Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
(повне найменування вищого навчального закладу)
Факультет комп'ютерно-інформаційних систем і програмної інженерії
(назва факультету)
Інженерія програмного забезпечення
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

Магістр
(освітній ступінь (освітньо-кваліфікаційний рівень))

на тему: Розробка реактивного фронтенд фреймворку для односторінкових додатків з власною системою реактивності

Виконав(ла): студент(ка) 6 курсу, групи СПм-62
спеціальності 121 Програмна інженерія
(шифр і назва спеціальності)

(підпис) (прізвище та ініціали)

Керівник (підпис) Цуприк Г.Б. (прізвище та ініціали)

Нормоконтроль (підпис) (прізвище та ініціали)

Завідувач кафедри (підпис) (прізвище та ініціали)

Рецензент (підпис) Литвиненко Я.В. (прізвище та ініціали)

Тернопіль
2023

АНОТАЦІЯ

Об’єкт дослідження: процес реалізації фреймворка для односторінкових веб додатків

Мета роботи: створення нового фреймворка для односторінкових веб-додатків з новою системою реактивності

Методи дослідження: методи порівняння, аналізу, синтезу, дедукції.

У роботі досліджено види мов програмування та фреймворків для створення реактивних односторінкових веб-додатків. Проаналізовано базові методи створення фреймворків. Запропонована реалізація фреймворку з унікальною системою реактивності.

Практичне значення роботи полягає у оптимізації роботи при створенні будь-яких веб-аплікацій.

Результати проведеного дослідження у цій роботі можуть бути використанні для створення власного веб-застосунку.

Наукова новизна полягає в тому, що вперше було використанно технології Proxy/Reflect у комбінації з JSX та функційними компонентами.Ключові слова: Proxy/Reflect, Javascript, SOLID.

ABSTRACT

Research Object: Implementation process of a framework for single-page web applications

Research Objective: Creating a new framework for single-page web applications with a novel reactivity system

Research Methods: Comparative, analytical, synthetic, deductive methods

The study explores programming languages and frameworks for creating reactive single-page web applications. Basic methods of framework creation are analyzed, and an implementation of a framework with a unique reactivity system is proposed.

The practical significance of the work lies in optimizing the process of creating any web applications. The research results presented in this work can be utilized for developing custom web applications.

The scientific novelty is in the first-time use of Proxy/Reflect technologies in combination with JSX and functional components. Keywords: Proxy/Reflect, Javascript, SOLID.

ЗМІСТ

ВСТУП

Актуальність дослідження. Станом на 2023 рік діяльність організацій все більше залежать від комп’ютерних технологій, тому проблеми вибору технологій для реалізації застосунків збільшуються. Під час та після пандемії COVID-19 кількість підприємств, що перевели свої системи в хмарні рішення, зросла в рази, а це демонструє зростаючий попит на швидкорозвиваючихся технології. Особливо важливим є створення таких рішень, які дозволяють швидко та ефективно розгортати сучасні веб-додатки в умовах стрімкого темпу цифрової трансформації.

На даний момент на ринку присутні різні рішення для даної проблеми, написані на різних мовах програмування. Однак, слід врахувати, що браузери інтерпретують лише Javascript, що робить його практично стандартом для веб-розробки. Обираючи фреймворк, написаний саме на цій мові, ми максимізуємо сумісність та легкість інтеграції, що сприяє оперативному створенню рішень.

Найпопулярніші фреймворки, такі як React, Vue, та Angular, базуються на старій системі реактивності, що використовує принципи get/set. У моєму фреймворку я пропоную новий підхід до реактивності, який має потенціал пришвидшити виконання коду.

У результаті, розробка реактивного фронтенд фреймворку з наявністю всіх типових рішень і власною системою реактивності, може бути потужним інструментом для розробки різноманітних сервісів, необхідних для бізнесу. Гнучкість та універсальність даного фреймворку дозволить розробникам творчо підходити до створення різноманітних веб-додатків.

Метою магістерської роботи (проекту) є створення фреймворку для односторінкових веб-додатків з унікальною системою реактивності

Завдання дослідження. Для досягнення зазначеної мети дипломної роботи поставлені окремі завдання:

– дослідити види мов програмування та фреймворків які присутні на ринку;

– проаналізувати плюси і мінуси фреймворків на основі Javascript;

– дослідити методи та практики методи та найкращі практики по створенню фреймворків

Об’єктом дослідження дипломної роботи є робота з реактивним фреймворком на основі javascript

Предметом дослідження є створення фреймворка на основі proxy/reflect для реактивності

Методи дослідження. В даній роботі використано системного підходу, дедукції, методи порівняння, аналізу і синтезу;

Наукова новизна роботи. Полягає в тому, що вперше було використано технології Proxy/Reflect у комбінації з JSX та функційними компонентами. Ключові слова: Proxy/Reflect, Javascript, SOLID.

1 ПРИНЦИПИ СТВОРЕННЯ ОДНОСТОРІНКОВИХ ДОДАТКІВ

В сучасному світі веб-додатки стають все складнішими, а вимоги до їхнього інтерфейсу неухильно зростають. У цьому контексті фронтенд-фреймворки для односторінкових додатків (SPA) стали невід'ємною частиною розробки.

З розвитком технологій з'явилася потреба в ефективному управлінні структурою веб-додатків та взаємодією компонентів на стороні клієнта. Фронтенд-фреймворки визначають новий стандарт, надаючи розробникам інструменти для створення SPA, що вирізняються швидкістю, ефективністю та масштабованістю.

Разом із зростанням складності веб-додатків, відбувається перехід від традиційного мультисторінкового підходу до SPA. Цей перехід створює потребу у фреймворках, які сприяють розробці високопродуктивних, динамічних та легко масштабованих SPA.

Фреймворки для SPA стають невід'ємною складовою сучасної веб-розробки, надаючи розробникам не лише зручний інтерфейс для організації коду, але й потужні засоби для управління станом додатків та взаємодії з сервером.

У світлі стрімкого розвитку веб-технологій фронтенд-фреймворки для SPA визначають новий етап в розробці веб-додатків. Їхня роль полягає в полегшенні та удосконаленні процесу створення SPA, роблячи їх більш швидкими, ефективними та пристосованими до сучасних вимог користувачів.

1.1 Основні принципи односторінкових додатків

Існують вісім основних принципів односторінкових веб додатків

Одна сторінка, багато можливостей:

Основна ідея SPA полягає в тому, що весь інтерфейс додатка завантажується однією HTML-сторінкою.

Всі подальші взаємодії та навігація здійснюються за допомогою асинхронних запитів на сервер і динамічної зміни вмісту без повторного завантаження сторінки.

Реактивність:

SPA реагує на дії користувача миттєво, оновлюючи лише ті частини сторінки, які потребують змін. Використання асинхронних запитів та оновлення стану дозволяє створювати більш динамічний та зручний інтерфейс.

Маршрутизація на клієнтському боці:

SPA використовує маршрутизацію на клієнтському боці для управління станом додатка та забезпечення навігації між різними частинами додатка без перезавантаження сторінки.

Спеціалізований API:

Використання спеціалізованого API (наприклад, REST або GraphQL) для взаємодії з сервером, зазвичай за допомогою асинхронних запитів, сприяє швидкій передачі даних між клієнтом та сервером.

Кешування та управління станом:

SPA активно використовує кешування та управління станом для забезпечення ефективності та швидкодії. Це дозволяє уникати повторних завантажень даних та забезпечує збереження стану додатка.

Асинхронне завантаження ресурсів:

Завантаження ресурсів, таких як зображення, стилі та скрипти, зазвичай відбувається асинхронно, що дозволяє прискорити завантаження сторінок та зменшити час очікування користувача.

Використання шаблонів та компонентів:

SPA часто базується на використанні шаблонів та компонентів для підтримки повторного використання коду та полегшення розробки.

Зосередженість на користувацькій взаємодії:

SPA акцентується на покращенні взаємодії з користувачем, забезпечуючи йому плавний та безшовний досвід використання додатка без перерв.

1.1.1 Основні напрямки використання односторінкових додатків

SPA надають багатофункціональні та динамічні інтерфейси, що дозволяє їм успішно використовуватися у різноманітних сферах для оптимізації користувацького досвіду та підвищення продуктивності:

1. Електронна комерція: SPA ідеально підходять для інтернет-магазинів та електронної комерції. Швидка навігація, миттєве оновлення кошика та ефективне

відображення товарів сприяють покращенню користувацького досвіду та підвищенню конверсії.

2. Соціальні мережі: У соціальних мережах важливий аспект - динаміка та миттєвість. SPA дозволяють користувачам плавно переміщатися між різними вкладками та отримувати оновлення без повторного завантаження сторінки.

3. Освіта та навчання: Платформи для навчання та електронні навчальні системи використовують SPA для створення інтерактивних та швидких інтерфейсів. Зручна навігація та негайна відповідь на дії користувача поліпшують процес навчання.

4. Аналітика та панелі управління: Додатки для аналізу даних та панелі управління часто використовують SPA для забезпечення швидкого та ефективного моніторингу та аналізу великих обсягів інформації.

5. Банківська та фінансова сфера: У фінансовому секторі, де важлива швидкість та безпека, SPA забезпечують можливість миттєвого доступу до фінансової інформації та здійснення операцій без затримок.

6. Телекомунікації: Односторінкові додатки використовуються для створення особистих кабінетів та платформ для управління послугами телекомунікаційних компаній. Це дозволяє користувачам легко керувати своїм акаунтом та послугами.

7. Транспорт та логістика: Додатки для служб таксі, доставки та логістики використовують SPA для забезпечення швидкого та ефективного взаємодії з користувачами, відстеження та оптимізації маршрутів.

8. Технічна підтримка та сервіси: Платформи для технічної підтримки та сервісів надають користувачам можливість отримати доступ до необхідної інформації та рішень швидко та без зайвих зусиль.

1.1.4 Заходи захисту інформації

Одна з основних проблем безпеки SPA полягає в тому, що вони часто використовують Ajax для обміну даними з сервером. Ajax дозволяє SPA синхронізувати дані без перезавантаження сторінки, що покращує досвід користувача, але також може зробити додатки більш вразливими до атак.

Атаки на Ajax-запити можуть використовуватися для крадіжки даних, таких як паролі або кредитні картки. Вони також можуть використовуватися для виконання шкідливого коду на клієнтській машині.

Щоб захистити SPA від атак, необхідно вжити ряд заходів безпеки. До таких заходів належать:

1. Криптування даних. Всі дані, які передаються між клієнтом і сервером, повинні бути зашифровані. Це допоможе захистити дані від перехоплення.
2. Використання HTTPS. HTTPS - це протокол безпеки, який шифрує весь трафік між клієнтом і сервером. Він забезпечує додатковий рівень безпеки для SPA.
3. Використання фільтрів на сервері. Сервер може використовувати фільтри для блокування потенційно шкідливих запитів. Наприклад, сервер може блокувати Ajax-запити, які містять певні сценарії.
4. Використання бібліотек безпеки. Існує ряд бібліотек безпеки, які можна використовувати для захисту SPA від атак. Ці бібліотеки можуть надавати такі функції, як криптування, фільтрація запитів і захист від шкідливого коду.

1.1.5. Рекомендації по захисту інформації в односторінкових додатках

1. Виконуйте ретельне тестування додатку. Тестування додатку на наявність уразливостей безпеки допоможе виявити потенційні проблеми до того, як їх зможуть використовувати зловмисники.
2. Стежте за додатком на наявність загроз. Існує ряд інструментів, які можна використовувати для моніторингу додатку на наявність загроз безпеки. Це допоможе виявити потенційні атаки на ранніх етапах.
3. Регулярно оновлюйте додаток. Розробники постійно випускають нові оновлення для SPA, які включають в себе виправлення безпеки. Регулярне оновлення додатка допоможе захистити його від останніх загроз безпеки.

Виконання цих рекомендацій допоможе захистити односторінкові додатки від атак і забезпечити безпеку даних користувачів.

1.2 Основні типи веб-додатків

1.2.1 Поняття веб-додатка

Веб-додатки - це програмне забезпечення, яке працює в веб-браузері. Вони можуть бути призначені для різних цілей, таких як надання інформації, виконання завдань або забезпечення взаємодії з користувачами.

1.2.2 Типи веб-додатків.

- Сторінка або клієнт-сервер. Сторінкові веб-додатки обробляються повністю на клієнтській стороні, тоді як клієнт-серверні веб-додатки обробляються як на клієнтській, так і на серверній сторонах.
- Односторінкові або багатосторінкові. Односторінкові веб-додатки (SPA) оновлюють вміст сторінки без її перезавантаження, тоді як багатосторінкові веб-додатки (MPA) перезавантажують сторінку при кожному зміні вмісту.
- Статичні або динамічні. Статичні веб-додатки містять лише статичний вміст, тоді як динамічні веб-додатки можуть генерувати вміст на основі даних, які отримуються з бази даних або інших джерел.

1.2.3 Мови програмування для створення веб-додатків

Для розробки веб-додатків можна використовувати широкий спектр мов програмування. Наприклад:

- HTML, CSS і JavaScript. Ці три мови є базовими для розробки веб-сторінок і веб-додатків.
- PHP. Мова програмування загального призначення, яка часто використовується для розробки веб-додатків.
- Java. Мова програмування загального призначення, яка також може використовуватися для розробки веб-додатків.
- Python. Мова програмування загального призначення, яка є популярною для розробки веб-додатків, особливо SPA.
- C#. Мова програмування, розроблена компанією Microsoft, яка може використовуватися для розробки веб-додатків.

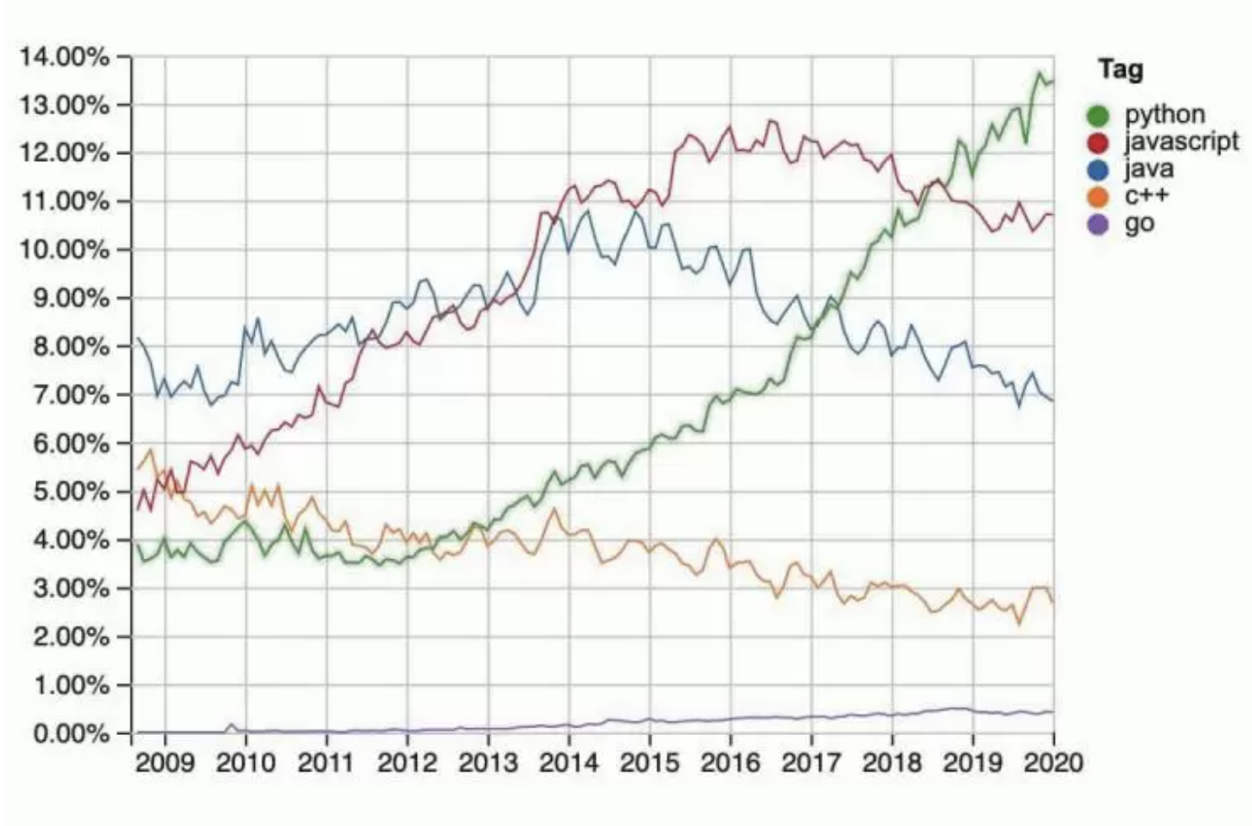

Рисунок 1.2.3.1 Графік популярності мов програмування для веб-додатків

1.2.2 Види фреймворків для створення веб-додатків

Кожна мова програмування по своєму унікальна і має в свому озброєнні декілька фреймворків для створення веб-додатків, як односторінкових так і багато-сторінкових. Розглянемо найпопулярніші з них та проаналізуємо.

1.2.2.1 C# та ASP.NET Core

ASP.NET Core - це сучасний фреймворк для веб-додатків, розроблений компанією Microsoft. Він підтримує широкий спектр функцій, включаючи реактивне програмування, міцну типізацію та ефективність.

Плюси ASP.NET Core:

- Реактивність. ASP.NET Core підтримує реактивне програмування, що робить його хорошим вибором для розробки SPA. Реактивне програмування дозволяє веб-додаткам оновлювати свій вміст без необхідності перезавантажувати сторінку.
- Міцна типізація. ASP.NET Core підтримує міцну типізацію, що допомагає запобігти помилкам і покращити читабельність коду.
- Ефективність. ASP.NET Core є ефективною мовою, яка може забезпечити високу продуктивність веб-додатків.
- Портованість. ASP.NET Core є портативним фреймворком, який можна запускати на різних платформах, включаючи Windows, macOS і Linux.
- Велика спільнота. ASP.NET Core має велику і активну спільноту розробників, яка може надати допомогу та підтримку.

Мінуси ASP.NET Core:

- Складність. ASP.NET Core може бути складним фреймворком для вивчення, особливо для початківців.

- Залежність від Microsoft. ASP.NET Core є фреймворком, розробленим компанією Microsoft, що може обмежувати його використання в деяких середовищах.

Реактивність в ASP.NET Core:

ASP.NET Core підтримує реактивне програмування на основі Rx.NET. Rx.NET - це бібліотека, яка дозволяє розробникам створювати реактивні програми, які реагують на зміни в даних.

Реактивність в ASP.NET Core працює за допомогою об'єктів Observable. Observable - це об'єкт, який генерує послідовність даних. Розробники можуть використовувати Observable для створення веб-додатків, які оновлюють свій вміст без необхідності перезавантажувати сторінку.

1.2.2.2 PHP та Laravel

Laravel - це фреймворк для веб-додатків на основі PHP, який є популярним вибором для розробки SPA. Він пропонує широкий спектр функцій і можливостей, які роблять його хорошим вибором для розробки різних типів веб-додатків.

Плюси Laravel:

Реактивність. Laravel підтримує реактивне програмування, що робить його хорошим вибором для розробки SPA. Реактивне програмування дозволяє веб-додаткам оновлювати свій вміст без необхідності перезавантажувати сторінку.

Легкість вивчення. Laravel є відносно легким фреймворком для вивчення, що робить його хорошим вибором для початківців.

Велика спільнота. Laravel має велику і активну спільноту розробників, яка може надати допомогу та підтримку.

Модульність. Laravel є модульним фреймворком, що дозволяє розробникам легко налаштовувати і розширювати його.

Безпека. Laravel має вбудовані функції безпеки, які допомагають захистити веб-додатки від атак.

Мінуси Laravel:

Складність. Laravel може бути складним фреймворком для вивчення, особливо для початківців.

Залежність від PHP. Laravel є фреймворком, розробленим на основі PHP, що може обмежувати його використання в деяких середовищах.

Реактивність в Laravel:

Laravel підтримує реактивне програмування за допомогою бібліотеки Illuminate/Support/Facades/Broadcast. Broadcast - це бібліотека, яка дозволяє розробникам створювати реактивні програми, які реагують на зміни в даних.

Реактивність в Laravel працює за допомогою об'єктів Channel. Channel - це об'єкт, який генерує послідовність даних. Розробники можуть використовувати Channel для створення веб-додатків, які оновлюють свій вміст без необхідності перезавантажувати сторінку.

1.2.2.4 Spring Boot та Java

Spring Boot - це фреймворк для веб-додатків на основі Java, який є популярним вибором для розробки SPA. Він пропонує широкий спектр функцій і можливостей, які роблять його хорошим вибором для розробки різних типів веб-додатків.

Плюси Spring Boot:

Реактивність. Spring Boot підтримує реактивне програмування, що робить його хорошим вибором для розробки SPA. Реактивне програмування дозволяє веб-додаткам оновлювати свій вміст без необхідності перезавантажувати сторінку.

Ефективність. Spring Boot є ефективним фреймворком, який може забезпечити високу продуктивність веб-додатків.

Легкість вивчення. Spring Boot є відносно легким фреймворком для вивчення, що робить його хорошим вибором для початківців.

Велика спільнота. Spring Boot має велику і активну спільноту розробників, яка може надати допомогу та підтримку.

Модульність. Spring Boot є модульним фреймворком, що дозволяє розробникам легко налаштовувати і розширювати його.

Безпека. Spring Boot має вбудовані функції безпеки, які допомагають захистити веб-додатки від атак.

Мінуси Spring Boot:

Складність. Spring Boot може бути складним фреймворком для вивчення, особливо для початківців.

Залежність від Java. Spring Boot є фреймворком, розробленим на основі Java, що може обмежувати його використання в деяких середовищах.

Реактивність в Spring Boot:

Spring Boot підтримує реактивне програмування за допомогою бібліотеки Spring WebFlux. WebFlux - це бібліотека, яка дозволяє розробникам створювати реактивні веб-додатки.

Реактивність в Spring Boot працює за допомогою об'єктів Mono і Flux. Mono - це об'єкт, який генерує одну одиницю даних, а Flux - це об'єкт, який генерує послідовність даних. Розробники можуть використовувати Mono і Flux для створення веб-додатків, які оновлюють свій вміст без необхідності перезавантажувати сторінку.

1.2.2.5 Spring Boot та Java

Django - це популярний фреймворк для веб-додатків на основі Python. Він підтримує широкий спектр функцій, включаючи реактивне програмування, міцну типізацію та ефективність.

Плюси Django:

Реактивність. Django підтримує реактивне програмування, що робить його хорошим вибором для розробки SPA. Реактивне програмування дозволяє веб-додаткам оновлювати свій вміст без необхідності перезавантажувати сторінку.

Легкість вивчення. Django є відносно легким фреймворком для вивчення, що робить його хорошим вибором для початківців.

Велика спільнота. Django має велику і активну спільноту розробників, яка може надати допомогу та підтримку.

Модульність. Django є модульним фреймворком, що дозволяє розробникам легко налаштовувати і розширювати його.

Безпека. Django має вбудовані функції безпеки, які допомагають захистити веб-додатки від атак.

Мінуси Django:

Складність. Django може бути складним фреймворком для вивчення, особливо для початківців.

Залежність від Python. Django є фреймворком, розробленим на основі Python, що може обмежувати його використання в деяких середовищах.

1.3 Створення фреймворків

1.3.3 Етапи створення фреймворків

У цьому пункті, ми розглянемо етапи створення фреймворку, незалежно від мови програмування.

1.3.3.1 Перший етап: Планування

Планування є важливим етапом у розробці будь-якого програмного забезпечення, включаючи фреймворки для односторінкових веб-додатків. На цьому етапі ви повинні визначити такі аспекти фреймворка:1. Визначення цілей і призначення фреймворка

1.3.3.1.1 Визначення цілей і призначення фреймворка

На цьому етапі ми повинні визначити, що ми хочемо від нашого фреймворку та відповісти на такі питання:

- Які функції повинні бути у фреймворка?
- Для кого призначений фреймворк?
- Які проблеми він повинен вирішувати?

1.3.3.1.2 Визначення цільової аудиторії

На цьому етапі ми повинні визначити, для кого призначений наш фреймворк. Ми повинні відповісти на такі питання:

- Які навички і досвід повинні мати розробники, які хочуть використовувати фреймворк?
- Які типи додатків розробники будуть створювати за допомогою фреймворка?

1.3.3.1.3 Визначення функціональних вимог

На цьому етапі ви повинні визначити, які функції повинен мати ваш фреймворк, щоб він відповідав його цілям і призначенню. Ви повинні відповісти на такі питання:

- Які компоненти повинні бути у фреймворка?
- Які методи повинні бути у компонентів?
- Які події повинен обробляти фреймворк?

1.3.3.1.4 Визначення нефункціональних вимог

На цьому етапі ми повинні визначити, які нефункціональні вимоги повинен мати наш фреймворк, такі як продуктивність, масштабованість і безпека. Ми повинні відповісти на такі питання:

- Як швидко повинен працювати фреймворк?
- Який розмір повинен мати фреймворк?
- Який рівень безпеки повинен забезпечувати фреймворк?

1.3.3.1.4 Розробка плану реалізації

На цьому етапі ми повинні розробити план реалізації фреймворка. План повинен включати в себе такі елементи:

- Розклад: Коли ви плануєте завершити різні етапи розробки фреймворка?
- Бюджет: Скільки коштів вам знадобиться для розробки фреймворка?
- Ресурси: Які ресурси вам знадобляться для розробки фреймворка, наприклад, люди, обладнання та програмне забезпечення?
- Успішні практики

1.3.3.2 Другий етап: Проектування

Проектування є наступним етапом після планування. На цьому етапі ми повинні розробити архітектуру вашого фреймворка і визначити, як його компоненти будуть взаємодіяти один з одним.

1.3.3.2.1 Архітектура

Архітектура фреймворка визначає, як він буде структурований і як його компоненти будуть взаємодіяти один з одним. Існує багато різних архітектур, які можна використовувати для фреймворків для односторінкових веб-додатків.

Одним з популярних варіантів архітектури є Model-View-Controller (MVC). Ця архітектура розділяє фреймворк на три компоненти:

- Model: Відповідає за зберігання даних
- View: Відповідає за рендеринг інтерфейсу користувача
- Controller: Відповідає за обробку подій і управління потоком виконання

Іншим популярним варіантом є архітектура Model-View-ViewModel (MVVM). Ця архітектура схожа на MVC, але View і Model більш тісно пов'язані між собою.

1.3.3.2.2 Компоненти

Компоненти є основними будівельними блоками фреймворка. Вони відповідають за виконання певних завдань, таких як рендеринг інтерфейсу користувача, обробка подій або доступ до даних. При проектуванні компонентів ви повинні враховувати такі фактори:

- Функціональність: Які функції повинен виконувати компонент?
- Інтерфейс: Який інтерфейс повинен мати компонент?
- Залежність: Від яких інших компонентів залежить компонент?

1.3.3.2.3 Взаємодія компонентів

Компоненти повинні взаємодіяти один з одним, щоб створювати повнофункціональні веб-додатки. Ви повинні визначити, як компоненти будуть взаємодіяти один з одним, за допомогою таких механізмів, як:

- Події: Компоненти можуть посилати події один одному.
- Потоки: Компоненти можуть обмінюватися даними через потоки.
- Інтерфейси: Компоненти можуть взаємодіяти один з одним через інтерфейси.

1.3.3.3 Третій етап: Реалізація

Реалізація - це етап, на якому ви фактично створюєте свій фреймворк. На цьому етапі ви повинні втілити в життя архітектуру та дизайн, які ви розробили на попередніх етапах.

1.3.3.3.1 Вибір мов та технологій

Першим кроком у реалізації є вибір мов та технологій, які ви будете використовувати для створення свого фреймворка. Існує багато різних мов і технологій, які можна використовувати для розробки фреймворків для односторінкових веб-додатків.

Одним з популярних варіантів є використання JavaScript. JavaScript є мовою програмування, яка широко використовується для розробки веб-додатків. Він також є мовою програмування, яка підтримується більшістю веб-браузерів.

Іншим популярним варіантом є використання TypeScript. TypeScript - це розширення JavaScript, яке надає додаткові функції, такі як перевірка типів. Це може допомогти уникнути помилок і зробити код більш чітким.

На цьому етапі ми повинні повинні відповісти на такі питання:

- Яку мову програмування я хочу використовувати?

- Які технології я хочу використовувати?

1.3.3.3.2 Розробка компонентів

Після того, як ми обрали мову, ми можете приступити до розробки компонентів свого фреймворка.

Компоненти є основними будівельними блоками фреймворка. Вони відповідають за виконання певних завдань, таких як рендеринг інтерфейсу користувача, обробка подій або доступ до даних.

При розробці компонентів ви повинні враховувати такі фактори:

- Функціональність: Які функції повинен виконувати компонент?
- Інтерфейс: Який інтерфейс повинен мати компонент?
- Залежність: Від яких інших компонентів залежить компонент?
- Які компоненти мені потрібні?
- Які функції повинні виконувати компоненти?
- Який інтерфейс повинні мати компоненти?
- Від яких інших компонентів будуть залежати компоненти?

1.3.3.3.3 Взаємодія компонентів

Після того, як ми розробили компоненти, ми повинні визначити, як вони будуть взаємодіяти один з одним.

Ми повинні поділити їх на такі механізми:

- Події: Компоненти можуть посилати події один одному.
- Потоки: Компоненти можуть обмінюватися даними через потоки.
- Інтерфейси: Компоненти можуть взаємодіяти один з одним через інтерфейси.
- Які події будуть підтримуватися компонентами?
- Які потоки будуть використовуватися компонентами?
- Які інтерфейси будуть використовуватися компонентами?

1.3.3.3.4 Успішні практики

- Розбийте роботу на невеликі кроки. Це допоможе вам не відчувати себе перевантаженим.
- Часто тестуйте свою роботу. Це допоможе вам виявити помилки на ранніх етапах.

- Створюйте документацію. Це допоможе вам і іншим розробникам зрозуміти, як використовувати ваш фреймворк.

1.4 Висновки до першого розділу

У даному відділі ми зосередили свою увагу на аналізі різних мов програмування та фреймворків, які використовуються для розробки веб-додатків. Ми детально розглянули кожну з цих технологій, оцінили їхні переваги та недоліки, щоб надати повністю обґрунтоване порівняння, корисне для розробників та професіоналів у сфері програмування.

Окрім того, ми провели докладне дослідження етапів створення фреймворку, виокремлюючи такі ключові фази, як планування, проектування та реалізація. Ми докладно розглянули кожен з цих етапів, висвітливши їхню важливість у процесі розробки, а також виявили потенційні труднощі та виклики, з якими можуть зіткнутися розробники.

Наш аналіз дозволяє не лише зрозуміти сучасний ландшафт мов програмування та фреймворків, але й надає практичні вказівки та усвідомлення для тих, хто цікавиться оптимізацією процесу створення веб-додатків. Такий підхід допомагає розробникам вибрати оптимальні технології та ефективно керувати проектами на різних етапах їхнього розвитку.

2 JAVASCRIPT - ЯК ОСНОВНА МОВА ПРОГРАМУВАННЯ ВЕБ-ДОДАТКІВ

2.1 Види фреймворків для односторінкових веб-додатків на основі Javascript

2.1.1 React

2.1.1.1 Історія створення React

Історія створення React починається в 2011 році, коли Джордан Волк (Jordan Walke), програміст з Facebook, почав працювати над проектом, який згодом став React. Волк був незадоволений тим, як існуючі фреймворки для створення веб додатків, такі як AngularJS, працювали з великими застосунками. Він хотів створити бібліотеку, яка була б швидкою, масштабованою та легкою у використанні.

Волк черпав натхнення з XHP, фреймворку HTML для PHP. XHP використовує концепцію "віртуального DOM", яка дозволяє розробникам оновлювати лише частину DOM-дерева, коли дані змінюються. Волк вирішив використовувати цю концепцію в React.

Перша версія React була випущена в 2011 році. Вона була закритим кодом і використовувалася лише в Facebook. У 2013 році React був відкритий як проект з відкритим кодом. Це дозволило іншим розробникам використовувати бібліотеку і вносити свої пропозиції.

Завдяки своїй швидкості, масштабованості та простоті у використанні React швидко став популярним. Його почали використовувати в таких компаніях, як Instagram, Airbnb, Netflix і Sony.

У 2015 році Facebook випустив React Native, фреймворк для створення мобільних застосунків на основі React. React Native дозволяє розробникам використовувати одні й ті ж навички для створення веб застосунків і мобільних застосунків.

Сьогодні React є однією з найпопулярніших бібліотек для створення веб застосунків. Вона використовується в мільйонах застосунків по всьому світу.

Ось деякі важливі події в історії створення React:

2011 рік: Джордан Волк починає працювати над React.

2011 рік: Випущена перша версія React.

2013 рік: React відкритий як проект з відкритим кодом.

2015 рік: Випущений React Native.

2.1.1.2 Основні принципи роботи React

Основні принципи роботи React можна звести до наступних:

- Віртуальний DOM. React використовує концепцію віртуального DOM, яка дозволяє розробникам оновлювати лише частину DOM-дерева, коли дані змінюються. Це робить React більш ефективним, ніж традиційні фреймворки, які оновлюють все DOM-дерево при зміні даних.
- Компоненти. React використовує компоненти для організації коду. Компоненти - це самодостатні блоки коду, які відповідають за певну частину інтерфейсу користувача. Це робить React більш зручним у використанні та масштабуванні.
- Статичні типи. React підтримує статичні типи, які дозволяють розробникам перевіряти типи даних у коді. Це допомагає уникнути помилок і покращує читабельність коду.

DOM (Document Object Model) - це абстрактна модель документа, яка використовується для представлення веб-сторінки. DOM-дерево - це структура, яка представляє DOM документа.

React використовує віртуальний DOM для представлення DOM-дерева. Віртуальний DOM - це динамічна структура, яка оновлюється при зміні даних.

Коли дані змінюються, React оновлює віртуальний DOM. Потім React порівнює віртуальний DOM з фактичним DOM-деревом. Якщо в них є відмінності, React оновлює фактичний DOM-дерево.

Оновлення фактичного DOM-дерева відбувається через бібліотеку ReactDOM. ReactDOM - це бібліотека, яка забезпечує інтерфейс між React і DOM.

Компоненти - це самодостатні блоки коду, які відповідають за певну частину інтерфейсу користувача. Компоненти описуються за допомогою функцій.

Функція компонента приймає в якості аргументів дані. Компонент використовує ці дані для створення HTML-коду, який відображається в інтерфейсі користувача.

Компоненти можуть бути скомпонованими один з одним. Це дозволяє розробникам створювати складні інтерфейси користувача з невеликої кількості компонентів.

React підтримує статичні типи, які дозволяють розробникам перевіряти типи даних у коді. Це допомагає уникнути помилок і покращує читабельність коду.

Статичні типи в React реалізуються за допомогою бібліотеки TypeScript. TypeScript - це підмножина JavaScript, яка підтримує статичні типи.

Використання статичних типів у React дозволяє розробникам писати код, який є більш надійним і читабельним.

Ці принципи роблять React швидкою, масштабованою та легкою у використанні бібліотекою для створення веб аплікацій.

2.1.1.3 Реактивність в React

Реактивність в React - це механізм, який дозволяє React автоматично оновлювати інтерфейс користувача при зміні даних. Реактивність забезпечується за рахунок використання віртуального DOM і диспетчера станів.

Як було сказано раніше, React використовує віртуальний DOM для представлення DOM-дерева. Віртуальний DOM - це динамічна структура, яка оновлюється при зміні даних.

Коли дані змінюються, React оновлює віртуальний DOM. Потім React порівнює віртуальний DOM з фактичним DOM-деревом. Якщо в них є відмінності, React оновлює фактичний DOM-дерево.

Оновлення фактичного DOM-дерева відбувається через бібліотеку ReactDOM. ReactDOM - це бібліотека, яка забезпечує інтерфейс між React і DOM.

Диспетчер станів - це компонент, який використовується для зберігання стану компонента. Стан - це змінна, яка може змінюватися при взаємодії користувача з інтерфейсом.

React використовує диспетчер станів для відстеження змін стану компонента. Коли стан компонента змінюється, React автоматично оновлює інтерфейс компонента.

Реактивність в React працює по принципу get/set, та реалізована наступним чином:

1. Компонент отримує дані від батьківського компонента або з інших джерел.
2. Компонент зберігає дані у своєму стані.
3. Користувач взаємодіє з інтерфейсом компонента.
4. Компонент реагує на взаємодію користувача, змінюючи свій стан.
5. React відстежує зміну стану компонента.
6. React оновлює віртуальний DOM компонента.

7. React порівнює віртуальний DOM з фактичним DOM-деревом.
8. Якщо в них є відмінності, React оновлює фактичне DOM-дерево.

Завдяки цьому механізму React може автоматично оновлювати інтерфейс користувача при зміні даних. Це робить React більш ефективним і зручним у використанні.

2.1.2 Angular

2.1.2.1 Історія створення Angular

Історія створення Angular починається в 2009 році, коли Мішко Хевері (Miško Hevery) і Адам Абронс (Adam Abrons) заснували компанію Brat Tech LLC. Метою компанії було розробити програмне забезпечення для зберігання JSON-даних, що вимірюються мегабайтами. Для цього Хевері і Абронс розробили фреймворк під назвою AngularJS.

AngularJS був написаний на JavaScript і використовував декларативне програмування для побудови користувальницького інтерфейсу (UI). Цей підхід до програмування дозволяв розробникам описувати, як повинен виглядати і поводитись UI, без необхідності прописувати конкретні дії.

AngularJS швидко став популярним серед розробників, які створювали односторінкові веб-додатки (SPA). Він був добре документований, легко вивчався і підтримувався великим співтовариством.

У 2016 році компанія Google придбала Brat Tech LLC і взяла на себе розвиток AngularJS. Google переробила AngularJS, щоб зробити його більш сучасним і ефективним. Новий фреймворк отримав назву Angular.

Angular був написаний на TypeScript, мові програмування, яка є супер складовою JavaScript. TypeScript забезпечує додаткові можливості, такі як типи даних, що полегшують розробку і відлагодження коду.

Angular також включав ряд нових функцій, таких як:

- Модулі: Angular використовує модулі для організації коду додатка. Модулі дозволяють розробникам групувати пов'язаний код разом і забезпечувати його повторне використання.
- Компоненти: Angular використовує компоненти для побудови UI. Компоненти є самодостатніми блоками UI, які можна повторно використовувати в різних місцях додатка.
- Директиви: Angular використовує директиви для додавання додаткових функцій до HTML-коду. Наприклад, директиви можна використовувати для додавання взаємодії з користувачем або форматування тексту.
- Сервіси: Angular використовує сервіси для надання спільних ресурсів для компонентів. Сервіси можуть використовуватися для зберігання даних, надання логіки або доступу до зовнішніх API.

Angular швидко став популярним серед розробників, які створювали односторінкові веб-додатки. Він пропонує потужний набір функцій, які дозволяють розробникам створювати високоякісні SPA.

Ось деякі з ключових моментів історії створення Angular:

2009: Мішко Хевері і Адам Абронс засновують компанію Brat Tech LLC і розробляють AngularJS.

2016: Google придбав Brat Tech LLC і переробив AngularJS.

2016: Angular був випущений як новий фреймворк.

2017: Angular 2.0 був випущений з рядом нових функцій.

2022: Angular 14.0 був випущений з новими функціями і поліпшеннями.

Angular є одним з найпопулярніших фреймворків для розробки односторінкових веб-додатків. Він пропонує широкий спектр функцій, які дозволяють розробникам створювати високоякісні SPA.

2.1.2.2 Основні принципи роботи Angular

Основні принципи роботи Angular можна описати наступним чином:

Декларативність: Angular використовує декларативне програмування для побудови користувальницького інтерфейсу (UI). Цей підхід до програмування дозволяє розробникам описувати, як повинен виглядати і поводитися UI, без необхідності прописувати конкретні дії.

Організація коду: Angular використовує модулі для організації коду додатка. Модулі дозволяють розробникам групувати пов'язаний код разом і забезпечувати його повторне використання.

Компоненти: Angular використовує компоненти для побудови UI. Компоненти є самодостатніми блоками UI, які можна повторно використовувати в різних місцях додатка.

Директиви: Angular використовує директиви для додавання додаткових функцій до HTML-коду. Наприклад, директиви можна використовувати для додавання взаємодії з користувачем або форматування тексту.

Сервіси: Angular використовує сервіси для надання спільних ресурсів для компонентів. Сервіси можуть використовуватися для зберігання даних, надання логіки або доступу до зовнішніх API.

Декларативність є одним з основних принципів Angular. Це означає, що розробники описують, як повинен виглядати і поводитись UI, без необхідності прописувати конкретні дії. Наприклад, щоб додати кнопку до HTML-коду, розробник може використовувати таку директиву:

```
<button [disabled]="isDisabled">Кнопка</button>
```

Рисунок 2.1.2.2.1 Приклад використання декларативності в Angular

Ця директива буде відображати кнопку як відключену, якщо значення isDisabled дорівнює true. Розробник не повинен вказувати, як це значення буде встановлюватися.

Angular використовує модулі для організації коду додатка. Модулі дозволяють розробникам групувати пов'язаний код разом і забезпечувати його повторне використання. Наприклад, додаток може мати модуль для управління користувачами, модуль для управління продуктами і модуль для управління замовленнями.

Angular використовує компоненти для побудови UI. Компоненти є самодостатніми блоками UI, які можна повторно використовувати в різних місцях

додатка. Наприклад, додаток може мати компонент для відображення списку продуктів. Цей компонент можна використовувати в різних місцях додатка, наприклад, на головній сторінці, в кошику і на сторінці деталей продукту.

Angular використовує директиви для додавання додаткових функцій до HTML-коду. Наприклад, директиви можна використовувати для додавання взаємодії з користувачем або форматування тексту. Наприклад, директива ngFor може використовуватися для відображення списку елементів в циклі:

```
<ul>
  <li *ngFor="let product of products">
    {{ product.name }}
  </li>
</ul>
```

Рисунок 2.1.2.2.2 Приклад використання директив в Angular

Ця директива буде відображати список елементів, що містяться в змінній products.

Angular використовує сервіси для надання спільних ресурсів для компонентів. Сервіси можуть використовуватися для зберігання даних, надання логіки або доступу до зовнішніх API. Наприклад, додаток може мати сервіс для управління користувачами. Цей сервіс можна використовувати в різних компонентах, наприклад, в компоненті для авторизації і в компоненті для профілю користувача.

Ці принципи роботи Angular дозволяють розробникам створювати високоякісні односторінкові веб-додатки. Вони забезпечують гнучкість, повторне використання коду і ефективність розробки.

2.1.2.3 Реактивність в Angular

Реактивність в Angular - це парадигма програмування, яка дозволяє компонентам реагувати на зміни даних. Angular використовує бібліотеку RxJS для реалізації реактивності.

RxJS - це бібліотека для роботи з асинхронними потоками даних. Вона використовує модель push, яка означає, що компоненти отримують сповіщення про зміни даних, а не запитують їх самостійно.

Angular використовує get/set для реактивності в наступних випадках:

Для визначення властивостей, які повинні бути реактивними. Властивість вважається реактивною, якщо вона має setter-метод, який повертає Observable.

Для отримання доступу до реактивних властивостей. Компонент може отримати доступ до реактивної властивості, використовуючи оператор get().

Ось приклад використання get/set для реактивності в Angular:

```
@Component({
  selector: 'my-app',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {

  @Input() title = 'Angular';

  // Setter-метод реактивної властивості
  set title(value: string) {
    this._title = value;
    this.titleChange.next(value);
  }

  titleChange = new Subject<string>();

}
```

Рисунок 2.1.2.2.2 Приклад використання директив в Angular

У цьому прикладі властивість title є реактивною, оскільки вона має setter-метод, який повертає Observable. Setter-метод title() присвоює значення властивості _title, а потім ініціює поточний Observable для інформування про зміни.

Компонент може отримати доступ до реактивної властивості title за допомогою оператора get(). Наприклад, наступний код відобразить значення властивості title в HTML-коді:

Цей код буде ініціювати поточний Observable для отримання значення властивості title і відображати його в HTML-коді.

Реактивність в Angular робить додатки більш ефективними і гнучкими. Вона дозволяє компонентам реагувати на зміни даних в реальному часі, без необхідності постійно запитувати дані.

2.1.3 Vue

2.1.3.1 Історія створення Vue

Історія створення Vue починається в 2013 році, коли Еван Ю, тодішній співробітник Google, працював над одним із проектів. Йому була потрібна технологія, яка б дозволила швидко побудувати прототип складного інтерфейсу. React тоді був на ранній стадії розробки, а такі фреймворки, як AngularJS і Backbone.js, були занадто громіздкими для прототипування.

Еван вирішив створити свій власний фреймворк, який би був простим у використанні, але при цьому потужним і ефективним. Він назвав його Vue.

Перша версія Vue була випущена в грудні 2014 року. Вона швидко завоювала популярність серед розробників, які цінували її простоту, легкість у використанні та ефективність.

З тих пір Vue постійно розвивається і вдосконалюється. У 2016 році була випущена друга версія фреймворка, яка принесла ряд нових функцій і поліпшень. У 2018 році була випущена третя версія Vue, яка стала ще більш потужною і гнучкою.

На сьогоднішній день Vue є одним із найпопулярніших JavaScript-фреймворків для створення веб-інтерфейсів. Він використовується в таких відомих проектах, як GitLab, Netlify, Lazada, Twitch і багато інших.

Ось основні етапи розвитку Vue:

2013 рік: Еван Ю починає розробку Vue.

2014 рік: випущена перша версія Vue.

2016 рік: випущена друга версія Vue.

2018 рік: випущена третя версія Vue.

Простота у використанні. Vue має чіткий і зрозумілий синтаксис, що робить його простим у вивченні та використанні.

Потужності. Vue є потужним і ефективним фреймворком, який дозволяє створювати складні веб-інтерфейси.

Гнучкість. Vue є гнучким і адаптивним фреймворком, який можна використовувати для різних типів веб-застосунків.

2.1.1.2 Основні принципи роботи Vue

Компоненти. Vue використовує компоненти для створення веб-інтерфейсів. Компоненти - це самостійні блоки коду, які відповідають за відображення певної частини інтерфейсу.

Відношення батьків і дітей. Компоненти можуть бути пов'язані між собою за допомогою відносин батьків і дітей. Це дозволяє компонентам передавати один одному дані і керувати один одним.

Двосторонній зв'язок. Vue підтримує двосторонню зв'язок між даними і їх відображенням. Це означає, що зміни в даних автоматично відображаються в інтерфейсі, і навпаки.

Ось більш докладний опис кожного з цих принципів:

Компоненти - це основа Vue. Вони дозволяють розробникам створювати веб-інтерфейси з невеликих, самостійних блоків коду. Компоненти мають такі властивості:

- Ім'я. Ім'я компонента визначає його унікальний ідентифікатор.
- Синтаксис. Синтаксис компонента визначається його ім'ям і атрибутами.
- Відображення. Відображення компонента визначає, як він відображається в інтерфейсі.

Компоненти можна створювати вручну або за допомогою інструментів, таких як Vue CLI.

Компоненти можуть бути пов'язані між собою за допомогою відносин батьків і дітей. Це дозволяє компонентам передавати один одному дані і керувати один одним.

Компонент, який містить інші компоненти, називається батьківським компонентом. Компоненти, які містяться в батьківському компоненті, називаються дочірніми компонентами.

Дочірні компоненти можуть отримувати дані від батьківського компонента за допомогою атрибутів.

Vue підтримує двосторонню зв'язок між даними і їх відображенням. Це означає, что зміни в даних автоматически відображаются в інтерфейсе, и навпаки.

2.1.1.3 Реактивність в Vue

Реактивність - це одна з ключових особливостей Vue. Вона дозволяє Vue автоматично оновлювати інтерфейс при зміні даних.

Реактивність реалізована за допомогою проксі-об'єктів. Коли Ви присвоюєте значення змінній, Vue створює проксі-об'єкт для цієї змінної. Проксі-об'єкт відстежує всі зміни в змінній і повідомляє про них Vue.

Коли Vue отримує повідомлення про зміну в змінній, він запускає цикл оновлення. Цикл оновлення оновлює всі елементи HTML, які залежать від змінної.

Реактивність працює на всіх типах даних, включаючи змінні, масиви, об'єкти та навіть функції.

Ось кілька прикладів того, як реактивність можна використовувати в Vue:

Для зв'язку даних з елементами HTML. Наприклад, Ви можете використовувати директиву v-model для зв'язку значення поля введення з змінною.

Для відображення даних у таблицях. Наприклад, Ви можете використовувати директиву v-for для циклу по масивові даних та відображення кожного елемента в таблиці.

Для створення складних інтерфейсів. Наприклад, Ви можете використовувати реактивність для створення інтерфейсів, які реагують на дії користувача.

Реактивність - це потужна функція, яка дозволяє створювати веб-інтерфейси, які легко зрозуміти і використовувати.

2.2 Проблеми сучасних фреймворків

Сучасні фреймворки для створення односторінкових додатків (SPA) на основі JavaScript мають ряд проблем, які можуть ускладнити їх використання і розробку. Ось деякі з найпоширеніших проблем:

- Складність. SPA-фреймворки можуть бути досить складними для вивчення та використання. Вони часто мають великі бібліотеки і API, які потрібно освоїти.
- Продуктивність. SPA-фреймворки можуть бути менш продуктивними, ніж традиційні веб-додатки. Це пов'язано з тим, що SPA-додатки повинні завантажувати всі необхідні ресурси (дані, графіка, тощо) при першому завантаженні.
- Безпека. SPA-додатки можуть бути більш вразливими до атак, ніж традиційні веб-додатки. Це пов'язано з тим, що SPA-додатки часто зберігають дані на клієнтській стороні.

SPA-фреймворки часто мають великі бібліотеки і API, які потрібно освоїти. Це може бути складним завданням для початківців розробників. Крім того, SPA-фреймворки часто використовують складні концепції, такі як реактивність і управління станом, які можуть бути важкими для розуміння.

SPA-додатки повинні завантажувати всі необхідні ресурси (дані, графіка, тощо) при першому завантаженні. Це може призвести до зниження продуктивності, особливо на пристроях з обмеженими ресурсами.

SPA-додатки часто зберігають дані на клієнтській стороні. Це може зробити їх більш вразливими до атак, таких як XSS і CSRF.

Деякі з цих проблем можна вирішити за допомогою використання інструментів і практик, які полегшують розробку і використання SPA-додатків. Наприклад, існують інструменти, які можуть допомогти розробникам автоматизувати завдання, такі як створення компонентів і маршрутизація. Крім того, існують практики, такі як використання шаблонів і компонентів, які можуть допомогти покращити продуктивність і безпеку SPA-додатків.

Незважаючи на ці проблеми, SPA-фреймворки залишаються популярним вибором для розробки односторінкових додатків. Вони пропонують ряд переваг, таких як швидкість, інтерактивність і зручне використання.

2.2.1 Реактивність get/set і proxy object

Get/Set і proxy object – це два різні підходи до реалізації реактивності в JavaScript-фреймворках. Get/Set – це традиційний підхід, який використовує методи get() і set() для доступу до даних. Proxy object – це більш сучасний підхід, який використовує проксі-об'єкт для відстеження змін даних.

Get/Set має ряд переваг, включаючи:

- Простота. Get/Set – це простий підхід, який легко зрозуміти і використовувати.

- Ефективність. Get/Set може бути ефективним, якщо він використовується правильно.

Однак Get/Set також має ряд недоліків, включаючи:

- Небезпечність. Get/Set може бути небезпечним, якщо він використовується неправильно. Наприклад, якщо ви випадково використовуєте set() для зміни даних, які не повинні бути змінені, це може призвести до проблем.

- Негнучкість. Get/Set може бути негнучким, якщо ви хочете відстежувати складні зміни даних. Наприклад, якщо ви хочете відстежувати зміни в масивові даних, вам потрібно буде написати власний код.

Proxy object має ряд переваг, включаючи:

- Безпека. Proxy object є більш безпечним, ніж Get/Set, оскільки він не дозволяє випадково змінити дані.

- Гнучкість. Proxy є більш гнучким, ніж Get/Set, оскільки він дозволяє відстежувати складні зміни даних.

Однак Proxy також має ряд недоліків, включаючи:

- Складність. Proxy object може бути складним для розуміння і використання.

- Ефективність. Proxy object може бути менш ефективним, ніж Get/Set, якщо він використовується неправильно.

Get/Set може бути небезпечним, якщо він використовується неправильно. Наприклад, якщо ви випадково використовуєте set() для зміни даних, які не повинні бути змінені, це може призвести до проблем. Наприклад, якщо у вас є компонент, який відображає список продуктів, і ви випадково використовуєте set() для зміни ціни продукту, це може призвести до того, що користувачі будуть бачити неправильні ціни.

Get/Set може бути негнучким, якщо ви хочете відстежувати складні зміни даних. Наприклад, якщо ви хочете відстежувати зміни в масивові даних, вам потрібно буде написати власний код. Наприклад, якщо у вас є масив продуктів, і ви хочете відстежувати, коли додається новий продукт, вам потрібно буде написати свій власний код для цього.

Proxy може бути складним для розуміння і використання. Якщо ви не знайомі з проксі-об'єктами, вам може бути важко зрозуміти, як вони працюють.

Proxy може бути менш ефективним, ніж Get/Set, якщо він використовується неправильно. Наприклад, якщо ви використовуєте проксі-об'єкт для відстеження змін даних, які не змінюються часто, це може призвести до зниження продуктивності.

2.2 Висновки до другого розділу

У даному відділі нашої дослідженської роботи ми глибоко розглянули різні типи фреймворків, які знаходять широке застосування в сучасному програмуванні. Зокрема, ми докладно вивчили їхні переваги та недоліки, зокрема звертаючи увагу на ті аспекти, які можуть бути вирішені або покращені в подальших версіях. Цей аналіз є ключовим для розуміння того, як вибір конкретного фреймворку може вплинути на продуктивність та якість розробленого програмного продукту.

Крім того, ми зосередили свою увагу на виявленні проблем, які наразі виникають у фреймворках, виокремлюючи ті аспекти, які вимагають уваги та

подальшого дослідження. Це дозволяє нам не лише визначити поточні обмеження, але й створити підґрунтя для подальших розвідок та інновацій.

Одним із ключових висновків є наш аналіз різниці у реакції між традиційним методом get/set та новітнім підходом, який використовує proxy для створення веб-застосунків. Ми не лише виявили цю різницю, але й докладно проаналізували вплив кожного методу на продуктивність та витрати ресурсів системи. Це надає глибоке розуміння того, як нові технології можуть вдосконалити роботу з веб-застосунками та сприяти їхньому більш ефективному функціонуванню.

Загалом, наш аналітичний підхід до вивчення фреймворків та їхніх характеристик, а також порівняльний аналіз методів розробки веб-застосунків, створює основу для подальших досліджень у цій сфері. Ми впевнені, що розкриті у цьому розділі висновки будуть корисні для розробників та фахівців у галузі програмування, які прагнуть максимізувати ефективність своїх проектів та уникати потенційних проблем у майбутньому.

3 РЕАЛІЗАЦІЯ РЕАКТИВНОГО ФРЕЙМВОРКУ ДЛЯ ОДНОСТОРІНКОВИХ ВЕБ-ДОДАТКІВ

3.1 План створення фреймворку

- Компоненти: необхідно визначити структуру компонентів, базово вони поділяються на класові та функційні.
- JSX парсер: для взаємодії з компонентами та DOM необхідно описати парсер для jsx який зможе грамотно працювати з процесами
- Реактивність з Proxy-Reflect: потрібно створити набір методів для реактивного оновлення DOM при зміні будь-яких вхідних даних
- Virtual DOM: віртуальний DOM необхідний для оптимізації рендеру компоненту, оскільки спочатку віртуальний DOM має порівняти дані з реальним, і змінити реальний тільки у випадку невідповідності значень.
- Routing: будь-який односторінковий застосунок має мати роутінг, для переходу між “сторінками”
- State Management: фреймворк повинен мати можливість глобальної передачі даних між компонентами. Необхідно мати стратегію для роботи глобального стану додатку
- Lifecycle Methods: методи життєвого циклу додатку, які допоможуть нам керувати ним у кожен етап ініціалізації компонентів.
- Event Handling: необхідно прийняти рішення як будуть працювати події в фреймворку

- Build: потрібно створити процес мініфікації файлів коду в зашифрований робочий вигляд для подальшої публікації його на хостингу

3.2 Реалізація фреймворку

3.2.1 Структура компонентів

Як вже було зазначено компоненти бувають двох видів - класові та функційні. Функціональні компоненти зазвичай короткі та прості, а класові компоненти можуть мати більше можливостей для розширення. Для даної реалізації було обрано саме функційні компоненти, з мінімальним використанням методів життєвого циклу.

```
class Component<T extends object> {
  protected _data: Reactivity<T>;
  private _shadowRoot: ShadowRoot;

  constructor(data: T, private _container: HTMLElement) {
    this._data = new Reactivity(data);
    this._data.createEffect(() => this.render());

    this._shadowRoot = this._container.attachShadow({ mode:
"open" });
  }

  protected render(): void {
    throw new Error("Render method must be implemented in
derived classes");
  }
```

```
  protected attachToShadowRoot(element: HTMLElement): void {
    this._shadowRoot.appendChild(element);
  }
}
```

Лістинг 3.2.1.1

3.2.2 JSX парсер

JSX (JavaScript XML) - це розширення синтаксису JavaScript, яке дозволяє вам писати вигляд компонентів, схожий на HTML або XML, прямо в коді JavaScript. JSX є частиною сучасних фреймворків, таких як React, і значно полегшує роботу з відображенням інтерфейсу користувача.

Транспіляція (Transpilation): JSX не є прямо виконуваним в JavaScript. Перед тим, як браузер чи середовище Node.js зможе розуміти JSX, код потрібно трансполювати в стандартний JavaScript. Для цього зазвичай використовують інструменти транспіляції, такі як Babel.

Створення елементів: JSX використовується для опису відображення компонентів. Після транспіляції JSX перетворюється в виклики функцій, які створюють відображення елементів у JavaScript.

JSX парсер використовується для аналізу рядка, що містить JSX, та перетворення його в елементи JavaScript. Це необхідно для того, щоб відображати відображення компонентів, які описані за допомогою JSX, в реальному відображенні в браузері чи іншому середовищі виконання JavaScript.

Лексичний аналіз (Lexical Analysis): JSX парсер аналізує рядок, розбиваючи його на токени (лексеми), такі як теги, атрибути, текст і т.д.

Синтаксичний аналіз (Syntax Analysis): Використовуючи лексичний аналіз, парсер конструює абстрактне синтаксичне дерево (AST), що відображає структуру JSX-коду.

Трансформація в JavaScript: На основі AST парсер генерує еквівалентний JavaScript-код, що можна використовувати в середовищі виконання JavaScript.

Використання результатів: Отриманий JavaScript-код може використовуватися бібліотеками чи фреймворками (наприклад, React) для створення та відображення відображення компонентів.

Загальною метою JSX парсера є забезпечення інтеграції JSX в JavaScript-код, що дозволяє ефективно працювати з компонентами та їх відображенням.

3.2.3 Реактивність з Proxy-Reflect

Реактивність в фреймворках використовується для автоматичного виявлення та відслідковування змін в стані даних і автоматичного оновлення відображення, коли стан змінюється. Одним з підходів для досягнення реактивності є використання Proxy та Reflect в JavaScript.

Proxy - це об'єкт, який дозволяє перехоплювати базові операції над об'єктами та забезпечує можливість змінювати їх поведінку. Зазвичай використовується для створення "прослойки" навколо об'єкта, яка відслідковує доступ до властивостей.

Reflect - це вбудований об'єкт, який містить методи для виконання операцій над об'єктами. Використовується для виклику методів об'єкта, встановлення та отримання значень властивостей, тощо.

```
class Reactivity<T extends object> {
  private _data: T;
   private _dependencies: Map<keyof T, Set<() => void>> = new
Map();
  private _activeEffect: (() => void) | null = null;

  constructor(data: T) {
    this._data = new Proxy(data, {
      get: this._get.bind(this),
      set: this._set.bind(this),
    });
  }

  private _get(target: T, key: keyof T, receiver: T): any {
    if (this._activeEffect) {
      if (!this._dependencies.has(key)) {
        this._dependencies.set(key, new Set());
      }
      this._dependencies.get(key)?.add(this._activeEffect);
    }

    return Reflect.get(target, key, receiver);
  }

   private _set(target: T, key: keyof T, value: any, receiver:
T): boolean {
    Reflect.set(target, key, value, receiver);

    if (this._dependencies.has(key)) {
      const effects = this._dependencies.get(key);
      effects?.forEach((effect) => effect());
    }

    return true;
```

```
  }

  public createEffect(effect: () => void): void {
    this._activeEffect = effect;
    effect();
    this._activeEffect = null;
  }
}
```

Лістинг 3.2.3.1 Створення реактивного об'єкта

Об'єкт handler містить методи, які викликаються при спробі читання (get), запису (set), видаленні (delete), тощо.

При зчитуванні або встановленні значення властивості можна викликати функції для відслідковування змін та оновлення відображення. Після виявлення зміни в стані даних викликається код для оновлення відображення, наприклад, перерендерення компонентів у відповідь на зміни.

Завдяки використанню даного методу ми отримаємо такі покращення в реактивності:

- Спрощення коду: Використання Proxy-Reflect дозволяє створювати чистий та зрозумілий код для відслідковування та реагування на зміни в стані.
- Автоматичне виявлення змін:Proxy автоматично виявляє зміни в об'єкті, що спрощує реалізацію реактивності без необхідності вручну встановлювати "слухачів" на кожну властивість.
- Гнучкість: Застосування Proxy дозволяє створювати різні варіанти реактивності в залежності від потреб конкретного фреймворку чи бібліотеки.

- Безпека: Використання Proxy не вимагає прямого втручання в об'єкт, що дозволяє зберігати чистоту коду та зменшує ймовірність помилок.

3.2.4 Віртуальний DOM (Virtual DOM)

Віртуальний DOM - це концепція оптимізації веб-додатків, що полягає в тому, щоб уникнути зайвих операцій прямого маніпулювання реальним DOM за допомогою створення віртуального представлення структури DOM в пам'яті. Це віртуальне представлення дозволяє ефективно оновлювати реальний DOM лише тоді, коли це дійсно необхідно, тим самим покращуючи продуктивність веб-додатків.

Створення віртуального DOM:

Початковий стан веб-додатка представляється як віртуальний DOM, який є об'єктом або структурою даних, що містить інформацію про структуру DOM, але не має прив'язки до реального DOM.

```
     function createElementFromVNode(vnode: VNode): HTMLElement |
Text {
       if (typeof vnode.type === "string") {
         const element = document.createElement(vnode.type);

         if (vnode.attributes) {
           Object.keys(vnode.attributes).forEach((key) => {
             element[key] = vnode.attributes[key];
           });
         }
```

```
    if (vnode.children) {
      vnode.children.forEach((child) => {
        const childElement = createElementFromVNode(child);
        element.appendChild(childElement);
      });
    }

    return element;
  } else if (typeof vnode.type === "function") {
    const component = vnode.type();
    return createElementFromVNode(component);
  } else {
    return document.createTextNode(vnode as string);
  }
}
```

Лістинг 3.2.4.1

Рендеринг віртуального DOM:

Зміни стану додатка призводять до рендерингу нового віртуального DOM. Це може бути виклик функції, що повертає новий об'єкт віртуального DOM або вручну оновлений об'єкт.

```
function Counter(): VNode {
  const data = new CounterData();

  function increment(): void {
    data.createEffect(() => {
      data.count++;
    });
  }

  return (
    <div>
      <style>{`
        /* Add your styles here */
      `}</style>
```

```
      <p>Count: {data.count}</p>
      <button onClick={increment}>Increment</button>
    </div>
  );
}
```

Лістинг 3.2.4.2

Порівняння віртуального DOM:

При змінах стану порівнюється новий віртуальний DOM з попереднім. Цей процес називається diffing.

Створення патчів:

Алгоритим порівняння порівнює віртуальний DOM з shadow DOM і визначає, які саме зміни були внесені. В результаті створюються патчі, які представляють собою набір інструкцій для оновлення реального DOM.

Оновлення реального DOM:

Застосовуються патчі до реального DOM. Цей процес називається reconciliation. Лише ті частини DOM, які фактично змінилися, оновлюються.

Використання віртуального DOM має ряд переваг:

Ефективність:

Оптимізує продуктивність, оновлюючи лише ті елементи, які змінилися, а не всю структуру DOM.

Універсальність:

Незалежний від браузера і фреймворка, що дозволяє використовувати в різних умовах.

Спрощена робота з DOM:

Розробникам не потрібно безпосередньо маніпулювати реальним DOM, що полегшує роботу та зменшує ймовірність помилок.

Легка реалізація анімацій:

Зручний механізм для створення плавних анімацій через зміни стилів та класів елементів.

Спрощення роботи зі станом:

Зручно працювати зі станом, так як оновлення віртуального DOM автоматично визначає зміни.

3.2.5 Routing

Маршрутизація веб-додатків - це процес визначення, який компонент або сторінка відображається користувачеві в залежності від шляху (URL) чи інших критеріїв. У багатьох веб-додатках маршрутизація реалізується за допомогою бібліотек чи фреймворків, які забезпечують інструменти для роботи з маршрутами та відображенням вмісту.

Отже роутинг - важлива частина будь-якого односторінкового веб-додатку, основі його переваги:

Навігація між сторінками:

Routing дозволяє користувачам переходити між різними частинами додатку, відображаючи вміст, який відповідає конкретному URL.

Підтримка глибоких посилань:

Дозволяє створювати постійні посилання на конкретні сторінки додатку, що полегшує спільний доступ та зберігання посилань.

Управління станом відображення:

Роутери дозволяють визначати, які компоненти чи сторінки відображати в залежності від введеного користувачем URL. Це спрощує роботу зі станом відображення.

Розділення коду:

Маршрутизація може бути використана для розділення коду на "ліниві" (lazy) завантаження. Це дозволяє завантажувати компоненти лише тоді, коли вони потрібні для відображення певної сторінки.

Обробка помилок:

Можливість перенаправлення на сторінку з помилкою або створення власних сторінок 404 дозволяє покращити взаємодію з користувачем при неправильних запитах.

3.2.6 State Management

Управління станом в контексті веб-додатків визначає, як додаток зберігає та управляє своїм станом, тобто даними, які можуть змінюватися в ході його роботи. В правильно спроектованому додатку механізм управління станом дозволяє ефективно та прогнозовано обробляти та оновлювати дані, що використовуються в інтерфейсі користувача.

Отже для створення свого фреймворку необхідно було реалізувати:

Стан (State):

Стан - це колекція даних, які описують стан додатка в певний момент часу. Це може бути будь-яка інформація, яка може змінюватися, наприклад, дані форми, дані користувача, тощо.

```
private _state: Reactivity<{ [key: string]: any }> = new Reactivity({});
```

Лістинг 3.2.6.1

Імутабельність (Immutability):

Засновано на принципі, що стан не повинен змінюватися напряму. Зміни в стані створюються за допомогою нового об'єкта, а не модифікації поточного. Це полегшує відслідковування та управління станом.

```
public setState(newState: { [key: string]: any }): void {
    Object.keys(newState).forEach((key) => {
      this._state._data[key] = newState[key];
    });
  }
```

Лістинг 3.2.6.2

Однозначність джерела правди (Single Source of Truth):

Принцип, за яким стан додатка повинен зберігатися в одному об'єкті, що дозволяє легко відстежувати та управляти даними.

```
  public getState(): { [key: string]: any } {
    return this._state._data;
  }
```

Лістинг 3.2.6.3

Управління Ефектами (Side Effects):

Ефекти, такі як асинхронні запити до сервера чи зміни в браузерному хеші, краще управляти за допомогою спеціальних посередників або ефектів.

```
public createEffect(effect: () => void): void {
  this._state.createEffect(effect);
}
```

Лістинг 3.2.6.4

3.2.7 Методи життєвого циклу компонентів

Методи життєвого циклу важливі в будь-якому фреймворку. Вони повинні допомогти розробнику контролювати компонент на кожному етапі його створення.

Основні задачі з якими допомагають такі методи - це виклики асинхроних функцій для отримання необхідних компоненту даних, підключення слухачів до компненти, відключення слухачів в кінці життя компонентів, очищення стейту після завершення роботи з компонентами.

Для даного фреймворку було обрано 2 важливих метода життєвого циклу:

Mount - метод життєвого циклу який буде викликаний на етапі приєднання компоненту до аплікації

```
public onMounted(effect: EffectCallback): void {
  this._mountedEffects.push(effect);
}
public mount(): void {
  this._mountedEffects.forEach((effect) => effect());
}
```

Лістинг 3.2.7.1

Unmount - метод життєвого циклу який буде викликаний на етапі від'єднання компоненту від аплікації

```
public onUnmounted(effect: EffectCallback): void {
  this._unmountedEffects.push(effect);
}

public unmount(): void {
  this._unmountedEffects.forEach((effect) => effect());
}
```

Лістинг 3.2.7.2

3.2.8 Керування подіями

Кожен компонент аплікацій має мати можливість взаємодіяти один з одним. Це мають бути не тільки компоненти які розташовані на 1 рівні, а також компоненти які мають відношення батько-дитина.

Є 2 принципово різних варіанта реалізації даної задачі - передача функцій компонентам з збереженням контексту виконання, або передача тільки виклику функції, сам виклик в такому випадку буде здійснений саме в батьківському компоненті

Для даного фреймворку був обраний другий метод, який за допомогою спеціальних функцій emit буде викликати функції в батьківському компоненті.

```
public setEmit(emit: EmitFunction): void {
   this._emit = emit;
 }

 public emit(event: string, payload?: any): void {
   if (this._emit) {
     this._emit(event, payload);
   }
 }
```

Лістинг 3.2.7.1

3.2.9 Build

Процес побудови та мініфікації коду для його захисту та зменшення розміру файлів. Процес побудови буде розділений на кілька етапів:

1. Трансляція (Transpilation):

Використовується, наприклад, Babel, для перетворення сучасного коду JavaScript (ES6+, JSX) в код, який може бути виконаний в більш старих версіях браузерів або середовищ Node.js.

2. Збирання (Bundling):

Багатофайлові додатки зазвичай складаються з багатьох файлів. Збірник, такий як Webpack або Parcel, збирає ці файли в один або кілька файлів (зазвичай один JavaScript файл та один або декілька файлів CSS).

3. Мініфікація (Minification):

Код JavaScript, CSS та інших ресурсів мініфікується для зменшення їхнього об'єму та прискорення завантаження додатка. Зазвичай видаляються зайві пробіли, рядки, коментарі та інші "зайві" символи.

4. Оптимізація картинок (Image Optimization):

Зображення оптимізуються для зменшення їхнього розміру без втрати якості. Це може включати стиснення та конвертацію форматів.

5. Видалення неактивного коду (Tree Shaking):

Техніка, що дозволяє видаляти не використовуваний код (функції, класи, змінні) з результуючого бандлу. Це покращує розмір і швидкість завантаження.

6. Розділення коду (Code Splitting):

Розділення коду дозволяє розділяти код додатка на невеликі частини (chunks), які можуть бути завантажені асинхронно за запитом. Це поліпшує швидкість завантаження та ефективність додатка.

7. Генерація збірок для різних середовищ (Environment Builds):

Зазвичай генеруються різні збірки для різних середовищ, таких як розробка (development), тестування (test), та продакшн (production). Кожне середовище може включати різні налаштування (наприклад, виведення чи видалення дебаг-інформації).

8. Підписання (Code Signing) та Шифрування (Encryption):

3.3 Висновки до третього розділу

У цьому розділі ми зосередитися на розгляді реалізації іноваційного реактивного фреймворку, спрямованого на розробку односторінкових веб-додатків. Наш аналіз включає наведення коду реалізації з детальним розгортанням крок за кроком. Ми намагалися забезпечити чіткий та зрозумілий план, який дозволяє розробникам легко орієнтуватися в процесі реалізації фреймворку.

У тексті ми також включили приклад простого веб-додатку, розробленого з використанням нашого фреймворку. Цей приклад призначений для тестування та демонстрації можливостей, що були досягнуті в рамках даного дослідження. Ми намагались надати читачам не лише технічну інформацію, але й конкретні приклади використання, щоб допомогти їм краще розуміти контекст і вигоди використання цього фреймворку.

В цілому, наш підхід в даному розділі дозволяє не лише представити технічні аспекти розробки, а й надає практичний зразок для тестування та валідації створеного фреймворку. Це створює можливість для здійснення конкретних порівнянь та визначення потенційних варіантів використання даного інструменту в реальних проектах.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Аналіз стану умов праці

Роботи теоретичного, аналітично-пошукового та суспільно- гуманітарного спрямування переважно виконуються в кабінетах чи інших приміщеннях, де використовують різноманітне електрообладнання, зокрема персональні комп'ютери (ПК) та периферійні пристрої. Під час роботи з персональним комп'ютером можуть виникнути такі небезпечні та шкідливі чинники: несприятливі мікрокліматичні умови, освітлення, електромагнітні випромінювання, забруднення повітря шкідливими речовинами (джерелом яких може бути принтер, сканер), шум, вібрація, електричний струм, електростатичне поле, напруженість трудового процесу.

Особливу увагу під час написання роботи потрібно звертати увагу на приміщення, в яких виконувалась, як сама робота, так і її оформлення. Найбільш поширеною є така схема опису приміщення:

- місцезнаходження приміщення;
- загальна площа та об'єм приміщення;
- кількість робочих місць та місць обладнаних ПК;
- параметри мікроклімату, систем вентилювання та опалення;
- наявні системи освітлення та величини освітленості робочого місця;
- наявне обладнання, в тому числі, електричне та газове;

- наявність джерел небезпечних хімічних речовин, пилу та випромінювань;
- параметри напруженості трудового процесу;
- наявність засобів пожежогасіння та засобів надання долікарської допомоги;
- додаткові інструкції з охорони праці, що діють на робочому місці.

Роботи теоретичного та аналітично-пошукового характеру супроводжуються значним напруженням трудового процесу, певним емоційним напруженням та потребують належного психофізіологічного мікроклімату у колективі. Таким чином, здійснюючи характеристику процесу праці, необхідно звернути увагу на такі аспекти:

- фізичне навантаження (переважна робоча поза, ергономічна характеристика робочого місця, категорія робіт за ступенем важкості трудового процесу);
- нервово-психічна напруженість праці (напруженість зору, кількість і складність оброблювальної інформації, тривалість зосередженого спостереження, кількість об’єктів спостереження, наявність та тривалість технологічних перерв протягом робочого дня, шумове забруднення).

Під час виконання більшості робіт використовують персональні комп’ютери та периферійні пристрої (лазерні та струменеві друкарки, копіювальну техніку, сканери). Негативний вплив цих пристроїв на організм людини виникає через неадекватне (надто велике або надто мале) навантаження на окремі системи організму. Такі перекоси у напруженні різних систем організму, що трапляються під час роботи з ПК, зокрема, значна напруженість зорового аналізатора і довготривале малорухоме положення перед екраном, не тільки не зменшують

загального напруження, а навпаки, призводять до його посилення і прояву стресових реакцій. Найбільшому ризику виникнення різноманітних порушень піддаються: органи зору, м’язово-скелетна система, нервово-психічна діяльність, репродуктивна функція у жінок. Роботу персональних комп’ютерів та периферійних пристроїв супроводжує виділення багатьох хімічних речовин, зокрема озону, оксидів нітрогену та аерозолів (високодисперсних частинок тонера).

4.2 Організаційно-технічні заходи

Робоче місце – це зона трудових дій працівника, обладнання для виконання певних операцій виробничого процесу, де взаємодіють три головні елементи праці – предмет, засоби і суб’єкт праці. На одному робочому місці можуть працювати два або кілька працівників, які виконують спільне завдання. Наукова організація робочого місця передбачає створення працівникові всіх необхідних умов для високопродуктивної і високоякісної праці за менших фізичних зусиль і мінімальному нервовому напруженні та передбачає:

- оснащеність робочого місця відповідним основним і допоміжним устаткуванням, технологічною і організаційною оснасткою;
- раціональне планування, тобто найзручніше і найефективніше розміщення усіх елементів робочого місця для трудового процесу;
- створення безпечних і здорових умов праці.

Просторова організація робочого місця повинна забезпечувати:

- відповідність планування робочого місця санітарним і протипожежним нормам і вимогам;
- безпеку працівникам;
- відповідність просторових відношень між елементами робочого місця, антропометричними, біомеханічними, фізіологічними, психофізіологічними і психічними можливостями людини, що працює;
- можливість виконання основних і допоміжних операцій в робочому положенні, що відповідає специфіці трудового процесу, в раціональній робочій позі і з використанням найбільш ефективних прийомів праці;
- вільне переміщення працівника за оптимальними траєкторіями;
- достатню площу для розміщення обладнання, інструменту, засобів контролю, деталей.

Просторові та розмірні співвідношення між елементами робочого місця повинні дозволяти:

- розміщення працівника з урахуванням робочих рухів і переміщень згідно з технологічним процесом;
- оптимальний огляд джерела візуальної інформації;
- зміну робочої пози і положення;
- раціональне розміщення основних і допоміжних засобів праці.

Обов’язковою умовою є те, що на робочому місці повинні знаходитись лише ті технічні засоби, які необхідні для виконання робочого завдання, і розміщуватися вони повинні в межах досяжності з метою виключення частих нахилів і поворотів корпусу людини, що працює.

Під час роботи з персональним комп’ютером повинні бути дотримані певні вимоги.

Вимоги до приміщення. Площу приміщень, в яких розташовують персональні комп’ютери, визначають згідно з чинними нормативними документами з розрахунку на одне робоче місце, обладнане ПК:

- площа − менше 6,0 $м^2$;
- об’єм − не менше 20,0 $м^3$, з урахуванням максимальної кількості осіб, які одночасно працюють у зміні;
- робочі місця повинні бути розташовані на відстані не менше ніж 1 м від стіни з вікном;
- відстань між тильною поверхнею одного комп’ютера та екраном іншого не повинна бути меншою 2,5 м;
- прохід між рядами робочих місць має бути не меншим 1 м.

Вимоги до організації робочого місця з ПК. Конструкція робочого місця користувача ПК має забезпечувати підтримання оптимальної робочої пози з такими ергономічними характеристиками:

- ступні ніг − на підлозі або на підставці для ніг;
- стегна − в горизонтальній площині;
- передпліччя − вертикально;
- лікті − під кутом 70–90° до вертикальної площини;
- зап’ястя зігнуті під кутом не більше 20° відносно горизонтальної площини;
- нахил голови – 15-20° відносно вертикальної площини.

Якщо користування ПК є основним видом діяльності, то ПК і його периферійні пристрої (принтер, сканер тощо) розміщується на основному робочому столі, як правило, з лівого боку. Якщо використання ПК є періодичним, то він, як правило, розміщується на приставному столі, переважно з лівого боку від

основного робочого столу.

Робоче сидіння (сидіння, стілець, крісло) користувача ПК повинно мати такі основні елементи: сидіння, спинку, стаціонарні або знімні підлокітники. У конструкцію сидіння можуть бути введені додаткові елементи, що не є обов'язковими: підголовник та підставка для ніг. Робоче сидіння користувача ПК повинно бути підйомно поворотним, таким, що регулюється за висотою, кутом нахилу сидіння та спинки, за відстанню спинки до переднього краю сидіння, висотою підлокітників. Регулювання кожного параметрама є бути незалежним, плавним або ступінчатим, мати надійну фіксацію.

Монітор та клавіатура мають розташовуватися на оптимальній відстані від очей користувача, але не ближче 600 мм, з урахуванням розміру алфавітно-цифрових знаків та символів. Розташування монітору має забезпечувати зручність зорового спостереження у вертикальній площині під кутом ±30 град. від лінії зору працівника. Клавіатуру слід розміщувати на поверхні столу або на спеціальній регульованій за висотою, робочій поверхні окремо від столу на відстані 100–300 мм від краю, ближчого до працівника. Кут нахилу клавіатури має бути в межах 5–15 градусів.

Режим праці та відпочинку користувачів ПК встановлюють з урахуванням психофізіологічної напруженості їхньої праці, динаміки функціонального стану систем організму та працездатності. Раціональний режим праці та відпочинку передбачає запровадження регламентованих перерв, рівномірний розподіл навантажень протягом робочого дня, регулярні комплекси вправ для очей, рук, хребта, поліпшення мозкового кругообігу та психофізіологічне розвантаження.

З урахуванням характеру трудової діяльності, напруженості та важкості праці з використанням ПК під час основної роботи за восьми годинної робочої

зміни встановлюють додаткові регламентовані перерви:

- для розробників програм тривалістю 15 хв через кожну годину роботи;
- для операторів персональних комп'ютерів тривалістю 15 хв через дві години роботи;
- для операторів комп'ютерного набору тривалістю 10 хв через кожну годину роботи.

За жодних умов безперервна робота з ПК не повинна перевищувати чотири години. Також одним з важливих чинників, від якого залежать працездатність і здоров'я людини, – це освітлення. Світло регулює всі функції людського організму і впливає на психологічний стан і настрій, обмін речовин, гормональний фон і розумову активність. Найздоровіше освітлення забезпечує природне світло. Його ефективне використання можливе, якщо глибина приміщень не перевищує 6 м. Окрім того, хорошим вирішенням можуть будуть скляні перегородки, що забезпечують зорову і звукову ізоляцію, але в той же час не перешкоджають проникненню природного світла.

Ще варто звернути увагу на такий чинник, як шум , що часто є причиною зниження рівня працездатності, підвищення рівня загальної та професійної захворюваності, частоти виробничих травм. Шум як стрес-чинник є загальнобіологічним подразником, який негативно впливає на всі органи і системи організму.

Рівень шуму, що супроводжує роботу користувачів персональних комп'ютерів (зумовлений як роботою системних блоків, клавіатури, так і друкуванням на принтерах, а також зовнішніми чинниками), коливається у межах 50–65 дБА. Шум такої інтенсивності на тлі високого ступеня напруженості праці негативно впливає на функціональний стан користувачів. Тому на практиці

рекомендують знижувати фактичний рівень шуму у приміщеннях, де створюють комп’ютерні програми, виконують теоретичні та творчі роботи.

Особливо важливим є дотримання заходів особистої гігієни на робочому місці, а саме щоденне вологе прибирання, утримання у чистоті робочого місця, наявність на робочому місці тільки необхідних для роботи засобів. На робочому місці необхідно дотримуватись вимог правил внутрішнього розпорядку.

4.3 Безпека в надзвичайних ситуаціях

Надзвичайна ситуація – порушення нормальних умов життя і діяльності людей на об’єкті або території, спричинене аварією, катастрофою,стихійним лихом чи іншою небезпечною подією, яка призвела (може призвести) до загибелі людей або значних матеріальних втрат.

Серед надзвичайних ситуацій техногенного характеру домінують пожежі та вибухи, а серед небезпек природного характеру – аномальні гідрометеорологічні явища та медико-біологічні загрози. Пожежа – це неконтрольоване горіння, яке супроводжується виділенням тепла, світла, диму та інших продуктів. Горіння виникає за таких трьох умов: наявності окисника, наявності горючої речовини, наявності температури, за якої горюча речовина може самостійно горіти. Якщо немає хоча б однієї із цих умов, горіння стає неможливим. На цьому постулаті ґрунтується переважна більшість профілактичних заходів, спрямованих на відвернення пожеж.

Описуючи пожежовибухонебезпечність середовища варто відзначити: пожежовибухонебезпечні властивості речовин і матеріалів, які використовують під час виконання дипломних (кваліфікаційних) робіт (горючість, верхня та нижня концентраційні межі загоряння, температура запалення).

Головними причинами виникнення пожеж та вибухів є:

- порушення пожежних норм і правил;
- порушення правил встановлення та експлуатації систем енергопостачання, опалення, вентиляції;
- порушення правил експлуатації електричного та газового обладнання;
- порушення правил зберігання пожежовибухонебезпечних матеріалів;
- використання відкритого вогню в заборонених місцях;
- погане знання персоналом протипожежних правил;
- необережна поведінка з вогнем.

Серед загальних вимог до евакуаційних шляхів та виходів необхідно відмітити, що ними можуть бути дверні отвори, якщо вони ведуть з приміщень:

- безпосередньо назовні;
- на сходовий майданчик з виходом назовні безпосередньо або через вестибюль;
- у прохід або коридор з безпосереднім виходом назовні або на сходовий майданчик;
- у сусідні приміщення того ж поверху, що не містять виробництв, які належать за вибухопожежною та пожежною небезпекою до категорій А, Б і В та мають безпосередній вихід назовні або на сходовий майданчик.

Для безпечної евакуації шляхи та виходи мають відповідати таким вимогам:

- евакуаційні шляхи і виходи повинні утримуватися вільними, не захаращуватися та у разі потреби забезпечувати евакуацію всіх людей, які перебувають у приміщеннях;
- кількість та розміри евакуаційних виходів, їхні конструктивні рішення, умови освітленості, забезпечення не задимленості, протяжність шляхів евакуації, їхнє оздоблення повинні відповідати протипожежним вимогам будівельних норм;
- у приміщенні, яке має один евакуаційний вихід, дозволяється одночасно розміщувати не більше 50 осіб, а у разі перебування в ньому понад 50 осіб повинно бути щонайменше два виходи, які відповідають вимогам будівельних норм;
- двері на шляхах евакуації повинні відчинятися в напрямку виходу з будівель (приміщень) і замикатися лише на внутрішні запори, які легко відмикаються.

ВИСНОВКИ

У першому розділі ми зосередили свою увагу на аналізі різних мов програмування та фреймворків, які використовуються для розробки веб-додатків. Ми детально розглянули кожну з цих технологій, оцінили їхні переваги та недоліки, щоб надати повністю обґрунтоване порівняння, корисне для розробників та професіоналів у сфері програмування.

Окрім того, ми провели докладне дослідження етапів створення фреймворку, виокремлюючи такі ключові фази, як планування, проектування та реалізація. Ми докладно розглянули кожен з цих етапів, висвітливши їхню важливість у процесі розробки, а також виявили потенційні труднощі та виклики, з якими можуть зіткнутися розробники.

Наш аналіз дозволяє не лише зрозуміти сучасний ландшафт мов програмування та фреймворків, але й надає практичні вказівки та усвідомлення для тих, хто цікавиться оптимізацією процесу створення веб-додатків. Такий підхід допомагає розробникам вибрати оптимальні технології та ефективно керувати проектами на різних етапах їхнього розвитку.

В другому розділі ми глибоко розглянули різні типи фреймворків, які знаходять широке застосування в сучасному програмуванні. Зокрема, ми докладно вивчили їхні переваги та недоліки, зокрема звертаючи увагу на ті аспекти, які можуть бути вирішені або покращені в подальших версіях. Цей аналіз є ключовим для розуміння того, як вибір конкретного фреймворку може вплинути на продуктивність та якість розробленого програмного продукту.

Крім того, ми зосередили свою увагу на виявленні проблем, які наразі виникають у фреймворках, виокремлюючи ті аспекти, які вимагають уваги та подальшого дослідження. Це дозволяє нам не лише визначити поточні обмеження, але й створити підґрунтя для подальших розвідок та інновацій.

Одним із ключових висновків є наш аналіз різниці у реакції між традиційним методом get/set та новітнім підходом, який використовує proxy для створення веб-застосунків. Ми не лише виявили цю різницю, але й докладно проаналізували вплив кожного методу на продуктивність та витрати ресурсів системи. Це надає глибоке розуміння того, як нові технології можуть удосконалити роботу з веб-застосунками та сприяти їхньому більш ефективному функціонуванню.

Загалом, наш аналітичний підхід до вивчення фреймворків та їхніх характеристик, а також порівняльний аналіз методів розробки веб-застосунків, створює основу для подальших досліджень у цій сфері. Ми впевнені, що розкриті у цьому розділі висновки будуть корисні для розробників та фахівців у галузі програмування, які прагнуть максимізувати ефективність своїх проектів та уникати потенційних проблем у майбутньому.

В третьому розділі ми зосередилися на розгляді реалізації іноваційного реактивного фреймворку, спрямованого на розробку односторінкових веб-додатків. Наш аналіз включає наведення коду реалізації з детальним розгортанням крок за кроком. Ми намагалися забезпечити чіткий та зрозумілий план, який дозволяє розробникам легко орієнтуватися в процесі реалізації фреймворку.

У тексті ми також включили приклад простого веб-додатку, розробленого з використанням нашого фреймворку. Цей приклад призначений для тестування та демонстрації можливостей, що були досягнуті в рамках даного дослідження. Ми намагались надати читачам не лише технічну інформацію, але й конкретні

приклади використання, щоб допомогти їм краще розуміти контекст і вигоди використання цього фреймворку.

В цілому, наш підхід в даному розділі дозволяє не лише представити технічні аспекти розробки, а й надає практичний зразок для тестування та валідації створеного фреймворку. Це створює можливість для здійснення конкретних порівнянь та визначення потенційних варіантів використання даного інструменту в реальних проектах.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Flanagan, D. (2011). "JavaScript: The Definitive Guide." O'Reilly Media.

2. Krill, P. (2017). "What is reactive programming? Learn to use this development paradigm." InfoWorld. [Online] Available: https://www.infoworld.com/article/3174427/what-is-reactive-programming-learn-to-use-this-development-paradigm.html

3. Newman, S. (2017). "Building Microservices: Designing Fine-Grained Systems." O'Reilly Media.

4. Mezzalira, L. (2018). "Building scalable and maintainable frontend architectures." O'Reilly Media.

5. Freeman, A., & Robson, J. (2015). "Programming Reactive Extensions and LINQ." O'Reilly Media.

6. Crockford, D. (2008). "JavaScript: The Good Parts." O'Reilly Media.

7. Lerner, J. (2017). "Functional Programming in Python: How to Improve Your Python Programs." Apress.

8. Wilson, E. (2019). "Reactive Programming with RxJS 5: Untangle Your Asynchronous JavaScript Code." Packt Publishing.

9. Cooper, D., & Harrison, P. (2018). "Full Stack Development with JHipster." O'Reilly Media.

10. Morrison, M. (2018). "Reactive Design Patterns." Manning Publications.

11. О. А. Гринчишин. Теорія та практика створення програмних систем: навч. посіб. для студ. вищ. навч. закл. III-IV рівнів акредитації / О. А. Гринчишин,

О. П. Кушніренко, О. Г. Пухальський, В. В. Кушніренко. — 2-ге вид., перероб. і доп. — Київ: Вид-во НТУУ "КПІ", 2012. — 488 с.

12. О. А. Гринчишин. Основи програмування: навч. посіб. для студ. вищ. навч. закл. III-IV рівнів акредитації / О. А. Гринчишин, О. П. Кушніренко, О. Г. Пухальський, В. В. Кушніренко. — 2-ге вид., перероб. і доп. — Київ: Вид-во НТУУ "КПІ", 2012. — 288 с.

13. О. А. Гринчишин. Вступ до обчислювальної техніки: навч. посіб. для студ. вищ. навч. закл. III-IV рівнів акредитації / О. А. Гринчишин, О. П. Кушніренко, О. Г. Пухальський, В. В. Кушніренко. — 2-ге вид., перероб. і доп. — Київ: Вид-во НТУУ "КПІ", 2012. — 320 с.

14. Martin Odersky. The Future of the Java Virtual Machine. ACM SIGPLAN Notices, 2015.

15. Robert Griesemer, Rob Pike, Ken Thompson. The Design and Implementation of the Go Programming Language. Proceedings of the 2009 ACM SIGPLAN Conference on Programming Language Design and Implementation, 2009.

16. Evan Czaplicki, David Nolen, Brian Lonsdorf, Matt Dey. Elm: A Functional Language for Web Applications. Proceedings of the 2012 ACM SIGPLAN Workshop on Haskell and the Real World, 2012.

17. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. Introduction to Algorithms. MIT Press, 2009.

18. Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman. Compilers: Principles, Techniques, and Tools. Addison-Wesley, 2007.

19. Andrew S. Tanenbaum, Maarten van Steen. Modern Operating Systems. Pearson Education, 2018.

20. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

21. Martin Fowler, Kent Beck, John Brant, William Opdyke, Erich Gamma. Refactoring: Improving the Design of Existing Code. Addison-Wesley, 1999.

22. Robert C. Martin. Clean Code: A Handbook of Agile Software Craftsmanship. Pearson Education, 2008.

ДОДАТКИ

Додаток А

Програмний код

```
      type Data = {
  count: number;
};

type VNode = {
  type: string | Function;
  attributes: { [key: string]: any } | null;
  children: (string | VNode)[];
};

type ElementAttributes = { [key: string]: string | ((event: Event) =>
void) };
type Child = string | VNode;

type EffectCallback = () => void;
type EmitFunction = (event: string, payload?: any) => void;

function createElement(
  type: string | Function,
  attributes: ElementAttributes | null,
  ...children: Child[]
): VNode {
  return { type, attributes, children };
}

class Reactivity<T extends object> {
  private _data: T;
  private _dependencies: Map<keyof T, Set<() => void>> = new Map();
  private _activeEffect: EffectCallback | null = null;
  private _mountedEffects: EffectCallback[] = [];
  private _unmountedEffects: EffectCallback[] = [];
  private _emit: EmitFunction | null = null;

  constructor(data: T) {
    this._data = new Proxy(data, {
      get: this._get.bind(this),
      set: this._set.bind(this),
    });
  }
```

```
        private _get(target: T, key: keyof T, receiver: T): any {
          if (this._activeEffect) {
            if (!this._dependencies.has(key)) {
              this._dependencies.set(key, new Set());
            }
            this._dependencies.get(key)?.add(this._activeEffect);
          }

          return Reflect.get(target, key, receiver);
        }

        private _set(target: T, key: keyof T, value: any, receiver: T):
boolean {
          Reflect.set(target, key, value, receiver);

          if (this._dependencies.has(key)) {
            const effects = this._dependencies.get(key);
            effects?.forEach((effect) => effect());
          }

          return true;
        }

        public createEffect(effect: EffectCallback): void {
          this._activeEffect = effect;
          effect();
          this._activeEffect = null;
        }

        public onMounted(effect: EffectCallback): void {
          this._mountedEffects.push(effect);
        }

        public onUnmounted(effect: EffectCallback): void {
          this._unmountedEffects.push(effect);
        }

        public mount(): void {
          this._mountedEffects.forEach((effect) => effect());
        }

        public unmount(): void {
          this._unmountedEffects.forEach((effect) => effect());
        }
```

```
  public setEmit(emit: EmitFunction): void {
    this._emit = emit;
  }

  public emit(event: string, payload?: any): void {
    if (this._emit) {
      this._emit(event, payload);
    }
  }
}

function createComponent<T extends object>(
  render: () => VNode,
  data: T,
  container: HTMLElement
): Reactivity<T> {
  const reactivity = new Reactivity(data);

  reactivity.createEffect(() => {
    const vdom = render();
    update(container, vdom);
  });

  return reactivity;
}

function update(container: HTMLElement, vnode: VNode): void {
  const newElement = createElementFromVNode(vnode);
  container.innerHTML = "";
  container.appendChild(newElement);
}

function createElementFromVNode(vnode: VNode): HTMLElement | Text {
  if (typeof vnode.type === "string") {
    const element = document.createElement(vnode.type);

    if (vnode.attributes) {
      Object.keys(vnode.attributes).forEach((key) => {
        element[key] = vnode.attributes[key];
      });
    }

    if (vnode.children) {
      vnode.children.forEach((child) => {
        const childElement = createElementFromVNode(child);
```

```
            element.appendChild(childElement);
          });
        }

        return element;
      } else if (typeof vnode.type === "function") {
        const component = vnode.type();
        return createElementFromVNode(component);
      } else {
        return document.createTextNode(vnode as string);
      }
    }

    // Example usage:

    class CounterData {
      count: number = 0;
    }

    function Counter(): VNode {
      const data = new CounterData();

      function increment(): void {
        data.createEffect(() => {
          data.count++;
        });
      }

      return (
        <div>
          <style>{`
            /* Add your styles here */
          `}</style>
          <p>Count: {data.count}</p>
          <button onClick={increment}>Increment</button>
        </div>
      );
    }

    const appContainer = document.getElementById("app");
    if (appContainer) {
      const counter = createComponent(Counter, new CounterData(),
appContainer);
    }
```

```
      // State manager
      class StateManager {
        private static _instance: StateManager | null = null;

        private _state: Reactivity<{ [key: string]: any }> = new
Reactivity({});

        private constructor() {}

        public static getInstance(): StateManager {
          if (!StateManager._instance) {
            StateManager._instance = new StateManager();
          }
          return StateManager._instance;
        }

        public getState(): { [key: string]: any } {
          return this._state._data;
        }

        public setState(newState: { [key: string]: any }): void {
          Object.keys(newState).forEach((key) => {
            this._state._data[key] = newState[key];
          });
        }

        public createEffect(effect: () => void): void {
          this._state.createEffect(effect);
        }
      }

// Router
import { VNode, update } from "./framework";

      export type RouteHandler = () => VNode;

      export interface Route {
        path: string;
        handler: RouteHandler;
      }

      export class Router {
        private routes: Route[] = [];
```

```
    constructor(private container: HTMLElement) {}

    addRoute(path: string, handler: RouteHandler): void {
      this.routes.push({ path, handler });
    }

    navigate(path: string): void {
      const route = this.routes.find((r) => r.path === path);

      if (route) {
        const vdom = route.handler();
        update(this.container, vdom);
      } else {
        console.error(`Route not found: ${path}`);
      }
    }
  }



  // Init

  const appContainer = document.getElementById("app");
  if (appContainer) {
    const app = createComponent(App, new AppData(), appContainer);
  }



  // Parent component with JSX
  const initialData = { count: 0 };

  const parentContainer = document.getElementById("parent-container")
as HTMLElement;

  const parentReactivity = createComponent(() => {
    const data = { count: 0 };
    const increment = () => {
      data.count += 1;
    };

    const counterReactivity = createComponent(
      (emit) => <Counter emit={emit} count={data.count}
increment={increment} />,
      data,
```

```
        parentContainer
      );

      counterReactivity.setEmit((event, payload) => {
        if (event === "customEvent") {
          console.log("Custom event triggered with payload:", payload);
        }
      });

      return (
        <div>
          <h2>Parent Component</h2>
          <button onclick={() => counterReactivity.emit("customEvent", {
message: "Hello from parent!" })}>
            Trigger Custom Event
          </button>
          <Counter emit={counterReactivity.emit.bind(counterReactivity)}
count={data.count} increment={increment} />
        </div>
      );
    }, initialData, parentContainer);
```

Додаток В
Тези

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ПУЛЮЯ

М А Т Е Р І А Л И

XI НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

13-14 грудня 2023 року

ТЕРНОПІЛЬ
2023

УДК 004.41

С.В. Осельський, Г. Б. Цуприк, канд. техн. наук, доц.
Тернопільський національний технічний університет імені Івана Пулюя, Україна

РОЗРОБКА РЕАКТИВНОГО ФРОНТЕНД ФРЕЙМВОРКУ ДЛЯ ОДНОСТОРІНКОВИХ ДОДАТКІВ З ВЛАСНОЮ СИСТЕМОЮ РЕАКТИВНОСТІ

S.V. Oselskyi, Tsupryk, PhD, Assoc.Prof.

DEVELOPMENT OF A REACTIVE FRONTEND FRAMEWORK FOR SINGLE-PAGE APPLICATIONS WITH ITS OWN REACTIVITY SYSTEM

Створення реактивного фронтенд фреймворку для односторінкових додатків з власною системою реактивності є суттєвим етапом у контексті високих вимог до сучасних технологій. Від початку військового стану кількість підприємств, що перевели свої системи в хмарні рішення зросла в рази, що демонструє зростаючий попит на ці технології. Особливо важливим є створення таких рішень, які дозволяють швидко та ефективно розгортати сучасні веб-додатки в умовах стрімкого темпу цифрової трансформації.

На даний момент на ринку присутні різні рішення для даної проблеми, реалізовані різними мовами програмування. Однак, слід врахувати, що браузери інтерпретують лише Javascript, що робить його практично стандартом для веб-розробки. Обираючи фреймворк, написаний саме на цій мові, ми максимізуємо сумісність та легкість інтеграції, що сприяє оперативному створенню рішень.

Найпопулярніші фреймворки, такі як React, Vue, та Angular, базуються на старій системі реактивності, що використовує принципи get/set. Обраним фреймворком ви пропонуєте новий підхід до реактивності, який має потенціал пришвидшити виконання коду.

Також, слід враховувати, що успішне впровадження обраного фреймворку вимагає від розробників певного рівня технічної компетентності. Тут ви подаєте перелік ключових технологій, якими розробники повинні володіти, зокрема це система управління станом, JSX для комбінації HTML з кодом, та Router для роботи з односторінковими додатками. Крім цього, важливим елементом є використання Shadow DOM, що сприяє оптимізації роботи рендеру сторінки.

Shadow DOM - це технологія, яка дозволяє ізолювати та інкапсулювати структуру HTML, CSS і JavaScript компонента, захищаючи від впливу зовнішніх стилів та скриптів. Це сприяє не тільки ефективній адаптації команди, але й робить процес розробки більш доступним і зрозумілим для широкого кола фахівців.

У результаті, розробка реактивного фронтенд фреймворку з наявністю всіх типових рішень і власною системою реактивності, може бути потужним інструментом для розробки різноманітних сервісів, необхідних для бізнесу. Гнучкість та універсальність даного фреймворку дозволить розробникам творчо підходити до створення різноманітних веб-додатків.

Література

1. Professional JavaScript for Web Developers (Tech Today) 5th / Matt Frisbie; 2023. – 1104 с.

2. Understanding client-side JavaScript frameworks, URL: https://shorturl.at/aNS58

3. М.Р. Петрик, Д.М. Михалик, О.Ю. Петрик, Г.Б. Цуприк. Методичні вказівки до виконання атестаційної роботи магістра за спеціальністю 121 – “Інженерія програмного забезпечення” для усіх форм навчання [Текст] – Тернопіль : Тернопільський національний технічний університет імені Івана Пулюя – 2020 – 27 с.