Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук (або центр післядипломної освіти, або навчально-науковий центр заочної форми навчання)
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження ефективності розробки веб-додатків з використанням фронтенд фреймворків
(тема)

Виконав:
студент (ка) 2 курсу, групи ІПЗм-22-3

Волохань В.В.
(прізвище, ініціали)

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

Керівник проф. Галуза О. А.
(посада, прізвище, ініціали)

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

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук (або центр післядипломної освіти, або навчально-науковий центр заочної форми навчання)
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 Волоханю Віктору Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Дослідження ефективності розробки веб-додатків з використанням фронтенд фреймворків»
Затверджена наказом по університету від 29.03. 2024р. № 250 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 25.06.2024
3. Вихідні дані до роботи інформація про веб-застосунки, веб-розробку, фреймворки
4. Перелік питань, що потрібно опрацювати в роботі
аналіз предметної галузі і постановка задачі, огляд веб-додатків, огляд та аналіз літературних джерел з дослідження, аналіз та порівняння популярних фронтенд фреймворків

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі та постановка задачі	29.03 – 05.04.24	*виконано*
2	Постановка задачі	05.04 – 15.04.24	*виконано*
3	Аналіз предметної області	15.04 – 25.04.24	*виконано*
4	Планування проведення порівняння фронтенд фреймворків	25.04 – 01.05.24	*виконано*
5	Проведення дослідження методів, що впливають на ефективність розробки	01.05 – 15.05.24	*виконано*
6	Написання та оформлення статті та тез доповіді	15.04 – 18.05.24	*виконано*
7	Підготовка пояснювальної записки	18.05 – 30.05.24	*виконано*
8	Підготовка презентації та доповіді	30.05 – 05.06.24	*виконано*
9	Нормоконтроль	05.06 – 15.06.24	*виконано*
10	Рецензування	15.06 – 20.06.24	*виконано*
11	Занесення диплома в електронний архів	20.06.2024	*виконано*
12	Попередній захист	21.06.2024	*виконано*
13	Допуск до захисту у зав. кафедри	22.06.2024	*виконано*

Дата видачі завдання 29 березня 2024р.

Студент (ка) _____ Волохань В. В.
(підпис)

Керівник роботи _____ проф. Галуза О. А.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до кваліфікаційної роботи магістра містить: 59 с., 7 рис., 10 джерел.

ВЕБ-ДОДАТОК, ІНТЕРНЕТ, ФРЕЙМВОРК, ФРОНТЕНД РОЗРОБКА.

Об’єктом дослідження є процес розробки веб-додатків з використанням фронтенд фреймворків.

Метою роботи є проведення дослідження ефективності розробки веб-додатків з використанням фронтенд фреймворків та без них.

Методами розробки та проектування є аналіз проблемної області дослідження, вибір фреймворків Next та Vue для проведення дослідження шляхом порівняння особливостей та функціоналу фреймворку.

У результаті кваліфікаційної роботи було проаналізовано популярні фреймворки: Next, Vue, Angular, Svelte. Визначено особливості фреймворків та їх вплив на продуктивність веб-розробки.

WEB APPLICATION, INTERNET, FRAMEWORK, FRONTEND DEVELOPMENT.

The object of the study is the process of developing web applications using frontend frameworks.

The purpose of the work is to conduct a study of the efficiency of web application development using frontend frameworks compared to not using them.

The development and design methods are the analysis of the research problem area, the selection of the Next and Vue frameworks for the study by comparing the features and functionality of the frameworks.

As a result of the qualification work, popular frameworks such as Next, Vue, Angular, and Svelte were analyzed. The features of the frameworks and their impact on web development productivity were determined.

Заява щодо самостійного виконання кваліфікаційної роботи та можливості її публікації в електронному архіві відкритого доступу ElArKhNURE.

Я, Волохань Віктор Володимирович, студент(ка) гр. ІПЗм-22-3, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження ефективності розробки веб-додатків з використанням фронтенд фреймворків», 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(на) 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

ВСТУП

В сучасному світі, де високі технології швидко розвиваються та трансформують усі сфери нашого життя, розробка веб-додатків стає необхідністю та водночас складнощами для програмістів та розробників. Цей процес визначається не лише потужністю серверів чи глибиною баз даних, але й тим, наскільки зручним та ефективним є процес створення веб-додатку.

Користувачі сучасних веб-додатків стають все більше вимогливими щодо зручності використання, швидкості реакції та відповідності до їхніх очікувань. Задача розробників полягає в постійному вдосконаленні інтерфейсу, а це, в свою чергу, робить актуальним дослідження ефективності використання фронтенд фреймворків у процесі розробки веб-додатків.

Зростання конкуренції на ринку вимагає від розробників швидкості внесення змін у додатки, адже швидка відповідь на нові тренди може визначити успіх чи невдачу продукту. Тому використання фронтенд фреймворків може бути ключовим фактором у забезпеченні швидкості та ефективності процесу розробки.

Обрана тема не лише відображає актуальність в сучасному інформаційному суспільстві, але і визначає стратегічну вагу вибору фронтенд фреймворків у контексті розробки веб-додатків. Вона є ключовою для досягнення успішного впровадження та використання веб-додатків в умовах, де вимоги ринку постійно змінюються, а швидкість реакції на зміни стає визначальною для успіху.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Проблематика

Розробка веб-додатків є складним та багатогранним завданням, яке вимагає від розробників постійного вдосконалення та вибору оптимальних інструментів для досягнення максимальної ефективності. У сучасному інформаційному середовищі, де технології швидко змінюються, та конкуренція на ринку продуктів зростає, ключовою стає не лише можливість швидко розробляти новий функціонал, але і забезпечення високої якості та зручності використання веб-додатків кінцевими користувачами.

Однією з головних проблем є вибір та використання фронтенд фреймворків у процесі розробки веб-додатків. Це питання стає особливо актуальним у зв'язку з постійним поглибленням функціоналу та складністю користувацького інтерфейсу, а також ростом очікувань щодо продуктивності та відклику веб-додатків.

Проблема полягає в тому, що існує велика кількість фронтенд фреймворків, і кожен з них має свої особливості та переваги. Отже, виникає необхідність в об'єктивному порівнянні цих фреймворків та визначенні їхнього впливу на процес розробки. При неправильному виборі фреймворку може виникнути низка проблем, таких як складність утримання коду, повільна реакція інтерфейсу чи недостатня масштабованість.

Другою ключовою аспект, що потребує уваги, є ефективність розробки веб-додатків за допомогою фронтенд фреймворків. Розробка має бути не лише швидкою, але й якісною, а вибір фреймворку може суттєво вплинути на обидві ці характеристики. Зручність використання інструментів фреймворку, його гнучкість та можливість інтеграції з іншими технологіями можуть визначити загальний успіх проекту.

1.2 Постановка задачі

У цій роботі буде досліджено вплив використання фронтенд фреймворків на ефективність розробки веб-додатків. Для проведення магістерського дослідження буде обрана мова програмування JavaScript [1].

Для проведення магістерського дослідження будуть використані наступні фреймворки:

– Next.js - популярний відкритий фреймворк React для створення веб-додатків. Він розроблений для того, щоб спростити створення готових до використання виробів з React, забезпечуючи зручний досвід розробки та оптимізацію продуктивності;
– Vue.js - прогресивний JavaScript фреймворк для створення користувацьких інтерфейсів. Він призначений для розробки односторінкових додатків (SPA) та для легкої інтеграції в існуючі проекти.

Стосовно фронтенд фреймворків потрібно дослідити такі пункти:

– визначити переваги, недоліки та особливості їх використання;
– визначити типи веб-додатків, які найкраще підходять для використання з кожним фреймворком;
– порівняти функціональність, яку надають фреймворки, та зручність;
– визначити вплив інструментів, які надають фреймворки, на ефективність розробки.

У дослідженні важливо встановити як використання фронтенд фреймворків допомагає у вирішенні типових завдань, що виникають при розробці веб-додатків.

1.3 Аналіз веб-сайтів

Веб-сайти — це важливий елемент сучасної інтернет-екосистеми, який виконує різноманітні функції та впливає на багато аспектів життя суспільства. Створення веб-сайтів — це складний та динамічний процес, здатний задовольняти різноманітні потреби користувачів та підприємств. Аналіз веб-сайтів є важливим інструментом для розуміння їхньої суті, цільової аудиторії та ефективності функціоналу.

Веб-сайти стали необхідністю у сучасному світі, що стрімко розвивається в інформаційній та технологічній сферах. Вони можуть бути розроблені для різних

цілей, таких як представлення бізнесу, надання інформації, взаємодія з користувачами, електронна комерція, навчання та багато інших.

Веб-сайти мають різний дизайном, структуру та функціонал в залежності від його призначення. Візуальний дизайн впливає на перше враження від взаємодії з користувачем, тоді як структура та функціонал визначають зручність та ефективність використання.

Створення веб-сайтів має глибокі корені, починаючи з самого появу Інтернету. З розвитком технологій змінювалися підходи до їх розробки. Починаючи з простих статичних сторінок, ми перейшли до динамічних сайтів, які використовують складні бази даних та інтерактивні скрипти.

Раніше, створення веб-сайтів було досить важким та займало багато часу. В сучасному світі з появою фреймворків та бібліотек цей процес став набагато простішим та швидшим, адже фреймворки та бібліотеки надають зручні інструменти для розробки та оптимізації веб-сайтів.

Фреймворк - це структурована або складна система, яка надає основу для розробки програмного забезпечення. В контексті веб-розробки фреймворк - це набір підготовлених та передбачених правил, бібліотек, стандартів та інструментів, які допомагають розробникам створювати ефективне та консистентне програмне забезпечення.

Однією з ключових характеристик фреймворків є вони визначають архітектуру додатка та встановлюють правила, якими слід керуватися під час розробки. Це робить їх важливим інструментом для розробників, оскільки вони можуть спростити процес розробки, забезпечити масштабованість коду та полегшити співпрацю великих команд розробників.

1.4 Типи веб-додатків

Коли Інтернет тільки з'явився, слово "веб-сайт" було достатнім для розуміння. Однак за короткий час веб-сайти урізноманітнилися. Наприклад, з'явилися веб-сайти, де люди могли дізнаватися новини. Були сайти, на яких можна було шукати інформацію. Були окремі сторінки. Одного слова вже було

недостатньо. І незабаром виникла проблема розрізнення типів сайтів, але усталеної і широко використовуваної класифікації ще не було.

Справжня проблема в класифікації веб-сайтів полягає не в одному, а в кількох аспектах одночасно — як з точки зору їхньої функціональності, так і їхнього змісту. Наприклад, сайт, який одна особа використовує для висловлення власних думок, може бути визначений як індивідуальний блог. Але якщо цей же продукт, навіть з тим самим дизайном, використовується для поширення, скажімо, новин і наповнюється групою фахівців, то це вже може бути визначено як інформаційний ресурс.

Також може бути неможливо однозначно віднести ресурс до тієї чи іншої категорії. Часто веб-додатки можуть одночасно мати декілька призначень.

Очевидно, що не існує чітко визначеної кількості типів веб-сайтів, але визначимо головні та найбільш поширені з них:

- презентаційні сайти;
- посадкові сторінки;
- веб-сервіси;
- електронна комерція;
- соціальні мережі та комунікації;
- медіаресурси.

Проаналізуємо кожен із цих типів більш детально.

Презентаційні сайти, які повністю відповідають своєму призначенню, служать засобом представлення компаній або продуктів у віртуальному просторі. Їхня головна мета полягає в тому, щоб створити офіційне враження про компанію або бренд через Інтернет. Зазвичай такі веб-ресурси складаються з декількох ключових розділів.

Першим є розділ, який містить інформацію про компанію. Тут розміщена загальна інформація, така як історія компанії, її цінності та місія. Цей розділ дозволяє відвідувачам отримати загальне уявлення про компанію.

Другий розділ присвячений опису послуг і продуктів компанії. У цьому розділі користувачі можуть докладніше дізнатися про те, що пропонує компанія, які товари або послуги вона надає, і які переваги вони можуть отримати.

Третій розділ містить контактну інформацію. Тут вказані різні способи зв'язку з компанією, такі як телефонні номери, адреса електронної пошти чи фізична адреса. Це дозволяє потенційним клієнтам легко зв'язатися з представниками компанії.

Також презентаційні сайти можуть включати розділ із портфоліо, особливо для творчих компаній, де демонструються реалізовані проекти та досягнення. Крім того, зазвичай є розділ корпоративних новин, де публікуються офіційні оголошення, прес-релізи та, у деяких випадках, корпоративний блог.

Загалом такі сайти відіграють важливу роль у налагодженні ефективної комунікації з аудиторією та створенні сприятливого враження про компанію чи бренд у віртуальному середовищі.

Посадкові сторінки, часто відомі як лендінги, є спеціально створеними веб-сторінками, які мають конкретну мету та орієнтовані на конкретну аудиторію. Основна їхньою задачею є переконати відвідувачів виконати певну дію, таку як покупка продукту, заповнення форми, підписка на розсилку чи інші конверсійні цілі.

Розглянемо детальніше особливості та функції посадкових сторінок:

– орієнтація на одну мету. Посадкові сторінки часто створюються для досягнення конкретної мети чи дії. Наприклад, це може бути придбання продукту, заповнення форми, взяття участі в акції або завантаження електронної книги;
– сильні заклики до дії. Посадкові сторінки завжди мають виразні та привабливі "заклики до дії", які надихають відвідувачів на виконання конкретних дій. Це може бути кнопка "Придбати зараз", "Підписатися", "Отримати безкоштовну пробну версію" або інші;

– привабливий та зрозумілий дизайн. Дизайн посадкової сторінки часто простий та привабливий. Графіка та текстовий контент спрямовані на викликання позитивних емоцій та вразливість цільової аудиторії.

Веб-сервіси представляють собою онлайн-платформи, які надають різноманітні послуги користувачам через Інтернет. Це можуть бути різні види послуг, і вони включають в себе такі сайти, як електронна пошта, яка дозволяє отримувати та відправляти електронну пошту. Також до веб-сервіси, які пропонують перегляд відео та аудіоконтенту.

Такі сервіси можуть бути вкрай різноманітними, оскільки існує велика кількість різних видів послуг, які можна надавати онлайн. Наприклад, Google є онлайн-сервісом, який надає послуги з пошуку інформації.

Щоб уникнути непорозумінь і систематизувати це розмаїття, до іменника "сервіс" часто додається описовий прикметник. Такі означення, як "стрімінговий сервіс", "файлообмінний сервіс" або "платіжний сервіс", допомагають уточнити конкретний характер наданої послуги.

Електронна комерція, або інтернет-торгівля, представляє собою онлайн-сервіси для продажу товарів чи послуг, інакше кажучи, це віртуальні магазини, які функціонують через Інтернет. Не зважаючи на те, що вони стали невід'ємною частиною сучасного електронного простору, існує така велика кількість інтернет-магазинів, що їх логічно виділити у окремий тип веб-ресурсів.

Майже кожен інтернет-магазин має такі риси та функціонал:

– каталог товарів. Електронні комерційні сайти мають деталізований каталог товарів або послуг, де кожен елемент має свою сторінку з описом, зображеннями та цінами;
– система кошика та оформлення замовлення. Покупці можуть додавати товари в кошик, переглядати його вміст, вносити зміни та здійснювати оплату через онлайн-платіжні системи;
– опції фільтрації та пошуку. Користувачі можуть швидко знаходити необхідні товари завдяки системам фільтрації та пошуку за різними

параметрами що забезпучує зручну взаємодію клієнта з інтернет-магазином;

– профілі користувачів. Сайти електронної комерції надають можливість користувачам створювати власні профілі, керувати своїми персональними даними та переглядати історію замовлень.

Наступним типом веб-сайтів є соціальні мережі. Вони є типом веб-сайтів, де користувачі можуть створювати свої профілі, публікувати матеріали, встановлювати віртуальні зв'язки та об'єднуватися у групи [2]. Ця ідея мережі взаємодій була предметом вивчення психологів та антропологів за довгий час до настання інтернет-ери, описуючи систему міжособистих відносин.

Концепція соціальних мереж отримала цифрове втілення в 2010-х роках, коли інтернетові платформи стали основними майданчиками для цифрових зв'язків між людьми. Насправді соціальні мережі існували і раніше, але не були такими популярними.

Спільно з налагодженням віртуальних зв'язків, соціальні мережі та блоги також використовуються як платформи для поширення різноманітної інформації. Наприклад, Instagram став популярним для обміну фотографіями, а YouTube — для відеоматеріалів.

Крім того, месенджери, такі як Telegram, Messenger, Viber також можна вважати соціальними мережами, де широко поширюється текстова інформація. Таким чином, соціальні мережі виступають як цифровий еквівалент площадок для спілкування та обміну різними формами вмісту.

Медіаресурси представляють собою веб-сайти, спеціалізовані на розповсюдженні інформації через різноманітні медіаформати. Цей тип веб-сайтів зорієнтований на надання користувачам новин, аналітики, розважального контенту та інших мультимедійних ресурсів.

Медіаресурси можна класифікувати за кількома основними типами:

– новинні портали. Медіаресурси часто виступають у вигляді новинних порталів, де публікуються актуальні новини, події та репортажі. Це може

бути загальний новинний портал або спеціалізований за тематикою, такий як політика, технології, спорт тощо;

- відеоплатформи та стрімінгові сервіси. Деякі медіаресурси спеціалізуються на відеоконтенті. Це можуть бути стрімінгові платформи, як YouTube, або спеціалізовані відеоагрегатори, де користувачі можуть переглядати відео, документальні фільми, веб-серіали тощо;
- аудіо та подкасти. Медіаресурси також можуть включати аудіоконтент, такий як подкасти та онлайн-радіо. Користувачі можуть слухати аудіо-інтерв'ю, дискусії, новини чи розважальні програми;
- ігрові портали. Деякі медіаресурси спеціалізуються на відеоіграх, надаючи новини, огляди, стріми та інший відомісті в галузі геймінгу;
- фотогалереї та зображення. Деякі медіаресурси спеціалізуються на фотографіях та зображеннях, надаючи галереї, фоторепортажі та інші візуальні матеріали.

Важливо ще раз підкреслити, що наведений вище перелік типів веб-сайтів не є вичерпним. Він охоплює лише найбільш типові проекти, з якими сьогодні доводиться мати справу інтернет користувачам. Однак з розвитком інтернет-технологій сфера, в якій може бути використаний веб-сайт, обмежується лише нашою уявою.

1.5 Етапи розробки веб-додатку

Розробка веб-сайту включає в себе значний обсяг робіт, незалежно від розміру майбутнього проекту. Таким чином, для досягнення успішного результату необхідно уважно скласти поетапний план та слідкувати всім етапам створення веб-сторінки [3].

Створення сайту розпочинається з усвідомлення проблеми, що потребує вирішення. Сайт є надзвичайно потужним інструментом для успішного розвитку бізнесу. Чітке усвідомлення основної мети майбутнього проекту є невід'ємною складовою, яка має вирішальне значення для успіху бізнесу.

Не існує чітко визначеного плану розробки для усіх видів веб-сайтів. В залежності від проекту, команди розробників та інших факторів план може мати різну структуру.

Розглянемо структуру одного із планів розробки веб-сайтів. Нижче наведені пункти плану:

1. збір інформації;
2. планування;
3. проектування;
4. заповнення контентом;
5. функціональність;
6. тестування;
7. запуск;
8. моніторинг і оновлення.

Розглянемо кожен із пунктів плану детально.

1.5.1 Збір інформації

Перш ніж фактично створити веб-сайт, потрібно зібрати інформацію. Це буде включати мету, основні цілі та цільову аудиторію.

Мета є причиною створення цього веб-сайту. Потрібно визначити для чого створюється веб-сайт. Цілі - це те, буде досягнуто за допомогою цього веб-сайту. Визначені цілі дозволять краще уявити, як створити веб-сайт і який контент включити. Цільова аудиторія - це ті, до кого буде спрямований веб-сайт. У кожного сайту є цільова аудиторія тому потрібно визначити цільову аудиторію розроблюваного сайту перш ніж перейти до його розробки. Інформація про цільову аудиторію включає в себе вік, стать, інтереси, країну проживання, соціальний статус та багато іншої.

1.5.2 Планування

Після збору основної інформацію для розроблюваного веб-сайту, потрібно розпочати його планування. Використовуючи інформацію з першого етапу плану,

потрібно створити карту сайту. Карта сайту - це список всіх тем і підтем веб-сайту. Карта сайту допоможе уявити структуру веб-сайту та те, як користувач буде переходити з однієї сторінки на іншу. Цей етап надзвичайно важливий для того аби створити привабливий веб-сайт, який легко користуватися на переходити між його сторінками.

1.5.3 Проектування

Після розробки зовнішню структуру веб-сайту, потрібно визначити його зовнішній вигляд. Це включатиме всі візуальні елементи, такі як фотографії та відео. При проектуванні дизайну веб-сайту потрібно враховувати інформацію про його цільову аудиторію.

Для прикладу, веб-сайт з продажу дитячих іграшок матиме зовсім інший вигляд в порівнянні з сайтом продажу автівок. Веб-сайт повинен відповідати потребам та очікуванням його цільової аудиторії.

1.5.4 Заповнення контентом

Контент веб-сайту є одним із найважливіших аспектів. Він передає головне повідомлення аудиторії та заохочує їх використовувати веб-сайт. Але перед створенням контенту потрібно визначити цілі та мету. Контент повинен бути актуальним і достатньо цікавим, щоб користувачі поверталися до веб-сайту.

1.5.5 Функціональність

На цьому етапі починається фактичне створення веб-сайт. Саме на цьому етапі всі вищезазначені дії об'єднаються, щоб створити вигляд та функціональність веб-сайту. Веб-сайт повинен бути зручним для користувачів і легким у користуванні. Зазвичай спочатку створюється домашня сторінка, а потім створюються всі інші підсторінки. Також потрібно створити адаптивність як для комп’ютерної версії так і для мобільних пристроїв.

1.5.6 Тестування

Як тільки веб-сайт успішно створений, він ще не готовий до запуску. Необхідно спочатку його протестувати. Тестування гарантує, що веб-сайт працює так як він повинен працювати. Під час цього етапу необхідне тестування всіх посилань і кнопок на веб-сайті, перевірка правильності написання тексту та переконатися, що веб-сайт виглядає однаково як на телефоні, так і на комп'ютері.

1.5.7 Запуск

Після ретельної перевірки веб-сайту, наступним етапом є його запуск. Для запуску веб-сайту потрібно завантажити його на сервер та налаштувати. Налаштування включає в себе такі пункти: вибір хостингу, вибір доменного імені, завантаження та налаштування бази даних, SEO налаштування, захист, резервне копіювання та інші. Після запуску веб-сайт стане доступним усім користувачам інтернету.

1.5.8 Моніторинг та оновлення

Незважаючи на успішний запуск веб-сайту, важливо час від часу повертатися до нього і перевіряти його помилки, які можуть виникати в процесі його роботи. Якщо будуть помилки то потрібно якнайшвидше виправляти їх та тримати веб-сайт і інформацію, що розміщена на ньому, в актуальному стані. Тому важливо відстежувати стан веб-сайту для того, щоб підтримувати його в ідеальному стані.

1.6 Фронтенд фреймворки

Спочатку розглянемо, що таке фронтенд фреймворк. Фронтенд фреймворк - це набір базових інструментів, бібліотек і готових компонентів. Ці фреймворки мають багаторазові компоненти (фрагменти коду) для JavaScript, CSS і HTML, які розробники можуть використовувати в різних проектах [4].

Фреймворки оптимізують процес розробки, забезпечуючи плавні переходи у взаємодії з користувачем і більш "застосункоподібний" вигляд під час взаємодії з користувачем. Фреймворки є важливими інструментами, які дозволяють

розробникам створювати цікаві та ефективні цифрові враження, роблячи процес розробки веб-додатків більш структурованим, організованим та зручним для розробника.

Головними перевагами використання фреймворків є:

- сильна підтримка спільноти. Фронтенд фреймворки мають великі спільноти, які готові допомогти та поділитися знаннями. Це забезпечує розробникам можливість швидко отримати відповіді на їх питання та вирішувати проблеми завдяки великому обсягу досвіду, який накопичується в цій спільноті;
- можливість повторного використання. Фреймворки мають повторно використовувані компоненти, що дозволяє розробникам використовувати фрагменти коду в різних проектах. Такі компоненти можуть надаватися розробникам «з коробки» або бути створені самими розробниками. Це спрощує розробку, зменшує кількість написаного коду та полегшує його підтримку;
- швидка розробка. Використання фронтенд фреймворків сприяє швидкому розвитку проектів. За рахунок готових компонентів та оптимізованих інструментів, розробка веб-додатків стає ефективнішою, прискорюється виробничий процес та полегшується масштабування проекту;
- зменшення витрат. Використання фронтенд фреймворків може призвести до зменшення витрат на розробку. Готові рішення та ефективні інструменти дозволяють оптимізувати робочі процеси, що в свою чергу впливає на зниження витрат на проект;
- легкість залучення фахівців. Фреймворки є популярними серед розробників, тому залучити кваліфікованих фахівців до проектів, які використовують ці технології, стає досить легкою задачею. Це забезпечує доступність професіоналів та полегшує формування ефективних розробницьких команд.

Отже, як бачимо, використання фронтенд фреймворків має безліч переваг, що в свою чергу впливає на ефективність розробки.

2 РОЗРОБКА ВЕБ-ДОДАТКІВ З ВИКОРИСТАННЯМ ФРЕЙМВОРКІВ

2.1 Загальний огляд популярних фронтенд фреймворків

Фронтенд фреймворк - це набір стандартизованого, попередньо написаного коду, який розробники використовують для розробки веб-додатків. Фронтенд фреймворки надають базову структуру, використовуючи яку можна ефективно створювати візуальні елементи веб-сторінок, з якими користувачі безпосередньо взаємодіють та бачать. Наприклад такими елементами можуть бути кнопки, форми, зображення та інші. Ці фреймворки зазвичай вже мають набір правил та повторно використовуваних компонентів, які спрощують загальний процес розробки, роблячи його швидшим та надійнішим. З їх допомогою розробники можуть зосередитися на налаштуванні дизайну та функціональності сайту, не будуючи кожен елемент з нуля.

Веб-фреймворки забезпечують просту, зрозумілу структуру, яку легко підтримувати і яка дозволяє розробникам створювати потужні веб-додатки за короткий час. Для досягнення цієї мети багато веб-фреймворків базуються на трьох основних принципах проектування [5]:

- DRY (Don't Repeat Yourself). Цей принцип спрямований на уникнення коду який повторюється при розробці програмного забезпечення. Основна ідея принципу DRY полягає у створенні єдиного вірного представлення інформації в системі. Усунення дублюючого коду підвищує стійкість коду, зменшує складність і мінімізує ризик помилок. Дублювання коду призводить до неузгодженостей і ускладнює внесення змін у майбутньому;
- KISS (Keep it short and simple). Одна проблема - одне рішення – це основний принцип підходу KISS. Якщо є кілька підходів до вирішення проблеми, слід обирати рішення з найменшою кількістю припущень і змінних. Для вирішення конкретної проблеми слід використовувати якомога менше коду;

– конвенція замість конфігурації (Convention over Configuration). Головною метою є зменшення кількості рішень, які доводиться приймати програмістам, і усунути складність налаштування всіх аспектів розробки додатків. В результаті вони можуть створювати більше за менший час. Саме тому веб-фреймворки повинні надавати найкращі підходи у своїх налаштуваннях за замовчуванням, а також пропонувати додаткові варіанти налаштування за бажанням.

Використання цих принципів дизайну веб-фреймворками у розробці програмного забезпечення має безліч переваг. Однак, особливості фреймворків також обмежують розробників. Ці обмеження іноді сприймаються як недолік.

Розглянемо загальні переваги використання фреймворків. Їх використання не завжди є необхідністю, особливо для не великих проектів. Але іноді розробка без фреймворку чи бібліотеки може стати великою проблемою.

Переваги використання фронтенд фреймворків:

– підвищення продуктивності. Фронтенд фреймворки забезпечують структурований та організований підхід до веб-розробки, що дозволяє спростити процес розробки та підвищити продуктивність. Ці фреймворки надають готові компоненти, шаблони та бібліотеки, які дозволяють розробникам зосередитися на логіці програми, а не на повторюваному написанні коду з нуля;
– багаторазове використання та модульна структура. Фронтенд фреймворки поширюють архітектуру на основі компонентів, де елементи користувацького інтерфейсу є компонентами які можна використовувати повторно. Такий модульний підхід спрощує використання коду і зменшує кількість дублюючого коду, який потрібно писати. Компоненти можна легко підтримувати, тестувати та використовувати у різними частинах програми;
– ефективне відображення та оптимізація продуктивності. Багато фреймворків використовують такі методи, як порівняння віртуального DOM та інтелектуальне оновлення для оптимізації продуктивності

рендерингу. Фреймворки та бібліотеки покращують загальну швидкість і реактивність веб-додатків та забезпечують безперебійну роботу користувачів тому що мінімізують непотрібні повторні рендери та ефективно оновлюють DOM;

- управління станами. Фронтенд фреймворки часто включають рішення для управління станами, які допомагають керувати та синхронізувати дані програми. Ці бібліотеки керування станами спрощують роботу зі складними потоками даних і полегшують керування змінами стану програми;
- великі та активні спільноти. Найпопулярніші фронтенд фреймворки, такі як React, Vue та Angular, мають великі та активні спільноти. Ці спільноти розробляють бібліотеки, плагіни та інструменти, які розширюють функціональність фреймворків та дозволяють іншим розробникам використовувати їх рішення. Такі спільноти також дбають про постійну підтримку, часті оновлення та безліч навчальних ресурсів.

Перед оглядом конкретних фреймворків, варто також розглянути недоліки з якими часто стикаються розробники. Так як в Інтернеті існує велика кількість фреймворків для веб-розробки, всі вони відрізняються як за основними принципами проектування, так і за функціональністю, яку вони надають. Різні програмні проекти можуть потребувати використання різних веб-фреймворків. Розробники стикаються з проблемою вибору правильного фреймворку для проекту, який вони розробляють. Тому при його виборі потрібно детально розглянути та враховувати особливості та обмеження які приносить фреймворк. Оскільки фреймворки розробляються як універсальне рішення, розробники рідко використовують всі можливості конкретного фреймворку. Залежно від обсягу фреймворку, додаток може містити більше коду, ніж потрібно.

Також великим недоліків використання фреймворків є навчання. Кожен фреймворк має власні правила та практики, які потребують часу для вивчення. Це може бути особливо складно для розробників-новачків. Використання фреймворку також означає, що розробник залежить від цього фреймворку. Якщо фреймворк

більше не підтримується, то розробникам може знадобитися вивчити новий фреймворк або переписати додаток використовуючи інші технології.

Часто фреймворки мають вбудовані функції для вирішення задач які часто зустрічаються. Наприклад, такими задачами може бути створення навігації у додатку або керування станами. Так як ці рішення вже надає фреймворк, іноді розробники можуть не знати особливості їх реалізації, що в свою чергу може спричити виникнення помилок при розробці.

Після огляду загальних переваг перейдемо до детального огляду популярних фреймворків.

Next.js - це фреймворк в основі якого лежить бібліотека React, він дозволяє створювати великі, оптимізовані для пошукових ботів та користувацьки-орієнтовані статичні веб-сайти та веб-застосунки. Next.js є зручним у використанні розробниками при створенні готових до використання додатків з усіма потрібними функціями. Він підтримує Server-Side Rendering (SSR), Client-Side Rendering (CSR), Static Site Generation (SSG), має підтримку TypeScript, розумний інструмент для збору додатку, та інші можливості, для яких не потрібна додаткова конфігурація.

Розглянемо особливості фреймворку Next.js:

- можливості React.js. Next.js побудований на основі бібліотеки React.js, а отже має всі переваги та особливості цієї бібліотеки, а також додає нові можливості рендерингу на стороні серверу (SSR);
- розділення коду. Розбиття коду Next.js на менші фрагменти дозволяє завантажуючи лише необхідні частини при переході користувача через додаток. Це призводить до оптимізації продуктивності, швидшого завантаження сторінок та кращого користувацького досвіду. Розробники також можуть використовувати динамічні імпорти, щоб явно контролювати розбиття коду;
- навігація. Next.js спрощує процес визначення навігації між сторінками у додатку. Структура каталогу сторінок автоматично має відповідну структуру з навігацією, що дозволяє розробникам організувати свій код в

інтуїтивно зрозумілий спосіб. Вкладені папки в каталозі сторінок створюють вкладені маршрути;

- підтримка TypeScript. Next.js має чудову інтеграцію з TypeScript, що дозволяє розробникам використовувати статичну типізацію як у серверному, так і в клієнтському коді; TypeScript допомагає виявити потенційні помилки під час розробки та покращити якість коду і підтримку;
- підтримка middleware. Next.js підтримує middleware, що дозволяє розробникам запускати код перед обробкою запитів. Це корисно для таких завдань, як автентифікація, ведення журналів і модифікація об'єктів запитів і відповідей. Функції проміжного програмного забезпечення можна додавати до файлу middleware.ts;
- оптимізація зображень. Next.js спрощує оптимізацію зображень, автоматично обробляючи їхнє завантаження та стиснення. Фреймворк підтримує компонент next/image, який дозволяє розробникам оптимізувати зображення для різних пристроїв і розмірів екранів. Він також надає такі функції, як ліниве завантаження та автоматичне створення адаптивних наборів зображень, щоб покращити продуктивність та зручність для користувачів;
- міжнародна локалізація (i18n). Next.js полегшує розробникам створення додатків, що підтримують кілька мов, завдяки вбудованій підтримці інтернаціоналізації. Налаштувавши файл next.config.js і використавши бібліотеку next-translate, розробники можуть керувати перекладами та надавати користувачам додатком з підтримкою багатьох мов.

Фреймворк Next.js варто використовувати якщо:

- SEO-оптимізаціє є важливою для веб-додатку;
- продуктивність та швидкість роботи важливі;
- додаток має складні інтерактивні сторінки.

Не варто використовувати Next.js якщо:

- веб-додаток простий та малий за розміром;

– SEO-оптимізація не має значення.

Наступним фронтенд фреймворком розглянемо Vue.js. Він створений Еваном Ю., набув популярності завдяки своїй простоті та легкості інтеграції. Це прогресивний фреймворк, який можна впроваджувати поступово. Його особливостями є:

– легка інтеграція. Vue.js можна легко інтегрувати в існуючі проекти, що робить його ідеальним для поетапного розгортання;
– реактивне зв'язування даних. Vue.js має систему реактивного зв'язування даних, яка автоматично оновлює користувацький інтерфейс при зміні даних;
– компонентна архітектура. Vue.js використовує компонентну архітектуру, яка дозволяє створювати багаторазові та модульні компоненти які можуть бути легко використані в різних частинах додатку за потреби;
– легкий і швидкий. Vue.js легкий і не має багато вбудованих функцій, що дає розробникам гнучкість у виборі інструментів та бібліотек, які потрібні для їх проекту;
– навігація та керування станами. Vue.js має власний маршрутизатор (Vue Router) та бібліотеку управління станом (Vuex), що забезпечує полегшує та покращує процес розробки.

Фреймворк Vue.js краще використовувати для додатків:

– малі та середні за розміром;
– швидкість та простота у використанні – важливі.

Краще не використовувати для великих за розміром веб-додатків.

Наступний фреймворк – Angular. Angular розроблений компанією Google та широко визнаний у спільноті інженерів програмного забезпечення. Має довгу історію від AngularJS до сучасних версій Angular на базі TypeScript. Angular вдалося залишатися розробленим з урахуванням потреб підприємств у створенні додатків великого масштабу, маючи ряд різноманітних функцій, які полегшують розробку складних додатків. Angular дотримується повної архітектури MVC (Model-View-Controller) та надає широкий спектр функцій. Він надає набір

інтегрованих бібліотек для маршрутизації, керування формами, комунікації клієнт-сервер та багатьох інших. Має фокус на продуктивності, масштабованості та розмірі додатку та продовжує розвиватися, щоб задовольняти потреби сучасних фронтенд-розробників.

Розглянемо особливості Angular:

- Angular CLI. Angular CLI (Command Line Interface) - це потужний інструмент для розробників, який дозволяє створювати, збирати та підтримувати Angular-проекти. Він надає командний рядок для генерації вихідного коду проекту, створення компонентів, сервісів та модулів, а також запуску сервера розробки та процесу збірки;
- обробка форм. Цей фреймворк має вбудовану підтримку обробки форм, включаючи валідацію форм та прив'язку даних;
- розділення коду. Розділення коду в Angular розділяє код програми на менші модулі, які за неохідності можна завантажити. Ця техніка допомагає зменшити час завантаження програми, адже дозволяє завантажувати лише код, необхідний для поточної сторінки або функції. В результаті користувачі відчувають прискорення завантаження та покращення продуктивності веб-додатку;
- навігація. Angular має потужний маршрутизатор для керування навігацією додатку, що має декілька сторінок;
- тестування. Angular надає повну підтримку тестування за допомогою таких інструментів, як Jasmine та Karma, що дозволяє розробникам писати модульні, інтеграційні та наскрізні тести для своїх додатків;
- керування станами. Angular надає кілька варіантів керування станами, включаючи роботу з зовнішніми бібліотеками, такими як NgRx, для керування складними станами у великих додатках, та керування локальними станами всередині компонентів.

Коли варто використовувати фреймворк Angular:

- підприємницькі застосунки великого масштабу;
- додатки, які потребують багатофункціональності;

– проекти, які мають переваги від створення на TypeScript.

Не слід використовувати Angular для простих та малих розміром веб-додатків.

Svelte - це сучасний JavaScript-фреймворк з унікальним підходом до побудови користувацьких інтерфейсів. На відміну від традиційних фронтенд-фреймворків, які покладаються на віртуальний DOM, Svelte переносить більшу частину роботи на час компіляції, що призводить до високоефективного і продуктивного коду.

Ключові особливості Svelte:

– реактивність. Svelte дозволяє створювати реактивні компоненти за допомогою простого синтаксису. Він використовує підхід реактивного оголошення, який дозволяє оголошувати змінні реактивно і автоматично оновлювати DOM при зміні їх значень;
– локальні стилі. Фреймворк має вбудовану підтримку ізольованих CSS стилів, який інкапсулює стилі компонентів і дозволяє уникнути колізій CSS. Ця функція спрощує стилізацію та покращує підтримку програми;
– ефективність. Цей фреймворк генерує високооптимізований код, який безпосередньо маніпулює DOM для швидкої та ефективної обробки. Завдяки відсутності віртуальної DOM і накладних витрат під час виконання, він є легким та ідеальним для проектів, де продуктивність є пріоритетом;
– компілятор. Svelte має компілятор, який аналізує код компонентів під час процесу збірки і генерує оптимізований JavaScript-код. Такий підхід усуває потребу в бібліотеках, зменшує розмір пакета та пришвидшує початковий рендерингу;
– компонентний підхід. Як і більшість фреймворків Svelte пропонує компонентну архітектуру, яка дозволяє створювати елементи інтерфейсу користувача багаторазового використання. Компоненти інкапсулюють власну логіку та стилі, що полегшує розробку модулів та повторне використання коду.

Як бачимо, головною особливістю Svelte є пріоритет на швидкості роботи, а отже він найкраще підходить до проектів де швидкість та ефективність є ключовими критеріями. Також добре підходить для проектів для яких важливо мати малий розмір пакетів та бути реактивними.

Фреймворк Svelte не підходить для веб-додатків які:

- вже написані з використанням іншого фреймворку, адже не пропонує нових функцій;
- потребують велику кількість сторонніх бібліотек.

Як бачимо, більшість популярних фреймворків пропонують схожий функціонал для розробки веб-додатків, але мають різні підходи до їх створення. Вони мають різні пріоритети як наприклад ефективність, швидкість, SEO-оптимізація та створені для розробки різних за розмірами веб-додатків. Саме тому не має одного найкращого фреймворку який був би вирішенням всіх проблем для всіх проектів. Тому при виборі фреймворку для розробки веб-додатку важливо враховувати всі плюси, мінуси та особливості, що пропонує вибраний фреймворк.

2.2 Проблема та їх вирішення за допомогою фреймворків

Веб-фреймворк - це набір вихідного коду або бібліотек, які забезпечують загальну функціональність додатків. Якщо окрема бібліотека зазвичай забезпечує певну функціональність, фреймворк надає широкий спектр функціональностей, які часто використовується в додатках. Програмісти можуть обмежити час, необхідний для створення програми, і зменшити ймовірність появи нових помилок використовуючи фреймворки, які надають функціональність, що часто використовується для розробки веб-додатків, замість того, щоб переписувати загальновживану логіку.

Для прикладу, функціоналом, який надають веб-фреймворки, може бути керування станами, сесіями користувачів, внутрішнім сховищем. Також існують й інші типи фреймворків. В залежності від того в якій сфері розробки вони застосовуються та для вирішення яких проблем були створені, вони надають розробникам різні можливості. Наприклад, вони так можуть надавати інструменти

для перевірки та валідації даних, обробки запитів та помилок, роботи з файлами, графічні елементи, віджети, кешування та інші.

Навіть для одного типу додатків існують різні типи фреймворків. Деякі з них надають найнеобхідніші можливості з лише кількома зручними функціями. Інші надають майже все необхідне для створення додатку і вимагають використання суворого набору правил організації коду та інших правил. Вибір найкращого фреймворку часто вимагає від програміста балансу між функціональністю, яку можна отримати від фреймворку, і гнучкістю.

Розглянемо детально як веб-фреймворки вирішують задачі та проблеми, що часто виникають у процесі розробки програмного продукту.

2.2.1 Створення додатку

Розглянемо процес створення додатку на прикладі фреймворків Next.js та Vue.js. Існує декілька способів створення нового додатку. Найпростішим серед них є використання CLI (Command Line Interface).

CLI - це програмний механізм взаємодії з операційною системою за допомогою клавіатури [6]. Іншим механізмом є графічний інтерфейс користувача (GUI), який сьогодні широко використовується у всіх прикладних і програмних системах. Графічні інтерфейси дозволяють користувачам здійснювати візуальну навігацію і натискати на іконки та зображення, щоб виконати певні дії. Однак графічні інтерфейси неефективні для завдань системного адміністрування, особливо у віртуальних або віддалених середовищах. За допомогою інтерфейсу командного рядка можна вводити текстові команди для налаштування, навігації та запуску програм на будь-якому сервері або комп'ютерній системі, включаючи всі операційні системи, в тому числі Linux, macOS і Windows, CLI для швидшої взаємодії з системою.

Основними перевагами використання CLI є:

– ефективність. Замість того, щоб витрачати час на пошук і натискання окремих файлів, CLI дозволяє використовувати командний рядок для роботи з декількома файлами за допомогою однієї текстової команди.

Також можна створювати скрипти, які запускають кілька команд у командному рядку для автоматизації монотонних або повторюваних завдань;

– віддалений доступ. CLI дозволяє віддалено керувати серверами, надсилаючи команди, навіть зі з'єданнямя, що має низьку пропускну здатністю. Таким способом часто керують серверами та хмарними екземплярами, де графічні користувацькі інтерфейси недоступні.

Для створення додатку за допомогою CLI з фреймворком Next використовують наступну команду: «npx create-next-app@latest». Для створення додатку на Vue використовують команду: «vue create [name]». Після введення цих команд необхідно пройти швидку конфігурацію додатку (рис.2.1).

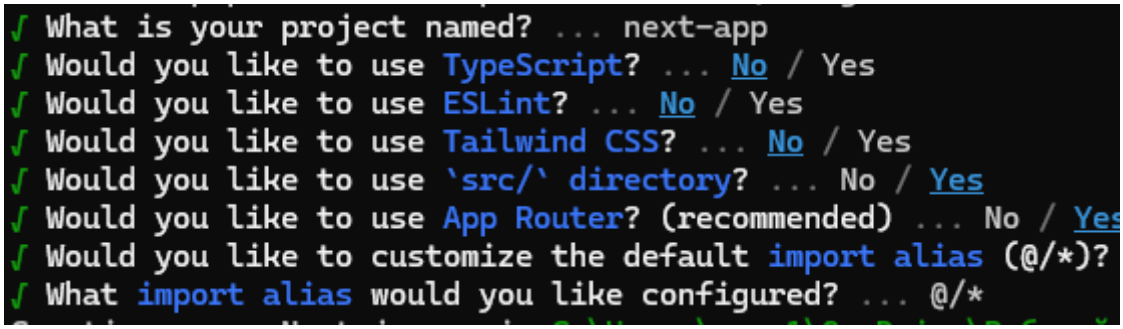

Рисунок 2.1 – Конфігурація додатку при створенні з Next.js

Після конфігурації додаток готовий до запуску.

Для створення додатку на Vue існує й інший спосіб за допомогою CDN. Для цього потрібно імпортувати скрипт, що підключає Vue. Такий спосіб краще підходить коли вже є існуючий проект, і потрібно керувати лише певною частиною додатку за допомогою Vue.

Тепер розглянемо як можна створити найпростіший додаток без використання фреймворків, а лише за допомогою HTML (Hyper Text Markup Language), CSS (Cascading Style Sheets), JavaScript. Для цього потрібно виконати наступні кроки:

1. створити каталог в якому буде зберігатися вихідний код додатку;
2. у каталозі вручну створити 3 файли що мають формати: *.html, *.css, *.js;

3. записати базову структуру HTML-документу у відповідний файл;
4. підключити файли стилів та JavaScript коду.

Як бачимо, процес створення нового додатку зручніший та простіший з використанням фреймворків та CLI, оскільки вимагає лише введення однієї команди та швидкої конфігурації, в той час як створення навіть найпростішого додатку без фреймворку вимагає багато ручної та монотонної роботи.

2.2.2 Синтаксис

Перед розглядом та детальним аналізом синтаксису конкретних фреймворків визначимо що таке синтаксис у контексті програмування та чому він такий важливий.

Синтаксис - це правила побудови граматично правильних речень. Аналогічно, у комп'ютерному програмуванні синтаксис - це правила, що регулюють структуру символів, слів і розділових знаків у мовах програмування.

Якщо б в мовах якими ми розмовляємо не було синтаксису, то кожному довелося б вигадувати власні правила для порядку слів і структури речень, а розуміння і сприйняття швидко б зруйнувалося.

Так само синтаксис у мовах програмування визначає, що означають різні комбінації літер. Синтаксис також визначає ключові слова та символи, які програмісти можуть використовувати для написання вихідного коду. Однак комп'ютери дуже вимогливі до читання коду. Будь-яка помилка у синтаксисі призведе до синтаксичної помилки і комп'ютер не зможе запустити код.

Щоб опанувати будь-яку мову програмування або фреймворк потрібно розуміти його синтаксис. Написання коду відповідно до встановлених структурних правил полегшує розуміння коду машинами та людьми.

Отже, розберемо синтаксис, який використовується у популярних веб-фреймворках на прикладі Next. Цей фреймворк використовує JSX (JavaScript Syntax eXtension) [7].

JSX - це розширення синтаксису JavaScript, яке дозволяє розробникам писати HTML-подібний код у файлах JavaScript. Воно було розроблене як простіший та

інтуїтивно зрозуміліший спосіб створення користувацьких інтерфейсів за допомогою React. Замість того, щоб вручну маніпулювати DOM, розробники можуть використовувати JSX для декларативного визначення компонентів та їхніх властивостей.

Важливо розуміти, що синтаксис JSX виглядає схожим на HTML, але це не одне і те ж саме. JSX перетворюється у виклик JavaScript функцій (рис. 2.2).

Однією з найпотужніших можливостей JSX є те, що він може мати динамічний вміст. Це означає, що JavaScript можна використовувати для обчислення значень атрибутів і дочірніх елементів у коді JSX.

```
const Home = () => {
    return (
        <>
            <div className="container">
                <h1>Hello, { name }!</h1>
            </div>
            <ul>
                { list.map(elem => {
                    return <li key={ elem.id }>{ elem.name }</li>
                })}
            </ul>
        </>
    );
};
```

Рисунок 2.2 – Приклад JSX коду

Розглянемо правила яких потрібно дотримуватися при використанні JSX:

– повертати один кореневий елементи. Під капотом JSX перетворюється на звичайні об'єкти JavaScript. У JavaScript неможливо повернути два об'єкти з функції без обгортання їх у масив. Тому не можна повернути два JSX теги без обгортання їх у інший тег або фрагмент;
– закривати всі теги. JSX вимагає закриття усіх тегів;

- не використовувати конструкції if-else. У JSX не можна використовувати конструкції if-else, але замість них можна тернарний оператор;
- використовувати camelCase для атрибутів. JSX конвертується в JavaScript, і атрибути, записані в JSX, стають ключами для JavaScript-об'єктів. В JavaScript є обмеження щодо назв змінних. Наприклад, їх назви не можуть містити дефісів або бути зарезервованими словами, такими як class. Тому багато атрибутів HTML та SVG записуються у camelCase;
- при створенні власних компонентів використовувати PascalCase. Згідно з конвенцією, потрібно називати власні компоненти з великою літери, щоб відрізняти їх від звичайних HTML тегів та інших змінних у коді. Це допомагає покращити читабельність коду та зрозуміти, що це саме є компонентом.

Перейдемо до розбору синтаксису, який викостовує фреймворк Vue.

Vue використовує HTML-подібний синтаксис шаблонів, який дозволяє відображеному DOM-елементу декларативно зв'язуватись з даними. Vue компілює шаблони у високооптимізований JavaScript код. У поєднанні з системою реактивності Vue може інтелектуально визначати мінімальну кількість компонентів, які потрібно переробити, і застосовувати мінімальні маніпуляції з DOM при зміні стану додатку.

Vue використовує директиви. Це спеціальні атрибути що мать префікс «v-». Vue надає низку вбудованих директив, але розробники мають можливість створювати власні. Вони використовуються для прив'язки даних, додання обробників подій, контрольованого відображення елементів та інших задач. Деякі директиви в Vue можуть приймати аргумент, який вказується за допомогою двокрапки після назви директиви.

Перейдемо до розгляду стандартної структури та синтаксису компонету створеного у додатку з використанням Vue. Він складається з трьох секції (рис. 2.3):

- script. В цій секцій знаходиться JavaScript код та логіка компоненту;
- template. Секція «template» у Vue.js містить HTML-код, який визначає структуру компонента або сторінки. Цей код може включати в себе

динамічні дані та вирази, які будуть замінені реальними даними при рендерингу компонента або сторінки. Для інтерпретації даних як звичайний текст використовуються подвійні хвилясті дужки;

– style. В цій секції зберігаються стилі для компонента.

Як бачимо, така структура дозволяє чітко відокремити код за його призначенням, що полегшує розуміння та підтримку компонентів.

```
<script>  Show component usages
export default {
    name: 'HelloWorld',
    props: {
        msg: String
    },
    data() {
        return {
            list: [{ id: 1, name: "Alex"}]
        }
    }
};
</script>

<template>
    <div class="hello">
        <h1>Hello {{ msg }}</h1>
        <ul>
            <li v-for="elem in list" :key="elem.id">{{ elem }}</li>
        </ul>
    </div>
</template>

<style scoped>
h1 {
    margin: 40px 0 0;
}
</style>
```

Рисунок 3.3 – Приклад стандартної структури Vue-компонента

Перейдемо до огляду директив, що найчастіше використовуються:

- v-show. Дозволяє змінювати видимість елементу в залежності від переданого значення. Елемент завжди відображається та залишається в DOM, а змінюється лише його CSS властивість «display»;
- v-if та v-else. Дозволяють умовно відображати елементи відповідно до умови, залежно від правдивості вказаного виразу. При перемиканні, елемент та всі директиви/компоненти, що містяться всередині, будуть видалені та перевідображені заново;
- v-for. Ця директива дозволяє багаторазово відображати елементи на основі вхідних даних. Значення директиви має використовувати спеціальний синтаксис «alias in expression», щоб оголосити змінну для поточного елемента ітерації;
- v-on. Додає до елементу обробник події, а тип події визначається аргументом. Значеннями можуть бути іменами методів або вирази, що записані у тому ж рядку. При використанні на звичайних елементах відстежуються лише події DOM, а при використанні на власноруч створених компонентах відстежуються користувацькі події. Через те, що директива часта використовується, існує короткий спосіб її запису через символ «@»;
- v-bind. Ця директива дозволяє динамічно прив'язувати один або декілька атрибутів або вхідних параметрів компонента до виразу. Через те, що директива часта використовується, існує короткий спосіб її запису через двокрапку;
- v-model. Створює двосторонній зв'язок між даними в додатку та полем для вводу даних. Це дозволяє автоматично оновлювати дані додатку, коли користувач вносить зміни до поля, дані автоматично оновлюються.

Отже, такий підхід з використанням директив Vue дозволяє ефективно керувати DOM, що значно спрощує та полегшує розробку веб-додатків.

Перейдемо до розгляду синтаксису, що використовується у додатках створених без використання фреймворків та інших інструментів. Такі застосунки

не використовують спеціальний синтаксис, а мають лише вбудовану підтримку HTML, CSS та JavaScript.

Головною проблемою такого підходу є керування DOM. Без спеціального синтаксису, розробнику доводиться вручну виконувати всі маніпуляції з DOM за допомогою чистого JavaScript.

Хоча існують успеціальні методи для доступу до DOM-елементів, їх створення, заміни, видалення, модифікації, такий спосіб взаємодії з DOM може призвести до створення коду, який важко читати та підтримувати. Це особливо проблематично у великих проектах або при спробі підтримувати код, написаний іншими розробниками. Без організованої структури та чіткого розподілу коду важко визначити, що робить кожна частина і як вона взаємодіє з рештою системи.

Як бачимо, синтаксис фреймворків, забезпечує більш організований та ефективний спосіб розробки веб-додатків. Основна перевага полягає в тому, що фреймворки значно спрощують процес розробки, надаючи спеціалізований синтаксис та інструменти для роботи з DOM. Фреймворки забезпечують декларативний підхід до веб-розробки, який не вимагає від розробників маніпулювання DOM безпосередньо. Натомість вони дозволяють розробникам фокусуватися на бізнес-логіці додатку, а не маніпуляції DOM. Це дозволяє підвищити продуктивність та знизити кількість можливих помилок.

2.2.3 Архітектура

Цифровий світ розвивається завдяки веб-додаткам, але для створення ефективних та перспективних додатків вимагає більше, аніж декілька рядків коду. Стрімкий розвиток веб-розробки створив безліч фреймворків, інструментів і додатків, створивши складну екосистему, яка вимагає глибокого розуміння.

Створити ефективний та надійний веб-додаток так само складно, як побудувати багатофункціональну будівлю. Тому розробники клієнтських застосунків часто стикаються з труднощами пов’язаними з архітектурою додатків. Їм потрібно використовувати архітектуру, яку легко розширювати і забезпечує вільний зв'язок і високу узгодженість між частинами програми.

Перед початком огляду популярних архітектурних підходів, що використовуються у веб-застосунках, розберемося що таке архітектура.

Архітектура веб-додатків - це структурна основа, що визначає дизайн та розробку веб-додатка. Вона включає набір принципів та кращих практик, які визначають як різні частини додатка взаємодіють між собою для виконання його функцій.

Головним завданням архітектури полягає в створенні єдиної та масштабованої системи, яка відповідає функціональним потребам додатка, а також враховує такі фактори, як продуктивність, безпека та підтримуваність.

Розглянемо, визначимо переваги та недоліки таких архітектур [8]:

– FSD (Feature-Sliced Design);

– класична.

FSD - це архітектурна методологія створена для побудови фронтенд додатків. Простіше кажучи, це набір правил і умовностей для організації коду. Основна мета цієї методології - зробити проекти більш зрозумілими та структурованими в бізнес-середовищі.

Згідно цієї методології додаток складається з трьох частин: шари, секції, сегменти (рис. 2.4).

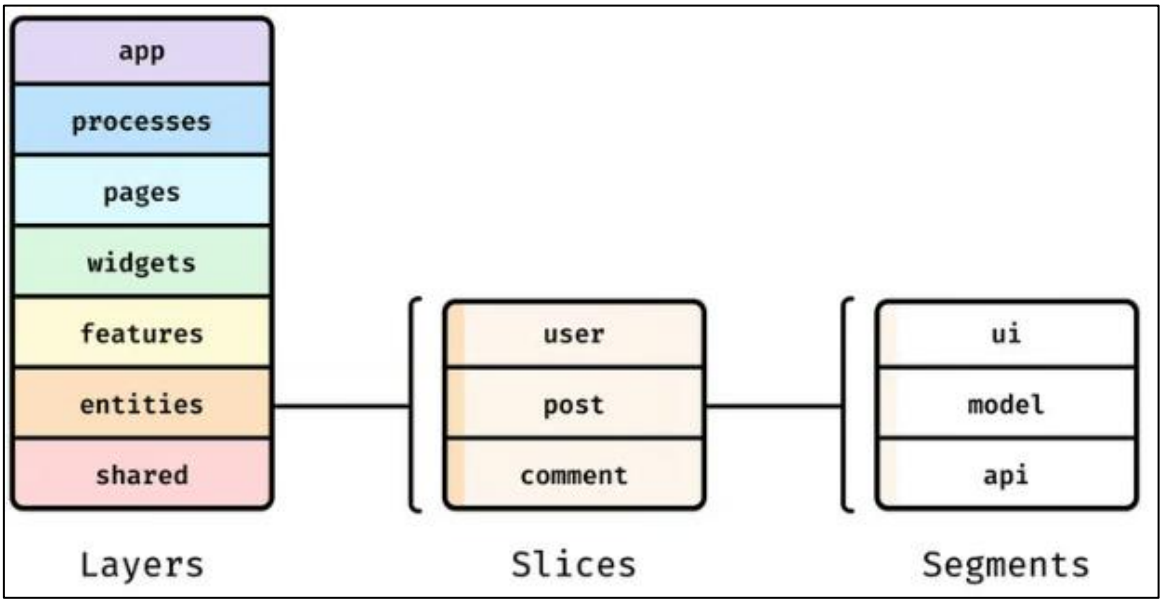

Рисунок 2.4 – Структура проекту згідно FSD

Шари - це каталоги верхнього рівня. Максимум може бути тільки 7 шарів. Деякі з них є необов'язковими. Виділяють наступні шари:

- app. Тут містяться налаштування, стилі, які застосовуються до всього додатку;
- processes. Цей шар обробляє складні сценарії міжсторінкової взаємодії. Він вже застарілий, але може використовуватися за потреби;
- pages. Цей шар є композиційним і дозволяє будувати сторінки з використанням елементів, функцій та віджетів;
- widgets. Композитний шар, який дозволяє групувати сутності і функції в змістовні блоки;
- features. Відповідає за взаємодію користувача та дії, які мають для користувача бізнес-значення;
- entities. Цей шар містить бізнес-сутності;
- shared. Містить функціональність, яка може бути повторно використана та не прив'язана до конкретної бізнес-логіки.

Ці шари допомагають організувати кодову базу та сприяють модульній, підтримуваній та масштабованій архітектурі.

Переходячи до секцій варто зазначити, що кожен шар має підкаталоги, які є розділами - другим рівнем декомпозиції додатку. У секціях зв'язок встановлюється з конкретними бізнес-сутностями. Основна мета секцій - групувати код за його значенням.

Назви секцій не мають стандартних назв через те, що вони безпосередньо визначаються бізнес-сферою проекту. У різних додатках назви секцій будуть різними та будуть залежати від конкретної специфіки та функціональності проекту.

Кожна секція складається з сегментів. Сегменти допомагають розділити код у секції відповідно до його призначення. Структура та назви сегментів можуть відрізнятися залежно від домовленостей команди. Найчастіше використовуються такі сегменти: api, UI, model, model, lib, config, consts.

При використанні фреймворку Next з архітектурою FSD можуть виникнути конфлікти.

Наприклад, Next використовує каталог "pages" для файлової навігації, так кожен каталог представляє маршрут. Але для FSD каталог "pages" - це шар з стандартизованою назвою. Один з способів вирішення цього конфлікту є розміщення каталогу "pages" для навігацію у кореневому каталогу проекту, а шар "pages" згідно FSD розмістити у вкладеному каталозі "src".

Перейдемо до класичної архітектури. Для неї не існує чіткого стандарту. Вона проста та добре підходить до невеликих проектів. Часто вона має каталоги для запитів, компонентів, сторінок, та функціоналу, що виконується в різних частинах додатку (рис.2.5).

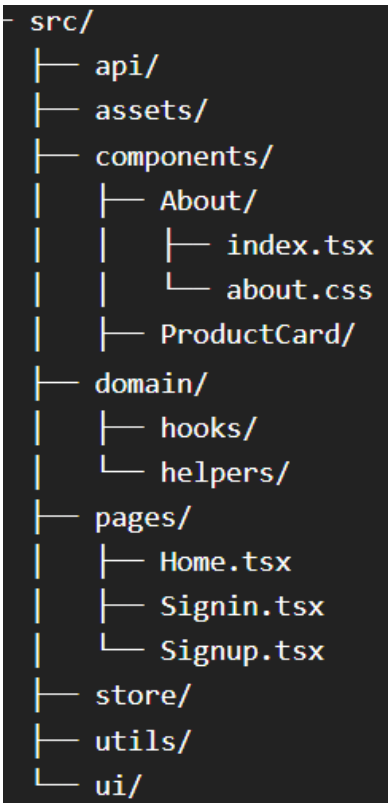

Рисунок 2.5 – Загальний вигляд класичної архітектури

Найбільша проблема такої архітектури полягає в тому, що неявні зв'язки між компонентами та переповнення модулів ускладнюють підтримку проектів. Чим довше проект розробляється, тим складнішою і важчою для розуміння стає архітектура додатку. Така архітектура підходить для невеликих проектів.

Як бачимо, фреймворки не забороняють використовувати різні архітектурні підходи, але це може призвести до певних обмежень або конфліктів, які потребують вирішення. Водночас, вона надає розробникам гнучкість, стандартизацію та можливість легко масштабувати додатки, що в свою чергу підвищує ефективність розробки.

2.2.4 Керування станами

Керування станом у динамічному світі веб-розробки є завжди актуальною темою, яка продовжує розвиватися протягом багатьох років. Зі збільшенням розміру додатку також збільшується й кількість станів, а тому розробникам потрібні ефективні рішення для управління ними.

Для розуміння стану розглянемо простий приклад із життя. Вимикач світла у кімнаті може приймати лише 2 стани: увімкнений та вимкнений. При цьому, від цього стану залежить чи буде увімкнене світло у кімнаті. Якщо вимикач має стан «увімкнений», то освітлення працює, тоді як вимкнений - припиняє подачу електроенергії до лампочок, що в результаті виключає освітлення. Ця проста аналогія допомагає зрозуміти що таке стан.

У веб-розробці концепція "стану" використовується для відстеження різних аспектів взаємодії користувача з веб-додатком. Наприклад, розглянемо форму для реєстрації на веб-сайті. Ця форма може мати різні стани, такі як "порожня", "заповнена", "надіслана" та інші.

Спочатку форма має стан "порожня". Після того, як користувач вводить свої дані, стан форми може змінитися на "заповнений". Після натискання на кнопку "Відправити" форма переходить в стан "надіслана", і користувач може бачити повідомлення про очікування результату.

Ці стани відображають різні етапи взаємодії користувача з веб-формою. В залежності від стану, користувачеві відображаються різні елементи інтерфейсу.

Тепер розглянемо які вбудовані рішення для керування станами надають веб-фреймворки на прикладі Next, та як це допомагає розробникам.

Для керування локальними станами у компоненті, Next надає можливість використовувати хук «useState». Для використання цього хука потрібно його імпортувати та викликати у верхньому рівні компоненту. Він повертає 2 елементи: стан та функцію для оновлення. Після оновлення стану, компонент, в якому знаходиться цей стан, буде перемальовано, що призведе до оновлення інтерфейсу. Стан може передаватися від «батьківського» компоненту до «дочірнього» як «пропс».

Частою помилкою при використанні цього хука є зміна стану напряму, без використання спеціальної функції оновлення. Це призводить до помилок та багів.

Локальні стани всередині компоненту можуть використовуватися для різних цілей. Найчастіше їх використовують для умовного відображення елементів на сторінці та для зв’язування даних з формою. Коли користувач вводить дані у поля форми, ці дані зберігаються і їх можна використовувати для подальшої обробки або відправки на сервер.

Також існують і глобальні стани. Головною відмінністю від локальних є те, що їх не потрібно передавати з компоненту в компонент, а доступ до них можна отримати з будь-якого компоненту додатку (рис. 2.6). Для цього існує інший хук – «useContext». Такий стан часто зберігають в окремому каталозі. А щоб отримати доступ до нього потрібно обгорнути весь додаток у провайдер та вказати шлях. В результаті ми маємо один стан та можемо отримати доступ до нього з будь-якого місця додатку.

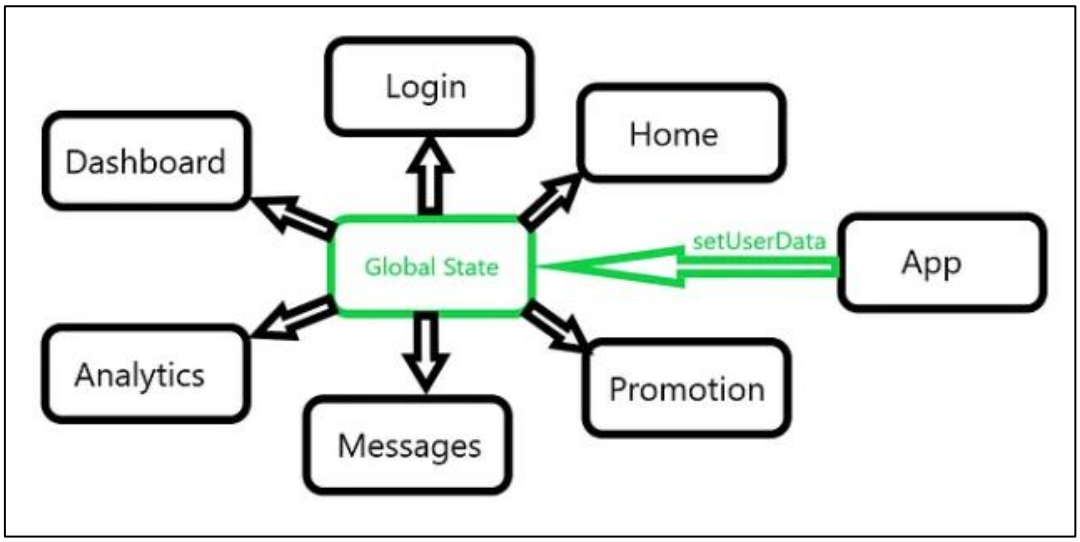

Рисунок 2.6 – Загальний схема глобального стану

Простим прикладом глобального стану може стати вибір мови у додатку. Веб-додатки часто підтримують декілька мов. При зміні мови ми хочемо змінити її у всьому додатку, а не лише у одній сторінці чи частині сайту.

Важливо пам’ятати, що існують й інші рішення для керування станами – сторонні бібліотеки. Вони не залежні від конкретних фреймворків і можуть бути використані в різних проектах. Однак варто пам'ятати, що їх використання також може внести додаткову складність у проект, особливо якщо вони вимагають вивчення нових концепцій та підходів. Тому варто ретельно оцінити її відповідність конкретним потребам проекту та зважити на переваги та недоліки її використання.

Хоча існують численні фреймворки та бібліотеки, які пропонують структуровані підходи до керування станом, розробники також можуть вручну керувати станом, використовуючи лише базові можливості JavaScript. Головним недоліком ручного керування станами є витрата ресурсів на його створення, складність в реалізації та підтримці порівняно з підходами що надають фреймворками.

2.2.5 Оптимізація

Оптимізація у контексті веб-розробки – це процес вдосконалення веб-сайту або веб-сторінки з метою забезпечення кращої видимості у пошукових системах, поліпшення швидкості завантаження та підвищення зручності користувачів. Вона включає в себе ряд методів, щоб зробити його більш ефективним і відповідним потребам користувачів.

Продуктивність веб-сайту є критично важливою з кількох причин. По-перше, вона впливає на користувацький досвід. Повільне завантаження сторінок і додатків може розчарувати користувачів і змусити їх взагалі відмовитися від продукту. Крім того, продуктивність веб-ресурсу є важливим фактором видимості для пошукових систем. Google, зокрема, заявив, що швидкість роботи веб-сайту впливає на всі алгоритми продукту. Коротший час завантаження призводить до більшої кількості відвідувачів веб-сайту і вищого доходу.

Зазвичай продуктивність вимірюється за тим, як швидко веб-додаток може завантажувати контент і як швидко він може реагувати на дії користувача.

Найкращий спосіб відстежувати продуктивність веб-додатку - це моніторинг системи, запис та аналіз показників продуктивності шляхом спостереження за певними функціями. Добре структурований інструмент моніторингу продуктивності допомагає виявити і вирішити відповідні проблеми до того, як вони виникнуть, в результаті чого веб-сайт буде оптимізованим, ефективним і проактивним.

На прикладі веб-фреймворку Next.js розглянемо які методи оптимізації та покращення ефективності надають фреймворки.

Однієї з головних проблем при роботі з DOM є повторний рендеринг, адже цей процес дуже повільний. Зміна та оновлення DOM відбуваються досить швидко, але після змін оновлений елемент та всі його дочірні елементи повторно відображаються, щоб оновити інтерфейс. Отже, чим більше у компонентів інтерфейсу, тим більше навантаження з точки зору продуктивності мають оновлення DOM.

Для прикладу, маємо список з 10 елементів. Змінюємо перший елемент. Часто фреймворки перебудують весь список. Це в 10 разів більше роботи, ніж потрібно. Змінився лише 1 елемент, інші 9 залишаються незмінними. Тому неефективне оновлення часто стає серйозною проблемою. Для вирішення цієї проблеми Next використовує Virtual DOM [9].

Віртуальний DOM (Virtual DOM) - це концепція програмування, де ідеальне або "віртуальне" представлення інтерфейсу користувача зберігається в пам'яті і синхронізується з "реальним" DOM, використовуючи бібліотеки, такі як ReactDOM. Цей процес називається узгодженням (reconciliation). Маніпуляції з реальним DOM є повільними. Маніпуляції з віртуальним DOM значно швидше, оскільки вони не відображаються на екрані.

Розглянемо як працює віртуальний DOM (рис. 2.7):

1. при відмалюванні сторінки створюється точна копія реального DOM – віртуальний DOM;
2. при оновленні елементу створюється оновлена версія віртуального DOM;
3. фреймворк порівнює оновлений віртуальний DOM з попередньою версією віртуального DOM та знаходить елементи, що були змінені;
4. фреймворк оновлює реальний DOM на основі змінених у віртуальному DOM елементів;
5. відбувається оновлення веб-сторінки.

Замість того, щоб рендерити всі зміни в реальний DOM, ми спочатку застосовуємо їх до віртуального DOM, який не рендериться. Тому зміни до нього є дуже дешевими.

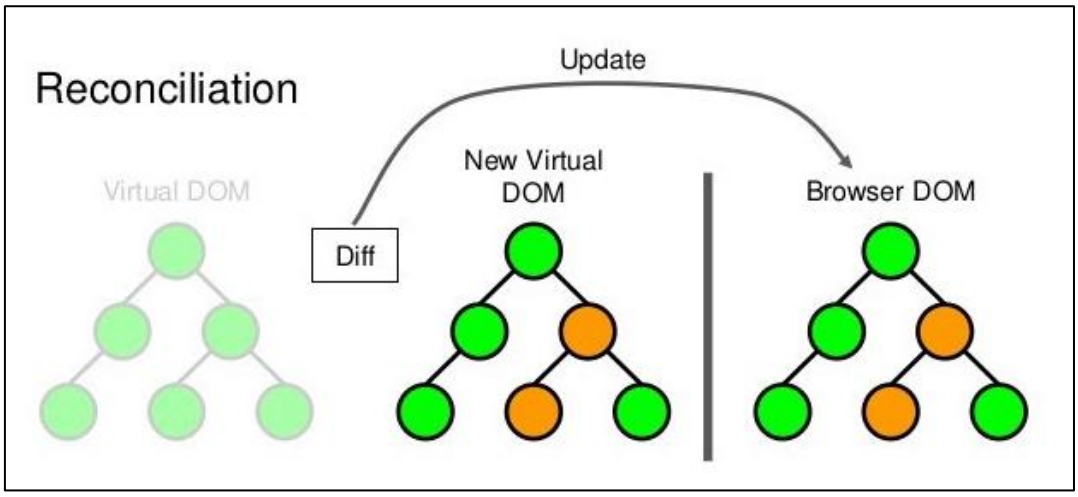

Рисунок 2.7 – Схема роботи віртуального DOM

Віртуальний DOM є ефективним механізмом, який допомагає зменшити кількість непотрібних оновлень і швидко виконувати рендеринг, підвищуючи продуктивність.

Частою особливістю роботи зі списками є передача ключів. Для того, щоб фреймворк чітко міг розуміти та відрізняти один елемент списку від іншого, при відображенні потрібно передавати ключ. Ключ повинен бути унікальним всередині списку. Якщо не передати ключ, то при видаленні, зміні порядку елементів у списку чи іншими маніпуляціями можуть виникнути непередбачувані помилки та баги.

Розглянемо інший тип оптимізації – SEO (Search Engine Optimization) [10]. Зі зростаючою кількістю контенту, що публікується щодня, для того щоб виділятися серед інших допомагають SEO-стратегії. SEO підвищує видимість веб-сайту в пошукових системах, таких як Google, що призводить до збільшення трафіку та потенційних конверсій.

Одним із способів для покращення SEO – використання семантичних тегів при створенні розмітки сторінки. Такий спосіб підтримується браузерами та не вимагає використання сторонніх інструментів. Семантична розмітка надає кращі інструкції для пошукових роботів при скануванні сторінок за їхнім вмістом та повідомляє їм, який вміст знаходиться на сторінці.

Для прикладу, семантичний тег <h1> надає більше інформації про вміст сторінки, ніж тег <p>, адже <h1> вказує на головний заголовок сторінки, який

зазвичай містить ключову інформацію про тему всього документа. Це дозволяє пошуковим роботам розуміти, що інформація, яка знаходиться в цьому тегу, є найбільш важливою та релевантною для запитів користувачів. Таким чином, семантична розмітка покращує видимість сайту в пошукових системах, що є важливою частиною успішної SEO-стратегії.

Розглянемо як фреймворки можуть допомогти покращити SEO. Наприклад, фреймворки Next та Nuxt підтримують SSR (Server Side Rendering).

SSR (серверний рендеринг) відноситься до процесу, коли сервер генерує HTML-сторінку на сервері та надсилає його клієнту. Цей процес відрізняється від рендерингу на стороні клієнта, де JavaScript використовується для рендерингу сторінку на боці клієнта.

Однією з головних переваг серверного рендерингу для SEO є те, що він дозволяє пошуковим системам краще індексувати та розуміти вміст веб-сторінки. Коли робот пошукової системи відвідує веб-сторінку, яка рендериться на стороні клієнта, він може не зуміти повністю обробити вміст сторінки, оскільки він залежить від JavaScript. Це може призвести до неповної або неправильної індексації сторінки, що може негативно вплинути на її пошукові рейтинги.

Серверний рендеринг вирішує цю проблему, надаючи повністю знегерований HTML-код при першому завантаженні сторінки. Це надає пошуковим роботам можливість бачити весь вміст сторінки одразу, без необхідності виконувати JavaScript. В результаті, сторінка краще індексується та знаходиться вище у пошукових результатах, оскільки пошукові системи можуть коректно визначити її зміст та релевантність для користувацьких запитів.

Наступною перевагою SSR для покращення SEO є швидкість завантаження сторінки. Сервер відправляє запити для отримання даних, стилів та вмісту сторінки, генерує та відправляє клієнту вже згенеровану HTML-сторінку. Такий підхід дозволяє швидше завантажити сторінку, щоб користувач зміг почати взаємодіяти з нею якнайшвидше.

Крім того, SSR дозволяє мати динамічний вміст на багатьох сторінках веб-додатку. Це означає, що вміст сторінки може змінюватися залежно від певних умов,

таких як дії користувача, поточний час, геолокація або інші параметри. Такі можливості створюють більш персоналізований і релевантний досвід для користувачів.

Як бачимо, існує безліч стратегій для покращення SEO. Однією з головних, які забезпечують веб-фреймворки є Server Side Rendering. Такий підхід з генерацією сторінки на стороні серверу має безліч переваг. Перш за все, він забезпечує швидше завантаження сторінок, що покращує користувацький досвід. Крім того, сторінки, згенеровані на сервері, легше індексуються пошуковими системами, оскільки вміст вже готовий і не потребує виконання JavaScript для його відображення.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи магістра було виконано дослідження ефективності розробки веб-додатків з використанням фронтенд фреймворків, проведено аналіз, визначені переваги та недоліки популярних фронтенд фреймворків.

Також у ході виконання роботи були вирішені наступні завдання:

- проведено аналіз веб-сайтів, визначена їх структура та типи;
- визначені головні етапи розробки веб-додатків;
- розглянута проблематика вибору фреймворку;
- проаналізовані особливості, переваги та недоліки популярних фронтенд фреймворків;
- проведено аналіз можливостей, що надають фреймворки та їх вплив на розробку.

Було досліджено питання доцільності розробки веб-додатків з використанням фронтенд фреймворків. В результаті було зроблено висновок, що залежно від типу проекту, його розміру та призначення, є правильним рішенням використовувати фронтенд фреймворки, адже вони підвищують ефективність та спрощують процес розробки веб-додатків. Розробка веб-додатків також можлива й без їх використання, але в такому випадку, ефективність розробки може бути нижчою.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web (дата звернення: 15.04.2024).

2. Голян В.В, Міллер П.П, "Веб-сервіс для контенту підписок із соціальних мереж" / Сб. научн. тр. По материалам 22-го Международного молодежного форума «Радиоэлектроника и молодежь в XXI веке», 2019. – с.117- 118.

3. Website Development Planning Process. URL: https://dynomapper.com/blog/487-website-development-planning-process (дата звернення: 02.05.2024).

4. Why use Front-end Frameworks. URL: https://belitsoft.com/what-do-front-end-frameworks-do (дата звернення: 01.05.2024).

5. Web frameworks – an overview. URL: https://www.ionos.com/digitalguide/websites/web-development/web-frameworks-an-overview (дата звернення: 04.05.2024).

6. Command-line interface. URL: https://en.wikipedia.org/wiki/Command-line_interface (дата звернення: 10.05.2024).

7. Writing Markup with JSX. URL: https://react.dev/learn/writing-markup-with-jsx (дата звернення: 05.05.2024).

8. Feature-Sliced Design: The Best Frontend Architecture. URL: https://dev.to/m_midas/feature-sliced-design-the-best-frontend-architecture-4noj (дата звернення: 10.05.2024).

9. React’s Virtual DOM. URL: https://medium.com/@BharathkumarV/reacts-virtual-dom-17fdcb290a10 (дата звернення: 26.01.2023).

10. Search engine optimization. URL: https://en.wikipedia.org/wiki/Search_engine_optimization (дата звернення: 18.05.2024).