Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

# КВАЛІФІКАЦІЙНА РОБОТА
# Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження методів рендерінгу користувацького інтерфейсу на базі фреймворків React та Svelte
(тема)

Виконав:
студент (ка) 2 курсу, групи ІПЗМ-22-6

Свиридов В.Е
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-наукова

Керівник проф. Четвериков Г.Г.
(посада, прізвище, ініціали)

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

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти другий (магістерський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми освітньо-наукова програма
Освітня програма Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2024 р.

**ЗАВДАННЯ**
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Свиридову Вадиму Едуардовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Дослідження методів рендерінгу користувацького інтерфейсу на базі фреймворків React та Svelte
Затверджена наказом по університету від 29.03. 2024р. № 250 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 21.06.2024
3. Вихідні дані до роботи офіційні документації фреймворків: статистичні дані з аналітичних платформ, репозиторії з прикладами коду та проектами-демонстраціями.
4. Перелік питань, що потрібно опрацювати в роботі
аналіз предметної галузі, методи вирішення проблеми, адаптація основних стратегій рендерингу та створення комбінованих стратегій для відповідності сучасним вимогам, методика порівняльного аналізу

# КАЛЕНДАРНИЙ ПЛАН

| Номер | Назва етапів роботи | Термін виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Видача завдання | 30.03.2024 | виконано |
| 2 | Аналіз предметної галузі | 05.04.2024 | виконано |
| 3 | Постановка задачі | 10.04.2024 | виконано |
| 4 | Експериментальні дослідження | 12.04 – 20.04.24 | виконано |
| 5 | Аналіз результатів експериментальних досліджень та розробка рекомендацій | 20.04 – 23.04.24 | виконано |
| 6 | Написання та оформлення статті та тез доповіді | 20.03 – 03.04.24 | виконано |
| 7 | Підготовка пояснювальної записки | 01.05 – 26.05.24 | виконано |
| 8 | Підготовка презентації та доповіді | 10.06 – 13.06.24 | виконано |
| 9 | Нормоконтроль | 13.06.24 | виконано |
| 10 | Рецензування | 17.06.24 | виконано |
| 11 | Занесення диплома в електронний архів | 17.06.24 | виконано |
| 12 | Попередній захист | 17.06.24 | виконано |
| 13 | Допуск до захисту у зав. кафедри | 17.06.24 | виконано |

Дата видачі завдання «26» січня 2024 р.

Студент _____ Свиридов В.Е.

(підпис)

Керівник роботи_____ проф. Четвериков Г.Г.

(підпис) (посада, прізвище, ініціали)

## РЕФЕРАТ / ABSTRACT

Пояснювальна записка містить 54 ст., 18 рис., 5 табл., 20 джерел.

АРХІТЕКТУРА, ВЕБ-ФРЕЙМВОРКИ, ПРОДУКТИВНІСТЬ, РЕНДЕРІНГ, CUMULATIVE LAYOUT SHIFT, FIRST INPUT DELAY, LARGEST CONTENTFUL PAINT, REACT, SVELTE, VUE.

Об’єктом дослідження є порівняння методів рендерінгу інтерфейсу користувача в двох фреймворках: React та Svelte. Метою є виявлення переваг і недоліків кожного підходу, а також визначення ситуацій, в яких вони можуть бути найбільш ефективними.

Головною метою цього дослідження є проведення аналізу та порівняння методів рендерінгу користувацького інтерфейсу в двох популярних фреймворках – React та Svelte. Дослідження спрямоване на виявлення особливостей кожного з підходів, їх переваг та недоліків, а також на визначення ситуацій, в яких один фреймворк може бути більш ефективним або зручним для використання в порівнянні з іншим. Результати цього дослідження можуть служити підставою для обґрунтованого вибору фреймворку при розробці веб-додатків з урахуванням конкретних умов та завдань проекту.

У цьому дослідженні використовується комплексний підхід до порівняння методів рендерінгу у фреймворках React та Svelte. По-перше, проводиться аналіз теоретичних аспектів кожного фреймворку, вивчення їхньої архітектури та основних концепцій, пов'язаних із рендерінгом.

ARCHITECTURE, CUMULATIVE LAYOUT SHIFT, FIRST INPUT DELAY, LARGEST CONTENTFUL PAINT, PRODUCTIVITY, REACT, RENDERING, SVELTE, VUE, WEB FRAMEWORKS.

The object of the investigation is the alignment of methods for rendering the backend interface in two frameworks: React and Svelte. The method is to identify the

advantages and disadvantages of the skin approach, as well as the specific situations in which stinks may be most effective.

The main purpose of this research is the analysis and alignment of methods for rendering the client interface in two popular frameworks – React and Svelte. The research is aimed at identifying the peculiarities of skin conditions, their advantages and shortcomings, as well as at certain situations in which one framework may be more effective or easier to achieve in a similar way. . The results of this research can serve as a basis for choosing a framework for the development of web applications based on specific minds and the direction of the project.

This study uses a comprehensive approach to compare rendering methods in React and Svelte frameworks. First, an analysis of the theoretical aspects of each framework, the study of their architecture and the main concepts related to rendering is carried out.

Я, Свиридов Вадим Едуардович, студент гр. ІПЗМ-22-6, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження методів рендерінгу користувацького інтерфейсу на базі фреймворків React та Svelte», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

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

# ЗМІСТ

# ПЕРЕЛІК СКОРОЧЕНЬ

API – Application Programming Interface

CSR – Client-Side Rendering

CSS – Cascading Style Sheets

DOM – Document Object Model

DX – Developer Experience

FCP – First Contentful Paint

FID – First Input Delay

JS – JavaScript

LCP – Largest Contentful Paint

PWA – Progressive Web Application

SEO – Search Engine Optimization

SSR – Server-Side Rendering

TTFB – Time to First Byte

TTI – Time to Interact

UX – User Experience

# ВСТУП

В сучасній веб-розробці вибір оптимального фреймворку для створення інтерфейсу користувача є стратегічно важливим завданням. Дослідження різних фреймворків дозволяє виявити їхні переваги та особливості, що відповідають потребам конкретного проекту.

Ця робота призначена для профундованого аналізу та порівняння методів рендерінгу у двох відомих фреймворках - React та Svelte. Питання продуктивності, ефективності рендерінгу та гнучкості використання становлять основний фокус у вступі.

Підкреслено важливість обрання правильного інструменту для вирішення конкретних завдань у контексті веб-розробки. Огляд основних характеристик React та Svelte розкриє їхні унікальні особливості, які визначають їхню привабливість для розробників.

Дослідження включатиме аналіз ключових метрик продуктивності та стратегій рендерінгу, що дозволить отримати глибше розуміння того, як кожен фреймворк пристосований для вимог сучасної веб-розробки.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

## 1.1 Аналіз предметної області

Розробка фронтенду є ключовим етапом створення веб-додатків, який визначає зовнішній вигляд та взаємодію з користувачем. Цей процес об'єднує три основні технології: HTML, CSS і JavaScript. HTML відповідає за структуру сторінки, CSS - за стилі та макет, а JavaScript - за інтерактивність та динамічну поведінку елементів [1].

Протягом розвитку фронтенду було створено безліч інструментів, спрямованих на поліпшення якості додатків та спрощення роботи розробників. Ці технології не лише надають кращий та конкурентоспроможний функціонал користувачам, але й полегшують та прискорюють роботу розробників.

У сучасному світі всі веб-додатки використовують ряд технологій та інструментів, які розширюють базовий набір HTML, CSS та JavaScript. Вибір інструментів зазвичай робиться досвідченими фахівцями на етапі планування проєкту і базується на вартості, строках, вимогах та цілях. Цей вибір також визначає стратегії для досягнення швидкого завантаження та високої інтерактивності для залучення клієнтів чи врахування мультифункціональності додатка для внутрішніх користувачів компанії.

Сучасні інструменти розробки фронтенду класифікуються за такими основними категоріями:

- фреймворки;
- стратегії рендерингу;
- css-препроцесори;
- збірники модулів.

### 1.1.1 Мета використання фреймворків для розробки фронтенду

Мета використання фреймворків для розробки фронтенду полягає в уніфікації та оптимізації процесу створення веб-інтерфейсів. Ці фреймворки надають структурований набір засобів, які дозволяють розробникам

концентруватися на логіці додатку та взаємодії з користувачем, замість того, щоб витрачати час на повторне вирішення загальних завдань.

Забезпечуючи готові шаблони, модулі та компоненти, фреймворки дозволяють швидше реалізувати стандартні функціональності, такі як навігація, обробка форм, анімації та інші аспекти інтерфейсу. Це не лише спрощує роботу розробників, але й забезпечує єдність структури та дизайну в межах проєкту.

Крім того, фреймворки допомагають вирішувати питання крос-браузерної сумісності, автоматизуючи частину завдань, пов'язаних з різницею в поведінці різних браузерів. Це забезпечує стабільну та однакову роботу веб-додатків на різних платформах та пристроях.

Використання фреймворків у фронтенд-розробці не лише спрощує та прискорює робочий процес, але й забезпечує високу якість та єдність розроблених веб-додатків.

### 1.1.2 Аналіз ролі стратегії рендерингу

Однією з ключових рішень, які приймаються під час планування IT-проєкту з розробкою веб-додатка, є визначення стратегії рендерингу. Це визначає, чи буде веб-додаток будуватися на стороні сервера, клієнта, частково на обох сторонах або ж буде статично зберігатися на сервері. Сторінки веб-сайту можуть оновлюватися при кожному запиті, зберігатися в кеш-пам'яті протягом певного періоду часу або залишатися статичними [2].

Вибір стратегії ґрунтується на бізнес-вимогах, ролі продуктивності веб-сайту, важливості SEO і не повинен залежати від фреймворку, який використовується для розробки. Визначення найбільш підходящої стратегії рендерингу може суттєво вплинути на швидкодію веб-сайту та його продуктивність під час завантаження, при цьому знижуючи витрати на обробку. З іншого боку, невдалий вибір стратегії може призвести до неефективності та великих ресурсних витрат для бізнесу.

Правильно обрана стратегія може значно змінити досвід розробника (DX), особливо коли додаток створюється для команди інженерів або внутрішніх членів

компанії, де швидке завантаження додатка або SEO не є ключовими вимогами. Також це впливає на досвід користувача (UX), особливо якщо фокус зрілеспрямований на кінцевих користувачів.

### 1.1.3 Мета використання CSS-процесора

CSS-процесор використовується для створення більш динамічних та гнучких стилів у порівнянні з традиційним CSS. Його мета полягає в полегшенні написання коду стилів і забезпеченні розширених можливостей для розробників.

Однією з ключових переваг CSS-процесорів є можливість використання змінних, які дозволяють зберігати значення і використовувати їх у різних частинах коду. Це спрощує управління стилями та робить їх більш модульними. Мета використання таких засобів - зробити код більш зрозумілим та легким у підтримці.

CSS-процесори також дозволяють використовувати міксини, які є відмінною можливістю для переиспользования коду. За допомогою міксинів можна організовувати та використовувати схожі стилі в різних частинах проєкту, що сприяє створенню консистентного дизайну та підтримує принцип DRY (Don't Repeat Yourself).

Окрім цього, CSS-процесори дозволяють використовувати вкладені стилі, що полегшує структуру коду та робить його більш читабельним. Вони також підтримують використання операторів, циклів та умов, розширюючи можливості для створення більш гнучких та динамічних стилів, які можуть адаптуватися до різних умов.

Основна мета використання CSS-процесора - це полегшення та покращення розробки стилів, забезпечуючи розширені можливості та ефективні інструменти для створення сучасних та гнучких веб-інтерфейсів.

### 1.1.4 Функція збірників модулів у фронтенд-проєктах.

Роль збірників модулів у фронтенд-проєктах полягає в організації та управлінні модульністю коду. Збірники модулів відповідають за збірку різних

фрагментів програмного коду та їх залежностей у єдиний файл чи набір файлів, що спрощує розробку, підтримку та розгортання веб-додатків.

Ці збірники виконують важливу функцію у забезпеченні ефективного управління залежностями між різними компонентами проєкту. Вони дозволяють розробникам структурувати код, розділяти його на логічні модулі та ефективно керувати ресурсами.

Крім того, збірники модулів допомагають у вирішенні проблеми завантаження, зменшуючи кількість запитів до сервера та покращуючи швидкість завантаження сторінок. Вони впроваджують механізми кешування та мінімізації коду, що сприяє оптимізації продуктивності веб-додатків.

Таким чином, функція збірників модулів у фронтенд-проєктах полягає у впорядкуванні та оптимізації кодової бази, що допомагає забезпечити ефективну та організовану розробку веб-додатків.

## 1.2 Актуальність проблеми

Деякі фреймворки, спрямовані на створення користувацького інтерфейсу, вирішують проблему управління даними "з коробки", однак, протягом тривалого часу для таких програм React вимагали сторонніх рішень. Це призвело до розробки різноманітних бібліотек, які стали своєрідним стандартним вибором для розробників. Основною метою цих бібліотек є організація та керування потоком даних у зовнішніх програмах, незалежно від того, чи написана програма на React чи іншому фреймворку [3].

Також проводиться дослідження методів управління станом у веб-додатках. Наприклад, у "Comparison of web application state management tools" детально розглядаються методи управління станом, такі як NgRx, Ngxs, Redux, Vuex. Аналіз включає п'ять критеріїв: метрики коду, структура рішення, наявність готових реалізацій, підтримка спільноти та вимірювання швидкодії. У рамках цього дослідження також проведено серію тестів для аналізу функціональності кожної бібліотеки.

Зазначено, що деякі з розглянутих бібліотек залежать від конкретного фреймворку і не можуть використовуватися з будь-якою бібліотекою для створення інтерфейсів користувача, наприклад, лише одна бібліотека з розглянутих може бути застосована до додатків, які базуються на React.

У виборі інструментів для створення веб-додатків, переважно вибирається бібліотека для створення користувацького інтерфейсу, а потім вивчаються методи управління станом та інші допоміжні бібліотеки. Таким чином, дослідження бібліотек, які сумісні з конкретною бібліотекою для створення інтерфейсу користувача, може вважатися більш доцільним.

Прикладом такого дослідження є «Comparison of State Management Solutions between Context API and Redux Hook in ReactJS». У цьому дослідженні детально розглядаються та порівнюються вибіркові проекти з відкритим вихідним кодом, використовуючи Redux та Context API як методи управління станом. Аналіз цих двох підходів проводиться з урахуванням їхньої зручності використання, а також вимірюється метрика складності вихідного коду. Дослідження також охоплює використання пам'яті на різних пристроях, що спрямовано на розуміння впливу цих підходів на швидкодію проектів.

Узагальнюючи, в дослідженні розглядаються наступні критерії:

- впровадження;
- відстеження змін;
- встановлення додаткових пакетів;
- складність кодової бази;
- споживання ресурсів;
- швидкість обробки та масштабованість.

Зазначено, що на основі результатів дослідження зроблено висновок, що Redux є найбільш підходящим рішенням для великих проектів, тоді як Context API може бути використано лише у невеликих проектах. Важливо відзначити, що розглянуті критерії та методи не вичерпно відображають повну картину сучасних методів управління станом для односторінкових додатків, розроблених з використанням React.

Враховуючи, що обсяг даних у складних зовнішніх програмах може бути значно великим, неправильний вибір методу керування станом може викликати серйозні проблеми з продуктивністю, особливо на обладнанні з обмеженими ресурсами, такому як смартфони, планшети та застарілі комп'ютери.

## 1.3 Мета та завдання дослідження

Основною метою роботи є аналіз та порівняння методів рендерінгу у фреймворках React та Svelte для створення користувацького інтерфейсу в веб-додатках. Робота спрямована на вивчення особливостей обох підходів, їх ефективності та впливу на продуктивність додатків.

Завдання дослідження:

а) теоретичний огляд:

1) розгляд основних принципів роботи фреймворків React та Svelte;
2) аналіз методів рендерінгу, які використовуються в обох фреймворках;

б) порівняльний аналіз:

1) вивчення та порівняння архітектури фреймворків React та Svelte;
2) оцінка ефективності різних методів рендерінгу в різних умовах за допомогою вимірювань продуктивності;

в) розробка прикладів:

1) створення простих веб-додатків з використанням фреймворків React та Svelte;
2) реалізація та порівняння методів рендерінгу у створених додатках;

г) оцінка впливу на продуктивність:

1) аналіз впливу вибору методу рендерінгу на продуктивність в реальних умовах використання;

д) висновки та рекомендації:

1) сформулювання висновків щодо ефективності методів рендерінгу у фреймворках React та Svelte;
2) надання рекомендацій щодо вибору конкретного підходу в залежності від вимог та завдань проєкту.

Очікується отримання глибокого розуміння роботи та ефективності методів рендерінгу у фреймворках React та Svelte, а також навичок у їх застосуванні через практичні приклади та аналіз впливу на продуктивність веб-додатків.

# 2 МЕТОДИ ВИРІШЕННЯ ПРОБЛЕМИ

## 2.1 Оцінка та порівняння широко використовуваних фреймворків JavaScript

Популярність мови програмування JavaScript зростає щодня, сприяючи розвитку як кількості, так і якості технологій та фреймворків, що базуються на цій мові. Спільнота з відкритим кодом постійно вдосконалює та розширює ці технології [4]. Проте існує значна кількість проблем, пов'язаних з проектуванням та розробкою веб-додатків:

- ускладнена масштабованість коду;
- низька швидкість розробки та впровадження;
- нехтування принципами SOLID;
- чутливість UI на стороні клієнта та велика кількість ресурсів;
- використовуваних додатком.

Для подолання цих недоліків розроблено фреймворки, такі як Vue, Svelte та React. Кожен з них частково вирішує ці проблеми, проте кожен має власний підхід до цього завдання. Фреймворки виступають каркасом для окремих сторінок веб-додатку, звільняючи розробників від необхідності зосередження на структурі коду та його підтримці, щоб вони могли сконцентруватися на створенні складних елементів інтерфейсу.

Серед переваг використання JavaScript фреймворків слід відзначити ефективність, безпеку та витрати. Вони дозволяють розробникам ефективно втілювати проекти та забезпечують безпеку за допомогою фірмових систем безпеки та підтримки великої спільноти. Більшість фреймворків є відкритими та безкоштовними, сприяючи зниженню витрат на розробку.

Однак, незважаючи на всі переваги фреймворків, розробники змушені здійснювати компроміс між використанням цих інструментів та продуктивністю проекту. Для масштабних проектів, спрямованих на використання мільйонами користувачів, кожен мегабайт пам'яті та кожна мілісекунда, витрачена на роботу фреймворку, стають важливими питаннями.

На сучасному етапі розвитку інформаційних технологій фреймворки, такі як React, Svelte та Vue, займають провідні позиції серед найпопулярніших JavaScript фреймворків. Із стрімким зростанням кількості нових фреймворків виникає необхідність для розробників обирати оптимальні рішення. Тому важливо проводити порівняльний аналіз JavaScript фреймворків за об'єктивними критеріями (див. табл. 1), які є критичними для розробників. Серед таких показників можна виділити:

- рендерінг: цей аспект визначає якість та ефективність відображення кінцевого результату, що є ключовим для користувацького досвіду;
- архітектура компонентів: розглядається структура та організація компонентів, які використовуються у фреймворках;
- спрямованість та класи залежностей: оцінюється напрямок розвитку фреймворка та ступінь залежності від сторонніх бібліотек чи модулів;
- зворотна сумісність: розглядається здатність фреймворка підтримувати попередні версії та код, розроблений за їхньою основою;
- документація та підтримка: оцінюється наявність чіткої документації та активність спільноти розробників, що є важливими для успішного використання фреймворка.

Таблиця 1 – Порівняння Популярних фреймворків

| Фреймворк | Рендерінг | Архітектура компонентів | Спрямованість та класи залежностей | Зворотна сумісність | Документація та підтримка |
|---|---|---|---|---|---|
| React | Використовує віртуальний DOM. | Концепція компонентів (функціональні та класові). | Частково спрямований (JSX). | Зазвичай дотримується, але можливі проблеми під час оновлень. | Широка документація та активна спільнота. |
| Фреймворк | Рендерінг | Архітектура компонентів | Спрямованість та класи залежностей | Зворотна сумісність | Документація та підтримка |

Кінець таблиці 1

| Фреймворк | Рендерінг | Архітектура компонентів | Спрямованість та класи залежностей | Зворотна сумісність | Документація та підтримка |
|---|---|---|---|---|---|
| Svelte | Компілює в нативний JavaScript. | Проста та декларативна архітектура. | Спрямований (декларативний) за замовчуванням. | Обмежена зворотна сумісність через філософію компіляції. | Докладна документація, менша спільнота. |
| Vue | Використовує реактивну систему. | Однофайлові компоненти. | Частково спрямований. | Зазвичай дотримується, робить оновлення менш болісним. | Добре структурована документація та активна спільнота. |

По-перше, сучасні JavaScript-фреймворки повинні відповідати специфікації веб-компонентів, оскільки у клієнтській розробці велика увага приділяється створенню користувацьких HTML-елементів.

По-друге, надійний фреймворк має власну екосистему, яка надає готові рішення для різних аспектів розробки на стороні клієнта, таких як маршрутизація, управління станом, інтеракція із серверною частиною та інші. Фреймворки, такі як Svelte, React, Vue, відповідають сучасним стандартам JavaScript ES6+ та мають власні екосистеми для забезпечення ефективної розробки.

### 2.1.1 React

React – це бібліотека JavaScript, розроблена та підтримувана компанією Facebook, призначена для створення інтерфейсів користувача (UI). Вона надає можливість розробникам створювати ефективні, масштабовані та легкі в управлінні веб-додатки, використовуючи компонентну модель (див. рис. 2.1).

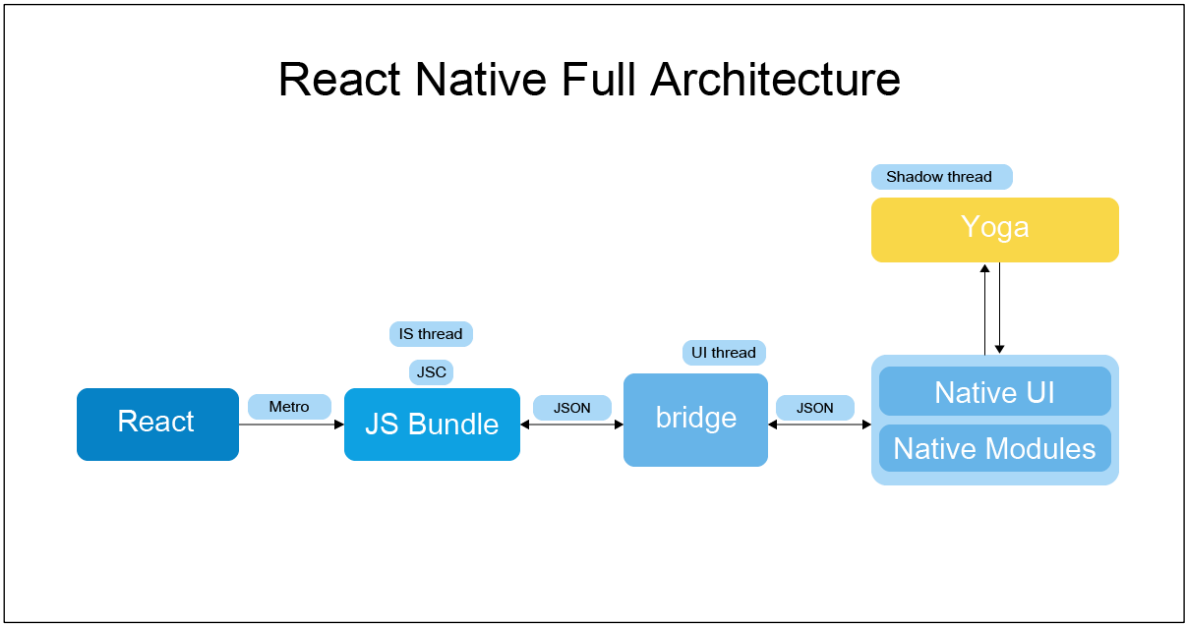


Рисунок 2.1 – Архітектура React (за даними [5])

Основні особливості React включають.

Компонентна модель: React дозволяє розробникам будувати веб-інтерфейси у вигляді окремих компонентів, які можуть бути повторно використані та комбіновані в більш складні інтерфейси. Це сприяє підходу "розділяй та володарюй" для поліпшення організації коду та прискорення розробки.

Віртуальний DOM: React використовує віртуальний DOM для ефективного оновлення інтерфейсу без прямої маніпуляції реальним DOM. Це допомагає досягти високої продуктивності веб-додатків, зменшуючи кількість зайвих операцій з DOM.

Односторонній потік даних: Використання одностороннього потоку даних у React дозволяє визначити єдину "правдиву" версію даних та автоматично оновлювати інтерфейс відповідно до цих даних.

Використання JSX: JSX - це розширення синтаксису JavaScript, що дозволяє використовувати HTML-подібний синтаксис в коді JavaScript, зроблюючи його більш зрозумілим та легким для розробки.

Розширюваність: React дозволяє розширювати функціональність за допомогою різноманітних бібліотек та інструментів, таких як Redux, React Router, PropTypes та інші.

React розділяється на низку компонентів, де один файл компонента містить як бізнес-логіку, так і HTML-розмітку у вигляді JSX, яка фактично перетворюється на функції JavaScript. Для забезпечення взаємозв'язку між цими компонентами можна використовувати Flux або аналогічні бібліотеки JavaScript. Також у React існують спеціальні об'єкти - state та props. Завдяки об'єктам стану та реквізиту можна передавати дані з компоненти у шаблон (використовуючи об'єкт стану) або від батьківської компоненти до дочірньої (використовуючи об'єкт реквізиту).

React пропонує різні інструменти для досягнення гнучкості. Наприклад, замість Flux можна обрати MobX, Redux, Fluxy, Fluxible або RefluxJS. Існує багато інших бібліотек управління станом для React, а також бібліотек HTTP, таких як jQuery AJAX, fetch API, Superagent і Axios.

Незважаючи на те, що гнучкість React є його головною перевагою, це також може призводити до деяких проблем. Вибір серед багатьох додаткових бібліотек часто ставить перед розробником дилему щодо того, які саме інструменти використовувати разом з React.

React намагається встановити галузеві стандарти, наприклад, Redux вважається найкращою та єдиною бібліотекою для програм React корпоративного рівня. Однак, в той же час, вибір Redux може призвести до ускладнення розробки, особливо коли потрібно впровадити нову функцію, і зміни вимагають великої кількості модифікацій у всьому додатку, що може вимагати створення власного робочого процесу для роботи з React.

### 2.1.2 Svelte

Svelte – це молода, але інноваційна технологія для розробки веб-інтерфейсів, яка відрізняється від традиційних фреймворків, таких як React, Angular чи Vue. Створений компанією Svelte Ltd., Svelte пропонує унікальний підхід до роботи з веб-компонентами та генерації коду (див. рис. 2.2).

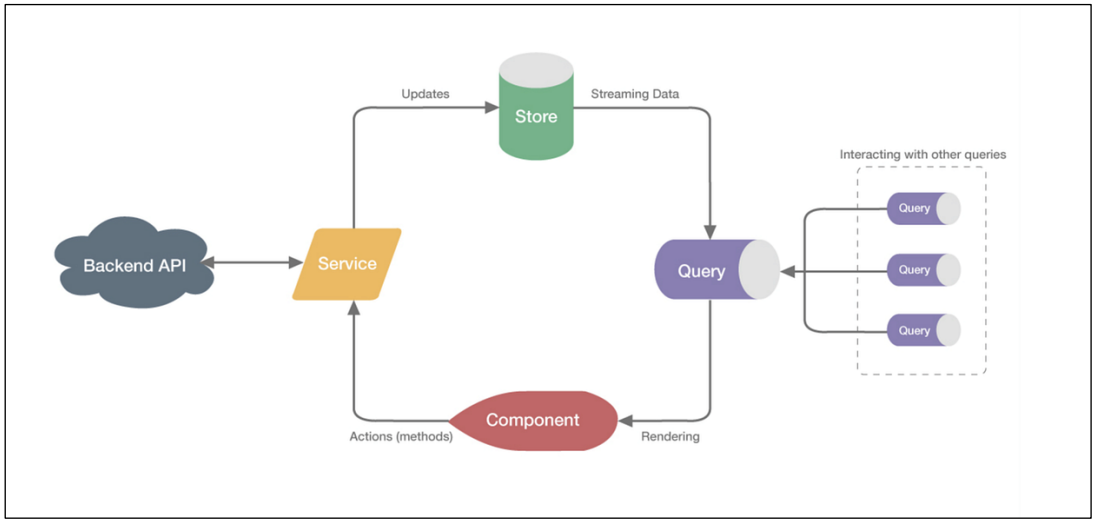


Рисунок 2.2 – Архітектура Svelte (за даними [6])

Основні особливості Svelte.

Компіляція на етапі збору: Основна відмінність Svelte полягає в тому, що весь код компонентів компілюється на етапі збору, а не під час виконання. Це означає, що більшість роботи відбувається під час збору проекту, і фінальний бандл, який використовується у веб-додатку, вже містить оптимізований та ефективний код.

Відсутність віртуального DOM: На відміну від інших фреймворків, Svelte не використовує віртуальний DOM для оновлення інтерфейсу. Замість цього, він генерує і оптимізований JavaScript-код, який безпосередньо маніпулює реальним DOM.

Маленький розмір вивчальної кривої: Завдяки своєму простому синтаксису та концепціям, Svelte є легким для вивчення, зменшуючи вивчальну криву для новачків.

Зниження кількості коду: Svelte пропонує чистий та лаконічний синтаксис, що дозволяє зменшити кількість коду порівняно з іншими фреймворками.

Вбудована обробка анімацій та переходів: Svelte надає зручні інструменти для створення анімацій та переходів без необхідності додаткових бібліотек.

Автоматичне визначення залежностей: Svelte автоматично визначає, які частини коду компонента залежать від змін у стані, і генерує відповідний код для оновлення лише необхідних елементів інтерфейсу.

Svelte, хоча і має численні переваги, також має свої недоліки. Серед них варто виділити обмежений досвід в розробці масштабних проектів, адже, як молода технологія, Svelte може не мати достатньої кількості ресурсів та розширених функціональних можливостей для великих та складних за розміром проектів.

### 2.1.3 Vue

На перший погляд може здатись, що бібліотека Vue – це комбінація Angular та React. Фактично, Еван Ю, автор Vue, взяв узагальнені концепції з Angular та React. Наприклад, Vue вимагає зберігати логіку компонентів та їхні розмітки, а також таблиці стилів у одному файлі, схоже на підхід React до розробки без таблиць стилів. Для сприяння взаємодії компонентів, Vue використовує властивості та об'єкти стану.

Vue – це прогресивний фреймворк для розробки інтерфейсів користувача (UI) на базі JavaScript. Він надає розробникам можливість створювати інтерактивні веб-додатки, використовуючи декларативний підхід до роботи із змінним станом та компонентною архітектурою (див. рис. 2.3).

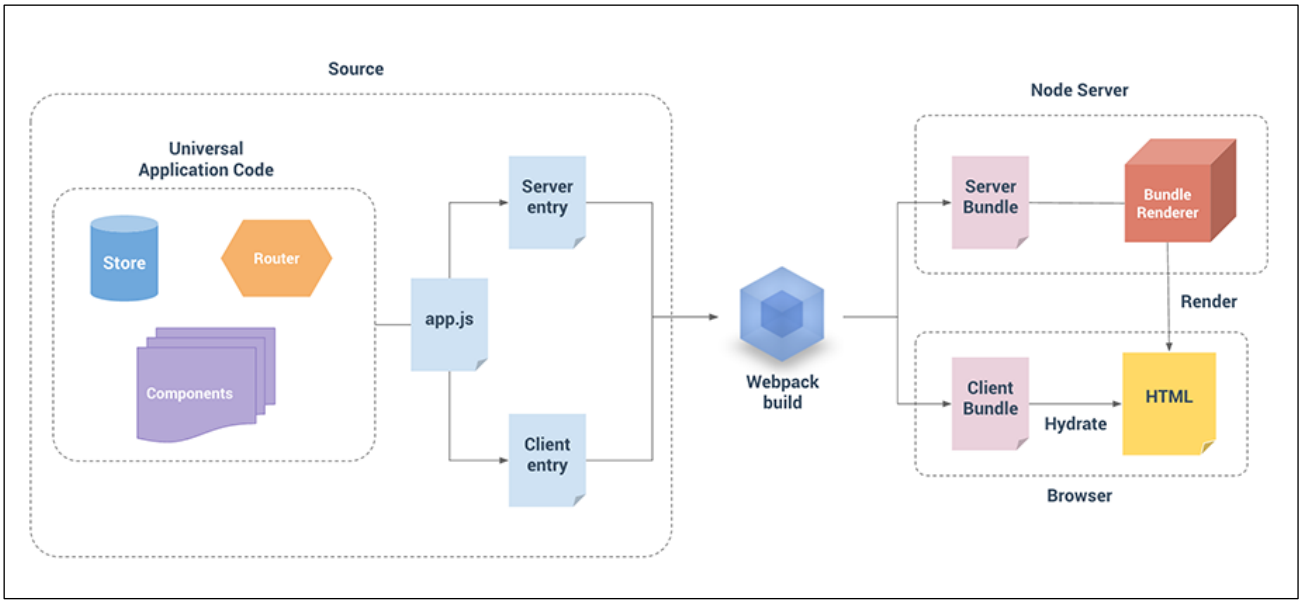


Рисунок 2.3 – Архітектура Vue (за даними [7])

Основні риси та характеристики Vue включають.

Декларативний синтаксис: Vue використовує шаблони у вигляді розширень HTML для визначення відображення компонентів, що полегшує розробку інтерфейсу користувача.

Компонентна архітектура: Vue дозволяє розробникам створювати веб-додатки, використовуючи компоненти – незалежні, повторно використовувані блоки коду, які можна вкладати один в одного для створення складних інтерфейсів.

Реактивність: Vue має реактивну систему, яка автоматично оновлює відображення компонентів при зміні їхнього стану, спрощуючи роботу зі змінним станом.

Модульність: Vue дозволяє розробникам використовувати лише ті частини фреймворку, які необхідні в їхньому проекті, забезпечуючи легкість та масштабованість.

Розширюваність: Vue підтримує багато доповнень та плагінів, що дозволяють розширити його функціональність та інтегрувати різноманітні зовнішні бібліотеки.

Екосистема Vue включає різноманітні інструменти, бібліотеки та розширення для розширення функціональності та можливостей Vue.js (див. рис. 2.4).

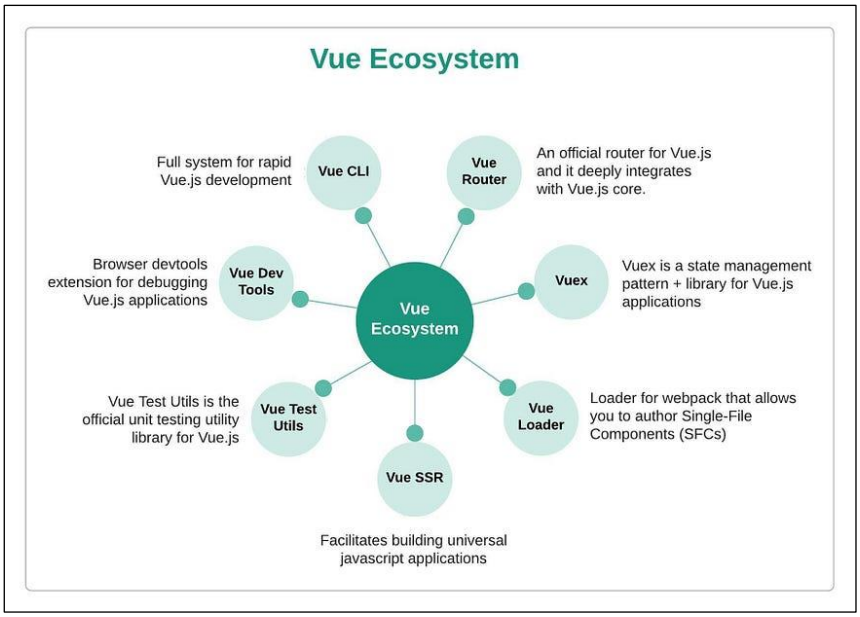


Рисунок 2.4 – Екосистема Vue (за даними [8])

Ключові компоненти екосистеми Vue охоплюють такі аспекти.

Vue Router: Бібліотека для реалізації маршрутизації в Vue додатках, яка дозволяє визначати маршрути, навігацію між сторінками та інші параметри.

Vuex: Стан-менеджер для Vue додатків, який спрощує керування станом додатка, зберігання даних та виконання асинхронних операцій.

Vue CLI: Командний інтерфейс для створення, налаштування та управління Vue проектами, надаючи шаблони та підтримку розширень.

Vue DevTools: Розширення для браузера, яке забезпечує розробникам інструменти для налагодження, профілювання та аналізу Vue додатків.

Vue Server-Side Rendering (SSR): Механізм рендерингу Vue додатків на сервері для покращення продуктивності та SEO-оптимізації.

Vue Test Utils: Набір утиліт для тестування Vue компонентів, який допомагає створювати та перевіряти різноманітні тести.

Vue Material, Vuetify, Element UI та інші UI бібліотеки: Готові компоненти та стилі для швидкої розробки красивого та функціонального інтерфейсу Vue додатків.

Vue Plugins: Велика кількість плагінів, розроблених спільнотою Vue, що розширюють функціональність фреймворку, такі як анімації, міжнародизація, валідація форм, кешування, аутентифікація та інші.

Vue дозволяє розробникам використовувати ці компоненти для розширення можливостей фреймворку, і його робочий процес схожий з іншими фреймворками, такими як React. Таке порівняння може виявитися корисним при оцінці ефективності застосування Vue та React для вирішення тих самих завдань в практичному плані.

## 2.2 Аналіз характеристик продуктивності та стратегій рендерингу

Аналіз характеристик продуктивності та стратегій рендерингу є важливою частиною дослідження ефективності веб-фреймворків. Давайте розглянемо ці аспекти для фреймворків React та Svelte.

Характеристики продуктивності:

а) продуктивність завантаження (Largest Contentful Paint - LCP):

1) React: Продуктивність завантаження може бути покращена за допомогою оптимізацій, таких як код-спліттінг та використання Lazy Loading;
2) Svelte: Завдяки компіляції на етапі збірки, Svelte може генерувати ефективний код, що призводить до швидшого завантаження;

б) інтерактивність (First Input Delay - FID):

1) React: Інтерактивність може впливати на FID. Оптимізація коду та використання Web Workers можуть поліпшити цей показник;
2) Svelte: Спрощена структура коду та ефективне використання ресурсів можуть сприяти низькому FID;

в) візуальна стабільність (Cumulative Layout Shift - CLS):

1) React: Використання правильних CSS правил та ретельна робота з макетами можуть допомогти у покращенні CLS;
2) Svelte: Як компільований фреймворк, Svelte може уникнути зсувів макету завдяки оптимізації на етапі компіляції;

Стратегії рендерингу:

а) віртуальний DOM (React):

1) React: Використовує віртуальний DOM для ефективного оновлення та відображення змін;
2) Svelte: У Svelte відсутній віртуальний DOM; замість цього, код компілюється в ефективний JavaScript код без зайвих абстракцій;

б) компіляція (Svelte):

1) React: Вимагає інтерпретації коду в браузері, що може впливати на продуктивність;
2) Svelte: Компілює код на етапі збірки, що дозволяє генерувати ефективний та оптимізований код;

в) робота зі станом:

1) React: Використовує state та props для передачі та управління станом компонентів;

2) Svelte: Використовує змінні, які автоматично відстежуються та сприяють реактивності без необхідності використання state.

Аналіз цих характеристик дозволяє зрозуміти сильні та слабкі сторони кожного фреймворку та визначити, який може бути оптимальним для конкретного проекту залежно від його вимог та контексту використання.

### 2.2.1 Largest Contentful Paint

Largest Contentful Paint (LCP) – це один із ключових показників веб-продуктивності, який вимірює час, необхідний для завантаження та відображення найбільшого контенту на сторінці. Даний показник враховує час, коли основний вміст, такий як текст, зображення або відео, стає видимим для користувача (див. рис. 2.5).


Рисунок 2.5 – Ефективність показників LCP (за даними [9])

Основні риси Largest Contentful Paint.

Основний вміст: LCP фокусується на найбільшому елементі, який відображається на екрані, і є частиною основного контенту сторінки. Це може бути заголовок, головне зображення або інший важливий для користувача елемент.

Вимірюється в часі: LCP вимірюється в мілісекундах і представляє собою час, який минув від початку завантаження сторінки до моменту, коли найбільший контент стає видимим для користувача.

Вплив на користувача: LCP важливий, оскільки він надає інформацію про те, наскільки швидко користувач може отримати доступ до основного вмісту сторінки. Повільний LCP може призвести до негативного враження користувача і зменшення задоволення від відвідування веб-сайту.

Оптимізація для високої продуктивності: Оптимізація LCP включає в себе використання ефективних засобів завантаження та відображення контенту, використання CDN (Content Delivery Network), асинхронне завантаження ресурсів та інші стратегії для зменшення часу завантаження.

LCP є одним із трьох показників Core Web Vitals, які визначають якість взаємодії користувача з веб-сайтом. Інші два показники - First Input Delay (FID) та Cumulative Layout Shift (CLS) - також важливі для загальної оцінки користувацького досвіду. Всі ці метрики входять у новий стандарт для вимірювання веб-продуктивності та є частиною стратегії Google для поліпшення якості веб-сайтів.

### 2.2.2 First Input Delay

First Input Delay (FID) – це метрика веб-продуктивності, яка вимірює інтерактивність веб-сайту. FID вказує на час затримки між першим взаємодією користувача з елементом на сторінці (наприклад, натисканням на кнопку) та відгуком браузера на цю дію (див. рис. 2.6).

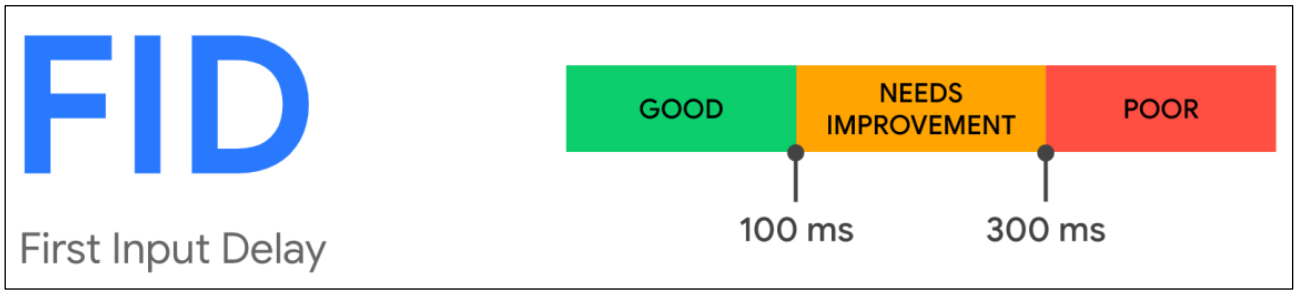


Рисунок 2.6 – Ефективність показників FID (за даними [12])

Основні риси First Input Delay.

Вимірюється в часі: FID вимірюється в мілісекундах і вказує на час, який пройшов між фактичним взаємодією користувача та відгуком браузера.

Значення нижче – краще: Мета – максимально зменшити значення FID, оскільки низьке значення свідчить про те, що сторінка реагує на дії користувача швидко і ефективно.

Індикатор інтерактивності: FID вимірюється тільки при першій взаємодії користувача з сайтом. Він не враховує події після першої взаємодії, такі як асинхронні завантаження або оновлення вмісту.

Вплив на користувача: Високе значення FID може призвести до враження, що сайт повільно реагує на дії користувача, що може призвести до негативного враження та зменшення задоволення від відвідування сторінки.

Оптимізація для високої продуктивності: Оптимізація FID може включати в себе використання асинхронного завантаження ресурсів, мінімізацію використання JavaScript та інші стратегії, спрямовані на покращення відгуку сайту на взаємодію користувача.

FID входить у склад Core Web Vitals, які визначають якість взаємодії користувача з веб-сайтом. Разом з Largest Contentful Paint (LCP) та Cumulative Layout Shift (CLS), FID стає частиною нового стандарту для вимірювання та поліпшення продуктивності веб-сайтів.

### 2.2.3 Cumulative Layout Shift

Cumulative Layout Shift (CLS) - це метрика веб-продуктивності, яка визначає стабільність візуального вмісту на сторінці під час завантаження та взаємодії з користувачем. CLS визначається сумою всіх безпорадних зміщень елементів на сторінці під час її відображення (див. рис. 2.7).


Рисунок 2.7 – Ефективність показників CLS (за даними [12])

Основні риси Cumulative Layout Shift.

Сприймається як важливий показник взаємодії: CLS вказує на те, наскільки суттєво змінюється макет сторінки під час її взаємодії з користувачем. Більш

низьке значення CLS свідчить про більш стабільний та приємний візуальний досвід.

Вимірюється відносно видимого вмісту: Зміщення елементів рахується тільки для видимої частини сторінки. Невидимі елементи не впливають на значення CLS.

Відслідковує зміни макету в реальному часі: CLS враховує зміщення елементів під час завантаження сторінки і при подіях, таких як натискання на кнопки або завантаження асинхронного вмісту.

Вплив на користувача: Високе значення CLS може викликати неприємний ефект "підривного" зміщення елементів, що може спричинити негативний враження від взаємодії з веб-сайтом.

Оптимізація для покращення стабільності: Оптимізація CLS може включати в себе використання атрибутів "width" та "height" для резервування місця для медіа-елементів, уникання вставок асинхронного вмісту у верхній частині сторінки та інші стратегії для зменшення зміщень.

CLS входить у склад Core Web Vitals, які визначають якість взаємодії користувача з веб-сайтом, спільно з Largest Contentful Paint (LCP) та First Input Delay (FID).

# 3 АДАПТАЦІЯ ОСНОВНИХ СТРАТЕГІЙ РЕНДЕРИНГУ ТА СТВОРЕННЯ КОМБІНОВАНИХ СТРАТЕГІЙ ДЛЯ ВІДПОВІДНОСТІ СУЧАСНИМ ВИМОГАМ

## 3.1 Гнучкі підходи до відтворення веб-додатків

Сучасні вимоги до веб-додатків постійно змінюються, що вимагає більш гнучких підходів у контексті конкурентного середовища та високих очікувань користувачів. Використання однакової стратегії для всіх сторінок суперечить основній меті - створенню продуктивного сайту та забезпеченню високої конверсії. Наприклад, деякі веб-додатки зараз потребують включення блогу як складової частини, або поєднання статичних сторінок з тими, що постійно оновлюються, зберігаючи при цьому ефективність у всіх випадках використання.

У випадках, коли додаток вимагає використання переважно серверного рендерингу та необхідно зберегти динамічність та гнучкість клієнтської сторони, і коли використання статичних HTML-сторінок не можливе через особливості системи або поток даних, доцільним є адаптування відомих стратегій під вимоги та розробка нових стратегій рендерингу.

## 3.2 Комбінування виконання рендерингу на стороні сервера та клієнта

У випадках, коли важливі якісні показники SEO та висока інтерактивність для клієнтів, можна скористатися перевагами рендерингу на обох сторонах – сервері та клієнта (SSR і CSR). Під час навігаційних запитів, таких як запроси сторінок або їх оновлення, сервер буде обробляти ці запити, будуючи та надсилаючи клієнту готовий HTML-документ, а потім JavaScript і дані, необхідні для відтворення, будуть вбудовані в кінцевий документ. При правильній реалізації цей підхід також забезпечить швидке відображення контенту (First Paint), подібне до серверного рендерингу.

Наприклад, зазвичай типовий веб-додаток, розроблений з використанням React, використовує процес гідратації для рендерингу на стороні клієнта. Цей процес полягає в тому, що React переглядає поточні елементи DOM і наповнює їх даними або модифікує за допомогою відповідного JavaScript-коду. Замість того,

щоб обробляти HTML для створення DOM, клієнтська візуалізація використовує JavaScript для його створення. Мінімальний HTML-документ виступає контейнером програми і містить лише посилання на JavaScript і CSS, необхідні для відтворення програми.

Такий підхід має основний недолік: хоча він може покращити швидкість першого відображення сторінки, це може суттєво погіршити загальний індекс сприйняття (FID). Вигляд сторінки може натякати на її повну готовність та інтерактивність, але фактично вона може бути неспроможною для взаємодії до тих пір, поки не буде повністю завантажений JavaScript-код на стороні клієнта та не буде приєднано обробники подій (наприклад, для миші або прокручування сторінки). Це може займати кілька секунд або навіть хвилин, особливо на мобільних пристроях.

За допомогою поетапного наповнення HTML дерева додатку можна відкласти модифікацію (та заблокувати інтерактивність) менш важливих частин сторінки. Це дозволяє зменшити обсяг JavaScript, необхідного для запитування з сервера, щоб зробити сторінку інтерактивною, і обробляти DOM-елементи лише тоді, коли це потрібно користувачеві. Використання поетапного hydration також допомагає уникнути ситуацій, коли дерево DOM, побудоване сервером, зруйнується, а потім негайно перебудується.

Для того, щоб всі елементи на сторінці стали інтерактивними за допомогою гідратації, код React для цих компонентів має бути включений у сбірку, яка завантажується з сервера на клієнт. Наприклад, високоінтерактивні односторінкові додатки, які головним чином контролюються JavaScript-кодом, вимагатимуть одразу всього пакету сбірки. Проте для веб-сайтів, які переважно статичні з кількома інтерактивними елементами на екрані, може бути необов'язково, щоб всі компоненти стали активними негайно.

Ця проблема вирішиться за допомогою поетапного наповнення, яке дозволить нам активувати зволоження для деяких компонентів під час завантаження сторінки, а інші частини будуть будуватися поступово, коли це буде

потрібно. На рисунку 3.1 показано, як працюватимуть серверні запити на сторінку та як буде оброблятися JS-код при застосуванні даної стратегії рендерингу.

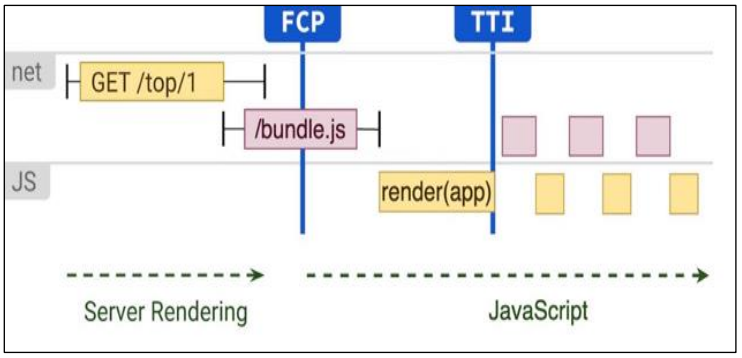


Рисунок 3.1 – Візуалізація стратегії покрокового заповнення.

Для реалізації поступового заповнення можна скористатися однією з наступних бібліотек: React Query або SWR. Наприклад, бібліотека «SWR» містить API під назвою «useSWR», яке дозволяє позначати більш інтерактивні компоненти для активації hydration в першу чергу для них. На рисунку 3.2 наведено приклад впровадження описаної техніки у коді з використанням JavaScript та React.

```
import useSWR from 'swr';
import Component from '../components/Component';

const fetcher = (url: string) => fetch(url).then(res => res.json());

const HomePage: React.FC = () => {
  const { data, error } = useSWR('api/data', fetcher);

  if (error) return <div>Failed to load</div>;
  if (!data) return <div>Loading...</div>;

  return (
    <main>
      <Component column={1} />
      <Component column={data.column2} />
      <Component column={data.column3} />

      <Component column={1} />
      <Component column={2} />
      <Component column={3} />

      <Component column={1} />
      <Component column={2} />
      <Component column={3} />
    </main>
  );
};

export default HomePage;
```

Рисунок 3.2 – Демонстрація застосування поетапного заповнення за допомогою бібліотеки "swr"

Компонент "Component" з котрий приймає “data” буде першим, з яким можна взаємодіяти у стовпцях 2 і 3. Після цього застосовується API "useSWR", включений у бібліотеку, щоб активувати інтерактивність інших компонентів на клієнті. На рисунку 3.3 можна побачити інтерфейс бібліотеки.

```
declare const unstable_serialize: (key: Key) => string;

declare const SWRConfig: ReactExports.FC<ReactExports.PropsWithChildren<{
    value?: SWRConfiguration | ((parentConfig?: SWRConfiguration | undefined) => SWRConfiguration) | undefined;
}>> & {
    defaultValue: FullConfiguration;
};

declare const useSWR: SWRHook;

export { SWRConfig, useSWR as default, unstable_serialize };
```

Рисунок 3.3 – Приклад інтерфейсу бібліотеки

Переваги такої стратегії рендерингу полягають у наступному:

– завантаження лише тих частин сторінки, які рідко використовуються користувачем, лише за необхідності;
– автоматичне розділення коду (code-splitting) призводить до зменшення розміру пакета, що допомагає зменшити час між FCP і TTI.

## 3.3 Потокове надсилання

Для скорочення часу до доступності інтерактивності сторінки (метрика TTI) можна використовувати техніку рендерингу на стороні сервера шляхом надсилання HTML-коду частинами. Це дозволяє браузеру поступово обробляти та відображати частини сторінки після їх отримання з серверу. Замість створення одного великого HTML-файлу, що містить необхідну розмітку для поточної сторінки, можна розбити його на менші фрагменти. Потокові HTML-елементи дозволять передавати дані у відповідному об'єкті, що призведе до постійного відправлення даних на клієнтську сторону. Коли браузер отримає фрагменти цих даних, він зможе розпочати їх візуалізацію (див. рис. 3.4).

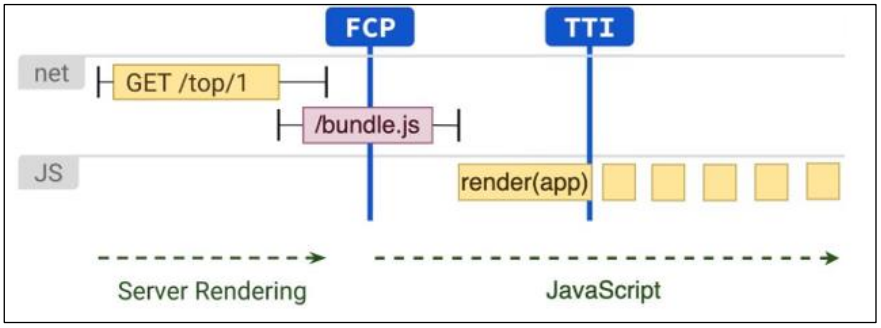


Рисунок 3.4 – Графічне відображення обчислення рендерингу на сервері

Наприклад, коли розробник використовує фреймворк Next.js для створення додатка, він може ввімкнути "concurrentFeatures" в конфігурації, щоб надсилати код додатка частинами у невеликих пакетах збірки. Такий підхід дозволяє браузеру почати будувати інтерфейс для користувача, навіть коли він ще отримує дані. Це забезпечує швидку візуалізацію сторінки (First Paint) та LCP, оскільки HTML надходить до користувачів швидше. Після цього можна викликати метод Next.js "dynamic" на отриманих DOM-елементах, щоб додати відповідні обробники подій (event listeners), що зробить інтерфейс користувача інтерактивним.

В якості прикладу розглядається веб-додаток зі списком даних на сторінці з понад 1000 елементами. На рисунку 3.5 показана функція для створення списку з даними.

```
const generateFakeData = (num: number) => {
  return Array.from({ length: num }, () => ({
    id: faker.datatype.uuid(),
    name: faker.name.fullName(),
    email: faker.internet.email(),
  }));
};

const fakeData = generateFakeData(1000);
```

Рисунок 3.5 – Створення списку з даними

На серверній стороні використовується функція "dynamic". У частині файлу, показаній на рисунку 3.6, ініціюється базовий сервер додатка за вбудованих можливостей Next.js. Також створено частину HTML-розмітки, яку клієнт отримає одразу. Цей HTML-фрагмент містить корисну інформацію, яку програма повинна

використовувати для правильного рендерингу сторінки, а саме назву документа, таблицю стилів та головний контент.

```
import React, { Suspense } from 'react';
import dynamic from 'next/dynamic';

const ServerComponent = dynamic(() => import('../components/ServerComponent'), {
  ssr: true,
});

export default function App() {
  return (
    <div>
      <h1>Welcome to Streaming SSR</h1>
      <Suspense fallback={<div>Loading...</div>}>
        <ServerComponent />
      </Suspense>
    </div>
  );
}
```

Рисунок 3.6 – Основна сторінка додатку

На рисунку 3.7 показано використання функції "dynamic". Головний компонент додатку App будується на сервері у потоці за допомогою Next.js. Перша частина HTML разом із пакетами даних з компонента "App" відправляється в об'єкті відповіді. Якщо використовується SSR для рендерингу компонента "App", користувачеві потрібно зачекати, доки програма отримає всі дані, щоб розпочати завантаження та обробку. Для прискорення цього процесу "dynamic" дозволяє додатку почати завантажувати та обробляти інформацію, паралельно продовжуючи отримувати фрагменти даних від компонента сервера.

```
import React from 'react';
import { faker } from '@faker-js/faker';

const generateFakeData = (num: number) => {
  return Array.from({ length: num }, () => ({
    id: faker.datatype.uuid(),
    name: faker.name.fullName(),
    email: faker.internet.email(),
  }));
};

const fakeData = generateFakeData(1000);

export default function ServerComponent() {
  return (
    <div>
      <h1>List of Fake Users</h1>
      <ul>
        {fakeData.map((item) => (
          <li key={item.id}>
            {item.name} - {item.email}
          </li>
        ))}
      </ul>
    </div>
  );
```

Рисунок 3.7 – Компонент з елементами

Такий підхід видає певні переваги для сайту. Розмітка досягає клієнта незабаром після початку рендерингу на сервері, що поліпшує показник Time to First Byte порівняно з базовим SSR. TTFB стає більш стабільним та передбачуваним на сторінках різного розміру. Браузер може почати обробляти HTML, як тільки його отримує, що призводить до кращих показників FP і FCP. Потокове передавання добре впорається з зворотним тиском мережі або перевантаженням, що дозволяє адаптувати веб-сайти навіть за складних умов. Сканери пошукової системи можуть зчитувати потокову відповідь, тому сторінка доступна для SEO.

# 4 МЕТОДИКА ПОРІВНЯЛЬНОГО АНАЛІЗУ

## 4.1 Підбір параметрів для аналізу

Вибір параметрів для аналізу є ключовим етапом у порівняльному дослідженні JavaScript-фреймворків, оскільки саме вони є основою для об'єктивної та всебічної оцінки. Для ефективного порівняння React, Vue.js та Svelte були обрані такі критерії:

- швидкість виконання та продуктивність охоплюють аналіз часу завантаження сторінки, часу реакції на дії користувача, а також ефективність використання ресурсів (процесора та пам'яті) [10];
- простота освоєння та розробки, оцінка часу та зусиль, необхідних для новачків, щоб почати працювати з фреймворком, а також зручність розробки і підтримки застосунків [11];
- гнучкість та масштабованість, оцінка здатності фреймворків адаптуватися до змінних вимог проєкту та їх здатність підтримувати великі та складні застосунки;
- безпека, аналіз вбудованих механізмів безпеки та здатність фреймворків захищати застосунки від поширених вразливостей [12];
- тестування та налагодження (debugging), дослідження інструментів та можливостей, які фреймворки надають для тестування та усунення помилок у застосунках;
- спільнота та підтримка, оцінка активності та розміру розробницької спільноти, наявність навчальних матеріалів, та доступність підтримки;
- сумісність та інтеграція, аналіз здатності фреймворків працювати разом з іншими бібліотеками та інструментами, а також їх інтеграція з існуючими системами [13];
- екосистема та доступні ресурси, дослідження наявності готових рішень, плагінів, компонентів та інших ресурсів, що можуть полегшити та прискорити розробку.

Ці показники дозволять сформувати об'єктивне уявлення про кожен фреймворк, оцінюючи їх не лише за технічними характеристиками [13], а також за зручністю у використанні, підтримкою спільноти та ефективністю в реальних проектах [14].

### 4.2 Методи збору та аналізу даних

Розділ, який приділяється методам збору та аналізу даних, акцентує на підходах та інструментах, які були використані для забезпечення достовірності та відтворюваності результатів порівняльного аналізу JavaScript-фреймворків [15]. Проведення докладного та всебічного огляду кожного з фреймворків потребувало використання комбінованого підходу, який включав:

- аналіз офіційних документів, перегляд документації для розробників та технічних специфікацій фреймворків для вивчення їх можливостей та особливостей [16];
- проведення бенчмаркінгу (benchmarking), стандартизованих тестів продуктивності за допомогою інструментів, таких як Lighthouse та WebPageTest для вимірювання швидкості завантаження, часу реакції користувача та використання системних ресурсів;
- аналіз ринкової частки, використання даних з різних аналітичних джерел, таких як Google Trends, Stack Overflow Insights та GitHub, для визначення популярності та активності спільноти навколо кожного фреймворка;
- розробка кількох тестових проєктів [17], щоб визначити гнучкість, масштабованість та інтеграцію фреймворків у різних сценаріях використання. Вибір методів збору та аналізу даних був спрямований на мінімізацію упередженості [18] та підвищення об'єктивності оцінки, а також забезпечення повноти та глибини дослідження.

## 4.3 Процес оцінки фреймворків

### 4.3.1 Розробка програмного забезпечення

Для порівняльного аналізу фреймворків React, Vue.js та Svelte було створено програму "Список справ" на кожному з них. Цей проект було обрано через його універсальність та відображення типових завдань веб-розробки, включаючи управління станом, взаємодію з користувачем та рендерінг інтерфейсу.

Наступним кроком була розробка дизайну інтерфейсу (рис. 4.1). Було створено простий, але функціональний користувацький інтерфейс, який був однаковий для всіх фреймворків, забезпечуючи справедливе порівняння візуального представлення.

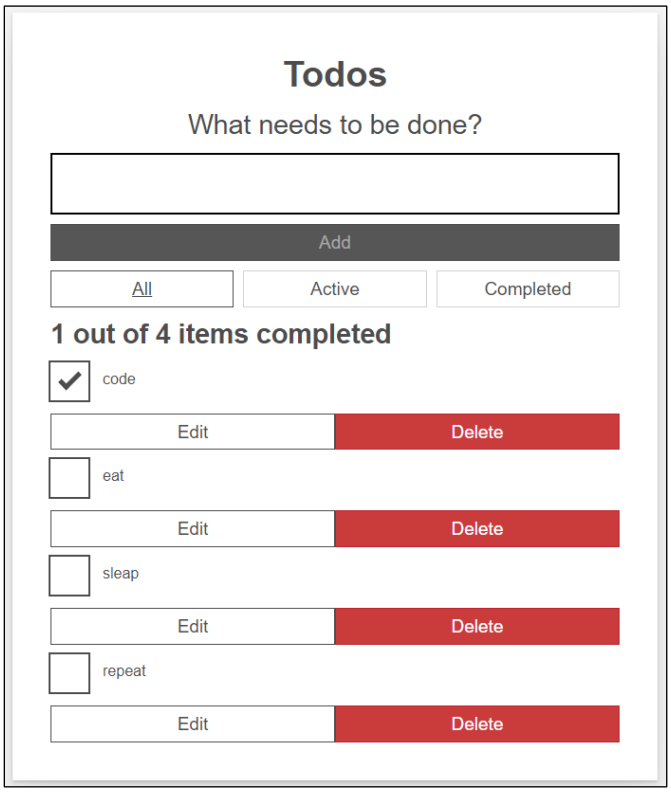


Рисунок 4.1 – Дизайн інтерфейсу

Розпочавши процес розробки, спочатку визначили функціональні вимоги до програми, які включали можливість додавання завдань, їх видалення (див. рис. 4.2) та відмічання виконаних (див. рис. 4.3). Це забезпечило однакові умови для всіх фреймворків.

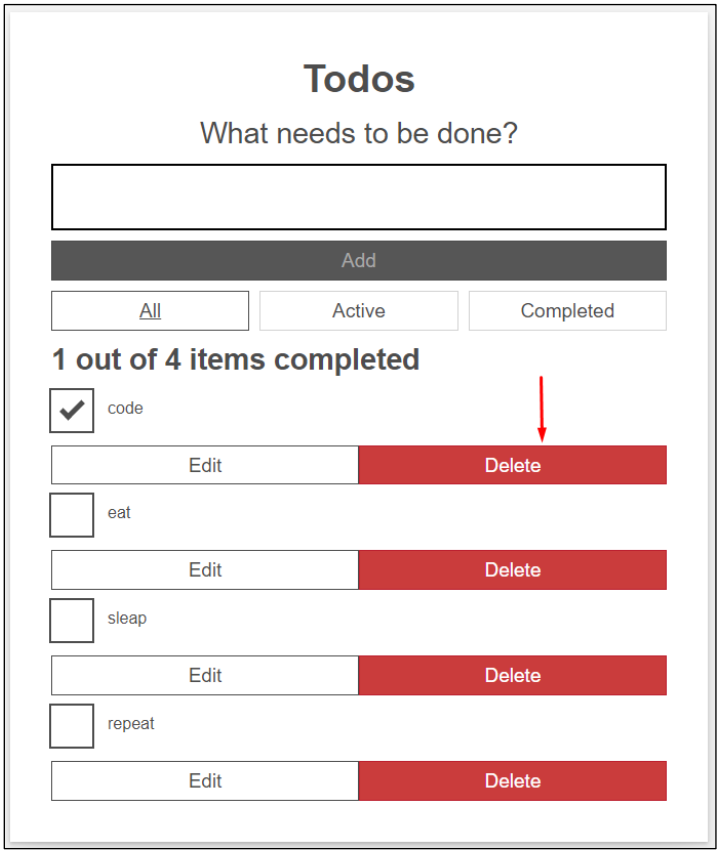


Рисунок 4.2 – Видалення завдань

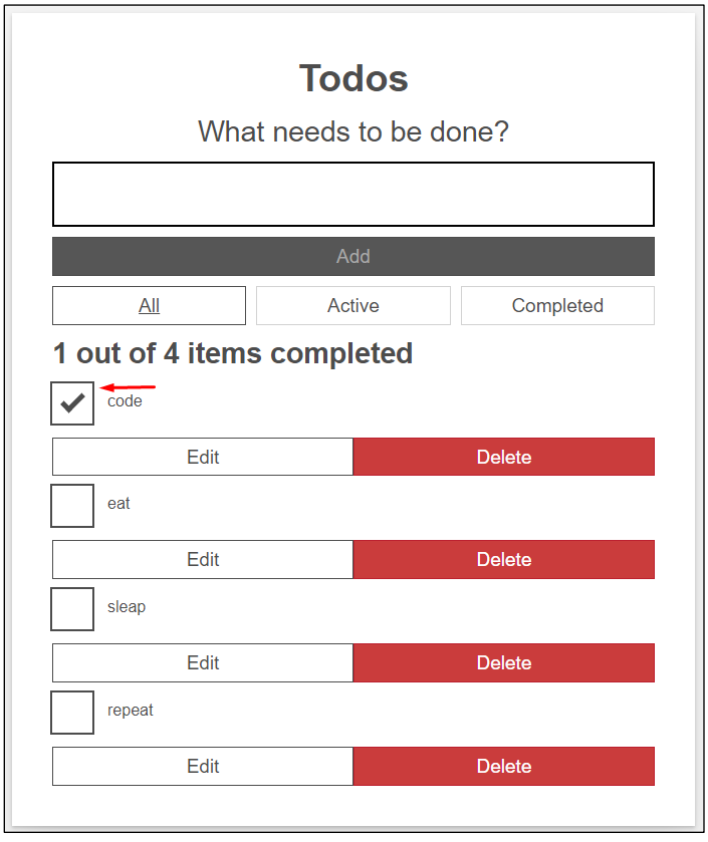


Рисунок 4.3 – Відмічення виконаних завдань

Кожен фреймворк був послідовно розроблений версією "Список справ", дотримуючись рекомендованих практик та архітектурних патернів кожного інструменту [11]. Після впровадження основного функціоналу, проводилась оптимізація коду для покращення продуктивності.

### 4.3.2 Оцінка продуктивності через тестування

Для виконання тестів продуктивності візуально ідентичних веб-сайтів на різних фреймворках був використаний інструмент Lighthouse, який є складовою частиною Chrome DevTools у браузері Google Chrome. Ми відкриваємо кожен сайт у Chrome та запускаємо Chrome DevTools. У вкладці Lighthouse вибираємо необхідні параметри для тестування, наприклад, емуляцію мобільного пристрою та тип мережі. Після цього ми запускаємо тест, натискаючи на кнопку "Generate report". Після завершення тесту Lighthouse формує звіт з оцінками та рекомендаціями щодо покращення продуктивності сайту. Проводимо цей процес для кожного сайту, розробленого на різних фреймворках. Оцінюємо звіти Lighthouse для кожного фреймворку, аналізуючи такі показники, як час завантаження сторінки, реакція на дії користувача та ефективність використання ресурсів. На основі цих даних ми робимо висновки щодо продуктивності кожного фреймворку, що дозволяє нам оцінити їхні переваги та недоліки в реальних умовах використання. Результати порівняння сайтів на мобільних пристроях наведено у таблиці 4.1.

Таблиця 4.1 – Порівняння роботи веб-сайтів на мобільних пристроях з використанням інструменту Lighthouse

| Фреймворк | Продуктивність (Performance) | Доступність (Accessibility) | Найкращі практики (Best Practices) | SEO |
|---|---|---|---|---|
| Vue | 95% | 78% | 100% | 64% |
| Svelte | 96% | 78% | 100% | 82% |
| React | 82% | 87% | 100% | 64% |

Результати порівняння веб-сайтів на десктопних пристроях показані у таблиці 4.2.

Таблиця 4.2 – Порівняння сайтів на десктопних пристроях у Lighthouse

| Фреймворк | Продуктивність (Performance) | Доступність (Accessibility) | Найкращі практики (Best Practices) | SEO |
|---|---|---|---|---|
| Vue | 100% | 78% | 100% | 78% |
| Svelte | 100% | 78% | 100% | 89% |
| React | 99% | 87% | 100% | 78% |

З аналізу даних таблиць можна зробити такі висновки:

Vue показує вражаючі результати продуктивності у всіх аспектах, але має найнижчу оцінку доступності серед усіх фреймворків.

Svelte вражає на мобільних пристроях з високою продуктивністю та SEO, але має відносно низький рівень доступності.

React показує стабільно високу продуктивність на десктопних пристроях і добрі результати за найкращими практиками, але рівень SEO залишається середнім без змін на різних пристроях.

Наступним кроком було використання програми WebPageTest. Після завершення тесту генерується звіт з ключовими метриками швидкості та візуальними показниками завантаження сторінки. Графік завантаження ресурсів надає детальний огляд кожного запиту та часу його обробки. Звіт також містить порівняльний аналіз і рекомендації щодо оптимізації. На основі цих даних вносяться поліпшення на сайті, після чого можливе повторне тестування для оцінки змін у продуктивності. Результати тесту представлені у таблиці 4.3.

Таблиця 4.3 – Порівняння веб-сайтів за допомогою WebPageTest

| Фреймворк | Time to First Byte (TTFB) | Start Render | First Contentful Paint (FCP) | Speed Index | Largest Contentful Paint (LCP) | Cumulative Layout Shift (CLS) | Total Blocking Time (TBT) | Page Weight |
|---|---|---|---|---|---|---|---|---|
| Vue | 0,743 s | 1,100 s | 1,070 s | 1,149 s | 1,451 s | 0,033 | 0,302 s | 194k |
| Svelte | 0,731 s | 2,300 s | 2,295 s | 2,300 s | 2,295 s | 0 | 0,000 s | 172k |
| React | 0,731 s | 1,100 s | 1,066 s | 1,246 s | 1,896 s | 0,014 | 0,770 s | 382k |

Згідно з результатами тестів, Vue видається лідером у швидкості відповіді сервера (Time to First Byte) та швидкості рендерингу (Start Render), що свідчить про ефективність ініціалізації застосунку. Svelte показує найповільніший час початку рендерингу та First Contentful Paint, що може свідчити про відносно повільніше завантаження контенту. React демонструє найкращий показник для Largest Contentful Paint, що свідчить про швидкість завантаження найбільшого елементу контенту, але має відносно високий Total Blocking Time. Найменша вага сторінки у Vue та Svelte може сприяти швидшому завантаженню сторінок, в той час як вища вага сторінки у React може бути пов’язана з більш великими бібліотеками або некомпактним кодом.

### 4.3.3 Перегляд популярності фреймворків у порівняльному аспекті

Спочатку було проведено порівняння у Google Trends, введено назву першого фреймворку для порівняння у поле пошуку. Додано додаткові фреймворки для порівняння, використавши опцію "Додати запит". Встановлено часовий діапазон на 5 років для аналізу трендів. Результати представлені на графіках у розділі 4.4. Отримані дані використано для оцінки загальної популярності та інтересу до кожного фреймворку.

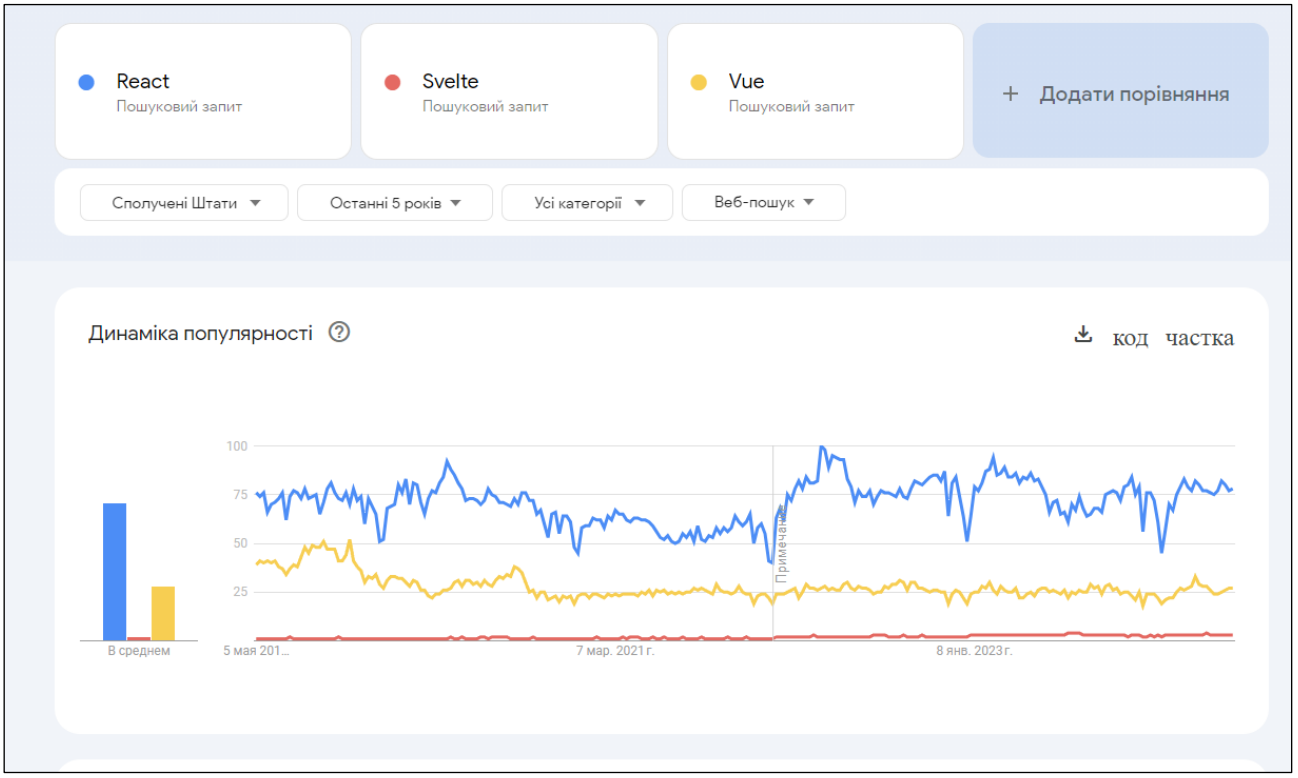


Рисунок 4.4 – Аналіз трендів у Google (за даними [20])

Згідно з даними Google Trends, React має найбільший рівень зацікавленості, який стабільно зберігається протягом останніх 5 років, з декількома піками, які можуть вказувати на особливі події або релізи. Vue і Svelte показують скромніші результати і обидва демонструють більш стабільний, але нижчий рівень зацікавленості порівняно з React. Svelte має найменший рівень зацікавленості серед цих фреймворків.

Наступним кроком буде порівняння за допомогою Stack Overflow Insights. Stack Overflow, як провідна платформа для професіоналів у сфері програмування, надає багатий набір даних про тенденції у запитах, відповідях, тегах та інших взаємодіях, які відображають реальний інтерес та виклики, з якими зіштовхуються розробники щоденно.

Аналізуючи ці дані, ми можемо отримати глибше розуміння популярності та прийняття фреймворків у порівнянні з іншими доступними інструментами. Це також дозволить нам зрозуміти рівень активності спільноти та їхню готовність ділитися знаннями та досвідом, що є важливим для новачків та досвідчених розробників, які обирають фреймворк для своїх майбутніх проектів. Це надасть

нам цінну інформацію для формування обґрунтованого висновку щодо кожного фреймворка, його місця в екосистемі сучасної веб-розробки та його потенціалу у майбутньому.

Інформацію про популярність фреймворків серед професійних розробників за 2023 рік представлено у таблиці 4.4

Таблиця 4.4 – Аналіз фреймворків на основі даних з Stack Overflow Insights

| Фреймворк | Відсоток використання |
|---|---|
| React | 42.87% |
| Vue.js | 17.64% |
| Svelte | 6.01% |

За даними у таблиці, React виявився найбільш популярним фреймворком серед переглянутих на Stack Overflow Insights, займаючи майже половину ринкової частки. Vue.js також є досить популярним, але поступається React, маючи меншу кількість використання серед розробників. Svelte, хоча і порівнювався, має значно менший відсоток використання, що свідчить про меншу популярність у порівнянні з іншими фреймворками.

### 4.3.4 Рекомендації щодо вибору

При виборі фреймворку для проекту важливо врахувати його розмір, вимоги до продуктивності, складність інтерфейсу та доступність розробників. React підходить для динамічних веб-застосунків з великою кількістю інтерактивних елементів, він пропонує велику гнучкість та має велику спільноту. Vue.js пропонує спрощений та прогресивний підхід, що робить його ідеальним для швидкої розробки та високої продуктивності у менших або середніх проектах. Svelte може бути корисним для тих, хто шукає конвенційно орієнтований фреймворк із суворим набором конвенцій та вбудованими рішеннями для швидкого старту проекту. Аналізуючи ці особливості та потреби проекту, можна зробити обґрунтований вибір фреймворку.

### 4.3.5 Пропозиції щодо подальших досліджень

Подальші дослідження в галузі веб-розробки можуть включати адаптацію сучасних веб-інтерфейсів і передових веб-додатків, вивчення впливу технологій штучного інтелекту на автоматизацію розробки, аналіз ефективності безсерверних архітектур і контейнеризації, оптимізацію активів і використання CDN для веб-додатків тощо. Можна розглянути дослідження стратегій підвищення продуктивності, а також оцінку безпеки веб-застосунків і розробку нових методів захисту від загроз. Також варто відзначити розробку та інтеграцію кросплатформенних рішень, підвищення доступності веб-контенту відповідно до WCAG і розробку інструментів для забезпечення якості та сумісності коду.

# ВИСНОВКИ

У заключенні дослідження методів рендерінгу інтерфейсу на основі React та Svelte важливо врахувати кілька ключових моментів. Гнучкість та розширюваність React вражають своєю варіативністю, проте це також призводить до великої кількості вибору вищезгаданих бібліотек та інструментів. З іншого боку, Svelte виглядає дуже ефективно завдяки компіляції на етапі розробки, що дозволяє генерувати менший та швидший код.

Оптимізація продуктивності є ключовим аспектом для забезпечення позитивного досвіду користувачів, зокрема, врахування метрик Core Web Vitals є критичним. І тут і React, і Svelte дозволяють розробникам ефективно покращувати ці метрики.

Під час розробки та тестування важливо користуватися різноманітними інструментами, такими як Chrome Dev Tools та Lighthouse, які допомагають виявляти проблеми та моніторити продуктивність.

У виборі фреймворку слід враховувати конкретні потреби проекту. React може бути більш відповідним для складних додатків, завдяки великій спільноті та екосистемі, тоді як Svelte вигідний для менших проектів завдяки простоті та ефективності.

Загально висновки свідчать про те, що обидва фреймворки мають свої переваги та можуть бути успішно використані в різних контекстах розробки веб-додатків.

## ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ


1. What is Front-End Web Development URL - https://www.romexsoft.com/blog/what-is-front-end-web-development/ (дата звернення: 10.12.2023).
2. Essential Guide to 3D Rendering: Process, Techniques URL - https://www.linkedin.com/pulse/essential-guide-3d-rendering-process-techniques-examples-marxent (дата звернення: 10.12.2023).
3. Webpage Rendering: How It Works URL - https://qarea.com/blog/webpage-rendering-how-it-works-tips-on-optimization (дата звернення: 10.12.2023).
4. Evaluation and comparison of widely used JavaScript frameworks URL - https://gearheart.io/articles/how-to-choose-the-best-javascript-framework-comparison-of-the-top-javascript-frameworks/ (дата звернення: 10.12.2023).
5. Архитектура React Native URL - https://dou.ua/forums/topic/34042/ (дата звернення: 10.12.2023).
6. Архитектура Svelte URL - https://netbasal.com/supercharge-your-svelte-state-management-with-akita-f1f9de5ef43d (дата звернення: 10.12.2023).
7. Архитектура Vue URL - https://dou.ua/lenta/articles/five-steps-to-vue-ssr/ (дата звернення: 10.12.2023).
8. Екосистема Vue URL - https://medium.com/js-dojo/vue-ecosystem-979773a9bf54 (дата звернення: 10.12.2023).
9. Показники Core Web Vitals URL - https://hostiq.ua/blog/ukr/core-web-vitals/ (дата звернення: 10.12.2023).
10. Оцінка юзабіліті освітніх сайтів: методи і технології / С. В. Тітов, О. В. Тітова // Вісник ХДАК / Зб. наук. праць. – Х: ХДАК., 2015. – Вип. 47. – С. 127-134.
11. Web-сайти органів місцевого самоврядування як складова впровадження е-урядування в Україні [Електронний ресурс] / С. В. Тітов, О. В. Тітова // Вісник Харківської державної академії культури . - 2013. - Вип. 39. - С. 146-155. - Режим доступу: http://nbuv.gov.ua/j-pdf/hak_2013_39_20.pdf

12. 28. Аналіз вебсайтів органів місцевої влади як механізму забезпечення права доступу до публічної інформації [Електронний ресурс] / Д. Е. Ситніков, О. В. Тітова // Вісник Харківської державної академії культури . - 2013. - Вип. 41. - С. 134-142. - Режим доступу: http://nbuv.gov.ua/j-pdf/hak_2013_41_18.pdf

13. Інформаційно-освітнє середовище навчального закладу: розвиток засобів і способів комунікаційної й інформаційної взаємодії [Електронний ресурс] / С. В.Тітов, О. В. Тітова // Вісник Харківської державної академії культури . - 2014. - Вип. 43. - С. 144-150. - Режим доступу: http://nbuv.gov.ua/j-pdf/hak_2014_43_20.pdf

14. Tvoroshenko, I., & Andrieieva, A. (2021). Development of web applications for remote learning of English.

15. Iryna, T., & Heorhii, M. (2021). Research of regression and modular testing of web applications. *Editorial Board*, 406.

16. Tvoroshenko, I. S., & Kuznetsov, M. (2021). About the role of testing in process of mobile application development.

17. Iryna, T., & Heorhii, M. (2021). TO THE QUESTION OF ANALYSIS OF EXISTING MECHANISMS OF WEB APPLICATION TESTING. Problems of modern science and practice, 1, 403.

18. Трет'якова, М. С. (2019). Інтелектуальні методи автоматизації тестування вебзастосунків.

19. Динаміка популярності React, Vue.js, Svelte Google Trends URl: https://trends.google.com.ua/trends/explore?date=today%205y&geo=US&q=React,Svelte,Vue&hl=ua (дата звернення 13.03.2024)

20. Web frameworks and technologies. URL: https://survey.stackoverflow.co/2023/#most-popular-technologies-webframe-prof (дата звернення 13.03.2024)

21. Shubin I., Kozyriev A., Liashik V., Chetverykov G. (2021) Methods of Adaptive Knowledge Testing Based on the Theory of Logical Networks.

22. Shubin I., Kyrychenko I., Goncharov P., Snisar S. (2017) Formal representation of knowledge for infocommunication computerized training systems

23. Vysotska, V. , Shubin, I. , Mezentsev, M. , Kobernyk, K. , Chetverikov, G. (2024) Ukrainian Big Data: The Problem Of Databases Localization