ВСТУП

На теперішній час стрімке зростання кількості засобів розробки програмних систем досить суттєво ускладнює життя програмістів, оскільки вимушує їх постійно знаходитись у пошуку нових та більш ефективних засобів розробки. Збільшення кількості бібліотек та фреймворків JavaScript, що були розроблені останнім часом, вимагає від розробників порівнювати їх ефективність. Розробникам не вистачає результатів порівняльного аналізу таких бібліотек між собою, а також – відсутність стандартизованого процесу їх оцінювання для отримання впевненості, що їх структура відповідає потребам та масштабу проекту.

За останні роки JavaScript стає однією з основних мов програмування на веб-сайтах, що призвело до зростання його популярності та інтересу до його вивчення, коли йдеться про використання його фреймворків та бібліотек. Привабливість JavaScript обумовлена тим, що пропонує безліч фреймворків та бібліотек, більшість з яких створено самими розробниками. Використання фреймворків та бібліотек суттєво спрощує процес розробки клієнтських веб-додатків завдяки чистому та структурованому коду.

В рамках проходження науково-дослідної практики було проведено дослідження методів порівняння і оцінювання різних платформ та бібліотек, що дозволить визначити, яка з них більш відповідає їх потребам та може бути більш ефективною для практичної реалізації проекту. Також такий підхід дозволить більш ефективно розробляти структуру проекту, враховувати існуючий досвід та переваги при розробці програмних систем.

Дослідження, що були проведені під час проходження науково-дослідної практики, базуються на порівняльному аналізі ефективності додатків, призначених для планування часу (ToDo-списки), розробку яких було виконано з використанням фремворків React та Vue. Під час дослідження фреймворк та бібліотеки оцінювались за декількома параметрами, до яких відносяться якість

документації, якості підтримки, частоти оновлення тощо. В результаті дослідження було виявлено домінуючі фактори для Vue та React, що стало підґрунтям для проведення порівняльного аналізу з метою виявлення відмінностей та подібностей між ними. В результаті були виявлені домінуючі фактори, а саме: React має детальну документацію, значну підтримку з боку спільноти, гнучкість у використанні зовнішніх бібліотек та може використовуватись як професіоналами, так і новачками у розробці JavaScript-додатків. На відміну від нього, Vue вимагає дотримання чітких правил використання цього фреймворку, що є досить ефективним у першу чергу для новачків.

Запропонований в результаті проходження науково-дослідної практики процес оцінювання дає інструментарій для виявлення розбіжностей між структурами, які розроблено з використанням різних інструментів програмування. Дослідження вмісту різних фреймворків вимагає від програмістів, щоб вони використовували їх у правильних умовах та цілях. Знаючи про відмінності фреймворків не буде витрачатись час на адаптацію до відповідного фреймворку або бібліотеки, які можуть не відповідати критеріям проекту. Ще однією з переваг є опис факторів, які впливають на фреймворк, та той факт, що розробник повинен мати уявлення про рівень необхідних знань та складність фреймворку, перш ніж обрати його.

Таким чином можна сказати, що метою цієї науково-дослідної роботи є дослідження методів аналізу ефективності використання фремворків React та Vue при розв’язанні задач планування часу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Концепції розробки веб-орієнтованих програмних систем

Сучасні технології розробки веб-додатків розвивалися дуже швидко протягом останніх кількох років. Традиційна модель веб-програми — це багатосторінкова програма, яка від початку домінувала у світі WWW. Одним з найважливіших недоліків традиційних веб-додатків (багатосторінкових додатків) є погана чутливість: коли користувач змінює сторінку, браузеру потрібен час, щоб отримати новий HTML-документ із сервера. Внутрішня обробка сервера також може тривати деякий час. На теперішній час користувацькі пристрої у світі WWW постійно мають більшу обчислювальну потужність і великий обсяг пам’яті. Завдяки цим фактам більшу частину логіки та обробки програм можна передати кінцевому пристрої, наприклад, настільному ПК або мобільному телефону. Це звільнить сервер від використання великої кількості ресурсів кожного клієнта. До цієї концепції краще підходить модель так званої односторінкової програми (SPA). Оскільки останнім часом швидкість передачі даних також покращилася, модель SPA пропонує значне поліпшення досвіду користувача. У SPA весь вміст програми завантажується відразу, тому початкове завантаження сторінки зазвичай довше, але останні зміни сторінки відбуваються миттєво.

Традиційна архітектура веб-додатків складається з клієнта та веб-сервера. Клієнтом може бути, наприклад, браузер на настільному ПК або мобільному пристрої. Клієнт відправляє запити HTTP на веб-сервер, який потім виконує деякі функції на основі запиту і, можливо, повертає деякий результат. На рис.1.1 зображено традиційну модель передачі даних веб-програми. У цій моделі клієнт браузера запитує веб-сторінку у веб-сервера, а веб-сервер, у свою чергу, запитує деякі дані з бази даних і, нарешті, повертає HTML-документ разом із стилями CSS в браузер користувача.

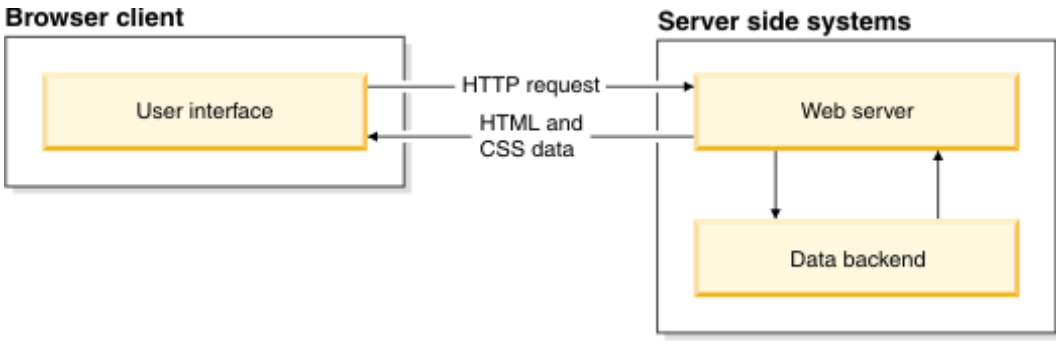


Рисунок 1.1 – Традиційна модель клієнт-серверної взаємодії

Багатосторінковий додаток зазвичай складається з декількох сторінок. Коли користувач натискає посилання на сторінці, він перенаправляє браузер на інший HTML-файл, який розміщено на сервері. Одночасно завантажуються файли таблиць стилів, якщо вони ще не збережені в кеші браузера. Крім повернення HTML-документів, сервер має традиційно й інші завдання. Сервер може автентифікувати користувача, вносити зміни до бази даних або, наприклад, надсилати електронні листи.

В архітектурі веб-додатків під маршрутизацією розуміють процес, в якому кожна окрема URL-адреса відповідає певній сторінці або вмісту. У моделі багатосторінкового додатка маршрутизація є частиною програми, оскільки сторінки також розділені у файловій системі. Використовуючи URL-адреси та маршрути, браузери можуть створювати закладки, ділитися посиланнями та переміщуватися веб-сайтам за допомогою кнопок «назад» та «вперед».

Деяка обробка може відбуватися і на боці клієнта. Обробка в браузері зазвичай використовує JavaScript і містить такі завдання, як зміна моделі DOM, створення інтерактивних візуалізацій або відображення попереджень на екрані. HTML DOM (об’єктна модель документа) – це стандартна деревоподібна структура або модель, яку браузер використовує для доступу до різних елементів. Існує багато видів налаштувань, але сама основа поділу задач зазвичай однакова у традиційних багатосторінкових веб-додатках.

У міру того, як односторінкові програми стають популярними і все більше обробки зосереджено на стороні клієнта, мови програмування зовнішнього

інтерфейсу (наприклад, JavaScript) також мають розвиватися. Середовище програм, що визначається як великий повторно використовуваний

Набір бібліотек для реалізації основної структури програми також стала тенденцією в розробці JavaScript. Фреймворки JavaScript мають розширити можливості розробника та спростити розробку JavaScript, надаючи готові оптимізовані функції для більш складних функцій. З погляду компанії, можна використовувати деяку структуру JavaScript поверх власного JavaScript, оскільки це прискорює процес розробки інтерфейсу у довгостроковій перспективі.

Поняття фреймворку тісно пов’язано з самим процесом розробки веб-орієнтованого програмного додатку – фактично це каркас веб-застосунку. З цієї причини розробник програмної системи повинен будувати її архітектуру відповідно до чітко визначеної логіки. Зазвичай середовище розробки надає відповідний інструментарій для реалізації окремих подій, побудови сховищ і з’єднання даних. Фактично, проводячи аналогії з машинобудуванням, майбутня машина безпосередньо залежить від принципів реалізації каркасу, кузову і двигуна. Змінюючи конфігурацію, існує можливість додавати, знімати або замінювати певні деталі за умови, що автомобіль залишається у працездатному стані.

Каркаси знаходяться на більш високому рівні абстракції, у порівнянні з бібліотеками, і дозволяють спростити процес розробки веб-орієнтованих програмних додатків.

Інструментальні засоби завжди значно спрощують процес розробки програм. Наприклад, досить ефективним є використання препроцесору Sass замість чистого CSS, оскільки він забезпечує значну кількість додаткових засобів програмування, не пов’язані безпосередньо зі стилями подання даних – це використання циклів, функцій, локальних змінних тощо. Разом з тим існує суттєва проблема – браузери не «розуміють» синтаксис Sass/SCSS, тому код необхідно перекласти на CSS.

Одним з найбільш популярних фреймворків з готовими стилями і скриптами є Bootstrap. Цей фреймворк вимагає перезаписати необхідні класи і

атрибути елементів HTML. Завдяки цьому досить легко за допомогою Bootstrap створювати мобільні проекти – це дозволяє досить легко налаштовувати будь-яку сторінку та відображати її на будь-якому пристрої. Особливість використання фреймворку Bootstrap полягає у тому, що це CSS framework – тобто набір файлів з готовим кодом, який підключається до html-сторінки в головному розділі, після чого може використовуватися елемент з цієї структури.

Аналогічно тому, як використовується Bootstrap, застосовують і JavaScript фреймворки – фактично це інструментарій для побудови значної кількості веб-орієнтованих, мобільних та настільних додатків на Javascript. Використання розробниками js-фреймворків обумовлено у першу чергу тим, що їх доцільно застосовувати там, де неможливо або складно виконувати програмування звичайними засобами розробки, особливо для веб-додатків – найбільше поширення вони знайшли при реалізації Single Page Applications.

1.2 Шаблони веб-орієнтованих систем

Особливістю Single Page Applications (SPA) слід вважати той факт, що він вміщує на одній сторінці увесь код – як HTML, так і JavaScript, та CSS, які завантажуються або одночасно із завантаженням сторінки, або динамічно – довантажуються у відповідь на дії користувача. Ще однією з особливостей SPA-додатків є той факт, що сторінка не підлягає оновленню і не перенаправляє користувача на інші сторінки. Частіш за все взаємодія із застосунком реалізується через динамічний зв’язок з веб-сервером.

SPA-додаток працює зовсім по-іншому з точки зору передачі або маршрутизації даних у порівнянні з багатосторінковим додатком. Коли клієнт запитує веб-сторінку, він отримує повний код веб-сайту. Отриманий контент містить лише частково завершений HTML-документ, який динамічно доповнюється на стороні клієнта, зазвичай, за допомогою JavaScript. Іншими

словами, видиме подання створюється серією функцій JavaScript, які доповнюють HTML DOM під час виконання. Жодних додаткових запитів не потрібно, але весь видимий контент створюється та модифікується за допомогою JavaScript. Таким чином, наприклад, зміна сторінки SPA насправді не є зміною сторінки, а швидше заміною вмісту поточної сторінки новим вмістом.

JavaScript використовується для розробки як повних веб-сайтів, так і окремих функціональних модулів, до яких відносяться онлайн-інструменти. Зазвичай, повні кадри більш підходять саме для розв'язання першої задачі, а для інших доцільно використання більш світлих кадрів або бібліотек.

В рамках застосування JavaScript однією з вимог сучасних технологій програмування є дотримання моделі обробки даних. Найбільш поширеними моделями на теперішній час вважаються MVC (Model-View-Controller), Model-View-Presenter (MVP) та Model-View-ViewModel (MVVM) [1]. Розглянемо їх відмінності та порівняємо між собою, визначивши для цього критерії. З цією метою визначимо наступні ознаки гарної архітектури, які і будемо використовувати у якості критеріїв:

– збалансований розподіл обов'язків між сутностями із жорсткими ролями;

– тестування;

– простота використання та низька вартість обслуговування.

Загальна спрямованість концепцій MVC, MVP та MVV полягає у тому, що у будь-яких додатках, побудованих за даними моделям, має забезпечуватися поділ даних від зовнішнього інтерфейсу. Ці концепції передбачають призначення сутностей додатку за однією з трьох категорій:

Models – відповідає за домен або шар доступу до даних, за допомогою яких виконується маніпулювання даними;

Views – відповідає за рівень уявлення (GUI);

Controller/Presenter/ViewModel – посередник між Model та View; загалом відповідає за зміни Model, реагуючи на дії користувача, що виконані на View, та оновлює View, використовуючи зміни з Model.

Шаблон MVC. Controller є посередником між View та Model, тому ці компоненти не мають уявлення про існування один одного (рис.1.2). Саме тому Controller важко перевикористати. У цій концепції Controller тісно пов’язані з життєвим циклом View, тому досить важко зробити висновок, що він є окремою сутністю. Є можливість розвантажити частину бізнес-логіки та перетворення даних з Controller на Model, але, коли справа доходить до відвантаження роботи у View, існує не так багато варіантів.

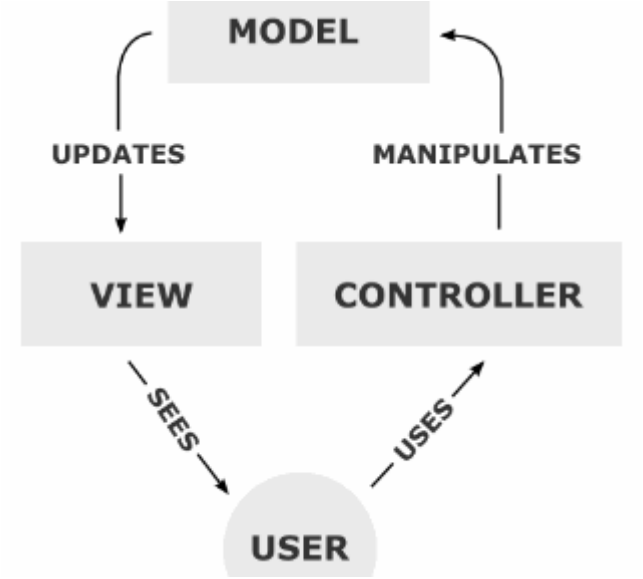


Рисунок 1.2 – Шаблон MVC

Частіш за все вся відповідальність View полягає у тому, щоб відправити дії до Controller. В результаті View Controller стає делегатом і джерелом даних, а також – місцем запуску та скасування серверних запитів і, загалом, усього чого завгодно.

Оцінемо MVC з погляду на критерії архітектури, що були визначені вище:

– розподіл: View та Model насправді розділені, але View та Controller тісно пов’язані між собою;

– тестування: через поганий розподіл можливо тестуванню підлягає тільки Model;

– простота використання: найменша кількість коду серед інших патернів програмування, крім того його досить легко може підтримувати навіть недосвідчений розробник.

Шаблон MVP. MVP (рис.1.3) є похідною від MVC, яка використовується в основному для побудови інтерфейсів користувача. Елемент Presenter в даному шаблоні бере на себе функціональність посередника (аналогічно тому, як використовується Controller в MVC) і відповідає за управління подіями інтерфейсу користувача так само, як в інших шаблонах зазвичай відповідає уявлення.

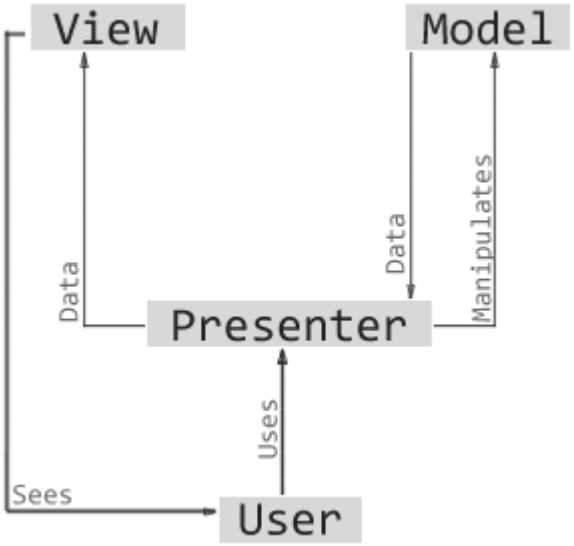


Рисунок 1.3 – Шаблон MVP

Шар Presenter тут є абстракцією над View, який управляє відображенням, але не містить деталі реалізації відображення. Також, Presenter є посередником між даними та відображенням.

Виділимо критерії, що притаманні архітектурі MVP:

– розподіл: більша частина відповідальності розділена між Presenter та Model, а View нічого не робить;

– тестування: відмінна – можемо перевірити більшу частину бізнес-логіки завдяки бездіяльності View;

– простота використання: кількість коду вдвічі більша у порівнянні з MVC, але, водночас, ідея MVP дуже проста.

Шаблон MVVM. Цей шаблон дизайну архітектури додатків був представлений у 2005 році Джоном Госсманом (John Gossman) як модифікація шаблону Presentation Model. Цей шаблон орієнтований на сучасні платформи розробки.

В рамках цього шаблону ViewModel викликає зміни у Model і самостійно оновлюється з вже оновленим Model. І оскільки зв'язування відбувається між View та ViewModel, то перша, відповідно, також оновлюється (рис.1.4).

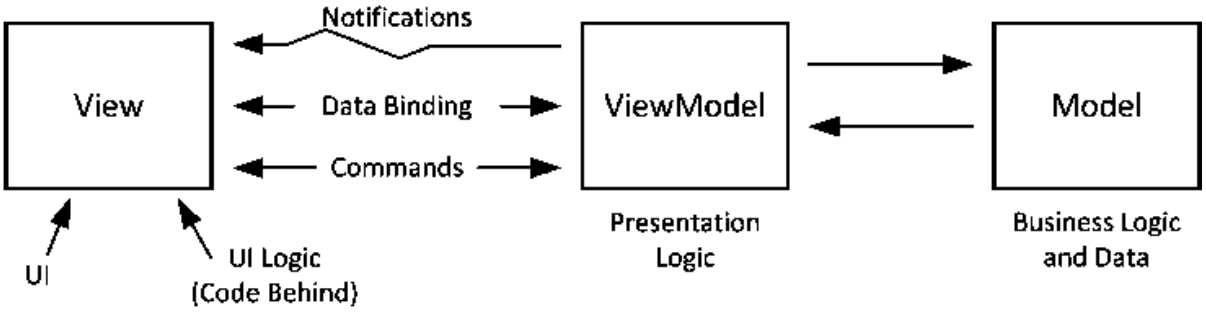


Рисунок 1.4 – Шаблон MVVM

Виділимо критерії, що притаманні архітектурі MVVM:

– розподіл: MVVM View має більше обов'язків, ніж View з MVP. Тому що перша оновлює свій стан з ViewModel за рахунок встановлення зв'язування, тоді як друга – спрямовує всі події в Presenter і не оновлює себе (це робить Presenter);

– тестування: ViewModel нічого не знає про уявлення – це дозволяє досить легко тестувати її. View також можна тестувати;

– простота використання: у реальному додатку, де необхідно буде спрямовувати всі події з View в Presenter та оновлювати View вручну, в MVVM буде набагато менше коду (якщо використовувати зв'язування).

MVVM є дуже привабливим патерном, тому що він поєднує в собі переваги вищезгаданих підходів і не вимагає додаткового коду для оновлення View при зв'язуванні на стороні View, тестування також знаходиться на хорошому рівні.

Разом з тим при проектуванні програмної системи, яку будемо розробляти в рамках виконання кваліфікаційної роботи, будемо дотримуватись концепції MVC.

1.3 Можливі шляхи вирішення задачі

У SPA вся програма, включаючи всі його сторінки, завантажується за одне початкове завантаження сторінки. Перша відповідь сервера містить повний веб-сайт, HTML, JavaScript та таблиці стилів. Через це обсяг даних у першому запиті, звичайно, великий, але час завантаження все ще досить малий через поточні високі швидкості широкосмугової технології.

Однак, деякі дані необхідно отримати після початкового завантаження сторінки. Деяку частину сторінки, наприклад, у стрічках соціальних мереж або рекламних об’явах, може знадобитися оновити, наприклад, через дії користувача. В односторінкових програмах такі дані витягуються за допомогою запитів Ajax. Ajax-запити – це асинхронні методи, які дозволяють обмінюватися даними із сервером без перезавантаження сторінки. На рис.1.5 верхня стрілка відображає перший запит, а нижня відображає всі інші запити. Як правило, дані передаються як об’єкт JSON, який є поширеним форматом об’єктів JavaScript [3].

Логіка на стороні сервера зазвичай набагато простіша в моделі односторінкового додатка і вимагає меншого обсягу обробки та логіки. Сервер досить часто є лише веб-службою без збереження стану або (в деяких випадках) з повним збереженням стану. RESTful веб-служби досить поширені серед односторінкових додатків. Веб-сервіси RESTful (репрезентативна передача стану) – це концепція уніфікованих операцій без збереження стану доступу до ресурсів. Різні методи HTTP підключаються до певних ресурсів, і програма запитує ці дані окремо (рис.1.5).

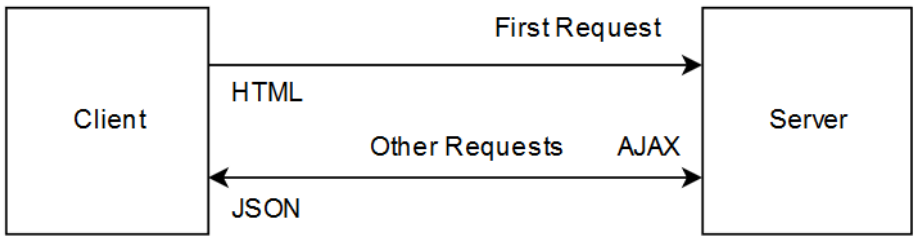


Рисунок 1.5 – Передача даних в SPA-додатках

У випадку односторінкових програм вся програма спочатку має лише одну URL-адресу. З погляду файлової системи це пов’язано з тим, що вихідний HTML-документ усієї програми включений в один файл. Функції маршрутизації, такі як створення закладок або навігація, повинні бути реалізовані програмно. Використовуються два поширені типи маршрутизації: статична маршрутизація та динамічна маршрутизація.

Статична маршрутизація найчастіше використовують у односторінкових додатках. У статичній маршрутизації маршрути визначаються як частина ініціалізації програми. Перш ніж відбудеться будь-який рендеринг, маршрути вже відомі додатку. На рис.1.6 представлена схема потоку даних моделі статичної маршрутизації.

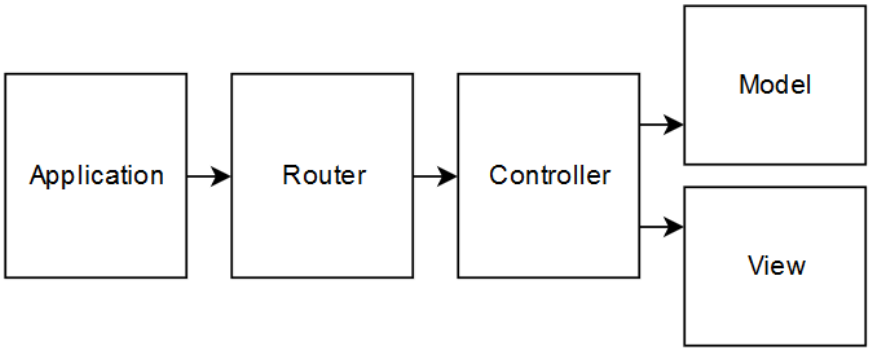


Рисунок 1.6 – Схема потоку даних моделі статичної маршрутизації

На рис. 1.6 маршрутизатор розташовано між додатком та контролерами. Це означає, що маршрути відомі контролеру на етапі ініціалізації і не можуть бути згодом змінені. Існує кілька відомих недоліків використання статичної

маршрутизації, до яких відноситься той факт, що оскільки маршрути статичні, то змінити маршрут під час виконання неможливо.

Динамічна маршрутизація – ще один варіант маршрутизації SPA. При динамічній маршрутизації маршрут може бути змінено в залежності від переважаючого середовища на стороні клієнта. На рис.1.6 представлена модель динамічної маршрутизації, в якій маршрутизатор розміщується поруч з контролером.

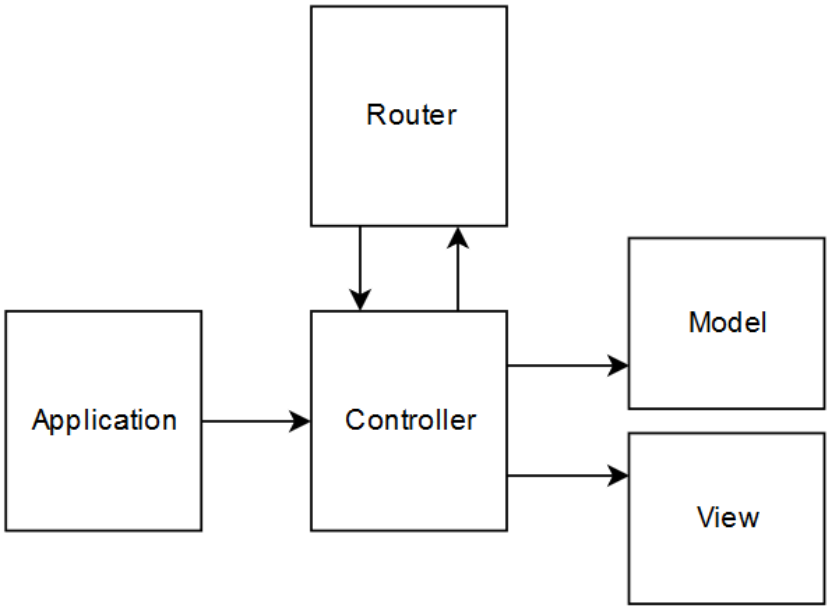


Рисунок 1.7 – Потоки даних при динамічній маршрутизації

З рис.1.7 зрозуміло, як контролер постійно працює з маршрутизатором. Саме тому він може передавати йому інформацію та отримувати нову інформацію про маршрут. Таким чином, динамічна маршрутизація дозволяє змінювати маршрути в будь-який час, залежно від переважаючого середовища клієнта, наприклад, розміру вікна браузера [8].

В SPA-моделі програми одночасне завантаження всього вмісту та динамічне створення сторінки значно підвищує загальну швидкість відгуку програми. Браузеру не потрібно завантажувати додатковий контент із сервера, коли користувач змінює сторінки, не потрібно перезавантажувати сторінки, тому модель SPA також пропонує офлайн-можливості. Це робить зовнішній вигляд

односторінкових програм більш схожим на настільну програму. Загальна продуктивність в односторінкових додатках також вища, оскільки ресурси обробки розподіляються більш ефективно. Клієнтські пристрої зазвичай мають значну обчислювальну потужність, яка використовується у моделі SPA.

Одним із факторів, що уповільнюють розробку моделі односторінкових програм, є пошукова оптимізація. Донедавна пошукові системи не могли проіндексувати односторінкові сайти, що призводило до поганої видимості сайту. У жовтні 2015 року Google було внесено деякі покращення, почавши індексувати динамічні сторінки [4]. Модель односторінкового застосунку також має деякі проблеми з безпекою з точки зору атак міжсайтового скриптингу (XSS). Шкідливий код JavaScript відносно легко впровадити на веб-сайті SPA [5]. Однак цього можна запобігти, але це потребує особливої уваги з боку розробника. Ще один недолік односторінкових програм полягає в тому, що вони також повністю залежать від підтримки JavaScript у браузері.

1.4 Постановка задачі

На основі проведеного аналізу предметної галузі можна сформулювати постановку задачі та визначити необхідні вимоги до об’єкту дослідження.

Метою кваліфікаційної роботи магістра є дослідження методів порівняння ефективності використання популярних JavaScript-фреймворків при розробці односторінкових веб-додатків.

Система, що буде створена в рамках дослідження, повинна надавати користувачеві можливість виконувати задачі тайм-планування. Розробка систем буде проводитись з використанням різних фреймворків, але забезпечувати однаковий функціонал.

Головною проблемою та завданням побудови цієї системи є порівняння та аналіз продуктивності систем, що розроблені з використанням різних фреймворків.

Для досягнення мети дослідження необхідно вирішити ряд локальних задач [9], а саме:

– дослідити технологічні рішення, які використовуються у поширених JavaScript-фреймворках;

– проаналізувати та виявити методи оцінювання ефективності використання фреймворків для побудови веб-орієнтованих додатків;

– дослідити шляхи досягнення кращої продуктивності у динамічних та частково-динамічних веб-додатках;

– виконати дослідження продуктивності веб-додатку на реальному прикладі реалізації архітектурно однакового ToDo-додатку за допомогою різних фреймворків.

– зробити висновки щодо ефективності досліджуваних фреймворків для досягнення мети дослідження.

У ході розробки програмного забезпечення буде розроблено алгоритм реалізації додатку для планування часу. Цей алгоритм планується уніфікувати для можливості його реалізації за допомогою різних фреймворків без внесення суттєвих відмінностей в практичну реалізацію.

Технології, які будуть використанні при дослідженні програмного забезпечення:

– мова програмування JavaScript;

– фреймворки React.js та Vue.js;

– браузер Google Chrome v. 113.0.5672.64.

2 ДОСЛІДЖЕННЯ МЕТОДІВ РОЗВ'ЯЗАННЯ ЗАДАЧІ

2.1 Аналіз та порівняння поширених фреймворків JavaScript

Популярність JavaScript зростає день у день. Кількість і якість технологій та фреймворків, які побудовані на цій мові, постійно зростає та покращується open-source спільнотою. Але існує значна кількість проблем, пов'язаних з проектуванням та розробкою веб-додатків. Серед них найбільш суттєвими є наступні:

– ускладнена масштабованість коду;

– невисока швидкість розробки та впровадження;

– нехтування принципами SOLID, що веде до повторення коду;

– чутливість UI на стороні клієнта;

– кількість ресурсів, які використовуються додатком.

Одним з ефективних шляхів подолання цих недоліків стала розробка фреймворків (серед них – Vue, AngularJS, React), які частково вирішують ці проблеми, але роблять це кожний по-своєму. Ці фреймворки є каркасом для окремих сторінок веб-додатку, що дозволяє розробникам менше приділяти увагу структурі коду або його підтримці у той час, коли вони зосереджені на створенні складних елементів інтерфейсу.

Серед переваг використання JavaScript фреймворків слід відзначити [5]:

а) ефективність – проекти, реалізація яких раніше зайняла б багато часу та умістилась би у сотні рядків коду, зараз можуть бути реалізовані набагато швидше з добре структурованими готовими шаблонами та функціями;

б) безпека – кращі JavaScript фреймворки мають фірмову систему безпеки та підтримуються великою спільнотою, члени якої (да і прості користувачі також) виступають у ролі тестувальників;

в) витрати – більшість фреймворків з відкритим кодом та безкоштовні. За рахунок того, що вони дозволяють програмістам підвищити швидкість розробки

користувальницьких рішень, кінцева вартість розробки веб-додатків буде нижчою.

Разом з тим, отримуючи всі переваги використання фреймворків, розробники вимушені жертвувати продуктивністю та кількістю споживаних ресурсів. Для невеликих веб-додатків це не є критичним, але у разі розробки масштабного проекту, який розрахований на використання мільйонами людей, то кожен мегабайт пам’яті і кожна мілісекунда, що витрачена на запуск і роботу фреймворку, є вкрай важливими.

На теперішній час React, Angular та Vue є одними з найпопулярніших фреймворків JavaScript. Швидка поява значної кількості нових фреймворків примушує розробників обирати кращі серед них. Саме тому доцільною є спроба порівняти JavaScript фреймворки за фактичними показниками, які є критичними для розробників [2]. До таких показників слід віднести (рис.2.1):

– рендерінг: відображення кінцевого результату;

– архітектура компонентів;

– спрямованість та класи залежностей;

– зворотня сумісність;

– документація та підтримка.

Name	Type	Shadow DOM EcmasScript 6+	Relative Popularity	Difficulty ofLearning
React	Library	Supported	*****	*****
Angular	Framework	Supported	***	*****
Ember	Framework	Supported	*	*****
Vue	Library	Supported	**	***
Backbone	Framework	Supported	*	***

Рисунок 2.1 – Популярні фреймворки

Середовище JavaScript повинно відповідати декільком вимогам [6].

По-перше, сучасні інтерфейсні JavaScript-фреймворки повиння відповідати специфікації веб-компонентів. В сучасній клієнтській розробці найбільшу увагу привертає створення користувацьких HTML-елементів.

По-друге, надійний JavaScript-фреймворк повинен мати власну екосистему. Готові рішення призначені для розв’язання різних проблем розробки на стороні клієнта таких як маршрутизація, управління станом програми, взаємодія із серверною частиною тощо. Angular, Backbone, React, Vue та Ember відповідають останній специфікації JavaScript ES6+; також вони всі мають власні екосистеми.

2.1.1 React

React - це бібліотека JavaScript для розробки інтерфейсів користувача (UI), що розробляється і підтримується компанією Facebook. Вона надає можливість розробникам будувати ефективні, масштабовані та зручні в управлінні веб-додатки за допомогою компонентної моделі.

До основних особливостей React слід віднести (див. рис.2.2):

– Компонентна модель: React дозволяє розробникам будувати веб-інтерфейси у вигляді окремих компонентів, які можуть бути повторно використані та композиційно об’єднані в більш складні інтерфейси. Це дозволяє розробникам використовувати підхід "розділяй та володарюй" (separation of concerns) для кращої організації коду та підтримки швидкої розробки;

– Віртуальний DOM: React використовує віртуальний DOM, що є ефективним механізмом оновлення інтерфейсу без необхідності маніпулювати реальним DOM. Це дозволяє забезпечувати високу продуктивність веб-додатків з меншою кількістю зайвих операцій з DOM;

– Односторонній потік даних: У React використовується односторонній потік даних, що дозволяє визначити одну "правдиву" версію даних та автоматично оновлювати інтерфейс відповідно до цих даних;

– Використання JSX: JSX – це розширення синтаксису JavaScript, що використовується в React для опису вигляду компонентів. Він дозволяє розробникам використовувати HTML-подібний синтаксис в JavaScript коді, що робить код більш зрозумілим та легким для розробки.

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

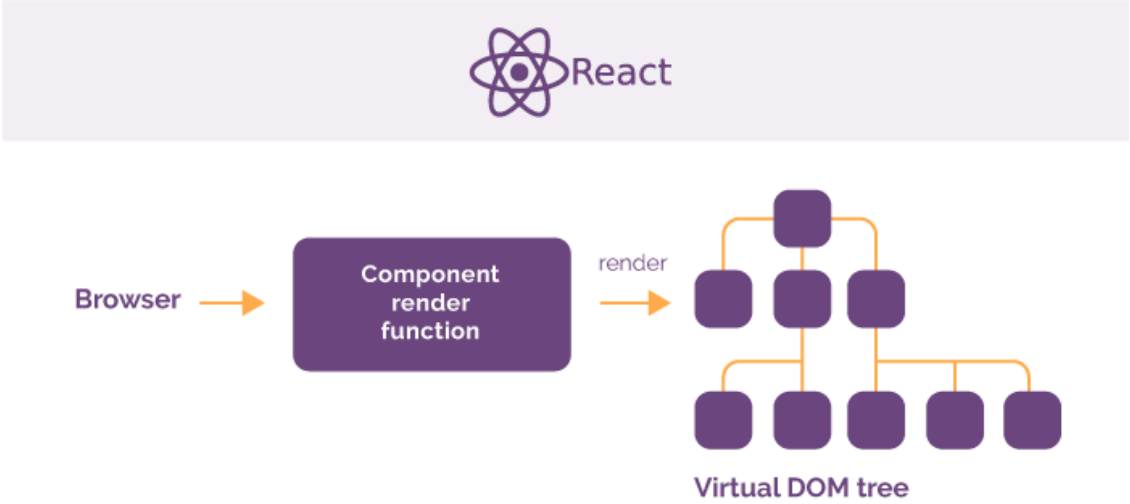


Рисунок 2.2 – React

React поділений на декілька компонентів. Один файл компоненти містить бізнес-логіку та HTML-розмітку (фактично це розмітка JSX, перетворена на функції JavaScript). Для зв’язків між компонентами можна використовувати або Flux, або аналогічну JS-бібліотеку. У React також з’явилися спеціальні об’єкти – state та props. Використовуючи об’єкти стану та реквізиту, ви можете просто передавати дані з компоненти у макету (використовуючи об’єкт стану) або від батьківського компонента – до дочірньої компоненти (використовуючи об’єкт реквізиту) [5].

React має багато додаткових інструментів для досягнення високої гнучкості. Наприклад, замість Flux можна вибрати MobX, Redux, Fluxy, Fluxible або RefluxJS. І цей список бібліотек управління станом для React не вичерпний. Теж саме стосується і бібліотек HTTP: React може працювати з jQuery AJAX, fetch API, Superagent і Axios.

Незважаючи на те, що гнучкість є його головною перевагою, React саме через його гнучкість має проблеми: коли доводиться вибирати з багатьох додаткових бібліотек, то досить часто виникає дилема – що саме слід використовувати зі складу React.

React намагається встановити галузеві стандарти. Наприклад, Redux вважається найкращою та єдиною бібліотекою для програм React корпоративного рівня [10]. У той же час Redux може серйозно знизити продуктивність розробки, оскільки Redux ускладнює роботу, коли необхідно впровадження нової функції, а в результаті необхідно змінити занадто багато речей у всьому додатку, що вимагає створення власного робочого процесу із React.

2.1.2 Angular 2+

Angular – це відкритий фреймворк для розробки веб-додатків, розроблений компанією Google. Angular 2+ є наступним поколінням AngularJS (Angular 1.x) і має ряд особливостей [5, 10]:

– Компонентна архітектура: Angular 2+ базується на компонентній архітектурі, де веб-додаток розбивається на незалежні компоненти, кожен з яких відповідає за свою частину інтерфейсу користувача. Компоненти можуть мати внутрішній стан, взаємодіяти між собою та зовнішніми сервісами;

– TypeScript: надає статичну типізацію, об’єктну орієнтованість та інші розширення JavaScript. Це робить розробку Angular-додатків більш розумними та масштабованими;

– Декларативний синтаксис: Angular 2+ використовує декларативний синтаксис для опису інтерфейсу користувача за допомогою HTML-шаблонів, що дозволяє розробникам описувати бажаний стан інтерфейсу у декларативному стилі;

– Двостороннє зв’язування даних: Angular 2+ дозволяє встановлювати двостороннє зв’язування даних між компонентами та їх шаблонами, що дозволяє автоматично оновлювати відображення при зміні стану;

– Модульність: Angular 2+ дозволяє розробникам створювати модулі, що допомагає організовувати код у логічні блоки, використовувати ліниве завантаження модулів та забезпечувати більшу реюзабельність коду.

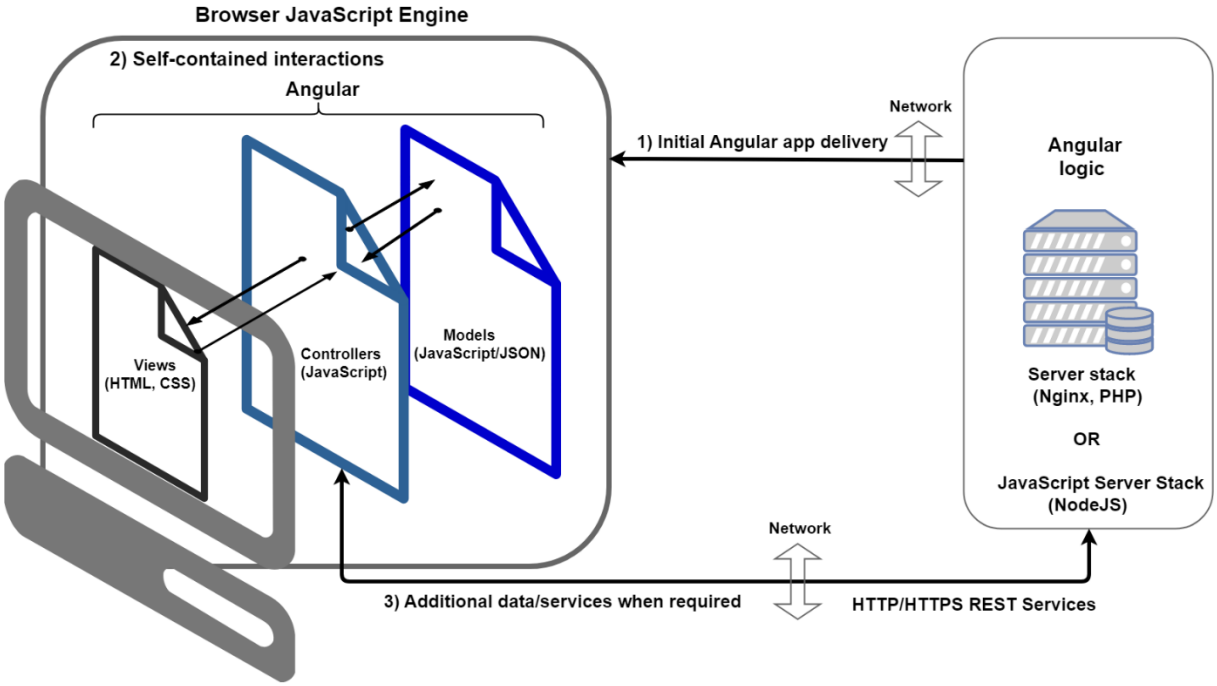


Рисунок 2.3 – Angular 2+

Головною відмінною рисою Angular став перехід від архітектури Model-View-Whatever (одна з форм шаблону MVC) до архітектури, заснованої на компонентах – тим самим Angular став модульним. Якщо раніше в основний

HTML-файл можна було вставити посилання на бібліотеку AngularJS, то тепер з’явилась можливість встановлювати лише окремі модулі, а не всі разом: можна легко відмовитися від встановлення деяких модулів Angular, наприклад @angular/forms або @angular/http, якщо вони не потрібні. Під час розробки програми за допомогою Angular необхідно поділити її на декілька компонентів із вкладеними або дочірніми компонентами. Кожен компонент Angular поділений на три файли: бізнес-логіка для компонента записується в основний файл, тоді як макет та стилі, пов’язані з цим компонентом, записуються у два інших файли.

Крім компонентів в Angular присутні сервіси – структури застосування залежностей (в React застосування залежностей відсутнє) і директив (спеціальних атрибутів HTML). Фактично, Angular бажає, щоб розробка інтерфейсної частини програми велась запропонованим чином, на відміну від React [11, 18].

Разом з тим Angular має додаткові проблеми: ми вимушені використовувати TypeScript для забезпечення безпеки типів у додатках Angular, а наявність TypeScript робить фреймворк Angular 2+ не дуже приємним у роботі.

2.1.3 Vue

На перший погляд може здатись, що бібліотека Vue – це суміш Angular та React. Фактично Еван Ю, автор Vue, запозичив концепції з Angular та React. Наприклад, Vue вимагає зберігати логіку компонентів та макети разом із таблицями стилів в одному файлі, так саме, як працює React без таблиць стилів. Щоб дозволити компонентам Vue спілкуватися один з одним, Vue використовує властивості та об’єкти стану.

Vue – це прогресивний фреймворк для розробки інтерфейсів користувача (UI) на основі JavaScript. Він дозволяє розробникам створювати інтерактивні

веб-додатки, використовуючи декларативний підхід до роботи зі змінним станом та компонентною архітектурою. Основні риси та ознаки Vue включають:

– Декларативний синтаксис: Vue використовує шаблони у вигляді розширень HTML для опису відображення компонентів, що робить розробку інтерфейсу користувача більш зрозумілою та простою;

– 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 [16];

– Реактивність: Vue має реактивну систему, яка автоматично оновлює відображення компонентів при зміні їхнього стану, що робить роботу зі змінним станом більш простою та зручною;

– Модульність: Vue дозволяє розробникам використовувати лише ті частини фреймворку, які потрібні в їхньому проекті, що робить його досить легким та масштабованим;

– Розширюваність: Vue має багато доповнень та плагінів, що дозволяють розширити його функціональність та використовувати різні зовнішні бібліотеки.

Одним з недоліків Vue є той факт, що він все ще менш популярний, ніж React чи Angular, в результаті чого у Vue набагато менше готових рішень.

Екосистема Vue включає різноманітні інструменти, бібліотеки та розширення, які розширюють функціональність та можливості Vue.js. Основні компоненти екосистеми Vue включають:

– Vue Router: Бібліотека для реалізації маршрутизації в Vue додатках. Вона дозволяє створювати маршрутизатори, визначати маршрути, навігацію між сторінками, передавати параметри та багато іншого;

– Vuex: Стан-менеджер для Vue додатків, який дозволяє керувати станом додатка, зберігати дані та виконувати асинхронні операції. Він забезпечує централізоване управління станом додатка та спрощує управління даними між компонентами;

– Vue CLI: Командний інтерфейс для створення, налаштування та керування Vue проектами, який надає шаблони проектів, підтримку розширень;

– Vue DevTools: Розширення для браузера, яке надає розробникам зручні інструменти для налагодження, профілювання та аналізу Vue додатків;

– Vue Server-Side Rendering (SSR): Механізм рендерингу Vue додатків на сервері, що дозволяє відправляти вже відрендерену HTML-розмітку клієнту, що покращує продуктивність та SEO-оптимізацію додатка;

– Vue Test Utils: Офіційний набір утиліт для тестування Vue компонентів, які допомагають створювати різні тести та перевіряти роботу компонентів;

– Vue Material, Vuetify, Element UI та інші UI бібліотеки: Багато бібліотек, які надають готові компоненти та стилі для швидкої розробки красивого та функціонального інтерфейсу Vue додатків. Наприклад, Vue Material, Vuetify, Element UI та багато інших відомих бібліотек надають готові рішення для розробки відповідного дизайну та компонентів;

– Vue Plugins: Велика кількість плагінів розроблені співтовариством Vue, які розширюють можливості фреймворку. Ці плагіни включають такі функціональності, як анімації, міжнародизація, валідація форм, кешування, аутентифікація, інтеграція з різними API та інші.

Vue надзвичайно близький з погляду робочого процесу до інших фреймворків. Саме тому дослідження ефективності застосування фреймворків Vue та React для розв’язання однієї й тієї ж самої задачі планування часу дозволить порівняти їх між собою в практичному аспекті.

2.2 Метрики продуктивності

Для порівняння ефективності застосування того чи іншого фреймворку для розв’язання конкретної проблеми необхідно визначитись з метриками, за якими ми будемо оцінювати продуктивність та кількість споживаних ресурсів [14, 18].

До них можна віднести декілька загальних метрик, за якими можна оцінювати продуктивність додатків, розроблених з використанням React та Vue (рис.2.4):

– час завантаження сторінки (Page Load Time): Час, необхідний для повного завантаження веб-сторінки, включаючи всі ресурси, такі як HTML, CSS, JavaScript, зображення та інші;

– час відгуку (Time to First Byte - TTFB): Час, який потрібно серверу для надсилання першого байта відповіді на запит в браузер. Він відображає час, який потрібний серверу для обробки запиту та генерації відповіді;

– час рендерингу (Rendering Time): Це час, який потрібно браузеру на відтворення веб-сторінки після отримання відповіді від сервера. Оптимізація часу рендерингу може поліпшити сприйняття користувачів щодо швидкості додатку.

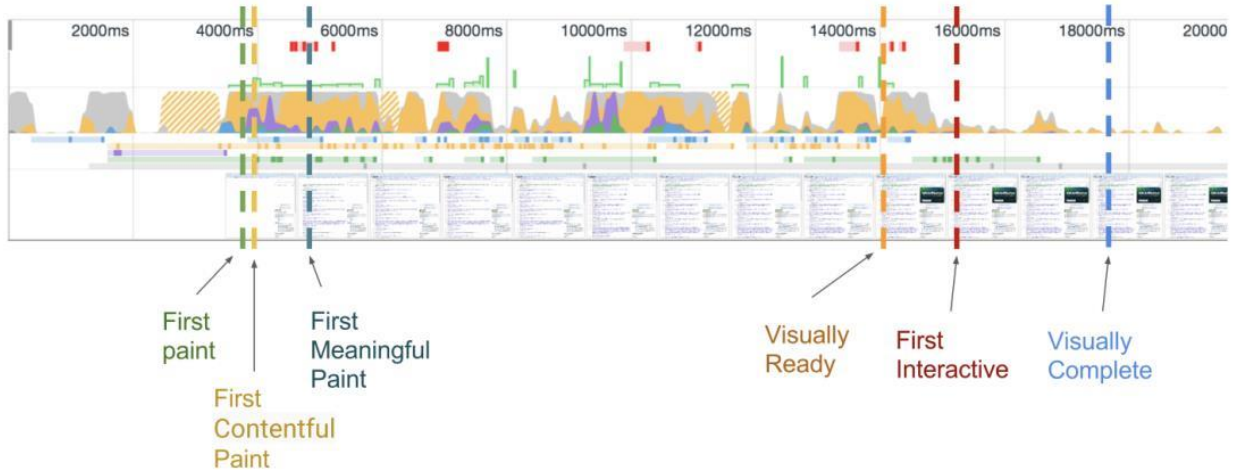


Рисунок 2.4 – Pipeline завантаження веб-сторінок

– кількість запитів до сервера (Number of Server Requests): Це кількість запитів, які відправляються до сервера для отримання різних ресурсів, таких як дані, зображення, стилі, скрипти тощо. Зменшення кількості запитів може знизити час завантаження сторінки та поліпшити продуктивність додатку;

– кількість запитів (Number of Requests): Кількість запитів, які браузер виконує для завантаження всіх ресурсів сторінки, таких як файли CSS, JavaScript,

зображення та інші. Менше кількість запитів може позитивно вплинути на продуктивність додатку;

– розмір бандлу (Bundle Size): Розмір вихідних файлів JavaScript, CSS та інших ресурсів, які передаються на клієнтську сторону для відображення сторінки. Менший розмір бандлу може знизити час завантаження сторінки;

– час відгуку користувача (User-perceived Performance): Час, який відчуває користувач при взаємодії з додатком, такий як час відкриття сторінки, відповідь на взаємодію користувача (наприклад, клік на кнопку) та інші. Швидка відповідь на дії користувача може поліпшити враження від використання додатка;

– частота оновлення інтерфейсу (UI Update Frequency): Це кількість оновлень інтерфейсу в секунду, таких як оновлення даних, анімації тощо. Висока частота оновлення може забезпечити плавну взаємодію користувачів з додатком та відчуття відгуку;

– використання пам’яті (Memory Usage): Це кількість пам’яті, використовуваної додатком, включаючи пам’ять, використовувану браузером, фреймворком та додатковими бібліотеками. Ефективне використання пам’яті може допомогти забезпечити стабільну продуктивність додатку та уникнути перевантаження пам’яті.

Для проведення аналізу за цими параметрами необхідно визначитись з інструментами оцінювання ефективності використання додатків. Існує декілька популярних інструментів, які можна використовувати для порівняння продуктивності додатків, розроблених з використанням React та Vue [14].

2.3 Інструментальні засоби для оцінювання продуктивності

Комбінація різних інструментів може допомогти отримати різнобічну оцінку продуктивності та виявити можливі проблеми, які можуть виникнути в

додатках, розроблених з використанням React та Vue. До таких інструментів можна віднести [14]:

– Lighthouse: Це безкоштовний вбудований інструмент в браузері Google Chrome, який надає деталізовану звітність про різні аспекти продуктивності веб-сторінки, включаючи швидкість завантаження, доступність, використання кешу, виконання JavaScript та інші;

– WebPageTest: Це безкоштовний онлайн-інструмент, який дозволяє вимірювати ряд метрик продуктивності, таких як час завантаження сторінки, швидкість першого контенту, бал від PageSpeed Insights та інші. Він також надає можливість порівнювати результати для різних браузерів та пристроїв;

– DevTools: Це набір інструментів розробника, вбудованих у ряд популярних веб-браузерів, таких як Google Chrome, Mozilla Firefox, Microsoft Edge тощо. Вони надають різні функції для аналізу продуктивності веб-сторінок, такі як профілювання JavaScript, вимірювання часу завантаження, аналіз використання ресурсів та багато іншого;

– Google Analytics: Популярні аналітичні інструменти, які можуть надавати деталізовані дані про продуктивність веб-сторінок, включаючи час завантаження сторінки, кількість переглядів сторінки, відхід від сторінки та інші метрики.

Будемо проводити дослідження продуктивності додатку за допомогою цих інструментальних засобів. Перейдемо до опису процесу розробки додатку для тестування.

3 ПІДГОТОВКА ДО ПРОВЕДЕННЯ ДОСЛІДЖЕНЬ

3.1 Визначення вимог

Обидва односторінкових додатка ToDo будуть побудовані з використанням однакових базових елементів для того, щоб отриманий результат порівняльного аналізу був найбільш точним. Програми будуть використовувати єдину базу даних, що містить дані для обох програм ToDo, які будуть отримані клієнтським додатком. Також для обох додатків будемо використовувати однаковий перелік вимог, оскільки вони повинні відповідати однаковим елементам.

Наведемо вимоги, яким повинен задовольняти додаток ToDo, що буде використаний для проведення порівняння ефективності використання фреймворків React та Vue [19]:

Функціональні вимоги:

– додаток повинен реалізовувати клієнт-серверний підхід до побудови програмної системи;

– додаток повинен забезпечувати можливість створення, відображення, редагування та видалення завдань (todo items), можливість встановлення статусу завдання (наприклад, «виконано» або «не виконано») та здійснення пошуку і фільтрації завдань за різними критеріями;

– додаток повинен бути адаптивним, тобто добре відображатися на різних пристроях, таких як десктопні комп’ютери, планшети та мобільні пристрої, забезпечуючи зручний користувацький досвід на різних пристроях та розмірах екранів;

– додаток повинен працювати ефективно та максимально швидко, з мінімальними затримками та зависаннями, що передбачає оптимізацію роботи з серверними запитами, мінімізацію витрат на рендеринг та оптимізацію роботи з DOM;

– додаток повинен бути здатним до масштабування, забезпечуючи ефективну роботу з великою кількістю завдань та користувачів, можливість додавання нових функцій та розширення функціональності без значного зниження продуктивності.

Під час розробки додатків будемо дотримуватись вимог створення додатку за методологією plank page. Під терміном plank page розуміється проста веб-сторінка, яка містить мінімальну кількість елементів та функцій. Вона часто використовується як проміжний етап під час створення веб-сайту або програми для тестування його базової функціональності та продуктивності. Plank page зазвичай складається з простого дизайну і містить лише основні елементи, такі як заголовок, невелике зображення, текстовий опис та кнопка, яка виконує основну функцію, наприклад, перехід на наступну сторінку або надсилання форми. Ця сторінка не містить складної анімації, великих зображень або інших складних елементів, які можуть сповільнити завантаження або використання ресурсів. Використання методології plank page дозволяє зосередитися на основній функціональності та продуктивності веб-сайту або програми, не відволікаючись на складні елементи та дизайн [8]. Вона може допомогти розробникам швидко визначити, які частини сайту потребують оптимізації і які можуть бути покращені для поліпшення продуктивності та досвіду користувача.

Користувацькі вимоги: додаток повинен мати зрозумілий та зручний користувацький інтерфейс, з можливістю додавання, перегляду, редагування та видалення завдань, відображення статусу завдання, можливістю швидкого відображення нагадувань про терміни виконання завдань, а також можливістю налаштування персональних налаштувань, таких як розмір шрифту, тема тощо.

Це лише загальний опис вимог для додатку ToDo, який ми будемо використовувати для проведення досліджень. Перейдемо до порівняння коду додатків ToDo, програмну реалізацією яких виконано за допомогою фреймворків React та Vue.

3.2 Загальна структура додатку ToDo

Для забезпечення необхідної функціональності додатку визначимось з обов’язковими функціями.

До функціональності списку завдань слід віднести наступні:

– додати нове завдання;

– змінити існуюче завдання;

– видалити існуюче завдання;

– належність завдання до відповідної категорії

– інформація складається із заголовка, тексту завдання та дати, що стосується завдання.

Список категорій повинен забезпечувати

– додавання нової категорії;

– змінення існуючої категорії;

– видалення існуючої категорії;

– перегляд всіх категорій.

Зв’язок між елементами додатку ToDo наведено на рис.3.1.

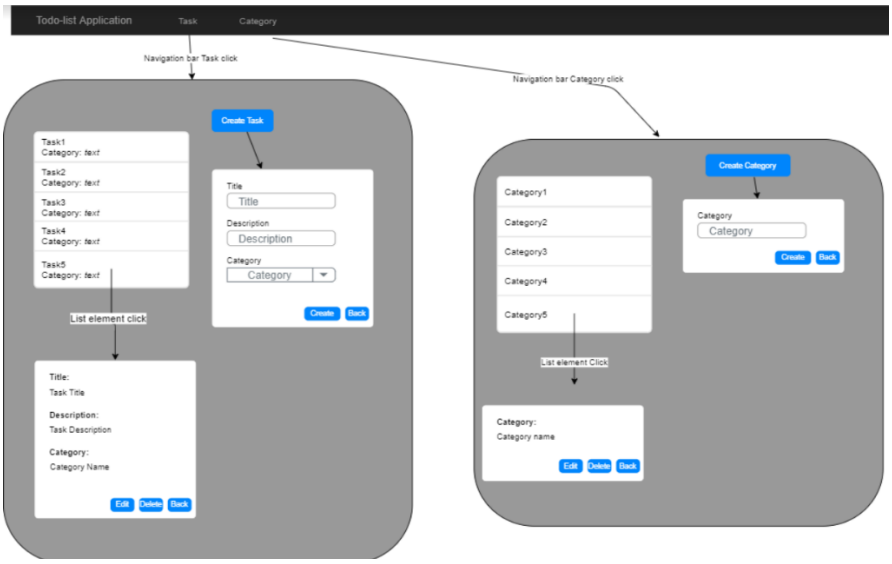


Рисунок 3.1 – Макет додатку ToDo

Реляційна модель сутностей – це визначення відношень між елементами бази даних, в якій розміщені її сутності, та те, як вони пов’язані один з одним [12]. Для програми ToDo розроблена та створена база даних із двома сутностями –завдання та категорії, співвідношення між якими визначаються як «багато до одного» (рис.3.2).

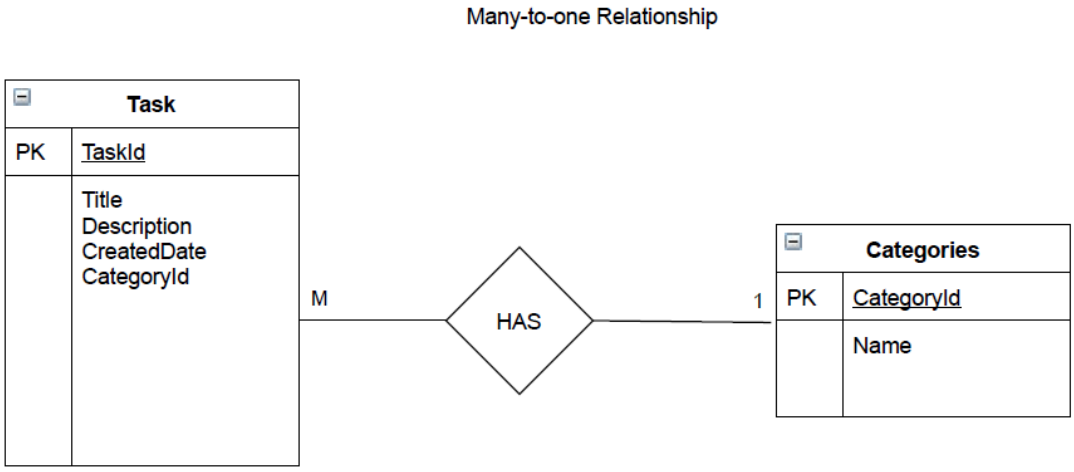


Рисунок 3.2 – ER-діаграма

Сутність завдання пов’язана з однією сутністю у категоріях. Сутність категорії може бути пов’язана з будь-якою кількістю сутностей у завданнях, де у завдання може бути лише одна категорія чи нуль.

3.3 Побудова додатку з використанням React

Розглянемо особливості побудові додатку з використанням бібліотеки React на прикладі наведеного нижче лістингу додатку ToDo.

Додаток ToDo використовує функціональні компоненти та хуки (useState) з бібліотеки React [13]. Наведений нижче код реалізує функціонал, який дозволяє користувачу додавати нові завдання, видаляти їх та відображати список завдань:

```
// App.js
import React, { useState } from 'react';

const App = () => {
 const [todos, setTodos] = useState([]);
 const [newTodo, setNewTodo] = useState('');

 const handleInputChange = (event) => {
 setNewTodo(event.target.value);
 };

 const handleFormSubmit = (event) => {
 event.preventDefault();
 if (newTodo.trim() === '') return;
 setTodos([...todos, newTodo]);
 setNewTodo('');
 };

 const handleTodoDelete = (todoIndex) => {
 setTodos(todos.filter((todo, index) => index !== todoIndex));
 };

 return (
 <div>
 <h1>ToDo List</h1>
 <form onSubmit={handleFormSubmit}>
 <input type="text" value={newTodo} onChange={handleInputChange} />
 <button type="submit">Add Todo</button>
 </form>
 <ul>
 {todos.map((todo, index) => (
 <li key={index}>
 {todo}
 <button onClick={() => handleTodoDelete(index)}>Delete</button>
 </li>
 ))}
 </ul>
 </div>
 );
};

export default App;
```

За допомогою компоненту App.js ми передаємо список задач todos та функцію deleteTodo в компонент TodoList. Компонент TodoList відображає список задач за допомогою методу map, який створює компоненти TodoItem для кожної задачі в масиві todos.

В компоненті TodoItem для видалення використовується функція deleteTodo, що викликається при натисканні на кнопку "Видалити", передаючи todo.id як аргумент для визначення, яку саме задачу необхідно видалити.

Форма для додавання нових задач використовує хук useState, через який зберігаємо значення поля вводу newTodo та функцію setNewTodo для його зміни. Обробник події handleInputChange викликається при зміні значення поля вводу та оновлює стан newTodo з введеним значенням.

Обробник події handleFormSubmit викликається при відправці форми. В цьому обробнику здійснюється виклик функції addTodo, передаючи в неї нову задачу з унікальним ID, що генерується за допомогою Date.now(), та значенням поля вводу newTodo. Після цього виконується очищення значення поля вводу newTodo за допомогою setNewTodo(‘‘).

Компонента TodoList також здійснює оновлення відображення задач шляхом додавання ID до кожної задачі для її подальшої ідентифікації.

3.4 Побудова додатку з використанням Vue

Основним файлом, з якого розпочинається розробка додатку, є App.vue. В файлі App.vue ми використовуємо компоненти Vue для створення інтерфейсу користувача: компонент v-list з пакету Vuetify для створення списку завдань. Наведений нижче код буде доданий у розділ <template>:

```
<template>
 <div>
 <v-app>
 <v-content>
 <v-container>
 <v-list>
 <v-list-item-group>
 <v-list-item
 v-for="item in items"
 :key="item.id"
 :value="item"
 >
 <v-list-item-action>
 <v-checkbox
 v-model="item.done"
 color="primary"
 hide-details
 ></v-checkbox>
 </v-list-item-action>
 <v-list-item-content>
 {{ item.text }}
```

Структура додатку складається з наступних компонентів:

– TodoApp: головний компонент, який містить список задач та форму для їх додавання;

– TodoList: компонент, який містить список задач та відображає їх на екрані;

– TodoForm: компонент, який містить форму для додавання нової задачі.

Для подальшої реалізації додатку необхідно визначити стейт додатку та відповідні мутації для його зміни. Стейт додатку повинен містити наступні поля:

– todos: список задач;

– nextId: ID наступної задачі.

Визначемо наступні мутації:

– ADD_TODO: додавання нової задачі до списку;

– REMOVE_TODO: видалення задачі зі списку;

– TOGGLE_TODO: зміна статусу задачі.

Для всіх компонентів та організації їх взаємодії зі станом додатку визначимо наступні методи:

– addTodo: додавання нової задачі до списку;

– removeTodo: видалення задачі зі списку;

– toggleTodo: зміна статусу задачі.

Розглянемо компонент Task.vue, який буде відповідати за відображення одного елементу списку:

```
<template>
 <div>
 <input type="checkbox" v-model="task.done" />
 <span :class="{ done: task.done }">{{ task.text }}</span>
 <button @click="$emit('delete', task)">Delete</button>
 </div>
</template>

<script>
export default {
 props: {
 task: {
 type: Object,
 required: true,
 },
 },
};
</script>

<style>
.done {
 text-decoration: line-through;
}
</style>
```

В цьому компоненті використовуються наступні директиви Vue:

– v-model – для зв’язування змінної done з чекбоксом, щоб вона автоматично оновлювалася при зміні стану чекбокса;

– :class – для динамічного встановлення класу done, якщо завдання виконано;

– @click – для виклику методу deleteTask, який відповідає за видалення завдання зі списку.

Файл App.vue використовується для відображення списку завдань.

Директива v-for використовується для організації перебору усіх завдань зі списку tasks та відображення їх за допомогою компоненти Task.vue. Директива @delete використовується щоб зв’язати подію видалення завдання з методом deleteTask.

Розглянемо компонент TaskForm, який будемо використовувати для додавання нових завдань. Він буде мати наступну структуру:

```
<template>
 <div>
 <h2>Add Task</h2>
 <form @submit.prevent="addTask">
 <label for="new-task">New Task:</label>
 <input id="new-task" v-model="newTask" type="text" required>
 <button type="submit">Add Task</button>
 </form>
 </div>
</template>
```

У цьому коді використовується директива v-model, яка дозволяє зв’язувати значення введеного тексту з властивістю newTask. Також ще одна директива @submit.prevent служить для перехоплення події подачі форми та виклику методу addTask, який додає нове завдання до списку. Також до цього ж компоненту додаємо метод addTask:

```
methods: {
 addTask() {
 if (this.newTask) {
 this.tasks.push(this.newTask);
 this.newTask = "";
```

```
  }
  },
}
```

Цей код використовується для перевірки, чи існує значення властивості newTask, та додає його до масиву tasks, якщо воно існує.

Підводячи підсумки щодо підготовки додатків для проведення подальших експериментів необхідно зазначити, що програмний код додатків, який наведено у додатку 2, не буде досліджуватись щодо оптимізації структури додатку. В наступному розділі буде розглянуто результати проведених експериментів.

4 АНАЛІЗ РЕЗУЛЬТАТІВ ДОСЛІДЖЕНЬ

У розділі 2 було визначено, що дослідження будуть проводитись за допомогою безкоштовних інструментальних засобів Lighthouse, DevTools та Google Chrome Developer Tools у середовищі браузера Google Chrome 112.0.5615.138 (64-розрядна версія). Під час проведення досліджень всі дії будуть виконуватись в режимі інкогніто з використанням лише однієї активної вкладки. Опишемо проведені дослідження та проведемо аналіз отриманих результатів.

4.1 Дослідження параметрів за допомогою інструментальних засобів

Для порівняння ефективності застосування фреймворків React та Vue будемо використовувати наступні найбільш загальні метрики [18]:

– First Contentful Paint (FCP): це метрика продуктивності веб-сторінки, яка вимірює час, що минув з початку завантаження сторінки до моменту, коли браузер вперше відобразив на екрані контент для користувача, наприклад текст, зображення або фонове зображення;

– Largest Contentful Paint (LCP): це метрика продуктивності веб-сторінки, яка вимірює час, за який браузер відобразить на екрані найбільший контентний елемент на сторінці, наприклад, велике зображення, відео або блок тексту;

– Total Blocking Time (TBT): це метрика продуктивності веб-сторінки, яка вимірює час блокування основного потоку браузера під час завантаження сторінки. Вона вимірює час, який потрібний браузеру для обробки та виконання завдань JavaScript, перш ніж сторінка стане повністю інтерактивною.

– Cumulative Layout Shift (CLS): це метрика продуктивності веб-сторінки, яка вимірює рівень нестабільності макета на сторінці. Вона оцінює, наскільки

багато елементів на сторінці змінюють свою позицію під час завантаження, що може призвести до незручностей для користувачів, наприклад, коли вони натискають на неправильну кнопку через усунення контенту на сторінці;

– Speed Index (SI): це показник швидкості завантаження веб-сторінки, який показує, як швидко контент на сторінці починає відображатися для користувача. Speed Index оцінює, наскільки швидко користувач отримує уявлення про те, що відбувається на сторінці, та наскільки плавно відбувається завантаження контенту.

Для дослідження продуктивності додатків ToDo, що були розроблені з використанням фреймворків React та Vue, проекти було розміщено на одному хостінгу для створення однакових умов. При дослідженні використовувався браузер Google Chrome версії 113.0.5672.64 (64-розрядна версія) зі встановленим інструментальним забезпеченням. Завантаження додатків відбувалось з єдиної платформи доступу (рис.4.1) з відкриттям вікон в режимі інкогніто за посиланням: https://ovsiannykov.github.io/masters-diploma-public.github.io/.

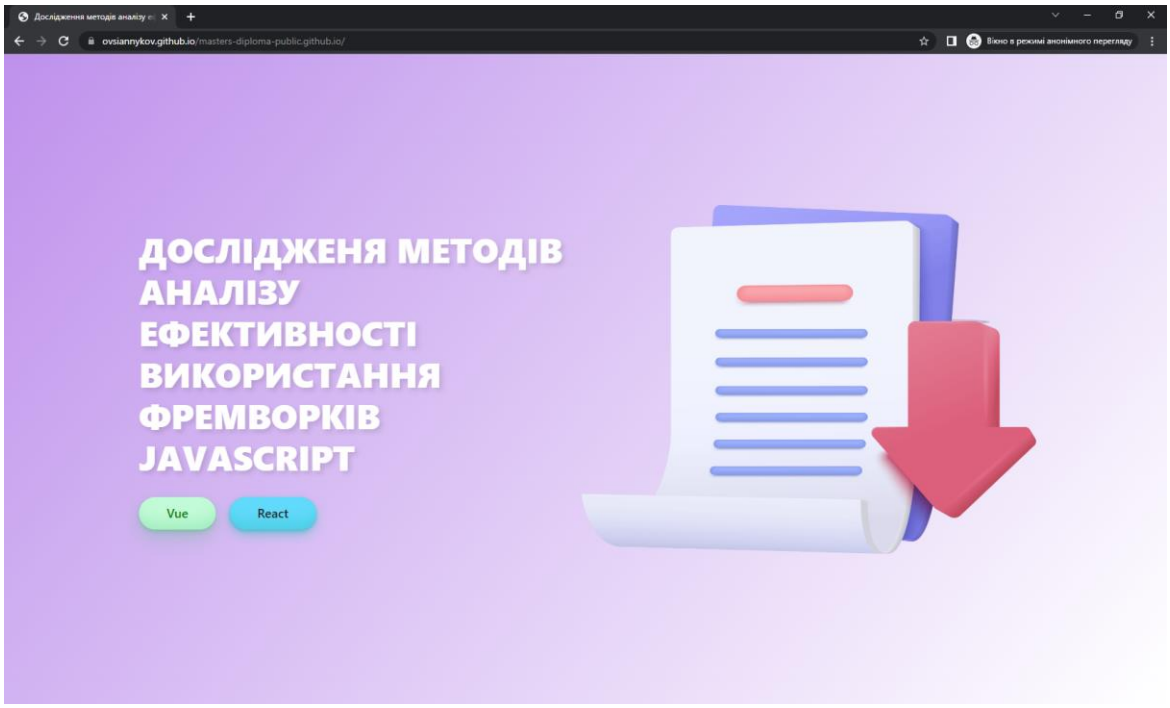


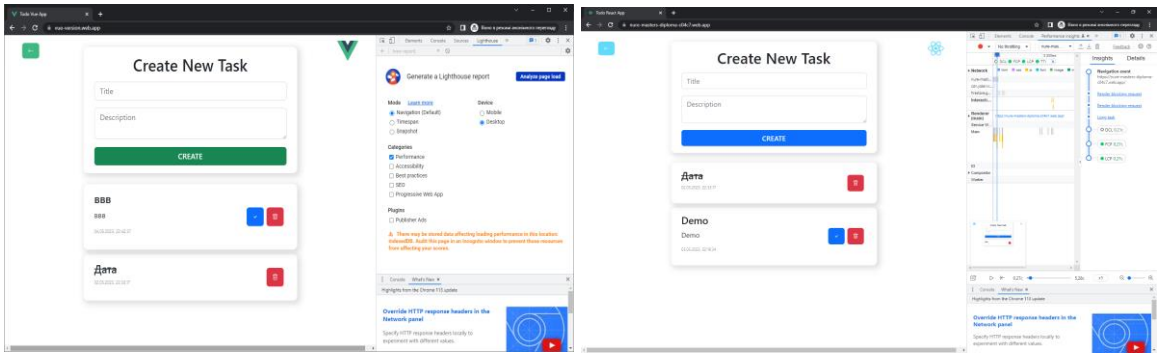
Рисунок 4.1 – Портал проекту

Під час порівняння проектів будемо дотримуватись наступних загальних вимог [19, 20]:

– всі дії в додатку виконуються в одному вікні браузера;

– для перевірки фунціоналу після відкриття додатку буде виконано додавання задачі, її перегляд, переведення у статус «виконано» та видалення виконаної задачі.

Загальний вигляд додатків на початку дослідження наведено на рис.4.2.



а) б)

Рисунок 4.2 –Додатки ToDo під час дослідження:

а) фреймворк Vue; б) фреймворк React

4.1.1 Дослідження за допомогою Lighthouse

Під час дослідження веб-сторінок за допомогою Lighthouse [21] у браузері Google Chrome. Конфігурацію наведено на рисунку (див.рис. 4.3).

По завершенню дослідження звіт Lighthouse буде містити показники продуктивності сторінки, включаючи обрані нами метрики, які узагальнені у вигляді таблиці 4.1.

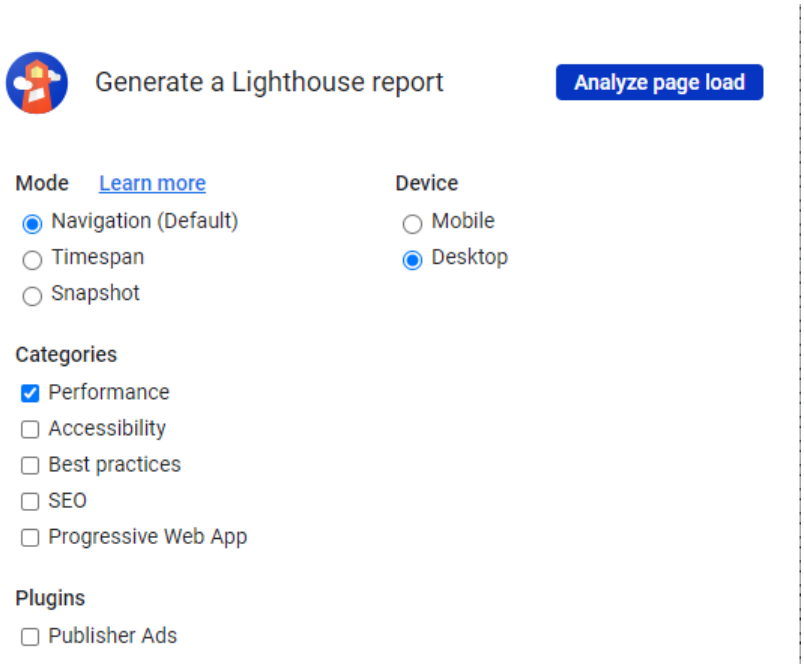


Рисунок 4.3 – Конфігурація Lighthouse

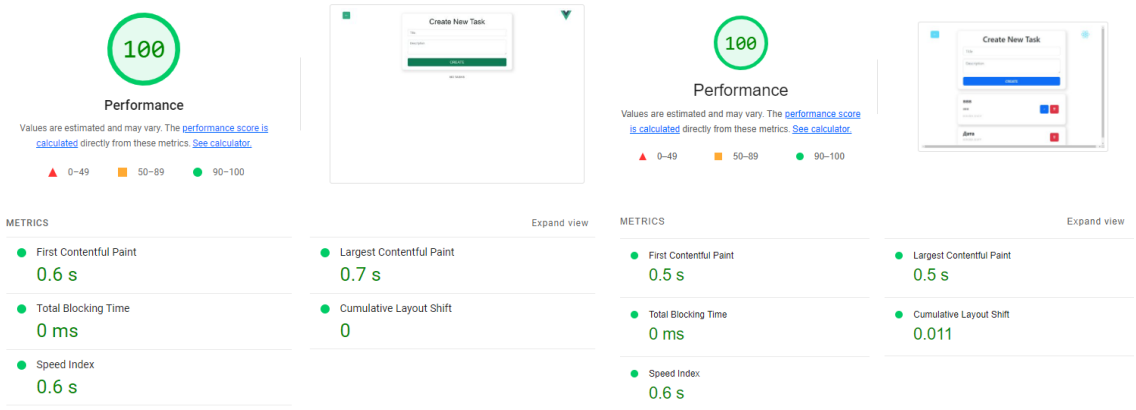
Після того, як Lighthouse завершив збір показників ефективності, він перетворює кожне необроблене значення показника в оцінку від 0 до 100, дивлячись на позицію значення показника в розподілі оцінок Lighthouse. Розподіл балів – це логнормальний розподіл, отриманий на основі метрик продуктивності фактичних даних про продуктивність веб-сайту. Модель кривої оцінки Lighthouse використовує дані HTTP-архіву для визначення двох контрольних точок, які потім задають форму логнормальної кривої. Оцінки мають кольорове кодування (див.рис.4.3), причому оцінки індикаторів та оцінки продуктивності забарвлюються відповідно до наступних діапазонів:

– від 0 до 49 (червоний): недостатній;

– від 50 до 89 (помаранчевий): потребує покращення;

– від 90 до 100 (зелений): добре.

При вимірюванні ефективності кожної структури в якості остаточної оцінки для аналізу в цій роботі було використано середнє значення з 10 вимірювань щоб запобігти похибки вимірювання, результати наведено на рисунку нижче (див.рис. 4.4).



а) б)

Рисунок 4.4 – Звіт Lighthouse: а) Vue; б) React

Відповідно до методики вимірювання, яку застосовується в Lighthouse, було визначено вагові коефіцієнти, що має кожна з метрик (рис.4.5).

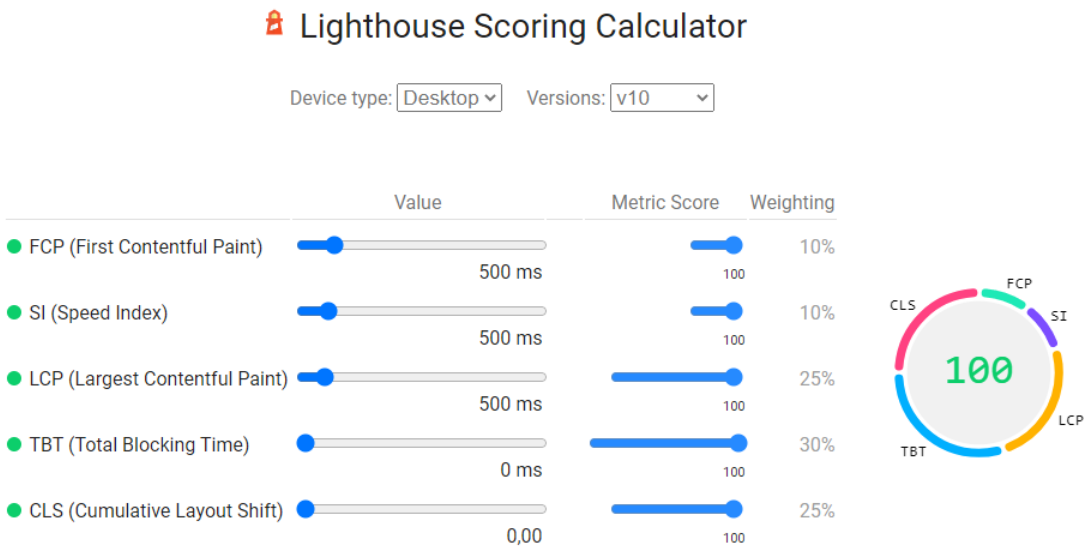


Рисунок 4.5 – Вагові коефіцієнти метрик

Також було розраховано середнє значення за кожним з параметрів за результатами 10 вимірів для кожної метрики, результати впорядковані у табл.4.1.

Причина відмінності значення TBT від інших метрик полягає у порядку його вимірювання. Всі інші метрики вимірюються від початку запиту сторінки, а вимірювання TBT починається з LCP і триває до кінця останнього найдовшого завдання, яке існує не завжди.

Таблиця 4.1 – Метрики продуктивності

Метрики / Фреймворки	FCP, сек	LCP, сек	TBT, сек	CLS, сек	SI, сек
React	0,76	1,04	0,007	0,0055	0,76
Vue	0,59	0,74	0,0	0,7034	0,59

За результатами вимірювання метрик Lighthouse можна припустити, що додаток на Vue має кращі показники продуктивності порівняно з додатком на React.

4.1.2 Дослідження за допомогою WebPageTest

Для дослідження параметрів додатків за допомогою WebPageTest необхідно виконати наступні кроки:

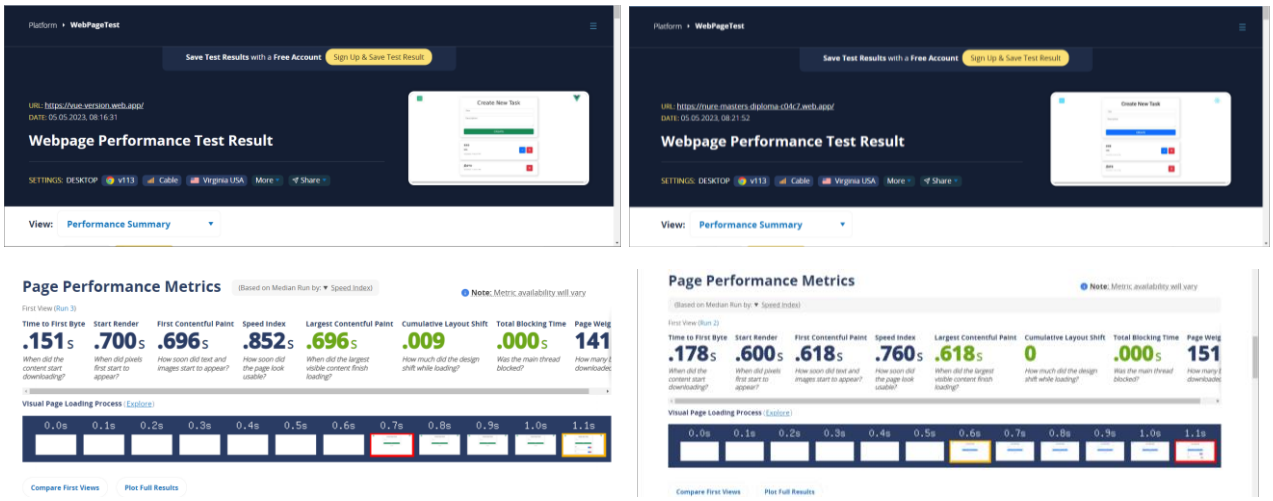
– зайти на сайт https://www.webpagetest.org/;

– обрати місцезнаходження тестового сервера зі списку «Test Location»;

– у полі URL ввести адресу сайту, який потрібно протестувати;

– натиснути кнопку «Start Test».

Результати тесту містять різні параметри , що характеризують додаток: час завантаження сторінки та інші показники продуктивності, такі як Time to First Byte, час рендерингу та інші (рис.4.6). Серед цих метрик головну увагу звернемо на ті метрики, які вже були визначені за допомогою інструменту Lighthouse для подальшого порівняння отриманих результатів: Largest Contentful Paint, Total Blocking Time, Cumulative Layout Shift, Speed Index. Слід зазначити, що досить цікавим показником є Time to First Byte, за яким фреймворк Vue має суттєву перевагу над React.



а) б)

Рисунок 4.6 – Результати дослідження з використанням WebPageTest: а) Vue; б) React

Середні значення за кожним з параметрів за результатами 10 вимірювань для кожної метрики впорядковані у табл.4.2.

Таблиця 4.2 – Метрики продуктивності

Метрики / Фреймворки	FCP, сек	LCP, сек	TBT, мсек	CLS, сек	SI, сек
React	0,618	0,618	0	0	0,760
Vue	0,696	0, 696	0	0,009	0, 852

Відмінність між результатами вимірювання CLS стабільності візуального вмісту під час завантаження є незначною, тому можна сказати, що обидва додатки на React та Vue показують дуже добрі результати щодо стабільності вмісту. Ця відмінність вказує на незначний рух елементів на сторінці під час завантаження: чим менше значення CLS, тим краще, оскільки це означає, що користувачі не будуть перешкоджати рухом елементів на сторінці під час першого завантаження [20]. Інші результати вимірювання метрик з використанням WebPageTest показують, що додаток на Vue має трохи гірші

показники продуктивності порівняно з додатком на React, однак відмінності незначні.

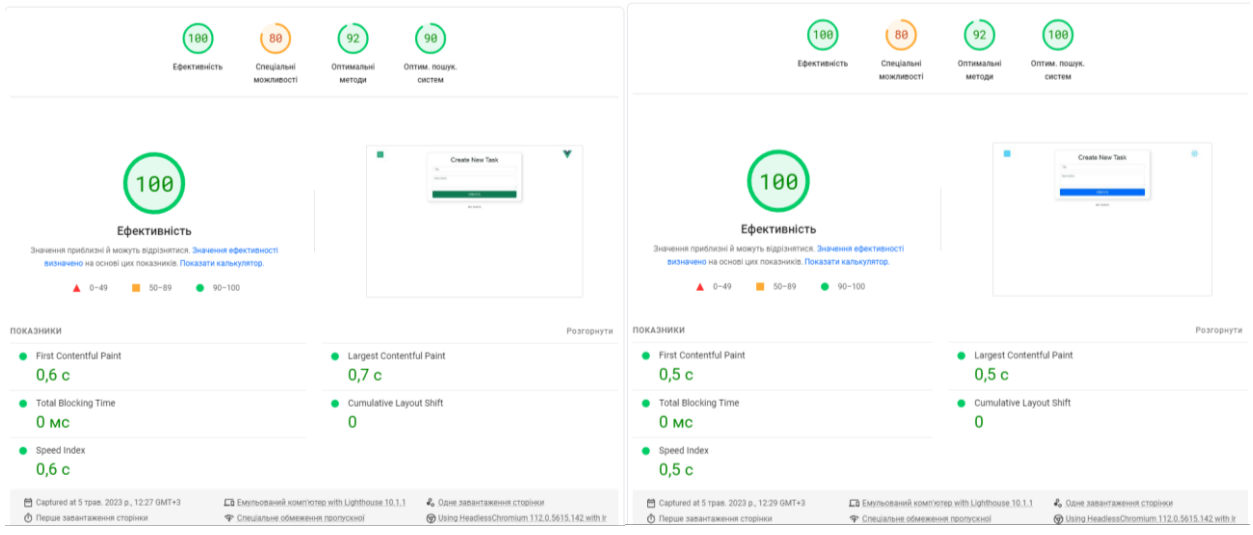
4.1.3 Дослідження з використанням Web Vitals Extension

Для дослідження параметрів завантаження сторінки за допомогою Web Vitals Extension необхідно виконати наступні дії:

– обрати значок Web Vitals Extension на панелі інструментів браузера;

– у вікні з результатами метрик обрати вкладку «Overview», щоб переглянути загальний огляд метрик для сторінки.

Перемикаючись між вкладками «Metrics», «Opportunities», «Diagnostics» та «Passed Audits» можна побачити докладні результати метрик та рекомендації щодо покращення продуктивності додатків. Слід зазначили, що Web Vitals Extension не забезпечує можливість неодноразового обчислення метрик та розрахований на отримання результатів у визначений проміжок часу. Результати проведення вимірювань представлено на рис. 4.7.



а) б)

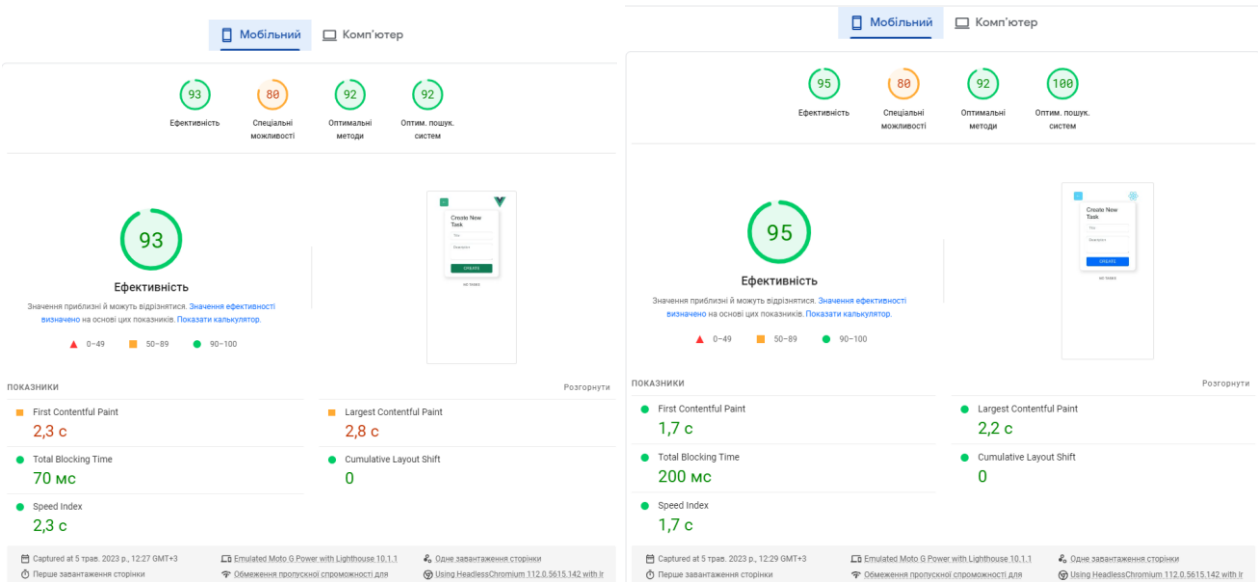
Рисунок 4.7 – Дослідження за допомогою Web Vitals Extension: а) Vue; б) React

Узагальнені результати вимірювань за кожним з параметрів за результатами 10 вимірювань для кожної метрики впорядковані у табл.4.3.

Таблиця 4.3 – Метрики продуктивності

Метрики / Фреймворки	FCP, сек	LCP, сек	TBT, мсек	CLS, сек	SI, сек
React	0,5	0,5	0	0	0,5
Vue	0,6	0,7	0	0,009	0,6

Web Vitals Extension надає можливість імітувати роботи мобільного додатку на обраній платформі, результати такого дослідження продуктивності наведено на рис.4.7, а результати вимірювань узагальнені у таблиці 4.4, однак під час підведення підсумків щодо продуктивності фреймворків ці результати враховуватись не будуть.



а) б)

Рисунок 4.7 – Дослідження за допомогою Web Vitals Extension метрик мобільного використання: а) Vue; б) React

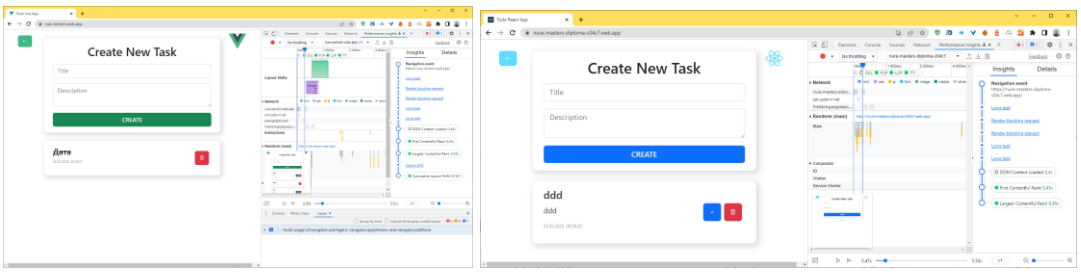
Таблиця 4.5 – Метрики продуктивності

Метрики / Фреймворки	FCP, сек	LCP, сек	TBT, мсек	CLS, сек	SI, сек
React	1,7	2,2	200	0	1,7
Vue	2,3	2,8	70	0	2,3

За результатами вимірювань можна зробити висновки, що додаток на Vue починає працювати раніш у порівнянні з додатком на React та є більш стабільним, однак відмінності незначні.

4.1.4 Дослідження за допомогою Perfomance Inside

Для проведення досліджень після завантаження сторінки у браузер необхідно відкрити інструмент Performance Inside у браузері та почати запис профілю продуктивності. В додатку необхідно виконати дії, що відповідають використанню додатку для планування часу користувачем, після чого припинити запис та виконати аналіз отриманих результатів (рис.4.8).



а) б)

Рисунок 4.8 – Результати дослідження з використанням Perfomance Inside: а) Vue; б) React

Інструментальний засіб розробника Google Chrome Perfomance Inside має спрямування на розробників додатків та орієнтований на надання інформації щодо поточної продуктивності додатку, тому для оцінювання продуктивності додатку для порівняння з результатами інших інструментальних засобів він підходить досить погано, що бачимо з результатів метрик продуктивності, наведених в таблиці 4.6.

Таблиця 4.6 – Метрики продуктивності

Метрики / Фреймворки	FCP, сек	LCP, сек	TBT, мсек	CLS, сек	SI, сек
React	0,45	0,45	0	0	0,4
Vue	0,45	0,45	0	0,0167	0,44

На підсумку аналізу результатів, отриманих за допомогою вбудованого інструменту Google Chrome Perfomance Inside можна зробити висновок щодо їх низької інформативності, оскільки додатки мають однакову продуктивність.

4.2 Порівняльний аналіз отриманих результатів

Базуючись на результатах проведеного аналізу метрик, отриманих за допомогою різних інструментальних засобів можна зробити узагальнені висновки щодо отримання відповіді на питання: який фреймворк JavaScript найкраще підходить з точки зору продуктивності, модульності та зручності використання для розробки односторінкового додатку. В своєму дослідження ми звернули увагу саме на головний аспект – продуктивність, хоча також бажано враховувати такі аспекти, як модульність та юзабіліті, однак з цієї точки зору фреймворки React та Vue порівнювати не бажано, оскільки протягом своєї

еволюції React отримав набагато більшу підтримку з боку розробників і має набагато довшу історію, ніж Vue. Тому повернемось саме до аналізу показників продуктивності. З цією метою проведемо визначення важливості критеріїв. Він включає в себе процес порівнянь за всіма критеріями, а також математичну процедуру розрахунку та перевірки узгодженості такого порівняння.

В таблиці 4.7 узагальнені результати попередніх вимірювань та визначено середнє значення, медіану, стандартне відхилення та середнє абсолютне відхилення під час аудиту веб-сторінки проекту ToDo. За винятком загального часу блокування, всі значення для перевірок становлять 624,29 мілісекунд для середнього значення, 623,65 мілісекунд для медіани, 1,16 мілісекунд для стандартного відхилення і 0,21 для медіанного абсолютного відхилення. Відносно середнього значення стандартне відхилення становить лише ~0,185%. Те, що всі значення для порівняння однакові, є очікуваним, оскільки браузеру майже нічого обробляти на веб-сторінці. Більша частина, якщо не вся затримка у вимірах, пов’язана із запитом сторінки з сервера. Тому це значення можна вважати базовим для інших тестів, оскільки воно здебільшого стосується саме фреймворків, які забезпечують зв’язок між браузером і сервером, а не те, що обробляється у браузері.

Таблиця 4.7 – Середнє значення, медіана, стандартне відхилення та медіанне абсолютне відхилення для початкової сторінки проекту

Метрика	Середнє значення, мс	Медіана, мс	Стандартне відхилення, мс	Медіана абсолютного відхилення, мс
FCP	500,00	498,84	1,16	0,21
LCP	500,00	498,84	1,16	0,21
SI	500,00	498,84	1,16	0,21
CLS	500,00	498,84	1,16	0,21
TBT	0,00	0,00	0,00	0,00

У Таблиці 4.8 наведено середнє, медіану, стандартне відхилення та середнє абсолютне відхилення для фреймворку React. Значення, що були отримані з усіх вимірювань, згруповані разом з усіх тестових запусків. FCP, Speed Index та TBT мають майже однакові значення в кожній виборці.

Таблиця 4.8 – Середнє значення, медіана, стандартне відхилення та медіанне абсолютне відхилення для фреймворку React

Метрика	Середнє значення, мс	Медіана, мс	Стандартне відхилення, мс	Медіана абсолютного відхилення, мс
FCP	561,6	530,8	15,40	7,80
LCP	536,2	518,1	9,05	4,63
SI	605,0	605,0	0,00	0,1
CLS	1,101	0,00	0,55	0,38
TBT	1,75	0,00	0,875	0,54

У таблиці 3 ми бачимо середнє, медіану, стандартне відхилення та середнє абсолютне відхилення для версії Vue. Значення розраховуються так само, як і у версії React, всі значення вимірювань з усіх тестових запусків. Всі статистичні дані Vue є досить однорідними.

Таблиця 2 – Середнє значення, медіана, стандартне відхилення та медіанне абсолютне відхилення для фреймворку Vue

Метрика	Середнє значення, мс	Медіана, мс	Стандартне відхилення, мс	Медіана абсолютного відхилення, мс
FCP	590,6	590,3	0,15	0,18
LCP	611,0	655,5	22,25	11,23
SI	615,75	600,0	7,875	4,0425
CLS	5,434	6,467	0,5165	0,363
TBT	0,00	0,00	0,0	0,105

На підсумку порівняння розглянемо графічне подання результатів, що представлено на рис.4.7. З рисунку наявно видно, що React та Vue працюють по-різному. Ці відмінності стосуються швидкості, дисперсії швидкості та однорідності між метриками.

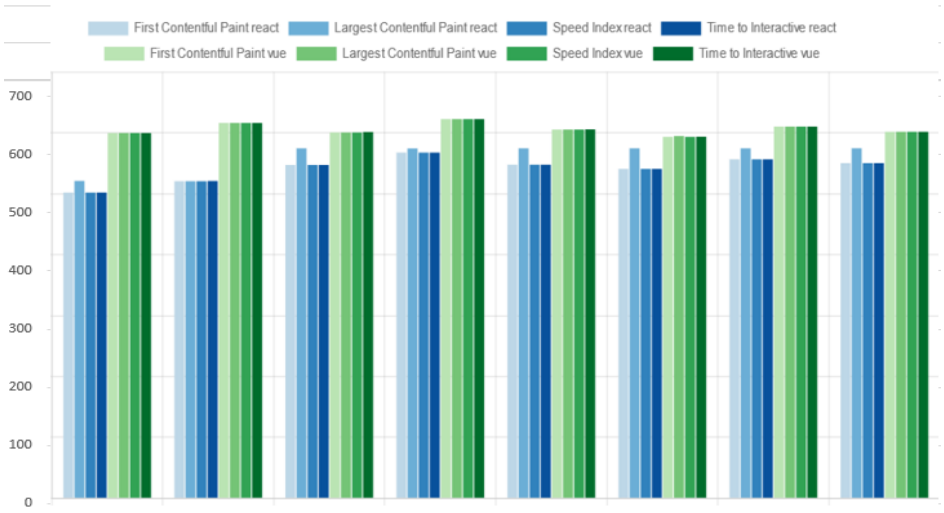


Рисунок 4.7 – Медіанні значення React та Vue

React мав в середньому майже 2 мс затримки в метриці Total Blocking Time, тоді як Vue не мав жодної затримки. Цю різницю можна пояснити різницею в FCP, оскільки TBT вимірюється від початку FCP і далі до кінця останнього завдання [15]. Отже, якби React був трохи повільнішим за метрикою FCP (як це не дивно звучить), він також міг би не мати жодного TBT. В середньому React працює трохи швидше за Vue: індекс швидкості FCP для React приблизно на 30 мс кращий за показники Vue, а за показником LCP React на 75 мс швидкіший за Vue.

Разом з тим слід зазначити, що хоча React працює швидше, але він має більший розкид результатів, ніж Vue: стандартне відхилення React лежить в діапазоні від 7,8 до 55,40 мс, тоді як у Vue – від 15 до 18 мс.

При звичайному використанні цих додатків не було помічено жодної різниці у продуктивності, якщо розглядати лише взаємодію користувача з додатками. Також слід зазначити, що значення LCP вважається хорошим, якщо

воно менше за 2,5 с, а додатки, за допомогою яких проводились дослідження, не мали значень LCP гірших за 2,8 с для мобільних додатків, а для десктопних додатків – не виходили за межі 1,2 с.

Додатки, що розробляються з використанням JavaScript, залежать від використання SSR (Server-Side Rendering) або CSR (Client-Side Rendering). React за замовчуванням використовує CSR, що є потенційною причиною високого рівня TBT, оскільки сервер повинен спочатку надіслати весь JavaScript-код для виконання на клієнтському браузері, перш ніж можна буде отримати вміст сторінки. Використання SSR в Vue може допомогти знизити TBT, оскільки він дозволяє серверу відправляти вже підготовлену сторінку разом з вмістом у клієнтський браузер, що може пришвидшити відображення контенту. Тому React краще підходить для розробки сервер-орієнтованих додатків, а Vue більш ефективно працює на стороні клієнта.

Реальна продуктивність додатків залежати від багатьох факторів, включаючи розмір проекту, оптимізацію коду, використання кешування та інші шляхи оптимізації. Тому під час вибору фреймворку бажано проводити додаткове тестування та вимірювання продуктивності, орієнтуючись на фактори, які можуть впливати на продуктивність додатків, серед яких головне місце займає оптимізація коду додатку.

ВИСНОВКИ

В результаті написання кваліфікаційної роботи магістра було виконано дослідження методів розробки веб-орієнтованих додатків із застосуванням фреймворків JavaScript, визначено переваги та недоліки застосування більшості JavaScript-фреймворків, проаналізовано наукові дослідження, що проводились вітчизняними та закордонними дослідниками та практичними фахівцями стосовно шляхів підвищення ефективності розробки веб-орієнтованих додатків.

Під час написання кваліфікаційної роботи було виконано дослідження науково-технічної літератури в предметній галузі дослідження, описано постановку задачі дослідження, визначено перелік локальних задач, розв’язання яких безпосередньо пов’язані з метою дослідження. На підставі проведеного аналізу першоджерел було визначено метрики ефективності, які були покладені в основу порівняння ефективності використання веб-орієнтованих додатків планування часу, визначено перелік бібліотек та визначено перелік досліджень, які були проведені над програмною системою планування часу.

В результаті проведених досліджень було зроблено наступні висновки:

– фреймворки React та Vue мають свої переваги і недоліки, тому вибір кращого залежить від конкретної ситуації і вимог проекту;

– React відомий своєю гнучкістю та широкою підтримкою від спільноти, має велику кількість різноманітних компонентів та інструментів, що дозволяють швидко створювати додатки та має високу продуктивність;

– Vue має простий та легкий синтаксис, що дозволяє швидко вивчати та розробляти додатки, має дуже високу та стабільну швидкодію та добру оптимізацію, що дозволяє досягти високої продуктивності додатків.

– React краще підходить для розробки сервер-орієнтованих додатків, а Vue більш ефективно працює на стороні клієнта.

Отже, якщо ви шукаєте максимальну гнучкість та широку підтримку від спільноти, React може бути кращим варіантом. Якщо ж ви шукаєте простоту та

швидкість розробки, то Vue може бути кращим варіантом. Проте, щоб визначити кращий фреймворк, вам потрібно врахувати конкретні потреби та вимоги вашого проекту. Не можна сказати, що одна бібліотека є більш продуктивною, ніж інша, лише на основі проведеного дослідження – необхідно провести більше досліджень для отримання більш вичерпних результатів, з більш різноманітними суб'єктами тестування. Однак слід зазначити, що разі висування високих вимог щодо стабільної продуктивності додатку не рекомендується обирати React для реалізації односторінкових веб-додатків.

Результати, що знайшли своє відображення у кваліфікаційній роботі магістра, обговорювались на 27 Міжнародному молодіжному форумі «Радіоелектроніка і молодь у XXI столітті» [21].