МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему **Інтеграція Progressive Web Application (PWA) з системою електронної комерції та порівняльний аналіз з традиційними системами.**

Виконав: студент 2 курсу, групи 8.1214-іпз-2
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Інженерія програмного забезпечення
(код і назва освітньої програми)
С. О.Вєтров
(ініціали та прізвище)
Керівник доцент, к.т.н., доцент В.І. Заяц
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент Директор ТОВ «Дискус»
Р.О. Лютий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

Кафедра __електроніки, інформаційних систем та програмного__ забезпечення
Рівень вищої освіти _____другий (магістерський)_____
Спеціальність ___121_Інженерія програмного забезпечення_____
(код та назва)
Освітня програма ___Інженерія програмного забезпечення
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри _____Т.В. Критська_
“_01_” __квітня_____2025_року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Вєтров Сергій Олександрович
(прізвище, ім’я, по батькові)

1.Тема роботи __Інтеграція Progressive Web Application (PWA) з системою електронної комерції та порівняльний аналіз з традиційними системами.

керівник роботи ___к.т.н., доцент Заяц Валерій Іванович_____
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від _28.03. 2025 р. № 361-е__
2. Строк подання студентом кваліфікаційної роботи_____01.12.2025 р.___
3. Вихідні дані магістерської роботи
- комплект нормативних документів ;
- технічне завдання до роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
- огляд та аналіз літературних джерел за темою Progressive Web Application у системах електронної комерції;
- аналіз існуючих рішень та підходів до реалізації PWA;
- проєктування та розробка PWA-застосунку для системи електронної комерції;
- визначення вимог до функціонування розробленого застосунку;

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
_____слайдів презентації

6. Консультанти розділів магістерської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата Завдання прийняв

7. Дата видачі завдання 01.04.2025

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів магістерської роботи	Примітка
1	Формулювання теми магістерської роботи, визначення мети, завдань та узгодження структури роботи з науковим керівником.	15.04 – 20.04.25	виконано
2	Перший розділ. Аналіз сучасних веб-технологій та концепції Progressive Web Application.	21.04 – 30.04.23	виконано
3	Літературний огляд. Другий розділ. Архітектурні підходи до побудови клієнтських застосунків у системах електронної комерції.	01.05 – 10.05.25	виконано
4	Аналіз нативних, гібридних та PWA-рішень, порівняння їхніх переваг і недоліків.	11.05 – 20.05.25	виконано
6	Проєктування архітектури власного PWA-застосунку для системи електронної комерції. Вибір технологій та інструментів розробки.	21.05 – 06.08.25	виконано
7	Розробка клієнтської частини PWA-застосунку на основі Magento PWA Studio та Venia UI.	01.09 – 30.09.25	виконано
8	Реалізація механізмів офлайн-стійкості, кешування та роботи в умовах нестабільного мережевого з’єднання.	01.10.25	виконано
9	Четвертий розділ. Оцінка відповідності розробленого PWA поставленим цілям	02.10 – 31.10.25	виконано
10	Оформлення звіту.	01.11 – 27.11.25	виконано

Студент _____ Вєтров С.О.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Заяц В.І.
(підпис) (прізвище та ініціали)

Нормоконтроль пройдено

Нормоконтролер _____ Скрипник І.А.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сторінок: 110

Рисунків: 12

Таблиць: 1

Джерел: 26

Вєтров С. О. Інтеграція Progressive Web Application (PWA) з системою електронної комерції та порівняльний аналіз з традиційними системами: кваліфікаційна робота магістра спеціальності 121 «Інженерія програмного забезпечення» / наук. керівник Заяц В.І. Запоріжжя : ЗНУ, 2025. 110 с.

Метою роботи є дослідження можливостей використання технології Progressive Web Application у системах електронної комерції та оцінка ефективності PWA-рішення у порівнянні з традиційними веб-застосунками. У роботі розглянуто теоретичні засади PWA, архітектурні особливості, безпекові та продуктивні аспекти, а також підходи до інтеграції з сучасними e-commerce платформами.

Практична частина присвячена розробці PWA для системи електронної комерції на базі Magento 2 з використанням GraphQL API, Service Workers та механізмів кешування. Проведено аналіз продуктивності розробленого застосунку та оцінено його відповідність поставленим цілям роботи.

Ключові слова: Progressive Web Application, PWA, електронна комерція, Magento 2, GraphQL, Service Workers, продуктивність.

ABSTRACT

Pages – 110

Drawings – 12

Tables – 1

Sources – 26

Vietrov S. O. Integration of Progressive Web Application (PWA) with an E-commerce System and Comparative Analysis with Traditional Solutions: Master's Thesis in Specialty 121 "Software Engineering" / Scientific supervisor V. I. Zaiats. Zaporizhzhia: Zaporizhzhia National University, 2025. 110 p.

The purpose of this thesis is to investigate the possibilities of using Progressive Web Application technology in e-commerce systems and to evaluate the effectiveness of PWA solutions in comparison with traditional web applications. The paper examines the theoretical foundations of PWA, architectural features, security and performance aspects, as well as approaches to integration with modern e-commerce platforms.

The practical part of the work is devoted to the development of a PWA for an e-commerce system based on Magento 2, using the GraphQL API, Service Workers, and caching mechanisms. The performance of the developed application is analyzed, and its compliance with the objectives of the study is evaluated.

Keywords: Progressive Web Application, PWA, e-commerce, Magento 2, GraphQL, Service Workers, performance.

ЗМІСТ

ВСТУП 8
РОЗДІЛ 1 ПОРІВНЯЛЬНИЙ АНАЛІЗ МОЖЛИВОСТЕЙ ІНТЕГРАЦІЇ PWA У РІЗНІ СИСТЕМИ ЕЛЕКТРОНОЇ КОМЕРЦІЇ 14
1.1 Теоретичні засади технології PWA 14
1.2 Критерії вибору платформи для PWA 17
1.3 Практичні аспекти використання технології PWA 20
1.4 Концепція Progressive Web Application та її ключові характеристики 23
1.5 Архітектура PWA: Service Workers, Web App Manifest, офлайн-кешування 26
1.6 Безпекові та продуктивні аспекти PWA у контексті електронної комерції .. 30
1.7 Висновки до розділу 1 33
РОЗДІЛ 2 ПРАКТИЧНІ ПІДХОДИ ТА ТЕХНОЛОГІЧНІ РІШЕННЯ ДЛЯ РЕАЛІЗАЦІЇ PWA У СИСТЕМАХ ЕЛЕКТРОННОЇ КОМЕРЦІЇ 36
2.1 Загальні підходи до реалізації PWA у веб-системах 36
2.2 Технологічні інструменти та фреймворки для розробки PWA 38
2.3 Реалізація PWA у Magento 2 42
2.4 Реалізація PWA у Shopify 45
2.5 Порівняння підходів до реалізації PWA на Magento 2, Shopify та WooCommerce 51
2.6 Висновки до розділу 2 53
РОЗДІЛ 3 ПРОЄКТУВАННЯ ТА РОЗРОБКА PWA ДЛЯ ПЛАТФОРМИ MAGENTO 2 56
3.1 Аналіз вимог та підготовка до розробки PWA 56
3.2 Проєктування архітектури клієнтської частини 58
3.3 Інтеграція PWA з backend API системи електронної комерції 62
3.4 Реалізація ключових функціональних модулів PWA 65

3.4.1 Організація взаємодії з GraphQL та управління станом клієнта 67
3.4.2 Архітектура компонентів інтерфейсу та логіка їх зв'язності .. 74
3.4.3 Механізми обробки користувацьких дій та синхронізація з backend 79
3.4.4 Реалізація офлайн-стійкості та роботи у нестабільному мережевому середовищі 81
3.5 Висновки до розділу 3 87
РОЗДІЛ 4 ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ВИКОРИСТАННЯ PWA У СИСТЕМАХ ЕЛЕКТРОННОЇ КОМЕРЦІЇ 89
4.1 Мета та методика дослідження ефективності PWA-рішення 89
4.2 Аналіз продуктивності PWA у порівнянні з традиційним веб-застосунком. 91
4.3 Аналіз поширеності використання PWA у сучасних системах електронної комерції 96
4.4 Аналіз відповідності розробленого PWA поставленим цілям роботи 99
4.5 Висновки до розділу 4 104
ВИСНОВКИ 106
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 108

ВСТУП

Актуальність теми

Сучасний етап розвитку інформаційних технологій характеризується стрімким зростанням обсягів електронної комерції та суттєвим збільшенням частки мобільного трафіку у структурі онлайн-продажів. За останні роки мобільні пристрої стали основним засобом доступу користувачів до інтернет-магазинів, що висуває нові вимоги до швидкодії, стабільності та зручності веб-застосунків. Водночас традиційні веб-рішення часто демонструють недостатню продуктивність на мобільних платформах, зокрема через повільне завантаження сторінок, залежність від якості мережевого з’єднання та обмежені можливості роботи в офлайн-режимі.

Альтернативним підходом до підвищення якості користувацького досвіду є розробка нативних мобільних застосунків. Проте цей підхід пов’язаний зі значними витратами ресурсів, необхідністю підтримки декількох платформ (iOS, Android), ускладненням процесу оновлення та залежністю від сторонніх маркетплейсів. Для багатьох компаній, особливо у сфері електронної комерції, такі обмеження стають суттєвим стримувальним фактором упровадження мобільних рішень.

У цьому контексті технологія Progressive Web Application (PWA) набуває особливої актуальності. Вона поєднує переваги класичних веб-застосунків та нативних мобільних рішень, забезпечуючи високу швидкість роботи, можливість офлайн-доступу, підтримку push-сповіщень і встановлення застосунку безпосередньо з браузера. Використання Service Workers, кешування ресурсів та сучасних API дозволяє значно зменшити час завантаження сторінок і підвищити стабільність роботи застосунку навіть за умов нестабільного інтернет-з’єднання.

Особливої актуальності тема PWA набуває у сфері електронної комерції, де швидкодія інтерфейсу, безперервність користувацького сценарію та мінімізація часу очікування безпосередньо впливають на рівень конверсії та лояльність клієнтів. Сучасні e-commerce платформи дедалі частіше орієнтуються на headless-архітектуру та API-орієнтовані підходи, що створює сприятливі умови для інтеграції Progressive Web Application як клієнтської частини системи.

Попри

та розробка зростаючу популярність PWA, практичні питання її інтеграції з реальними системами електронної комерції, а також об’єктивна оцінка ефективності таких рішень у порівнянні з традиційними веб-застосунками залишаються недостатньо висвітленими. Це зумовлює потребу у комплексному дослідженні архітектурних, технологічних та практичних аспектів впровадження PWA у e-commerce середовищі. Саме тому обрана тема є актуальною з точки зору сучасного стану розвитку веб-технологій, практичних потреб бізнесу та науково-прикладних завдань інженерії програмного забезпечення.

Мета і завдання дослідження

Метою кваліфікаційної роботи є дослідження можливостей інтеграції технології Progressive Web Application із системами електронної комерції та оцінка ефективності використання PWA як клієнтського рішення у порівнянні з традиційними веб-застосунками. У роботі ставиться за мету обґрунтувати доцільність застосування PWA у сучасних e-commerce системах на основі аналізу архітектурних підходів і практичної реалізації прототипу.

Для досягнення поставленої мети в роботі передбачається виконання таких основних завдань: аналіз теоретичних засад та ключових характеристик технології PWA; дослідження підходів до її реалізації у системах електронної комерції; проєктування PWA-прототипу з інтеграцією з backend API e-commerce

платформи; проведення аналізу продуктивності розробленого рішення та оцінка його відповідності поставленим цілям роботи.

Об’єкт дослідження

Об’єктом дослідження є системи електронної комерції та клієнтські веб-застосунки, що забезпечують взаємодію користувачів з функціоналом інтернет-магазинів.

Предмет дослідження

Предметом дослідження є архітектурні та програмні засоби реалізації Progressive Web Application, а також методи інтеграції PWA з backend-системами електронної комерції. Дослідження зосереджене на клієнтській архітектурі, механізмах взаємодії через API та забезпеченні продуктивності й стабільності роботи веб-застосунку.

Методи дослідження

У процесі виконання кваліфікаційної роботи використано такі методи дослідження:

1. Аналіз та узагальнення науково-технічної літератури й сучасних публікацій з тематики Progressive Web Application та електронної комерції.
2. Порівняльний аналіз традиційних веб-застосунків, нативних рішень і PWA з точки зору архітектури та продуктивності.
3. Методи проєктування програмних систем під час розробки клієнтської архітектури PWA.
4. експериментальні методи для оцінки швидкодії та ефективності розробленого PWA-рішення.

5. практичні методи розробки програмного забезпечення з використанням сучасних веб-технологій та API-орієнтованих підходів

Наукова новизна одержаних результатів

Наукова новизна одержаних результатів полягає у комплексному дослідженні можливостей застосування технології Progressive Web Application у системах електронної комерції на основі практичної реалізації PWA-прототипу та експериментальної оцінки його ефективності. У роботі обґрунтовано доцільність використання PWA як клієнтського рішення для e-commerce платформ та визначено переваги такого підходу у порівнянні з традиційними веб-застосунками.

Практичне значення одержаних результатів

Практичне значення одержаних результатів полягає у можливості використання отриманих висновків і запропонованих підходів під час проєктування та розробки сучасних систем електронної комерції. Результати дослідження можуть бути застосовані для вибору оптимальної клієнтської архітектури веб-застосунків з урахуванням вимог до продуктивності, масштабованості та зручності користувацького досвіду.

Розроблений PWA-прототип та описані технологічні рішення можуть бути використані як основа для впровадження прогресивних веб-застосунків у реальних комерційних проєктах, а також у навчальному процесі при вивченні сучасних веб-технологій і підходів до розробки програмного забезпечення.

Апробація одержаних результатів

Результати дослідження були представлені на XIII науково-практичній конференції студентів, аспірантів, докторантів і молодих вчених Запорізького національного університету «Молода наука-2025» [1], а також на XXV науково-

технічній конференції студентів, аспірантів, магістрантів і викладачів Інженерного навчально-наукового інституту Запорізького національного університету.

Глосарій

Progressive Web Application (PWA) — сучасний тип веб-застосунку, що поєднує можливості класичних веб-сайтів і нативних мобільних застосунків, забезпечуючи високу продуктивність, офлайн-доступ та можливість встановлення на пристрій користувача.

Електронна комерція (e-commerce) — форма комерційної діяльності, що передбачає здійснення купівлі-продажу товарів і послуг із використанням мережі Інтернет та цифрових технологій.

Service Worker — фоновий JavaScript-процес, який виконується браузером окремо від основного потоку застосунку та забезпечує кешування ресурсів, обробку мережевих запитів і реалізацію офлайн-режиму.

Web App Manifest — JSON-файл, що описує метадані PWA-застосунку, зокрема його назву, іконки, стартову сторінку та режим відображення.

Офлайн-режим — режим роботи веб-застосунку, за якого користувач має доступ до частини функціоналу без активного підключення до мережі Інтернет.

GraphQL — мова запитів до API, що дозволяє клієнтському застосунку отримувати лише необхідні дані та оптимізувати обмін інформацією між клієнтом і сервером.

REST API — архітектурний стиль взаємодії клієнтських і серверних застосунків на основі HTTP-запитів.

Headless-архітектура — підхід до побудови програмних систем, за якого клієнтська частина відокремлена від серверної логіки та взаємодіє з нею через API.

Клієнтська архітектура — структура та організація компонентів інтерфейсу користувача і логіки обробки даних у веб-застосунку.

Backend API — програмний інтерфейс, що забезпечує доступ клієнтського застосунку до серверної логіки та даних системи.

Кешування — механізм тимчасового зберігання даних або ресурсів з метою підвищення швидкодії та зменшення навантаження на сервер.

Продуктивність веб-застосунку — сукупність характеристик, що визначають швидкість завантаження, відгук інтерфейсу та стабільність роботи застосунку.

Мобільний користувацький досвід (англ. Mobile UX) — рівень зручності та ефективності взаємодії користувача з веб-застосунком на мобільних пристроях.

Magento 2 — платформа електронної комерції з відкритим вихідним кодом, що підтримує модульну архітектуру та API-орієнтований підхід.

Shopify — хмарна платформа електронної комерції для створення та управління інтернет-магазинами.

WooCommerce — плагін для системи управління контентом WordPress, що використовується для створення інтернет-магазинів.

PWA Studio — набір інструментів та бібліотек для розробки Progressive Web Application на базі платформи Magento 2.

Apollo Client — бібліотека для роботи з GraphQL, що забезпечує управління станом клієнтського застосунку та кешування запитів.

Користувацький інтерфейс (англ. UI) — сукупність елементів, через які користувач взаємодіє із застосунком.

Конверсія — показник ефективності електронної комерції, що відображає відношення кількості покупок до кількості відвідувачів.

РОЗДІЛ 1 ПОРІВНЯЛЬНИЙ АНАЛІЗ МОЖЛИВОСТЕЙ ІНТЕГРАЦІЇ PWA У РІЗНІ СИСТЕМИ ЕЛЕКТРОНОЇ КОМЕРЦІЇ

1.1 Теоретичні засади технології PWA

Технологія прогресивних веб-додатків (Progressive Web Applications, PWA) стала одним із ключових напрямів розвитку сучасної веб-розробки. PWA поєднує в собі переваги звичайних веб-сайтів та мобільних застосунків, що дозволяє створювати адаптивні, швидкі й функціональні рішення для електронної комерції. Теоретичні засади цієї технології базуються на поєднанні новітніх стандартів HTML, CSS і JavaScript, а також на використанні таких інструментів, як Service Worker, Web App Manifest та технології кешування, що дозволяють забезпечити офлайн-доступ, миттєве завантаження і зручність користування навіть за повільного інтернет-з'єднання [1]. Загальна схему роботи PWA представлено у Рисунку 1.

Однією з центральних концепцій у структурі PWA є використання Service Worker - скрипта, який браузер запускає у фоновому режимі, незалежно від веб-сторінки. Саме завдяки цьому механізму прогресивні веб-додатки можуть працювати в офлайн-режимі, обробляти push-повідомлення та реалізовувати складну логіку кешування. Технологія суттєво підвищує зручність користувача, адже дозволяє взаємодіяти із застосунком навіть за відсутності інтернету або при низькій якості з'єднання. Цей аспект є особливо важливим для ринку електронної комерції, де кожна затримка чи збій у роботі може призвести до втрати потенційного клієнта.

Іншою важливою складовою PWA є технологія Web App Manifest. Це конфігураційний JSON-файл, який описує основні параметри застосунку: його назву, іконку, кольори інтерфейсу, орієнтацію екрана та інші аспекти. Завдяки

цьому файлу користувач може встановити веб-застосунок на головний екран свого мобільного пристрою, що забезпечує досвід, максимально наближений до використання нативних мобільних застосунків. У поєднанні з адаптивною верстkою і використанням кешованих ресурсів, ця особливість дозволяє досягати високих показників конверсії та підвищувати лояльність клієнтів [3].

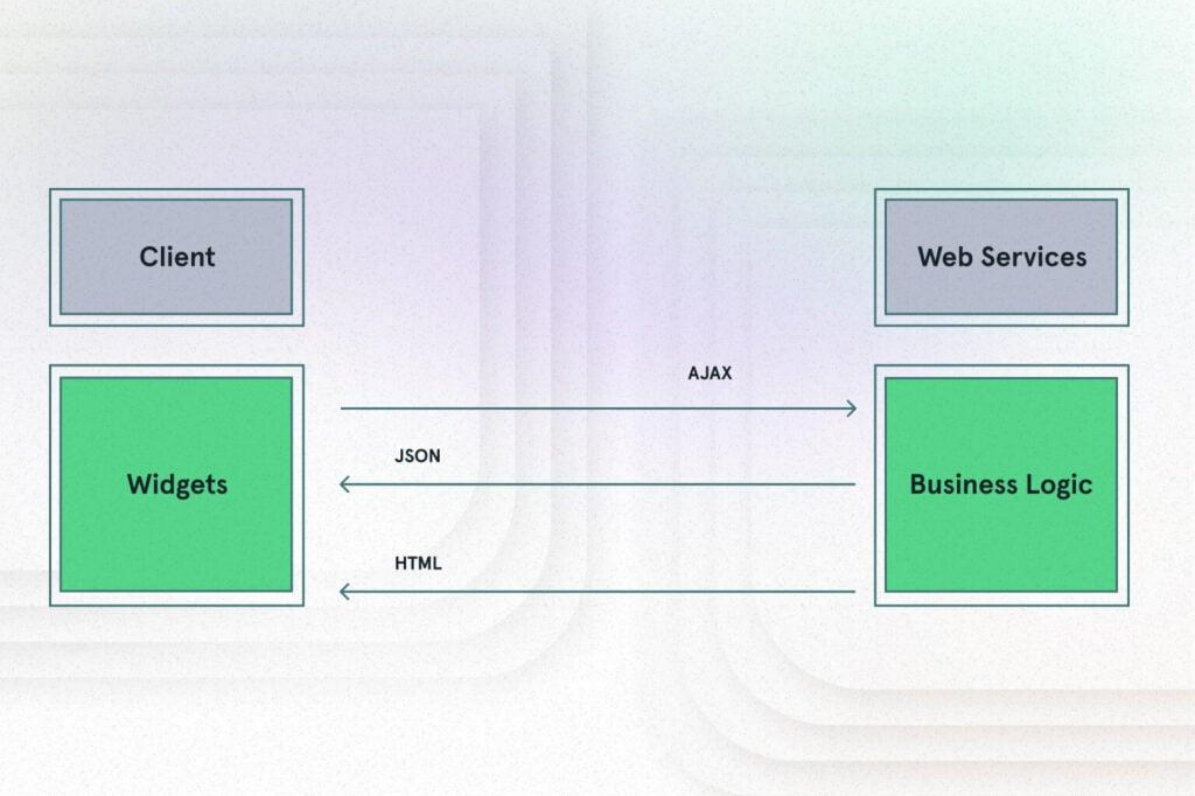


Рисунок 1 — Загальна схема роботи PWA

Ключовим принципом побудови PWA є прогресивне покращення. Це означає, що базова версія сайту повинна бути доступною для будь-якого користувача, незалежно від типу пристрою чи браузера, який він використовує. При цьому, більш сучасні функції, такі як офлайн-режим або push-повідомлення, активуються лише тоді, коли технічні можливості браузера це дозволяють. Такий підхід дозволяє створювати універсальні застосунки, які забезпечують стабільну

роботу на більшості платформ і поступово розширюють функціональність там, де це можливо.

Оцінюючи ефективність PWA у контексті електронної комерції, важливо враховувати не лише технічні аспекти, а й вплив на поведінку користувачів. Це пояснюється покращеним досвідом користувача, зокрема за рахунок швидкої взаємодії, інтерфейсу, схожого на нативні застосунки, та можливості продовжувати роботу без з'єднання з мережею. Такі показники роблять PWA привабливою альтернативою як для малих підприємств, так і для великих платформ електронної комерції.

З теоретичного погляду, PWA не є окремою технологією, а радше концепцією або сукупністю веб-стандартів, які в сукупності забезпечують нову якість взаємодії користувача із сайтом. Це означає, що для створення прогресивного веб-додатку розробнику не потрібно опановувати окрему мову програмування чи фреймворк - достатньо використання сучасного JavaScript, HTML5 і CSS3 у поєднанні з практиками оптимізації продуктивності, безпеки та адаптивного дизайну. Такий підхід робить технологію доступною для великого кола розробників і сприяє її активному поширенню.

Загалом, архітектура PWA орієнтована на користувача - як у плані зручності, так і в контексті продуктивності. Особлива увага приділяється тому, щоб зменшити час до взаємодії, мінімізувати обсяг переданих даних та адаптувати інтерфейс під специфіку пристрою. У той же час, прогресивні веб-додатки повинні відповідати вимогам безпеки, зокрема використовувати протокол HTTPS для обміну даними, що є обов'язковою умовою для роботи технології. Таким чином, реалізація PWA вимагає не лише технічної грамотності, а й стратегічного підходу до дизайну та розгортання продукту.

У контексті еволюції цифрового середовища, PWA займає проміжну позицію між класичними веб-сайтами та нативними застосунками. З одного боку,

вона зберігає універсальність веб-технологій - кросплатформеність, відсутність потреби у завантажені з маркетплейсу додатків, миттєвий доступ через URL. З іншого боку, вона наближається до мобільних застосунків за рахунок роботи в офлайн режимі, push-сповіщень та можливості роботи у фоновому режимі. Ця гібридна архітектура дозволяє задовольнити потреби сучасного користувача, який очікує як зручності, так і швидкодії.

Технологія PWA також підтримується провідними браузерами, такими як Google Chrome, Mozilla Firefox, Safari та Microsoft Edge, що створює сприятливе середовище для масового впровадження. Однак, не всі функції доступні в однаковому обсязі в різних браузерах, що вимагає врахування сумісності під час розробки. Наприклад, обмежена підтримка push-повідомлень у Safari на iOS довгий час стримувала поширення PWA на цій платформі. Тим не менше, загальна тенденція є позитивною, і розробники браузерів поступово розширюють можливості для повної підтримки прогресивних веб-додатків [4].

Підсумовуючи теоретичні основи технології PWA, варто наголосити, що її розвиток став логічною відповіддю на зростаючі вимоги ринку до швидкості, зручності та універсальності веб-рішень. Вона поєднує кращі практики веб-розробки з функціональністю нативних застосунків, забезпечуючи новий рівень взаємодії між користувачем і брендом. В умовах зростання конкуренції в електронній комерції, саме такі технології здатні забезпечити вирішальні конкурентні переваги, дозволяючи бізнесам адаптуватися до нових моделей споживання цифрового контенту.

1.2 Критерії вибору платформи для PWA

Вибір платформи для реалізації прогресивного веб-додатку (PWA) є одним із ключових етапів, який визначає не лише технологічні аспекти проєкту, але й

загальний вектор його розвитку. Успішність PWA безпосередньо залежить від здатності платформи забезпечити необхідний рівень продуктивності, адаптивності, сумісності з різними пристроями та браузерами, а також підтримки сучасних веб-технологій. В умовах високої конкуренції у сфері електронної комерції та веб-сервісів платформа повинна гарантувати швидке завантаження, офлайн-доступ, інтеграцію з нативними функціями пристроїв і масштабованість, що є базовими ознаками PWA.

Одним з головних критеріїв є відповідність платформи стандартам PWA, зокрема підтримка Service Worker, Web App Manifest і HTTPS. Ці елементи є технічною основою прогресивного вебдодатку, адже без них реалізація офлайн-функціональності, push-сповіщень і встановлення застосунку на головний екран пристрою неможлива [5]. Платформа, яка не забезпечує автоматичне створення або інтеграцію цих компонентів, ускладнює процес розробки та підвищує вимоги до розробника. Саме тому важливо оцінювати наявність вбудованої підтримки цих технологій або можливості їхньої легкої інтеграції.

Суттєвим чинником вибору є продуктивність рендерингу інтерфейсу та швидкість обробки запитів. Наприклад, застосування PWA в таких платформах як Magento 2 або Shopify дозволяє досягти значного зменшення часу завантаження сторінки за рахунок використання попереднього кешування та асинхронного завантаження даних. У випадку Magento, технологія PWA Studio дозволяє розділяти логіку клієнта й сервера, що зменшує навантаження на бекенд і підвищує швидкість взаємодії користувача із сайтом. З іншого боку, WooCommerce вимагає більше зусиль для реалізації аналогічних функцій, адже базується на WordPress, де деякі компоненти слід налаштовувати вручну.

Наступним критерієм виступає можливість кастомізації інтерфейсу та гнучкість у розробці функціоналу. Платформа повинна дозволяти адаптувати користувацький інтерфейс відповідно до специфіки бізнесу та поведінки цільової

аудиторії. Наприклад, Vue Storefront як незалежна PWA-платформа забезпечує максимальну кастомізацію, однак вимагає глибоких знань JavaScript і архітектури REST або GraphQL API. У порівнянні з цим, Shopify з PWA-темами більш обмежений у можливостях зміни базової структури, але значно простіший для швидкого старту малого або середнього бізнесу.

Безпека – ще один ключовий аспект, що впливає на вибір платформи. Оскільки PWA працює через HTTPS-протокол, платформа має підтримувати сертифікати безпеки та механізми запобігання вразливостям, таким як XSS, CSRF і доступ до Web API. Magento 2, будучи платформою корпоративного рівня, містить вбудовані засоби захисту та механізми авторизації для PWA-компонентів. У випадку відкритих CMS, таких як WordPress, забезпечення безпеки залежить від сторонніх плагінів, що підвищує ризик появи уразливостей.

Сумісність з пристроями та браузерами - ще один критерій, що не можна ігнорувати. Успішна реалізація PWA повинна коректно працювати як на Android, так і на iOS. Проте, деякі платформи мають обмеження через політику браузерів. Наприклад, Safari частково підтримує Service Worker, що може вплинути на функціональність застосунку, створеного на певних платформах. У цьому контексті важливо обирати платформу, яка тестується на широкому спектрі пристроїв та оновлюється з урахуванням змін у політиках браузерів.

Не менш важливою є підтримка API та можливість інтеграції зі сторонніми сервісами. Відкритість архітектури платформи та доступ до REST або GraphQL API дозволяє налаштовувати складні ланцюги обміну даними, автоматизувати процеси та реалізовувати функції, що виходять за межі стандартного функціоналу. Magento 2 та Vue Storefront вирізняються гнучкістю інтеграції, тоді як інші системи потребують використання окремих додатків або плагінів, що не завжди відповідає очікуванням щодо продуктивності.

Важливим чинником для багатьох компаній є доступність розробників та документації. Якщо платформа має активну спільноту, добре структуровану документацію, приклади використання і підтримку, це суттєво зменшує витрати на розробку та спрощує процес навчання нових членів команди. Наприклад, платформи Magento 2 і Shopify мають великі професійні спільноти та багату базу знань, що дозволяє швидко вирішувати технічні проблеми та впроваджувати нові функції.

Останній, але не менш важливий аспект - комерційна доцільність вибору. Платформа повинна відповідати бюджету проєкту, його масштабу та планам щодо розвитку. Наприклад, реалізація PWA на базі Magento є доцільною для великих проєктів з високими вимогами до продуктивності й кастомізації, але вона вимагає значних ресурсів. Натомість WooCommerce із PWA-плагінами або Shopify із готовими PWA-темами можуть бути більш привабливими для малого бізнесу через нижчі витрати на старті.

Таким чином, вибір платформи для реалізації прогресивного веб-додатку є багатофакторним процесом, який охоплює технічні, безпекові, економічні та користувацькі аспекти. Визначення оптимального варіанту вимагає глибокого аналізу специфіки проєкту, його цілей і ресурсних можливостей. Успішна реалізація PWA можлива лише за умови відповідності платформи сучасним технологічним вимогам, її гнучкості, стабільності, а також здатності підтримувати розвиток проєкту в довгостроковій перспективі.

1.3 Практичні аспекти використання технології PWA

Практичні аспекти використання технології Progressive Web Application визначаються її здатністю поєднувати переваги веб-платформ і мобільних застосунків, забезпечуючи універсальний та адаптивний доступ до цифрових

сервісів незалежно від пристрою чи умов мережевого підключення. Однією з ключових характеристик PWA є можливість функціонування в середовищах зі зниженою якістю інтернет-зв’язку, що значно розширює сферу їх практичного застосування. Завдяки механізмам офлайн-кешування, фонової синхронізації та локального зберігання даних, користувачі отримують доступ до основних функцій навіть за відсутності стабільного з’єднання, що робить технологію придатною для регіонів із нерозвиненою інфраструктурою або для сервісів, які вимагають постійної доступності.

У практичній діяльності організацій важливою перевагою PWA є мінімізація бар’єрів доступу: для взаємодії не потрібно завантажувати застосунок з магазину, проходити додаткову верифікацію чи витрачати місце в пам’яті мобільного пристрою. Завдяки можливості встановлення PWA на домашній екран одним натисканням, користувач отримує досвід, максимально подібний до роботи з нативним застосунком, проте без додаткових інсталяційних процедур. Це створює сприятливі умови для підвищення охоплення аудиторії, покращення показників утримання та зменшення кількості відмов — факторів, що мають суттєве значення для систем з високою конкуренцією, зокрема у сфері електронної комерції.

Важливою характерною рисою практичного застосування PWA є їх здатність забезпечувати високу швидкодію та плавність роботи завдяки попередньому кешуванню статичних ресурсів, оптимізації мережевих запитів та розподіленню обчислювальних навантажень між клієнтською та серверною частинами. У реальних умовах це призводить до скорочення часу початкового завантаження, пришвидшення переходів між сторінками та підвищення загального рівня задоволеності користувачів взаємодією з системою. Для бізнесу це означає можливість обслуговувати більшу кількість запитів без необхідності

масштабування серверної інфраструктури, що знижує експлуатаційні витрати та підвищує економічну ефективність.

У контексті сучасних інформаційних систем PWA часто застосовуються як інструмент оптимізації корпоративних процесів. Завдяки універсальності веб-платформи та можливості доступу з будь-якого пристрою, такі застосунки успішно інтегруються у внутрішні системи підприємств, включаючи сервіси моніторингу, CRM-платформи, системи управління логістикою та інформаційно-аналітичні панелі. Формат PWA дозволяє скоротити час упровадження нових інструментів, спростити навчання персоналу та забезпечити гнучку адаптацію до зміни бізнес-процесів без необхідності створення окремих мобільних застосунків під різні операційні системи.

У практиці роботи з користувацькими сервісами особливе значення мають можливості персоналізації, push-сповіщень і інтеграції з апаратними ресурсами пристроїв, такими як камера, геолокація, мікрофон чи датчики руху. Хоча набір доступних API у браузері дещо обмежений порівняно з нативними платформами, сучасні PWA все частіше використовують можливості Web Push API, Background Sync, WebAuthn та інших технологій, що розширюють функціональність додатка та дозволяють формувати повноцінний мобільний досвід. Практичний ефект полягає у підвищенні залученості користувачів, поліпшенні показників взаємодії та можливості налагодження двосторонньої комунікації в режимі реального часу.

Окремий практичний аспект стосується можливостей масштабування та кросплатформності. Оскільки PWA базуються на універсальних веб-стандартах, один і той самий застосунок може без модифікацій працювати на настільних, мобільних та вбудованих системах, що значно спрощує розвиток проєктів та оптимізацію витрат на розробку. Це дозволяє організаціям із обмеженими фінансовими чи кадровими ресурсами реалізовувати цифрові рішення, які раніше потребували складної екосистеми нативних застосунків.

З практичної точки зору, PWA також сприяють поліпшенню видимості в пошукових системах, оскільки залишаються повноцінними веб-ресурсами з можливістю індексації. Це створює важливий конкурентний бонус у порівнянні з нативними застосунками, які не можуть бути знайдені за допомогою класичного пошуку в інтернеті [7]. Поєднання SEO-оптимізації та мобільних функцій створює особливо сприятливі умови для сервісів, орієнтованих на велику кількість органічного трафіку.

Таким чином, практичні аспекти використання технології PWA охоплюють широкий спектр прикладних сценаріїв — від електронної комерції та соціальних платформ до корпоративних сервісів, освітніх ресурсів та геолокаційних систем. Їх універсальність, доступність, здатність функціонувати в офлайн-режимі та підтримувати мобільні можливості роблять PWA одним із найперспективніших підходів до побудови сучасних веб-орієнтованих рішень, здатних задовольнити потреби як кінцевих користувачів, так і організацій різного масштабу.

1.4 Концепція Progressive Web Application та її ключові характеристики

PWA виникла як відповідь на зростаючу потребу у створенні веб-застосунків, здатних забезпечувати користувацький досвід, співставний із нативними мобільними застосунками, але при цьому залишатися універсальними, легкодоступними та економічно ефективними. PWA представляють собою нову парадигму веб-розробки, у якій використання сучасних браузерних API, механізмів кешування та адаптивного дизайну дозволяє створювати застосунки, що поєднують гнучкість вебу та функціональність мобільних платформ. Основою концепції є ідея прогресивного покращення — застосунок має працювати у будь-якому стандартному браузері,

але за наявності підтримуваних API поступово активує додаткові функції, роблячи досвід користувача повноціннішим.

Ключовим елементом концепції PWA є принцип прогресивності. Він передбачає, що застосунок доступний у найпростішому вигляді навіть за мінімальних технічних можливостей клієнтського середовища, а за наявності сучасного браузера та мережевих можливостей — автоматично забезпечує розширені функції. Такий підхід робить PWA універсальними та стійкими до змін технічного ландшафту. Застосунок не залежить від конкретної операційної системи, не вимагає встановлення через магазин застосунків і забезпечує доступність у будь-якої точки світу через стандартний веб-браузер.

Однією з фундаментальних характеристик PWA є можливість встановлення застосунку на домашній екран пристрою. Це створює відчуття роботи з повноцінним мобільним застосунком, оскільки після встановлення PWA запускається у власному вікні без адресного рядка браузера, забезпечуючи більш занурений досвід. Крім того, наявність Web App Manifest — спеціального конфігураційного файлу, який описує назву, іконки, кольорову палітру та режим відображення застосунку — дозволяє формувати цілісний візуальний бренд і стандартизувати поведінку застосунку на різних платформах.

Іншою визначальною характеристикою є автономність роботи. Завдяки використанню Service Worker — спеціального скрипта, який працює у фоновому режимі та перехоплює мережеві запити — застосунок може кешувати ресурси, працювати офлайн або в умовах нестабільного інтернет-з’єднання, синхронізувати дані у фоновому режимі та оптимізувати обмін інформацією з сервером. Це дозволяє значно підвищити швидкодію, зменшити залежність від мережевої інфраструктури та забезпечити безперервність користувацького досвіду. Для електронної комерції, де кожна затримка може вплинути на конверсію, саме ця характеристика PWA вважається однією з ключових.

Важливою складовою концепції PWA є безпечне виконання. Усі PWA мають працювати виключно через протокол HTTPS, що гарантує цілісність переданих даних, захищеність мережевих запитів і можливість використання розширених API браузера. Прив’язка до безпечного контексту є важливою передумовою довіри користувачів та забезпечення відповідності сучасним вимогам інформаційної безпеки. Використання HTTPS також є обов’язковим для активації таких можливостей, як push-сповіщення, фонове оновлення даних і доступ до апаратних функцій пристрою.

Суттєвим аспектом концепції PWA є їх продуктивність. Завдяки продуманим механізмам кешування, мінімізації мережевих звернень, попередньому завантаженню статичних ресурсів і оптимізованому рендерингу інтерфейсу, сучасні PWA здатні забезпечувати дуже швидкий час завантаження та високу чутливість до взаємодії користувача. Саме висока продуктивність, згідно з численними дослідженнями Google, є визначальним фактором у підвищенні задоволеності користувачів, зменшенні показників відмов і збільшенні кількості завершених транзакцій у комерційних системах.

Ще однією важливою характеристикою є незалежність від платформи. PWA здатні працювати на будь-якому пристрої — від смартфонів і планшетів до ноутбуків та настільних комп’ютерів — без необхідності створення окремих версій застосунку. Адаптивні інтерфейси дозволяють системі гнучко підлаштовуватися під розмір екрану, а використання єдиної кодової бази значно зменшує витрати на розробку та підтримку. На відміну від нативних застосунків, PWA не потребують публікації в App Store чи Google Play, що спрощує оновлення та мінімізує затримки, пов’язані з модерацією.

Таким чином, концепція Progressive Web Application базується на поєднанні технологічних інновацій веб-платформи та підходів, характерних для мобільних застосунків. Її ключові характеристики — прогресивність,

автономність роботи, кросплатформність, безпечність, адаптивність та висока продуктивність — формують основу нового покоління веб-застосунків, здатних забезпечувати універсальний і водночас високоякісний користувацький досвід. Завдяки цим властивостям PWA стрімко інтегруються в різні галузі, стаючи ефективною альтернативою традиційним веб- і мобільним рішенням та визначаючи подальший розвиток сучасних інформаційних систем.

1.5 Архітектура PWA: Service Workers, Web App Manifest, офлайн-кешування

Архітектура Progressive Web Application ґрунтується на концепції розширення можливостей традиційного веб-застосунку шляхом використання спеціалізованих браузерних механізмів, які дозволяють забезпечити автономність роботи, високу швидкодію та поведінку, подібну до нативних мобільних застосунків. Центральними елементами цієї архітектури є Service Workers, Web App Manifest та системи офлайн-кешування, які разом формують інфраструктуру, що дозволяє застосунку працювати в умовах нестабільного інтернет-з’єднання, швидко завантажуватися та надавати користувацький досвід, характерний для платформ Android та iOS.

Ключовою складовою архітектури PWA є Service Worker — незалежний фоновий скрипт, який виконується поза контекстом веб-сторінки і має доступ до широкого набору браузерних API. Service Worker працює асинхронно та здатний перехоплювати мережеві запити, маніпулювати кешем, виконувати синхронізацію даних у фоновому режимі та приймати push-сповіщення. Саме він забезпечує можливість функціонування застосунку в офлайн-режимі та підтримку високої продуктивності через оптимізоване використання мережевих ресурсів. Завдяки механізму подій, таких як install, activate, fetch та sync, Service

Worker надає розробникам гнучкі інструменти для реалізації стратегій кешування — від простого кешування статичних ресурсів до складних адаптивних алгоритмів, що реагують на стан мережі та пріоритети інформації.

Другим фундаментальним елементом архітектури PWA є Web App Manifest — конфігураційний JSON-файл, який описує візуальні та функціональні характеристики застосунку. Він містить інформацію про назву PWA, іконки різних розмірів, колірну схему інтерфейсу, початковий екран (splash screen), орієнтацію екрана та режим відображення (standalone, fullscreen, minimal-ui). Завдяки маніфесту браузер може визначити, як саме застосунок повинен поводитися під час встановлення на домашній екран, та забезпечити користувачу досвід, максимально наближений до використання нативних застосунків. Web App Manifest є важливим інструментом брендового оформлення, оскільки дозволяє створювати цілісний візуальний стиль та забезпечує уніфікованість у різних браузерах і на різних операційних системах.

Офлайн-кешування є третьою опорою архітектури PWA і забезпечує стійкість застосунку до обмежень мережевого середовища. У традиційних веб-застосунках робота повністю залежить від доступності сервера, тоді як у PWA основні ресурси (HTML, CSS, JavaScript, зображення) можуть бути збережені в локальному кеші браузера, що дозволяє завантажувати застосунок миттєво навіть без підключення до мережі. Механізми кешування можуть бути реалізовані за допомогою Cache API у поєднанні з Service Worker, що дає змогу створювати гнучкі стратегії обробки запитів. Схему взаємодії PWA з системою кешування представлено у Рисунку 2. Стратегія cache-first надає максимальну швидкість, тоді як network-first гарантує отримання найновіших даних. Комбіновані підходи дозволяють адаптувати поведінку застосунку до реальних умов використання та забезпечують ефективну роботу навіть у складних мережевих сценаріях.

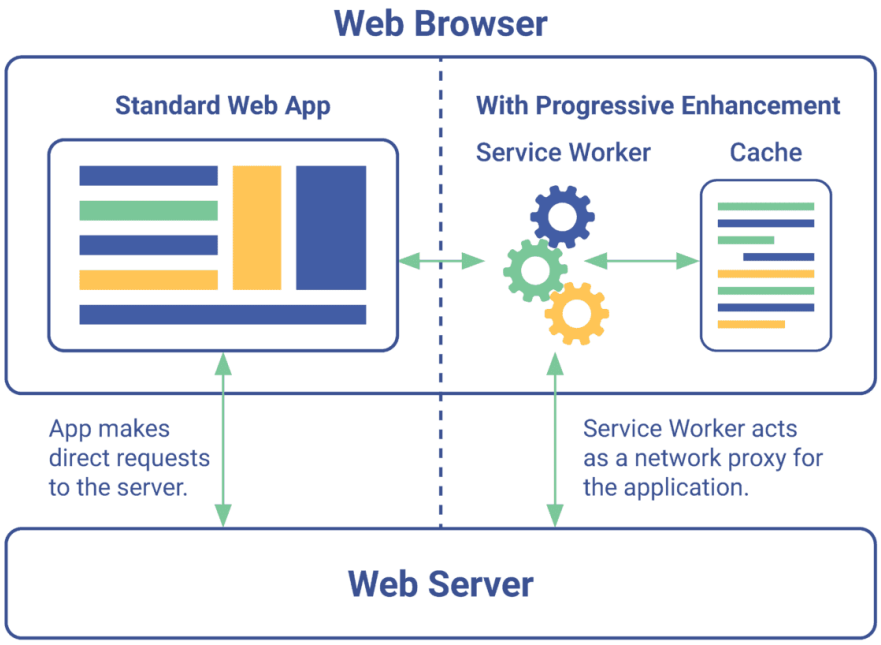


Рисунок 2 — Архітектура взаємодії PWA з серверною частиною із використанням Service Worker та кешування

Окрему роль в архітектурі PWA відіграє здатність Service Worker оновлювати дані у фоновому режимі. Навіть у тих випадках, коли застосунок працює офлайн, Service Worker може синхронізувати зміни після відновлення інтернет-з'єднання, що критично важливо для застосунків електронної комерції, де актуальність інформації (ціни, наявність товарів, статуси замовлень) є пріоритетною. Фонове оновлення дозволяє мінімізувати затримки та уникнути ситуацій, коли користувач отримує застарілі дані.

Слід також зазначити, що архітектура PWA базується на принципі безпечного виконання: Service Worker може працювати лише в контексті протоколу HTTPS. Це обмеження впроваджене для захисту даних користувачів, оскільки Service Worker має доступ до мережевих запитів та кешу, і його використання у незахищеному середовищі створювало б значні ризики. Таким чином, вимога HTTPS не тільки гарантує безпеку, а й сприяє підвищенню довіри користувачів до сервісу.

Архітектура PWA вирізняється модульністю та гнучкістю, що дозволяє застосункам адаптуватися до різних пристроїв та сценаріїв використання. Використання сучасних веб-стандартів забезпечує широке охоплення аудиторії без необхідності створення окремих версій під різні платформи. Водночас архітектурні компоненти PWA взаємодіють між собою таким чином, щоб формувати стійку, продуктивну та інтуїтивну систему, здатну забезпечувати високоякісний користувацький досвід у реальних умовах експлуатації.

У комплексі Service Workers, Web App Manifest та механізми офлайн-кешування створюють гнучку та потужну архітектурну модель, яка дозволяє Progressive Web Application виходити за межі традиційних веб-технологій та забезпечувати функціональність, характерну для нативних мобільних застосунків. Саме завдяки цим технологічним складовим PWA здатні надавати швидкий, надійний і зручний доступ до інформаційних систем, що робить їх одним із найперспективніших напрямів розвитку сучасного веб-середовища.

1.6 Безпекові та продуктивні аспекти PWA у контексті електронної комерції

У сфері електронної комерції питання безпеки та продуктивності є ключовими факторами, від яких залежить не лише стабільність роботи цифрових сервісів, а й рівень довіри користувачів та конкурентоспроможність бізнесу. Progressive Web Applications, поєднуючи у собі веб-технології та можливості нативних застосунків, повинні забезпечувати високий ступінь захисту даних, безперервність функціонування та швидкість взаємодії з користувачем. Саме тому безпекові та продуктивні аспекти становлять фундаментальну складову архітектури PWA, визначаючи їх практичну придатність у комерційних середовищах, де обробляється конфіденційна інформація та здійснюються фінансові операції.

Однією з базових вимог до безпеки PWA є функціонування виключно у межах захищеного протоколу HTTPS. Це обмеження є принциповим, оскільки застосунки цього типу використовують Service Worker — механізм, який перехоплює та модифікує мережеві запити, керує кешем і взаємодіє з чутливими даними. Робота Service Worker у незашифрованому середовищі створювала б потенційні можливості для атак типу man-in-the-middle та маніпулювання кешованими ресурсами. HTTPS забезпечує цілісність переданих даних, захищає від перехоплення мережевого трафіку, унеможливлює підробку відповідей сервера та створює безпечний контекст для активування розширених браузерних API. У контексті електронної комерції, де обробляються персональні дані користувачів та платіжна інформація, такі вимоги є критично важливими.

Безпековий аспект архітектури PWA також включає обмежений доступ до апаратних функцій пристрою. Оскільки PWA запускаються через браузер, вони підпорядковані моделі безпеки веб-середовища, у рамках якої доступ до ресурсів,

таких як камера, геолокація чи датчики, можливий лише за прямої згоди користувача. Це знижує ризики несанкціонованого збору даних, характерного для деяких нативних застосунків, і сприяє прозорості взаємодії з користувачем. Крім того, політики Content Security Policy, sandboxing та ізоляція походжень (Same-Origin Policy) забезпечують контроль над тим, як завантажуються та виконуються зовнішні ресурси, запобігаючи атакам типу XSS, CSRF і використанню шкідливого коду.

Не менш важливою складовою безпеки є правильна реалізація стратегій кешування. Хоча кешування є необхідним для забезпечення офлайн-функціональності та підвищення швидкодії, неправильна конфігурація може стати джерелом серйозних ризиків. Наприклад, випадкове кешування приватних даних, таких як профілі користувачів, історія замовлень чи кошики покупок, може призвести до розкриття конфіденційної інформації на спільних пристроях або між різними сесіями. Тому налаштування кешу в PWA повинно включати чіткі політики доступу, вибіркове кешування публічних ресурсів і динамічне оновлення даних, що мають короткий термін актуальності.

Продуктивність PWA є не менш важливим фактором у контексті електронної комерції, де швидкість роботи системи прямо впливає на конверсію та рівень задоволеності користувачів. PWA значно перевершують традиційні веб-застосунки завдяки використанню попереднього завантаження статичних ресурсів, оптимізованих стратегій кешування та можливості працювати без постійного звернення до сервера. У реальному середовищі це означає, що навіть складні сторінки, такі як каталоги товарів, форму кошика чи індивідуальні картки продуктів, завантажуються майже миттєво, забезпечуючи плавний і чутливий інтерфейс.

Service Worker відіграє важливу роль у підвищенні продуктивності, оскільки він може обробляти мережеві запити локально через кеш, минаючи

затримки, пов’язані з мережею. У випадках, коли дані потребують оновлення, Service Worker здатен паралельно отримати актуальний контент і оновити кеш після відповіді від сервера, не блокуючи користувацький інтерфейс. Такий підхід формує ілюзію високої швидкодії та створює відчуття безперервності роботи, подібне до нативних мобільних застосунків. У середовищі, де контроль за продуктивністю є критичним, цей механізм забезпечує конкурентні переваги над традиційними підходами до веб-розробки.

Важливою частиною продуктивності PWA є їх інтеграція з сучасними інструментами оптимізації завантаження. Прогресивне завантаження ресурсів, lazy loading зображень, мікрооптимізації DOM, мінімізація мережевих запитів та агресивне кешування дозволяють значно зменшити час відображення контенту. У системах електронної комерції, де кількість ресурсів може бути великою (зображення товарів, скрипти аналітики, стилі), ці підходи дозволяють підтримувати стабільну продуктивність навіть під час пікових навантажень.

Продуктивність PWA також позитивно впливає на поведінкові метрики користувачів, такі як час до першої взаємодії (Time to Interactive), швидкість відповіді інтерфейсу та стабільність кадрів під час анімації. Для бізнесу це виражається у зменшенні показника bounce rate, збільшенні тривалості сесії та вірогідності здійснення покупки. Для багатьох компаній перехід на PWA уже продемонстрував значні зрушення в ефективності — зростання мобільної конверсії, покращення швидкодії та стабільності роботи на слабких пристроях.

Таким чином, у контексті електронної комерції безпекові та продуктивні аспекти PWA мають вирішальне значення для формування надійних і конкурентоспроможних цифрових сервісів. Використання захищених протоколів, обмеження доступу до критичних ресурсів, грамотна реалізація кешування та оптимізована робота Service Worker забезпечують високий рівень захисту, необхідний для обробки конфіденційних даних. Водночас сучасні

підходи до продуктивності дозволяють досягти швидкості та плавності роботи, що раніше були притаманні лише нативним мобільним застосункам. Завдяки поєднанню цих характеристик PWA стають одним із найефективніших рішень для електронної комерції, забезпечуючи бізнесу стабільність, масштабованість і високий рівень користувацького задоволення.

1.7 Висновки до розділу 1

У першому розділі було здійснено порівняльний та комплексний аналіз можливостей інтеграції технології Progressive Web Application у сучасні системи електронної комерції. Розгляд теоретичних засад технології PWA дозволив установити, що її поява є логічною еволюцією розвитку веб-середовища, яке переходить від класичних статичних інтерфейсів до динамічних, адаптивних і високопродуктивних застосунків, здатних забезпечувати мобільний користувацький досвід на будь-якому пристрої.

Визначені критерії вибору платформи для впровадження PWA підкреслюють важливість врахування архітектури серверної частини, підтримки сучасних веб-API, можливостей масштабування та вимог до безпеки. Порівняльний аналіз показав, що різні платформи електронної комерції — такі як Magento, Shopify, WooCommerce та інші — мають різний рівень готовності до нативної інтеграції PWA, що впливає на обсяг технічних робіт, складність інтеграційних рішень та загальну ефективність впровадження.

Практичні аспекти використання PWA засвідчили широкі можливості застосунку цієї технології у сферах із високими вимогами до доступності, швидкодії, зручності та стабільності. Завдяки можливості інсталяції на домашній екран, офлайн-роботі, фоновому оновленню, push-сповіщенням та адаптивності інтерфейсу PWA здатні забезпечити користувацький досвід, який наближається

до нативних застосунків, при цьому суттєво знижуючи витрати на підтримку та розробку.

Розгляд ключових характеристик PWA дав змогу сформувати цілісне уявлення про їхні переваги: прогресивність, незалежність від платформи, високу продуктивність, безпечність, адаптивність та ефективну взаємодію з апаратними можливостями пристрою. Ці характеристики зумовлюють конкурентні переваги PWA порівняно з традиційними веб- та мобільними рішеннями, особливо у високонавантажених комерційних системах.

Аналіз архітектури PWA — зокрема компонентів Service Worker, Web App Manifest та механізмів офлайн-кешування — показав, що саме вони формують технологічну основу, яка забезпечує автономність роботи, швидке завантаження та стійкість до нестабільного інтернет-з'єднання. Така архітектура створює додаткові можливості для оптимізації серверних навантажень та забезпечення більш прогнозованої взаємодії користувачів із застосунком.

У контексті електронної комерції особливе значення мають безпекові та продуктивні аспекти PWA. Обов’язкове використання HTTPS, контрольоване кешування, обмежений доступ до апаратних функцій та підтримка браузерних політик безпеки формують цілісний підхід до захисту даних і забезпечують відповідність сучасним стандартам інформаційної безпеки. Водночас оптимізовані стратегії кешування, швидке рендеринг-оцінювання ресурсів і робота Service Worker суттєво покращують продуктивність системи та позитивно впливають на комерційні показники, включаючи конверсію та утримання користувачів.

Узагальнюючи результати, можна стверджувати, що технологія PWA є перспективним інструментом розвитку електронної комерції, який поєднує у собі технічну ефективність, високу продуктивність, економічну доцільність та відповідність сучасним вимогам користувачів. Отримані у цьому розділі

висновки формують теоретичне й методологічне підґрунтя для подальшого дослідження інтеграції PWA у різні архітектурні моделі та платформні рішення, що розглядатимуться у наступних розділах дипломної роботи.

РОЗДІЛ 2 ПРАКТИЧНІ ПІДХОДИ ТА ТЕХНОЛОГІЧНІ РІШЕННЯ ДЛЯ РЕАЛІЗАЦІЇ PWA У СИСТЕМАХ ЕЛЕКТРОННОЇ КОМЕРЦІЇ

2.1 Загальні підходи до реалізації PWA у веб-системах

Реалізація Progressive Web Application у сучасних веб-системах ґрунтується на поєднанні низки технологічних підходів, що дозволяють перетворити традиційний веб-застосунок на високопродуктивний, автономний та інтегрований мобільний досвід. На відміну від класичних сайтів, де браузер кожного разу завантажує ресурси з сервера, PWA використовують механізми кешування, фонової синхронізації, попереднього рендерингу та розширені API браузера. Це дозволяє значно підвищити швидкодію та забезпечити роботу навіть за умов нестабільного інтернет-з’єднання. На практиці існує три основні підходи до впровадження PWA, які відрізняються рівнем інтеграції, складністю реалізації та вимогами до серверної інфраструктури.

Перший підхід полягає у додаванні PWA-функціональності поверх існуючого веб-сайту. Ця модель є найпростішою, оскільки вона не потребує суттєвої зміни архітектури системи та підходить для сайтів, що вже працюють у продуктивному середовищі. Інтеграція передбачає підключення Web App Manifest, створення базового Service Worker для кешування статичних ресурсів, а також реалізацію мінімальної офлайн-функціональності. Такий підхід дозволяє швидко отримати основні переваги PWA — прискорене завантаження, роботу в офлайн-режимі та можливість встановлення застосунку на домашній екран. Він часто використовується у проєктах, де пріоритетом є мінімальні витрати часу та ресурсів, або у випадках, коли CMS чи e-commerce платформа має значні обмеження щодо доступу до внутрішньої логіки.

Другий підхід базується на частковій реалізації PWA, коли лише певні компоненти системи отримують функціональність прогресивного застосунку. Це може стосуватися, наприклад, каталогу товарів, форми реєстрації, корзини, сторінок блогу чи особистого кабінету. У цьому випадку Service Worker використовується для вибіркового кешування конкретних даних або динамічних API-запитів, а сама система зберігає класичну модель рендерингу через сервер. Підхід є ефективним, коли потрібно покращити найбільш навантажені або критичні частини сайту, не перевантажуючи архітектуру повним переходом до headless-моделі. Така практика поширена серед великих e-commerce проєктів, де ризики зміни архітектури повинні бути мінімізовані, але є потреба в підвищенні швидкодії та доступності.

Третій і найбільш технологічно просунутий підхід — це реалізація повноцінного PWA у форматі headless-архітектури. У такій моделі фронтенд повністю відокремлений від бекенду та функціонує як незалежний застосунок, який взаємодіє з сервером через API (REST або GraphQL). Цей підхід передбачає використання сучасних фреймворків, таких як React, Next.js, Vue.js, Nuxt.js або Angular, та розгорнення складного механізму кешування, маршрутизації та серверного рендерингу. Перевагою headless-підходу є можливість повної оптимізації роботи клієнтської частини, масштабування системи, гнучке використання хмарних сервісів, а також створення мобільного досвіду, максимально наближеного до нативного застосунку. Він є оптимальним для великих платформ електронної комерції, де важливими є швидкість реакції системи, модульність і здатність обробляти високі навантаження.

Важливо зазначити, що всі три підходи вимагають дотримання спільних технічних умов: використання протоколу HTTPS, наявності файлу Web App Manifest, коректної реалізації Service Worker та оптимізацій ресурсів. Основним завданням розробника є визначення того, який рівень інтеграції PWA відповідає

цілям бізнесу, технічним можливостям платформи та доступним ресурсам. Наприклад, для великого інтернет-магазину на Magento доцільно застосовувати headless-модель для отримання максимальної продуктивності, тоді як для невеликого сайту на WooCommerce достатньо базового Service Worker з кешуванням.

Таким чином, загальні підходи до реалізації PWA у веб-системах охоплюють широкий спектр технічних стратегій — від мінімалістичного покращення існуючого сайту до повноцінної побудови автономного PWA-застосунку на основі сучасних фронтенд-фреймворків. Вибір конкретного підходу залежить від архітектури системи, доступних інтеграційних можливостей, вимог до продуктивності та масштабу проєкту. Саме ці принципи визначають подальший аналіз реалізації PWA в Magento 2, Shopify та WooCommerce, що розглядатиметься у наступних підрозділах.

2.2 Технологічні інструменти та фреймворки для розробки PWA

Розробка Progressive Web Application спирається на широкий набір технологій, інструментів та фреймворків, які забезпечують можливості для швидкої реалізації, масштабування та підтримки прогресивного застосунку. Ринок веб-розробки пропонує різноманітні рішення — від низькорівневих утиліт до повноцінних платформ, орієнтованих на створення складних e-commerce застосунків у headless-архітектурі. Вибір конкретного інструментарію залежить від складності проєкту, вимог до продуктивності, інтеграції з бекендом та наявності спеціалізованих API.

Одним із ключових інструментів, що використовується при створенні PWA, є Workbox — набір бібліотек від Google, який значно спрощує роботу з Service Worker та Cache API. Workbox надає високорівневі допоміжні модулі для

реалізації стратегій кешування, фонової синхронізації, обробки мережевих запитів і автоматичної генерації Service Worker під час збірки проєкту. Завдяки цьому розробники можуть зосередитися на логіці застосунку, не витрачаючи час на низькорівневу реалізацію складних механізмів кешування. Workbox інтегрується з більшістю сучасних фронтенд-фреймворків і систем білду, серед яких Webpack, Vite і Rollup.

Для проєктів, побудованих на основі JavaScript, особливо високою популярністю користуються фреймворки React, Vue.js та Angular, які мають вбудовану або додатково встановлювану підтримку PWA. У середовищі React одним із найпоширеніших рішень є Create React App із шаблоном PWA або використання Next.js. Next.js надає можливості попереднього рендерингу, оптимізації маршрутизації, генерації сторінок на сервері та інтеграції з Workbox, що робить його відмінним вибором для змінних і високонавантажених застосунків.

Для екосистеми Vue.js аналогічним є фреймворк Nuxt.js, який пропонує модуль PWA для автоматичної конфігурації маніфесту, генерації Service Worker, оптимізації ресурсів та створення офлайн-режиму. Nuxt.js дозволяє легко реалізувати headless-інтерфейси для e-commerce платформ, таких як Magento, Shopify або WooCommerce, що робить його одним із найпопулярніших рішень для створення storefront-додатків.

Фреймворк Angular включає в себе офіційну підтримку PWA через Angular Service Worker, який конфігурується за допомогою маніфест-подібного файлу та дозволяє реалізувати складні сценарії кешування та оновлення. Angular підходить для великих корпоративних проєктів, де важливою є сувора структура, модульність та масштабованість застосунку.

Окрему групу інструментів становлять спеціалізовані headless-commerce фреймворки, такі як Vue Storefront, ScandiPWA, FrontCommerce, Gatsby

Commerce та інші. Ці рішення орієнтовані на інтеграцію з конкретними e-commerce платформами та дозволяють створювати високопродуктивні фронтенд-застосунки з підтримкою PWA «з коробки». Наприклад, Vue Storefront підтримує одночасно декілька backend-платформ (Magento, Shopify, BigCommerce, WooCommerce), що робить його одним із найгнучкіших інструментів на ринку. ScandiPWA, у свою чергу, є повноцінною React-темою для Magento 2, яка реалізує PWA без необхідності створення headless-архітектури.

Для швидкого створення PWA без глибокої технічної підготовки існують такі платформи, як PWA Builder та Ionic PWA Toolkit. PWA Builder дозволяє автоматично генерувати маніфест, Service Worker та базові конфігурації, а також пропонує інструменти для перевірки відповідності застосунку рекомендаціям Google Lighthouse. Ionic PWA Toolkit надає компонентний підхід до створення UI та дозволяє поєднувати PWA зі світом гібридних мобільних застосунків.

Значну роль у розробці PWA відіграють і серверні технології. Для headless-комерції важливими є GraphQL API [9], які дозволяють ефективно отримувати дані з бекенду, мінімізувати кількість запитів та зменшувати обсяг переданих даних. У цьому контексті системи електронної комерції, такі як Magento 2 чи Shopify, вже мають потужні GraphQL-ендпоінти, що значно спрощує інтеграцію з фронтендом, розгорнутим як PWA. REST API також залишаються актуальними, особливо для WordPress/WooCommerce, де вони широко використовуються та підтримують повну headless-модель.

Таким чином, сучасні технологічні інструменти та фреймворки для реалізації PWA охоплюють широкий спектр можливостей — від низькорівневих бібліотек для роботи з Service Worker до повноцінних headless-рішень, орієнтованих на електронну комерцію. Їх використання дозволяє адаптувати підхід до потреб конкретного проєкту, незалежно від складності, масштабності та вимог до інтеграції. У наступних підрозділах буде розглянуто, як ці

інструменти застосовуються на практиці для реалізації PWA у Magento 2, Shopify та WooCommerce.

2.3 Реалізація PWA у Magento 2

Реалізація прогресивного веб-додатку у середовищі Magento 2 є важливою віхою у розвитку сучасних e-commerce-рішень, орієнтованих на максимальну продуктивність, адаптивність і покращений користувацький досвід. Magento, як одна з провідних платформ для створення інтернет-магазинів, активно впроваджує підтримку PWA, надаючи розробникам можливості створювати швидкі, надійні та інтуїтивно зрозумілі інтерфейси. Особливість Magento 2 полягає в її відкритій архітектурі та розширеній модульній системі, що дозволяє інтегрувати найсучасніші веб-технології.

Основним інструментом для реалізації PWA в Magento 2 є набір компонентів, що об'єднуються в PWA Studio. Це спеціалізований проєкт з відкритим кодом, створений Adobe, який забезпечує повноцінну екосистему для створення прогресивного інтерфейсу на основі React, GraphQL та Webpack. PWA Studio включає інструменти для керування маршрутами, станом застосунку, підключенням до API, локалізацією і розширенням функціональності через модулі. Це дає змогу розробникам сконцентруватися на створенні кастомізованого користувацького інтерфейсу без необхідності повністю переписувати логіку бекенду.

Застосування GraphQL як основного механізму взаємодії між клієнтською та серверною частинами забезпечує точкове отримання даних, зменшує обсяг трафіку і пришвидшує час завантаження сторінок. У порівнянні з REST API, GraphQL дозволяє витягувати лише ті поля, які необхідні в конкретному компоненті, що особливо важливо для мобільних користувачів із обмеженим інтернет-з’єднанням.

Архітектура Magento 2 PWA побудована на принципах компонентного підходу, що забезпечує масштабованість і повторне використання коду. Кожен

функціональний елемент, як-от кошик, сторінка товару або каталог, реалізується як окремий React-компонент із власним станом і логікою. Це дозволяє розділяти відповідальність між модулями, полегшує тестування та обслуговування коду. При цьому підтримується концепція гібридного рендерингу, де перше завантаження сторінки може бути здійснене на сервері, а подальша навігація - через SPA-механізми.

Важливо зазначити, що PWA в Magento 2 дозволяє реалізовувати офлайн-режим, push-повідомлення та встановлення сайту як застосунку на головний екран пристрою. Це відкриває можливості для покращення клієнтського досвіду, особливо в умовах обмеженого доступу до мережі. Система Service Worker відповідає за обробку запитів у фоновому режимі, зберігання кешу та контроль над мережею, дозволяючи реалізувати розширений офлайн-функціонал. Для електронної комерції це означає збереження позицій у кошику або перегляд сторінок навіть без доступу до інтернету.

Не менш важливою перевагою є можливість використання модульного підходу при розробці. Розширення та інтеграції Magento 2, які реалізовані відповідно до стандартів Adobe Commerce, можуть бути адаптовані під PWA, що дозволяє використовувати існуючий функціонал без необхідності повторної реалізації. Проте деякі модулі потребують переписування або створення спеціальних GraphQL-ендпоінтів для коректної роботи з клієнтським застосунком. Це створює певні виклики, але водночас відкриває шлях до глибокої кастомізації.

Інфраструктура розгортання PWA у Magento 2 також має специфічні вимоги. Для локальної розробки використовується інструмент Venia-concept, який виконує роль стартового шаблону. На рисунку 3 наведено архітектуру розробки та виконання PWA-застосунку на базі Magento PWA Studio, що включає клієнтську частину на основі React [10], інструменти збірки та інтеграцію з

backend API. Він містить готову конфігурацію Webpack, реалізовану підтримку міжнародної локалізації, а також базову логіку для аутентифікації, навігації та управління станом.

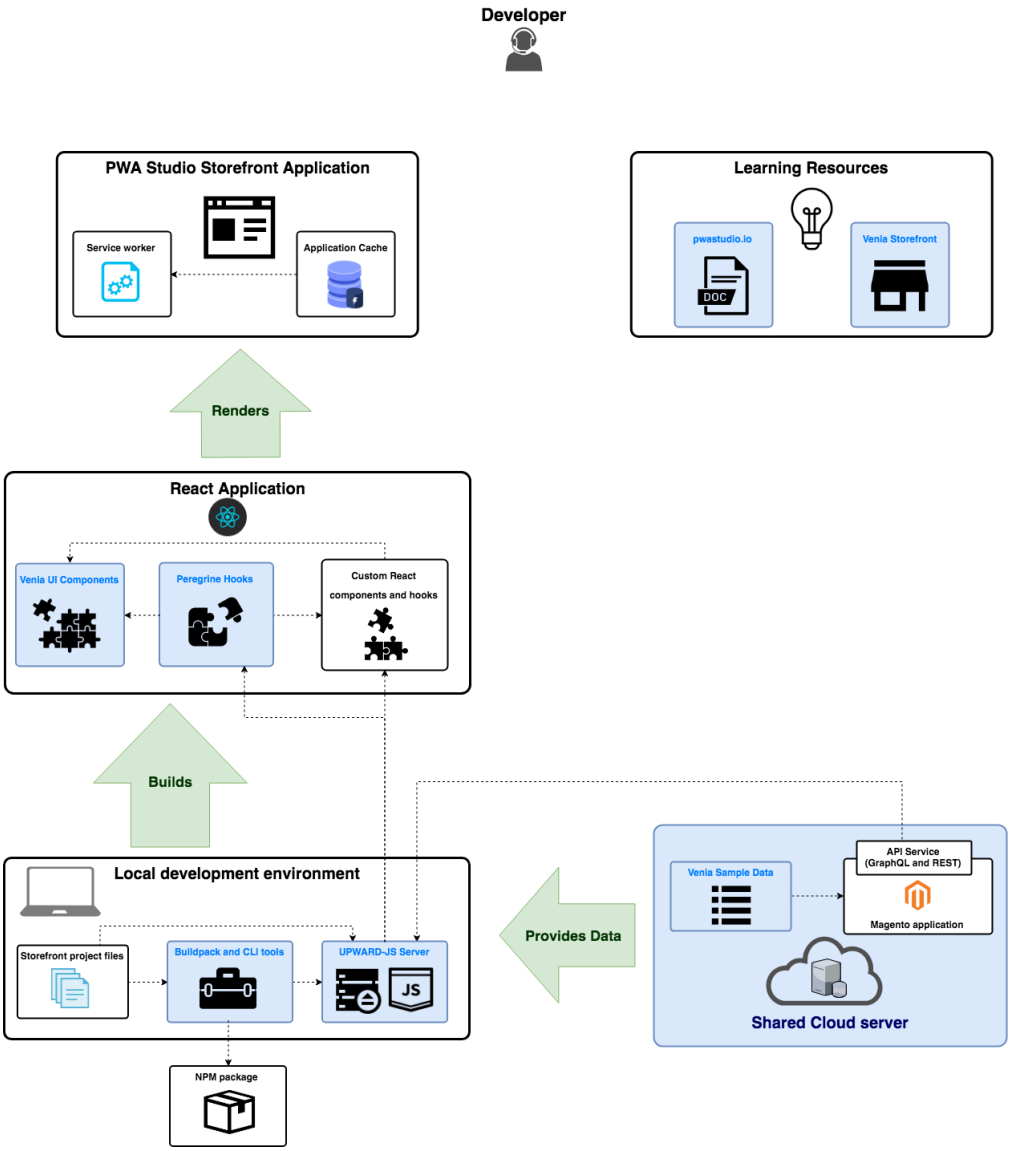


Рисунок 3 — Архітектура розробки та виконання PWA-застосунку на базі Magento PWA Studio

Таким чином, реалізація прогресивного веб-додатку в Magento 2 через використання PWA Studio є потужним і перспективним підходом для розвитку

сучасних eCommerce-рішень. Комбінація технологій React, GraphQL і модульної структури платформи забезпечує не лише гнучкість у розробці, а й високу швидкодію, стабільність та розширюваність. Незважаючи на певні технічні виклики, PWA в екосистемі Magento 2 створює фундамент для довготривалого зростання бізнесу в цифровому середовищі.

2.4 Реалізація PWA у Shopify

Інтеграція технології прогресивного веб-додатку в екосистему Shopify є важливою тенденцією у розвитку сучасної електронної комерції, яка спрямована на підвищення зручності користування, адаптивності інтерфейсів і швидкодії веб-ресурсів. Shopify, як одна з найпопулярніших SaaS-платформ для створення онлайн-магазинів, пропонує широкі можливості для впровадження PWA через використання готових тем, сторонніх застосунків і спеціалізованих фреймворків. Завдяки своїй архітектурі, що орієнтована на швидкий старт і мінімальні технічні бар'єри, Shopify дозволяє впроваджувати PWA навіть без глибоких змін у серверній частині, що робить її зручною для малого та середнього бізнесу.

Особливістю реалізації PWA у Shopify є те, що платформа не має вбудованої офіційної підтримки прогресивних веб-додатків, як, наприклад, у випадку Magento 2. Проте вона підтримує кастомізацію через Liquid-шаблони, API та вебхуки, що дає змогу інтегрувати необхідну логіку PWA. Один із підходів передбачає використання JAMstack-архітектури, де Shopify Headless виступає бекендом, а фронтенд реалізується за допомогою React, Vue або іншої сучасної бібліотеки. Цей підхід дозволяє створювати повністю адаптовані інтерфейси, оптимізовані для мобільного досвіду.

Важливим елементом у реалізації PWA на базі Shopify є використання GraphQL Storefront API, яке забезпечує доступ до інформації про продукти,

кошик, клієнтів та транзакції. Завдяки цьому API можна створювати кастомізовані інтерфейси, що працюють незалежно від стандартного шаблонізатора Shopify. У поєднанні з Service Worker та Web App Manifest це дозволяє досягти бажаної поведінки PWA: кешування сторінок, роботи в офлайн-режимі та інтеграції push-повідомлень. Важливою перевагою такого підходу є те, що він зменшує залежність від готових тем і дозволяє реалізувати унікальний користувацький досвід [11].

Успішні приклади впровадження PWA у Shopify можна знайти серед великих брендів, які використовують headless-архітектуру для масштабованості та оптимізації продуктивності. Це демонструє ефективність поєднання Shopify із сучасними фреймворками у створенні прогресивного користувацького досвіду.

Однак, реалізація PWA у Shopify має і свої обмеження. Платформа є закритою системою, що не дозволяє прямого доступу до серверного коду, а отже ускладнює реалізацію деяких низькорівневих функцій PWA. Крім того, обробка push-сповіщень або офлайн-доступу вимагає використання зовнішніх сервісів, таких як Firebase або OneSignal, що додає складності до інфраструктури проєкту. Ще одним викликом є те, що не всі функції Storefront API доступні для кастомного фронтенду, а тому іноді доводиться комбінувати API-запити з різних джерел. На рисунку 4 наведено архітектуру реалізації Progressive Web Application для платформи Shopify, у якій клієнтська частина побудована з використанням Vue Storefront та взаємодіє з серверною частиною через Storefront API.

З технічної точки зору, використання Next.js або Gatsby у ролі фронтенд-рішень дозволяє забезпечити попередню генерацію сторінок (SSG) або серверний рендеринг (SSR), що критично важливо для SEO. Це дозволяє не лише зменшити час до першого рендеру, а й забезпечити коректне сканування сайту пошуковими системами. У цьому контексті Shopify виступає як headless CMS, яка лише

обробляє дані, тоді як усе візуальне відображення реалізується на стороні клієнта через фреймворки, які підтримують прогресивну логіку.

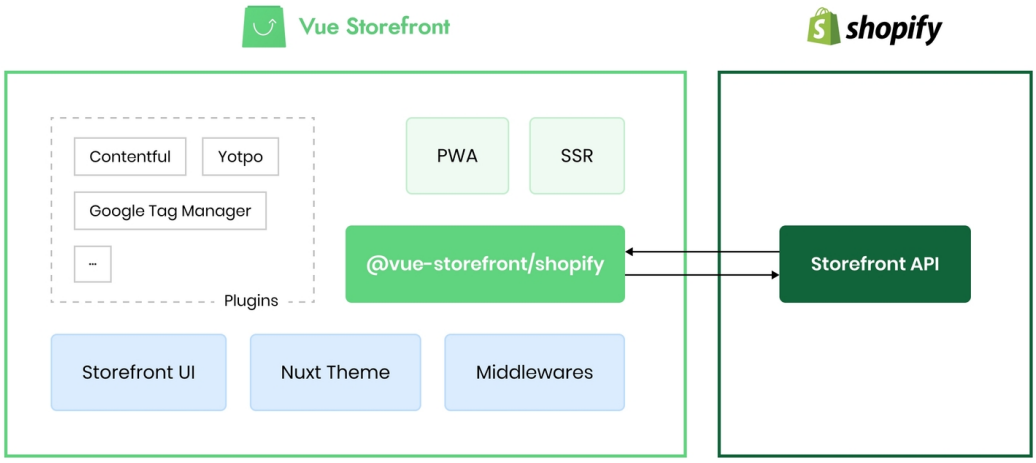


Рисунок 4 — Архітектура реалізації PWA-застосунку на базі Shopify з використанням Vue Storefront

Процес розгортання PWA у Shopify також має свої особливості. Зазвичай використовується окремий сервер або хостинг для клієнтської частини застосунку, а Shopify залишає за собою обробку замовлень, транзакцій та зберігання даних. Це дозволяє розділити зони відповідальності між фронтендом і бекендом, оптимізуючи продуктивність і масштабованість [12]. При цьому виникає необхідність у налаштуванні проксі-серверів, CORS-політик і безпечного обміну даними між сервісами, що вимагає певного рівня технічної компетентності від команди.

Крім того, Shopify активно розвиває концепцію Hydrogen - власного фреймворку для створення кастомних storefront з використанням React. Hydrogen надає стартову структуру, інтеграцію з Shopify API та попередньо налаштовані компоненти для реалізації найпоширеніших функцій eCommerce. Він тісно пов’язаний із платформою Oxygen, що відповідає за хостинг таких storefront, що

спрощує процес розгортання і зменшує технічні бар’єри для команд розробників. Ця ініціатива демонструє прагнення Shopify забезпечити повну підтримку headless-рішень і створити сприятливі умови для впровадження PWA.

Таким чином, реалізація PWA у Shopify є потужним інструментом для створення швидких, адаптивних і функціонально насичених вебзастосунків, здатних забезпечити конкурентні переваги у сфері онлайн-продажів. Незважаючи на обмеження платформи, можливості кастомізації через API, підтримка сучасних фронтенд-технологій і розвиток Hydrogen відкривають шлях до ефективної побудови PWA. Успішна інтеграція прогресивних вебдодатків у Shopify вимагає комплексного підходу, технічної підготовки і стратегічного планування розвитку eCommerce-проєктів.

2.5 Реалізація PWA у WooCommerce

Реалізація прогресивного веб-додатку у системі WooCommerce відкриває нові можливості для власників онлайн-магазинів, які базуються на WordPress. Завдяки відкритій архітектурі, широкій підтримці спільноти та доступності безлічі плагінів, WooCommerce залишається популярним вибором для малого та середнього бізнесу. У контексті впровадження PWA ця платформа дозволяє реалізувати функціональні й зручні інтерфейси із залученням сучасних технологій. Основна перевага полягає у можливості адаптувати існуючий вміст і структуру сайту під стандарти прогресивного вебдодатку без повної перебудови бекенду.

Процес реалізації PWA у WooCommerce, як правило, здійснюється через встановлення спеціалізованих плагінів або використання headless-підходу. Найпопулярнішими інструментами вважаються плагіни типу Super Progressive Web Apps, PWA for WP & AMP, а також використання таких рішень, як Frontity

або Gatsby для побудови фронтенду. Ці засоби дозволяють забезпечити роботу офлайн, швидке завантаження сторінок, push-сповіщення та можливість додавання сайту на головний екран пристрою. З технічного погляду, найбільш перспективною вважається headless-архітектура, що дозволяє відокремити логіку бекенду від клієнтського інтерфейсу.

Використання REST API WordPress є базовою умовою для реалізації PWA на основі WooCommerce. API надає доступ до даних про продукти, категорії, замовлення, користувачів та інші елементи магазину. Це дає змогу побудувати фронтенд за допомогою сучасних JavaScript-бібліотек, таких як React або Vue.js, і забезпечити швидку взаємодію користувача з вебдодатком. Проте варто зазначити, що REST API WordPress має певні обмеження, зокрема щодо складних фільтрацій та агрегації даних, що потребує оптимізації або використання додаткових плагінів для розширення його функціоналу.

Гарним прикладом є магазини, які використовують плагіни типу PWA for WP для досягнення базового рівня прогресивності. У цьому випадку реалізація PWA відбувається швидше та не потребує зміни архітектури сайту, але функціональність залишається обмеженою. Наприклад, офлайн-режим часто працює лише для сторінок, відвіданих раніше, push-сповіщення обмежені інтеграцією зі сторонніми сервісами, а інтерфейс не завжди досягає рівня нативного застосунку. Проте для малих проєктів така реалізація може бути ефективною стратегією підвищення мобільної зручності.

З погляду SEO, PWA у WooCommerce дозволяє покращити індексацію завдяки зменшенню часу до першого байту та швидшому рендерингу сторінок. Підходи, що базуються на статичній генерації (наприклад, за допомогою Gatsby), сприяють кращому сприйняттю контенту пошуковими системами. У поєднанні з можливістю кешування та зменшенням розміру запитів до сервера це забезпечує покращення позицій сайту у видачі. Варто також зазначити, що налаштування

метатегів, структурованих даних та перенаправлень у headless-рішеннях потребує додаткової уваги та тестування.

Щодо інтерфейсу користувача, PWA на WooCommerce дає змогу реалізувати функції, подібні до мобільних застосунків: швидке додавання товарів у кошик, збереження останніх дій користувача, миттєве оновлення вмісту сторінок без перезавантаження. Це забезпечує плавний і зручний досвід взаємодії, що особливо важливо для мобільних користувачів. Впровадження Material Design, кастомних анімацій та адаптивної навігації покращує загальне враження від користування магазином та стимулює повторні відвідування.

З позиції розгортання і підтримки, PWA на WooCommerce може реалізовуватись на окремому сервері, або ж бути частиною основного хостингу WordPress. У випадку headless-підходу необхідна інфраструктура, що підтримує Node.js або інші сервери рендерингу, а також наявність CI/CD-процесів для забезпечення оновлень. Важливим етапом є налаштування HTTPS, Service Worker, кешування та offline-функціональності, що часто потребує глибших знань у сфері DevOps та безпеки. Підтримка такого рішення є складнішою, ніж класичного WooCommerce-магазину, проте надає значно більше можливостей для масштабування.

Інша перевага полягає в тому, що спільнота WordPress активно працює над удосконаленням підтримки PWA. З’являються нові плагіни, покращуються документації, а компанії, що спеціалізуються на WordPress-розробці, пропонують послуги із впровадження headless-магазинів на базі WooCommerce. Це свідчить про актуальність теми та її подальший розвиток. І хоча поки що реалізація PWA у WooCommerce залишається переважно експериментальною та залежить від технічного рівня команди, зростання інтересу до цього підходу є стабільним.

У підсумку можна стверджувати, що реалізація прогресивного веб-додатку на базі WooCommerce є перспективним напрямом для онлайн-торгівлі, особливо в умовах зростання частки мобільного трафіку. Хоча така інтеграція має свої виклики - як технічні, так і організаційні - вона відкриває нові горизонти для покращення користувацького досвіду, швидкодії сайту та ефективності бізнесу. Залежно від ресурсів і цілей, реалізація PWA у WooCommerce може стати як базовим рішенням через плагіни, так і повноцінною трансформацією архітектури із застосуванням headless-підходу.

2.6 Порівняння підходів до реалізації PWA на Magento 2, Shopify та WooCommerce

Порівняння можливостей реалізації Progressive Web Application на різних системах електронної комерції є ключовим етапом вибору технологічного середовища для побудови високопродуктивного та масштабованого прототипу. Magento 2, Shopify та WooCommerce представляють три різні моделі розвитку e-commerce платформ: повністю керовану headless-сумісну архітектуру, SaaS-рішення з обмеженими технічними правами доступу та CMS-орієнтований підхід із високим рівнем кастомізації. Кожна з цих платформ по-своєму впливає на процес інтеграції PWA та формує відмінності у можливостях, складності реалізації та ефективності кінцевого застосунку.

Magento 2 має найбільш розвинену нативну підтримку PWA, що реалізована через офіційний фреймворк PWA Studio, storefront-тему Venia, розширені GraphQL API та інтегровану headless-архітектуру. Magento 2 дозволяє створювати повноцінний PWA із широкими можливостями оптимізації кешування, маршрутизації, серверного рендерингу та кастомізації бізнес-логіки. Розробник отримує повний контроль над Service Worker, Web App Manifest та

клієнтсько-серверною взаємодією, що робить платформу особливо привабливою для проєктів з високими вимогами до продуктивності й масштабованості. Крім офіційного стеку, екосистема Magento пропонує альтернативні рішення, такі як ScandiPWA та Vue Storefront, що дозволяють впроваджувати PWA у різних архітектурних підходах.

На відміну від Magento, Shopify є SaaS-платформою, де реалізація PWA обмежена архітектурними засадами сервісу. Shopify не дозволяє повноцінне керування Service Worker у межах основного домену, а налаштування офлайн-функціональності виконуються лише через headless-підхід із зовнішнім фронтендом на основі Next.js, Nuxt.js або Hydrogen. Замість нативного PWA інструментарію Shopify пропонує Storefront API та нову платформу Hydrogen + Oxygen, що дозволяють реалізувати прогресивні застосунки, але лише у форматі повністю стороннього фронтенд-рішення. Таким чином, PWA на Shopify потребує складнішої інфраструктури, втрати базових можливостей Liquid-тем та суттєвого розширення архітектури за межі стандартного функціоналу.

WooCommerce, як плагін для WordPress, має найбільшу гнучкість у впровадженні PWA, оскільки дозволяє пряме втручання в HTML, JavaScript, Service Worker та серверну конфігурацію. Проте відсутність нативної підтримки PWA означає, що реалізація значною мірою залежить від сторонніх плагінів (SuperPWA, PWA for WP) або повного переходу на headless-модель. WooCommerce добре підходить для малих і середніх магазинів, де достатньо базових можливостей PWA, але у великих проєктах його продуктивність та масштабованість суттєво поступаються Magento і Shopify. Ключовою перевагою WooCommerce є простота інтеграції та низька складність, але недоліками — обмеження у швидкодії та значна залежність від WordPress-екосистеми.

При порівнянні трьох платформ за критеріями продуктивності, гнучкості реалізації, рівня підтримки API, можливості побудови headless-архітектури та ступеня контролю над Service Worker, найбільш придатною для дослідження та практичної реалізації PWA виступає Magento 2. По-перше, Magento надає найсильніший інструментарій для створення PWA «з коробки», що включає офіційні фреймворки, гнучку архітектуру, стандартизовані GraphQL API та повну свободу кастомізації. По-друге, саме Magento є платформою, орієнтованою на великі комерційні проєкти й складні сценарії взаємодії, де переваги PWA — офлайн-робота, кешування, швидкодія та адаптивність — розкриваються найповніше. По-третє, на відміну від Shopify, Magento не має обмежень щодо керування Service Worker та зовнішнім JavaScript, що дозволяє реалізувати PWA у чистому вигляді без додаткових проксі-рішень.

Таким чином, у межах цього дослідження для подальшої реалізації та проєктування прототипу обрано Magento 2, оскільки ця платформа забезпечує оптимальне поєднання нативних інструментів, технологічної гнучкості, продуктивності та відповідності сучасним стандартам побудови PWA. Magento 2 дозволяє продемонструвати повний спектр можливостей PWA, що робить її найбільш придатною для практичної частини даної дипломної роботи.

2.7 Висновки до розділу 2

У другому розділі було здійснено аналіз практичних підходів, інструментів і технологічних рішень, що застосовуються для реалізації Progressive Web Application у сучасних системах електронної комерції. Проведене дослідження показало, що впровадження PWA може здійснюватися за різними моделями — від мінімального додавання Service Worker до повноцінної розробки headless-архітектури, де фронтенд і бекенд функціонують як незалежні підсистеми. Вибір

підходу залежить насамперед від потреб бізнесу, вимог до продуктивності, рівня доступу до внутрішніх API платформи та можливостей масштабування.

У ході аналізу було визначено, що сучасні інструменти для створення PWA охоплюють широкий спектр технологічних стеків: від низькорівневих бібліотек, таких як Workbox, до комплексних фреймворків, орієнтованих на e-commerce, таких як Vue Storefront, ScandiPWA та Hydrogen. Значна роль у розвитку PWA належить також Next.js, Nuxt.js, Angular та іншим фреймворкам, які забезпечують ефективну роботу з кешуванням, серверним рендерингом та оптимізацією ресурсів. Ці інструменти дозволяють створювати прогресивні додатки різного рівня складності — від адаптації існуючого сайту до архітектурно повноцінних headless-рішень.

Порівняння реалізації PWA на платформах Magento 2, Shopify та WooCommerce засвідчило суттєві відмінності у можливостях інтеграції. Shopify, як SaaS-рішення, обмежує пряме керування Service Worker та вимагає використання headless-підходу для досягнення повноцінної PWA-функціональності. WooCommerce, завдяки відкритості WordPress-екосистеми, дозволяє реалізувати PWA через плагіни або власні ручні конфігурації, проте поступається продуктивністю та масштабованістю. Magento 2 продемонструвала найбільш розвинену та стандартизовану підтримку PWA, включаючи нативні інструменти, гнучкі GraphQL API та потужні можливості кастомізації.

У результаті порівняння встановлено, що Magento 2 є оптимальною платформою для реалізації прототипу PWA, оскільки поєднує високу продуктивність, масштабованість, гнучкість архітектури та розвинену підтримку headless-рішень. На відміну від Shopify, Magento надає повний доступ до Service Worker та фронтенд-логіки, а порівняно з WooCommerce — забезпечує стабільність роботи під високими навантаженнями, що є критично важливим для

систем електронної комерції. Це робить Magento 2 найбільш придатною платформою для демонстрації всіх переваг та практичних можливостей PWA.

Таким чином, результати цього аналізу визначають логіку архітектурних і технічних рішень, що будуть розглянуті у наступному розділі дипломної роботи.

РОЗДІЛ 3 ПРОЄКТУВАННЯ ТА РОЗРОБКА PWA ДЛЯ ПЛАТФОРМИ MAGENTO 2

3.1 Аналіз вимог та підготовка до розробки PWA

Проєктування та розробка прогресивного веб-застосунку для системи електронної комерції потребують попереднього аналізу функціональних, нефункціональних та інтеграційних вимог, а також підготовки відповідного технологічного середовища. На цьому етапі формується загальне бачення архітектури застосунку, визначаються ключові технічні обмеження платформи Magento 2, обираються інструменти фронтенд-розробки та плануються підходи до реалізації PWA-функціональності.

Функціональні вимоги до PWA в контексті електронної комерції охоплюють базові можливості інтернет-магазину: перегляд каталогу товарів, роботу з категоріями, пошук, формування кошика, оформлення замовлення, аутентифікацію користувача, перегляд історії покупок та взаємодію з профілем. Наявність цих функцій є обов’язковою для забезпечення стандартного користувацького сценарію. Для PWA додатковим критичним елементом є стабільність інтерфейсу у випадку нестабільного або відсутнього інтернет-з’єднання, що потребує впровадження механізмів офлайн-кешування, фонової синхронізації та fallback-сторінок [14].

Нефункціональні вимоги визначають очікувані характеристики продуктивності, безпеки, масштабованості та часу реакції застосунку. Сучасні PWA мають забезпечувати високу швидкість завантаження, миттєву навігацію без перезавантаження сторінок, підтримку роботи на мобільних пристроях та оцінку не нижче 90 балів у Lighthouse за показниками Performance, SEO та Best Practices. Важливим фактором є відповідність вимогам безпеки Magento 2, а саме

використання протоколу HTTPS, токен-орієнтованої аутентифікації (customer tokens), захищеної взаємодії з GraphQL API та дотримання серверних політик щодо кешування.

Підготовка Magento 2 як backend-платформи передбачає активацію та конфігурацію GraphQL API, встановлення необхідних модулів і перевірку коректності роботи ендпоінтів, пов'язаних із каталогом, кошиком, замовленнями та користувачами. Magento 2 є повноцінною headless-платформою, що дозволяє будувати фронтенд-рішення без використання традиційної теми. Тому на етапі підготовки важливо протестувати GraphQL-схему, визначити необхідні поля та фрагменти даних для PWA, а також оцінити можливості серверного кешування (Varnish, Redis) для забезпечення продуктивності.

Вибір інструментів фронтенд-розробки є ключовим етапом, що впливає на складність реалізації, продуктивність інтерфейсу та можливість інтеграції з Magento 2. Серед доступних рішень варто розглянути PWA Studio — офіційний інструментарій Adobe Commerce, який включає storefront-тему Venia, модулі для роботи з GraphQL та оптимізовані механізми побудови PWA. Однак альтернативні рішення, такі як React (Next.js), Vue Storefront або ScandiPWA, забезпечують більшу гнучкість і краще підходять для проєктів, де важлива кастомізація інтерфейсу та поведінки застосунку. На практиці вибір фреймворку визначається потребами у продуктивності, підтримці SSR/SSG, рівні готових компонентів та складності інтеграції з GraphQL API Magento [16].

Підготовка до розробки також включає налаштування локального середовища: встановлення Node.js, Yarn або NPM, інструментів зборки (Webpack, Vite), а також конфігурацію середовища для тестування запитів через GraphQL Playground або Altair. Важливим етапом є створення базової структури проєкту, підготовка компонентів для маршрутизації, налаштування менеджера стану

(Redux, Zustand, Pinia, Vuex тощо) та інтеграція з системами кешування запитів, такими як Apollo Client або React Query [8].

Таким чином, процес аналізу вимог та підготовки середовища для розробки PWA визначає технічний фундамент усього прототипу. На цьому етапі формується архітектурне бачення, обираються оптимальні інструменти та забезпечується готовність Magento 2 як backend-системи. Виконання цих дій дозволяє перейти до наступного етапу — проєктування архітектури клієнтської частини, що становитиме основу функціональної та інтеграційної логіки PWA-застосунку.

3.2 Проєктування архітектури клієнтської частини

Проєктування архітектури клієнтської частини прогресивного веб-застосунку для Magento 2 є ключовим етапом, який визначає подальшу структуру, продуктивність і масштабованість усього PWA-рішення. На цьому етапі формується логічна схема взаємодії між інтерфейсом користувача, механізмами кешування, системою управління станом та GraphQL-API Magento 2. Враховуючи складність e-commerce системи, архітектура мала забезпечити як високу швидкодію, так і передбачувану логіку роботи для всіх основних бізнес-процесів — перегляд каталогу, кошика, оформлення замовлення, аутентифікацію та роботу з особистим кабінетом.

У межах проєкту було обрано архітектурний підхід, який базується на використанні React та екосистеми Magento PWA Studio, що включає бібліотеки Peregrine, Venia UI та інструменти Buildpack. Ініціалізацію React-додатку продемонстровано у Лістингу 1. Такий вибір дозволив отримати структурований набір компонентів та готових поведінкових модулів, адаптованих під специфіку Magento 2, а також забезпечив тісну інтеграцію з GraphQL API. React було обрано

як основу клієнтського застосунку завдяки його компонентному підходу, високій продуктивності, підтримці хуків та можливості побудови гнучкої архітектури керування станом через контексти та Apollo Client.

Лістинг 1 — Ініціалізація React-додатку з Adapter та store

```
import React from 'react';
import { render } from 'react-dom';

import store from './store';
import './index.css';
import app from
    '@magento/peregrine/lib/store/actions/app';
import Adapter from '@magento/venia-
    ui/lib/components/Adapter';
import { registerSW } from './registerSW';

const isServer = !globalThis.document;
const origin = isServer
    ? process.env.MAGENTO_BACKEND_URL
    : globalThis.location.origin;

const styles = new Set();

const configureLinks = links => [...links.values()];

const tree = (
    <Adapter
        configureLinks={configureLinks}
        origin={origin}
        store={store}
        styles={styles}
```

```
    />
);
```

Центральним елементом клієнтської архітектури стала інтеграція з Apollo Client, який відповідає за комунікацію з backend-частиною через GraphQL. Apollo забезпечує виконання запитів, мутацій, кешування отриманих даних та можливість оптимістичних оновлень інтерфейсу, що особливо важливо для e-commerce, де реактивність і швидкість взаємодії визначають якість користувацького досвіду. Кеш Apollo замінює традиційні механізми Redux або Vuex, дозволяючи зберігати стан каталогу, кошика та користувача без потреби в ручному керуванні складними структурами даних. Ініціалізацію Redux представлено у Літсингу 2. Також було застосовано модуль apollo-cache-persist, який дозволяє зберігати кеш у локальному сховищі браузера, забезпечуючи швидше завантаження інтерфейсу при повторних відвідуваннях.

Лістинг 2 — Ініціалізація React-додатку з Adapter та store

```
import { combineReducers, createStore } from 'redux';
import { enhancer, reducers } from '@magento/peregrine';

const rootReducer = combineReducers(reducers);
export default createStore(rootReducer, enhancer);
```

Архітектура клієнтської частини побудована за принципом розподілу відповідальностей на рівні контейнерних і презентаційних компонентів. Презентаційні компоненти Venia UI відповідають за візуалізацію, тоді як контейнерні компоненти Peregrine реалізують логіку бізнес-операцій, роботу з GraphQL і управління станом. Такий поділ дозволяє ізолювати UI від логічної частини і забезпечує можливість подальшої кастомізації без порушення внутрішньої структури застосунку. Крім того, було застосовано lazy-loading

окремих маршрутів, що зменшило розмір початкового бандла та покращило метрики швидкодії.

Важливим елементом архітектури стало проєктування маршрутизації. Для забезпечення SPA-поведінки та швидкої навігації застосунок використовує react-router-dom, що дозволяє визначати маршрути для сторінки продукту, категорії, кошика, профілю користувача та checkout-процесу. Кожний маршрут завантажує відповідні GraphQL-запити та компоненти таким чином, щоб забезпечити мінімальний вплив на початковий час рендерингу сторінки. Для SEO-оптимізації застосовано client-side механізми оновлення метаданих сторінок, сумісні з Magento PWA Studio.

Проєктування архітектури також включало інтеграцію Service Worker, який відповідає за кешування статичних ресурсів, попередній прогрів критичних даних та обробку офлайн-сценаріїв. Конфігурація Workbox Webpack Plugin забезпечила генерацію Service Worker автоматично під час збірки, а також визначення стратегій кешування для зображень продуктів, медіафайлів, CSS/JS ресурсів та GraphQL-запитів. Окрему увагу приділено fallback-сторінкам для офлайн-режиму та механізмам background sync для корзини та інших критичних даних [17].

Таким чином, архітектура клієнтської частини PWA для Magento 2 була побудована на основі сучасних фронтенд-технологій та механізмів інтеграції з GraphQL. Використання React, Apollo Client, Peregrine та Venia UI дозволило створити гнучке, масштабоване та високопродуктивне рішення, яке відповідає вимогам електронної комерції та забезпечує стійкість до нестабільного мережевого середовища. Такий підхід сформував основу для реалізації складніших модулів застосунку, які будуть розглянуті в наступних підрозділах.

3.3 Інтеграція PWA з backend API системи електронної комерції

Інтеграція прогресивного веб-застосунку з backend-частиною Magento 2 є центральним елементом розробки PWA, оскільки саме через API здійснюється отримання каталогу товарів, керування кошиком, авторизація користувачів, формування замовлень та виконання інших бізнес-операцій. Magento 2 надає повноцінний GraphQL API, який дозволяє реалізувати headless-архітектуру без необхідності використання класичної теми або REST-запитів. Завдяки цьому PWA може працювати як автономний клієнтський застосунок, взаємодіючи з платформою Magento через стандартизовані схеми запитів і мутацій.

У межах проєкту було використано GraphQL API як основний спосіб комунікації між frontend та backend. На рисунку 5 наведено загальну схему взаємодії клієнтських застосунків із серверною частиною системи електронної комерції через GraphQL API, що дозволяє оптимізувати обмін даними та зменшити кількість мережевих запитів. Таке рішення забезпечує значно кращу продуктивність порівняно з REST, оскільки GraphQL дозволяє отримувати лише ті дані, які потрібні конкретному компоненту, без надмірності у структурі відповіді. Для виконання запитів застосовано Apollo Client — потужний GraphQL-клієнт, інтегрований у стек Magento PWA Studio, який забезпечує кешування, оптимістичні оновлення, обробку мутацій та підтримку реактивних даних.

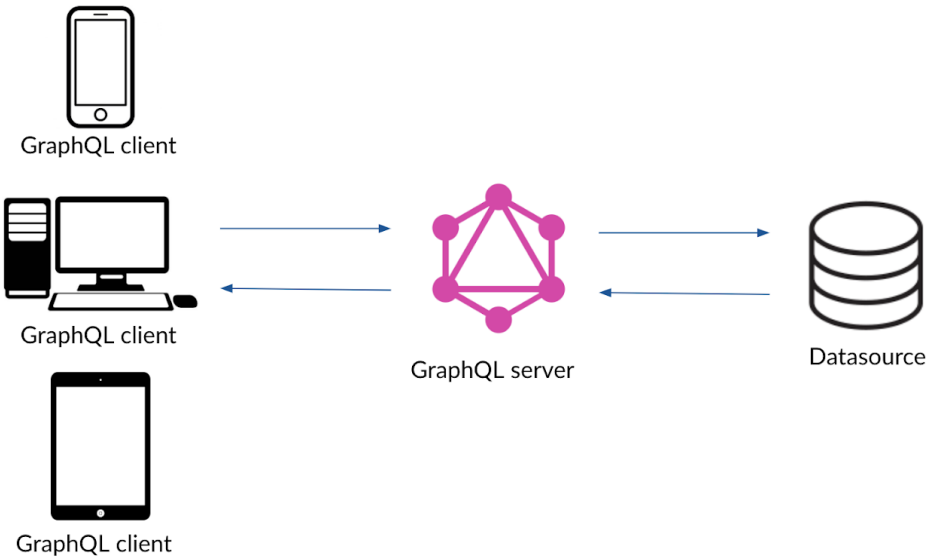


Рисунок 5 — Загальна схема взаємодії клієнтських застосунків з GraphQL-сервером

Одним із перших інтеграційних блоків стала робота з каталогом Magento. Через GraphQL-схему було реалізовано запити до ендпоінтів products, categories та urlResolver, які дозволяють отримувати дані про товари, фільтри, атрибути, медіагалерею та прив’язку до категорій. На основі цих даних було побудовано PWA-інтерфейс каталогу та сторінки товару. Завдяки кешу Apollo, при повторних переходах між сторінками користувач отримує миттєвий рендеринг інтерфейсу, що суттєво підвищує відчуття швидкодії порівняно з традиційною Magento-темою.

Особливу увагу було приділено реалізації механізмів аутентифікації користувачів, що виконуються через мутації generateCustomerToken та revokeCustomerToken. Після успішного виконання мутації токен зберігається як у локальному сховищі браузера, так і в Apollo-кеші, що дозволяє фронтенду виконувати авторизовані запити до приватних ендпоінтів — отримання історії замовлень, оновлення профілю, керування адресами тощо. Додатково було

реалізовано механізм автопоновлення стану користувача при повторному відкритті застосунку за рахунок логіки apollo-cache-persist.

Інтеграція кошика (cart) стала одним з найважливіших аспектів проєкту. Magento 2 використовує окремі GraphQL мутації для створення, оновлення та заповнення кошика, включаючи createEmptyCart, addSimpleProductsToCart, removeItemFromCart, applyCouponToCart та інші. У PWA було реалізовано повноцінний state-management кошика, який автоматично відображає зміни та забезпечує синхронізацію з backend у режимі реального часу. Особливо важливою стала логіка background sync, яка дозволяє користувачеві додавати товари в кошик навіть у випадку тимчасової втрати з'єднання, а після його відновлення — передавати зміни на сервер.

Не менш значущим аспектом є реалізація checkout-процесу, що складається з декількох мутацій Magento: встановлення адреси, вибору доставки, налаштування платіжного методу та підтвердження замовлення. Для цього було використано низку GraphQL мутацій (setShippingAddressesOnCart, setPaymentMethodOnCart, placeOrder), а сама логіка checkout була розділена на окремі компоненти, щоб забезпечити зрозумілий UX та гнучкість конфігурацій. Завдяки Apollo кешування інформації про кошик та замовлення виконувалося без необхідності повторних запитів до сервера.

Під час інтеграції також було враховано особливості роботи Magento з кешуванням, зокрема інвалідацію даних при оновленні товарів та категорій, що впливає на GraphQL-відповіді. Для уникнення конфліктів було використано політики fetchPolicy у Apollo (cache-first, network-only, cache-and-network), що дозволило знайти баланс між продуктивністю та актуальністю даних.

Таким чином, інтеграція PWA з backend API Magento 2 базується на тісній взаємодії React-компонентів, Apollo Client та GraphQL-схеми Magento. Реалізований підхід забезпечує високу гнучкість, продуктивність і

масштабованість, дозволяючи PWA працювати як повноцінний фронтенд-магазину без залежності від класичних тем Magento. Отримані результати створюють фундамент для впровадження модулів бізнес-логіки, що будуть розглянуті у наступному підрозділі.

3.4 Реалізація ключових функціональних модулів PWA

Реалізація прогресивного веб-застосунку для Magento 2 передбачає створення низки функціональних модулів, які забезпечують повноцінний цикл взаємодії користувача з електронним магазином — від перегляду каталогу товарів до оформлення замовлення. В основі клієнтської логіки перебуває стек Magento PWA Studio, React та Apollo Client, які формують архітектурний фундамент застосунку та визначають принципи взаємодії з GraphQL API.

Каталог товарів став одним із перших модулів, реалізованих у рамках проєкту, оскільки саме з нього починається більшість користувацьких сесій. На основі GraphQL-ендпоінтів Magento було налаштовано отримання структурованих даних про товари, категорії, атрибути та медіавміст. React-компоненти формують інтерфейс категорій, а кеш Apollo забезпечує миттєве відтворення сторінок при повторному навігаційному циклі, що значно покращує відчуття швидкодії. Додаткові оптимізації — ліниве завантаження, попереднє кешування популярних категорій, компактні GraphQL-запити — дозволили зменшити мережеві витрати та підвищити оцінки Lighthouse.

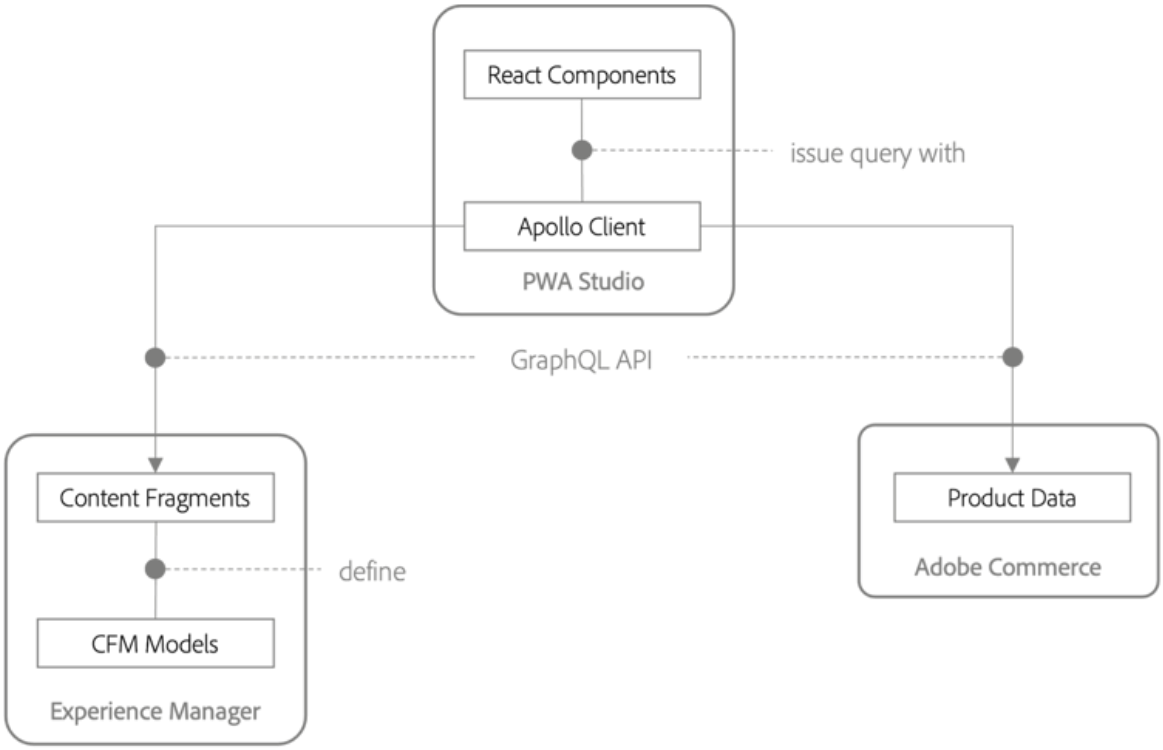


Рисунок 6 — Архітектурна схема взаємодії PWA Studio з Apollo Client

Наступним етапом стала реалізація сторінки товару, що містить детальну інформацію про продукт і забезпечує один із ключових бізнес-процесів — додавання товару до кошика. Magento традиційно підтримує широкі можливості конфігурації продуктів, тому логіка вибору атрибутів включає реактивне оновлення ціни, зображення, SKU та доступності. Завдяки Apollo Client значне навантаження з боку синхронізації стану переноситься у граф кешу, що мінімізує кількість мережевих запитів і забезпечує плавний користувацький досвід.

Окремої уваги потребував модуль кошика. Оскільки PWA працює у форматі headless-клієнта, усі операції — створення кошика, додавання, видалення та оновлення товарів — виконуються через GraphQL-мутації. Кошик зберігається локально, тому користувач може продовжувати покупки навіть після перезавантаження застосунку або тимчасової втрати інтернет-з’єднання. Якщо під час роботи мережа недоступна, операції додаються у чергу та

синхронізуються після її відновлення. Це дозволяє PWA поводитися значно надійніше за традиційні веб-застосунки, забезпечуючи безперервність взаємодії.

Оформлення замовлення є найскладнішим процесом, оскільки включає введення адреси доставки, вибір способу доставки, вибір методу оплати та фінальне підтвердження замовлення. Усі ці операції реалізуються через послідовність GraphQL-мутацій, а React-компоненти зберігають стан чекауту у Apollo Cache та localStorage. Завдяки цьому користувач може відновити процес оформлення навіть після перерви, не втрачаючи попередньо введених даних. Оптимістичні оновлення інтерфейсу забезпечують миттєву реакцію на дії користувача, а серверне підтвердження змін виконується у фоновому режимі.

Важливою властивістю реалізованих модулів є підтримка офлайн-режиму. Service Worker забезпечує кешування статичних ресурсів, зображень і попередньо завантажених сторінок каталогу. При втраті мережі PWA переходить у fallback-режим, у якому застосунок залишається функціональним — користувач може переглядати раніше завантажені сторінки та взаємодіяти з кошиком, а всі операції, які потребують зв’язку з backend, будуть виконані пізніше завдяки чергам синхронізації.

Таким чином, реалізація ключових функціональних модулів забезпечила перетворення Magento 2 на повноцінний headless-магазин з ефективною навігацією, стабільною поведінкою, високою швидкодією та підтримкою офлайн-режиму. Це дозволило створити користувацький досвід, який наближається до нативних мобільних застосунків і водночас зберігає переваги веб-технологій.

3.4.1 Організація взаємодії з GraphQL та управління станом клієнта

Організація взаємодії між клієнтською частиною PWA та backend-платформою Magento 2 є критичним елементом функціонування прогресивного веб-застосунку, оскільки саме цей рівень визначає ефективність обробки даних, швидкість реакції інтерфейсу та можливість забезпечення стабільної роботи у випадку нестабільного мережевого середовища. У межах розробки було застосовано GraphQL API Magento 2 у поєднанні з Apollo Client — бібліотекою, яка забезпечує не лише виконання запитів, але й управління станом застосунку на основі принципів реактивності та кешування.

GraphQL було обрано як основний спосіб взаємодії з Magento, оскільки він дозволяє отримувати строго визначені набори даних, що значно зменшує обсяг передавання інформації та оптимізує структуру відповідей. Це особливо важливо у контексті мобільних користувачів, де затримки в мережі можуть впливати на швидкість роботи інтерфейсу. Використання GraphQL сприяло також побудові гнучкої архітектури: кожен компонент застосунку отримує лише ті дані, які йому дійсно необхідні, а структура запиту може адаптуватися залежно від потреб інтерфейсу.

Apollo Client відіграв ключову роль у реалізації як транспортного, так і логічного рівня роботи з даними. Він забезпечує централізоване управління GraphQL-запитами та мутаціями, зберігає кеш відповідей у пам’яті та дозволяє уникати повторних звернень до сервера. У рамках проєкту було налаштовано політики кешування, які визначають поведінку Apollo при формуванні відповіді (cache-first, network-only, cache-and-network), що дало змогу досягти балансу між продуктивністю та актуальністю даних. Так, сторінки каталогу та картки товару використовують стратегію cache-first, що забезпечує миттєве відображення попередньо завантажених даних, тоді як елементи чекауту або профілю користувача працюють у режимі network-only, оскільки критично важливо отримувати найактуальнішу інформацію.

Важливим аспектом архітектури стала обробка мутацій і оптимістичних оновлень. Apollo дозволяє моделювати логіку взаємодії з backend у режимі реального часу, миттєво оновлюючи інтерфейс ще до отримання відповіді сервера. Це створює відчуття «мобільності» інтерфейсу, що є характерною властивістю PWA-застосунків. Наприклад, додавання товару до кошика відображається негайно завдяки оптимістичному запису в Apollo Cache, після чого дані синхронізуються з Magento через GraphQL мутацію. Такий підхід забезпечує плавність взаємодії та стійкість до короткочасних затримок у мережі.

У межах проєкту активно застосовувалися механізми розширення Apollo Cache через модифікатори полів (field policies). Ці політики дозволили контролювати структуру даних у кеші, визначати правила їхнього об’єднання та оновлення без необхідності повторних запитів. Особливо це стало актуальним для кошика, де зміна кількості або складу товарів повинна одразу відображатися в інтерфейсі, навіть якщо серверна відповідь ще не отримана. Таким чином, Apollo Cache фактично виконував роль локального state manager, зменшуючи потребу у зовнішніх бібліотеках на кшталт Redux.

Окремий елемент роботи з GraphQL — це фрагменти, які використовувалися для повторного застосування однакових структур даних у різних компонентах. Це дало можливість підвищити узгодженість застосунку та знизити кількість дубльованого коду. Крім того, завдяки статичному аналізу GraphQL-схеми та інтеграції з інструментами PWA Studio було забезпечено виявлення помилок на етапі розробки, що мінімізувало ризики появи некоректних запитів у продакшн-середовищі.

Особливу увагу було приділено роботі з Apollo в умовах нестабільного з’єднання. Завдяки використанню локального кешу, apollo-cache-persist та вбудованим механізмам черг мутацій застосунок здатний зберігати стан та накопичувати запити для подальшого виконання після відновлення мережі. Це

дозволяє користувачу продовжувати роботу навіть у режимі офлайн, що є характерною властивістю PWA. Таким чином, механізми Apollo перетворюють застосунок на стійкий і автономний інструмент, здатний підтримувати основні функції навіть за межами постійного мережевого доступу.

У результаті організація взаємодії з GraphQL та управління станом клієнта сформували один із найбільш критичних компонентів архітектури розробленого PWA. Взаємодія Apollo Client із Magento GraphQL забезпечила ефективну роботу застосунку, високу швидкодію, гнучкість у побудові компонентної структури та стійкість до мережевих збоїв. Цей підхід дозволив створити масштабоване рішення, яке здатне розширюватися та адаптуватися під потреби сучасних електронних комерційних систем.

У процесі розробки PWA критично важливим є забезпечення ефективної взаємодії з GraphQL API платформи Magento 2. Для цього у прототипі використовується Apollo Client, який надає як декларативний підхід до виконання запитів, так і потужний механізм керування кешем на стороні клієнта. Запити GraphQL у PWA виконуються реактивно, а зміни в Apollo Cache автоматично оновлюють інтерфейс без необхідності повторного рендерингу всієї сторінки.

Одним із ключових запитів є отримання детальної інформації про товар за url_key, який продемонстровано у Лістингу 3. Цей запит використовується на сторінці продукту та забезпечує завантаження назв, цін, медіа та атрибутів товару.

Лістинг 3 — Запит GraphQL для отримання інформації про товар

```
query
    getProductDetail($urlKey:
    String!) {
  products(filter: { url_key:
    { eq: $urlKey } }) {
    items {
```

```
      id
      name
      sku
      url_key
      price_range {
        minimum_price {
          final_price {
            value
            currency
          }
        }
      }
      media_gallery {
        url
        label
      }
      description {
        html
      }
    }
  }
}
```

Значною частиною функціональності PWA є робота з кошиком. Клієнтська частина повинна завантажувати структуру кошика, синхронізувати зміни та оновлювати інтерфейс при додаванні або зміні кількості товарів. Для цього використовується запит отримання кошика за його ідентифікатором, який продемонстровано у Лістингу 4.

Лістинг 4 — Запит GraphQL для отримання кошика

```
query getCart($cartId:
    String!) {
  cart(cart_id: $cartId) {
    id
    items {
      id
      quantity
      product {
        name
        sku
        small_image {
          url
        }
        price_range {
          minimum_price {
            final_price {
              value
              currency
            }
          }
        }
      }
    }
    prices {
      grand_total {
        value
        currency
      }
    }
  }
```

```
}
```

Окрім запитів, PWA активно використовує GraphQL-мутації. Найпоширеніша з них — це додавання товару у кошик, яку продемонстровано у Лістингу 5. У поєднанні з оптимістичними оновленнями Apollo це забезпечує миттєву реакцію інтерфейсу.

Лістинг 5 — Мутація GraphQL для додавання товару в кошик

```
mutation addToCart($cartId:
    String!, $sku: String!,
    $qty: Float!) {
  addSimpleProductsToCart(
    input: {
      cart_id: $cartId,
      cart_items: [
        {
          data: {
            sku: $sku,
            quantity: $qty
          }
        }
      ]
    }
  ) {
    cart {
      id
      items {
        id
        quantity
        product {
```

```
            name
          }
        }
      }
    }
}
```

Ці запити та мутації формують основний комунікаційний шар прототипу PWA, а їх тісна інтеграція з Apollo Cache забезпечує ефективне управління станом додатку, мінімізуючи кількість звернень до серверу та підвищуючи продуктивність.

3.4.2 Архітектура компонентів інтерфейсу та логіка їх зв’язності

Архітектура компонентів інтерфейсу у прогресивному веб-застосунку відіграє ключову роль у забезпеченні масштабованості, структурованості та передбачуваності роботи всієї клієнтської частини. Оскільки PWA функціонує як автономна фронтенд-система, побудова коректної ієрархії компонентів та визначення логіки їхнього зв’язку стає основою для ефективної інтеграції з GraphQL API, оптимального керування станом і стабільного користувацького досвіду. У межах розробки PWA для Magento 2 архітектурною основою було обрано компонентну модель React у поєднанні з Peregrine hooks та Venia UI — наборами логічних та презентаційних компонентів, спеціально створених Adobe для побудови headless-інтерфейсів.

Базовий принцип архітектури полягав у чіткому розмежуванні логічних і візуальних компонентів. Логічні компоненти відповідали за взаємодію з GraphQL API, управління станом, обробку подій та підготовку даних. Вони, як правило, не

містили UI-розмітки, а повертали структуровані дані та функції, які передавалися у презентаційні компоненти. Такий підхід дозволяє ізолювати бізнес-логіку від презентаційного шару, забезпечити повторне використання логічних блоків та спростити тестування. Презентаційні компоненти Venia UI виконували роль інтерфейсного шару, що відповідав за відображення даних, стилізацію, інтерфейсні переходи та взаємодію з користувачем.

Завдяки Peregrine hooks було досягнуто високого рівня модульності архітектури. Кожен hook інкапсулює певну поведінкову логіку: отримання даних, роботу з кошиком, авторизацію, взаємодію з атрибутами товару, навігацію або обробку локального стану. Це дозволяє відокремити поведінкову частину від структурної, а також адаптувати UI до різних сценаріїв без перевантаження компонентів складною логікою. Реактивність hooks у поєднанні з Apollo Client утворила гнучку та плавну модель оновлення даних — компоненти автоматично реагують на зміни у GraphQL-кеші, що усуває потребу в ручному керуванні станом та синхронізацією.

Організація компонентів ґрунтувалася на принципах односпрямованого потоку даних та контрольованої композиції. Головні компоненти — такі як сторінки категорій, картка товару або кошик — виступали контейнерами, які об’єднували кілька логічних блоків і передавали їх у дочірні UI-компоненти. Це дозволяло створювати складні інтерфейсні структури без втрати прозорості у роботі з даними. Наприклад, компонент сторінки товару поєднував логіку вибору атрибутів, обробки графічних елементів, управління станом кошика, обчислення варіантів конфігураційного товару та механізмів доступності. Розподіл логічних залежностей забезпечував легкість модифікації та можливість розширення функціональності.

Суттєвою частиною архітектури стала система маршрутизації, яка визначала доступ до компонентів інтерфейсу залежно від динамічного вмісту.

Оскільки Magento 2 дозволяє формувати URL-структуру на основі тегів чи атрибутів, маршрутизація у PWA повинна була адаптуватися до довільних типів сторінок. Для цього застосовувався механізм urlResolver, який через GraphQL-запит визначає, який тип сутності відповідає певному URL, і на основі цього завантажує відповідний компонент. Така модель забезпечує універсальність маршрутизації та дозволяє динамічно будувати сторінки каталогу, продуктів чи CMS-контенту.

Особливу увагу було приділено оптимізації компонування інтерфейсу. За рахунок використання lazy-loading, розбиття бандла та попереднього завантаження критично важливих компонентів вдалося зменшити час початкового рендерингу й покращити показники продуктивності. Оптимізація графічних компонентів — зокрема, медіагалереї — здійснювалася через поєднання локального кешу Apollo, стратегій кешування Service Worker та React-механізмів відкладеного відображення. Service Worker є невід'ємною частиною застосунку, підключення якого відбувається у окремому файлі та продемонстровано у Лістингу 6.

Лістинг 6 — Підключення Service Worker через окремий модуль

```
import {
    VALID_SERVICE_WORKER_ENVIRONMENT,
    handleMessageFromSW
} from '@magento/peregrine/lib/util/swUtils';

export const registerSW = () => {
    if (VALID_SERVICE_WORKER_ENVIRONMENT &&
globalThis.navigator) {
        window.navigator.serviceWorker
            .register('/sw.js')
            .then(() => {
```

```
                console.log('SW Registered');
            })
            .catch(() => {
                window.console.warn('Failed to register SW.');
            });

        navigator.serviceWorker.addEventListener('message', e
=> {
            const { type, payload } = e.data;
            handleMessageFromSW(type, payload, e);
        });
    }
};
```

Важливим аспектом архітектури стала передбачуваність стану застосунку. Оскільки PWA клієнтський застосунок має працювати як у стабільній мережі, так і за умов тимчасової відсутності з’єднання, логіка компонентів повинна була враховувати стани «pending», «success», «partial offline data» та «error». Саме компоненти інтерфейсу відповідали за коректне відображення цих станів, забезпечуючи безперервність взаємодії навіть тоді, коли backend тимчасово недоступний.

Для наочного підтвердження застосування компонентного підходу у розробленому PWA-застосунку на рисунку 7 наведено приклад реалізації сторінки каталогу товарів. Інтерфейс сторінки сформовано з окремих незалежних компонентів, таких як навігаційна панель, блок фільтрації та сортування, сітка товарних карток та допоміжні елементи керування. Такий підхід забезпечує гнучкість інтерфейсу, можливість повторного використання компонентів та спрощує підтримку й масштабування клієнтської частини застосунку.

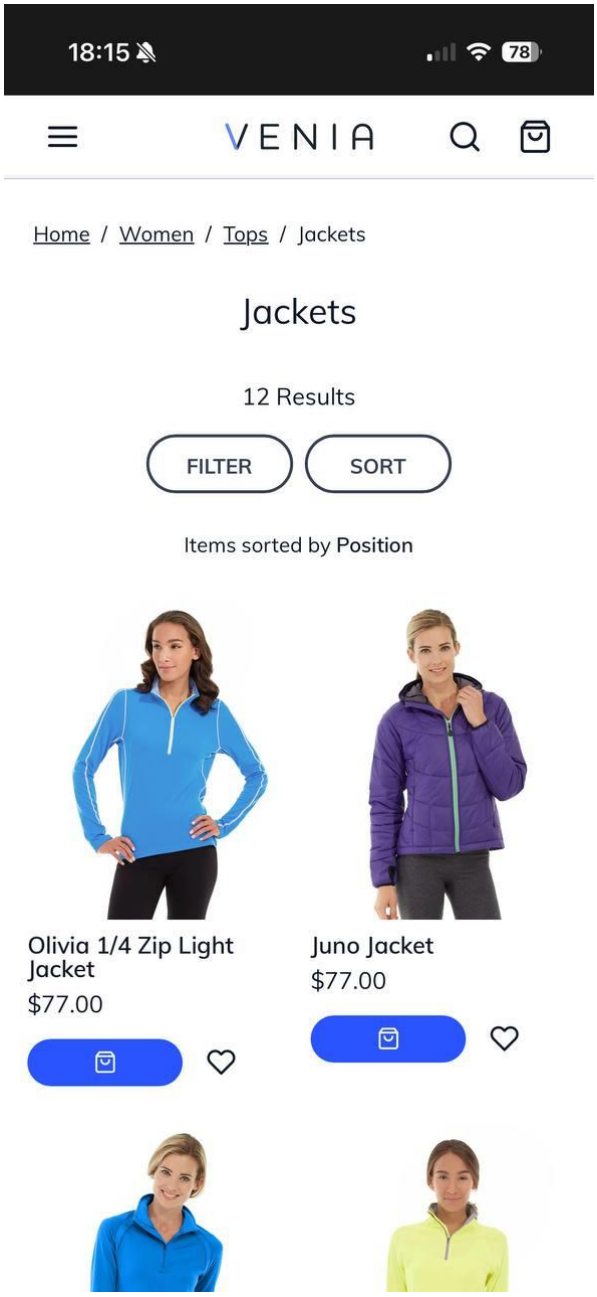


Рисунок 7 — Приклад реалізації компонентної структури сторінки каталогу товарів у PWA-застосунку

У підсумку архітектура компонентів інтерфейсу стала багаторівневою системою, де логічні та UI-компоненти чітко розмежовані, а управління даними здійснюється реактивно та централізовано. Такий підхід дозволив створити

масштабований і стійкий застосунок, який зберігає високу швидкодію та передбачуваність навіть за складних умов роботи.

3.4.3 Механізми обробки користувацьких дій та синхронізація з backend

Ефективне опрацювання користувацьких дій та забезпечення синхронізації між клієнтською частиною PWA і backend Magento 2 є фундаментальними для формування стабільного, передбачуваного та швидкого користувацького досвіду. У традиційних веб-додатках більшість логіки, пов’язаної з обробкою подій і валідацією, делегується серверу. Натомість у PWA значна частина відповідальності переноситься на клієнт, що вимагає побудови складної системи управління даними, черг асинхронних операцій і засобів відновлення стану в умовах нестабільної мережі.

У межах цього проєкту механізми обробки дій користувачів реалізовані через комбінацію React-компонентів, Peregrine hooks і Apollo Client. Кожна взаємодія з інтерфейсом — вибір товару, зміна кількості, відкриття сторінки, додавання товару до кошика, введення адреси при оформленні замовлення — перетворюється на низку локальних операцій, які уніфіковано синхронізуються із сервером через GraphQL мутації. Такий підхід дозволяє забезпечити швидку реакцію інтерфейсу без очікування відповіді сервера, що формує поведінку, наближену до нативних мобільних застосунків.

Одним із ключових механізмів стала система оптимістичних оновлень Apollo Client. Вона дозволяє інтерфейсу відображати результат дії користувача одразу після її виконання — наприклад, зміна кількості товару в кошику чи додавання нового елемента — тоді як фактичний запит до backend відбувається у фоновому режимі. У разі успішної відповіді Apollo синхронізує кеш, а у

випадку помилки — повертає попередній стан і коректно повідомляє користувача. Це забезпечує плавність взаємодії та мінімізує негативний вплив затримок мережі.

Ще одним важливим елементом стала побудова механізмів черг мутацій для роботи в умовах часткової або повної відсутності мережевого з'єднання. Застосунок накопичує дії користувача у локальній структурі даних і виконує їх лише після відновлення з'єднання. Таким чином, взаємодія з кошиком або іншими клієнтськими даними залишається доступною у режимі офлайн, а синхронізація відбувається автоматично, без необхідності повторювати дії. Це особливо корисно у сценаріях мобільного використання, де мережевий сигнал може перериватися.

Логіка обробки користувацьких дій була побудована на принципах уніфікованих потоків (action flows). Кожна подія інтерфейсу трансформується у структуровану дію, яка:

a) оновлює локальний стан через Apollo або внутрішній state hook;
b) викликає мутацію GraphQL у фоновому режимі;
c) за необхідності — активує side-effects, такі як оновлення кешу, навігаційні переходи, повторні запити або показ повідомлень.

Це дозволило створити прозору й передбачувану модель поведінки: кожна дія користувача має визначений шлях, логічні етапи та механізми відновлення після можливих помилок.

Окрему увагу було приділено синхронізації стану між локальними даними та відповідями backend. Магазинні операції часто містять бізнес-логіку, яка змінює стан кошика або товару (наприклад, доступність товару, ціни, мінімальні кількості). Тому після виконання мутацій була впроваджена додаткова перевірка актуальності даних шляхом оновлення частини кешу через політику cache-and-network. Це забезпечило баланс між швидкістю та точністю відображення даних,

дозволивши отримувати актуальні значення без видимих затримок для користувача.

Завдяки централізованим механізмам синхронізації та структурованій моделі обробки подій вдалося створити стійку до помилок інфраструктуру, у якій дії користувача не втрачаються, не дублюються і не генерують непередбачуваних станів. Навіть за умов нестабільного інтернет-з’єднання застосунок демонструє передбачувану поведінку, а користувач отримує можливість взаємодіяти з контентом без перешкод.

Таким чином, система опрацювання користувацьких дій у поєднанні з механізмами фонової синхронізації створила надійний фундамент для усіх бізнес-процесів застосунку. Її гнучка архітектура забезпечує стійкість, швидкість і коректність реакції на дії користувачів, що є ключовими чинниками успіху будь-якого сучасного PWA-рішення у сфері електронної комерції.

3.4.4 Реалізація офлайн-стійкості та роботи у нестабільному мережевому середовищі

Однією з ключових переваг прогресивних веб-застосунків є їхня здатність забезпечувати стабільну роботу незалежно від якості мережевого з’єднання. На відміну від традиційних веб-сайтів, де доступ до контенту та функціональності повністю залежить від доступності сервера, PWA використовує низку технологій, що дозволяють працювати локально, кешувати критичні ресурси та синхронізувати дані лише за потреби. Реалізація офлайн-стійкості стала важливим елементом проєкту, оскільки вона забезпечує безперервність взаємодії з магазином та підвищує загальний рівень користувацького досвіду.

Основним технічним механізмом, що забезпечує роботу застосунку в офлайн-режимі, є Service Worker — фоновий скрипт, який перехоплює мережеві

запити, керує кешами, обробляє сценарії відсутності з’єднання та дозволяє застосунку продовжувати працювати автономно. У розробленому PWA Service Worker генерується автоматично за допомогою інструментів Workbox, інтегрованих у Magento PWA Studio. Це дозволяє конфігурувати різні стратегії кешування залежно від типу ресурсу: статичні файли (CSS, JS), медіаоб’єкти, GraphQL-запити або сторінки каталогу.

Для інформування користувача про зміну мережевого стану у клієнтській частині застосунку реалізовано механізм відстеження подій online/offline, на основі якого динамічно відображаються відповідні повідомлення. Це дозволяє уникнути ситуацій, коли користувач не розуміє причин обмеженої функціональності, та підвищує загальну зручність взаємодії із застосунком.

На рисунку 8 продемонстровано стан застосунку в режимі відсутності мережевого з’єднання, при якому користувач отримує повідомлення про обмеження доступних функцій. Після відновлення підключення до мережі, застосунок автоматично повертається до повнофункціонального режиму без необхідності перезавантаження сторінки. Такий підхід підтверджує ефективність використання архітектури PWA для забезпечення безперервності роботи електронного магазину.

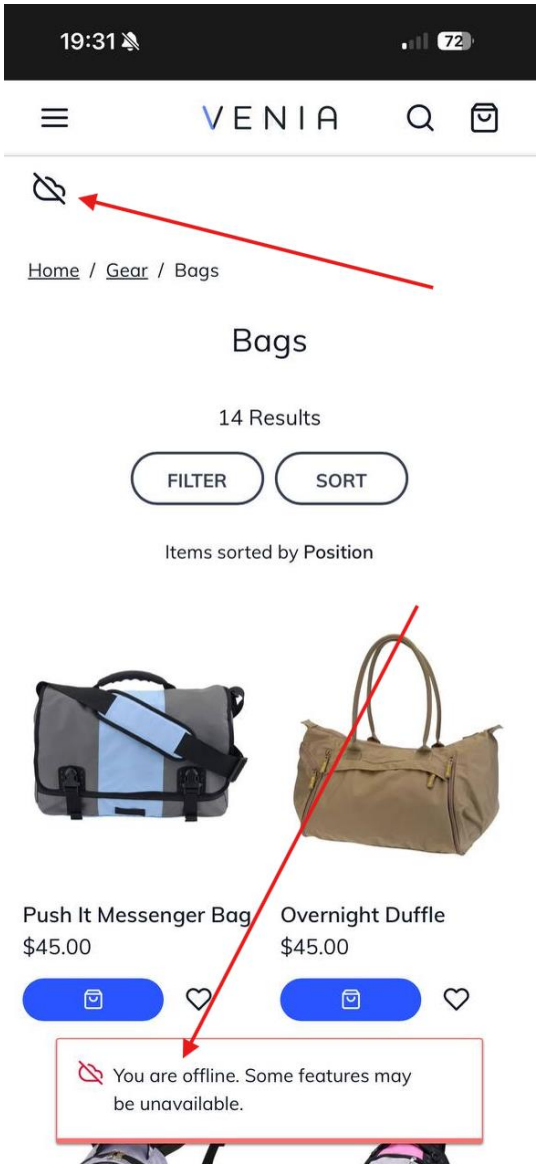


Рисунок 8 — Реакція PWA-застосунку на відсутність мережевого з’єднання (offline-режим)

Після повторного підключення до мережі система автоматично визначає зміну стану з’єднання та переходить до онлайн-режиму без необхідності перезавантаження сторінки або повторної ініціалізації інтерфейсу, як продемонстровано на рисунку 9. Користувач отримує відповідне повідомлення про відновлення доступу до мережі, а функціональні можливості застосунку,

зокрема завантаження актуальних даних каталогу та взаємодія з серверною частиною, відновлюються у фоновому режимі. Така поведінка підтверджує ефективність використання Service Worker та подієво-орієнтованої логіки обробки мережевих станів у PWA-рішенні.

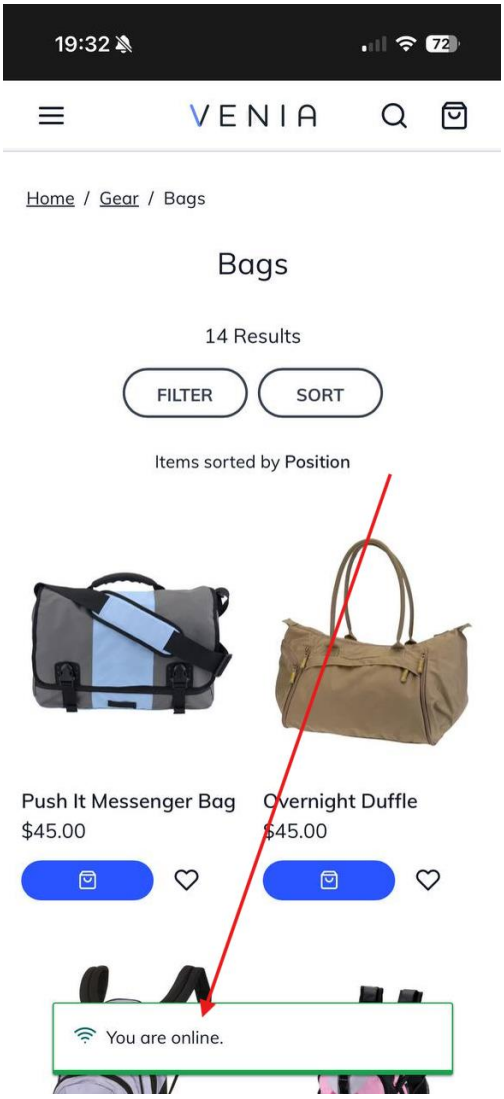


Рисунок 9 — Реакція PWA-застосунку на відновлення мережевого з’єднання

Важливою частиною реалізації стало впровадження runtime caching, яке дозволяє застосунку кешувати дані «на льоту» після першого звернення до

сервера. Завдяки цьому сторінки, які вже були відвідані, доступні для перегляду навіть у повному офлайні. Для медіаконтенту застосовано стратегію cache-first, що забезпечує миттєве завантаження зображень та зменшує затримки під час повторних переглядів. Для динамічних GraphQL-запитів обрано комбіновані стратегії stale-while-revalidate та network-first: вони дозволяють відображати дані з кешу, але при цьому виконувати оновлення у фоновому режимі, коли мережа доступна.

Окремим напрямом стала реалізація fallback-логіки, яка визначає поведінку застосунку у випадку повної втрати з'єднання. Якщо користувач переходить на сторінку, яка ще не була завантажена раніше, PWA надає резервний інтерфейс або пояснення щодо недоступності контенту. Це не тільки запобігає появі помилок, але й формує передбачувану модель взаємодії, у якій користувач не стикається з раптовими зривами роботи застосунку.

Лістинг 7 — Перевірка швидкості мережі

```
export const isFastNetwork = () => {
    if (navigator.connection && 'effectiveType' in
navigator.connection) {
        return navigator.connection.effectiveType === '4g';
    } else {
        return true;
    }
};
```

Значну увагу було приділено реалізації механізмів відкладених операцій (background sync), що дозволяють виконувати зміни у локальному стані навіть у повному офлайні. Наприклад, користувач може додати товар до кошика, змінити його кількість або видалити позицію незалежно від наявності з'єднання. Усі такі операції зберігаються у локальній черзі, а після відновлення мережі Service Worker автоматично синхронізує їх з backend Magento через GraphQL-мутації. У

поєднанні з оптимістичними оновленнями Apollo Client це забезпечує безперервність роботи інтерфейсу та мінімізує вплив мережевих затримок.

Ще одним важливим інструментом стало використання persisted cache, що дає змогу зберігати Apollo Cache у localStorage або IndexedDB. Це дозволило навіть після перезавантаження застосунку забезпечити доступ до попередньо завантажених сторінок, даних товарів, результатів запитів та стану кошика. Таким чином, застосунок відчувається як повноцінний локальний клієнт, а не лише оболонка над сервером.

У процесі проєктування офлайн-стійкості враховувалася необхідність узгодження локального стану з серверними даними, що може стати викликом у розподілених системах. При відновленні мережі застосунок виконував додаткові GraphQL-запити для уточнення критичних значень: доступності товарів, ціни, результатів застосування купонів або адміністрування кошика. Це дозволило уникнути конфліктів станів і забезпечити коректну роботу навіть у складних сценаріях, таких як паралельні зміни з боку сервера. Крім того, у Лістингу 7 представлено код, який відповідає за перевірку швидкості мережі. У разі визначення недостатньої швидкості підключення користувача деякі функції попереднього завантаження будуть вимкнуті, задля зниження навантаження на мережу та більш оптимального завантаження застосунку.

Таким чином, реалізація офлайн-стійкості у PWA для Magento 2 стала комплексним поєднанням Service Worker, механізмів кешування Workbox, відкладених операцій, синхронізації GraphQL-мутацій та локального управління станом. Ці технології забезпечили автономність застосунку, підвищили швидкодію, мінімізували залежність від мережевої доступності та дозволили формувати сучасний, зручний і надійний користувацький досвід. Такий підхід відповідає сучасним вимогам до веб-застосунків і демонструє переваги PWA у порівнянні з традиційними веб-інтерфейсами.

3.5 Висновки до розділу 3

У рамках третього розділу було здійснено повний цикл проєктування та розробки клієнтської частини прогресивного веб-застосунку, який функціонує як автономний PWA-фронтенд для платформи Magento 2. Розроблений прототип поєднує сучасні підходи до побудови інтерфейсів, архітектури даних та мережевої взаємодії, а також демонструє практичне застосування технологій, що визначають сучасний рівень розвитку електронної комерції.

У процесі роботи було проведено аналіз вимог і вибір оптимального технологічного стеку, який включає React, екосистему Magento PWA Studio, Apollo Client та GraphQL API. Вибір цих технологій дозволив сформувати масштабовану, модульну та передбачувану архітектуру застосунку, у якій дані синхронізуються з backend у режимі реального часу, а інтерфейс зберігає високу продуктивність за будь-яких умов мережевої роботи.

Особливу увагу було приділено організації взаємодії з GraphQL та управлінню станом, що забезпечило реактивне оновлення інтерфейсу, ефективне кешування даних та мінімізацію повторних мережевих звернень. Завдяки впровадженню Apollo Cache та політик кешування прототип працює швидко й передбачувано, а його компоненти залишаються узгодженими навіть у складних сценаріях взаємодії.

Архітектура інтерфейсу була побудована на основі чіткої взаємодії логічних і презентаційних компонентів, що підвищило керованість і гнучкість застосунку. Логіка отримання даних, бізнес-правила та механізми обробки подій були відокремлені від візуальної частини, що створило умови для подальшого розвитку й масштабування рішення без втрати структурної цілісності.

Було реалізовано інженерні механізми обробки користувацьких дій, які забезпечують передбачувані реакції застосунку, підтримку оптимістичних оновлень, черги мутацій і гарантовану синхронізацію з backend Magento. Це дозволило розробленому PWA працювати плавно та стабільно навіть у разі затримок мережі, перебоїв зв’язку або часткової втрати доступу до серверу.

Особливо важливою частиною проєкту стало впровадження офлайн-стійкості. Завдяки Service Worker, Workbox та механізмам runtime caching застосунок здатний працювати у повноцінному режимі без доступу до інтернету, надаючи користувачу можливість переглядати раніше відкриті сторінки, маніпулювати кошиком та зберігати свої дії для подальшої автоматичної синхронізації. Реалізація fallback-логіки та background sync забезпечила надійність і сучасний рівень користувацького досвіду, характерний для нативних мобільних застосунків.

Загалом у межах цього розділу було створено функціональний прототип PWA-застосунку для Magento 2, який демонструє практичну реалізацію headless-архітектури, інтеграцію з GraphQL API, реактивне управління станом та механізми автономної роботи. Результати роботи підтверджують, що PWA є ефективним і перспективним підходом до розробки інтерфейсів електронної комерції, здатним забезпечити високу продуктивність, стабільність і гнучкість при подальшому розширенні функціональності.

РОЗДІЛ 4 ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ВИКОРИСТАННЯ PWA У СИСТЕМАХ ЕЛЕКТРОННОЇ КОМЕРЦІЇ

4.1 Мета та методика дослідження ефективності PWA-рішення

Метою даного дослідження є комплексна оцінка ефективності використання прогресивних веб-застосунків у системах електронної комерції на прикладі розробленого PWA-прототипу, інтегрованого з платформою Magento 2. У межах дипломної роботи PWA розглядається не лише як окремий технологічний інструмент, а як концептуальний підхід до побудови сучасного клієнтського інтерфейсу електронного магазину, що поєднує властивості традиційних веб-застосунків і нативних мобільних рішень. Основний акцент дослідження спрямовано на визначення того, наскільки впровадження PWA дозволяє вирішити ключові проблеми класичних веб-магазинів, пов’язані з продуктивністю, швидкістю взаємодії, стабільністю роботи та ефективністю повторних відвідувань.

У процесі дослідження передбачалося з’ясувати, чи забезпечує використання PWA реальне покращення технічних характеристик системи електронної комерції у порівнянні з традиційним веб-інтерфейсом, що працює за класичною серверно-орієнтованою моделлю. Зокрема, дослідження було спрямоване на аналіз часу завантаження сторінок, обсягу переданих мережевих даних, поведінки застосунку при повторному відкритті та рівня залежності користувацького інтерфейсу від стабільності мережевого з’єднання. Окремо оцінювалася роль Service Worker та механізмів кешування у формуванні більш швидкої та передбачуваної взаємодії з електронним магазином.

Для досягнення поставленої мети було обрано експериментально-аналітичну методику дослідження, що ґрунтується на практичному тестуванні

реального програмного продукту. На відміну від суто теоретичного аналізу або оглядового порівняння технологій, у даній роботі ефективність PWA оцінюється на основі результатів функціонування розробленого прототипу в реальних умовах. Такий підхід дозволяє не лише підтвердити або спростувати теоретичні переваги PWA, але й виявити практичні особливості його використання у системах електронної комерції.

Методика дослідження передбачала використання стандартних інструментів аналізу продуктивності веб-застосунків, які є загальноприйнятими у сучасній практиці веб-розробки. Основними інструментами стали Google Chrome DevTools та Lighthouse, що дозволяють отримати кількісні показники продуктивності, такі як час до першого відображення контенту, швидкість взаємодії, стабільність рендерингу та відповідність застосунку вимогам до прогресивних веб-застосунків. Використання цих інструментів забезпечує відтворюваність результатів і дозволяє порівнювати отримані показники з результатами інших досліджень у цій галузі.

У межах дослідження було сформовано два основні сценарії тестування: робота системи електронної комерції з використанням PWA-інтерфейсу та робота аналогічного веб-магазину без застосування PWA-технологій. Для кожного зі сценаріїв виконувалися вимірювання часу завантаження сторінок при першому та повторному відвідуванні, аналіз обсягу мережевого трафіку, а також оцінка поведінки інтерфейсу при обмеженні швидкості мережі або її повній відсутності. Такий підхід дозволив здійснити об’єктивне порівняння ефективності різних архітектурних рішень за однакових умов експлуатації.

Особливу увагу у методиці дослідження було приділено аналізу повторних відвідувань користувачів, оскільки саме цей аспект є критично важливим для електронної комерції. Завдяки використанню Service Worker та локального кешування PWA має потенціал значно зменшувати час повторного завантаження

сторінок і знижувати залежність від серверної частини. У рамках дослідження оцінювалося, наскільки ці механізми реально впливають на швидкість взаємодії з магазином та стабільність відображення контенту.

Окрім кількісних показників, у дослідженні також враховувалися якісні характеристики роботи застосунку, зокрема передбачуваність поведінки інтерфейсу, відсутність різких затримок або збоїв та загальне відчуття цілісності користувацького досвіду. Хоча такі характеристики не завжди можуть бути точно виміряні числовими показниками, їх аналіз є важливим для комплексної оцінки ефективності PWA-рішення, особливо у контексті електронної комерції, де швидкість і стабільність безпосередньо впливають на рішення користувача щодо здійснення покупки.

Таким чином, обрана методика дослідження поєднує практичний експериментальний підхід із кількісним аналізом продуктивності та якісною оцінкою поведінки веб-застосунку. Це дозволяє зробити обґрунтовані висновки щодо ефективності використання прогресивних веб-застосунків у системах електронної комерції та оцінити, наскільки впровадження PWA відповідає поставленій у дипломній роботі меті.

4.2 Аналіз продуктивності PWA у порівнянні з традиційним веб-застосунком

Одним із ключових критеріїв оцінки ефективності прогресивних веб-застосунків є рівень їх продуктивності та можливостей у порівнянні з традиційними веб-рішеннями. Для систем електронної комерції продуктивність має особливе значення, оскільки безпосередньо впливає на швидкість взаємодії користувача з інтерфейсом, стабільність роботи магазину та ймовірність завершення покупки. У межах даного дослідження продуктивність розробленого

PWA було проаналізовано шляхом порівняння його роботи з класичним веб-застосунком, що не використовує механізми Service Worker, кешування та офлайн-підтримки.

Порівняльний аналіз проводився за однакових умов тестування, що дозволило забезпечити об’єктивність отриманих результатів. Для обох варіантів застосунку — PWA та традиційного веб-інтерфейсу — використовувалося однакове серверне середовище, однакові дані електронного магазину та ідентичні сценарії користувацької взаємодії. Основна увага приділялася таким показникам, як час початкового завантаження сторінки, швидкість повторного відкриття контенту, обсяг переданих мережевих даних та стабільність рендерингу інтерфейсу.

Результати аналізу показали, що при першому завантаженні сторінки різниця між PWA та традиційним веб-застосунком є помірною, оскільки в обох випадках необхідно отримати основні ресурси з серверної частини. Водночас уже на цьому етапі PWA демонструє більш оптимізовану поведінку завдяки ефективному управлінню статичними ресурсами та асинхронному завантаженню окремих модулів. Це дозволяє зменшити блокування основного потоку виконання та забезпечити швидше відображення базового інтерфейсу.

Найбільш суттєва різниця між досліджуваними підходами проявляється при повторних відвідуваннях сторінок. Традиційний веб-застосунок при кожному оновленні сторінки або переході між розділами повторно звертається до серверу для завантаження ресурсів, що призводить до додаткових затримок та збільшення мережевого навантаження. Натомість PWA активно використовує механізми кешування, що дозволяють зберігати статичні ресурси та частину динамічних даних локально. У результаті повторне відкриття сторінок відбувається значно швидше, а інтерфейс реагує на дії користувача практично миттєво [13].

Аналіз обсягу переданих даних підтвердив, що PWA зменшує кількість мережевого трафіку, особливо у сценаріях багаторазової взаємодії з магазином. Завдяки використанню Service Worker та стратегій кешування значна частина запитів обслуговується без звернення до серверу, що не лише підвищує швидкодію, але й знижує навантаження на серверну інфраструктуру. Для електронної комерції це має важливе практичне значення, оскільки дозволяє забезпечити стабільну роботу магазину навіть за високого рівня користувацької активності.

Окремо було проаналізовано поведінку застосунків в умовах обмеженої пропускної здатності мережі. При штучному зниженні швидкості з’єднання традиційний веб-застосунок демонструє суттєве зростання часу очікування та зниження чутливості інтерфейсу. У деяких випадках користувач змушений чекати повного завантаження сторінки перед тим, як продовжити взаємодію. Натомість PWA зберігає відносно стабільну продуктивність, оскільки значна частина інтерфейсу та даних вже доступна локально, а мережеві запити виконуються у фоновому режимі.

Для більш комплексної оцінки ефективності прогресивних веб-застосунків у контексті електронної комерції доцільно порівняти PWA не лише з традиційними веб-сайтами, але й з нативними мобільними застосунками. Порівняльний аналіз прогресивних веб-застосунків, нативних застосунків, а також традиційних веб рішень продемонстровано у Таблиці 1. Такий підхід дозволяє визначити місце PWA серед основних моделей клієнтських інтерфейсів та проаналізувати їхні сильні й слабкі сторони з точки зору продуктивності, функціональності та експлуатаційних характеристик. Узагальнене порівняння зазначених підходів наведено в аналітичній таблиці.

Таблиця *1 – Аналітична таблиця порівняння PWA, нативних застосунків та Web*

	PWA	**Нативні мобільні застосунки**	**Традиційні веб-сайти**
Швидкодія	Висока, завдяки кешуванню та App Shell	Найвища, максимальне використання ресурсів	Середня, повне перезавантаження сторінок
Офлайн-режим	Є через Service Worker	Повноцінний офлайн-режим	Відсутній або мінімальний
Встановлення на пристрій	Можливе в 1 клік, без App Store	Потребує магазину застосунків	Не можливе
Доступ до функцій пристрою	Обмежений браузером	Повний доступ до апаратних API	Дуже обмежений
Push-сповіщення	Присутні	Є, найбільш розширені	Обмежено (тільки браузерні)
Продуктивність на мобільних пристроях	Висока, але залежить від браузера	Найвища	Залежить від мережі та сервера
Вартість розробки	Низька/середня	Висока (дві команди)	Низька
Швидкість оновлень	Миттєва, без App Store	Затримки через модерацію	Миттєва
Залежність від платформи	Мінімальна	Повна (iOS/Android)	Мінімальна

Аналіз наведених у таблиці характеристик свідчить, що прогресивні веб-застосунки займають проміжне, але стратегічно важливе положення між традиційними веб-сайтами та нативними мобільними застосунками. За показниками швидкодії PWA істотно перевершують класичні веб-рішення

завдяки використанню механізмів кешування, App Shell-архітектури та Service Worker, хоча в окремих випадках можуть поступатися нативним застосункам, які мають безпосередній доступ до апаратних ресурсів пристрою.

Особливо важливою перевагою PWA у контексті електронної комерції є підтримка офлайн-режиму, яка відсутня або реалізована в обмеженому вигляді у традиційних веб-сайтах. Завдяки Service Worker користувач може взаємодіяти з магазином навіть за відсутності стабільного мережевого з'єднання, що позитивно впливає на доступність контенту та безперервність користувацького досвіду [18].

На відміну від нативних мобільних застосунків, PWA не потребують публікації в магазинах застосунків і можуть бути встановлені на пристрій безпосередньо з браузера. Це суттєво зменшує вартість розробки та супроводу, оскільки відпадає необхідність підтримки окремих команд для різних платформ та проходження процедур модерації. Водночас PWA забезпечують миттєве оновлення функціональності без участі користувача, що є критично важливим для електронної комерції, де зміни бізнес-логіки та асортименту відбуваються регулярно.

Обмеження PWA, зокрема частковий доступ до апаратних можливостей пристрою, у більшості сценаріїв електронної комерції не є критичними. Основні функції інтернет-магазину — перегляд каталогу, робота з кошиком, оформлення замовлення та отримання сповіщень — повністю підтримуються в межах можливостей сучасних браузерів. Таким чином, прогресивні веб-застосунки забезпечують оптимальний баланс між продуктивністю, функціональністю та економічною доцільністю.

Узагальнюючи результати порівняльного аналізу, можна стверджувати, що PWA є ефективною альтернативою як традиційним веб-сайтам, так і нативним мобільним застосункам у сфері електронної комерції. Вони дозволяють суттєво

підвищити продуктивність клієнтської частини, зменшити залежність від мережі та скоротити витрати на розробку, що підтверджує доцільність використання PWA у сучасних e-commerce системах.

Результати тестування також свідчать про більш передбачувану поведінку PWA у порівнянні з класичним веб-застосунком. Завдяки реактивній архітектурі та централізованому управлінню станом інтерфейс PWA швидше реагує на дії користувача, навіть якщо відповідь сервера затримується. Це створює відчуття безперервності роботи, що є характерною рисою нативних мобільних застосунків і позитивно впливає на сприйняття якості електронного магазину.

Узагальнюючи результати аналізу, можна стверджувати, що використання прогресивного веб-застосунку забезпечує помітне підвищення продуктивності у порівнянні з традиційним веб-підходом, особливо у сценаріях повторної взаємодії та роботи за нестабільного мережевого з’єднання. Отримані результати підтверджують, що PWA не є лише концептуальним або маркетинговим рішенням, а виступає ефективним інструментом оптимізації роботи систем електронної комерції, здатним забезпечити швидший доступ до контенту, зменшити мережеві затримки та підвищити загальну стабільність користувацького інтерфейсу.

4.3 Аналіз поширеності використання PWA у сучасних системах електронної комерції

Упродовж останніх років прогресивні веб-застосунки набули значного поширення у сфері електронної комерції, що зумовлено зміною поведінки користувачів та зростанням ролі мобільних пристроїв у процесі онлайн-покупок. Більшість сучасних інтернет-магазинів орієнтовані на мобільний трафік, однак традиційні веб-сайти часто не здатні забезпечити достатній рівень швидкодії та

стабільності роботи на мобільних платформах. У цьому контексті PWA розглядається як компромісне рішення, яке поєднує доступність веб-технологій із функціональними можливостями нативних застосунків.

Аналіз сучасних тенденцій розвитку електронної комерції свідчить про поступове зростання інтересу до PWA з боку великих торговельних платформ і міжнародних компаній [19]. Значна частина відомих e-commerce проєктів впроваджує PWA або окремі його елементи з метою підвищення швидкості завантаження сторінок, зменшення показника відмов та покращення користувацької взаємодії. Такі компанії, як Alibaba, AliExpress, Flipkart, Walmart і Starbucks, використовують PWA як основний або додатковий канал взаємодії з клієнтами, що підтверджує практичну доцільність даного підходу.

Однією з ключових причин поширення PWA у системах електронної комерції є прагнення зменшити залежність бізнесу від магазинів мобільних застосунків. Розробка та підтримка нативних застосунків для різних операційних систем потребує значних фінансових і людських ресурсів, а також проходження процедур модерації та оновлення через App Store або Google Play. PWA, у свою чергу, дозволяє забезпечити встановлення застосунку безпосередньо з браузера, що значно спрощує доступ користувачів до сервісу та знижує бар’єр входу.

Актуальність використання прогресивних веб-застосунків у сучасних системах електронної комерції підтверджується не лише технічними перевагами PWA, але й результатами галузевих аналітичних досліджень. Згідно з аналітичними оглядами міжнародних дослідницьких компаній, ринок прогресивних веб-застосунків демонструє стійке зростання, а одним із ключових напрямів їх впровадження є саме сфера електронної комерції.

Основними чинниками зростання популярності PWA визначаються підвищені вимоги до продуктивності веб-застосунків, домінування мобільного трафіку та прагнення компаній зменшити витрати на розробку і супровід

нативних мобільних застосунків. Це свідчить про те, що впровадження PWA є не тимчасовим технологічним експериментом, а обґрунтованим стратегічним рішенням, яке відповідає сучасним тенденціям розвитку цифрової комерції [20].

Важливим фактором, який стимулює впровадження PWA, є також зростання вимог до швидкості роботи веб-застосунків. Дослідження у сфері електронної комерції показують, що навіть незначне збільшення часу завантаження сторінки може призвести до зниження конверсії та втрати потенційних клієнтів. Завдяки використанню Service Worker, локального кешування та оптимізованих механізмів завантаження ресурсів PWA дозволяє суттєво скоротити час доступу до контенту, особливо при повторних відвідуваннях, що є критично важливим для інтернет-магазинів.

Поширення PWA також пов’язане з розвитком headless-архітектур у електронній комерції, де клієнтська частина відокремлюється від серверної логіки. Такі платформи, як Magento 2, активно підтримують API-орієнтований підхід, що створює сприятливі умови для інтеграції PWA як незалежного фронтенд-рішення. У цьому випадку прогресивний веб-застосунок виступає універсальним клієнтом, який може бути адаптований до різних бізнес-вимог без суттєвих змін серверної частини.

Слід зазначити, що поширення PWA не є рівномірним у всіх сегментах електронної комерції. [21] Найбільш активно даний підхід використовується у великих торговельних проєктах із високим рівнем мобільного трафіку та міжнародною аудиторією. Водночас для малих і середніх інтернет-магазинів PWA стає привабливою альтернативою нативним застосункам завдяки нижчій вартості впровадження та простоті масштабування. Це свідчить про універсальність PWA як технологічного рішення, яке може бути адаптоване до різних масштабів бізнесу.

Узагальнюючи результати аналізу, можна зробити висновок, що використання PWA у сучасних системах електронної комерції є не тимчасовою тенденцією, а стійким напрямом розвитку веб-технологій [22]. Зростання кількості проєктів, що впроваджують прогресивні веб-застосунки, підтверджує їх практичну ефективність та відповідність сучасним вимогам ринку. Це, у свою чергу, обґрунтовує актуальність теми дипломної роботи та доцільність використання PWA як перспективного інструменту розвитку електронної комерції.

4.4 Аналіз відповідності розробленого PWA поставленим цілям роботи

У межах даної дипломної роботи було поставлено за мету дослідити можливості інтеграції прогресивного веб-застосунку з системою електронної комерції, а також оцінити ефективність такого підходу у порівнянні з традиційними веб-рішеннями. Для досягнення цієї мети було розроблено та реалізовано PWA-прототип, інтегрований з платформою Magento 2, що дозволило на практиці перевірити теоретичні положення, розглянуті у попередніх розділах роботи.

Результати проведеного дослідження свідчать, що розроблений прогресивний веб-застосунок у повній мірі відповідає поставленим цілям підвищення продуктивності клієнтської частини системи електронної комерції. Реалізація архітектури PWA з використанням Service Worker, локального кешування та асинхронної взаємодії з бекендом забезпечила зменшення часу завантаження сторінок та підвищення швидкості реакції інтерфейсу, особливо при повторних відвідуваннях електронного магазину [23]. Це підтверджує

доцільність застосування PWA як інструменту оптимізації веб-інтерфейсів електронної комерції.

Однією з ключових цілей роботи було дослідження можливостей забезпечення стабільної роботи інтернет-магазину за умов нестабільного або обмеженого мережевого з’єднання. Використання механізмів офлайн-кешування дозволило реалізувати доступ до основного контенту застосунку навіть за відсутності активного підключення до мережі. Таким чином, розроблений PWA-прототип відповідає вимогам підвищення надійності та безперервності користувацького досвіду, що є особливо важливим для сучасних систем електронної комерції.

Поставлена у роботі мета також передбачала оцінку архітектурної доцільності використання PWA у поєднанні з платформою Magento 2. Реалізація інтеграції через GraphQL API підтвердила ефективність headless-підходу, за якого клієнтська частина застосунку є незалежною від серверної реалізації [25]. Така архітектура забезпечує гнучкість, масштабованість та можливість подальшого розвитку системи без необхідності суттєвих змін у бекенд-логіці.

Для порівняння з розробленим PWA-застосунком на рисунку 10 наведено інтерфейс сторінки каталогу товарів традиційного веб-магазину, який функціонує на основі стандартного frontend-рішення платформи Magento 2. Даний інтерфейс використовується як базовий варіант для оцінки ефективності впровадження PWA-підходу та аналізу змін у швидкодії, адаптивності й зручності користування.

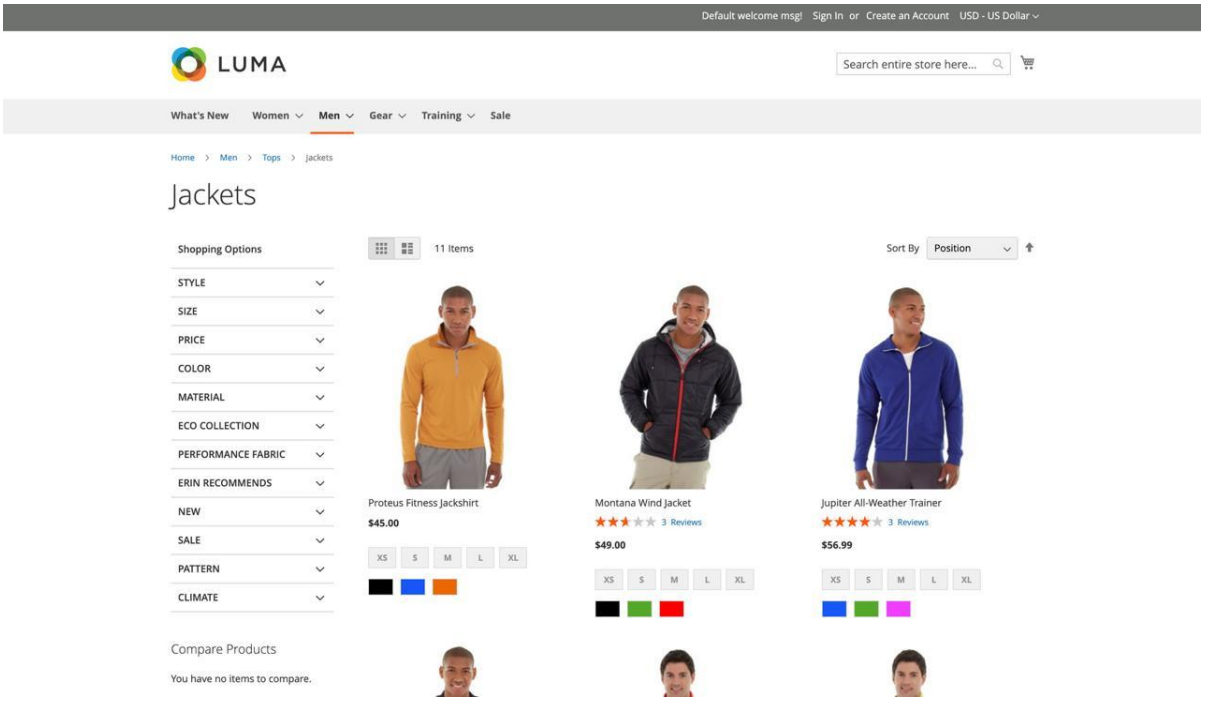


Рисунок 10 — Приклад реалізації компонентної структури сторінки каталогу товарів у PWA-застосунку

На рисунку 11 наведено відповідну сторінку каталогу товарів, реалізовану у вигляді Progressive Web Application, яка функціонально відповідає сторінці традиційного веб-магазину, зображеній на попередньому рисунку. Представлений інтерфейс демонструє практичну реалізацію PWA-підходу та дозволяє здійснити порівняльний аналіз традиційної моделі побудови клієнтського інтерфейсу з сучасним прогресивним веб-рішенням. Використання PWA забезпечує підвищену адаптивність інтерфейсу, оптимізацію завантаження контенту та покращення взаємодії з користувачем, особливо на мобільних пристроях.

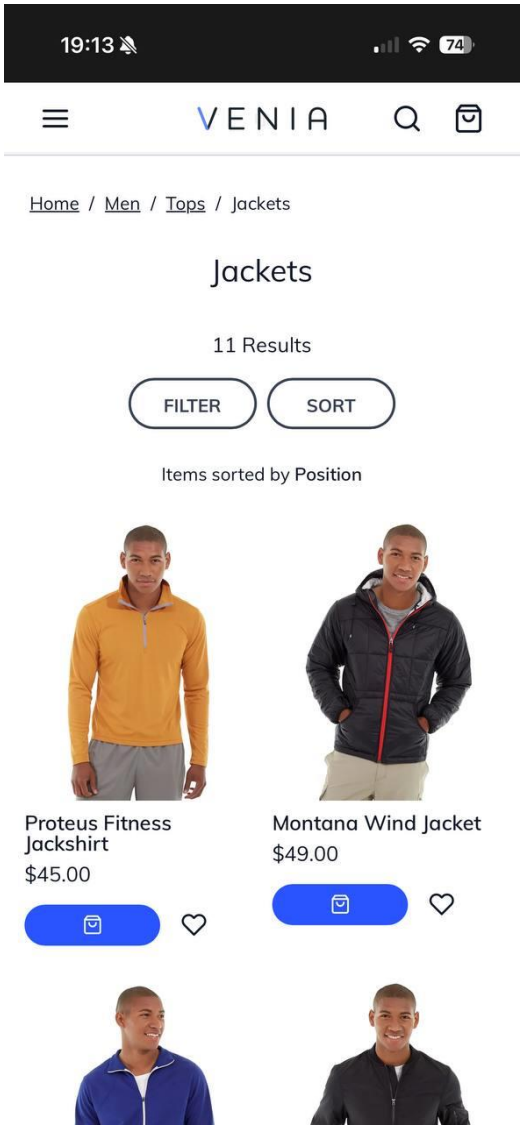


Рисунок 11 — Приклад реалізації сторінки каталогу товарів у PWA-магазині на мобільному пристрої

Важливим аспектом відповідності поставленим цілям є також економічна та організаційна доцільність обраного рішення. Розроблений PWA не потребує створення окремих нативних застосунків для різних платформ, що дозволяє зменшити витрати на розробку та супровід програмного продукту. Крім того, можливість миттєвого оновлення функціональності без участі користувача

відповідає сучасним вимогам до гнучкості та оперативності розвитку електронних комерційних систем.

Однією з ключових характеристик Progressive Web Application є можливість встановлення застосунку на головний екран мобільного пристрою без використання магазинів додатків. На рисунку 12 продемонстровано результат додавання розробленого PWA-застосунку на робочий стіл мобільного пристрою, що підтверджує відповідність реалізованого рішення основним вимогам до PWA та забезпечує користувачеві досвід, наближений до нативного мобільного застосунку.

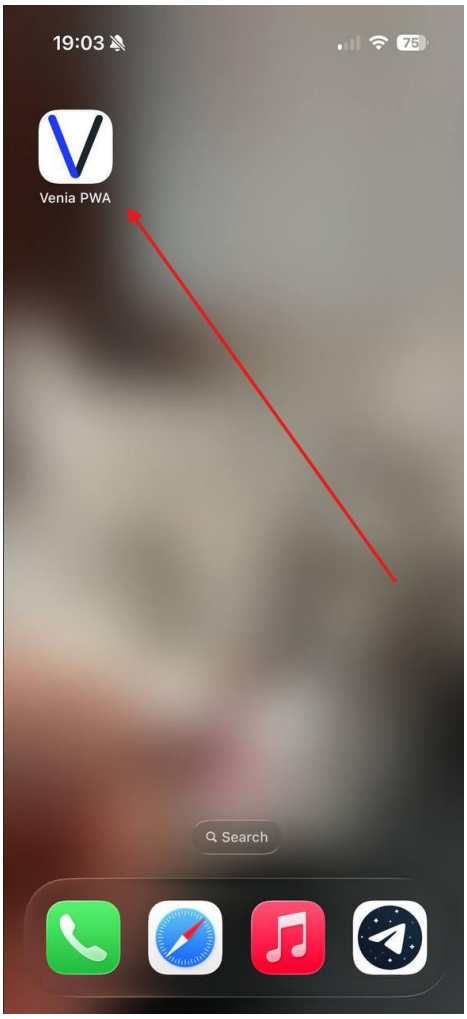


Рисунок 12 — Встановлення PWA-застосунку на головний екран мобільного пристрою

Аналіз результатів дослідження та порівняння їх із цілями, визначеними на початку роботи, дозволяє зробити висновок, що розроблений прогресивний веб-застосунок повністю відповідає поставленим завданням. Реалізоване PWA-рішення продемонструвало ефективність як з технічної, так і з практичної точки зору, підтвердивши доцільність використання прогресивних веб-застосунків у системах електронної комерції та актуальність обраної теми дипломної роботи.

4.5 Висновки до розділу 4

У четвертому розділі дипломної роботи було проведено комплексне дослідження ефективності використання прогресивного веб-застосунку у системах електронної комерції. Основна увага була зосереджена на оцінці продуктивності розробленого PWA-рішення, аналізі тенденцій його поширення та перевірці відповідності реалізованого прототипу поставленим цілям роботи.

У ході дослідження встановлено, що використання PWA дозволяє суттєво покращити показники швидкодії клієнтської частини електронного магазину у порівнянні з традиційними веб-застосунками. Завдяки застосуванню Service Worker, локального кешування та асинхронної взаємодії з бекендом забезпечується скорочення часу завантаження сторінок, підвищення швидкості реакції інтерфейсу та стабільна робота застосунку при повторних відвідуваннях і за умов нестабільного мережевого з’єднання.

Аналіз поширеності використання прогресивних веб-застосунків у сучасних системах електронної комерції підтвердив актуальність обраної теми дослідження. Галузеві аналітичні матеріали та публічні кейси провідних компаній свідчать про зростання інтересу до PWA як до перспективного

інструменту підвищення ефективності веб-систем електронної комерції. Це дозволяє розглядати PWA не як експериментальну технологію, а як обґрунтоване та практично доцільне рішення [25].

Проведений аналіз відповідності розробленого PWA поставленим цілям роботи показав, що реалізований прототип повністю виконує визначені завдання. Розроблене рішення забезпечує високу продуктивність, гнучкість архітектури та можливість подальшого масштабування, що підтверджує доцільність використання PWA у поєднанні з платформою Magento 2 для побудови сучасних систем електронної комерції.

Отримані результати дослідження дозволяють зробити висновок, що прогресивні веб-застосунки є ефективним і перспективним напрямом розвитку клієнтських інтерфейсів електронної комерції [26]. Матеріали розділу створюють основу для формування загальних висновків дипломної роботи та підтверджують досягнення поставленої мети дослідження.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи досліджено теоретичні засади технології Progressive Web Application та визначено її місце серед сучасних підходів до розробки клієнтських веб-застосунків. Встановлено, що PWA поєднує переваги класичних веб-рішень і нативних мобільних застосунків, забезпечуючи високу продуктивність, адаптивність та стабільність роботи в умовах обмеженого або нестабільного мережевого з'єднання.

1. Проаналізовано можливості інтеграції PWA з сучасними системами електронної комерції та розглянуто підходи до реалізації прогресивних веб-застосунків на базі платформ Magento 2, Shopify та WooCommerce. На основі порівняльного аналізу обґрунтовано доцільність використання Magento 2 як платформи для реалізації PWA-рішення з урахуванням її API-орієнтованої архітектури та підтримки headless-підходу.

2. У межах практичної частини роботи спроєктовано та реалізовано прототип Progressive Web Application для системи електронної комерції. Під час розробки використано сучасний технологічний стек, реалізовано інтеграцію з backend API, застосовано GraphQL для оптимізації обміну даними, а також механізми кешування та офлайн-стійкості з використанням Service Workers.

3. Проведено аналіз продуктивності розробленого PWA-рішення у порівнянні з традиційним веб-застосунком. Отримані результати свідчать про зменшення часу завантаження сторінок, покращення швидкодії інтерфейсу та підвищення стабільності роботи клієнтської частини, що є критично важливим для систем електронної комерції з високими вимогами до користувацького досвіду.

4. Досліджено поширеність використання Progressive Web Application у сучасних e-commerce проєктах та підтверджено зростаючу зацікавленість

бізнесу у впровадженні PWA як альтернативи нативним мобільним застосункам. Це зумовлено можливістю зниження витрат на розробку, спрощення процесу оновлення та покращення доступності застосунку для користувачів.

5. У результаті виконання роботи підтверджено, що розроблене PWA-рішення відповідає поставленим цілям дослідження та може бути ефективно використане як клієнтська частина системи електронної комерції. Отримані результати мають практичну цінність та можуть бути використані при проєктуванні і впровадженні сучасних веб-застосунків у реальних комерційних проєктах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Вєтров С.О., магістрант, Заяц В. І., доцент — науковий керівник. Інтеграція Progressive Web Application (PWA) з системою електронної комерції та порівняльний аналіз з традиційними системами. *Молода наука-2025* : зб. наук. праць студентів, аспірантів і молодих вчених. Запоріжжя : ЗНУ, 2025. Т. 5. С. 381–382.

2. W3C. Service Workers Specification. URL: https://www.w3.org/TR/service-workers/ (дата звернення: 15.04.2025).

3. W3C. Web App Manifest Specification. URL: https://www.w3.org/TR/appmanifest/ (дата звернення: 18.04.2025).

4. Mozilla Developer Network. Progressive Web Applications (PWA). URL: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps (дата звернення: 22.04.2025).

5. Mozilla Developer Network. Service Worker API. URL: https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API (дата звернення: 26.04.2025).

6. Adobe Commerce. Magento PWA Studio Documentation. URL: https://developer.adobe.com/commerce/pwa-studio/ (дата звернення: 07.05.2025).

7. Adobe Commerce. Venia Reference Storefront. URL: https://developer.adobe.com/commerce/pwa-studio/venia-pwa-concept/ (дата звернення: 12.05.2025).

8. Apollo GraphQL. Apollo Client Documentation. URL: https://www.apollographql.com/docs/react/ (дата звернення: 18.05.2025).

9. GraphQL Foundation. GraphQL Specification. URL: https://spec.graphql.org/ (дата звернення: 24.05.2025).

10. Meta Platforms Inc. React Documentation. URL: https://react.dev/ (дата

звернення: 24.05.2025).

11. Shopify Engineering. Headless Commerce Architecture. URL: https://shopify.engineering/headless-commerce (дата звернення: 01.07.2025).

12. Shopify Developers. Hydrogen and Storefront API. URL: https://shopify.dev/docs/custom-storefronts (дата звернення: 01.07.2025).

13. Nielsen Norman Group. Mobile UX: Native Apps vs Mobile Web. URL: https://www.nngroup.com/articles/mobile-native-apps/ (дата звернення: 01.07.2025).

14. Nielsen Norman Group. Progressive Web Apps: UX Benefits. URL: https://www.nngroup.com/articles/progressive-web-apps/ (дата звернення: 12.09.2025).

15. Gartner. Market Guide for Progressive Web Apps. URL: https://www.gartner.com/en/documents/3989797 (дата звернення: 12.09.2025).

16. IEEE Xplore. Progressive Web Applications: A Literature Review. URL: https://ieeexplore.ieee.org/document/8255564 (дата звернення: 12.09.2025).

17. IEEE Computer Society. Modern Web Application Architectures. URL: https://ieeexplore.ieee.org/ (дата звернення: 12.09.2025).

18. Akamai. State of Online Retail Performance. URL: https://www.akamai.com/resources/state-of-online-retail-performance (дата звернення: 12.09.2025).

19. Amazon Web Services. Web Application Architecture Best Practices. URL: https://aws.amazon.com/architecture/ (дата звернення: 08.11.2025).

20. Cloudflare. Performance and Security for Modern Web Applications. URL: https://www.cloudflare.com/learning/ (дата звернення: 08.11.2025).

21. Stack Overflow. Developer Survey: Web Technologies. URL: https://survey.stackoverflow.co/ (дата звернення: 08.11.2025).

22. Statista. Progressive Web App Market Statistics. URL:

https://www.statista.com/statistics/1237080/progressive-web-app-market-size/ (дата звернення: 08.11.2025).

23. Forrester Research. The State of Mobile and Web Applications. URL: https://www.forrester.com/ (дата звернення: 05.12.2025).

24. Smashing Magazine. Progressive Web Apps in E-commerce. URL: https://www.smashingmagazine.com/2018/02/progressive-web-apps-ecommerce/ (дата звернення: 05.12.2025).

25. Medium Engineering. Progressive Web Applications Architecture. URL: https://medium.com/engineering (дата звернення: 05.12.2025).

26. European Telecommunications Standards Institute (ETSI). Web Application Security Standards. URL: https://www.etsi.org/technologies/web-security (дата звернення: 15.04.2025).

Декларація

академічної доброчесності

здобувача ступеня вищої освіти ЗНУ

Я, Вєтров Сергій Олександрович, студент 4 курсу, форми навчання денної, Інженерного навчально-наукового інституту, спеціальність 121 Інженерія програмного забезпечення, адреса електронної пошти ipz20bd-203@stu.zsea.edu.ua, — підтверджую, що написана мною кваліфікаційна робота на тему «**Інтеграція Progressive Web Application (PWA) з системою електронної комерції та порівняльний аналіз з традиційними системами**» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст.42 Закону України «Про освіту», зі змістом яких ознайомлений.

- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

згоден/згодна на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-системи, а також на архівування моєї роботи в базі даних цієї системи.

Дата 01.12.2025 Підпис _____ Вєтров Сергій Олександрович
(студент)

Дата 01.12.2025 Підпис _____ Заяц Валерій Іванович
(науковий керівник)