Специфікація програмного забезпечення

СПЕЦИФІКАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«Дослідження методів оптимізації та архітектурних рішень для підвищення продуктивності та масштабованості застосунків на основі React»

Автор:

ст. гр. ІПЗм-23-1

Філіпенко Анатолій Вадимович

Харків, 2025

**ВСТУП**

* 1. Огляд програмного забезпечення

Сучасні веб-застосунки, побудовані на основі React, з кожним роком стають дедалі складнішими, що зумовлює потребу в ефективному управлінні рендерингом, структурою компонентів та оптимізації завантаження ресурсів. Погано оптимізовані інтерфейси призводять до затримок у завантаженні, надмірного використання мережевих ресурсів і зниження якості взаємодії з користувачем, особливо на слабких пристроях або в умовах повільного інтернету.

Для вирішення цих проблем необхідне програмне забезпечення, яке реалізовує:

* містить продуману універсальну архітектуру;
* ефективні підходи до оптимізації рендерингу компонентів, зокрема за допомогою мемоізації, розбиття коду (code splitting) та lazy loading;
* оптимізацію виводу великих обсягів даних через віртуалізацію списків;
* гнучке керування асинхронними запитами та кешуванням даних за допомогою бібліотеки TanStack Query;
* поєднання SSR, SSG та CSR в рамках фреймворку Next.js для досягнення балансу між швидкістю, актуальністю контенту та SEO;
* можливість оцінки ефективності впроваджених рішень за допомогою метрик Lighthouse (FCP, LCP, TTI).

Розроблений прототип демонструє практичне застосування вищезазначених методів та підходів, дозволяє зменшити час завантаження сторінок, покращити стабільність роботи інтерфейсу при високому навантаженні та є основою для побудови масштабованих і продуктивних веб-застосунків.

* 1. Мета

Метою даної роботи є дослідження сучасних архітектурних підходів і методів оптимізації продуктивності веб-застосунків, створених за допомогою React та Next.js, для підвищення продуктивності та масштабованості застосунків, а також розробка прототипу, що демонструє ефективність впроваджених рішень у контексті покращення швидкодії та масштабованості.

Для досягнення цієї мети необхідно:

* проаналізувати існуючі проблеми продуктивності у React-застосунках, пов’язані з рендерингом, повторними обчисленнями та завантаженням великих обсягів даних;
* дослідити можливості покращення продуктивності за допомогою мемоізації, lazy loading, віртуалізації списків, оптимізації мережевих запитів та SSR/SSG/CSR;
* реалізувати прототип веб-застосунку з використанням фреймворку Next.js, який поєднує різні методи рендерингу та оптимізації;
* провести експериментальне порівняння версій застосунку з і без оптимізацій для оцінки впливу кожного з методів, використовуючи інструменти, такі як Lighthouse.

Досягнення поставленої мети дозволить сформулювати практичні рекомендації щодо побудови продуктивних, масштабованих та зручних для користувача веб-застосунків, що відповідають сучасним вимогам якості та швидкодії.

* 1. Межі

У рамках цього дослідження охоплюються наступні аспекти:

* оптимізація продуктивності React-застосунків за допомогою архітектурних рішень та технік рендерингу;
* використання Next.js для реалізації SSR, SSG та CSR;
* застосування мемоізації, lazy loading, віртуалізації списків та оптимізація мережевих запитів з використанням TanStack Query;
* розробка та тестування прототипу веб-застосунку з метою порівняння продуктивності до та після впроваджених оптимізацій;
* оцінка ефективності методів оптимізації за допомогою інструменту Lighthouse та Web Vitals.

У дослідженні не розглядаються:

* оптимізація серверної логіки поза межами Next.js;
* інтеграція з backend-технологіями поза REST/API рівнем;
* специфічні аспекти SEO, що виходять за рамки рендерингу контенту;
* безпека, аутентифікація або управління доступом у веб-застосунках;

Встановлені межі дозволяють зосередити увагу на фронтенд-оптимізаціях, що безпосередньо впливають на швидкодію, масштабованість та користувацький досвід у сучасних React/Next.js-застосунках.

# **2 ЗАГАЛЬНИЙ ОПИС**

2.1. Перспективи програмного забезпечення

У сучасній веб-розробці зростає потреба у створенні швидких, масштабованих і зручних для користувача застосунків, що можуть ефективно працювати навіть при великому навантаженні. Розробка веб-застосунків із використанням React та Next.js, із чітко продуманою архітектурою та впровадженими методами оптимізації, відкриває значні перспективи для подальшого розвитку подібних рішень у комерційних і корпоративних проєктах.

Запропоновані підходи, такі як мемоізація, динамічне завантаження компонентів, віртуалізація списків та гнучке рендеринг-середовище на базі Next.js дозволяють не лише зменшити навантаження на клієнтську частину, а й покращити показники продуктивності та адаптувати застосунок до різних сценаріїв використання. Це особливо важливо для продуктів із великою кількістю інтерактивного контенту чи динамічних даних.

У майбутньому розроблене рішення може бути розширене за рахунок інтеграції з backend-сервісами для динамічного завантаження даних у реальному часі, впровадження персоналізації контенту на основі поведінки користувачів та покращення механізмів обробки великих обсягів інформації. Також можливе додавання аналітичних інструментів для моніторингу продуктивності та користувацьких дій, що дозволить точніше оцінювати ефективність оптимізацій і приймати обґрунтовані рішення щодо подальшого розвитку продукту.

Таким чином, реалізоване ПЗ має потенціал до масштабування, повторного використання в інших проєктах і подальшого вдосконалення відповідно до сучасних вимог веб-індустрії.

2.2. Функції програмного забезпечення

На поточному етапі реалізації проєкту передбачається, що розроблене програмне забезпечення на основі React і Next.js виконуватиме такі основні функції:

* використання гібридної моделі компонентів у Next.js, що поєднує серверні компоненти для швидкого первинного рендерингу та клієнтські — для реалізації інтерактивної логіки, що забезпечує кращу продуктивність і гнучкість при розробці;
* застосування мемоізації компонентів і значень для зменшення кількості повторних обчислень і зайвих рендерів, що покращує швидкодію інтерфейсу;
* реалізація віртуалізації списків для продуктивного відображення великої кількості елементів без перевантаження DOM;
* інтеграція з TanStack Query для оптимізованої роботи з асинхронними запитами, кешуванням отриманих даних і повторного використання вже завантаженої інформації без надлишкових звернень до API;
* ліниве завантаження компонентів та зображень, що дозволяє зменшити початкове навантаження на браузер та покращити показники First Contentful Paint (FCP) і Largest Contentful Paint (LCP);
* модульна структура архітектури, що дозволяє легко масштабувати застосунок, додавати нові функції або адаптувати його під різні сценарії використання.

Сукупність зазначених функцій забезпечує гнучку, продуктивну та адаптивну основу для створення сучасного веб-застосунку, який відповідає високим вимогам до якості, швидкодії та зручності взаємодії з користувачем.

2.3. Характеристики користувачів

У процесі розробки та оптимізації веб-застосунку важливо враховувати інтереси та потреби всіх зацікавлених сторін, які взаємодіють із системою або приймають рішення щодо її розвитку. Розуміння ролей і очікувань таких користувачів дозволяє ефективніше планувати архітектурні рішення, вибирати відповідні технології та забезпечувати якісну реалізацію проєкту.

Характеристики зацікавлених користувачів наведені у таблиці 2.1.

Таблиця 2.1 – Характеристика зацікавлених користувачів

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Зацікавлена сторона** | **Основна цінність** | **Обов’язки** | **Головна ціль** | **Обмеження** |
| **Фронтенд-розробники (React/Next.js)** | Готові рішення для підвищення продуктивності; модульність та повторне використання | Інтеграція оптимізованих компонентів у кодову базу; налаштування кешу, запитів, мемоізації | Забезпечити швидкодію, плавну роботу інтерфейсу та зменшити обсяг повторного рендерингу | Час на впровадження; сумісність зі спадковим кодом |
| **Архітектори веб-систем** | Масштабованість, ефективність при зростанні навантаження | Вибір підходів SSR/CSR; налаштування структури застосунку; вибір стратегій оптимізації | Побудувати стійку та адаптивну архітектуру | Необхідність узгодження з іншими командними рішеннями |
| **Тестувальники** | Метрики продуктивності, стабільність у різних умовах | Створення кейсів на продуктивність; перевірка навантаження та поведінки після оптимізацій | Виявити проблеми із продуктивністю; оцінити вплив змін на швидкодію | Час на автоматизацію тестів; обмежений доступ до системних логів |
| **Менеджмент / Бізнес-власники** | Покращення користувацького досвіду, зменшення часу відгуку | Прийняття рішень щодо доцільності оптимізацій, планування релізів | Підвищити конкурентоспроможність застосунку через продуктивність | Обмежений бюджет та дедлайни |

2.4. Загальні обмеження

У межах даного проєкту визначено низку обмежень, пов’язаних із середовищем розробки, архітектурою та обсягом реалізованого функціоналу. Рішення базується на Next.js версії 14 у поєднанні з React 18 та архітектурною моделлю App Router. Підтримка старіших версій або проєктів, створених із використанням Pages Router, не передбачена.

Прототип орієнтований на веб-платформу та не включає мобільну адаптацію або інтеграцію з фреймворками типу React Native. Архітектура зосереджена на фронтенд-частині застосунку з використанням клієнтських та серверних компонентів без прямого з’єднання з базами даних або зовнішніми API (замість них застосовано симуляцію даних).

Під час розробки не реалізовувалися механізми аутентифікації, авторизації, маршрутизації для багаторівневих ролей користувачів або засоби зберігання персональних даних. Оцінка продуктивності проводилася в умовах обмеженого середовища (локальний сервер, базові апаратні ресурси), тому результати тестування можуть відрізнятись у хмарному або продакшн-оточенні.

Ці обмеження дозволили зосередити увагу на дослідженні впливу фронтенд-оптимізацій на швидкодію та стабільність інтерфейсу, забезпечивши чіткий фокус і контрольоване тестове середовище.

2.5. Припущення і залежності

У процесі реалізації проєкту зроблено кілька ключових припущень, необхідних для стабільної роботи оптимізованого прототипу:

* застосунок розгортається в сучасному середовищі, яке підтримує ES-модулі, SSR та React 18 (наприклад, Vercel або локальний сервер із Node.js 18+);
* серверна частина (у разі наявності) забезпечує стабільну відповідь API-запитів і не викликає непередбачуваних затримок, що могли б вплинути на рендеринг або кешування;
* пристрої користувачів мають достатній обсяг оперативної пам’яті та продуктивності для відображення віртуалізованих списків, lazy-завантаження зображень і виконання рендерингу на клієнтському боці;
* усі компоненти застосунку використовують однакову версію бібліотек (React, Next.js, TanStack Query), без конфліктів або сторонніх модифікацій.

Залежності проєкту включають:

* Next.js 15 як основний фреймворк для побудови структури застосунку;
* React 18 для реалізації сучасної логіки рендерингу;
* Lighthouse як інструмент для збору метрик продуктивності;
* Tailwind CSS або інша утилітарна система стилізації для побудови адаптивного інтерфейсу.

# **3 КОНКРЕТНІ ВИМОГИ**

3.1. Вимоги до зовнішніх інтерфейсів

3.1.1. Апаратний інтерфейс

Для стабільної роботи веб-застосунку, побудованого на базі React та Next.js із впровадженими методами оптимізації продуктивності, треба використовувати серверну або локальну платформу з такими мінімальними характеристиками:

* x64-сумісний CPU процесор з не менш ніж 4 ядрами, тактова частота — від 2.5 ГГц;
* оперативна пам’ять, мінімум 4 ГБ, та оптимально 8 ГБ для ефективної роботи під час розробки, тестування та запуску dev-режиму в Next.js;
* SSD-накопичувач.
* мережева інфраструктура: стабільне з’єднання із пропускною здатністю не менше 20 Мбіт/с, особливо під час завантаження великих обсягів даних або виконання API-запитів;
* операційна система: Windows 10/11, macOS 12+ або сучасні Linux-дистрибутиви;
* відкриті порти: при використанні локального сервера Next.js слід забезпечити доступ до вказаного порту та портів, що використовуються API або бекендом.

Ці апаратні вимоги гарантують стабільну роботу під час розробки та тестування застосунку, швидке збірку компонентів, а також коректне виконання SSR/CSR логіки під типовим навантаженням.

3.1.2. Програмний інтерфейс

Програмний інтерфейс реалізується у вигляді веб-застосунку, побудованого на основі бібліотеки React 18 та фреймворку Next.js 14 з використанням архітектури App Router. Інтерфейс забезпечує взаємодію між користувачем, внутрішніми компонентами застосунку та зовнішніми API-запитами, зокрема для отримання або обробки динамічних даних.

У межах проєкту реалізовано гібридний підхід до рендерингу, при якому використовується поєднання серверних (server components) та клієнтських компонентів (client components). Це дозволяє оптимізувати початковий час завантаження, зменшити обсяг JavaScript, що виконується на клієнті, і покращити продуктивність.

Для роботи з асинхронними запитами та кешуванням відповідей використовується бібліотека TanStack Query, яка забезпечує контроль над життєвим циклом запитів, повторне використання даних і автоматичну інвалідацію кешу. Обмін даними з сервером (або мок-сервісом) здійснюється через протокол HTTP(S), запити виконуються через вбудований механізм fetch.

Конфігураційні параметри та взаємодія з інтерфейсом користувача реалізовані безпосередньо у застосунку через інтуїтивно зрозумілий UI. Зокрема, користувач має змогу виконувати пошук за ім’ям, сортувати список користувачів, переглядати картки з детальною інформацією (вік, email, телефон, місто, країна), а також відкривати модальне вікно з історією покупок. Дані завантажуються динамічно, а важкі компоненти (наприклад, вміст модального вікна) підвантажуються ліниво (lazy-load) через динамічний імпорт, що сприяє кращій продуктивності сторінки. Зміна налаштувань, таких як сортування чи пошук, відбувається без перезавантаження сторінки, повністю на клієнтській стороні, з використанням React-стану та обробників подій.

Для моніторингу продуктивності використовуються інструменти Lighthouse та Web Vitals, які надають метрики, що вимірюють швидкодію, доступність і якість користувацького досвіду. Інтерфейс протестовано на сучасних браузерах, підтримка застарілих платформ не реалізована.

Таким чином, програмний інтерфейс забезпечує зручну, модульну і розширювану архітектуру для інтеграції продуктивних рішень у сучасні веб-застосунки без використання зайвих залежностей або складних інструментів конфігурації.

3.1.3. Операції

Система реалізує такі ключові операції, орієнтовані на ефективну роботу користувацького інтерфейсу та динамічну взаємодію з даними:

* візуалізація даних про користувачів у вигляді карток із базовою інформацією (ім’я, вік, email, місто, країна) з підтримкою динамічного сортування;
* фільтрація користувачів у реальному часі за введеним ім’ям через текстове поле пошуку;
* асинхронне завантаження даних про покупки з відображенням у модальному вікні, яке відкривається за потреби (lazy-loaded з використанням dynamic import);
* перемикання між представленнями (список користувачів ↔ останні покупки) з підтримкою базової навігації;
* виведення відформатованих дат та структурованої інформації про обʼєкти (наприклад, дата покупки, модель телефону, user ID);
* зберігання стану пошуку та сортування в межах сесії користувача (на рівні React-компонентів) без повного перезавантаження сторінки.

3.2. Властивості програмного забезпечення

Програмне забезпечення розроблено з використанням актуальних веб-технологій, зокрема React 18, Next.js 15, TypeScript та Tailwind CSS. Вся логіка побудована відповідно до сучасного підходу компонентної архітектури з розділенням серверних і клієнтських компонентів.

3.3. Атрибути програмного забезпечення

Програмне забезпечення, розроблене в межах даного проєкту, має такі ключові атрибути:

* надійність. Система стабільно працює при обробці великої кількості користувацьких компонентів та динамічного контенту. Завдяки ізольованому рендерингу клієнтських та серверних частин мінімізуються збої в роботі інтерфейсу. Компоненти тестуються на коректну роботу з реальними та симульованими даними;
* ефективність. Використання віртуалізації списків, мемоізації, лінивого завантаження компонентів (lazy-loading) та кешування запитів через TanStack Query забезпечує швидкий рендеринг інтерфейсу, зменшення затримок і ефективне використання ресурсів браузера;
* супроводжуваність. Код організований за принципами модульності та розділення відповідальностей. У проектіs використано TypeScript, що підвищує передбачуваність типів і спрощує налагодження. Компоненти мають чітку структуру, а логіка написана згідно з сучасними стандартами React і Next.js. Ведеться контроль змін у системі керування версіями (Git);
* гнучкість. Архітектура дозволяє легко змінювати кількість елементів на сторінці, алгоритми сортування, логіку фільтрації, а також оновлювати зовнішній вигляд або API без потреби в переробці всієї структури. Компоненти легко масштабуються і повторно використовуються в інших частинах застосунку;
* масштабованість. Програмне рішення здатне обробляти велику кількість динамічних даних завдяки асинхронній обробці запитів, кешуванню результатів і оптимізації виводу на рівні рендерингу. Також реалізована підтримка динамічного імпорту важких компонентів лише за потреби.