відповідальний за view layer (шар зовнішнього вигляду) застосунку.

Реакт декларативний. Він дозволяє легко розробляти інтерактивний користувацький інтерфейс. Варто лише створити декілька views (різні варіації зовнішнього вигляду сторінки) для кожного можливого стану застосунку, а Реакт самостійно оновить і відрендерить тільки необхідні компоненти, дані в яких було змінено. Декларативний стиль робить код більш читабельним і полегшує процес налагодження (усунення помилок).

Реакт компоненто-орієнтований. Він дозволяє створювати незалежні програмні компоненти, які керують власним станом і даними. Поєднуючи різні компоненти між собою, вдається легко й ефективно створювати навіть найскладніший і найдивакуватіший користувацький інтерфейс. Оскільки компоненти повністю написані на JavaScript (на відміну від використання шаблонів, як, наприклад, у Angular), можна легко передавати навіть найбільш складні структури даних через увесь додаток, зберігаючи стани компонент і застосунку в цілому, ізольовано від DOM. Саме його компоненто-орієнтована архітектура зробила React.js однією з найпопулярніших JavaScript бібліотек з активною спільнотою, яка постійно покращує та доповнює її.

Однією із найпотужніших технологій Реакту є так званий віртуальний DOM. Віртуальний DOM – це in-memory (що зберігається у пам’яті) структура даних, яка кешується Реактом і являє собою копію реального DOM. Після зміни стану компонент, у наслідок яких буде модифіковано реальний DOM, React.js змінює не його, а саме віртуальний DOM, а потім порівнює поточний стан реального DOM та віртуального і, визначивши різницю між ними, оновлює в реальному DOM тільки ті елементи, стан яких було модифіковано. Цей процес є значно швидшим, аніж оновлення всього DOM, оскільки це є надзвичайно ресурсозатратною операцією. Використання віртуального DOM робить швидкодію та продуктивність React.js кращими, ніж в його конкурентів.

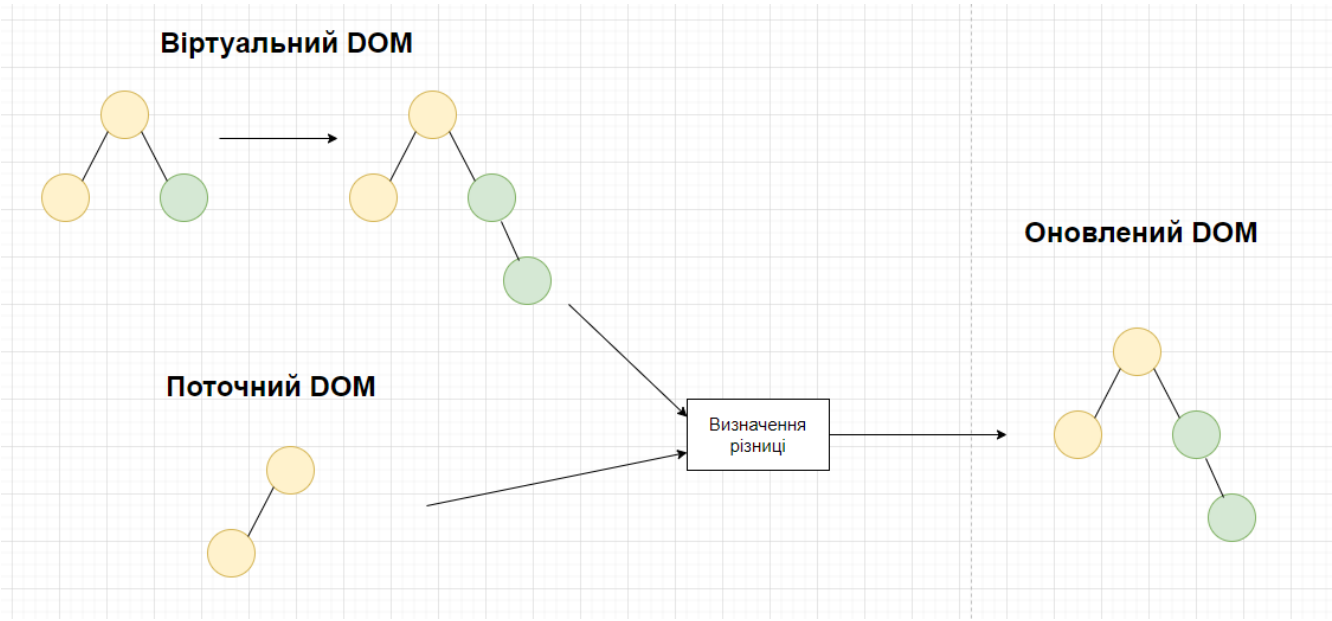


Рисунок 4.2 Ілюстрація роботи віртуального DOM

React.js використовує JSX (JavaScript XML) як мову для написання компонентів. Він дозволяє поєднувати HTML та JavaScript і створювати нові елементи в DOM без необхідності використовувати такі методи, як наприклад createElement() і appendChild(). Хоча використання JSX значно полегшує розробку застосунку, він не є обов'язковим: можна писати код і на звичайному JavaScript.

```
const HelloMessage = (props) => {
    return (
        <div>Hello {props.name}</div>
    );

}

root.render(<HelloMessage name="Maxim" />);
```

Рисунок 4.3 Створення елементу з використанням JSX

```
const helloMessage = React.createElement('div', {}, 'Hello, there is no JSX here');

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(helloMessage);
```

Рисунок 4.4 Створення елементу без використання JSX

JSX допомагає легко писати компоненти у декларативному стилі. JSX має XML/HTML подібний синтаксис, який розширює ECMAScript, що робить можливим співіснування HTML та JavaScript. Це дозволяє поєднувати верстку та логіку компоненти в рамках одного файлу. Увесь JSX код застосунку пізніше компілюється у звичайний JavaScript за допомогою Babel, що робить його сумісним з усіма існуючими браузерами.

Create React App (CRA) – інструмент від Facebook, який дозволяє швидко створити застосунок на основі React.js з мінімальним необхідним набором технологій, таких як Babel, Webpack, ESLint, Jest, а також створює development та production конфігурації для середовища розробки. Цей інструмент дозволяє створити повноцінний робочий застосунок на основі React.js за лічені секунди.

```
npx create-react-app my-app
```

Рисунок 4.5 Команда для швидкого створення застосунку на основі React.js

Компоненти є основними будівельними блоками для всього React.js застосунку. Вони еквівалентні функціям у JavaScript і дозволяють поділити користувацький інтерфейс на окремі, незалежні, reusable (такі, що можна використовувати повторно) частини. Всього існують компоненти двох типів: functional (функціональні) та class-based (основані на класах) [3]. Після появи хуків, які розширили можливості функціональних компонент, вони набули більшої популярності, ніж class-based компоненти. [4]. Оскільки React.js - це всього лише

бібліотека, а не фреймворк, то для розробки повноцінного застосунку в рамках створення проекту були використані й деякі інші бібліотеки.

3.1.5 Redux

Redux – це бібліотека з відкритим вихідним кодом для централізованого контролю станом даних застосунку на клієнтському боці. Redux необхідний для створення сховищ, де будуть зберігатися стани визначених змінних застосунку. Він створює процеси і процедури для взаємодії зі сховищем, щоб уникнути неконтрольований доступ і модифікацію даних різними компонентами. Redux складається з трьох основних будівельних блоків: actions (дії), reducers (редуктори) і store (сховище).

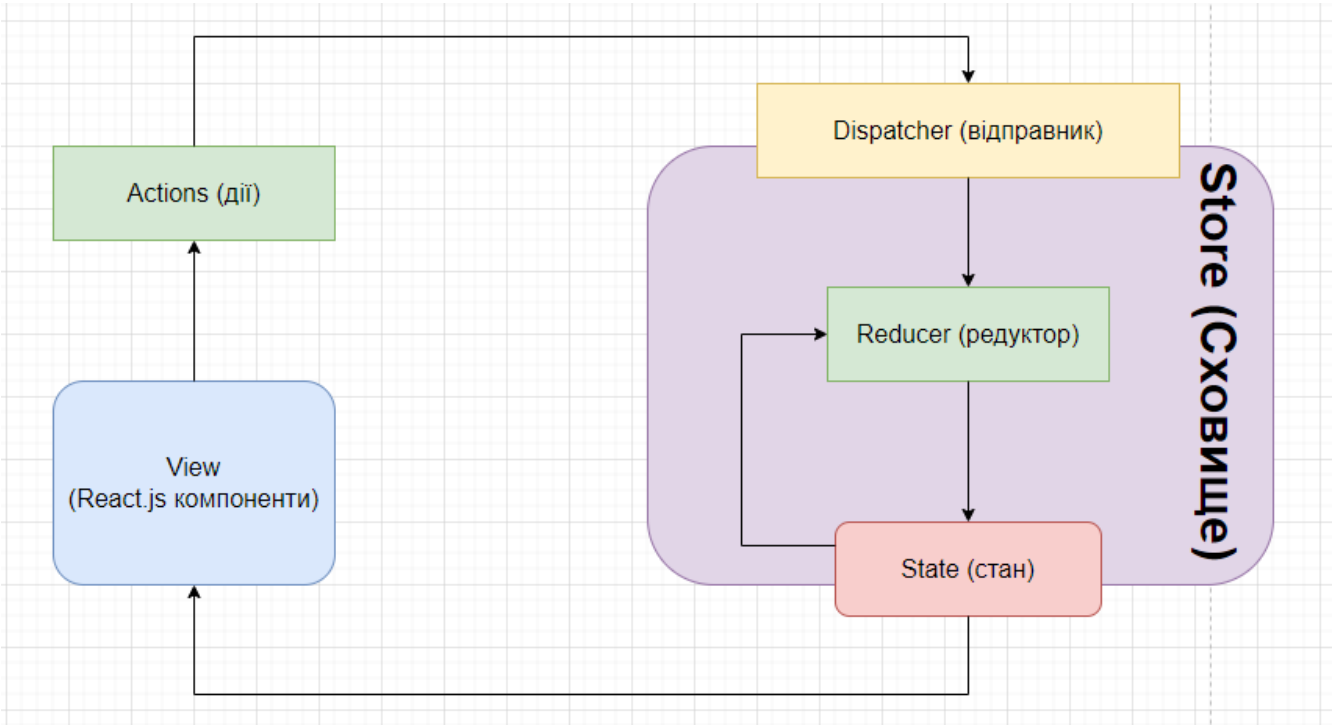


Рисунок 4.6 Схема потоків даних в Redux

Redux зберігає стан в унікальному immutable (незмінному) об’єкті, який називають станом в рамках глобального сховища. Стани застосунку підтримуються Redux’ом незмінними – це означає, що вони не можуть бути модифікованими, а при необхідності будуть відтвореними на основі образу попереднього стану із доданими бажаними змінами. Сховище виступає єдиним і надійним джерелом визначених станів для застосунку та надає доступ до них всім компонентам.

Action creators (створювачі подій) викликаються щоразу, коли користувач у рамках певної компоненти хоче взаємодіяти зі станом, збереженим в Redux сховищі. Об’єкт події складається із типу події та може містити дані, якими вона хоче оновити поточний стан.

```
1   // action types
2   export const ADD_TO_CART = 'cart/addToCart';
3   export const REMOVE_FROM_CART = 'cart/removeFromCart';
4   export const INCREASE_QUANTITY = 'cart/increaseQuantity';
5   export const DECREASE_QUANTITY = 'cart/decreaseQuantity';
6   export const UPDATE_PRODUCT = 'cart/updateProduct';
7   export const CLEAR_CART = 'cart/clearCart'
8
9   // action creators
10  export const addToCart = (product) => ({
11      type: ADD_TO_CART,
12      payload: product
13  })
14
15  export const removeFromCart = (product) => ({
16      type: REMOVE_FROM_CART,
17      payload: product
18  })
```

Рисунок 4.7 Приклад деяких подій у Redux

Далі за допомогою відправника подія буде передана до редуктора, який і виконає операції зі станом. Функція-редуктор приймає два аргументи: сам поточний стан і об’єкт події. Редуктори написані як чисті функції, які визначають, як стан має змінитися у відповідь на певну подію. Спираючись на попереднє значення стану і передану подію, редуктор вираховує нове значення стану й оновлює його. Все це відбувається всередині Redux сховища, яке певною мірою може бути охарактеризоване як своєрідна централізована база даних, що працює на клієнтській частині застосунку. Після того, як компонента отримує нове значення стану, вона ререндериться.

```
export default function cart(state = initialState, action) {
    switch (action.type) {
        case ADD_TO_CART: {
            const items = [...state.items];
            if (!(state.items.find(product =>
                ((product.id === action.payload.id))
            ))) {
                items.push(action.payload)
            } else {
                return increaseQuantity(state, action.payload)
            }
            saveLocalState(storageKey, items);
            return { ...state, items };
        }
        case REMOVE_FROM_CART: {
            const filteredItems = state.items.filter(product => product.id !== action.payload.id
            );
            saveLocalState(storageKey, filteredItems);
            return { ...state, items: filteredItems };
        }
```

Рисунок 4.8 Приклад реакції деякого редуктора на певні події

3.1.6 Bootstrap

Bootstrap – це безкоштовний CSS фреймворк з відкритим вихідним кодом, який надає безліч попередньо стилізованих компонентів, таких як кнопки, поля для введення, каруселі, діалогові вікна та багато іншого. Окрім цього, в ньому є надзвичайно зручна сітка для побудови адаптивної та респонсивної верстки.

3.1.7 Axios

Axios – це promise-based (заснована на Promise'ах) бібліотека для HTTP запитів із Node.js або XMLHttpRequests із браузерів. Це чудова зручніша альтернатива Fetch API, який використовується у Реакті за замовчуванням.

3.2 Створення застосунку PWA Marketplace

PWA Marketplace – це прогресивний веб-застосунок, створений на основі технологічного стеку MERN. Основна ціль – на практиці продемонструвати переваги PWA, розробивши повноцінний додаток, який виступатиме у ролі маркетплейсу.

3.2.1 Архітектура застосунку

Архітектура застосунку представляє собою шаблони та техніки, які використовувалися для проектування та подальшої розробки застосунку. Для даного проекту був обраний стек MERN, оскільки він є одним із найкращих рішень для побудови веб-застосунку. Була створена тришарова архітектура, яка складається з back-end'у, front-end'у та рівня даних. Взаємодія браузеру і серверу застосунку відбувається за допомогою REST API. Back-end або бізнес логіка виступає посередником між front-end'ом та шаром даних застосунку. Для реалізації серверу було обрано Node.js як середовище виконання разом із фреймворком Express.js для розробки API. Із баз даних була обрана MongoDB, яка розгортається та керується за допомогою хмарного сервісу MongoDB Atlas. Взаємодія із базою даних, яка включає в себе запис, формування запитів і видобування інформації

реалізована за допомогою Mongoose в ролі ODM. Автентифікація була реалізована за допомогою JWT.

Клієнтська частина або front-end була створена з використанням React.js. Redux використовується для керування глобальними станами застосунку, React-router – для управління route'ами (маршрутами). Axios – для запитів до сервера. Bootstrap, Fontawesome, MaterialUI – для попередньо стилізованих компонент і зручних засобів реалізації респонсивного та адаптивного дизайну.

Розгортався застосунок за допомогою платформ із надання хмарних сервісів Heroku (для back-end'у) і Netlify (для front-end'у). Сам застосунок доступний за адресою https://pwa-marketplace.netlify.app/ .

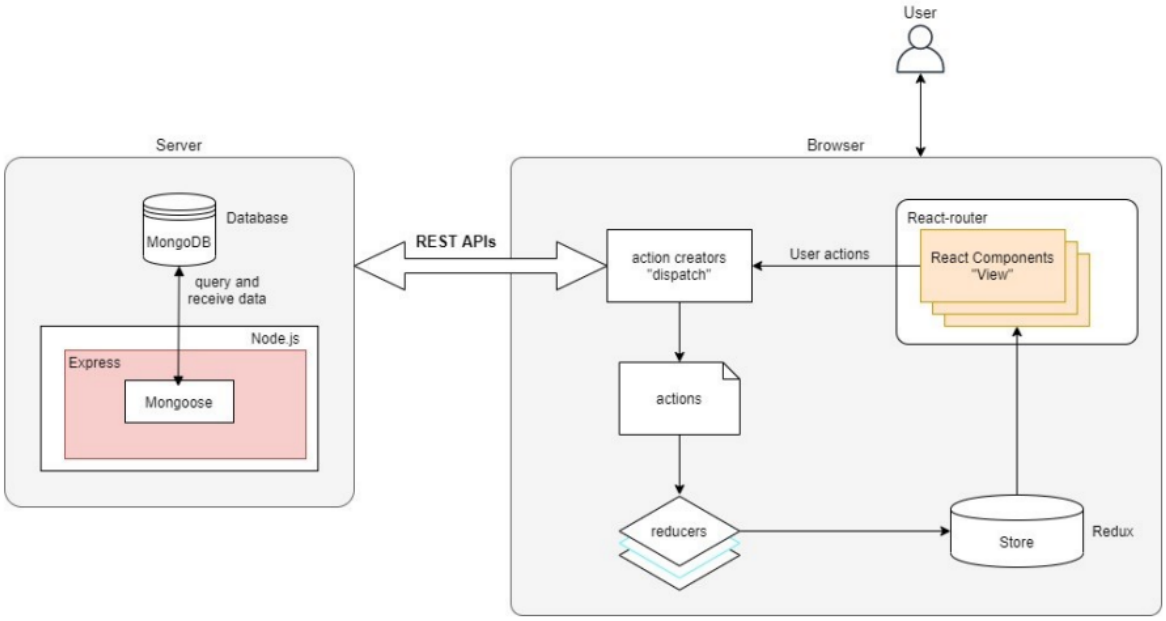


Рисунок 4.9 Архітектура створеного застосунку

3.2.2 Користувацький інтерфейс

Спочатку усіх користувачів зустрічає головний екран, де надана загальна інформація, висвітлені нові продукти, а також присутній розділ з усіма продуктами по категоріям (Додаток А).

При натисканні “Buy now” на картці продукту, він буде доданий до кошику з можливістю регулювання кількості одиниць, при використанні розробленого застосунку для справжнього бізнесу, необхідно буде під’єднати платіжну систему і сконфігурувати функцію, яка викликається при натисканні на кнопку “Buy” в кошику, оскільки наразі вона просто очищає кошик. (Додаток В).

Присутня можливість реєстрації із всією необхідною валідацією полів (Додаток С).

Існує кабінет користувача, при чому він залежить від його ролі: у продавця буде можливість додавати, редагувати та видаляти свої продукти, у адміністратора – призначати користувачів продавцями, проглядати і редагувати усю продукцію на сайті. (Додаток D)

Так як це повноцінний PWA, є можливість контролювати контент, який буде доступний користувачу в режимі офлайн. У даному застосунку у вигляді простої демонстрації буде виведено повідомлення про відсутність підключення до мережі (Додаток Е). Також застосунок може бути завантажений на пристрій користувача, у додатку F демонструється завантажений додаток на персональний комп’ютер та смартфон.

Висновки

У ході даної курсової роботи було досліджено й оцінено перспективи розвитку і можливості технології Progressive Web Application, а також розроблено власний застосунок для практичної демонстрації описаних аспектів даної технології. Було детально проаналізовано найпопулярніші типи додатків на ринку, виокремлено їх ключові особливості, переваги та недоліки. Після цього їх було зіставлено з PWA і чітко та вичерпно продемонстровано, чому PWA є більш універсальним, дешевим, кращим та ефективним рішенням для більшості задач сучасного бізнесу. PWA забезпечує користувацький досвід, що нічим не поступається досвіду, який клієнти отримують, користуючись нативними застосунками, ба більше, при цьому PWA активно використовує усі переваги веб-додатків.

І хоча технології, які використовуються в PWA, на сьогоднішній день все ще не настільки досконалі і відомі широкій публіці, щоб повністю витіснити із вжитку нативні застосунки, рано чи пізно це неодмінно станеться. Google продовжує активно розвивати концепцію PWA, а світові корпорації вже починають її активно використовувати [2]. Без сумніву можна стверджувати, що Progressive Web Applications – це майбутнє всієї індустрії веб-розробки, а також революція у світі мобільних застосунків.

Список використаних джерел

1. Introduction to progressive web apps – 2022. – [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Introduction

2. 12 Best Examples of Progressive Web Apps (PWAs) in 2022 – 2022. – [Електронний ресурс] – Режим доступу до ресурсу:

   https://www.simicart.com/blog/progressive-web-apps-examples/

3. Differences between Functional Components and Class Components in React – 2022. – [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/differences-between-functional-components-and-class-components-in-react/#:~:text=A%20functional%20component%20is%20just,method%20used%20in%20functional%20components.

4. Introducing Hooks – 2018. – [Електронний ресурс] – Режим доступу до ресурсу: https://reactjs.org/docs/hooks-intro.html

5. Progressive web app structure – 2022. – [Електронний ресурс] – Режим доступу до ресурсу:

   https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/App_structure

6. What does it take to be installable? – 2022. – [Електронний ресурс] – Режим доступу до ресурсу:

   https://web.dev/install-criteria/

7. How to define your install strategy – 2020. – [Електронний ресурс] – Режим доступу до ресурсу: https://web.dev/define-install-strategy/

8. Add a web app manifest – 2021. – [Електронний ресурс] – Режим доступу до ресурсу: https://web.dev/add-manifest/

9. Progressively enhance your Progressive Web App – 2020. – [Електронний ресурс] – Режим доступу до ресурсу: https://web.dev/progressively-enhance-your-pwa/

10. Service workers – 2022. – [Електронний ресурс] – Режим доступу до ресурсу:

https://web.dev/learn/pwa/service-workers/

11. Caching – 2022. – [Електронний ресурс] – Режим доступу до ресурсу: https://web.dev/learn/pwa/caching/

12. Serving – 2022. – [Електронний ресурс] – Режим доступу до ресурсу: https://web.dev/learn/pwa/serving/

13. Capabilities – 2022. – [Електронний ресурс] – Режим доступу до ресурсу:

https://web.dev/learn/pwa/capabilities/

14. Enhancements – 2022. – [Електронний ресурс] – Режим доступу до ресурсу:

https://web.dev/learn/pwa/enhancements/

15. Configuring Your Store – 2022. – [Електронний ресурс] – Режим доступу до ресурсу:

https://redux.js.org/usage/configuring-your-store

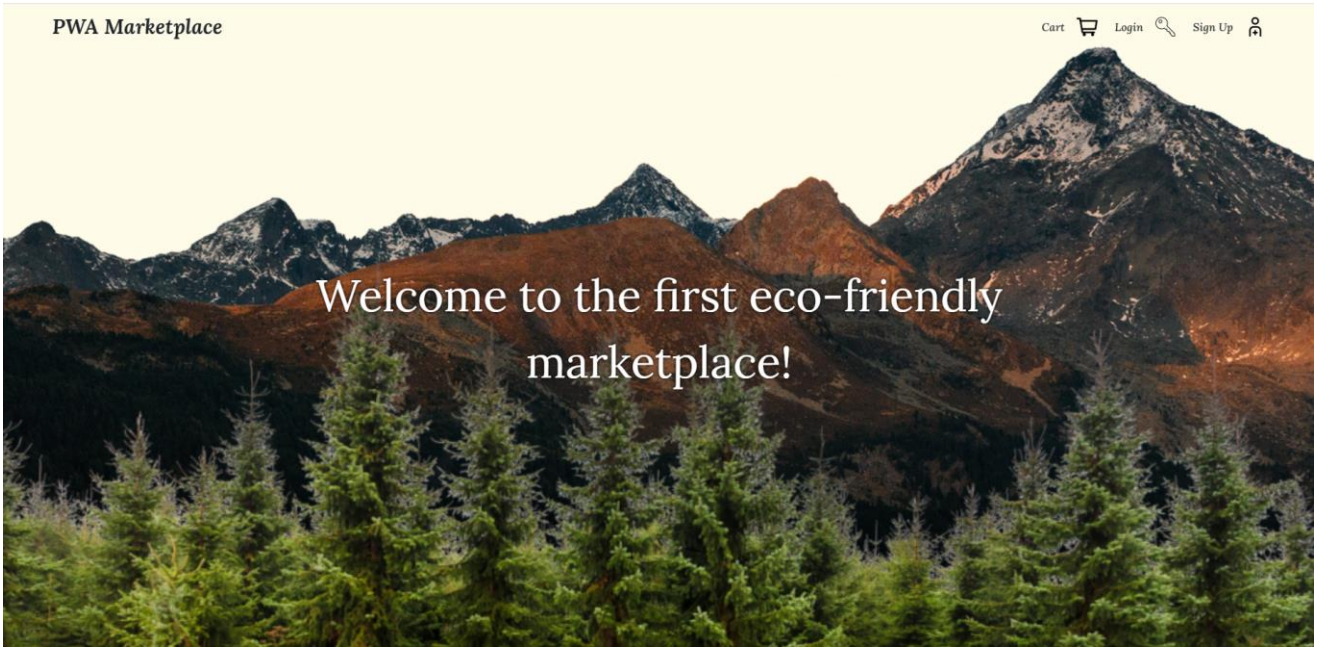
16. Code Splitting – 2021. – [Електронний ресурс] – Режим доступу до ресурсу: https://redux.js.org/usage/code-splitting

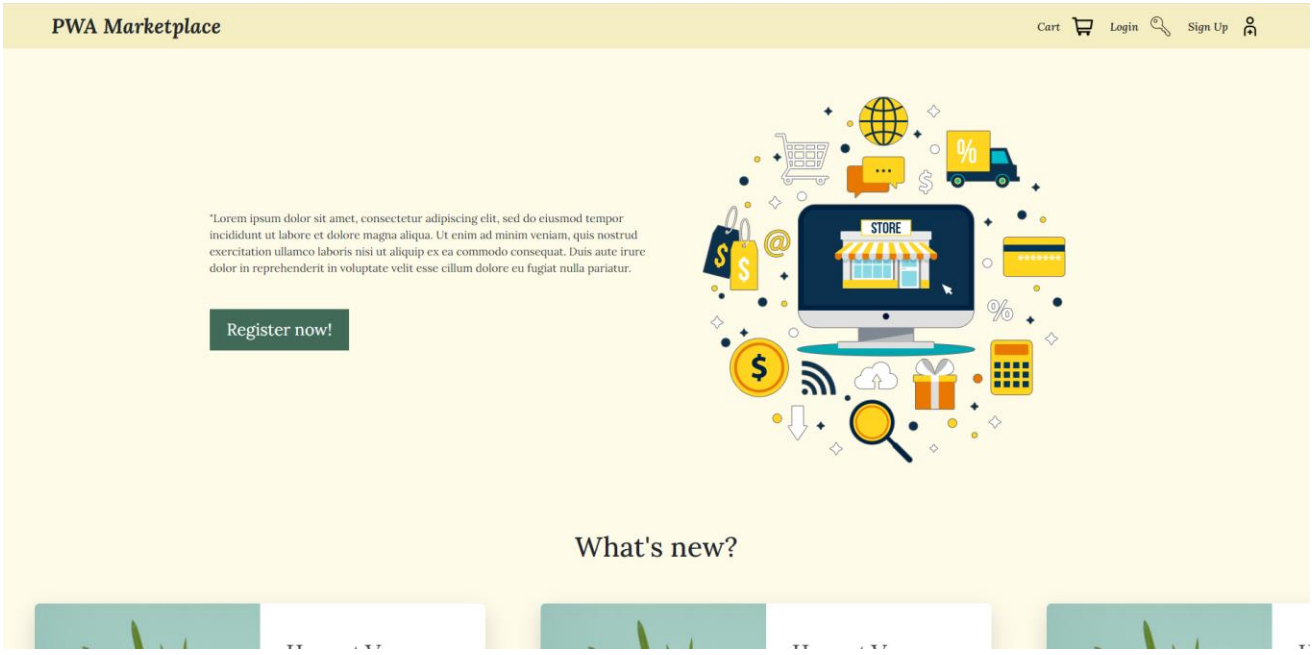
17. Deriving Data with Selectors – 2021. – [Електронний ресурс] – Режим доступу до ресурсу: https://redux.js.org/usage/deriving-data-selectors

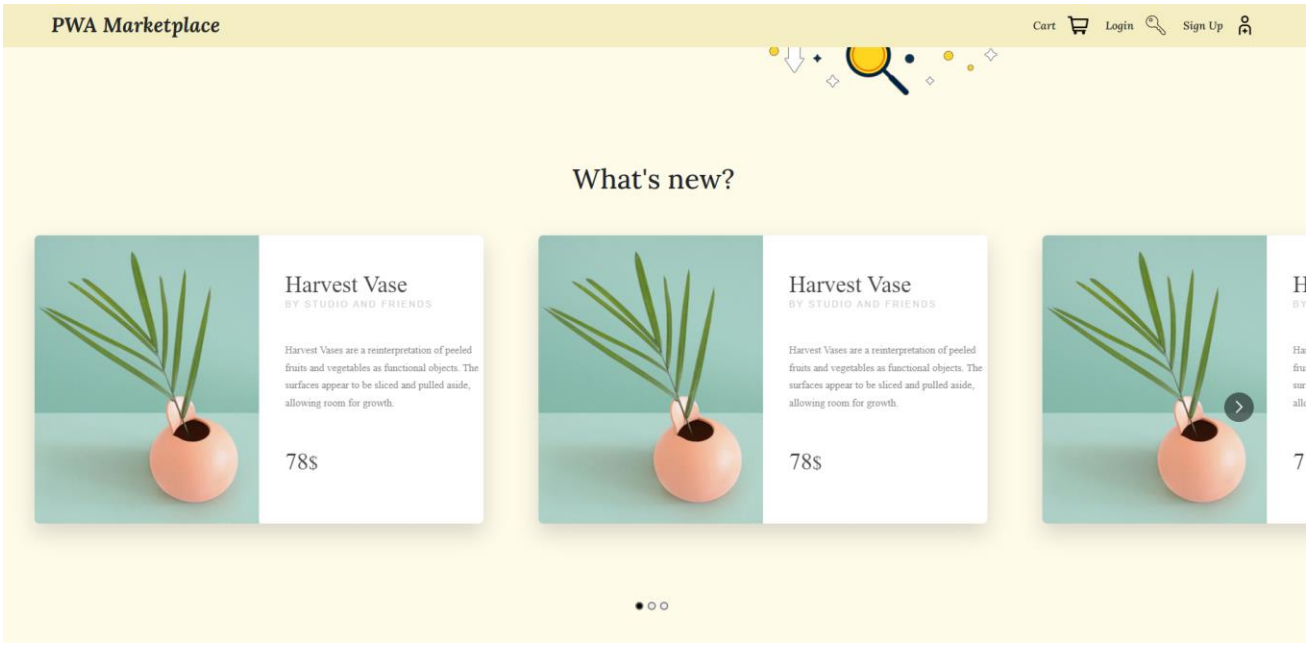
Додаток А

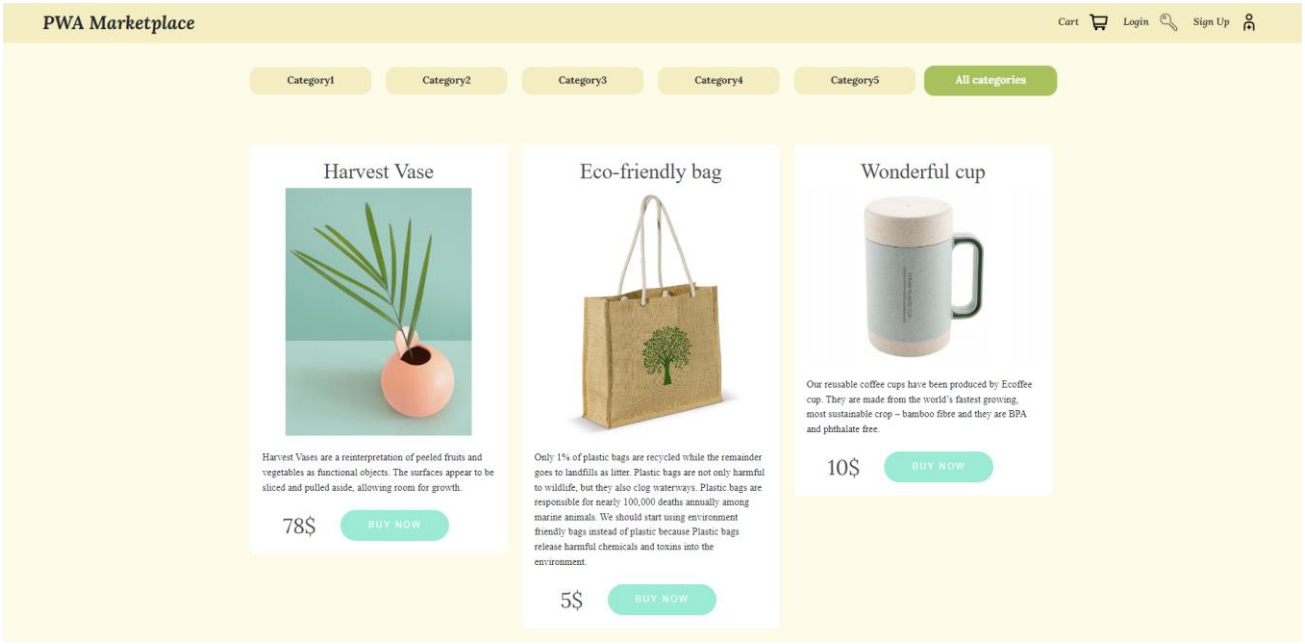
(Обов'язковий)

Головний екран



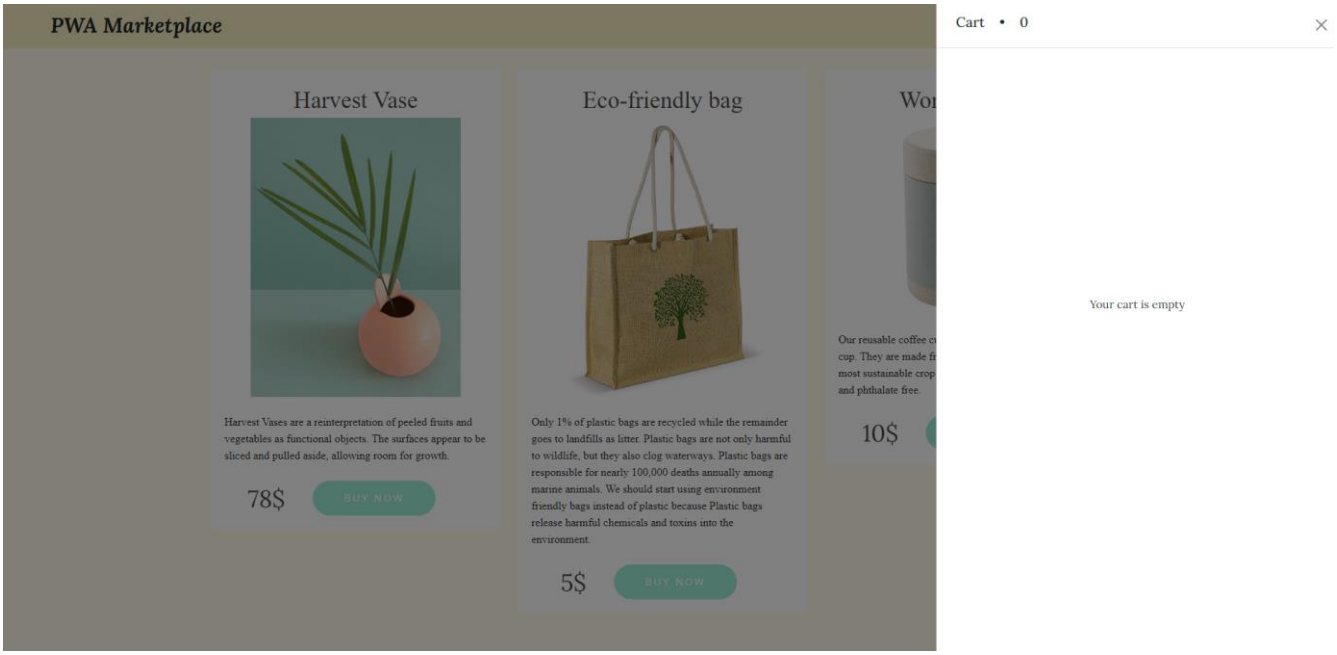


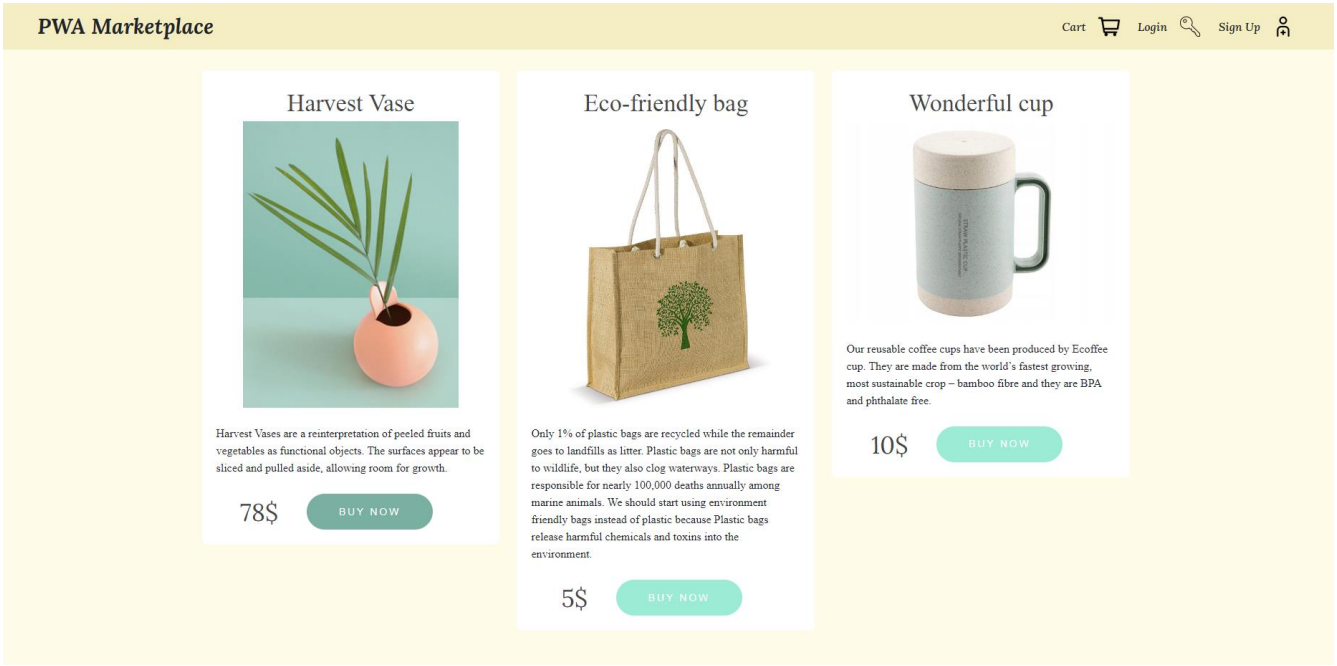

PWA Marketplace
Cart
Login
Sign Up
What's new?
Harvest Vase
BY STUDIO AND FRIENDS
Harvest Vases are a reinterpretation of peeled fruits and vegetables as functional objects. The surfaces appear to be sliced and pulled aside, allowing room for growth.
78$
Harvest Vase
BY STUDIO AND FRIENDS
Harvest Vases are a reinterpretation of peeled fruits and vegetables as functional objects. The surfaces appear to be sliced and pulled aside, allowing room for growth.
78$

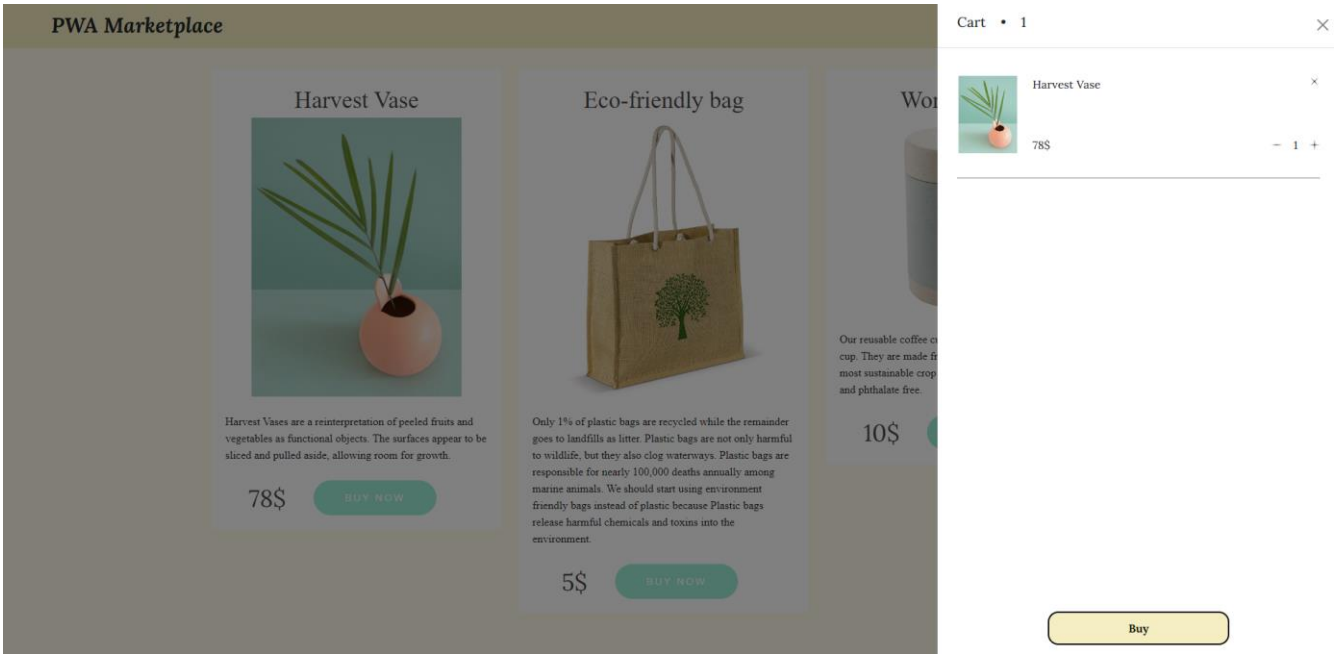

PWA Marketplace
Cart
Login
Sign Up
Category1
Category2
Category3
Category4
Category5
All categories
Harvest Vase
Harvest Vases are a reinterpretation of peeled fruits and vegetables as functional objects. The surfaces appear to be sliced and pulled aside, allowing room for growth.
78$
BUY NOW
Eco-friendly bag
Only 1% of plastic bags are recycled while the remainder goes to landfills as litter. Plastic bags are not only harmful to wildlife, but they also clog waterways. Plastic bags are responsible for nearly 100,000 deaths annually among marine animals. We should start using environment friendly bags instead of plastic because Plastic bags release harmful chemicals and toxins into the environment.
5$
BUY NOW
Wonderful cup
Our reusable coffee cups have been produced by Ecoffee cup. They are made from the world's fastest growing, most sustainable crop – bamboo fibre and they are BPA and phthalate free.
10$
BUY NOW

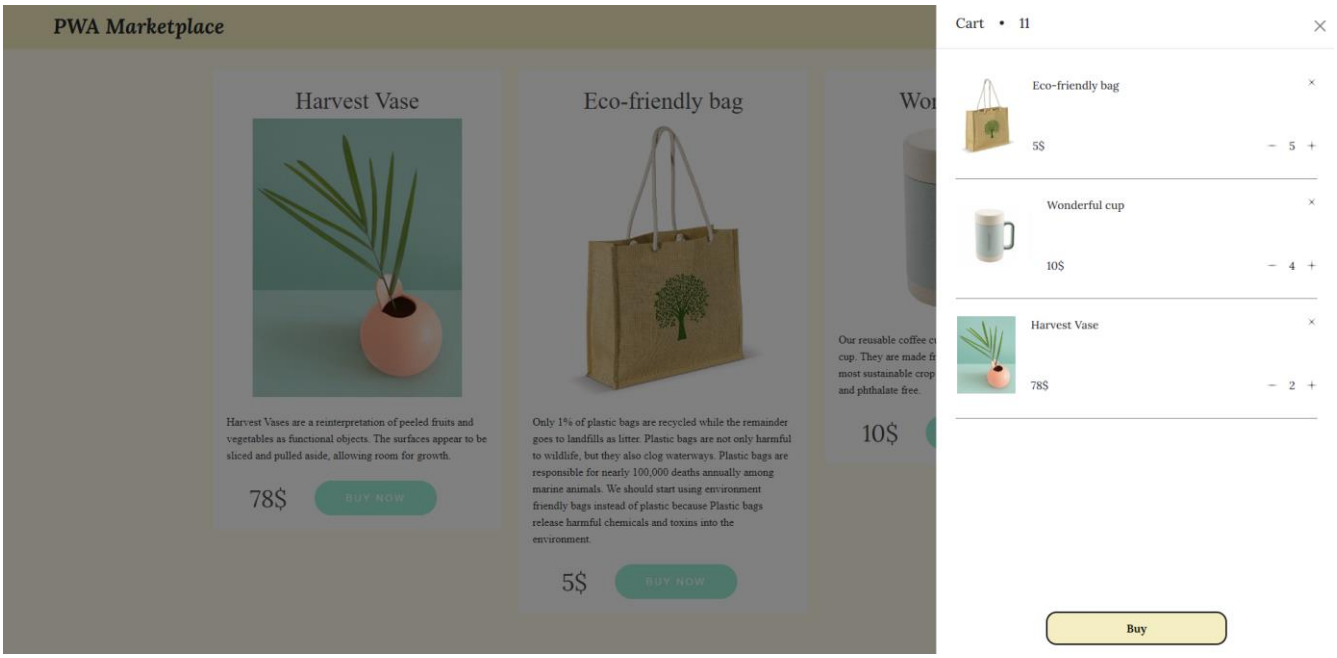
Додаток В
(Обов'язковий)

Додавання продуктів до кошику



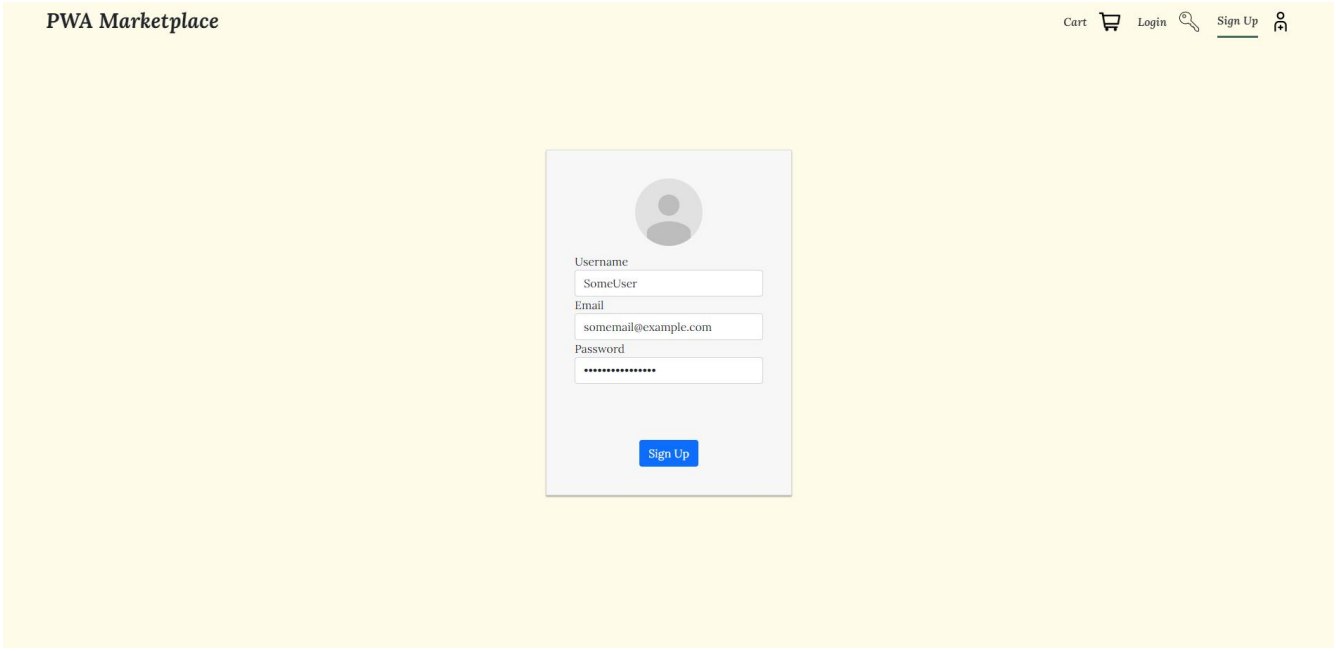


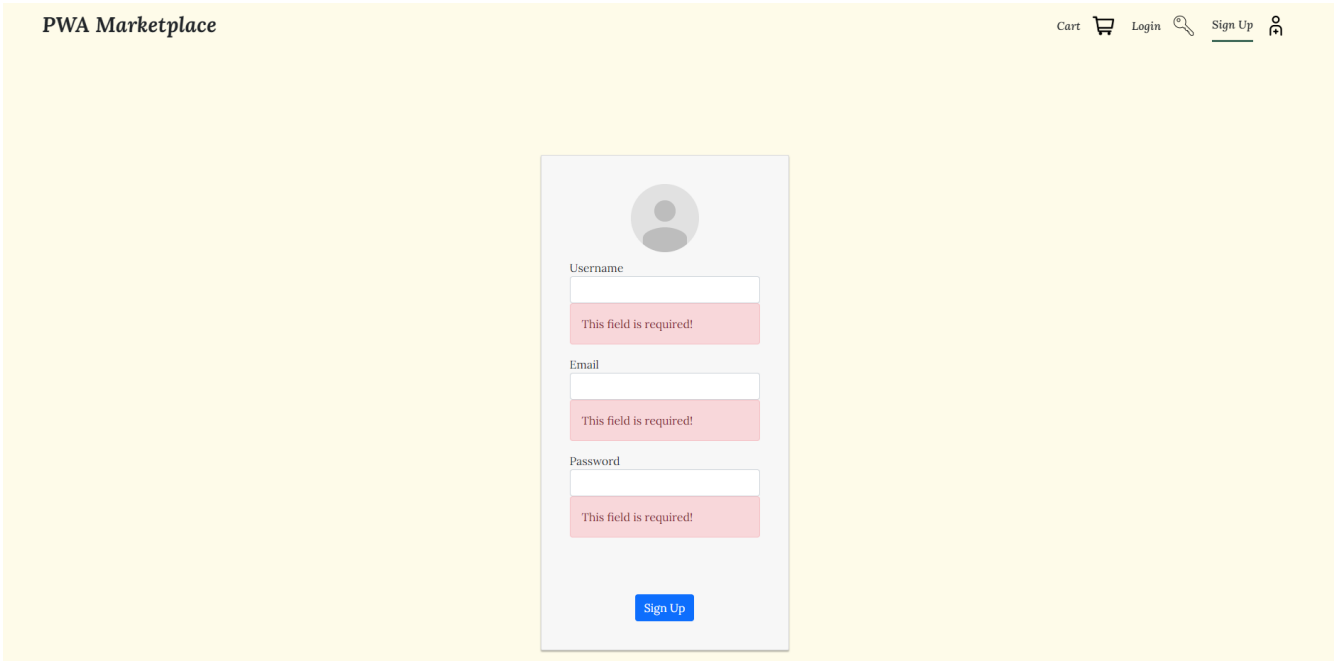

PWA Marketplace
Harvest Vase
Harvest Vases are a reinterpretation of peeled fruits and vegetables as functional objects. The surfaces appear to be sliced and pulled aside, allowing room for growth.
78$
BUY NOW
Eco-friendly bag
Only 1% of plastic bags are recycled while the remainder goes to landfills as litter. Plastic bags are not only harmful to wildlife, but they also clog waterways. Plastic bags are responsible for nearly 100,000 deaths annually among marine animals. We should start using environment friendly bags instead of plastic because Plastic bags release harmful chemicals and toxins into the environment.
5$
BUY NOW
10$
Cart • 1
Harvest Vase
78$
− 1 +
Buy

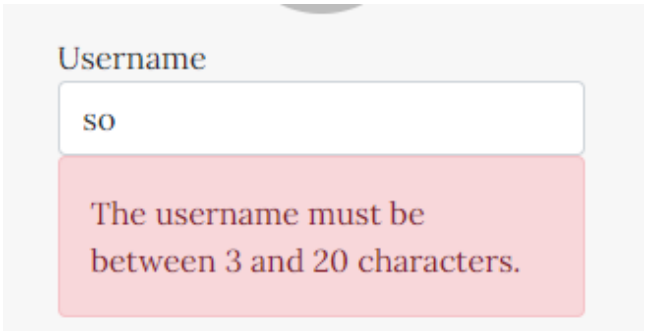

PWA Marketplace
Harvest Vase
Harvest Vases are a reinterpretation of peeled fruits and vegetables as functional objects. The surfaces appear to be sliced and pulled aside, allowing room for growth.
78$
BUY NOW
Eco-friendly bag
Only 1% of plastic bags are recycled while the remainder goes to landfills as litter. Plastic bags are not only harmful to wildlife, but they also clog waterways. Plastic bags are responsible for nearly 100,000 deaths annually among marine animals. We should start using environment friendly bags instead of plastic because Plastic bags release harmful chemicals and toxins into the environment.
5$
BUY NOW
10$
Cart • 11
Eco-friendly bag
5$
− 5 +
Wonderful cup
10$
− 4 +
Harvest Vase
78$
− 2 +
Buy

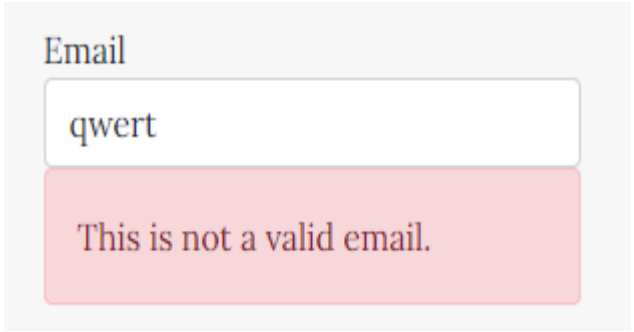
Додаток С
(Обов'язковий)

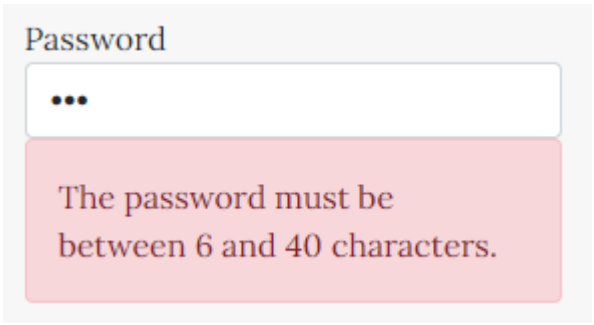
Реєстрація і логін

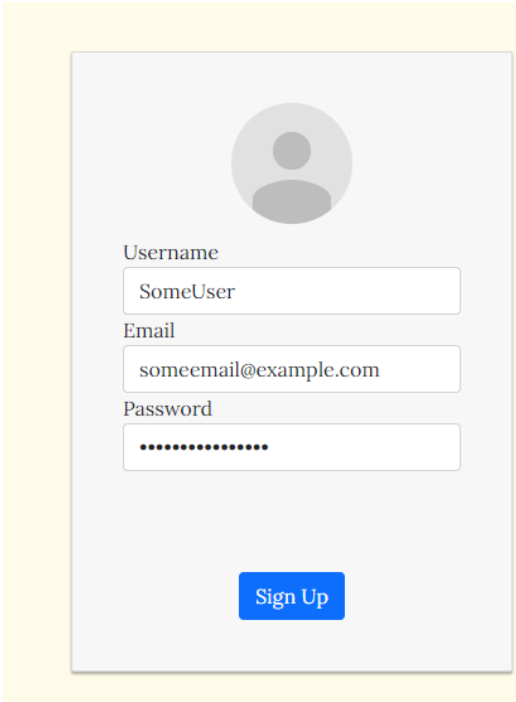


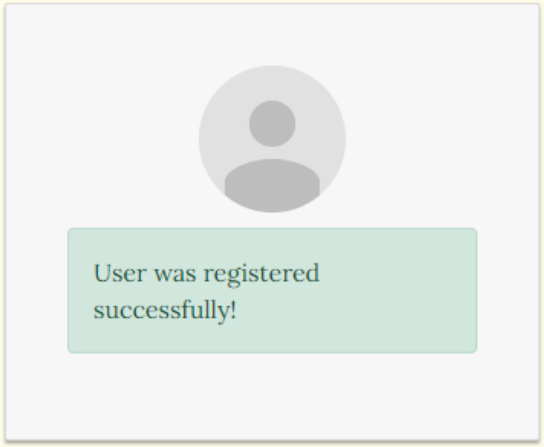



Username
so
The username must be between 3 and 20 characters.


Email
qwert
This is not a valid email.

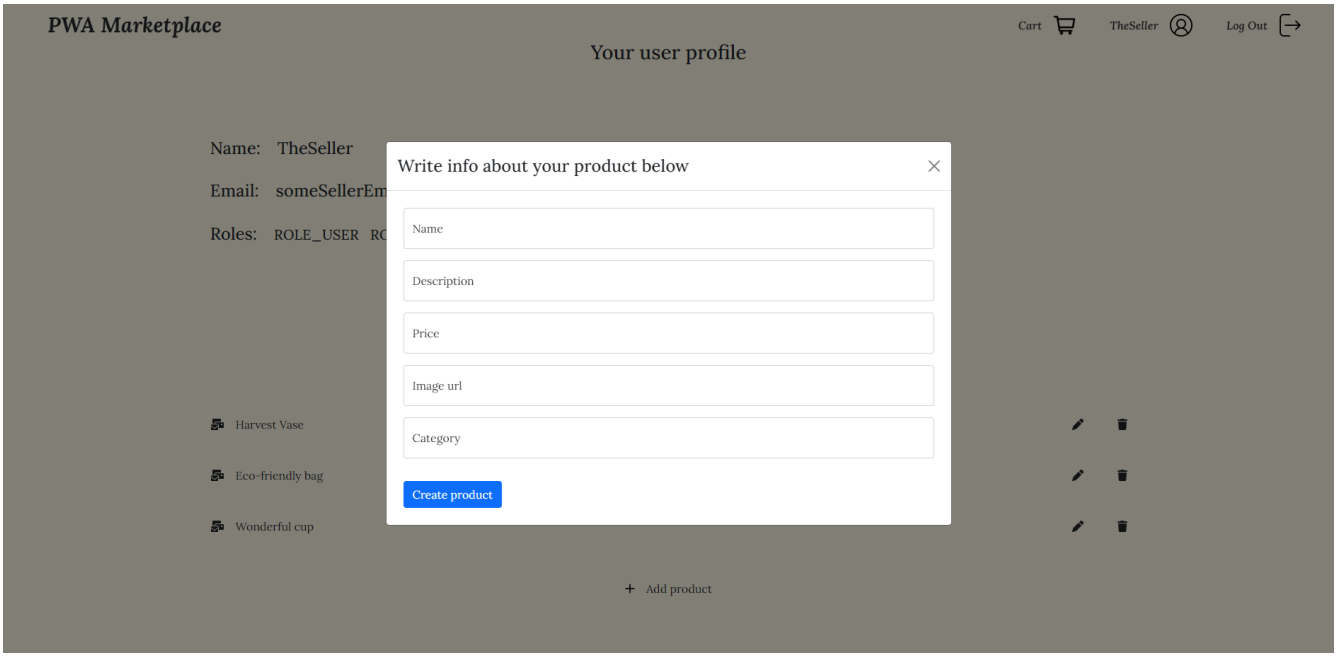

Password
•••
The password must be between 6 and 40 characters.


Username
SomeUser
Email
someemail@example.com
Password
Sign Up


User was registered successfully!

Додаток D
(Обов'язковий)

Приклад кабінету користувача



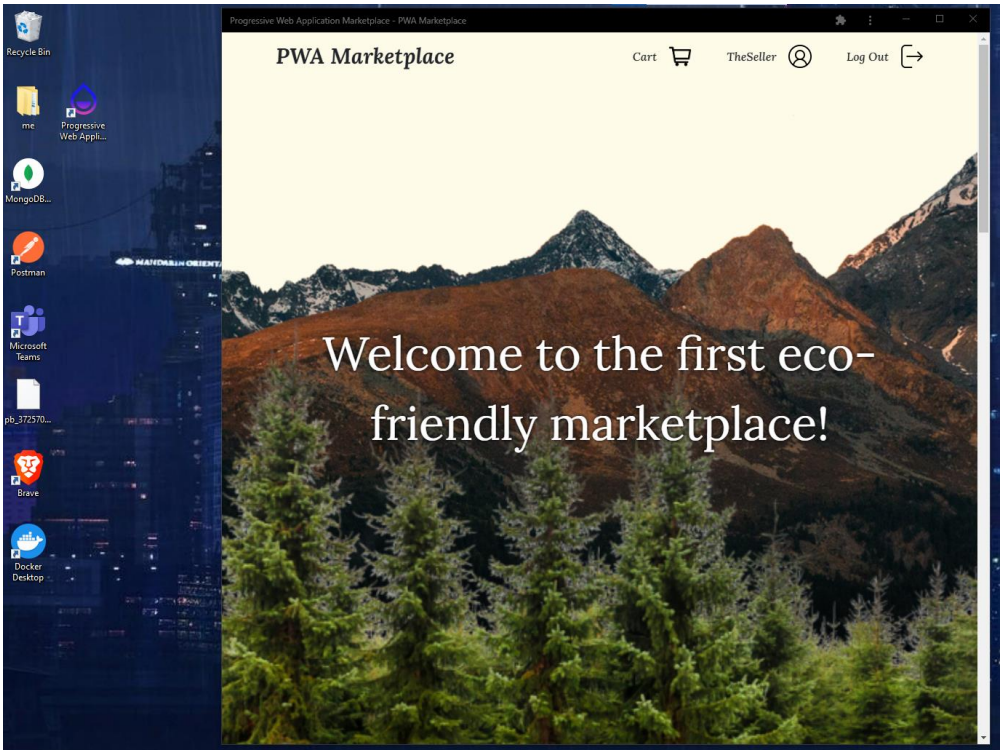
Додаток Е

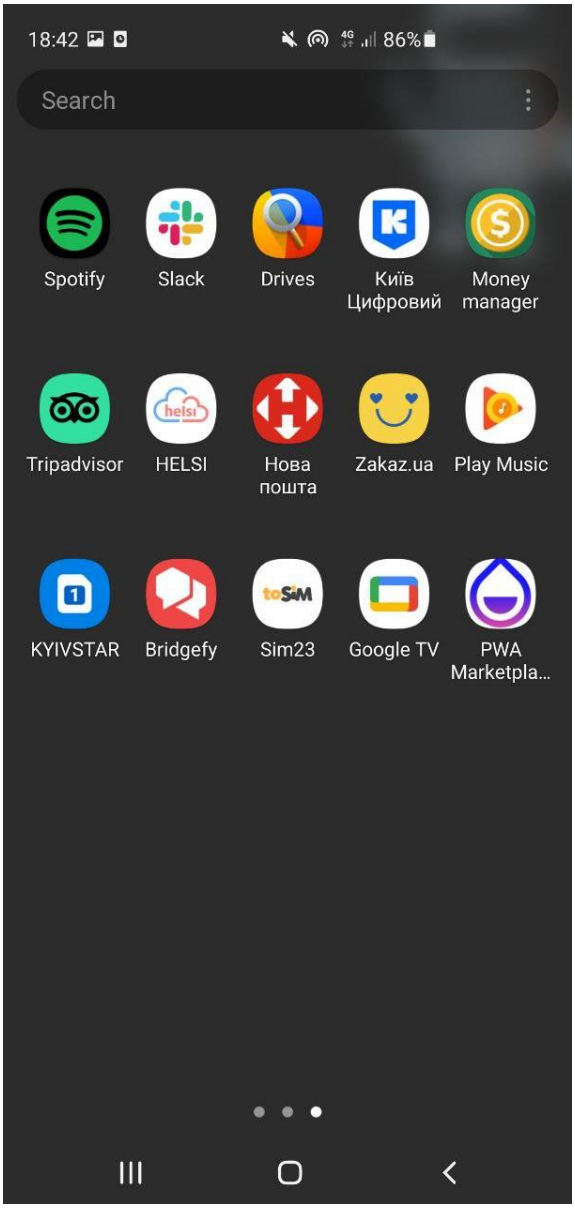
(Обов’язковий)

Приклад контенту в режимі офлайн

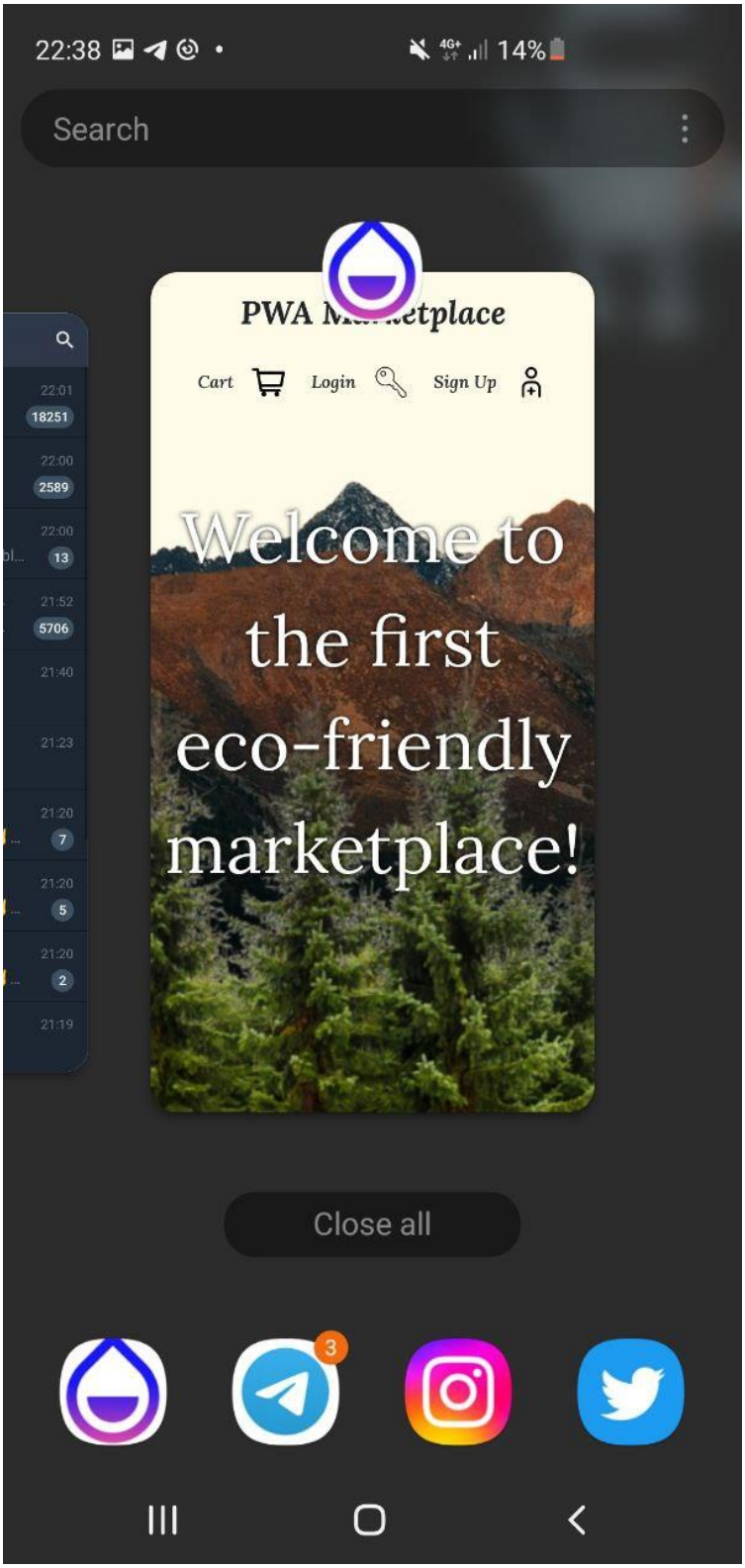
Додаток F
(Обов'язковий)

Приклад завантаженого застосунку на комп'ютер та смартфон




18:42
86%
Search
Spotify
Slack
Drives
Київ Цифровий
Money manager
Tripadvisor
HELSI
Нова пошта
Zakaz.ua
Play Music
KYIVSTAR
Bridgefy
Sim23
Google TV
PWA Marketpla...


22:38
14%
PWA Marketplace
Cart
Login
Sign Up
Welcome to the first eco-friendly marketplace!


22:38
14%
Search
PWA Marketplace
Cart
Login
Sign Up
Welcome to the first eco-friendly marketplace!
Close all