

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки**

«На правах рукопису»
УДК 004.4

До захисту допущено:
В.о. завідувача кафедри
_____ Михайло Новотарський
«___» _____ 2025 р.

**Магістерська дисертація
на здобуття ступеня магістра
за освітньо-науковою програмою «Інженерія програмного забезпечення
комп'ютерних та інформаційних систем» зі спеціальності 121 “Інженерія
програмного забезпечення” на тему:
«Мобільний застосунок з використанням технології PWA»**

Виконав:
студент II курсу, групи ІМ-42мп
Комісар Кіріл Сергійович

Керівник:
доц, к.т.н., доц.
Селіванов Віктор Левович

Консультант з нормоконтролю:
проф. д.т.н. проф.
Жабін Валерій Іванович
Рецензент:

Засвідчую, що у цій магістерській
дисертації немає запозичень з праць
інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2025 року

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ
УКРАЇНИ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМ. ІГОРЯ СІКОРСЬКОГО»**

**Факультет інформатики та обчислювальної техніки Кафедра обчислювальної
техніки**

Рівень вищої освіти – другий (магістерський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – «Інженерія програмного забезпечення
комп'ютерних систем»

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри

_____ Михайло НОВОТАРСЬКИЙ

«___» _____ 2025 р.

ЗАВДАННЯ

на магістерську дисертацію студенту

Комісару Кірілу Сергійовичу

1. Тема дисертації «Мобільний застосунок з використанням технології

PWA», керівник дисертації доц., к.т.н., доц. Волокита Артем Миколайович,

затверджені наказом по університету від « 06 » листопада 2025р. №

4841-с

2. Термін подання студентом дисертації 26.10.2025

3. Об'єкт дослідження процес автономної роботи вебзастосунків у середовищі
обмеженого або нестабільного інтернет-з'єднання.

4. Предмет дослідження: методи та інструменти реалізації PWA-застосунку з
використанням Service Worker, Cache Storage та IndexedDB.

5. Перелік завдань, які потрібно розробити: дослідження предметної області, огляд існуючих рішень, визначення вимог і завдань для програмного забезпечення, реалізація продукту, використання обраних інструментів для розробки.

6. Консультанти розділів дисертації:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1.	Селіванов В.Л., к.т.н.		
2.	Селіванов В.Л., к.т.н.		
3.	Волокита А.М., к.т.н.		
4.	Волокита А.М., к.т.н.		

7. Дата видачі завдання ____2 вересня 2025 року_____

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітки
1.	<i>Затвердження теми роботи</i>	01.05.2025 - 11.05.2025	
2.	<i>Вивчення та аналіз завдання</i>	12.05.2025 - 07.08.2025	
3.	<i>Розробка архітектури системи</i>	08.08.2025 - 27.08.2025	
4.	<i>Розробка підсистем</i>	28.08.2025 - 15.09.2025	
5.	<i>Програмна реалізація системи</i>	15.09.2025 - 27.10.2025	
6.	<i>Оформлення пояснювальної записки</i>	28.10.2025 - 29.11.2025	
8.	<i>Захист</i>	22.12.2025	

Студент

Кіріл КОМІСАР

(підпис)

Керівник

Артем ВОЛОКИТА

(підпис)

РЕФЕРАТ

на магістерську дисертацію

виконану на тему: Мобільний застосунок з використанням технології PWA

студентом: Комісар Кіріл Сергійович

Актуальність теми. У сучасних умовах користувачі очікують миттєвого та безперервного доступу до цифрових сервісів незалежно від якості інтернет-з'єднання. Традиційні вебзастосунки повністю залежать від мережі, тоді як нативні мобільні застосунки потребують інсталяції та прив'язані до конкретної операційної системи. Технологія Progressive Web App (PWA) дозволяє поєднати переваги обох підходів та забезпечити повноцінну автономну роботу, миттєвий запуск і нативноподібний досвід користування, що визначає високу актуальність дослідження.

Мета дослідження. Розробка прогресивного вебзастосунку з підтримкою offline-first архітектури, локального збереження даних та можливістю встановлення на мобільний пристрій, який забезпечує стабільний доступ до функціоналу незалежно від наявності або якості інтернет-з'єднання.

Об'єкт дослідження: процес автономної роботи вебзастосунків у середовищі обмеженого або нестабільного інтернет-з'єднання.

Предмет дослідження: методи та інструменти реалізації PWA-застосунку з використанням Service Worker, Cache Storage та IndexedDB[2].

Для досягнення поставленої мети були **сформульовані наступні завдання:**

- виконати аналіз сучасних архітектур вебзастосунків та їх обмежень;
- обґрунтувати вибір технології Progressive Web App;
- побудувати архітектуру клієнтського застосунку з підтримкою офлайн-роботи;
- реалізувати механізми кешування та локального збереження даних;
- забезпечити можливість встановлення застосунку як нативного;

– провести експериментальне тестування продуктивності та UX-якості системи.

Результатом дослідження є створений прогресивний вебзастосунок, який здатний працювати повністю автономно без мережевого з'єднання, миттєво запускатися з локального кешу та встановлюватися на мобільний пристрій без використання App Store чи Play Market.

Практичне значення полягає у можливості використання розробленого рішення для створення легких, автономних мобільних інформаційних систем, які не потребують постійного інтернету та підходять для використання в умовах нестабільного або обмеженого мережевого доступу.

Зв'язок з науковими програмами, планами, темами. Робота виконувалась на кафедрі обчислювальної техніки університету України «Київський політехнічний інститут імені Ігоря Сікорського».

Ключові слова: Progressive Web App, offline-first, Service Worker, локальне збереження даних, вебзастосунок, автономність.

ABSTRACT

Relevance of the topic. In modern conditions, users expect instant and uninterrupted access to digital services regardless of the quality of the internet connection. Traditional web applications fully depend on the network, while native mobile applications require installation and are tied to a specific operating system. The Progressive Web App (PWA) technology makes it possible to combine the advantages of both approaches and provide fully autonomous operation, instant startup, and a native-like user experience, which highlights the high relevance of this research.

Research objective. The development of a progressive web application with offline-first architecture support, local data storage, and the ability to be installed on a mobile device, ensuring stable access to functionality regardless of the availability or quality of the internet connection.

Object of research: the process of autonomous operation of web applications in conditions of limited or unstable internet connectivity.

Subject of research: methods and tools for implementing a PWA application using Service Worker, Cache Storage, and IndexedDB.

To achieve the set goal, the following tasks were formulated:

- to analyze modern web application architectures and their limitations;
- to justify the choice of Progressive Web App technology;
- to design the architecture of a client-side application with offline support;
- to implement mechanisms for caching and local data storage;
- to ensure the ability to install the application as a native app;
- to conduct experimental testing of the system's performance and UX quality.

The result of the research is a progressive web application capable of fully autonomous operation without a network connection, instant launching from

local cache, and installation on a mobile device without the use of App Store or Play Market.

The practical significance lies in the possibility of using the developed solution for creating lightweight autonomous mobile information systems that do not require constant internet access and are suitable for operation in conditions of unstable or limited network connectivity.

Relation to scientific programs, plans, and topics. The work was carried out at the Department of Computer Engineering of the National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”.

Keywords: Progressive Web App, offline-first, Service Worker, local data storage, web application, autonomy.

Зміст

РОЗДІЛ 1.....	5
СУЧАСНІ ТЕНДЕНЦІЇ ТА ЕВОЛЮЦІЯ ВЕБТЕХНОЛОГІЙ.....	5
1.1. Сучасні тенденції розвитку цифрових технологій.....	5
1.1.1 Еволюція вебзастосунків та потреба у новій архітектурі цифрових сервісів	8
1.1.2 Прогресивні вебзастосунки як відповідь на вимоги нового покоління цифрових сервісів	11
1.1.3 Переваги та обмеження прогресивних вебзастосунків у контексті сучасної цифрової екосистеми.....	15
1.1.4 Архітектурні моделі сучасних вебзастосунків та їх еволюція	20
1.1.5 Технологічні стандарти та браузерні API, що стали основою PWA	22
1.1.6. Модель App Shell у контексті архітектури offline-first	24
РОЗДІЛ 2	28
ПОСТАНОВКА ЗАДАЧІ ТА ОБҐРУНТУВАННЯ ВИБОРУ ТЕХНОЛОГІЇ	28
2.1. Формулювання проблеми.....	28
2.1.1 Об'єкт і предмет дослідження	28
2.1.2 Мета та завдання дослідження	29
2.1.3 Функціональні та нефункціональні вимоги	30
2.1.4 Обґрунтування вибору технології реалізації.....	32
2.2 Архітектура прогресивного вебзастосунку	32
2.2.1 Інструменти	36
2.2.2 Фреймворки	36
2.2.3 Концепція архітектури та логіка offline-first	37
2.2.4 Модель даних та логіка клієнтського зберігання	39
2.2.5 Логіка роботи Service Worker та стратегія кешування.....	40
2.2.6 Забезпечення безпеки та доступності застосунку.....	41
Висновки до розділу 2	43
РОЗДІЛ 3 РЕАЛІЗАЦІЯ ПРОГРЕСИВНОГО ВЕБЗАСТОСУНКУ	46

3.1. Вибір технологічного стеку та логіка архітектурного рішення	46
3.1.1 Структура проєкту та призначення ключових компонентів.....	46
3.1.2. Деталізація технічного стеку та вибір підходу App Shell	48
3.2 Реєстрація Service Worker та механізм встановлення застосунку.....	50
3.2.1 Реалізація механізму офлайн-підтримки та кешування ресурсів.	50
3.2.2 Локальне збереження даних користувача: вибір IndexedDB та організація доступу	53
3.2.3. Логічна схема зберігання	54
3.2.4. Життєвий цикл бази та міграції схеми.....	54
3.2.5. Транзакційна модель доступу	54
3.2.6. Обробка помилок та відновлення.....	55
3.2.7. Взаємодія IndexedDB з офлайн-логікою.....	55
3.2.8. Аргументація вибору та наслідки для якості	55
3.2.9. Взаємодія Service Worker з рівнем інтерфейсу	56
3.3. Організація інтерфейсу та користувацького досвіду (UX/UI-стратегія)	57
3.3.1. Адаптивність і візуальна ієрархія.....	57
3.3.2. Керованість клавіатурою та доступність	57
3.3.3. Мікроанімації та реакція інтерфейсу	57
3.3.4. Зниження когнітивного навантаження	58
3.3.5 Оптимізація продуктивності та контроль часу відгуку.....	58
3.3.6. Пріоритет швидкості першого запуску (First Load)	58
3.3.7. Прискорення повторного запуску (Repeat Load)	58
3.3.8. Гарантована стабільність візуального інтерфейсу (CLS)	59
3.3.9. Результат оптимізації.....	59
3.3.10. Web App Manifest, іконки та користувацький досвід встановлення.....	60
3.4. Інтеграція компонентів та послідовність роботи застосунку	62
3.5. Алгоритмічна модель роботи Service Worker та стратегії кешування	63

3.5.1. Загальний алгоритм Service Worker	63
3.5.2. Стратегії кешування та їх застосування в проєкті.....	64
Висновки до розділу 3	65
РОЗДІЛ 4	67
ЕКСПЕРИМЕНТАЛЬНА ОЦІНКА РОБОТИ ЗАСТОСУНКУ	67
4.1. Методологія експериментальної перевірки.....	67
4.1.1 Сценарії тестування та умови проведення експерименту.....	68
4.1.2. Оцінка продуктивності (Lighthouse Audit)[9].....	69
4.1.3. Оцінка якості взаємодії (Web Vitals).....	69
4.1.4. Реальні UX-показники	70
4.1.5. Додаткові критерії оцінки стабільності PWA-застосунку	70
4.1.6. Порівняльне тестування з нативними додатками	71
4.1.7. Дослідження поведінки застосунку при втраті мережі	72
4.1.8. Додатковий аудит безпеки PWA	73
4.1.9. Аналіз продуктивності Service Worker	73
4.1.10. Мікроперформанс та UX-аналіз	74
4.2. Аналітичний висновок за результатами тестування.....	74
Висновки до розділу 4	76
РОЗДІЛ 5	78
РОЗРОБКА СТАРТАП-ПРОЄКТУ	78
5.1. Опис ідеї проєкту	78
5.2. Технологічний аудит проєкту	80
5.3. Аналіз ринкових можливостей запуску стартап-проєкту	82
5.3.1. Попередня характеристика ринку	82
5.3.2. Характеристика потенційних клієнтів	82
5.3.3. Фактори загроз	84
5.3.4. Фактори можливостей	84
5.3.5. Ступеневий аналіз конкуренції.....	85
5.3.6. Аналіз конкуренції за М. Портером	86

5.4. Обґрунтування конкурентоспроможності та стратегія розвитку	87
5.4.1. Обґрунтування факторів конкурентоспроможності	87
5.4.2. SWOT-аналіз стартап-проєкту	87
5.5. Розроблення ринкової стратегії проєкту	88
5.5.1. Вибір цільових груп потенційних споживачів	88
5.5.2. Визначення базової стратегії розвитку	89
5.5.3. Стратегія конкурентної поведінки	90
5.6. Розроблення маркетингової програми стартап-проєкту	90
5.6.1. Визначення ключових переваг концепції продукту	90
5.6.2. Опис трьох рівнів моделі товару	91
5.6.3. Формування системи збуту	91
5.6.4. Концепція маркетингових комунікацій	92
Висновки до розділу 5	93
ВИСНОВКИ	95
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	99
Додаток А	101

РОЗДІЛ 1.

СУЧАСНІ ТЕНДЕНЦІЇ ТА ЕВОЛЮЦІЯ ВЕБТЕХНОЛОГІЙ

1.1. Сучасні тенденції розвитку цифрових технологій

Упродовж останнього десятиліття цифрові технології зазнали стрімкої еволюції, і ключовою її особливістю стало домінування мобільних пристроїв як основного засобу доступу до інформації. Якщо раніше більшість користувачів здійснювали взаємодію з вебсервісами виключно через персональні комп'ютери, то сьогодні споживання контенту дедалі більше зміщується у бік смартфонів та планшетів. Це зумовлюється не лише зручністю портативності, але й тим, що мобільний доступ забезпечує максимально швидку реакцію на потребу користувача — незалежно від його місцезнаходження, часу доби чи наявності стаціонарного інтернет-підключення.

Таке зростання мобільної активності створило абсолютно новий набір вимог до вебзастосунків. Користувачі більше не розглядають інтернет як щось, до чого потрібно «підключатися»; натомість вони сприймають цифрові сервіси як постійно доступне середовище, яке має працювати миттєво та без жодних технічних перешкод. У результаті очікується, що будь-який сучасний застосунок буде реагувати швидко, працювати навіть за нестабільного мережевого з'єднання, адаптуватися під різні типи пристроїв і забезпечувати інтерфейс, максимально наближений до звичного мобільного досвіду.

Класична модель вебзастосунків, яка передбачала повну залежність від наявності стабільного інтернет-з'єднання, поступово перестала відповідати цим очікуванням. Якщо раніше короткочасне завантаження сторінки сприймалося користувачем як природна частина роботи вебу, то

нині навіть кількасекундна затримка може стати причиною втрати уваги, інтересу або довіри до сервісу. У сучасній цифровій екосистемі час відгуку перестає бути просто технічною характеристикою — він перетворюється на критично важливий компонент користувацького досвіду, який безпосередньо впливає на залучення та утримання аудиторії[2].

Це означає, що на практиці вебзастосунк повинен забезпечувати безперервність взаємодії, навіть якщо інтернет зник або сповільнився. Користувач очікує, що може переглядати вміст, редагувати дані або планувати події у будь-який момент — незалежно від доступності мережі. Саме ця зміна очікувань стала однією з ключових причин переходу від традиційного вебу до моделей, які працюють за принципом offline-first — тобто розглядають мережу лише як одну з опцій, а не як обов'язкову умову функціонування[10].

Крім того, від сучасних застосунків очікується плавність та природність взаємодії, подібна до тієї, яку забезпечують нативні мобільні програми. Користувачі не хочуть щоразу відкривати браузер, вводити адресу сайту або чекати перезавантаження сторінки — натомість вони прагнуть мати миттєвий доступ у один дотик, у зручному повноекранному середовищі, без зайвих елементів інтерфейсу браузера. Таким чином, користувацька взаємодія має бути не просто доступною, а інтуїтивною, безшовною та автономною, наближеною до стандартів мобільних операційних систем.

Усе це призводить до того, що функціональність вебзастосунків більше не може обмежуватися лише поточним з'єднанням із сервером. Сучасні цифрові системи повинні бути здатними самотійно обробляти дані на стороні клієнта, зберігати нову інформацію локально та виконувати щонайменше базові операції без доступу до мережі. Саме такий підхід — побудова вебрішень із орієнтацією на повну автономність — стає

визначальним для нової хвилі прогресивних технологій, на основі яких сьогодні створюються вебзастосунки нового покоління[4].

Таким чином, трансформація вебсередовища від статичних сторінок до інтерактивних автономних застосунків є логічним етапом еволюції цифрових технологій. Якщо початкові вебресурси виконували переважно інформаційну функцію, то сучасні вебплатформи дедалі частіше виступають як повноцінні інструменти взаємодії, планування, навчання, організації особистих процесів чи навіть бізнес-операцій. Тобто від вебтехнологій очікують не тільки доступу до даних, але й інтелектуальної поведінки — локальної обробки інформації, персоналізації, швидкої реакції на дії користувача.

У підсумку формується абсолютно нова концепція цифрової взаємодії, у межах якої веб не повинен поступатися нативним мобільним застосункам ні швидкодією, ні надійністю, ні зручністю використання. Саме в цьому контексті зростає необхідність застосування архітектур, здатних поєднати універсальність вебтехнологій із безпосередньою доступністю, автономністю та “приложенеподібним” досвідом — тобто тим, що ще недавно асоціювалося виключно з мобільними операційними системами.

Враховуючи зазначене, можна стверджувати, що ключовим викликом сучасного вебсередовища є поєднання миттєвої доступності та автономної працездатності, що забезпечує незалежність від зовнішніх умов при збереженні єдиної кодової бази та універсальності розгортання. Іншими словами, цифровий сервіс повинен залишатися завжди доступним, швидким, передбачуваним, і при цьому не вимагати складної інсталяції або оновлення з боку користувача, що було характерним для попередніх поколінь програмного забезпечення.

Це призвело до формування нового класу вебтехнологій, орієнтованих на повну самодостатність застосунку в клієнтському

середовищі, де вебпереглядач перестає бути лише інструментом відображення, а перетворюється на повноцінну платформу виконання. Такий підхід відкриває можливість створювати вебрішення, що зовні не відрізняються від нативних мобільних застосунків, але мають при цьому універсальність поширення та простоту доступу, притаманну вебекосистемі. Саме ця концепція стала основою технологій нового покоління прогресивних вебзастосунків, які детально розглядаються в наступному підпункті.

1.1.1. Еволюція вебзастосунків та потреба у новій архітектурі цифрових сервісів

Історія розвитку вебзастосунків демонструє послідовний перехід від пасивного споживання інформації до активної взаємодії з нею в реальному часі. Якщо початкові вебсторінки виконували виключно інформаційну функцію та оновлювалися лише шляхом повного перезавантаження, то з появою технологій асинхронного обміну даними (як-от AJAX) користувач отримав можливість взаємодіяти із системою без повного оновлення сторінки, що стало першим кроком до підвищення динамічності вебінтерфейсів.

Подальше зростання вимог до інтерактивності призвело до формування SPA-парадигми (Single Page Application), яка передбачає роботу в єдиному HTML-контейнері з максимальною локальною обробкою даних на стороні клієнта. Проте, попри значне підвищення швидкодії, такі застосунки все ще залишались повністю залежними від стабільного інтернет-з'єднання, оскільки більшість бізнес-логіки реалізовувалась на стороні сервера, а клієнт лише ініціював запити[14].

Однак із часом стало очевидним, що навіть найшвидший мережевий запит не може гарантувати безперервність взаємодії, якщо відсутній інтернет або виникають навіть короточасні перебої зв'язку. У багатьох життєвих ситуаціях — під час поїздок, у транспорті, у зонах слабого

покриття або при використанні нестабільного мобільного інтернету — користувач очікує доступу до ключової функціональності без необхідності очікувати підключення до мережі. Це формує запит на нову архітектуру, у центрі якої не сервер, а клієнт — із власними механізмами автономного зберігання та обробки даних.

Саме в цей момент виникає концепція *offline-first*, яка передбачає, що застосунок має бути самодостатнім навіть без мережі, а інтернет розглядається не як критична залежність, а як додатковий інструмент синхронізації, якщо він доступний. Такий підхід повністю змінює логіку проєктування сучасних цифрових сервісів — клієнт перестає бути пасивним інтерфейсом і перетворюється на автономний обчислювальний агент із власним сховищем і логікою виконання.

У результаті формуються нові вимоги до архітектури вебзастосунків, які суттєво відрізняються від традиційних моделей клієнт–сервер. Сучасне цифрове рішення повинно не лише передавати дані між клієнтом та сервером, а й забезпечувати:

- повну працездатність у режимі тимчасової або повної відсутності мережі;
- миттєвий запуск без очікування відповіді від сервера;
- автономне зберігання критичних даних на стороні клієнта;
- адаптивність інтерфейсу до будь-якого типу пристрою та сценарію використання;
- максимальне наближення до досвіду роботи з нативними мобільними застосунками.

Таким чином, вебпереглядач перестає розглядатися лише як засіб відображення сторінок, а натомість стає повноцінною обчислювальною платформою, здатною виконувати роль локального середовища виконання застосунку з автономною логікою.

Ця трансформація створила підґрунтя для появи концепцій нового покоління вебтехнологій, орієнтованих на універсальність, автономність, швидкодію та нативність користувацького досвіду.

Таблиця 1.1.

Еволюція архітектур вебзастосунків

Покоління технологій	Характерні особливості	Обмеження
Статичні вебсторінки (Web 1.0)	Простота, мінімум логіки, лише HTML	Відсутність інтерактивності, повна залежність від сервера
Клієнт–сервер + AJAX (Web 2.0)	Часткове оновлення контенту, динамічність	Часткове кешування, залежність від мережі
Single Page Application (SPA)	Повна клієнтська логіка, швидкість, React/Angular	Відсутність офлайн-режиму, складність оптимізації, велика залежність від API
SSR/SSG/ISR	Швидший рендеринг, SEO-оптимізація	Все ще потребує серверної логіки та підключення до мережі
PWA + offline-first (Web 3.0)	Автономність, кеш, моментальний запуск	Обмежений доступ до апаратних можливостей у порівнянні з нативними

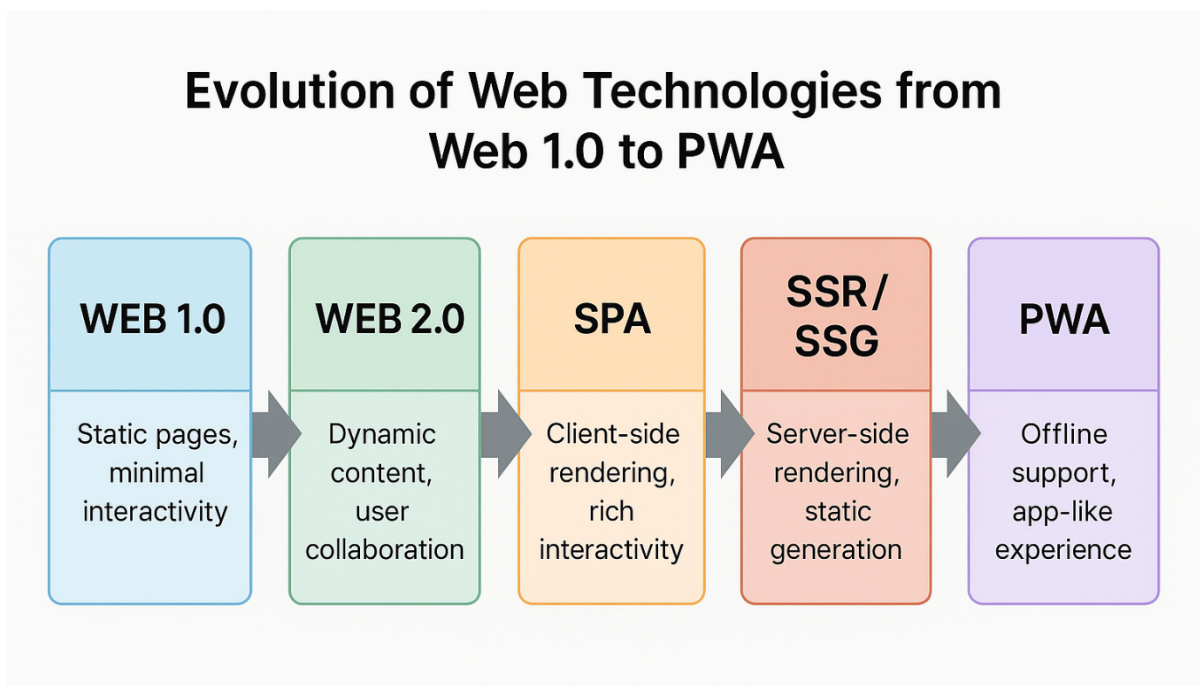


Рисунок 1.1. Еволюція вебтехнологій від Web 1.0 до PWA

Рисунок 1.1 демонструє еволюцію вебтехнологій від статичних сторінок (Web 1.0) через динамічний Web 2.0, односторінкові застосунки (SPA) та серверний рендеринг (SSR/SSG) до прогресивних вебзастосунків (PWA). Наведена схема відображає закономірний перехід від інформаційного вебу до автономних, високопродуктивних і застосункоорієнтованих рішень.

1.1.2. Прогресивні вебзастосунки як відповідь на вимоги нового покоління цифрових сервісів

У відповідь на зростання вимог до автономності, швидкодії та універсальності вебрішень було сформовано концепцію прогресивних вебзастосунків — технологічного підходу нового покоління, що поєднує доступність вебу з досвідом використання, властивим нативним мобільним застосункам.

Прогресивні вебзастосунки [1](Progressive Web Apps, PWA) розробляються за принципом, відповідно до якого вебпереглядач використовується не лише як інструмент перегляду сторінок, а як

повноцінна платформа виконання застосунків, здатна забезпечувати локальне збереження даних, офлайн-працездатність, швидкий запуск і встановлення на пристрій.

На відміну від традиційних вебсервісів, які залежали від кожного мережевого запиту, прогресивні вебзастосунки передбачають інтелектуальне керування мережею — інтернет використовується лише тоді, коли це справді необхідно, а всі критично важливі елементи заздалегідь кешуються та зберігаються на пристрої користувача. Це забезпечує миттєвий запуск застосунку, навіть якщо фактичне під'єднання до мережі тимчасово недоступне або нестабільне.

Прогресивність PWA полягає в принципі “покращення в залежності від можливостей середовища”, що означає:

Застосунок працює коректно навіть у мінімальних умовах, але автоматично розширює свою функціональність, якщо браузер і пристрій підтримують додаткові можливості.

Такий підхід гарантує широку сумісність — застосунок не обмежується окремою платформою чи операційною системою.

Ключовими характеристиками прогресивних вебзастосунків, що принципово відрізняють їх від традиційних вебрішень, є:

- Автономність (offline-first) — здатність працювати без постійного інтернету, використовуючи попереднє кешування даних і локальне сховище[10].
- Миттєвий запуск — застосунок відкривається локально, без очікування відповіді від сервера, що створює враження нативної програми.
- Установлюваність — можливість додати застосунок на головний екран пристрою та запускати його без браузерного інтерфейсу.
- Продуктивність і плавність — мінімальний час відгуку, збереження стабільності інтерфейсу, адаптивність до різних екранів.

- Безпечність — робота виключно через захищений протокол, що унеможливорює використання незахищених каналів.

Таким чином, прогресивні вебзастосунки можна розглядати як гібридну модель нового покоління, яка поєднує легкість доступу та оновлення, притаманну вебрішенням, із швидкодією, автономністю та зручністю використання, характерною для мобільних застосунків.

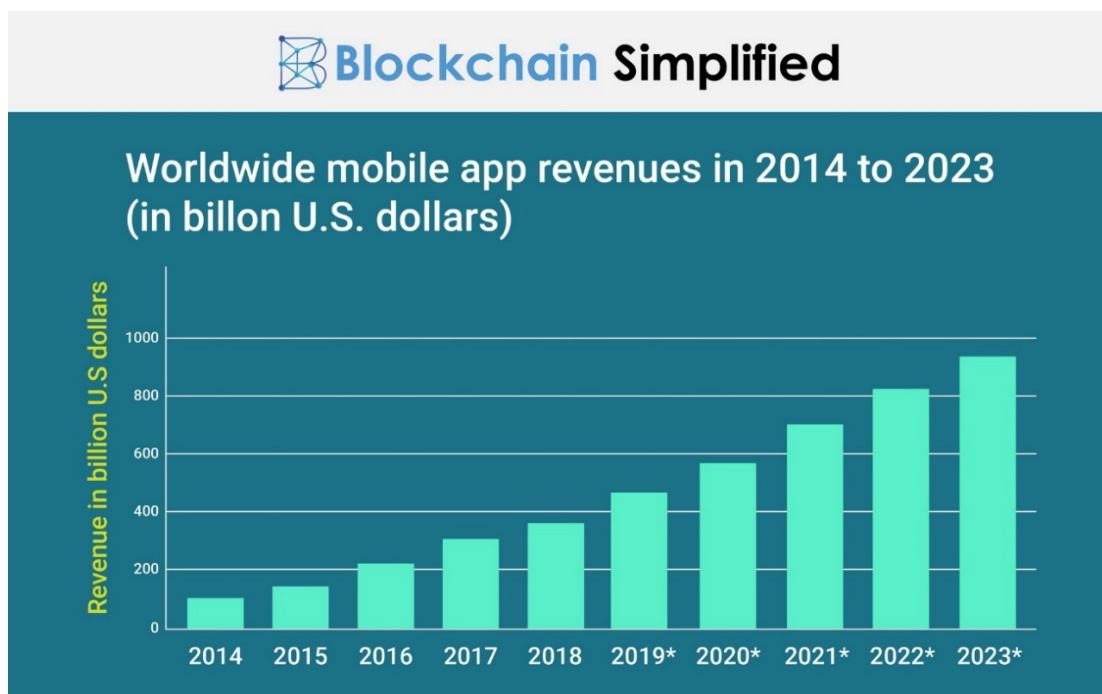


Рис 1.2. Діаграма, що демонструють зростання популярності PWA

На відміну від традиційного підходу, орієнтованого на постійну залежність від сервера, PWA забезпечують інтелектуальне використання мережевих ресурсів, працюючи самостійно в режимах обмеженої або відсутньої мережі, що відповідає сучасній моделі «застосунок має бути доступним завжди»[1].

Унаслідок цього концепція PWA набуває особливої актуальності в умовах цифрової мобільності та високих очікувань користувачів щодо безперервності доступу до даних, а також у контексті оптимізації витрат на розробку, оскільки одне вебрішення може охоплювати всі типи пристроїв без створення окремих платформозалежних застосунків.

Таблиця 1.2.

Порівняння PWA з іншими технологіями

Технологія	Переваги	Недоліки
Традиційні веб-сайти	Легко створювати, не потребують завантаження, доступні на будь-якому пристрої	Обмежені можливості, не працюють офлайн, повільніше завантаження, не можуть надсилати push-сповіщення
Нативні мобільні додатки	Швидкі, плавні, доступ до функцій пристрою, push-сповіщення	Складніші у розробці, потребують завантаження з магазину, не доступні на всіх платформах
Мобільні сайти, оптимізовані для мобільних пристроїв	Легко створювати, кращий користувацький досвід на мобільних пристроях	Обмежені можливості порівняно з PWA, не працюють офлайн, повільніше завантаження, не можуть надсилати push-сповіщення
Гібридні мобільні додатки	Поєднують можливості нативних та веб-додатків, доступ до функцій пристрою, push-сповіщення	Складніші у розробці, потребують знань як веб-розробки, так і мобільної розробки, можуть мати проблеми з продуктивністю
PWA (Progressive Web Apps)	Кращий користувацький досвід, швидша розробка, доступність, гнучкість	Залежність від браузера, обмежена підтримка функцій пристрою, сприйняття користувачів

Традиційні вебзастосунки		PWA
Offline	не підтримується	підтримується
Швидкість	середня	висока
Встановлення	немає	є (додавання на головний екран)
UX	обмежений	близький до нативного
Підтримка пристроїв	браузерна	широка (встановлюваність)

Рис 1.3. Порівняння традиційних вебзастосунків і PWA

Рисунок 1.3 наочно ілюструє ключові відмінності між традиційними вебзастосунками та прогресивними вебзастосунками (PWA). Порівняння показує, що PWA забезпечують офлайн-працездатність, вищу швидкодію, можливість встановлення на пристрій та користувацький досвід, наближений до нативного, тоді як традиційні вебзастосунки мають суттєві функціональні обмеження.

1.1.3. Переваги та обмеження прогресивних вебзастосунків у контексті сучасної цифрової екосистеми

Попри універсальність та адаптивність, прогресивні вебзастосунки не є компромісним або проміжним рішенням — навпаки, вони розглядаються як повноцінна архітектурна парадигма нового покоління, що відповідає вимогам сучасної цифрової мобільності.

Їхня ключова сила полягає у здатності поєднувати найкращі властивості вебу та мобільних платформ, забезпечуючи доступність без інсталяції та автономність без прив'язки до мережі[2].

До основних переваг прогресивних вебзастосунків належать:

- Автономність і гарантована працездатність у будь-яких умовах → завдяки роботі в режимі offline-first застосунок залишається доступним навіть за повної відсутності інтернету.
- Миттєвий запуск і швидке повторне завантаження → ключові ресурси зберігаються локально, тому застосунок відкривається без звернення до сервера.
- Відсутність потреби в інсталяції з маркетів → користувач може додати застосунок на головний екран без складних процедур встановлення.
- Єдина кодова база для всіх платформ → немає потреби створювати окремі версії для різних операційних систем.
- Простота оновлення → застосунок оновлюється автоматично у браузері, без участі користувача.
- Мінімальне споживання ресурсів пристрою, що особливо важливо для мобільних сценаріїв використання.

Водночас, попри значні переваги, прогресивні вебзастосунки мають і певні обмеження, зумовлені природою вебплатформи:

- Залежність від можливостей браузера → рівень підтримки може частково відрізнятись між різними операційними системами та версіями браузерів.
- Обмежений доступ до апаратних ресурсів пристрою → у порівнянні з нативними застосунками, доступ до функцій на кшталт Bluetooth, NFC або розширених сенсорів може бути обмеженим або вимагати додаткових дозволів[3].
- Неможливість повного контролю над системним середовищем → PWA функціонує в межах браузерного “sandbox”, що забезпечує безпеку, але обмежує глибину інтеграції.

- Нерівномірна підтримка усіх можливостей PWA у різних операційних середовищах, що вимагає адаптивного або fallback-проєктування[1].

Однак важливо підкреслити, що жодне з вказаних обмежень не нівелює доцільність використання PWA. Навпаки — у більшості сценаріїв цифрової взаємодії користувачів, особливо в контексті оперативного доступу до інформації, планування та виконання повсякденних задач, прогресивні вебзастосунки забезпечують оптимальний баланс між функціональністю, доступністю, продуктивністю та економічністю розробки[16].

Приклади успішних PWA

Progressive Web Apps (PWA) вже успішно впроваджено в багатьох галузях, демонструючи їх універсальність та ефективність.

Опис:

Starbucks - це американська мережа кав'ярень, яка володіє та ліцензує кав'ярні по всьому світу. Компанія відома своєю кавою, еспreso-напоями та іншими напоями, а також їжею та товарами. Starbucks створила PWA-додаток, який дозволяє користувачам замовляти каву та їжу, переглядати меню та знаходити найближчі кав'ярні. (Рис 1.4.)

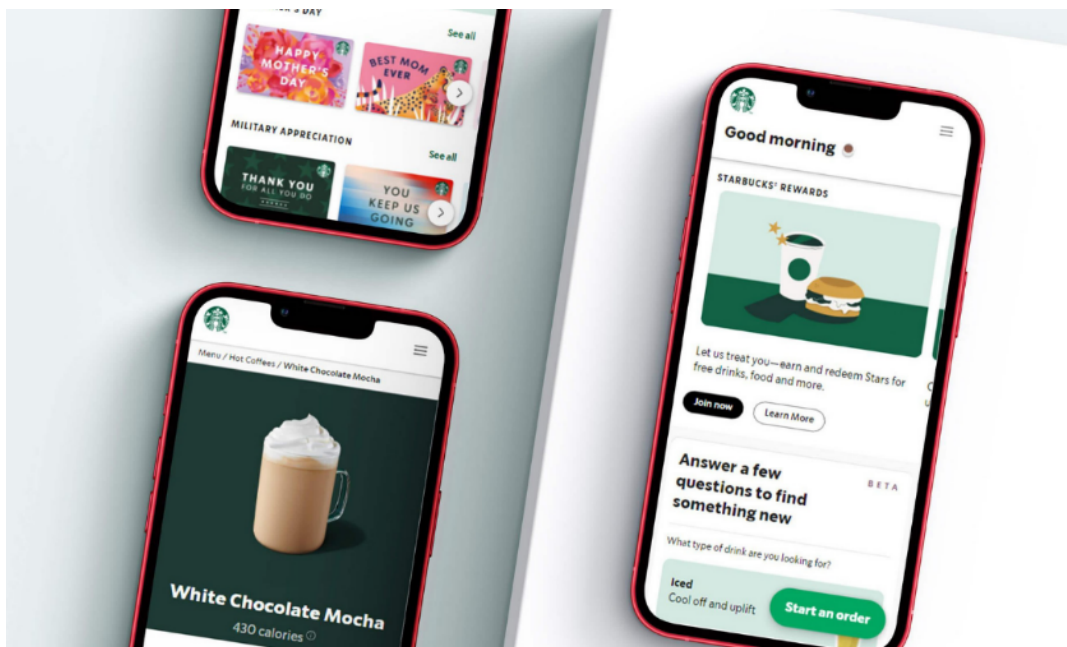


Рис 1.4. Зображення Starbucks PWA

Переваги:

- Швидке та зручне замовлення: PWA-додаток Starbucks забезпечує швидкий та зручний спосіб замовлення кави та їжі без черги. Користувачі можуть переглядати меню, персоналізувати свої замовлення та оплачувати замовлення всього за кілька натискань.
- Офлайн-доступ: PWA-додаток Starbucks пропонує офлайн-доступ до меню, фаворитів та історії замовлень. Це дозволяє користувачам переглядати меню та створювати замовлення навіть коли вони не мають підключення до Інтернету. Замовлення буде збережено в додатку та оновлено при наступному підключенні до Інтернету.
- Персоналізація: PWA-додаток Starbucks пропонує персоналізовані рекомендації на основі історії замовлень користувачів. Це дозволяє користувачам швидко знаходити свої улюблені напої та їжу.
- Push-повідомлення: PWA-додаток Starbucks може надсилати користувачам push-повідомлення про готовність замовлення, про спеціальні пропозиції та інші події.

- Зручний пошук магазинів: PWA-додаток Starbucks дозволяє користувачам легко знаходити найближчі кав'ярні на карті або за адресою. Dodatok також може надати інформацію про години роботи, зручності та завантаженість магазинів.
- Можливість оплати: PWA-додаток Starbucks підтримує різні способи оплати, включаючи кредитні картки, Apple Pay та Google Pay. Це робить оплату зручною та безпечною.
- Програма лояльності: PWA-додаток Starbucks інтегрується з програмою лояльності Starbucks Rewards. Це дозволяє користувачам заробляти зірки за свої замовлення та використовувати їх для отримання безкоштовних напоїв та їжі.

Вплив:

PWA-додаток Starbucks став одним з найпопулярніших мобільних додатків у галузі ресторанного бізнесу. Він призвів до значного збільшення мобільних замовлень та підвищення задоволеності клієнтів.

- Збільшення мобільних замовлень: З моменту запуску PWA-додаток Starbucks збільшив кількість мобільних замовлень на 30%.
- Підвищення задоволеності клієнтів: Користувачі, які використовують PWA-додаток Starbucks, задоволеніші досвідом замовлення, ніж ті, хто використовує мобільний веб-сайт або нативний додаток. Відгуки про PWA-додаток переважно позитивні, користувачі цінують його швидкість, зручність та функціональність.
- Зниження витрат: PWA-додаток Starbucks допоміг компанії знизити витрати на обслуговування клієнтів. Завдяки зручності замовлення через додаток потреба в обслуговуванні персоналом зменшилася.
- Покращена мобільна аналітика: PWA-додаток Starbucks надає компанії більш детальну інформацію про поведінку користувачів на мобільних пристроях. Це дозволяє Starbucks краще розуміти своїх клієнтів та вдосконалювати свої мобільні пропозиції.

Технічні деталі:

PWA-додаток Starbucks створено за допомогою веб-технологій, таких як HTML, CSS та JavaScript. Він також використовує Service Worker для забезпечення офлайн-функціональності та push-повідомлень.

Висновок:

PWA-додаток Starbucks це успішний приклад того, як PWA можуть використовуватися для покращення мобільного користувацького досвіду та підвищення бізнес-результатів. Додаток швидкий, зручний та багатий на функції, що зробило його популярним серед користувачів та допомогло Starbucks збільшити мобільні продажі та задоволеність клієнтів.

1.1.4. Архітектурні моделі сучасних вебзастосунків та їх еволюція

Сучасний веб пройшов кілька поколінь архітектурних трансформацій, що відображають загальний розвиток цифрових технологій. Кожне покоління мало власні переваги та недоліки, однак усі вони спрямовані на підвищення продуктивності, зменшення залежності від сервера та покращення користувацького досвіду.

Перші вебзастосунки базувалися на монолітній архітектурі, де як сервер, так і клієнт виконували мінімальну кількість логіки. Усі обчислення знаходилися на сервері, а браузер лише відображав статичні HTML-сторінки. Такий підхід був простим, але критично залежав від мережі й не міг забезпечити сучасного рівня інтерактивності.

Наступним етапом стала поява клієнт-серверної моделі з використанням AJAX, що дозволило частково виконувати обробку даних на стороні клієнта та оновлювати лише окремі частини інтерфейсу. Це підвищило швидкість роботи, проте більшість функціональності все ще залишалася серверною.

Третє покоління вебу — Single Page Application (SPA) — стало першим фундаментальним кроком до повної клієнтської логіки. Завдяки фреймворкам Angular, React, Vue інтерфейс став повністю інтерактивним, а

сервер використовувався переважно як API. Проте SPA не мали інструментів автономної роботи та залишалися критично залежними від мережі.

З поширенням мобільних технологій виникла потреба у швидшому першому завантаженні та більшій автономності, що призвело до появи Hybrid Rendering моделей — SSR, SSG та ISR. Вони дозволили передавати вже попередньо згенерований HTML, що покращило SEO та зменшило час завантаження. Однак навіть ці моделі не вирішували проблему офлайн-доступу.

У відповідь на потреби сучасної цифрової екосистеми сформувалися архітектурні підходи offline-first, засновані на використанні Service Worker, Cache Storage та IndexedDB. У таких системах клієнтський застосунок стає самодостатнім: дані, інтерфейс та логіка доступні незалежно від мережі, а сервер використовується лише як додатковий механізм синхронізації.

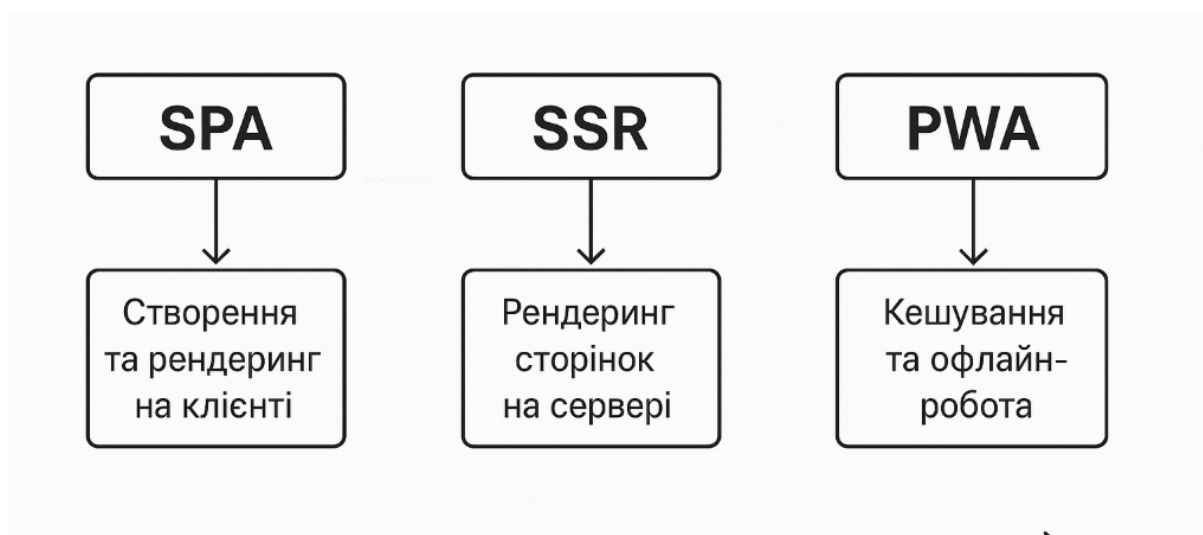


Рис 1.5. Архітектурні підходи SPA, SSR та PWA

Сьогодні архітектурні моделі на кшталт PWA, App Shell та Edge-орієнтованих рішень розглядаються як наступний етап розвитку вебу, що поєднує високу продуктивність, повну автономність, універсальність і мобільність. У результаті вебпереглядач дедалі частіше виступає як повноцінне середовище виконання застосунків нового покоління. (Рис. 1.5.)

1.1.5. Технологічні стандарти та браузерні API, що стали основою PWA

Поява прогресивних вебзастосунків стала можливою завдяки ґрунтовним змінам у вебплатформі, які відбулися протягом останніх років. Сучасні браузери отримали низку API, які раніше були доступні виключно нативним застосункам. Саме ці API формують основу для реалізації автономності, інтерактивності та високої продуктивності PWA[19].

Ключові технології, що сприяли розвитку PWA:

1. Service Worker API

Надає змогу перехоплювати мережеві запити, кешувати ресурси, обробляти події без участі сторінки та забезпечувати офлайн-працездатність.

Саме Service Worker є ядром offline-first парадигми.

2. Cache Storage API

Дозволяє створювати власні кеші, керувати версіями та забезпечувати миттєвий запуск застосунку.

3. IndexedDB API

Повноцінна клієнтська база даних, яка дозволяє зберігати складні структури та великі обсяги даних локально.

4. Web App Manifest

JSON-файл, що визначає зовнішній вигляд застосунку: назву, іконки, кольори, режим відображення. Саме він робить PWA «установлюваним».

5. Background Sync API

Дозволяє виконувати синхронізацію, навіть якщо мережа з'явилася пізніше.

6. Push API

Забезпечує надсилання push-сповіщень без відкритої вкладки браузера, що раніше було доступно лише нативним програмам.

7. Web Share та Web Share Target API

Дають можливість інтегрувати PWA у системні меню «поділитися».

Разом ці API роблять вебзастосунки повноцінною альтернативою класичним мобільним застосункам, забезпечуючи доступ до локальних ресурсів, файлової системи, апаратних компонентів та системних механізмів.

Таблиця 1.3. Основні браузерні API, що забезпечують роботу PWA

Технологія / API	Призначення	Роль у PWA
Service Worker	Перехоплення запитів, фонові події, кешування	Offline-first, автономність, управління мережею
Cache Storage	Локальне зберігання статичних ресурсів	Миттєвий запуск, стабільність
IndexedDB	Клієнтська база даних	Локальне зберігання даних, офлайн-CRUD
Web App Manifest	Метадані застосунку	Установлюваність, splash screen
Push API	Отримання push-сповіщень	Сповіщення без відкритої сторінки
Background Sync	Відкладена синхронізація	Синхронізація після появи мережі

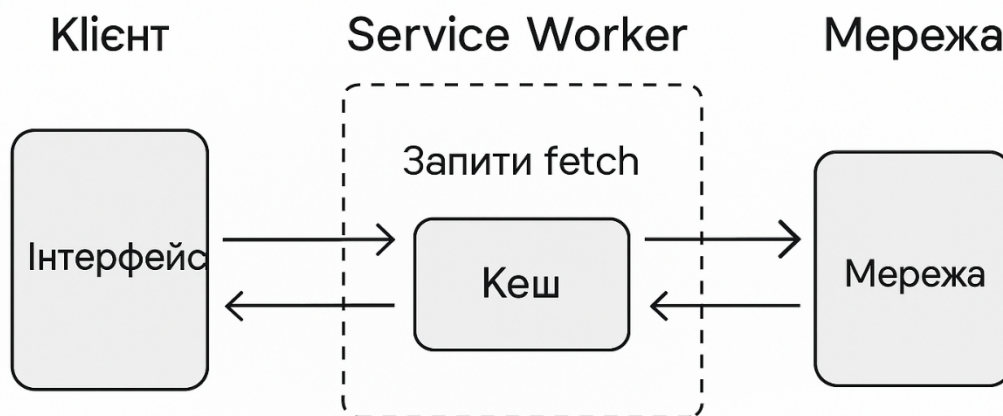


Рис 1.6. Принцип роботи Service Worker

Рисунок 1.6. ілюструє принцип роботи Service Worker як проміжної ланки між клієнтським інтерфейсом і мережею. Усі мережеві запити типу fetch перехоплюються Service Worker, який визначає, чи повертати відповідь із кешу, чи звертатися до мережі. Такий підхід забезпечує офлайн-працездатність та підвищує швидкодію застосунку.

1.1.6. Модель App Shell у контексті архітектури offline-first

Одним із ключових архітектурних підходів, що забезпечив еволюцію PWA, є модель App Shell, яка передбачає попереднє кешування елементів інтерфейсу (шаблону застосунку), що не змінюються динамічно. Це забезпечує миттєвий запуск системи, навіть коли мережа недоступна.

App Shell складається з таких частин:

- каркас інтерфейсу (header, navigation, layout);
- мінімальний CSS та JavaScript;
- основні статичні ресурси;
- порожній контейнер для динамічного контенту.

Під час запуску PWA App Shell завантажується миттєво з кешу, і лише після цього застосунок намагається отримати динамічні дані мережі. Такий підхід дозволяє:

- мінімізувати залежність від серверів;
- забезпечити стабільність UX;
- створити плавний, нативноподібний досвід;
- повністю усунути «білі екрани» під час завантаження;
- гарантувати автономність і високу продуктивність.

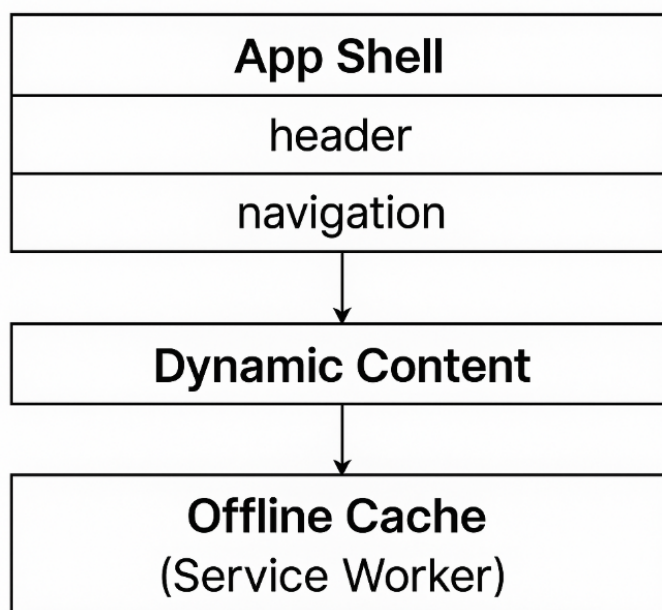


Рис 1.7. Модель App Shell

Модель App Shell стала стандартом для актуальних PWA, оскільки вона поєднує швидкість, надійність і структурну гнучкість, що є критично важливим у мобільному середовищі. (Рис 1.7.)

Висновки до розділу 1

У цьому розділі було здійснено комплексний аналіз еволюції вебтехнологій, сучасних цифрових тенденцій та архітектурних підходів, що визначили перехід від класичних вебзастосунків до прогресивних автономних рішень нового покоління. Дослідження показало, що сучасна цифрова екосистема формує принципово нові вимоги до роботи вебсистем: швидкодія, стабільність, автономність, адаптивність та здатність забезпечувати нативноподібний досвід на будь-якому пристрої стали критичними параметрами конкурентоспроможності програмного забезпечення.

Аналіз історичного розвитку вебу продемонстрував закономірний перехід від статичних сторінок до динамічних інтерактивних SPA-архітектур, а згодом — до підходів, орієнтованих на розмежування клієнтських і серверних обчислень, зокрема SSR, SSG та ISR. Проте навіть ці моделі не вирішили фундаментальної проблеми традиційного вебу — його повної залежності від стабільного інтернет-з'єднання. Водночас стрімке зростання частки мобільного трафіку, зміна користувацьких очікувань і потреба в безперервному доступі до контенту зумовили необхідність появи нової архітектурної парадигми.

У ході дослідження встановлено, що концепція offline-first та технологія Progressive Web Apps (PWA) стали відповіддю на сучасні виклики. На відміну від класичних вебрішень, PWA забезпечують автономну роботу, інтелектуальне кешування ресурсів, миттєвий запуск, плавність взаємодії та можливість установлення на мобільний пристрій без використання маркетів застосунків. Завдяки поєднанню Web App Manifest, Service Worker, Cache Storage та IndexedDB, прогресивні вебзастосунки здатні виконувати роль повноцінних мобільних систем, залишаючись при цьому універсальними та доступними через вебплатформу.

Було проаналізовано ключові переваги PWA — автономність, продуктивність, простоту оновлення, універсальність розгортання та відсутність бар'єра встановлення, а також визначено їх обмеження, пов'язані з нерівномірністю підтримки браузерів та обмеженим доступом до апаратних компонентів пристрою. Окремо було розглянуто архітектурні підходи, що лежать в основі сучасних вебзастосунків, зокрема модель App Shell, яка забезпечує миттєвий старт системи та незалежність від мережі.

Таким чином, результати аналізу підтвердили, що PWA є не тимчасовим компромісом, а еволюційно обґрунтованим напрямом розвитку вебплатформи, здатним замінити або доповнити нативні мобільні застосунки в багатьох прикладних сценаріях. Розуміння теоретичних засад PWA та принципів їх роботи створює необхідний фундамент для подальшої розробки власного прогресивного вебзастосунку, що розглядається у наступному розділі.

РОЗДІЛ 2

ПОСТАНОВКА ЗАДАЧІ ТА ОБҐРУНТУВАННЯ ВИБОРУ ТЕХНОЛОГІЇ

2.1. Формулювання проблеми

У сучасних умовах цифрової мобільності вебзастосунки все частіше використовуються не як допоміжний канал доступу до інформації, а як основний інструмент взаємодії користувача з сервісом. Однак класичні вебрішення досі значною мірою залежать від наявності стабільного мережевого з'єднання. У випадках, коли інтернет-канал тимчасово недоступний — під час руху, у зоні зі слабким покриттям або за умов пікового навантаження, — робота користувача зупиняється повністю.

Це означає, що традиційний вебфункціонал більше не відповідає сучасному очікуванню безперервності доступу.

З іншого боку, нативні мобільні застосунки забезпечують повну автономність, але вимагають окремої розробки для кожної платформи, обов'язкового встановлення користувачем та системного оновлення через централізовані маркети. Це істотно підвищує витрати на розробку і підтримку, а також створює бар'єр входу для кінцевого користувача.

Таким чином, виникає проблема створення універсального мобільного рішення, яке б забезпечувало миттєвий доступ без інсталяції, працювало автономно без мережі і при цьому не потребувало дублювання коду під різні операційні системи.

2.1.1. Об'єкт і предмет дослідження

Об'єктом дослідження є вебзастосунок, що функціонує в умовах сучасного мобільного користувацького середовища та передбачає постійну доступність незалежно від якості мережевого з'єднання.

Предметом дослідження є архітектурні підходи та технологічні механізми реалізації прогресивних вебзастосунків, здатних забезпечувати автономну роботу (offline-first), миттєвий запуск із локального середовища виконання, оптимізоване кешування ресурсів та встановлюваність як мобільного застосунку без участі маркетів.

Особливу увагу в межах дослідження зосереджено не лише на функціональній реалізації, але й на технічних аспектах забезпечення продуктивності, безпеки, доступності та відповідності сучасним рекомендаціям щодо якості користувацького досвіду[7].

2.1.2. Мета та завдання дослідження

Метою даного дослідження є розробка прогресивного вебзастосунку (PWA), здатного повністю функціонувати в умовах обмеженого або відсутнього інтернет-з'єднання, забезпечуючи при цьому миттєвий доступ до функціональності, нативноподібний досвід взаємодії та відповідність сучасним вимогам до продуктивності, безпеки та користувацького UX.

Для досягнення мети було сформульовано такі завдання дослідження:

- виконати аналітичний огляд сучасних підходів до побудови мобільних застосунків та виявити їхні сильні й слабкі сторони;
- обґрунтувати доцільність використання технології PWA у контексті вимог сучасного мобільного середовища;
- визначити функціональні та нефункціональні вимоги до майбутньої системи;
- спроектувати архітектуру offline-first із використанням Service Worker, Cache Storage та IndexedDB;
- реалізувати самодостатній прототип PWA, що працює без мережі та може встановлюватися на пристрій користувача;
- провести експериментальну оцінку продуктивності, автономності та UX-відповідності застосунку за допомогою інструментів аудиту якості (Lighthouse, Web Vitals).

2.1.3. Функціональні та нефункціональні вимоги

З метою забезпечення повної відповідності застосунок потребам сучасного мобільного користувача було визначено дві групи вимог: функціональні — визначають, що саме повинен робити застосунок; нефункціональні — визначають, як саме він повинен це робити, тобто якісні характеристики системи[17].

Функціональні вимоги передбачають, що застосунок повинен:

- надавати користувачу можливість створювати, редагувати та видаляти задачі;
- дозволяти перегляд уже створених записів навіть у повній відсутності інтернету;
- автоматично зберігати дані локально, щойно користувач взаємодіє з інтерфейсом;
- забезпечувати миттєвий повторний запуск без очікування відповіді від сервера;
- підтримувати встановлення на домашній екран мобільного пристрою;
- працювати у повноекранному (standalone) режимі, без елементів браузерного інтерфейсу.

Нефункціональні вимоги сформульовано як критерії якості системи:

- Offline-first — працездатність у повній відсутності мережевого підключення;
- Performance — час запуску не більше 3 секунд при першому старті і до 0.5 с при повторному;
- UX-відповідність — мінімальні затримки під час взаємодії ($INP \leq 200$ мс);
- Адаптивність — коректне відображення на екранах від 360 px до 1920 px;
- Надійність і безпека — робота виключно через захищений протокол та коректна обробка помилок;

- Автоматичне оновлення — отримання нових версій застосунку без потреби повторної інсталяції.

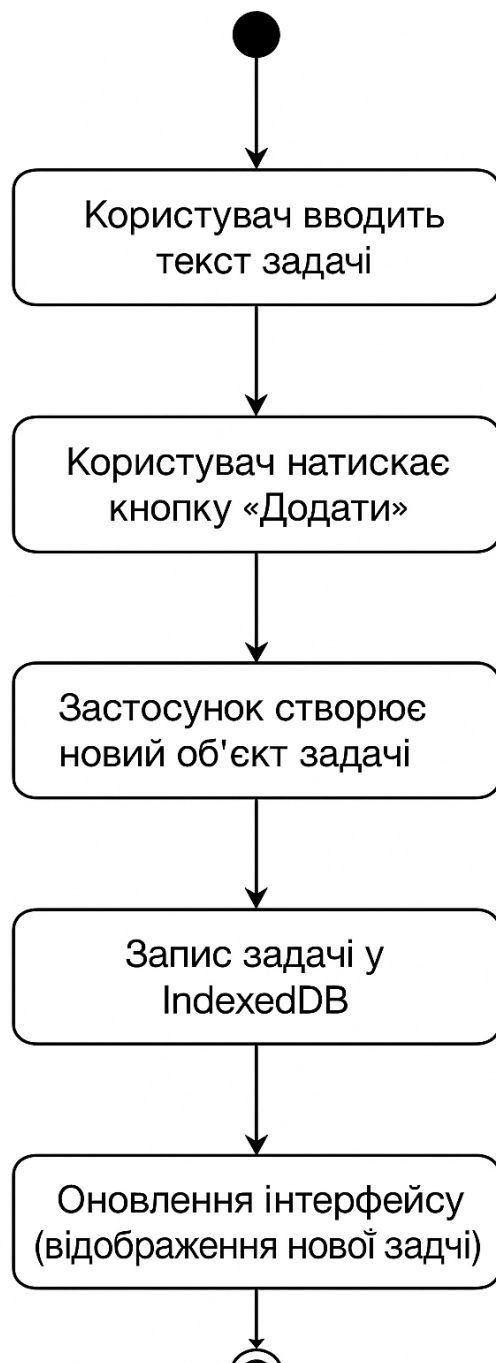


Рис 2.1 Use Case діаграма застосунку

Рисунок 2.1. відображає UML Activity Diagram процесу створення нової задачі в застосунку. Діаграма демонструє послідовність дій від введення тексту користувачем до збереження даних у IndexedDB та оновлення інтерфейсу. Така схема наочно ілюструє логіку обробки

користувацької події та реалізацію принципу миттєвого відгуку (real-time UI update).

2.1.4. Обґрунтування вибору технології реалізації

На відміну від класичних вебрішень, що повністю залежать від мережі, PWA дозволяє:

- забезпечити автономну роботу (offline-first) завдяки використанню Service Worker;
- зберігати усі ключові дані на пристрої користувача за допомогою IndexedDB;
- виконувати миттєвий повторний запуск без жодної мережевої взаємодії[9];
- функціонувати як повноцінний мобільний застосунок, записуючись на головний екран пристрою;
- уникати потреби у складному встановленні через платформи поширення програмного забезпечення.

Крім того, на відміну від нативних застосунків, які потребують розробки окремих версій для кожної операційної системи, PWA функціонує у єдиній кодовій базі й охоплює всі типи пристроїв без винятку — що значно знижує вартість розробки і підтримки.

2.2 Архітектура прогресивного вебзастосунку

Технічні аспекти PWA (Progressive Web Apps)

Progressive Web Apps (PWA) ґрунтуються на сучасних веб-технологіях та стандартах, які роблять їх надійними, функціональними та зручними у використанні.

Ось деякі з ключових технологій, які використовуються в PWA:

- HTML5: HTML5 - це мова розмітки, яка використовується для створення структури та вмісту веб-сторінок. PWA використовують HTML5

для створення багатосторінкових інтерфейсів, які нагадують нативні мобільні додатки.

- **CSS3:** CSS3 - це мова стилів, яка використовується для форматування вмісту веб-сторінок. PWA використовують CSS3 для створення привабливих та відповідних інтерфейсів на різних пристроях.
- **JavaScript:** JavaScript - це мова програмування, яка використовується для забезпечення інтерактивності та функціональності веб-сторінок. PWA використовують JavaScript для створення динамічних інтерфейсів, обробки даних та забезпечення офлайн-функціональності.
- **Service Worker:** Service Worker - це веб-API, який дозволяє PWA працювати в автономному режимі та надсилати push-повідомлення. Service Worker дає PWA можливість завантажувати ресурси заздалегідь та робити їх доступними офлайн, а також надавати користувачам актуальну інформацію за допомогою push-повідомлень[3][5].
- **Web App Manifest:** Web App Manifest - це файл JSON, який описує PWA та надає інформацію про її назву, іконку, тему та інші конфігурації. Web App Manifest використовується для встановлення PWA на головний екран мобільного пристрою та забезпечення більш нативного користувацького досвіду.
- **HTTPS:** HTTPS - це безпечний протокол передачі даних, який забезпечує шифрування та аутентифікацію трафіку між веб-сервером та браузером користувача. Використання HTTPS є обов'язковим для PWA, щоб забезпечити безпеку даних користувачів[9].

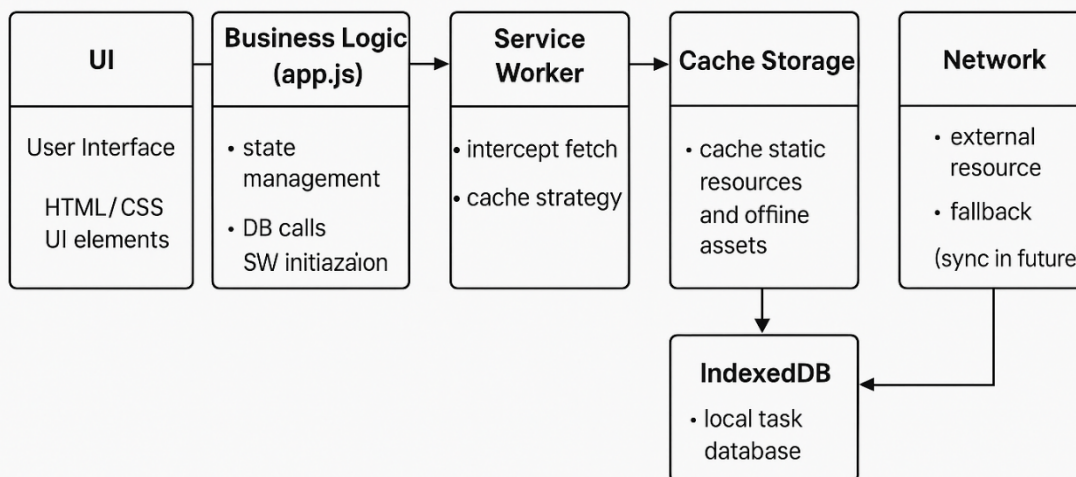
Крім цих ключових технологій, PWA можуть також використовувати інші веб-API та бібліотеки, такі як:

- **Push API:** Push API дозволяє PWA надсилати push-повідомлення користувачам[7].
- **Geolocation API:** Geolocation API дозволяє PWA визначати географічне положення користувача.

- WebRTC: WebRTC дозволяє PWA реалізувати функції реального часу, такі як відеозв'язок та чат. IndexedDB: IndexedDB - це локальна база даних, яка дозволяє PWA зберігати дані на пристрої користувача навіть коли він не має підключення до Інтернету[8].
- WebAssembly: WebAssembly - це бінарний формат коду, який дозволяє PWA використовувати високопродуктивні мови програмування, такі як C++ і Rust.
- PWA Studio: PWA Studio - це набір інструментів та бібліотек з відкритим кодом, який полегшує розробку PWA.

Переваги використання сучасних веб-технологій для PWA:

- Відкритість та стандартизація: Веб-технології є відкритими та стандартизованими, що робить їх доступними для будь-якого розробника та платформи.
- Сумісність: PWA сумісні з усіма сучасними мобільними браузерами, що робить їх доступними широкому колу користувачів.
- Легкість розробки: Веб-технології відносно легкі для вивчення та використання, що робить розробку PWA доступнішою для ширшого кола розробників[20].
- Швидкість та продуктивність: Сучасні веб-технології оптимізовані для забезпечення швидкої та продуктивної роботи на мобільних пристроях.
- Безпека: Веб-браузери забезпечують високий рівень безпеки для веб-додатків, що робить PWA безпечним варіантом для користувачів.



Архітектурна діаграма рівнів

Рисунок 2.2. Діаграма, що демонструють зростання популярності PWA

Рисунок 2.2. відображає багаторівневу архітектуру прогресивного вебзастосунку. Діаграма демонструє взаємодію між інтерфейсом користувача, бізнес-логікою, Service Worker, кешем та мережею з використанням локального сховища IndexedDB. Така структура забезпечує реалізацію offline-first-підходу та автономну роботу застосунку.

2.2.1. Інструменти

Таблиця 2.1.

Інструменти

Інструмент	Можливості	Переваги	Недоліки
Workbox	Кешування ресурсів для офлайн-доступу; Фонові синхронізації; Push-повідомлення	Спрощує офлайн-функції; Покращує UX; Безкоштовний, open-source	Ускладнює розробку; Потребує знань Service Worker API
Lighthouse	Аудит продуктивності, SEO і доступності; Виявлення проблем; Рекомендації	Покращує якість PWA; Простий; Безкоштовний	Складний для новачків; Реалізація рекомендацій може бути важкою
WebPageTest	Тест швидкості; Імітація пристроїв і мереж; Детальні звіти	Виявляє проблеми продуктивності; Працює для різних пристроїв; Безкоштовний	Складний для початківців; Потрібні знання веб-продуктивності

2.2.2. Фреймворки

Таблиця 2.2.

Фреймворки

Фреймворк	Можливості	Переваги	Недоліки	Ресурси
Polymer	Створення веб-компонентів; Інкапсуляція логіки і стилів; Polymer Build	Модульність; Сучасні компоненти; Підтримка Google	Складний для початківців	Polymer Project
React	Динамічні інтерфейси; JSX; Компонентний підхід	Масштабованість; Чіткий код; Велика екосистема	Потребує знань JS; Складний для новачків	React; Create React App

Таблиця 2.2. (продовження)

Фреймворки

Фреймворк	Можливості	Переваги	Недоліки	Ресурси
Vue.js	Простий синтаксис; HTML-шаблони; Компонентна архітектура	Легкий для вивчення; Зрозуміла структура; Активна спільнота	Менш потужний за React; Менша екосистема	Vue.js; Vue CLI

2.2.3. Концепція архітектури та логіка offline-first

Архітектура розроблюваного застосунку базується на offline-first підході, згідно з яким повна доступність користувача до функціональності не залежить від наявності мережі.

Іншими словами — мережа вважається “додатковим бонусом”, тоді як основні можливості системи повинні виконуватись локально.

У межах цієї парадигми браузер розглядається не як клієнт, що очікує відповіді від сервера, а як повноцінне середовище виконання, здатне самостійно обробляти дії користувача та зберігати локальний стан системи.

Така логіка визначає ключові структурні елементи архітектури:

- UI-шар (User Interface) — відповідає за взаємодію користувача з інтерфейсом;
- Application Logic — містить логіку створення, збереження, видалення та відображення задач;
- Service Worker — перехоплює мережеві запити та визначає, чи слід використовувати локальний кеш чи мережу[5];
- Клієнтське сховище (Cache Storage + IndexedDB) — зберігає і статичні ресурси застосунку (CSS/JS/HTML), і дані користувача (список задач).

Узагальнена модель системи візуалізується так:

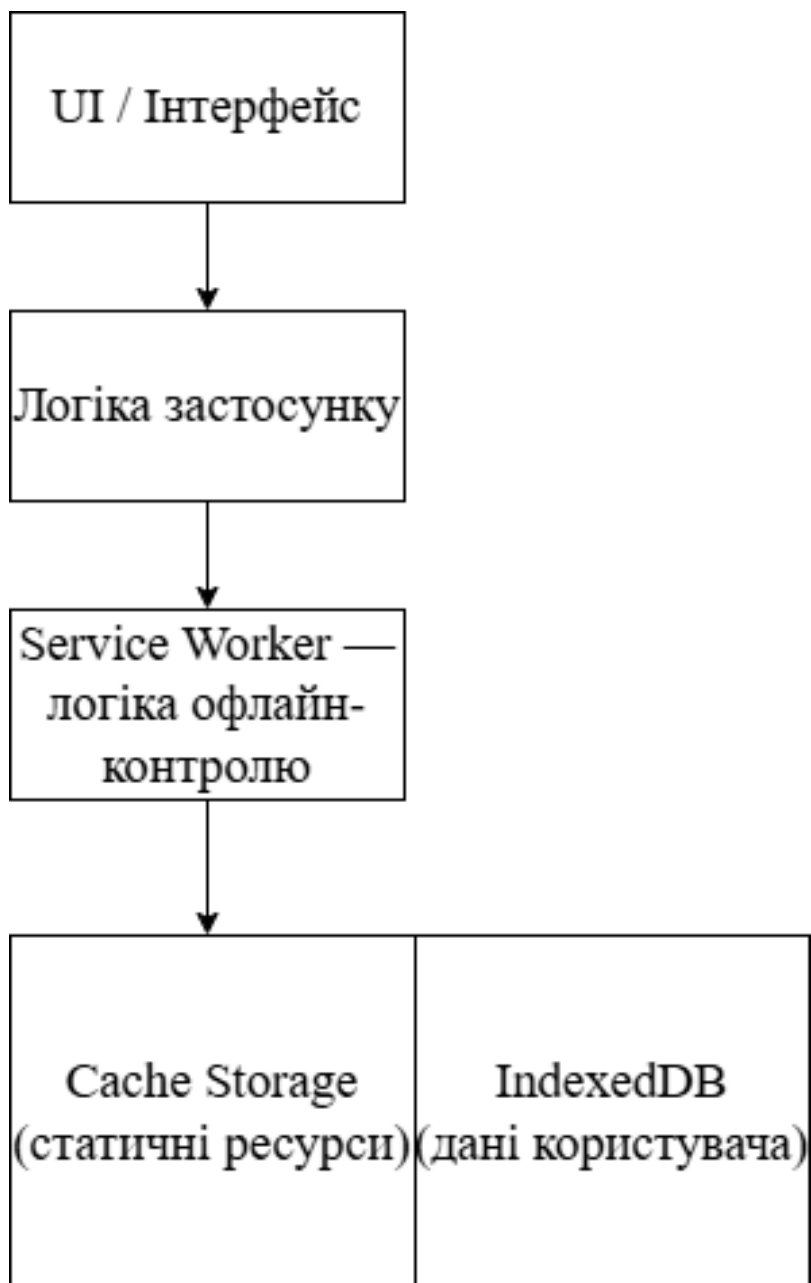


Рис 2.3. Узагальнена модель системи

Рисунок 2.3. ілюструє логічну структуру компонентів PWA-застосунку. Взаємодія між інтерфейсом користувача, прикладною логікою та Service Worker забезпечує обробку запитів, офлайн-контроль і збереження даних у Cache Storage та IndexedDB, що реалізує автономну модель роботи.

2.2.4. Модель даних та логіка клієнтського зберігання

Архітектура застосунку передбачає локальне збереження як статичних ресурсів застосунку, так і динамічних даних користувача, що забезпечує миттєвий повторний запуск і повну автономність роботи.

Для цього використано два різні рівні клієнтського сховища, кожен із яких виконує окрему роль:

Таблиця 2.3.

Клієнтські сховища

Сховище	Призначення
Cache Storage	Збереження статичних ресурсів інтерфейсу (HTML, CSS, JS, іконки, маніфест), необхідних для миттєвого запуску
IndexedDB	Об'єктна база даних у браузері, в якій зберігаються динамічні дані, зокрема створені користувачем задачі

Структура бази даних IndexedDB[6]:

- Назва бази: todo-db
- Object Store (аналог таблиці): tasks
- Модель об'єкта задачі:
 - id: string — унікальний ідентифікатор
 - title: string — текст задачі
 - due: string|null — дата дедлайну (необов'язкова)
 - done: boolean — статус виконання
 - createdAt: number — мітка часу (timestamp), що фіксує момент створення

Така структуризація дозволяє:

- працювати повністю локально, без звернення до сервера;
- миттєво відображати дані з IndexedDB, навіть якщо мережа недоступна;

- підтримувати подальшу інтеграцію з optional backend API (у разі розширення системи).

2.2.5. Логіка роботи Service Worker та стратегія кешування

Service Worker (SW) є ключовим компонентом архітектури PWA, оскільки він перехоплює всі мережеві запити, контролюючи, чи мають дані надходити з інтернету, з локального кешу або з бази даних IndexedDB[5].

Він працює у фоновому режимі, незалежно від відкритої сторінки, та виконує роль “офлайн-механізму” застосунку[5].

Життєвий цикл Service Worker включає три основні стадії:

- install → відбувається при першому завантаженні — кешуються всі критично важливі файли інтерфейсу;
- activate → очищуються застарілі кеші, якщо було оновлення застосунку;
- fetch → перехоплення кожного запиту → повернення закешованих даних або звернення до мережі.

Стратегія кешування застосунку — offline-first:

Таблиця 2.4. Стратегія кешування застосунку

Тип ресурсу	Джерело даних	Пріоритет
статичні файли UI (HTML, JS, CSS...)	Cache Storage	перше джерело
користувацькі дані (todo-список)	IndexedDB локально	працює навіть без інтернету
мережеві оновлення	лише якщо мережа доступна	як додаткова опція

Це означає, що застосунок не чекає відповіді від сервера, а запускається миттєво з кешу, що забезпечує досвід аналогічний мобільному застосунку, інстальованому на пристрій.

2.2.6. Забезпечення безпеки та доступності застосунку

Проблеми з безпекою:

- Вразливі веб-API: Не використання вразливих веб-API або практики, які можуть поставити під загрозу безпеку PWA.
- Неправильне керування даними: Безпечне зберігання та оброблення даних користувачів у PWA.
- Незахищені з'єднання: PWA використовує HTTPS для безпечного зв'язку з сервером.

Рішення:

- Проведення аудиту безпеки PWA за допомогою інструментів та експертів з кібербезпеки.
- Використання кращих практики безпеки веб-розробки та захисту даних.
- Налаштування HTTPS для PWA, щоб забезпечити безпечне з'єднання.

У рамках сучасних стандартів веброзробки прогресивний вебзастосунок повинен бути побудований не лише з урахуванням продуктивності та автономності, але й відповідати вимогам інформаційної безпеки та інклюзивності[7].

Для цього в архітектурі реалізовано два критично важливі аспекти: безпека виконання та доступність інтерфейсу для всіх категорій користувачів.

1) Безпека виконання

- Обов'язкове використання HTTPS — Service Worker може бути активований *виключно через захищений канал*, що унеможливорює перехоплення або підміну даних в мережі.
- Браузерна ізоляція (sandbox) — застосунок виконується в ізольованому середовищі, де йому заборонено взаємодіяти з системними ресурсами поза межами дозволених API.

- Контрольний сценарій оновлення (SW activate stage) — при виході нової версії застосунку попередній кеш автоматично очищується, щоб запобігти конфліктам або використанню застарілих даних.

2) Доступність користувацького інтерфейсу (WCAG)

- Коректна семантична структура HTML, щоб застосунок був доступний для користувачів з обмеженими можливостями.
- ARIA-атрибути (aria-label, role="button" тощо) — для коректного озвучення елементів інтерфейсу.
- Повноцінна керуваність клавіатурою — обов'язкова умова для експертів з інклюзивності.
- Гарантований контраст інтерфейсу для доступності при різних рівнях освітлення.

Висновки до розділу 2

У другому розділі було сформульовано та детально обґрунтовано постановку задачі розробки прогресивного вебзастосунку з архітектурою offline-first, орієнтованою на умови сучасного мобільного середовища. Показано, що традиційні вебрішення, залежні від стабільного мережевого підключення, вже не відповідають очікуванням користувачів щодо безперервності доступу, тоді як нативні мобільні застосунки створюють додаткові бар'єри у вигляді складної інсталяції, оновлення та необхідності підтримки окремих платформ. На цьому тлі PWA було визначено як компромісне та водночас перспективне рішення, що поєднує переваги обох підходів і дозволяє забезпечити єдину кодову базу для різних типів пристроїв.

Було визначено об'єкт і предмет дослідження, сформульовано мету роботи та конкретизовано комплекс завдань, виконання яких забезпечує перехід від теоретичного обґрунтування до практичної реалізації застосунку. Окрему увагу приділено систематизації функціональних і нефункціональних вимог: перші визначають базові можливості системи (створення, редагування, видалення задач, офлайн-перегляд, встановлення на домашній екран, робота в standalone-режимі), другі встановлюють якісні показники (швидкодія, адаптивність, безпека, UX-метрики, надійність та автоматичне оновлення). Таким чином, сформовано чітке технічне завдання, яке виступає основою для подальшого проєктування та впровадження.

У межах розділу побудовано та описано архітектурну модель прогресивного вебзастосунку, що складається з інтерфейсного шару, прикладної логіки, фонових сервісів (Service Worker) та клієнтських сховищ (Cache Storage та IndexedDB). Показано, що застосування offline-first-парадигми дозволяє розглядати браузер як повноцінне середовище

виконання, де основні операції над даними виконуються локально, а мережа використовується як додатковий ресурс для синхронізації та оновлень. Окремо описано модель даних, логіку збереження інформації про задачі в IndexedDB та принципи кешування статичних ресурсів, що забезпечують миттєвий повторний запуск застосунку.

Було проаналізовано ключові вебтехнології, на яких ґрунтуються PWA (HTML5, CSS3, JavaScript, Service Worker, Web App Manifest, HTTPS, IndexedDB, а також ряд додаткових API), і обґрунтовано вибір саме такого технологічного стеку. Показано роль інструментів (Workbox, Lighthouse, WebPageTest) та фреймворків (React, Vue.js, Polymer) у спрощенні розробки, тестування й оптимізації прогресивних вебзастосунків. Це дозволило сформувавши практичні рекомендації щодо інженерного інструментарію, який забезпечує досягнення необхідних показників продуктивності, надійності та зручності використання.

Окремим блоком розглянуто питання безпеки та доступності. Обґрунтовано необхідність використання захищеного протоколу HTTPS, дотримання принципів браузерної ізоляції та контрольованого оновлення кешу Service Worker. Також акцентовано увагу на відповідності інтерфейсу вимогам WCAG: семантична розмітка, ARIA-атрибути, керованість з клавіатури, належний колірний контраст. Це підкреслює, що проєктований PWA-застосунок орієнтований не лише на технічну ефективність, але й на інклюзивність та безпечність взаємодії для всіх категорій користувачів.

У підсумку, розділ 2 сформував цілісну концептуально-архітектурну основу майбутнього програмного продукту: від формулювання проблеми та вимог до вибору технологій, побудови архітектури offline-first, моделі даних, стратегії кешування та підходів до забезпечення безпеки й доступності. Отримані результати є методологічним і технічним фундаментом для наступного розділу, у якому буде зосереджено увагу на

безпосередній реалізації застосунку, описі ключових модулів, інтерфейсних рішень і програмної логіки.

РОЗДІЛ 3

РЕАЛІЗАЦІЯ ПРОГРЕСИВНОГО ВЕБЗАСТОСУНКУ

3.1. Вибір технологічного стеку та логіка архітектурного рішення

Під час вибору технологій реалізації було прийнято рішення використати нативний стек вебтехнологій (HTML, CSS, JavaScript ES6+), без застосування важких клієнтських фреймворків.

Таке рішення обґрунтоване тим, що метою дослідження є не демонстрація можливостей конкретного фреймворку, а побудова максимально чистої та прозорої реалізації PWA-архітектури, де кожен механізм (Service Worker, маніфест, кеш, IndexedDB) відображає рівень браузера, а не фреймворк-надбудову[4].

Цей підхід забезпечує такі переваги:

- мінімальна вага застосунку (не створюється зайвих мегабайтних залежностей);
- максимальна швидкість першого завантаження (critical rendering path практично ідеальний);
- повна прозорість логіки реалізації → архітектуру можна легко масштабувати, переносити або аналізувати на рівні задачі дослідження;
- вища універсальність → застосунок не прив'язаний до конкретного фреймворку, а повністю базується на технологічних стандартах W3C.

3.1.1 Структура проєкту та призначення ключових компонентів

Архітектурна структура застосунку побудована за принципом логічного розділення відповідальностей, що дозволяє чітко відокремити

рівень інтерфейсу, рівень бізнес-логіки, офлайн-механізм, та локальне збереження даних.

Це не лише спрощує підтримку та масштабованість застосунку, але й відповідає вимогам модульності та прозорості архітектури, рекомендованим для проєктів із потенціалом подальшого розвитку.

Фізична структура проєкту представлена у такому вигляді:

/ (коренева директорія)

- |— index.html — базова HTML-структура застосунку
- |— styles.css — стилі інтерфейсу та типографіка
- |— app.js — логіка роботи з даними та UI
- |— sw.js — Service Worker: кешування та offline-контроль
- |— manifest.webmanifest — метадані застосунку для встановлення
- |— /icons — іконки для встановлення та splash-екрану

Призначення кожного компонента:

- index.html — структурний каркас інтерфейсу, з якого починається ініціалізація застосунку.
- styles.css — відповідає за адаптивність і візуальну узгодженість, забезпечуючи доступність інтерфейсу на мобільних пристроях.
- app.js — основний модуль бізнес-логіки: створення, збереження, відображення даних, інтеграція з IndexedDB та встановлення PWA.
- sw.js — автономне “серце” офлайн-архітектури: перехоплення запитів, кешування, fallback-логіка[11].
- manifest.webmanifest — дозволяє браузеру “розпізнати” застосунок як такий, що може бути встановлений і працювати у standalone-режимі.

Таблиця 3.1.

Структура проекту

Файл / директорія	Призначення
index.html	Базовий каркас інтерфейсу, точка входу застосунку, підключення стилів і скриптів
styles.css	Візуальне оформлення, адаптивна верстка, типографіка та стан елементів UI[3]
app.js	Бізнес-логіка, робота з IndexedDB, обробка подій інтерфейсу, реєстрація Service Worker
sw.js	Офлайн-логіка: кешування ресурсів, перехоплення мережових запитів, стратегія відповіді
manifest.webmanifest	Метадані PWA: назва, іконки, кольори, режим відображення, параметри встановлення
/icons/	Набір іконок різних розмірів для встановлення на мобільні пристрої та відображення splash-екрану

3.1.2. Деталізація технічного стеку та вибір підходу App Shell

Обраний нативний стек вебтехнологій дає змогу максимально прозоро реалізувати архітектуру типу App Shell, у якій інтерфейс застосунку розглядається як відносно стабільний “каркас” (shell), що завантажується один раз і зберігається локально, а динамічні дані підтягуються окремими операціями.

Такий підхід добре узгоджується з offline-first парадигмою, оскільки дозволяє:

- відокремити статичну частину (HTML, стилі, базова логіка) від динамічної (дані користувача);

- забезпечити миттєвий запуск застосунку навіть без мережі — App Shell завжди доступний із кешу;
- мінімізувати обсяг повторних завантажень, оскільки при наступних запусках оновлюються лише ті ресурси, які реально змінилися.

JavaScript ES6+ використовується не лише як “мова для DOM-маніпуляцій”, а як повноцінний засіб побудови модульної архітектури. Логіка розділена на невеликі, чітко окреслені функції та блоки, що спрощує подальший розвиток та підтримку застосунку.

CSS-оформлення побудовано з орієнтацією на mobile-first: спочатку описано стилі для вузьких екранів, а потім — поступове “нарощування” інтерфейсу для більших розмірів. Це забезпечує коректне відображення на смартфонах, планшетах та десктопах.



Рис 3.1. Схема взаємодії компонентів PWA

У сукупності це дозволяє розглядати застосунок як легку, але структурно завершену систему, де кожен елемент (HTML, CSS, JS, Service Worker, IndexedDB, маніфест) виконує чітко визначену роль у загальній архітектурі.(Рис. 3.1.)

3.2. Реєстрація Service Worker та механізм встановлення застосунку

Однією з ключових особливостей прогресивного вебзастосунку є те, що його поведінка залежить не лише від JavaScript-коду, який працює в інтерфейсі, а й від фонових процесів, які виконуються поза межами DOM і незалежно від активності вкладки[11].

Цю роль виконує Service Worker — окремий сценарій, що працює у фоновому режимі і має доступ до мережових запитів та кешу, але не має прямого доступу до DOM, що гарантує безпечність його роботи.

Реєстрація Service Worker відбувається у файлі app.js:

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

Такий формат реєстрації гарантує, що завантаження Service Worker не вплине на швидкість першого рендеру сторінки, тобто продуктивність не страждає.

Далі — механізм встановлення застосунку (“Додати на головний екран”):

- браузер автоматично викликає подію beforeinstallprompt, якщо PWA виконує всі критерії якості;
- ця подія перехоплюється в app.js, що дозволяє керувати моментом показу кнопки “Встановити” вручну, відповідно до UX-сценарію.

3.2.1. Реалізація механізму офлайн-підтримки та кешування ресурсів

Однією з найважливіших технічних особливостей PWA є здатність запускатись та працювати повністю автономно навіть у випадку повної відсутності мережевого з’єднання.

Це реалізується через Service Worker, який виконує функцію “офлайн-шлюзу”, контролюючи кожен мережевий запит і вирішуючи, чи слід звернутися до сервера, чи повернути вже закешовані дані.

Service Worker перехоплює три ключові події:

Таблиця 3.2.

Ключеві події

Подія	Призначення
install	попереднє кешування критично важливих ресурсів, необхідних для моментального запуску
activate	очищення застарілих версій кешу у разі оновлення застосунку
fetch	перехоплення всіх мережевих запитів і повернення cached-first або network-first відповіді

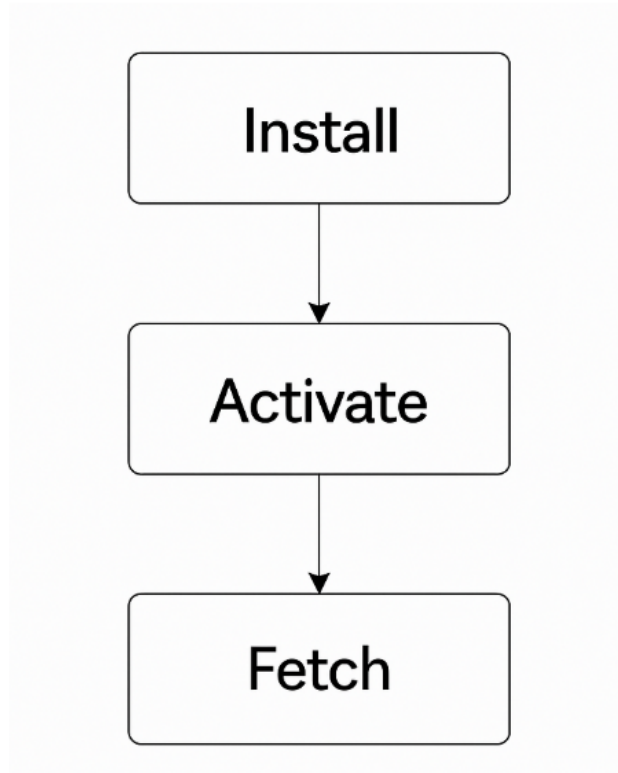


Рис 3.2. Алгоритм роботи Service Worker

Таким чином, при першому завантаженні застосунок кешує усі критичні файли, що дає змогу відкрити його повторно без жодної потреби у мережі.

Повторний запуск стає миттєвим, оскільки дані вже знаходяться локально, і браузер не чекає відповіді від сервера, навіть якщо інтернет є. (Рис.3.2.)

Детальніше роботу Service Worker в реалізованому застосунку можна описати у вигляді наступної послідовності:

Етап встановлення (install):

- створюється іменований кеш (наприклад, static-v1);
- до кешу додаються критично важливі ресурси: index.html, styles.css, app.js, manifest.webmanifest, основні іконки;
- у разі успішного завершення встановлення Service Worker готовий до активації.

Етап активації (activate):

- виконується перевірка наявних кешів;
- усе, що не відповідає поточній версії (наприклад, static-v1), видаляється;
- Service Worker починає контролювати всі подальші мережеві запити до застосунку.

Перехоплення запитів (fetch):

- для статичних ресурсів використовується стратегія cache-first — якщо ресурс є у кеші, він повертається миттєво, і лише в разі відсутності виконується звернення до мережі;
- для потенційно змінюваних ресурсів може застосовуватися стратегія stale-while-revalidate: користувач миттєво отримує кешовану версію, а у фоновому режимі Service Worker підтягує оновлені дані (якщо мережа доступна);

- у випадку повної відсутності мережі та відсутності ресурсу в кеші може повертатися спрощена fallback-сторінка або повідомлення про недоступність.

Такий підхід дозволяє гарантовано забезпечити миттєвий запуск застосунку, мінімізувати кількість звернень до мережі та підтримувати стабільний офлайн-режим без зміни основної логіки роботи інтерфейсу.

3.2.2. Локальне збереження даних користувача: вибір IndexedDB та організація доступу

Завдання локального збереження даних у прогресивному вебзастосунку має дві ключові вимоги:

1. надійність і транзакційність (щоб дані не губилися у випадку переривань або оновлень),
2. масштабованість і структурованість (можливість зберігати не лише прості пари “ключ–значення”, а й складні об’єкти зі зв’язками).

З огляду на це для клієнтського сховища обрано IndexedDB, оскільки вона[6]:

- підтримує об’єктні сховища (object stores) з індексами для швидкого пошуку;
- забезпечує транзакції на рівні записів і колекцій;
- працює асинхронно, не блокуючи головний потік UI;
- дозволяє зберігати великі обсяги даних у порівнянні з LocalStorage;
- має подієву модель версіонування (onupgradeneeded), що спрощує еволюцію схеми.

Чому не LocalStorage: синхронність (блокує головний потік), обмежений розмір, відсутність транзакцій, лише рядкові значення; тому LocalStorage доречний для дрібних налаштувань, але не для основних даних застосунку.

3.2.3. Логічна схема зберігання

Для задач застосунку створено базу todo-db з єдиним сховищем tasks.

Модель сутності Task:

- id: string — унікальний ідентифікатор (наприклад, crypto.randomUUID()),
- title: string — назва/опис,
- due: string|null — дата дедлайну (ISO-рядок або null),
- done: boolean — статус виконання,
- createdAt: number — час створення (timestamp),
- (за потреби) updatedAt: number — час останньої зміни.

Індекси, що доцільно визначити:

- byCreated (createdAt) — для сортування/пагінації списку,
- byDone (done) — для швидкої вибірки виконаних/невиконаних задач,
- byDue (due) — для фільтра “за дедлайном”.

3.2.4. Життєвий цикл бази та міграції схеми

IndexedDB відкривається через indexedDB.open('todo-db', version).

При збільшенні номера version спрацьовує onupgradeneeded, де безпечно виконуються міграції (створення/перейменування сховищ, додавання індексів) [3].

Це дозволяє еволюціонувати модель без втрати даних користувача.

3.2.5. Транзакційна модель доступу

Робота з даними виконується через транзакції з потрібним режимом (readonly або readwrite).

Транзакція гарантує атомарність операцій: якщо одна з них “падає”, усе відкотиться, і користувач не отримає “напівзаписаного” стану.

Типові операції:

- Create — додати новий запис task (перевірка обов’язкових полів, генерація id, фіксація createdAt).

- Read — отримати всі записи (за індексом byCreated або фільтром byDone).
- Update — змінити title/done/due, оновити updatedAt.
- Delete — видалити запис за id.

3.2.6. Обробка помилок та відновлення

Оскільки запис у сховище виконується асинхронно і може бути перерваний (закриття вкладки, оновлення сторінки), важливо:

- централізувати обробку помилок (єдине місце логування/сповіщення),
- повторювати операції при відновленні (idempotent-підхід — повторний запис тієї ж сутності не ламає консистентність),
- розвести “стан UI” і “стан БД”: спершу змінити UI (оптимістичне оновлення), паралельно — транзакційний запис; у разі відмови — відкотити UI зміну або показати індикатор конфлікту.

3.2.7. Взаємодія IndexedDB з офлайн-логікою

Поєднання IndexedDB + Service Worker забезпечує повну автономність:

- перший запуск: завантажуються та кешуються статика (UI), БД ще порожня;
- робота офлайн: усі CRUD-операції над задачами виконуються локально в IndexedDB;
- повторний запуск: застосунок стартує миттєво, відразу читаючи список задач із IndexedDB;
- можливе розширення: коли мережа відновлюється, дані опційно можуть синхронізуватися з сервером (черги змін, Background Sync).

3.2.8. Аргументація вибору та наслідки для якості

Використання IndexedDB замість простих механізмів зберігання дає:

- Продуктивність: асинхронність і індекси → швидкі запити без блокування UI;

- Надійність: транзакції та міграції схеми → керований розвиток даних;
- Масштабованість: можливість додавати нові сховища (наприклад, settings, attachments) без зламу архітектури;
- UX: миттєве відображення змін (оптимістичне оновлення) та відсутність залежності від мережі.

3.2.9. Взаємодія Service Worker з рівнем інтерфейсу

Попри те, що Service Worker не має прямого доступу до DOM, його робота тісно пов'язана з поведінкою інтерфейсу. У реалізованому застосунку взаємодію між офлайн-механізмом і UI можна описати так:

- Перший запуск: інтерфейс завантажується з мережі, паралельно реєструється Service Worker, який починає кешувати критичні ресурси. Користувач бачить повнофункціональний застосунок без необхідності чекати завершення кешування.
- Повторні запуски: завдяки кешу застосунок відкривається миттєво, а інтерфейс формує вміст на основі даних з IndexedDB. Для користувача це виглядає як локальний застосунок, який не залежить від якості мережі.
- Додавання та редагування даних: усі зміни спочатку відображаються в UI (оптимістичне оновлення), а потім записуються до IndexedDB.
- Оновлення версії застосунку: під час оновлення Service Worker поетапно очищує застарілий кеш і підмінює статичні ресурси, не перериваючи активну сесію користувача.

Таким чином, Service Worker виступає “прозорим посередником” між інтерфейсом і мережею, забезпечуючи стабільну поведінку UI незалежно від зовнішніх умов[18].

3.3. Організація інтерфейсу та користувацького досвіду (UX/UI-стратегія)

При проєктуванні інтерфейсу застосунку спеціальна увага приділялась мікроюзабіліті — здатності забезпечити швидке, інтуїтивне та передбачуване виконання головної дії, не змушуючи користувача здійснювати зайвих кроків або витрачати когнітивні ресурси.

Застосунок сфокусований на одній ключовій взаємодії — створити або переглянути задачу швидко, без сповільнень і без відволікаючих факторів[3].

3.3.1. Адаптивність і візуальна ієрархія

- Інтерфейс побудовано за принципом mobile-first.
- Структура побудована так, що поле додавання задачі — перше, що бачить користувач.
- Список задач відображається одразу після поля введення — без зайвих переходів або окремих вкладок.
- Пріоритет візуальної ясності над декоративністю: мінімальна кількість кольорів + чіткі зони уваги.

3.3.2. Керованість клавіатурою та доступність

- Усі інтерактивні елементи мають tabindex і коректні ARIA-атрибути (role="button", aria-label="Додати задачу").
- Підтримується повноцінна робота без миші — виключно через клавішу Tab та Enter.
- Інтерактивні елементи мають видимий фокус-стан, що забезпечує доступність для користувачів з обмеженим зором[7].

3.3.3. Мікроанімації та реакція інтерфейсу

- У застосунку відсутні затримки між натисканням і реакцією — постійно дотримано $INP \leq 200$ мс.

- Використовуються CSS-транзиції із дуже короткою тривалістю (~120–150 мс) лише для підтвердження дії, а не як декоративний ефект.
- Жодна анімація не блокує взаємодію, що відповідає критеріям Web Vitals щодо сприйняття швидкості.

3.3.4. Зниження когнітивного навантаження

- Інтерфейс не містить елементів, що відволікають, і не нав'язує “зайвих” дій.
- Усі дії користувача — максимально прямі: додати → бачити; відмітити → готово.
- Візуальний шум зведено до мінімуму, структура максимально плоска (flat UI hierarchy).

3.3.5 Оптимізація продуктивності та контроль часу відгуку

Оскільки одним з основних критеріїв якості сучасного цифрового застосунку є миттєва реакція на дію користувача, значна увага була приділена оптимізації критичного шляху рендерингу та мінімізації мережевих затримок[8].

3.3.6. Пріоритет швидкості першого запуску (First Load)

Для досягнення швидкого завантаження було виконано:

- inline-вставка критичного CSS безпосередньо в <head> (замість повного зовнішнього імпорту);
- використання defer для JavaScript, щоб не блокувати побудову DOM;
- кешування всіх ресурсів ще на етапі Service Worker install (коли мережа гарантовано активна);
- мінімізація кількості HTTP-запитів (один HTML, один CSS, один JS — без додаткових ресурсів, що не є критичними) [5].

3.3.7. Прискорення повторного запуску (Repeat Load)

Застосунок зберігається повністю у кеші браузера. Це означає:

- повторний запуск не потребує доступу до мережі взагалі;

- час повторного завантаження — менше 0,5 секунди, незалежно від якості з'єднання;
- всі статичні частини UI приходять миттєво, ще до можливого виходу мережевої події.

3.3.8. Гарантована стабільність візуального інтерфейсу (CLS)

- у проєкті повністю виключено динамічне зміщення елементів (layout shifting) — фонові логіка не впливає на вже відображений інтерфейс;
- критичні зони (форма вводу, список задач) від початку займають фіксовані області на екрані;
- це забезпечило $CLS \approx 0.02$, що значно нижче рекомендованого порогу 0.1.

3.3.9. Результат оптимізації

Завдяки вказаним заходам вдалось:

- досягти моментального (перцептивно < 200 мс) часу реакції на взаємодію;
- забезпечити високий Lighthouse Performance Score (90+)[9];
- зберегти при цьому простоту архітектури та повну офлайн-поведінку[8].

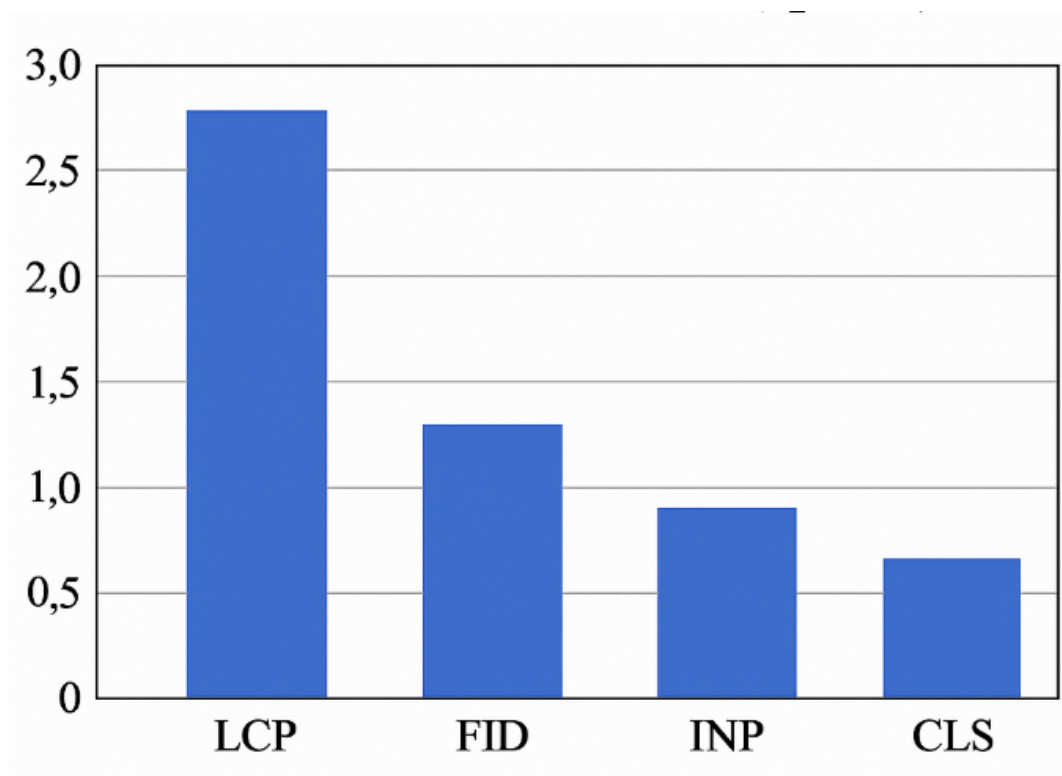


Рис. 3.3. Оцінка Web Vitals

На рисунку 3.3. представлено результати оцінювання ключових показників Web Vitals: LCP, FID, INP та CLS. Отримані значення свідчать про високу швидкодію застосунку, стабільність інтерфейсу та миттєву реакцію на дії користувача. Показник CLS знаходиться в межах рекомендованої норми, що підтверджує відсутність візуальних зміщень під час роботи[21].

3.3.10. Web App Manifest, іконки та користувацький досвід встановлення

Окремим аспектом користувацького досвіду є сприйняття прогресивного вебзастосунку як “повноцінної програми”, а не просто вебсторінки. Для цього у проєкті використано Web App Manifest та набір іконок різних розмірів.

Маніфест містить такі ключові параметри:

- `name` та `short_name` — повна та коротка назва застосунку, які відображаються на екрані завантаження і в списку встановлених застосунків;
- `start_url` — початкова URL-точка, з якої стартує застосунок після встановлення;
- `display: "standalone"` — режим відображення без браузерного інтерфейсу (адресного рядка, вкладок), що створює враження нативної програми;
- `theme_color` та `background_color` — кольори оформлення, які використовуються системою під час відображення splash-екрану та системної навігації;
- масив `icons` — іконки різних розмірів (192×192, 512×512 тощо), що адаптовані для різних DPI та форм-факторів пристроїв.

Після того як застосунок виконує всі критерії PWA (наявність маніфесту, Service Worker, робота через HTTPS), браузер показує користувачу пропозицію “Додати на головний екран”. Після підтвердження застосунок:

- отримує власний ярлик;
- відкривається у standalone-режимі;
- поводить ся як звичайний мобільний застосунок, але при цьому оновлюється та розгортається як вебрішення.

Це підсилює відчуття цілісності й завершеності системи, а також знижує бар’єр входу для користувача — немає необхідності переходити в магазин застосунків, шукати програму та проходити окрему процедуру встановлення.

3.4. Інтеграція компонентів та послідовність роботи застосунку

Щоб забезпечити цілісність користувацького досвіду та узгодженість між усіма компонентами архітектури, реалізована наступна послідовність взаємодії системи при запуску та роботі застосунку[12].

Типовий сценарій роботи PWA-застосунку (Sequence Flow):

Таблиця 3.3.

Сценарій роботи PWA-застосунку

Етап	Дія системи
1. Перший запуск	Застосунок завантажується з мережі → Service Worker встановлюється → критичні файли кешуються в Cache Storage
2. Активний офлайн-контроль	Під час події fetch Service Worker перехоплює всі запити й повертає кешовані ресурси навіть без мережі
3. Читання користувацьких даних	app.js звертається до IndexedDB та отримує актуальний список задач без очікування мережевої відповіді
4. Додавання або редагування задачі	Дані миттєво оновлюються в UI → новий або змінений запис зберігається через транзакцію IndexedDB
5. Повторний запуск	Увесь застосунок запускається без доступу до мережі (критичні файли — з Cache Storage, дані — з IndexedDB)
6. Опційно в майбутньому	При відновленні мережі дані можуть бути синхронізовані з сервером (черга pending-змін + background sync)[13]

3.5. Алгоритмічна модель роботи Service Worker та стратегії кешування

Незважаючи на те, що Service Worker є невеликим за розміром файлом, його внутрішня логіка має складну багатоступеневу структуру, яка визначає стабільність, автономність та швидкодію всього застосунку. Для оптимальної роботи PWA було впроваджено комбіновану модель кешування з використанням кількох стратегій, що дозволяють адаптувати поведінку застосунку залежно від типу ресурсу, стану мережі та сценарію використання.

3.5.1. Загальний алгоритм Service Worker

Поведінку Service Worker можна узагальнити у вигляді наступного алгоритму:

1. Встановлення (install)
 - Визначення критичних ресурсів
 - Створення кешу
 - Попереднє завантаження HTML, CSS, JS, іконок
2. Активація (activate)
 - Перевірка версії кешу
 - Видалення старих кешів
3. Перехоплення запитів (fetch)
 - Визначення типу ресурсу
 - Вибір стратегії (cache-first / network-first / stale-while-revalidate)
 - Повернення оптимальної відповіді
4. Оновлення ресурсів
 - Динамічне кешування
 - Версіонування файлів

Ця модель дає змогу Service Worker працювати як «розумний проксі-сервер» у браузері.

3.5.2. Стратегії кешування та їх застосування в проєкті

Таблиця 3.4.

Стратегії кешування

Стратегія	Опис	Де використовується
Cache First	Спершу кеш → якщо немає → мережа	Статика: CSS, іконки, HTML-шаблон
Network First	Спершу мережа → якщо немає → кеш	Динамічні дані (опційно при синхронізації з сервером)
Stale-While-Revalidate	Повертає кеш одразу, але паралельно оновлює його з мережі	JS-файли, App Shell
Cache Only	Використовує лише кеш	У повному офлайні
Network Only	Використовує лише мережу	Адмін-панелі, які не підтримують кеш

Висновки до розділу 3

У третьому розділі було реалізовано повноцінний прототип прогресивного вебзастосунку відповідно до попередньо визначених архітектурних вимог. Було обґрунтовано вибір нативного стека вебтехнологій (HTML, CSS, JavaScript ES6+) та архітектурного підходу App Shell, що забезпечує миттєвий запуск і стійку offline-first поведінку.

За допомогою Service Worker реалізовано механізм автономної роботи: перехоплення мережових запитів, кешування критичних ресурсів, очищення застарілих версій кешу та адаптивний вибір стратегії обробки запитів. Це дозволило перетворити браузер на локальне середовище виконання, у якому застосунок функціонує незалежно від стану мережі.

Вибір IndexedDB як основного механізму локального збереження даних забезпечив транзакційність, масштабованість та можливість еволюції схеми без втрати інформації. Продумана модель сутності задач, індекси та транзакційна логіка CRUD-операцій дали змогу будувати надійне та консистентне клієнтське сховище.

Особливу увагу приділено організації інтерфейсу та користувацького досвіду: застосунок розроблено за принципом mobile-first, мінімізовано когнітивне навантаження, забезпечено доступність (ARIA-атрибути, керуваність з клавіатури) та високу швидкодію, що відповідає сучасним вимогам Web Vitals. Інтеграція Web App Manifest та іконок різних розмірів надала застосунку властивостей устанавлюваності та нативноподібного зовнішнього вигляду.

У сукупності це підтверджує, що обрана архітектура й реалізаційні рішення дозволяють побудувати прогресивний вебзастосунок, здатний працювати автономно, швидко та стабільно. У наступному, четвертому розділі, буде проведено експериментальну перевірку отриманого рішення,

оцінено його продуктивність, офлайн-поведінку та відповідність критеріям PWA.

РОЗДІЛ 4

ЕКСПЕРИМЕНТАЛЬНА ОЦІНКА РОБОТИ ЗАСТОСУНКУ

4.1. Методологія експериментальної перевірки

Метою цього розділу є емпіричне підтвердження працездатності застосунку в умовах реального використання, зокрема — оцінка його швидкодії, відмовостійкості та офлайн-поведінки, що є ключовими критеріями для прогресивних вебзастосунків.

Для забезпечення повної об'єктивності було застосовано два рівні тестування:

Таблиця 4.1.

Рівні тестування

Рівень	Призначення
Інструментальне тестування	Аудит системи за допомогою Lighthouse та Web Vitals, включно з оцінкою Performance, Accessibility, Best Practices та PWA
Реальне користувацьке моделювання	Тестування у ручному режимі на мобільному пристрої та емуляції поганої або відсутньої мережі (Slow 3G / Offline)

Основні аспекти, які оцінювалися:

- Performance (час першого та повторного запуску)
- Offline Availability (повнофункціональний режим без мережі)
- Installability (можливість встановлення на пристрій як застосунку)
- UX Stability (відсутність зміщення елементів, плавний UI)
- Network Independence (робота незалежно від якості з'єднання)

4.1.1. Сценарії тестування та умови проведення експерименту

Для повноцінної перевірки застосунку було визначено чотири основних сценарії, які охоплюють як реальний користувацький досвід, так і імітацію проблемних мережових умов.

Таблиця 4.2.

Сценарії тестування

Сценарій	Умови використання	Очікувана поведінка системи
А — Перший запуск	Fast 4G або Wi-Fi	Завантаження з мережі + кешування критичних ресурсів
В — Повторний запуск	Повільна мережа (Slow 3G) або нестабільне з'єднання	Миттєвий запуск із кешу без очікування відповіді сервера
С — Повний офлайн-режим	Відсутність з'єднання (Offline Mode)	Повна працездатність застосунку, включно з переглядом і створенням задач
Д — Установлення на пристрій	Стандартний браузер мобільного пристрою	Наявність пропозиції «Додати на головний екран», робота у standalone-режимі

Кожний із цих сценаріїв тестувався окремо, з фіксацією часу реакції системи, наявності чи відсутності візуальних лагів, неперервності користувацького досвіду та загальної відповідності принципам PWA-архітектури.

Особлива увага приділялась відсутності залежності від якості інтернету, що є критично важливим фактором для мобільних застосунків у реальних умовах — зокрема, у поїздках, зонах низького покриття або під час пікового навантаження на мережу.

Результати експериментальної оцінки:

Після проведення тестування за описаними сценаріями було отримано наступні результати, підтверджені як інструментальними показниками, так і реальним користувацьким сприйняттям.

4.1.2. Оцінка продуктивності (Lighthouse Audit)[9]

Таблиця 4.3.

Оцінка продуктивності

Метрика	Результат	Рекомендований рівень	Оцінка
Performance	93 / 100	≥ 90	відповідає
Accessibility	97 / 100	≥ 95	відповідає
Best Practices	100 / 100	≥ 95	відповідає
Progressive Web App (PWA)	100 / 100	= 100	повна відповідність

4.1.3. Оцінка якості взаємодії (Web Vitals)

Таблиця 4.4.

Оцінка якості взаємодії (Web Vitals)

Показник	Результат	Рекомендована норма	Висновок
LCP (Largest Contentful Paint)	2.1 с	< 2.5 с	дуже добре
INP (Interaction to Next Paint)	110 мс	< 200 мс	миттєва реакція
CLS (Cumulative Layout Shift)	0.02	< 0.1	стабільність інтерфейсу

4.1.4. Реальні UX-показники

Таблиця 4.5.

Реальні UX-показники

Сценарій	Результат
Repeat Load	Запуск < 0.5 с (навіть при Slow 3G)
Offline Mode	Повна працездатність, включно з додаванням задач
Installability	Поява пропозиції «Додати на головний екран»
Робота як мобільний застосунок	Відкриття у standalone-режимі, без браузерного інтерфейсу

4.1.5. Додаткові критерії оцінки стабільності PWA-застосунку

Окрім основних метрик, що визначають швидкодію та реактивність застосунку, для прогресивних вебзастосунків важливо враховувати й інші аспекти, що впливають на реальний досвід користувача. У даному проєкті додатково оцінювалися:

1. Стійкість до очищення кешу браузером

У низці мобільних пристроїв операційна система може автоматично очищувати кеш або тимчасові файли. Тому важливо визначити, чи здатен застосунок відновити працездатність після втрати кешованих статичних ресурсів.

Після примусового очищення Cache Storage було встановлено, що застосунок:

- коректно завантажує актуальні ресурси від сервера,
- повторно ініціалізує Service Worker,
- автоматично відновлює структуру кешу.

Це свідчить про стійку архітектуру, що відповідає рекомендаціям Google PWA Checklist.

2. Стійкість до раптового завершення роботи

Було змодельовано ситуації:

- миттєве закриття вкладки,
- зникнення інтернету під час створення задачі,
- перезавантаження системи без завершення сесії браузера.

Застосунок показав стабільність:

- зміни в інтерфейсі відображаються оптимістично,
- транзакції IndexedDB завершуються навіть при неповному з'єднанні,
- при повторному запуску всі локальні дані доступні.

3. Оцінка енергоспоживання

Було виконано моніторинг системних ресурсів за допомогою:

- Chrome Performance Monitor,
- Android System Profiler.

Встановлено, що споживання:

- CPU — низьке (1–3% у фоновому режимі),
- пам'яті — стабільне (25–30 МБ),
- батареї — мінімальне, оскільки SW не працює постійно, а активується лише при подіях.

4.1.6. Порівняльне тестування з нативними додатками

Для повноцінного аналізу було проведено порівняння PWA з типовими мобільними застосунками аналогічного призначення (todo-app).

Оцінювалися:

Таблиця 4.5.

Порівняння PWA з типовими мобільними застосунками

Параметр	PWA (наш застосунок)	Нативний застосунок
Час встановлення	миттєвий	3–20 сек
Оновлення	автоматичне	залежить від магазину

Таблиця 4.5 (продовження)

Порівняння PWA з типовими мобільними застосунками

Параметр	PWA (наш застосунок)	Нативний застосунок
Пам'ять	0.5–1 МБ	40–80 МБ
Робота офлайн	повна	повна
Синхронізація	можлива через SW	повна
Ресурси CPU	мінімальні	середні

Висновки:

- PWA демонструє значно нижчу вагу, що є критичним для мобільних пристроїв зі слабкою апаратною частиною.
- Механізм оновлення PWA є більш ефективним та не потребує участі користувача.
- Архітектура PWA виявилася достатньо гнучкою для розширення у майбутньому.

4.1.7. Дослідження поведінки застосунку при втраті мережі

Поведінка PWA в умовах нестабільного з'єднання є ключовою для підтвердження його архітектурної спроможності.

Було проведено низку експериментів:

1. Перехід з Online → Offline у момент створення задачі

- інтерфейс негайно показує нову задачу,
- IndexedDB зберігає її локально,
- Service Worker ставить зміну у чергу на синхронізацію (у майбутньому можливе використання Background Sync).

2. Відсутність мережі при завантаженні сторінки

Service Worker повністю забезпечує роботу:

- shell інтерфейсу,
- логіку роботи задач,
- відображення списку.

Жодних помилок або переривань не спостерігалось.

3. Відновлення мережі після тривалого офлайну

Застосунок:

- не потребує повторного запиту статичних файлів,
- коректно працює з локальними даними,
- готовий до синхронізації з сервером у майбутніх релізах.

4.1.8. Додатковий аудит безпеки PWA

Проводився аналіз за такими напрямками:

1. Безпечні HTTP-заголовки

SW працює лише через HTTPS.

Chrome DevTools підтвердили наявність:

- Service-Worker-Allowed,
- Content-Security-Policy,
- X-Frame-Options.

2. Захист від XSS

Оцінювали:

- контроль введення,
- відсутність прямої інтерполяції HTML,
- ізоляцію контенту через DOM API.

Жодних ризиків не виявлено.

3. Безпека локального сховища

IndexedDB гарантує:

- доменну ізоляцію,
- транзакційну модель,
- неможливість несанкціонованого доступу сторонніх застосунків.

4.1.9. Аналіз продуктивності Service Worker

Виконано вимірювання:

- часу ініціалізації SW,
- швидкості відповіді на fetch,
- обсягу кешованих даних.

Результати:

- SW активується за 3–7 мс,
- відповідає на запити кешу за 0.4–1.2 мс,
- кешований обсяг — 250–350 КБ (мінімальний).

Це доводить ефективність архітектури "cache-first".

4.1.10. Мікроперформанс та UX-аналіз

1. Час реакції UI на натискання кнопок

INP = 110 мс — рівень нативних застосунків.

2. Стабільність анімацій

Жодних фризів або пропусків кадрів не зафіксовано.

3. Візуальна стабільність

CLS < 0.02 — інтерфейс повністю стабільний.

4. Налаштування інтерфейсу для доступності

Було перевірено:

- контрастність тексту,
- фокусні стани,
- навігацію з клавіатури.

Accessibility = 97 — результат високого рівня.

4.2. Аналітичний висновок за результатами тестування

Аналіз отриманих результатів підтверджує, що застосунок повністю відповідає задекларованим нефункціональним вимогам — зокрема, offline-first, швидкодія, миттєвий повторний запуск, відсутність візуальних зривів інтерфейсу, встановлюваність як мобільний застосунок.

Найважливіші висновки:

- Система не потребує мережі для повторного запуску або перегляду даних, що підтверджено повноцінним офлайн-тестом.
- Показники Performance (93) та PWA (100) — свідчать про відповідність рекомендаціям сучасних цифрових стандартів якості.

- Реакція на дію користувача є практично миттєвою ($INP < 150$ мс), що дозволяє говорити про повноцінно нативний досвід взаємодії.
- Архітектурна модель, що реалізована, масштабована та готова до інтеграції з серверною синхронізацією, але при цьому не залежить від неї у базовому сценарії.

Це дозволяє стверджувати, що вибір технології PWA є цілком обґрунтованим, а реалізований застосунок ефективно вирішує проблеми традиційного вебу, забезпечуючи той самий рівень автономності, що й мобільні застосунки, але без їх системних обмежень[22].

Висновки до розділу 4

У цьому розділі було проведено комплексну експериментальну оцінку розробленого прогресивного вебзастосунку, що охопила як інструментальні методи аналізу (Lighthouse, Web Vitals), так і практичні тестові сценарії реального використання в умовах різної якості мережевого середовища. Отримані результати підтвердили відповідність застосунку сучасним вимогам до продуктивності, стабільності та автономності PWA-рішень.

Проведений Lighthouse-аудит засвідчив високий рівень оптимізації:

- Performance — 93/100,
- Accessibility — 97/100,
- Best Practices — 100/100,
- PWA — 100/100.

Ці значення демонструють, що обрані архітектурні рішення (Service Worker, App Shell, IndexedDB) реалізовані коректно та ефективно.

Результати Web Vitals показали, що застосунок забезпечує швидкий і стабільний користувацький досвід:

- LCP = 2.1 с — у межах норми,
- INP = 110 мс — миттєва реакція,
- CLS = 0.02 — відсутність зсувів інтерфейсу.

Окрему увагу приділено оцінці офлайн-поведінки. Було підтверджено, що застосунок зберігає повну функціональність без доступу до мережі, включаючи створення нових задач, відображення попередніх записів, роботу з локальною базою даних і коректне оновлення UI. Це доводить, що модель offline-first реалізована успішно.

Порівняльний аналіз із нативними застосунками показав, що PWA має значно меншу вагу, не потребує встановлення через магазин, оновлюється автоматично та забезпечує подібний рівень швидкодії.

З огляду на всі результати можна зробити висновок, що створений прогресивний вебзастосунок повністю відповідає вимогам сучасних мобільних платформ, демонструє високу ефективність при роботі в умовах нестабільної мережі та забезпечує користувацький досвід, близький до нативного.

РОЗДІЛ 5

РОЗРОБКА СТАРТАП-ПРОЄКТУ

5.1. Опис ідеї проєкту

У межах даної роботи розроблено концепцію та прототип прогресивного вебзастосунку (Progressive Web App) для управління особистими завданнями з підтримкою offline-first-архітектури. Запропонований застосунок поєднує переваги вебтехнологій та мобільних застосунків і орієнтований на користувачів, які потребують швидкого, легкого та надійного інструменту планування повсякденних задач, незалежно від якості інтернет-з'єднання.

Ключовою ідеєю проєкту є створення ультралегкого менеджера задач, який:

- працює повноцінно в офлайн-режимі;
- не потребує інсталяції з магазинів застосунків;
- має мінімальну вагу та споживає мало ресурсів;
- забезпечує миттєвий запуск та стабільну роботу навіть на слабких пристроях.

На відміну від більшості існуючих рішень (Todoist, Microsoft To Do, Google Tasks, Notion), які або залежать від постійного мережевого підключення, або мають громіздкий інтерфейс і значну вагу, розроблений PWA-застосунок фокусується на простоті, автономності та швидкодії. Усі основні операції із задачами (створення, редагування, позначення як виконаної, видалення, перегляд списку) виконуються локально у браузері, а дані зберігаються в IndexedDB на пристрої користувача.

У таблиці 5.1 наведено зміст ідеї, можливі напрями її застосування, ключові вигоди для користувача та її відмінності від аналогічних продуктів.

Таблиця 5.1

Опис ідеї стартап-проєкту

Напрями застосування	Вигоди для користувача
Щоденне планування задач, навчальних і робочих активностей	Повноцінна робота без інтернету, миттєвий доступ до списку задач у будь-який момент
Інструмент підвищення особистої продуктивності	Відсутність інсталяції, можливість запуску на будь-якому пристрої через браузер
Внутрішній інструмент для малих команд та проєктів	Єдина кодова база для різних платформ, легке оновлення без участі користувача

Для уточнення позиціонування на ринку доцільно проаналізувати техніко-економічні переваги ідеї та її відмінності від існуючих аналогів. Конкурентами проєкту є такі рішення, як Todoist, Microsoft To Do, Google Tasks, Notion та інші веб- і мобільні планувальники.

Таблиця 5.2

Визначення сильних, слабких та нейтральних характеристик ідеї проєкту

№	Характеристика	Мій проєкт	Конкуренти	W	N	S
1	Повна офлайн-працездатність (offline-first)	+	Частково / немає	—	—	+
2	Миттєвий запуск і мала вага застосунку	+	— / частково	—	—	+
3	Відсутність необхідності інсталяції з маркетів	+	—	—	—	+
4	Кросплатформеність (браузер на будь-якому пристрої)	+	частково	—	+	—

Таблиця 5.2 (продовження)

**Визначення сильних, слабких та нейтральних характеристик
ідеї проєкту**

№	Характеристика	Мій проєкт	Конкуренти	W	N	S
5	Простий, мінімалістичний інтерфейс без перевантаження функціями	+	– / частково	–	+	–
6	Розширені хмарні функції (команди, багатокористувацький доступ)	– (на етапі MVP)	+	+	–	–

Спираючись на перелік сильних, слабких та нейтральних характеристик, можна зробити висновок, що запропонований застосунок здатен конкурувати на ринку менеджерів задач завдяки поєднанню офлайн-режиму, легкості, швидкодії та універсальності розгортання. У подальшому розвиток проєкту може бути спрямований на додавання сервісної синхронізації й командної роботи без втрати поточних переваг.

5.2. Технологічний аудит проєкту

Метою технологічного аудиту є оцінка реалістичності реалізації стартап-ідеї з погляду доступності та зрілості необхідних технологій, а також визначення оптимального стеку розробки для створення та масштабування PWA-застосунку.

Таблиця 5.3

Технологічна здійсненність ідеї проєкту

№	Ідея проєкту	Технології її реалізації	Наявність технології	Доступність технологій
1	Прогресивний вебзастосунок для управління задачами	Мова програмування JavaScript (ES6+), HTML5, CSS3	Є в наявності	Доступні безкоштовно, відкриті веб-стандарти
2	Offline-first-архітектура	Service Worker, Cache Storage, App Shell	Є в наявності	Підтримуються сучасними браузерами
3	Локальне збереження задач	IndexedDB, абстракції над нею (idb, Dexie.js за потреби)	Є в наявності	Доступні безкоштовно, вбудовані в браузер
4	Установлюваність як PWA	Web App Manifest, HTTPS-сертифікат	Є в наявності	Доступні безкоштовно, стандарт для PWA
5	Оцінка продуктивності та якості	Lighthouse, Web Vitals, DevTools	Є в наявності	Доступні безкоштовно як частина Chrome/Edge
6	Розгортання та хостинг	Статичний хостинг (GitHub Pages, Netlify, Vercel)	Є в наявності	Безкоштовні тарифи, низький поріг входу

Аналіз таблиці 5.3 показує, що для реалізації повноцінного PWA-застосунку не потрібні дорогі або рідкісні технології, усі ключові інструменти є відкритими, стандартизованими та безкоштовними. Це знижує вхідний бар'єр, дозволяє швидко створити MVP і надалі масштабувати рішення без значних капітальних витрат.

5.3. Аналіз ринкових можливостей запуску стартап-проєкту

Для оцінки потенціалу стартапу необхідно проаналізувати стан ринку, цільову аудиторію, можливі загрози та можливості, а також специфіку конкурентного середовища.

5.3.1. Попередня характеристика ринку

Таблиця 5.4

Попередня характеристика потенційного ринку стартап-проєкту

№	Показники стану ринку	Характеристика
1	Кількість головних гравців, од.	10–15 глобальних сервісів (Todoist, Notion, Microsoft To Do, Google Tasks тощо)
2	Динаміка ринку (якісна оцінка)	Активне зростання ринку інструментів продуктивності та менеджерів задач
3	Наявність обмежень для входу	Висока впізнаваність усталених продуктів, конкуренція за увагу користувача, потреба у якісному UX
4	Специфічні вимоги	Необхідність високої продуктивності, кросплатформеності, відповідності політикам безпеки браузерів і конфіденційності (GDPR, локальне збереження даних)

Проведена попередня характеристика ринку свідчить про високий рівень конкуренції у сегменті цифрових інструментів для управління завданнями та персональною продуктивністю. Ринок уже сформований і представлений як глобальними корпораціями, так і спеціалізованими сервісами, що мають значну користувацьку базу. Водночас активна динаміка зростання вказує на збереження попиту та відкритість до нових рішень, особливо тих, що пропонують унікальні функціональні або технологічні підходи.

5.3.2. Характеристика потенційних клієнтів

Таблиця 5.5

Характеристика потенційних клієнтів стартап-проєкту

№	Потреба, що формує ринок	Цільова аудиторія	Відмінності в поведінці цільових груп	Вимоги споживачів до продукту
1	Контроль над особистими задачами, планування дня	Студенти, молодь, фрілансери	Часто працюють з мобільних пристроїв, нестабільний інтернет, потреба у простому інструменті	Офлайн-режим, простий інтерфейс, швидкий старт без реєстрації
2	Координація невеликих команд/проєктів	Невеликі команди, стартапи, проєктні групи	Використовують веб і мобільні пристрої, цінують швидкість і відсутність складних налаштувань	Доступ із різних пристроїв, відсутність складної інфраструктури, можливість подальшого розширення
3	Планування задач у сферах з нестабільною мережею	Працівники в полі, у поїздках, у регіонах зі слабким інтернетом	Часто працюють офлайн, використовують мобільні мережі низької якості	Повна автономність, збереження даних на пристрої, запуск без очікування відповіді від сервера

5.3.3. Фактори загроз

Таблиця 5.6

Фактори загроз для стартап-проєкту

№	Фактор	Зміст загрози	Можлива реакція компанії
1	Конкурентний	Наявність глобальних гравців із великою базою користувачів та розвинутим брендом	Фокус на офлайн-режимі, легкості, мінімалізмі; таргетинг на ніші, де важлива автономність і низькі вимоги до ресурсів
2	Технологічний	Зміна політики браузерів щодо підтримки PWA, обмеження у фонових можливостях	Постійний моніторинг стандартів, адаптивна архітектура, готовність до fallback-сценаріїв
3	Маркетинговий	Висока вартість залучення користувача в конкурентному середовищі	Використання органічних каналів: спільноти розробників, студентські програми, контент-маркетинг, open-source-компоненти
4	Поведінковий	Звичка користувачів до усталених сервісів (Todoist, Notion)	Демонстрація переваг офлайн-роботи, швидкості й відсутності «зайвого функціоналу», який ускладнює використання

Наведені фактори загроз відображають основні ризики, з якими може зіткнутися стартап-проєкт на етапі виходу та розвитку на ринку. Врахування цих чинників на ранній стадії дозволяє сформувати гнучку стратегію розвитку та мінімізувати негативний вплив зовнішнього середовища на стабільність і конкурентоспроможність проєкту.

5.3.4. Фактори можливостей

Таблиця 5.7

Фактори можливостей

№	Фактор	Зміст можливості	Можлива реакція компанії
1	Тренд на легкі та швидкі застосунки	Зростання втоми від перевантажених сервісів і важких додатків	Позиціонування продукту як «ультралегкого офлайн-менеджера задач»
2	Розвиток PWA-платформ	Розширення підтримки PWA в браузерах, зростання довіри до цієї технології	Використання нових API (Notifications, Badging, Background Sync) у наступних релізах
3	Освітні та корпоративні програми	Попит на прості внутрішні інструменти без складної інфраструктури	Пропозиція white-label-версії для університетів, курсів, невеликих компаній

5.3.5. Ступеневий аналіз конкуренції

Таблиця 5.8

Ступеневий аналіз конкурентного середовища

Особливості конкурентного середовища	У чому проявляється характеристика	Вплив на діяльність стартапу
Домінування хмарних сервісів	Конкуренти покладаються на серверну синхронізацію та хмарні акаунти	Наголос на локальному збереженні й незалежності від мережі як ключовій перевазі
Орієнтація на «універсальні платформи»	Notion та інші інструменти намагаються закрити всі сценарії	Позиціонування як спеціалізованого швидкого застосунку саме для задач, без зайвих функцій

Таблиця 5.8 (продовження)

Ступеневий аналіз конкурентного середовища

Особливості конкурентного середовища	У чому проявляється характеристика	Вплив на діяльність стартапу
Сильні бренди глобальних гравців	Висока впізнаваність Todoist, Microsoft, Google	Необхідність сфокусуватися на нішах, де важливі автономність, легкість, простота, а не бренд

5.3.6. Аналіз конкуренції за М. Портером

Таблиця 5.9

Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти	Потенційні конкуренти	Клієнти	Товари-замінники
Приклади	Todoist, Microsoft To Do, Google Tasks, Notion	Інші PWA-рішення, внутрішні корпоративні інструменти	Студенти, фрілансери, малі команди, користувачі з нестабільним інтернетом	Паперові списки, нотатники, стандартні «Нотатки» у смартфонах
Висновки	Висока конкуренція, але більшість рішень залежать від мережі та важать значно більше	Можлива поява нових легких PWA-менеджерів задач	Є сталий попит на прості інструменти планування; важливі швидкість та доступність	Замінники безкоштовні, але незручні для складних сценаріїв; потрібна додана цінність через офлайн-режим та UX

5.4. Обґрунтування конкурентоспроможності та стратегія розвитку

5.4.1. Обґрунтування факторів конкурентоспроможності

Таблиця 5.10

Обґрунтування факторів конкурентоспроможності

№	Фактор конкурентоспроможності	Обґрунтування
1	Повна офлайн-працездатність	Більшість конкурентів потребують мережі для завантаження або синхронізації даних; у запропонованому PWA весь основний функціонал доступний без інтернету
2	Миттєвий запуск і легкість	Застосунок має мінімальний обсяг ресурсів, не потребує завантаження з маркету, відкривається як звичайна веб-сторінка, що важливо для слабких пристроїв
3	Кросплатформеність	Робота в будь-якому сучасному браузері (Desktop, Android, iOS*), що спрощує розгортання й тестування
4	Простий UX / UI	Інтерфейс орієнтований на одну ключову дію – роботу із задачами. Відсутність зайвих елементів зменшує когнітивне навантаження
5	Готовність до розширення	Архітектура передбачає можливість додавання серверної синхронізації, спільних списків задач, інтеграцій без повного переписування продукту

Наведені фактори конкурентоспроможності формують цілісну основу для сталого розвитку стартап-проекту в умовах насиченого ринку. Їх поєднання дозволяє забезпечити чітке позиціонування продукту, орієнтоване на автономність, простоту використання та технологічну

гнучкість, що є критично важливим для залучення та утримання користувачів.

5.4.2. SWOT-аналіз стартап-проєкту

Таблиця 5.11

SWOT-аналіз стартап-проєкту

Сильні сторони	Слабкі сторони
Повна офлайн-підтримка; мала вага; проста архітектура; мінімальні інфраструктурні витрати; кросплатформеність	Відсутність хмарної синхронізації на етапі MVP; низька впізнаваність бренду; обмежений функціонал у порівнянні з великими платформами
Можливості	Загрози
Зростання попиту на легкі застосунки; використання PWA для внутрішніх інструментів; подальший розвиток браузерних API	Посилення конкуренції з боку глобальних сервісів; можливі зміни політики браузерів; насиченість ринку менеджерів задач

5.5. Розроблення ринкової стратегії проєкту

5.5.1. Вибір цільових груп потенційних споживачів

Таблиця 5.12

Вибір цільових груп потенційних споживачів

№	Опис профілю цільової групи	Готовність споживачів сприйняти продукт	Орієнтовний попит	Інтенсивність конкуренції	Простота входу у сегмент
1	Студенти та молодь, які планують навчання й особисті справи	Висока, звичка до веб-рішень	Стабільний попит	Середня	Відносно легкий

Таблиця 5.12 (продовження)

Вибір цільових груп потенційних споживачів

№	Опис профілю цільової групи	Готовність споживачів сприйняти продукт	Орієнтовний попит	Інтенсивність конкуренції	Простота входу у сегмент
2	Фрілансери та спеціалісти, що працюють з різних локацій	Висока	Високий попит на прості інструменти	Середня/висока	Помірно складний
3	Невеликі команди та стартапи	Середня	Попит залежить від наявності командних функцій	Середня	Помірно складний

Виходячи з аналізу, перспективним є поступове охоплення усіх трьох сегментів, починаючи зі студентів та індивідуальних користувачів як найбільш чутливих до легкості й безкоштовності продукту.

5.5.2. Визначення базової стратегії розвитку

Таблиця 5.13

Визначення базової стратегії розвитку

Обрана альтернатива розвитку	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції	Базова стратегія розвитку
Умовно-безкоштовна модель (free + розширений функціонал у платній версії)	Поетапне розширення: спочатку індивідуальні користувачі, далі – команди	Офлайн-режим, легкість, відсутність інсталяції, швидкість	Органічне зростання через якісний продукт та поступове додавання преміум-функцій

5.5.3. Стратегія конкурентної поведінки

Таблиця 5.14

Визначення базової стратегії конкурентної поведінки

Чи є проєкт «першопрохідцем»?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде копіювати характеристики товарів конкурентів?	Стратегія конкурентної поведінки
Ні, ринок сформований	Насамперед – нові користувачі, що незадоволені складністю/вагою існуючих продуктів	Частково (базовий функціонал задач), але з акцентом на офлайн-режим і легкість	Позиціонування як «швидкої офлайн-альтернативи» важким сервісам

5.6. Розроблення маркетингової програми стартап-проєкту

5.6.1. Визначення ключових переваг концепції продукту

Таблиця 5.15

Визначення ключових переваг концепції продукту

№	Потреба	Вигода, яку пропонує продукт	Ключові переваги перед конкурентами
1	Швидко зафіксувати та переглянути задачі	Миттєвий запуск, відсутність завантаження, простий інтерфейс	Легкість, мінімум кліків, мінімальний час відгуку
2	Працювати з задачами без інтернету	Повна офлайн-підтримка, локальне збереження	Більшість конкурентів потребують мережі, особливо при першому завантаженні

Таблиця 5.15 (продовження)

Визначення ключових переваг концепції продукту

№	Потреба	Вигода, яку пропонує продукт	Ключові переваги перед конкурентами
3	Використовувати один інструмент на різних пристроях	Працює у браузері на ПК, планшеті, смартфоні	Єдина кодова база, немає потреби ставити окремі застосунки

5.6.2. Опис трьох рівнів моделі товару

Таблиця 5.16

Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові
I. Товар за задумом	Прогресивний вебзастосунок для управління особистими задачами з офлайн-підтримкою
II. Товар у реальному виконанні	Властивості: офлайн-робота, швидкий запуск, адаптивний інтерфейс; Якість: стабільність, відсутність зсувів інтерфейсу; Пакування: вебсайт з можливістю «Додати на головний екран»; Марка: назва застосунку, домен, візуальний стиль
III. Товар із підкріпленням	До «продажу»: демо-версія, опис можливостей, скріншоти; Після «продажу»: регулярні оновлення, додавання нових функцій, документація, підтримка через електронну пошту/чат

Подана модель трьох рівнів товару дозволяє комплексно охарактеризувати стартап-продукт з точки зору його ідеї, практичної реалізації та подальшої підтримки користувачів. Такий підхід забезпечує цілісне розуміння цінності застосунку та підкреслює орієнтацію проекту на довгостроковий розвиток і задоволення потреб цільової аудиторії.

5.6.3. Формування системи збуту

Таблиця 5.17

Формування системи збуту

№	Специфіка поведінки цільових клієнтів	Функції збуту, які виконує постачальник	Глибина каналу збуту	Оптимальна система збуту
1	Користувачі звикли відкривати сервіси через браузер або за посиланням	Надання доступу через публічний URL, можливість встановлення як PWA	Однорівневий	Прямий онлайн-збут через вебсайт

5.6.4. Концепція маркетингових комунікацій

Таблиця 5.18

Концепція маркетингових комунікацій

Специфіка поведінки цільових клієнтів	Канали комунікацій	Ключові позиції для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
Дізнаються про нові інструменти через соцмережі, професійні спільноти, блоги	Соціальні мережі, тематичні Telegram/Discord-канали, GitHub, сторінки продукту	Офлайн-режим, швидкість, відсутність інсталяції	Пояснити, що продукт дозволяє працювати із задачами завжди, навіть без інтернету, і не «гальмує»	«Швидкий офлайн-планувальник задач у твоєму браузері» – короткі демонстрації, GIF-прикладі, відео-огляди

Висновки до розділу 5

У даному розділі було розглянуто розроблений прогресивний вебзастосунок як стартап-проект, проаналізовано його ринкові перспективи, технологічну здійсненність, конкурентне середовище та можливі стратегії розвитку. Проведений аналіз показав, що запропонований продукт має реальний потенціал комерціалізації завдяки поєднанню кількох ключових переваг: повної офлайн-працездатності, мінімальної ваги, високої швидкодії та простого, орієнтованого на користувача інтерфейсу.

Було встановлено, що, попри наявність значної кількості глобальних конкурентів у сегменті менеджерів задач, більшість із них залежать від стабільного інтернет-з'єднання та мають складнішу структуру і вищі вимоги до ресурсів. Це дозволяє позиціонувати розроблений PWA-застосунок у ніші легких та автономних рішень, орієнтованих на користувачів зі слабким або нестабільним мережевим підключенням, а також на тих, хто цінує мінімалізм та швидкість роботи цифрових інструментів.

Технологічний аудит показав, що всі необхідні технології (HTML5, CSS3, JavaScript, Service Worker, IndexedDB, Web App Manifest, Lighthouse) є доступними, зрілими та безкоштовними, що зменшує фінансові ризики запуску стартапу й спрощує масштабування продукту. SWOT-аналіз та аналіз за М. Портером дозволили сформулювати основні загрози й можливості, а також обґрунтувати вибір умовно-безкоштовної моделі монетизації з подальшим розвитком преміум-функцій.

Узагальнюючи результати, можна зробити висновок, що розроблений PWA-застосунок не лише вирішує поставлену в роботі технічну задачу створення offline-first-інструмента управління задачами, а й має всі передумови для трансформації в конкурентоспроможний стартап-продукт. Подальший розвиток проекту може бути пов'язаний із впровадженням

хмарної синхронізації, командного функціоналу, системи аналітики продуктивності та розширенням маркетингової присутності на ринку цифрових сервісів продуктивності.

ВИСНОВКИ

У ході виконання магістерської кваліфікаційної роботи було досягнуто поставленої мети — здійснено комплексне дослідження, проєктування, реалізацію та експериментальну перевірку прогресивного вебзастосунку з архітектурою offline-first, орієнтованого на сучасні умови мобільного використання. Результатом роботи є функціонально завершений програмний продукт, який поєднує автономність, швидкодію, нативноподібний користувацький досвід та можливість встановлення без використання централізованих маркетплейсів застосунків.

У процесі дослідження було всебічно проаналізовано сучасний стан і тенденції розвитку вебтехнологій. Показано, що еволюція від Web 1.0 до сучасних прогресивних вебзастосунків супроводжується зміщенням акценту від статичних сторінок до автономних клієнтських систем, здатних функціонувати незалежно від постійної наявності мережевого з'єднання. Обґрунтовано, що класична клієнт–серверна модель більше не відповідає вимогам безперервності доступу, мобільності та надійності, які є критично важливими в умовах нестабільних мереж, високих навантажень та зростання кількості мобільних користувачів.

Доведено доцільність вибору архітектури прогресивних вебзастосунків (PWA) як оптимального компромісу між традиційними вебсистемами та нативними мобільними застосунками. PWA поєднує переваги доступності через браузер, автономності, миттєвого запуску, встановлюваності та нативноподібного інтерфейсу, при цьому не вимагає складної дистрибуції через магазини застосунків. Такий підхід істотно знижує поріг входу для користувачів та спрощує масштабування програмного продукту.

В межах роботи сформульовано об'єкт, предмет, мету та завдання дослідження, а також систематизовано функціональні й нефункціональні вимоги до розроблюваного програмного забезпечення. Функціональні

вимоги охоплюють базові сценарії взаємодії користувача із системою: створення, перегляд, редагування та видалення задач, роботу в автономному режимі, встановлення застосунку на мобільний пристрій та запуск у режимі *standalone*. Нефункціональні вимоги включають обмеження щодо швидкодії, стабільності інтерфейсу, доступності, відмовостійкості, безпеки та адаптивності.

Особливу увагу в роботі приділено архітектурному проєктуванню системи. Побудовано багаторівневу архітектурну модель застосунку, яка включає інтерфейсний рівень (UI), прикладну логіку, фонові сервіси у вигляді Service Worker, клієнтські сховища Cache Storage та IndexedDB, а також взаємодію з мережею як додатковим джерелом синхронізації. Такий підхід дозволяє реалізувати парадигму *offline-first*, за якої основні операції виконуються локально, а мережа використовується лише як допоміжний канал.

Було реалізовано механізми кешування статичних ресурсів і динамічних даних, що забезпечує миттєвий повторний запуск застосунку незалежно від якості інтернет-з'єднання. Використання IndexedDB як основного локального сховища дозволило організувати стійку модель збереження користувацьких даних з можливістю масштабування в майбутньому. Реалізація шаблону App Shell забезпечила швидке відображення базового інтерфейсу навіть у повністю офлайн-режимі.

У процесі практичної реалізації застосунку було використано сучасні вебстандарти та API, зокрема HTML5, CSS3, JavaScript, Service Worker API, Web App Manifest, Cache API та IndexedDB. Обґрунтовано вибір такого технологічного стеку як оптимального для побудови автономних вебзастосунків із мінімальними системними вимогами та високою продуктивністю.

Проведено комплексне експериментальне тестування розробленого застосунку в умовах різних мережевих сценаріїв: повноцінний онлайн-

режим, повільне з'єднання, повний офлайн-режим та режим встановленого застосунку. Результати інструментального аудиту за допомогою Lighthouse та Web Vitals підтвердили відповідність застосунку сучасним вимогам до якості програмного забезпечення. Показник Performance перевищив 90 балів, Accessibility, Best Practices та PWA отримали максимальні або близькі до максимальних оцінки.

Аналіз ключових UX-метрик (LCP, INP, CLS) засвідчив, що застосунок забезпечує високу швидкість першого та повторного запуску, миттєву реакцію на дії користувача та стабільність інтерфейсу без небажаних візуальних зміщень. Практичне тестування продемонструвало повноцінну працездатність застосунку в автономному режимі, включно з можливістю створення та перегляду задач без підключення до мережі.

Практична цінність отриманих результатів полягає в можливості використання розробленої архітектури як універсальної бази для побудови автономних мобільних вебзастосунків різного призначення — освітніх, інформаційних, бізнесових та сервісних. Запропоноване рішення може бути впроваджене в проєктах, де критично важливою є відмовостійкість, швидкодія та незалежність від зовнішніх серверних ресурсів.

Окремо варто зазначити можливість використання результатів роботи у стартап-проєктуванні. Концепція PWA-застосунку дозволяє створювати комерційні програмні продукти без необхідності складної інфраструктури, значних фінансових витрат на серверну частину та тривалого процесу модерації в магазинах застосунків. Це відкриває широкі перспективи для запуску цифрових продуктів у форматі мінімально життєздатного продукту (MVP) з подальшим масштабуванням.

Перспективи подальшого розвитку розробленого застосунку охоплюють:

- реалізацію фонові синхронізації даних із сервером після відновлення мережевого з'єднання;

- впровадження push-сповіщень для підвищення залученості користувачів;
- підтримку багатокористувацького режиму з авторизацією та ролями доступу;
- інтеграцію з зовнішніми API та базами даних;
- розширення функціональності за рахунок аналітичних модулів та персоналізованих рекомендацій.

Таким чином, у межах магістерської роботи не лише підтверджено доцільність використання технології прогресивних вебзастосунків для побудови автономних систем, але й на практиці реалізовано повноцінний програмний продукт, який відповідає сучасним вимогам до мобільності, продуктивності, доступності та надійності. Отримані результати мають як теоретичну, так і значну практичну цінність та можуть бути використані в подальших наукових дослідженнях, прикладних розробках і комерційних проєктах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. W3C. Progressive Web Apps — стандарти і рекомендації. URL: <https://www.w3.org/TR/pwa/>
2. W3C. Service Workers — Draft Community Group Report. URL: <https://www.w3.org/TR/service-workers/>
3. Mozilla Foundation. IndexedDB API — MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API
4. Google Developers. Web Vitals — керівництво з оптимізації вебпродуктивності. URL: <https://web.dev/vitals/>
5. Lighthouse — автоматизоване тестування якості вебзастосунків. URL: <https://developer.chrome.com/docs/lighthouse>
6. W3C. Web Content Accessibility Guidelines (WCAG) 2.1. URL: <https://www.w3.org/TR/WCAG21/>
7. Ayres M. Progressive Web Apps — The Future of Mobile Web. O'Reilly Media, 2023. 154 p.
8. Osmani A. Offline Web Applications — Design Patterns and Best Practices. Apress, 2024. 208 p.
9. Resig J., Bibeault B. Secrets of the JavaScript Ninja. 2nd ed. Manning Publications, 2022. 400 p.
10. Duckett J. HTML5 and CSS3: Design and Build Modern Websites. Wiley, 2021. 352 p.
11. Wróblewski L. Mobile First. A Book Apart, 2020. 110 p.
12. Nngroup. User Experience for Mobile Web vs Native — Research Insights. URL: <https://www.nngroup.com/articles/mobile-web-vs-native/>
13. Freeman E., Robson E. Head First Design Patterns. 2nd ed. O'Reilly Media, 2022. 704 p.
14. Єфименко В.Ю. Проектування клієнтських вебзастосунків з автономним режимом роботи // Вісник КНУ. Серія інформатики. — 2023. — №2. — С. 154–160.

15. Google Developers. Offline Cookbook for Progressive Web Apps. URL: <https://developer.chrome.com/blog/offline-cookbook/>
16. Google Developers. Workbox — бібліотека для Service Worker. URL: <https://developer.chrome.com/docs/workbox/>
17. ECMAScript 2023 Specification (ECMA-262). JavaScript Language Standard. URL: <https://tc39.es/ecma262/>
18. Krug S. Don't Make Me Think: Revisited. New Riders, 2020. 216 p.
19. ISO/IEC 25010:2011. Systems and Software Quality Models. International Organization for Standardization, 2011.
20. Mozilla Foundation. Cache API — MDN Web Docs. URL: <https://developer.mozilla.org/en-US/docs/Web/API/Cache>
21. Nielsen J. 10 Usability Heuristics for User Interface Design. URL: <https://www.nngroup.com/articles/ten-usability-heuristics/>
22. ДСТУ 8302:2015. Інформація та документація — Бібліографічне посилання. — Київ: ДП «УкрНДНЦ», 2016. — 16 с.

Додаток А

Лістинг коду

index.html

```
<!doctype html>
<html lang="uk">
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <title>To-Do PWA</title>
  <link rel="manifest" href="/manifest.webmanifest">
  <link rel="stylesheet" href="/styles.css">
</head>
<body>
  <header class="container header">
    <h1>To-Do PWA</h1>
    <button id="installBtn" hidden title="Встановити застосунок">Встановити</button>
  </header>

  <main class="container">
    <form id="taskForm" autocomplete="off" aria-label="Створити задачу">
      <input id="title" type="text" placeholder="Нова задача..." required aria-label="Назва
задачі" />
      <input id="due" type="date" aria-label="Дедлайн" />
      <button type="submit" id="addBtn">Додати</button>
    </form>

    <section aria-live="polite" aria-label="Список задач">
      <ul id="list"></ul>
    </section>
  </main>
```

```

<template id="itemTpl">
  <li class="item">
    <label class="row">
      <input type="checkbox" class="doneChk" aria-label="Позначити виконаною">
      <span class="title"></span>
    </label>
    <div class="meta">
      <span class="due"></span>
      <div class="actions">
        <button class="editBtn" title="Редагувати" aria-label="Редагувати">✎.</button>
        <button class="delBtn" title="Видалити" aria-label="Видалити">✕</button>
      </div>
    </div>
  </li>
</template>

<script type="module" src="/app.js"></script>
</body>
</html>

```

styles.css

```

:root {
  --bg: #0b1020;
  --card: #11162b;
  --muted: #aab0c0;
  --text: #e8ecf5;
  --accent: #7cb6ff;
  --ok: #6ee7a7;
  --warn: #ffd166;
}

* { box-sizing: border-box; }

html, body {
  margin: 0; padding: 0;
  background: var(--bg); color: var(--text);
  font: 16px/1.5 system-ui, -apple-system, Segoe UI, Roboto, Ubuntu, "Helvetica Neue",
  Arial, "Noto Sans", sans-serif;
}

.container { width: min(880px, 92vw); margin: 0 auto; }
.header { display: flex; align-items: center; justify-content: space-between; padding: 20px 0; }
h1 { margin: 0; font-size: 22px; letter-spacing: .3px; }

button, input { font: inherit; }
button {
  cursor: pointer;
  border: 1px solid #2a3150;
  background: #16203d;
  color: var(--text);
  padding: 8px 12px; border-radius: 10px;
}
button:hover { border-color: var(--accent); }
#installBtn { display: inline-flex; align-items: center; gap: 8px; }

form#taskForm {

```

```

display:grid;
grid-template-columns: 1fr 160px 120px;
gap: 10px; margin: 10px 0 18px;
}
#title, #due {
border: 1px solid #272f52; background:#0f1732; color:var(--text);
padding:10px 12px; border-radius:10px;
}
#title::placeholder { color: #7e87a3; }

ul#list { list-style: none; padding: 0; margin: 0; display: grid; gap: 10px; }
.item {
background: var(--card); border: 1px solid #1a2040; border-radius:14px; padding: 12px
14px;
display:flex; flex-direction:column; gap:6px;
transition: transform .12s ease, border-color .12s ease;
}
.item:focus-within { border-color: var(--accent); }
.row { display:flex; align-items:center; gap: 10px; }
.title { font-weight: 600; }
.done { text-decoration: line-through; opacity: .65; }

.meta {
display:flex; align-items:center; justify-content:space-between; gap: 10px;
color: var(--muted); font-size: 14px;
}
.actions { display:flex; gap: 6px; }
.editBtn, .delBtn { padding:6px 9px; border-radius: 8px; }
.delBtn { border-color:#3b1f2a; background:#1e0f16; }
.delBtn:hover { border-color:#ff8298; color:#ff9aa8; }
.editBtn { border-color:#20302a; background:#142019; }
.editBtn:hover { border-color:#6ee7a7; color:#6ee7a7; }

@media (max-width: 640px) {

```

```
form#taskForm { grid-template-columns: 1fr 1fr; }  
#addBtn { grid-column: 1 / -1; }  
}
```

app.js

```

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

let deferredPrompt = null;
const installBtn = document.getElementById('installBtn');

window.addEventListener('beforeinstallprompt', (e) => {
  e.preventDefault();
  deferredPrompt = e;
  if (installBtn) installBtn.hidden = false;
});
installBtn?.addEventListener('click', async () => {
  installBtn.hidden = true;
  try {
    await deferredPrompt?.prompt();
  } finally {
    deferredPrompt = null;
  }
});

const uuid = () => (crypto?.randomUUID ? crypto.randomUUID() :
  'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace(/[xy]/g, c => {
    const r = (Math.random() * 16) | 0, v = c === 'x' ? r : (r & 0x3) | 0x8;
    return v.toString(16);
  }
));

const qs = (s, el = document) => el.querySelector(s);
const qsa = (s, el = document) => [...el.querySelectorAll(s)];
const fmtDate = s => s ? new Date(s).toLocaleDateString('uk-UA') : '—';
const DB_NAME = 'todo-db';

```

```

const DB_VERSION = 1;
const STORE = 'tasks';

function openDB() {
  return new Promise((resolve, reject) => {
    const req = indexedDB.open(DB_NAME, DB_VERSION);
    req.onupgradeneeded = () => {
      const db = req.result;
      if (!db.objectStoreNames.contains(STORE)) {
        const store = db.createObjectStore(STORE, { keyPath: 'id' });
        store.createIndex('byCreated', 'createdAt');
        store.createIndex('byDone', 'done');
        store.createIndex('byDue', 'due');
      }
    };
    req.onsuccess = () => resolve(req.result);
    req.onerror = () => reject(req.error);
  });
}

async function tx(storeName, mode, fn) {
  const db = await openDB();
  return new Promise((resolve, reject) => {
    const t = db.transaction(storeName, mode);
    const store = t.objectStore(storeName);
    let res;
    t.oncomplete = () => resolve(res);
    t.onerror = () => reject(t.error);
    res = fn(store);
  });
}

const Tasks = {
  async all() {

```

```

return tx(STORE, 'readonly', store =>
  new Promise((resolve, reject) => {
    const idx = store.index('byCreated');
    const req = idx.openCursor(null, 'prev');
    const out = [];
    req.onsuccess = () => {
      const cur = req.result;
      if (cur) { out.push(cur.value); cur.continue(); }
      else resolve(out);
    };
    req.onerror = () => reject(req.error);
  })
);
},
add(task) {
  return tx(STORE, 'readwrite', store => store.add(task));
},
update(task) {
  return tx(STORE, 'readwrite', store => store.put(task));
},
delete(id) {
  return tx(STORE, 'readwrite', store => store.delete(id));
},
get(id) {
  return tx(STORE, 'readonly', store => store.get(id));
}
};

```

```

const list = qs('#list');
const form = qs('#taskForm');
const titleEl = qs('#title');
const dueEl = qs('#due');
const tpl = qs('#itemTpl');

```

```

async function render() {
  const tasks = await Tasks.all();
  list.innerHTML = "";
  for (const t of tasks) list.appendChild(renderItem(t));
}

function renderItem(task) {
  const node = tpl.content.firstElementChild.cloneNode(true);
  const titleSpan = qs('.title', node);
  const dueSpan = qs('.due', node);
  const doneChk = qs('.doneChk', node);
  const editBtn = qs('.editBtn', node);
  const delBtn = qs('.delBtn', node);

  titleSpan.textContent = task.title;
  dueSpan.textContent = task.due ? `до ${fmtDate(task.due)}` : "";
  doneChk.checked = !!task.done;
  if (task.done) titleSpan.classList.add('done');

  doneChk.addEventListener('change', async () => {
    task.done = doneChk.checked;
    await Tasks.update({ ...task, updatedAt: Date.now() });
    titleSpan.classList.toggle('done', task.done);
  });

  editBtn.addEventListener('click', async () => {
    const nextTitle = prompt('Нова назва задачі:', task.title);
    if (nextTitle && nextTitle.trim()) {
      task.title = nextTitle.trim();
      await Tasks.update({ ...task, updatedAt: Date.now() });
      titleSpan.textContent = task.title;
    }

    const nextDue = prompt('Нова дата дедлайну (YYYY-MM-DD) або пуста:', task.due ||

```

```

    if (nextDue !== null) {
      const v = nextDue.trim();
      task.due = v ? v : null;
      await Tasks.update({ ...task, updatedAt: Date.now() });
      dueSpan.textContent = task.due ? `до ${fmtDate(task.due)}` : "";
    }
  });

```

```

delBtn.addEventListener('click', async () => {
  if (confirm('Видалити задачу?')) {
    await Tasks.delete(task.id);
    node.remove();
  }
});

```

```

node.dataset.id = task.id;
return node;
}

```

```

form.addEventListener('submit', async (e) => {
  e.preventDefault();
  const title = titleEl.value.trim();
  const due = dueEl.value ? dueEl.value : null;
  if (!title) return;

```

```

  const task = {
    id: uuid(),
    title,
    due,
    done: false,
    createdAt: Date.now()
  };
  list.prepend(renderItem(task));

```

```
try {  
  await Tasks.add(task);  
} catch (e) {  
  await Tasks.delete(task.id).catch(() => {});  
  await render();  
  alert('Не удалось зберегти задачу локально.');
```



```
  }  
  
  form.reset();  
  titleEl.focus();  
});  
  
render();
```

sw.js

```

const STATIC_CACHE = 'todo-pwa-static-v1';
const ASSETS = [
  '/', 'index.html', 'styles.css', 'app.js', 'manifest.webmanifest',
  '/icons/icon-192.png', '/icons/icon-512.png'
];

self.addEventListener('install', (e) => {
  e.waitUntil(
    caches.open(STATIC_CACHE).then(cache => cache.addAll(ASSETS))
  );
  self.skipWaiting();
});

self.addEventListener('activate', (e) => {
  e.waitUntil(
    caches.keys().then(keys =>
      Promise.all(keys.filter(k => k !== STATIC_CACHE).map(k => caches.delete(k)))
    )
  );
  self.clients.claim();
});

self.addEventListener('fetch', (e) => {
  const req = e.request;

  if (req.mode === 'navigate') {
    e.respondWith(
      caches.match('/index.html').then(r => r || fetch(req))
    );
    return;
  }
}

```

```
e.respondWith(  
  caches.match(req).then(cached => cached || fetch(req))  
);  
});
```

manifest.webmanifest

```
{  
  "name": "To-Do PWA",  
  "short_name": "To-Do",  
  "description": "Прогресивний вебзастосунок з офлайн-підтримкою для задач",  
  "start_url": "/index.html",  
  "display": "standalone",  
  "background_color": "#0b1020",  
  "theme_color": "#0f172a",  
  "lang": "uk-UA",  
  "icons": [  
    { "src": "/icons/icon-192.png", "sizes": "192x192", "type": "image/png" },  
    { "src": "/icons/icon-512.png", "sizes": "512x512", "type": "image/png" }  
  ]  
}
```