БАКАЛАВРСЬКА РОБОТА

ДРБ. ІП - 15.00.00.000 ПЗ

Група ІП-21-1

Каблак Максим

2025

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Каблак Максим Миколайович

(прізвище, ім'я, по батькові)

УДК 004.942
(індекс)

БАКАЛАВРСЬКА РОБОТА

Технології та фреймворки ВЕБ-розробки (назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня _____Каблак Максим Миколайович_____
(підпис, ініціали та прізвище здобувача)

Науковий керівник _____Гобир Ліда Мирославівна, асистент_____
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри

_доц._____Бандура В.В.__
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2025

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доцент. В.В. Бандура
“ ____ ” _____2024 р.

ЗАВДАННЯ
НА ДИПЛОМНУ РОБОТУ БАКАЛАВРА СТУДЕНТОВІ

Каблак Максим Миколайович

(прізвище, ім’я, по-батькові)

1. Тема проекту (роботи) "Технології та фреймворки ВЕБ-розробки" .

керівник проекту (роботи) асистент_Гобир Ліда Мирославівна

затверджені наказом вищого навчального закладу від “ 28_ ” _квітня 2025 р. № ___264/7____

2. Строк подання студентом проекту (роботи) 10 червня 2025 р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

4. Зміст розрахунково - пояснювальної записки(перелік питань, які потрібно розробити)

1. Основи технологій та інструментів веб-розробки
2. Основи веб-розробки та технологічний стек ....
3. Інструменти та процеси веб-розробки

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Мову відображення можна визначити у властивості (рис.2.1 ст.17)
2. Шаблон HTML, який використовує React. (рис.2.3 ст.18)
3. Як DOM і Virtual DOM повторно візуалізують дерево компонентів (рис.2.9 ст.27)
4. Порожній модуль експортовано для представлення активів (рис.3.19 ст.60)
5. Як використовується функція налаштування магазину Redux (рис.3.21 ст.63)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

2. Дата видачі завдання 10 травня 2025 р.

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Визначення та обгрунтування теми роботи	15.02.2025	виконано
2	Огляд існуючих концепцій, рішень та сервісів в даній області	25.02.2025	виконано
3	Побудова моделі або алгоритму власного рішення	15.03.2025	виконано
4	Документування реалізації власного оригінального рішення вибраними засобами	25.04.2025	виконано
5	Оформлення пояснювальної записки бакалаврської роботи	10.06.2025	виконано

Студент _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Бакалаврська робота містить 70 сторінок, 44 рисунки, список використаних джерел із 20 найменування,

Метою роботи проаналізувати основні технології, фреймворки та інструменти веб-розробки, визначити їх роль у створенні сучасних веб-додатків, а також дослідити процеси оптимізації, тестування та розгортання у середовищі веб-розробки.

Об'єкт дослідження: процес веб-розробки як складова сучасної IT-середовища.

Предмет дослідження: сучасні технології, фреймворки, інструменти та методології, що застосовуються у веб-розробці на різних етапах створення веб-додатків.

Результати дослідження: робота має практичну цінність та сприяє розумінню сильних і слабких сторін кожного фреймворку в різних сценаріях використання.

В першому розділі розглянуто теоретичні основи веб-розробки, включаючи принципи роботи клієнтських і серверних технологій, архітектурні підходи

В другому розділі роботи здійсненно порівняльне тестування фреймворків на основі розробки прототипів веб-додатків із однаковим функціоналом.

В третьому розділі здійснено аналіз результати тестування доповнено аналізом документації, екосистеми бібліотек і підтримки спільноти для кожного фреймворку.

Висновок: У ході виконання бакалаврської роботи було досліджено сучасні підходи до веб-розробки як ключового елементу цифрової економіки.

КЛЮЧОВІ СЛОВА: ВЕБ-РОЗРОБКА, ФРЕЙМВОРКИ, REACT, ANGULAR, VUE.JS, DJANGO, FLASK, NODE.JS, ПРОДУКТИВНІСТЬ, МАСШТАБУВАННЯ, ПОРІВНЯЛЬНИЙ АНАЛІЗ.

ANNOTATION

The bachelor's thesis contains 70 pages, 44 figures, a list of used sources with 20 names,

The purpose of the work is to analyze the main technologies, frameworks and tools of web development, determine their role in creating modern web applications, as well as to investigate the processes of optimization, testing and deployment in the web development environment.

Object of research: the web development process as a component of the modern IT environment.

Subject of research: modern technologies, frameworks, tools and methodologies used in web development at different stages of creating web applications.

Research results: the work has practical value and contributes to understanding the strengths and weaknesses of each framework in different usage scenarios.

The first section examines the theoretical foundations of web development, including the principles of client and server technologies, architectural approaches

In the second section of the work, comparative testing of frameworks is carried out based on the development of prototypes of web applications with the same functionality.

In the third section, the analysis of the testing results is supplemented by an analysis of the documentation, library ecosystem and community support for each framework.

Conclusion: During the bachelor's thesis, modern approaches to web development as a key element of the digital economy were investigated.

KEYWORDS: WEB DEVELOPMENT, FRAMEWORKS, REACT, ANGULAR, VUE.JS, DJANGO, FLASK, NODE.JS, PRODUCTIVITY, SCALABILITY, COMPARATIVE ANALYSIS.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

HTML - мови гіпертекстової розмітки

CSS - каскадних таблиць стилів

ООР - об’єктно-орієнтоване програмування

WWW - Всесвітньої павутини

MVC - Model-View-Controller

UI - інтерфейс користувача

IDE - інтегрованого середовища розробки

SPA - Single Page Application

ЗМІСТ

					БР.ІП – 15.00.00.000 ПЗ		
Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*			
Розроб.		Каблак М. М			**Порівняльний аналіз технологій та фреймворків ВЕБ-розробки Пояснювальна записка**	*Літ.*	*Арк.* 6 / *Акрушів*
Перевір.		Гобир Л.М.					
Реценз.						**ІФНТУНГ ІП-21-1**	
Н. Контр.		Піх М.М.					
Затверд.		Бандура В. В.					

ВСТУП

Веб-розробка є однією з найдинамічніших галузей інформаційних технологій, яка постійно еволюціонує завдяки появі нових технологій, фреймворків і методологій. Сучасні веб-додатки вимагають високої продуктивності, інтерактивності та адаптивності, що зумовлює необхідність глибокого розуміння технологічного стеку, інструментів і процесів розробки. Технології, такі як HTML, CSS, JavaScript, а також фреймворки, такі як React, Angular чи Vue.js, стали основою для створення складних клієнтських і серверних рішень. Водночас інструменти для компіляції, лінтування, тестування та розгортання відіграють ключову роль у забезпеченні якості коду та ефективності розробки. У контексті швидкого зростання попиту на веб-додатки, здатні працювати в умовах високого навантаження та на різних пристроях, актуальність вивчення технологій і фреймворків веб-розробки є беззаперечною.

Актуальність теми зумовлена стрімким розвитком цифрової економіки, де веб-додатки є основним інструментом взаємодії між бізнесом і користувачами. За даними аналітичних звітів, таких як State of JavaScript, фреймворки та бібліотеки, що спрощують розробку, стають стандартом де-факто, тоді як інструменти автоматизації, такі як Webpack, ESLint і Jest, значно підвищують продуктивність команд розробників. Однак складність сучасних веб-проєктів вимагає системного підходу до вибору технологій, управління кодовою базою та оптимізації процесів розгортання. Виклики, пов’язані з кросбраузерною сумісністю, продуктивністю та масштабуванням, роблять цю тему важливою для досліджень і практичного застосування.

Метою цього дослідження є дослідити сучасні підходи до веб-розробки, охарактеризувати технологічний стек, середовища розробки, інструменти налагодження та тестування, а також оцінити їх вплив на якість та ефективність створення веб-продуктів.

Для досягнення поставленої мети в роботі необхідно вирішити **наступні завдання:** Проаналізувати основні складові технологічного стеку веб-розробки, розглянути сучасні методології та підходи до організації процесу розробки, дослідити популярні середовища, фреймворки та інструменти, що використовуються веб-розробниками, вивчити методи роботи з DOM та принципи модулярності у веб-додатках, оцінити роль компіляції, лінтингу, тестування та налагодження у процесі створення веб-продуктів, розглянути способи оптимізації та підготовки додатків до розгортання.

Об'єктом дослідження процес веб-розробки як складова сучасної IT-середовища.

Предмет дослідження: технології, фреймворки, інструменти та методи, що використовуються у веб-розробці на різних етапах створення веб-додатків.

Методи дослідження – Аналіз літературних джерел, порівняльний аналіз, систематизація та класифікація, практичне моделювання, аналіз ефективності.

Бакалаврська робота містить 70 сторінки, 34 рисунків, 6 таблиці, 3 розділи, список використаних джерел із 20 найменуванням.

РОЗДІЛ 1. ОСНОВИ ТЕХНОЛОГІЙ ТА ІНСТРУМЕНТІВ ВЕБ-РОЗРОБКИ

1.1 Основний стек технологій

По суті, веб-сторінка складається з мови гіпертекстової розмітки (HTML), каскадних таблиць стилів (CSS) і JavaScript. HTML визначає структуру веб-сторінки, CSS стилізує сторінку, а JavaScript забезпечує взаємодію користувача. Три з них разом вважаються веб-розробкою переднього плану [9]. Веб-сторінка, як випливає з її назви, – це сторінка з текстами та зображеннями, розміщена в Інтернеті. У традиційному розумінні веб-сторінка більше зосереджена на простому представленні інформації. Однак популяризація розумних портативних пристроїв, таких як смартфони та планшетні комп'ютери, назавжди змінила те, як користувачі сприймають і споживають цифровий контент. Завдяки набагато потужнішому апаратному забезпеченню, можливості браузера як на мобільних, так і на настільних платформах також змінилися відповідно.

У результаті сьогодні очікується, що веб-сайт не лише представлятиме інформацію, але й надаватиме користувачам адаптивний користувальницький інтерфейс, багаті функції, високу продуктивність і чудовий досвід користувача (UX). З огляду на все це складність середньостатистичного сучасного веб-проекту зросла, і традиційний спосіб розробки веб-сайту вже не є достатнім для підтримки його масштабу та масштабу. Таким чином, багато випробуваних концепцій, шаблонів і принципів більш традиційної інженерії програмного забезпечення, як-от визначення модульності, контролери представлення моделі (MVC), поділ проблем (SoC), об'єктно-орієнтоване програмування (OOP) і навіть статичне типування, потрапили у сферу веб-розробки в надії забезпечити чудовий досвід розробника (DX). Однак ці ідеї розробки програмного забезпечення часто несумісні з HTML, CSS і JavaScript. Крім того, швидкість, з якою робочі групи W3C і Ecma International випускають свою нову ітерацію

HTML, CSS і JavaScript, не встигають за швидким розвитком Інтернету. Отже, створено багато зовнішніх веб-бібліотек, фреймворків і утиліт, щоб допомогти розробникам модульувати, автоматизувати та тестувати свої проекти, але відслідковувати всі нові інструменти стає дедалі складніше [6].

Оскільки численні бібліотеки, фреймворки, утиліти тощо створено для вирішення фундаментальних проблем HTML, CSS і JavaScript, також створено експоненціальну кількість інструментів, щоб покращити досвід розробника під час створення зовнішнього веб-проекту. У цьому розділі будуть розглянуті проблеми, які лежать в сучасній веб-розробці, і їх вирішення. Крім того, у цьому розділі також описано інші інструменти, які зазвичай використовуються, і процедуру створення повної веб-програми. Спочатку для розробника, щоб писати рядки коду, дуже важливо отримати середовище розробки, і оскільки немає жодних автентичних організацій чи підприємств, які придумали офіційне середовище розробки для веб-розробників, щоб працювати з їх кодовою базою, тому багато редакторів було створено приватними компаніями чи окремими особами, щоб заповнити прогалину. Крім того, менеджер пакунків це інструмент, який допомагає відстежувати, яке програмне забезпечення встановлено на комп'ютері, і дозволяє розробникам легко виконувати необхідні дії. Щоб інсталювати менеджер пакетів node, інсталяція Node.js є обов'язковою.

Спочатку Node.js був розроблений як серверне середовище для програм, але розробники почали використовувати його для створення інструментів, які допомагають їм у локальній автоматизації завдань. Веб-індустрія постійно розвивається, проте браузери можуть не підтримувати новітні технології, якщо користувачі ще не оновили браузери. Це викликає іронію в тому, що нові функції, представлені в останній версії, значно покращують роботу розробників, але вони не підтримують програмне забезпечення користувачів. Отже, інструменти від джерела до джерела також винайдено для того, щоб компілювати коди з останньої версії, написаної програмістами, а потім виводити користувачам їх поточну версію. Існує багато типів компіляторів типу «джерело-до-джерела», які дозволяють вирішувати сумісність синтаксису

(Babel), систему введення (Typescript), препроцесор (Sass). Хороший проект складається з послідовних рядків коду. Сучасні лінтери та форматери допомагають скоротити розрив, визначаючи чіткий набір правил, які необхідні для всіх розробників, які працюють над цим проектом. Ці інструменти також допомагають розробникам писати кращий код, показуючи типові помилки та дотримуючись хороших шаблонів.

Форматувальник може забезпечувати узгоджену максимальну довжину рядків, гарантувати, що він не змішує одинарні та подвійні лапки, додавати кінцеву крапку з комою після кожного кінцевого рядка коду та допомагати з іншими проблемами форматування. Це особливо важливо під час роботи в команді розробників із різноманітним фоном Більш молодші розробники можуть прийняти загальні практики, дотримуючись правил, і кожен може бути впевнений, що код залишається послідовним, навіть якщо практики змінюються з часом. Наступним етапом процедури є тестування та налагодження. Це важливі процеси під час розробки та обслуговування програмного забезпечення. Тестування зосереджено на пошуку проблем, а об’єкт налагодження – на вирішенні проблеми. За допомогою налагодження розробники виділяють проблеми в додатку. Після того, як розробник виправить помилку, тестер повторно перевірить, щоб переконатися, що помилки/помилки буде усунено. До етапу, коли веб-проект можна буде розгорнути, буде враховано кілька оптимізацій, інакше це вплине на продуктивність продукту та, як наслідок, негативний досвід користувача. Враховуючи, що немає повноважень надавати офіційні інструменти розробки, для вирішення кожної частини головоломки потрібні різні інструменти сторонніх розробників. Хоча в процесі є ще більше процедур, наступні кроки є найбільш помітними та ключовими: мініфікація, стиснення, поділ коду та кешування. Підводячи підсумок, у цій дипломній роботі розглядаються деякі з найбільш поширених інтерфейсних веб-бібліотек, фреймворків і утиліт на ринку. Аналізуючи їхню історію, переваги та недоліки, ця теза повинна відповісти на запитання, чому

існує так багато бібліотек, фреймворків і утиліт у сфері веб-розробки інтерфейсу.

1.2 Методологія

У цьому розділі пояснюються методи, які використовуються в цій дипломній роботі для архівування її цілей шляхом аналізу проблем, які виникають під час розробки веб-сайту в стеку традиційних веб-технологій, і того, як деякі інструменти сторонніх розробників створюються для перемикання цих проблем. Питання спочатку будуть вивчені, класифіковані та розділені на категорії. На початку кожного аналізу конкретної категорії Дуонг Дінь надасть історію відповідного предмета, щоб забезпечити контекст, а потім Чжуаньян Ванг далі проаналізує проблеми, надавши тематичні дослідження. Після цього будуть обговорені альтернативні підходи, щоб дати ширше бачення предмета.

Враховуючи кількість тем, пов’язаних із інтерфейсною веб-розробкою, неможливо охопити їх усі. Ця робота буде зосереджена на обговоренні проблем і викликів, які виникають під час розробки веб-сайту з використанням лише Vanilla HTML, CSS і JavaScript. Розділивши їх на різні категорії залежно від того, до якої частини процесу розробки вони належать, роботаможе глибоко зануритися в конкретну проблему, не будучи приголомшливою. З точки зору класифікації проблем, роботапочинається з найбільш фундаментальних проблем HTML і CSS. Потім у роботі обговорюються проблеми з JavaScript з точки зору DOM і модульності. Нарешті, роботаохоплює середовище розробки зовнішнього веб-сайту.

Щоб зрозуміти, чому така велика кількість сторонніх інструментів існує у зовнішній веб-розробці, потрібно спочатку запитати про всі проблеми, для вирішення яких ці інструменти створені. Надаючи історичний контекст для цих

питань, читачі зможуть краще зрозуміти причини появи всіх бібліотек, фреймворків тощо.

Немає кращого способу провести поглиблений аналіз проблеми, ніж надати приклади. Для кожної проблеми спочатку буде надано коди, написані в HTML, CSS і JavaScript, щоб висвітлити проблему. Потім будуть надані фрагменти коду бібліотеки, фреймворку або утиліти, щоб продемонструвати, як вони підходять до проблеми. Автори повністю усвідомлюють масштаби веб-розробки. Для вирішення кожної проблеми HTML, CSS і JavaScript спільнота веб-розробників створить незліченну кількість рішень. Щоб визнати існування численних рішень, не вдаючись до надто широкого та загального, обговорюватимуться інші альтернативні підходи.

1.3 Інструменти та середовища розробки

Сучасна веб-розробка неможлива без ефективних інструментів і середовищ, які дозволяють прискорити процес створення, тестування та впровадження веб-додатків. Вибір правильного інструментарію допомагає знизити кількість помилок, покращити продуктивність і забезпечити спільну роботу в команді. Одним із таких інструментів є **системи контролю версій**, серед яких **Git** є найбільш популярним. Він дозволяє відслідковувати всі зміни в коді, що забезпечує зручність при роботі над великими проектами та дає змогу кільком розробникам одночасно працювати над різними частинами коду. Використання **GitHub**, **GitLab** чи **Bitbucket** як хмарних репозиторіїв додає можливість спільної роботи, автоматизації процесів за допомогою CI/CD та інтеграції з іншими інструментами.

Не менш важливим є вибір середовища для розробки, яке дозволяє зосередитися на програмуванні, зменшуючи потребу в налаштуванні. **Visual Studio Code** є одним із найпоширеніших редакторів завдяки своїй легкості, підтримці плагінів та інтеграції з іншими інструментами. Цей редактор підтримує мовні розширення для роботи з HTML, CSS, JavaScript, Python та

іншими мовами, що робить його універсальним для багатьох видів веб-розробки.

Для управління залежностями та інтеграції різних бібліотек використовуються **менеджери пакетів**, такі як **npm** для JavaScript або **Composer** для PHP. Ці інструменти автоматизують процес завантаження та оновлення бібліотек, зменшуючи час на налаштування та забезпечуючи зручний доступ до найбільш популярних фреймворків та пакетів.

У процесі розробки важливими є інструменти для тестування та налагодження. **Jest** і **Mocha** дозволяють виконувати юніт-тести, що допомагає перевірити функціональність коду на ранніх етапах розробки, запобігаючи виникненню помилок у процесі реалізації. Крім того, **Chrome DevTools** та інші браузерні інструменти допомагають відслідковувати продуктивність додатка, налагоджувати JavaScript та тестувати адаптивність веб-сторінок.

Окремо слід виділити інструменти для автоматизації зборки проектів, такі як **Webpack** або **Parcel**, які дозволяють зменшити час на обробку ресурсів, таких як JavaScript, CSS, зображення та шрифти, і забезпечують оптимізацію коду для швидшого завантаження сторінок. Завдяки цьому зменшується кількість HTTP-запитів і оптимізується завантаження веб-сторінок.

Насамкінець, **Docker** став важливим інструментом для розробки, особливо при роботі з контейнерами. Це дозволяє ізолювати додатки та їх залежності в окремі середовища, що робить процес розробки більш гнучким і переносним. Використання контейнеризації дозволяє зберігати однакові умови для запуску додатка як у розробницькому середовищі, так і на продакшн-серверах.

Таким чином, інструменти та середовища розробки є важливою частиною сучасного процесу веб-розробки, оскільки вони допомагають знизити час розробки, покращити продуктивність і забезпечити високий рівень якості кінцевого продукту.

1.4 Висновки до розділу

Сучасна веб-розробка є складним і динамічним процесом, що вимагає не лише знання базових технологій HTML, CSS і JavaScript, а й глибокого розуміння численних інструментів і фреймворків, які були створені для подолання їхніх обмежень. Зі зростанням вимог до функціональності, продуктивності та якості користувацького досвіду веб-проекти перетворилися з простих сторінок на масштабні та інтерактивні додатки. Це зумовило потребу в системному підході, заснованому на принципах традиційної розробки програмного забезпечення, таких як модульність, повторне використання коду, тестування та автоматизація.

Щоб відповідати цим викликам, розробники активно використовують цілий арсенал інструментів: редактори коду, менеджери пакетів, системи контролю версій, засоби тестування, збірки, лінтери, форматери, а також середовища віртуалізації та контейнеризації. Вони дозволяють не лише прискорити розробку, а й підтримувати високу якість коду, забезпечити командну роботу та стабільну інтеграцію змін. Таким чином, розуміння і вміння працювати з цими інструментами стало необхідною умовою ефективної веб-розробки.

Кількість і різноманітність бібліотек, фреймворків і утиліт у сфері веб-розробки зумовлена потребою в адаптації до стрімкого розвитку технологій і зростаючих вимог користувачів. Аналізуючи проблеми, пов’язані з традиційним стеком технологій, і шляхи їх вирішення, можна краще зрозуміти логіку еволюції сучасних інструментів та обрати найбільш доцільні з них для конкретного проекту. Це дозволяє забезпечити не лише якісний результат, але й покращити досвід розробника (DX), що у підсумку впливає на загальний успіх цифрового продукту.

РОЗДІЛ 2. ОСНОВИ ВЕБ-РОЗРОБКИ ТА ТЕХНОЛОГІЧНИЙ СТЕК

2.1 Аналіз первинного технологічного стеку

Аналіз основного стеку технологій глибоко занурюється в виклики та проблеми, пов’язані з розробкою веб-сайту на Vanilla HTML, CSS і JavaScript, спочатку розділивши розділ на три частини: повторення, керування DOM і модульність. Спочатку будуть представлені виклики, пов’язані з Vanilla HTML, CSS або JavaScript, після чого буде розглянуто тематичне дослідження з фрагментами коду, які демонструють, як конкретна технологія підходить до викликів. Наприкінці деяких розділів будуть обговорені альтернативні підходи, якщо такі є.

HTML розшифровується як Hypertext Markup Language, яка є найважливішим будівельним блоком Всесвітньої павутини (WWW). Спочатку він був розроблений для використання як мови розмітки для наукових документів, але його загальний дизайн дозволив прийняти його в інших галузях протягом багатьох років [12].

Проблеми

Хоча синтаксис HTML забезпечує інтуїтивно зрозумілий спосіб перевірити структуру та ієрархію елементів веб-сторінки, спосіб його використання передбачає, що кожен файл HTML обслуговується як окрема веб-сторінка [15]. Наприклад, мова відображення веб-сайту в більшості випадків є узгодженою на всіх сторінках, але оскільки один файл HTML представляє одну сторінку, значення властивості «lang» усередині тегів «<html>» повторюватиметься в кількох файлах HTML.

Як показано на рисунку 2.1, мова відображення може бути визначена у властивості 'lang', але якщо значення властивості 'lang' коли-небудь знадобиться змінити, тоді потрібно відвідати кожен файл HTML і змінити їх один за одним вручну. Крім того, значення таких властивостей, як «charset», «viewport», «theme-color» у всьому проекті, ймовірно, будуть однаковими, а отже, також

повторюватимуться.

Рисунок 2.1 - Мову відображення можна визначити у властивості «lang».

Рисунок 2.2 - Вміст усередині тегів '<head>' файлу HTML.

Як видно на рисунку 2.2, вміст тегів <head>, ймовірно, повторюватиметься в усіх файлах HTML. Таким чином, неважко зробити висновок, що зі збільшенням розміру проекту той самий рядок коду, який повторюватиметься у всіх HTML-файлах, також збільшується, а ремонтопридатність такої кодової бази буде погіршуватися.

Рішення та приклад:

React Щоб усунути проблему повторення, потрібно переконатися, що всі

параметри для веб-сайту, як-от «charset», «description» і «lang», потрібно визначати лише один раз в одному місці, забезпечуючи при цьому оновлення лише вмісту, який потрібно змінити, наприклад «тіло» HTML-файлу. React, наприклад, є однією з інтерфейсних веб-бібліотек, які пропонують вирішення цієї проблеми. React - це бібліотека JavaScript для створення інтерфейсів користувача [24] Його прототип FaxJS був спочатку створений Джорданом Уоком у Facebook [17] але згодом у 2013 році був відкритий вихідний код під назвою React [13].

Як показано на рисунку 2.3, у React є лише один файл HTML, і його зазвичай називають шаблоном HTML.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="favicon.ico" />
    <meta name="viewport" content=
"width=device-width, initial-scale=1.0" />
    <meta name="theme-color" content="#000000" />
    <meta name="description" content=
"Description of the project." />
    <link rel="apple-touch-icon" href="logo192.png" />
    <link rel="manifest" href="manifest.json" />
    <title>Demo</title>
  </head>

  <body>
    <noscript>
      <span>
        JavaScript is required. Please 
        <a href=
"https://duckduckgo.com/?q=how+to+enable+javascript"
          >enable JavaScript</a
        >.
      </span>
    </noscript>

    <div id="root">
      <!-- Content of the page will be placed here. -->
    </div>
  </body>
</html>
```

Рисунок 2.3 - Шаблон HTML, який використовує React.

Як React працює з HTML, це змінює лише ті частини, які потрібно оновити, залишаючи недоторканим інший вміст, як-от «<head>» і «<noscript>». Це означає, що користувачі завжди будуть на одній веб-сторінці. Коли користувачі запитують новий вміст (або іншу веб-сторінку в традиційному розумінні), замість завантаження повністю нової сторінки або HTML-файлу з сервера поточна сторінка оновлюватиметься динамічно на основі взаємодії користувача за допомогою JavaScript. Таким чином, повторення проблему з HTML вирішено. Оскільки цей метод став широко прийнятим у веб-бібліотеках і фреймворках усіх типів, був створений новий термін під назвою Single Page Application (SPA), щоб відобразити природу їх єдиного HTML-файлу [5].

Альтернативи React

Односторінкова програма працює всередині браузера і не вимагає перезавантаження сторінки під час використання. Натомість він дозволяє браузеру отримати весь необхідний код HTML, JavaScript і CSS одним завантаженням або лише отримує необхідні ресурси та додає їх на сторінку, як правило, для відповіді на дії кінцевого користувача. SPA запитують файл HTML і дані, що повертаються в незалежній дії, а потім відображають веб-сторінку в кінцевому користувачеві. Існує багато фреймворків JavaScript, таких як Angular, Vue.js, Svelte тощо, які також забезпечують ту саму інфраструктуру для досягнення тієї ж мети [23]. Хоча раніше згадані фреймворки дозволяють розробникам створювати SPA, вони мають різні підходи та принципи. Хоча раніше згадані фреймворки мають різні способи реалізації своєї кодової бази, усі вони мають схожість; усі вони містять суміш кодів JavaScript і HTML. З огляду на React, він зазвичай має один файл 'index.html' для з'єднання з 'index.js', а файл 'index.html' не має жодного вмісту, крім того, що містить необхідні теги '<metadata>' і '<script>'. Причина в тому, що в React HTML живе у файлі JavaScript. Те, як конструкції React дозволяють зберегти логіку та стиль в одному файлі. Це допомагає забезпечити чистішу та ефективнішу кодову базу. Однак з Angular відбувається протилежне: коди JavaScript живуть у файлі

HTML. Svelte і Vue також мають однаковий підхід, і всі вони досягають однієї мети: усунути

повторення.

Каскадні таблиці стилів (CSS) були вперше представлені в 1996 році. На той момент поточною версією HTML була HTML2. HTML відповідає за макет, зовнішній вигляд і введення інформації на сторінці. CSS відповідає за опис представлення документа, написаного мовою розмітки, наприклад HTML. Це фундаментальна технологія Всесвітньої павутини, поряд з HTML і JavaScript (CSS WG, 2020).

CSS було створено, щоб надати веб-дизайнерам інструмент для керування стилем елементів HTML. Крім того, він не призначений для керування візуальним виглядом веб-сторінок; він був призначений лише для доступу та контролю властивостей окремих елементів, які існували у файлі HTML. Під час створення CSS більшість веб-сайтів виглядали простими, і архітектори не могли передбачити багатство та складність дизайну поточного стану веб-сторінок у наш час. Причину цього чітко вказав один із творців CSS: «Змінні, константи, умови, вирази над змінними тощо – це функції, які найчастіше використовуються програмістами, однак вони не існують у CSS. Ці речі дають велику силу розробникам, оскільки вони знають, як налаштувати та використовувати ці функції якнайкраще. Однак недосвідчені користувачі відчуватимуть більший тиск, швидше за все, злякаються та перестануть використовувати CSS» (Bos, nd).

Проблеми

Початкові творці CSS стурбовані усиновленням. CSS був розроблений як достатньо потужний для стилізації веб-сторінок і відокремлення структури від візуального, при цьому простий для розуміння та використання. Тим не менш, він все ще вкорінений як мова стилів, а не мова розмітки. Він пропонував розробникам простий спосіб керування шрифтами, кольорами та розмірами в одному файлі. І все ж, коли справа доходить до стилізації макета

веб-сторінки, CSS не може впоратися з цим належним чином до виходу версії 2.1 [14].

Як видно на рисунку 2.4, є два класи: '.class-a' і '.class-b'. Ці два класи мають однаковий колір і ширину, але мають різне розташування. Наразі коди кольору та ширини однакові, тому вони повторюються. Якщо потрібно змінити колір або ширину теми, єдиний спосіб - знайти всі «помаранчеві» та «128 пікселів», а потім змінити їх вручну або скористатися вбудованими функціями, як-от пошук і заміна з редакторів. Без використання сторонніх інструментів один із способів усунути цю проблему в CSS – згрупувати елементи, які мають однакові значення властивостей.

Рисунок 2.4 - Два класи, які мають однакові значення для деяких властивостей.

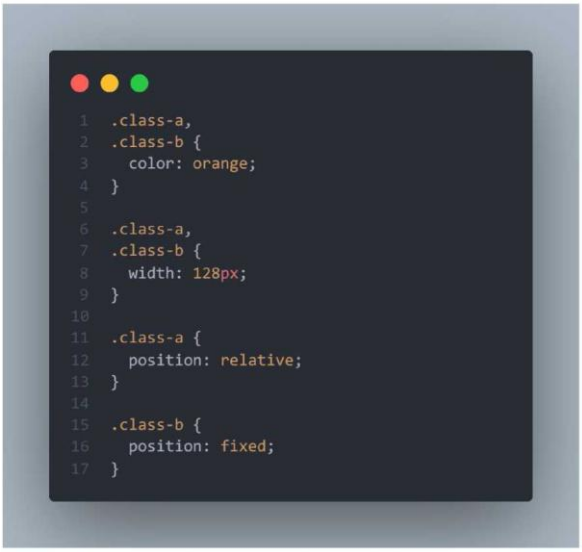


Рисунок 2.5 - Групування класів, які мають однакове значення властивості.

Хоча підхід на рисунку 2.5 робить коди менш повторюваними, він серйозно погіршує читабельність і зручність обслуговування кодової бази. Маючи кілька '.class-a' і '.class-b', тепер також набагато важче відслідковувати, який клас має яку властивість. Якщо в майбутньому буде додано більше елементів і властивостей, ситуація ще більше погіршиться.

Рішення та приклад: Sass

Щоб вирішити цю проблему, найбільш очевидним підходом є введення концепції змінних і функцій у CSS, а також концепції препроцесорів CSS [3]. Препроцесор CSS — це програма, яка дозволяє генерувати CSS на основі унікального синтаксису препроцесора та надавати функції, які зараз недоступні в CSS. Sass означає Syntactically Awesome Stylesheets, це препроцесор CSS, який дозволяє розробникам створювати необхідні рядки коду для написання коду CSS. Це також дозволяє розробникам миттєво вносити зміни, оскільки їм не потрібно змінювати дубльовані селектори, як це робив CSS. Sass є одним із найпоширеніших препроцесорів CSS у сучасних веб-додатках [8]. Препроцесор CSS допомагає розробникам писати більш зрозумілий і простий для розуміння код. Sass отримує дані з файлів «.scss» або «.sass», які пізніше компілюються у звичайні файли «.css». Нещодавно скомпільовані файли CSS розгортаються у браузері для стилізації веб-програми. Хоча CSS стає надійнішим, поступово

охоплюючи функції, які наразі доступні в препроцесорах (змінна вже доступна в CSS 3.0), він вимагає певних функцій, таких як повторне використання коду, що ускладнює «підтримку коду», коли проекти стають більш значними та складними. Щоб досягти того самого результату на рисунку 5 без недоліку гіршої читабельності, Sass пропонує просте рішення.

Як видно на рисунку 2.6, тепер, якщо розробник хоче шукати '.class-a' або '.class-b', назва класу з'явиться лише один раз. Так само, якщо потрібно змінити колір теми, потрібно лише змінити значення змінної '$color-theme' без необхідності переглядати всі елементи, які мають колір теми, і змінювати їх один за одним. Що, якщо блок кодів CSS спільний для багатьох елементів? Наприклад, що, якщо два елементи з різними класами мають однакові налаштування для їх позиціонування? SCSS також може легко впоратися з такими ситуаціями.

Рисунок 2.6 - Sass дозволяє використовувати змінні в CSS.

Як показано на рисунку 2.7, «mixin» SCSS забезпечує спосіб легкого повторного використання блоку кодів CSS за допомогою синтаксису «@mixin» і «@include». За допомогою змінних і «mixin» більшість проблем із повторенням CSS вирішується.

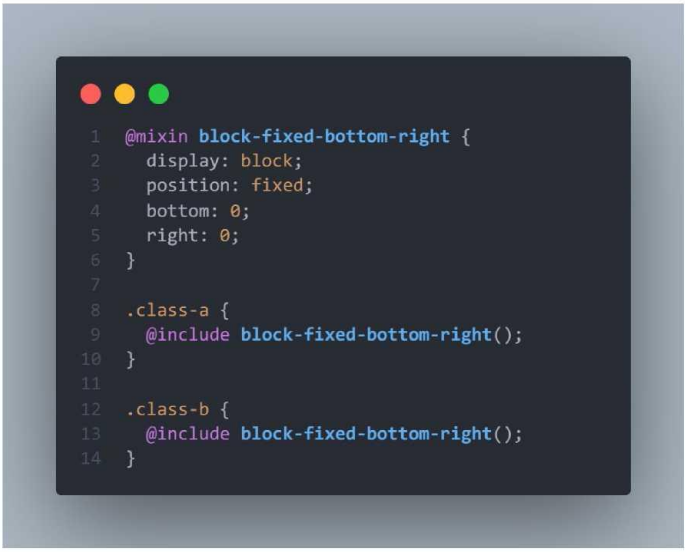


Рисунок 2.7 - SCSS дозволяє розробникам повторно використовувати блок кодів CSS.

Альтернативи Sass

Що стосується альтернатив для Sass, існує багато доступних препроцесорів CSS, які допомагають розробникам досягти тієї самої мети. Less.js (зазвичай його називають простішою формою Less), Stylus, PostCSS, Styled Components тощо. Хоча Sass і Stylus є найбільш поширеними та універсальними, все ж є альтернативи, які слід розглянути. Ці препроцесори CSS мають спільні риси, але деякі з них відрізняються. Більшість препроцесорів підтримуватимуть змінні, вкладення, успадкування, умови, цикли, функції, «@mixin» і «@include». Ці інноваційні функції роблять препроцесори CSS більш об’єктно-орієнтованими та дозволяють програмістам ефективно виконувати свої ідеї. Хоча кожна із зазначених альтернатив має свої переваги та недоліки. Розробники повинні вибрати найбільш підходящий для кожного проекту. Пізніше в розділі «Модуляризація» буде надано практичне дослідження, щоб дослідити, як SCSS може покращити та вдосконалити CSS.

2.2 Управління DOM

DOM - це акронім Document Object Model, це API для документів HTML

і XML. DOM представляє структуру веб-документів і спосіб доступу до веб-сторінки та керування ними. За допомогою об’єктної моделі документа розробники можуть маніпулювати документом, а також змінювати структуру. Будь-що, що міститься у файлі HTML або XML, можна отримати, змінити, видалити чи додати за допомогою об’єктної моделі документа. Все в DOM також представляє вузол. Документ є основним вузлом. Усі елементи HTML представляють вузли елементів тощо (Robie, 2020).

Як показано на рисунку 2.8, DOM формується як деревоподібна структура, що допускається структурою документа HTML. Таким чином, це дозволяє легше обходити дерева та між вузлами.

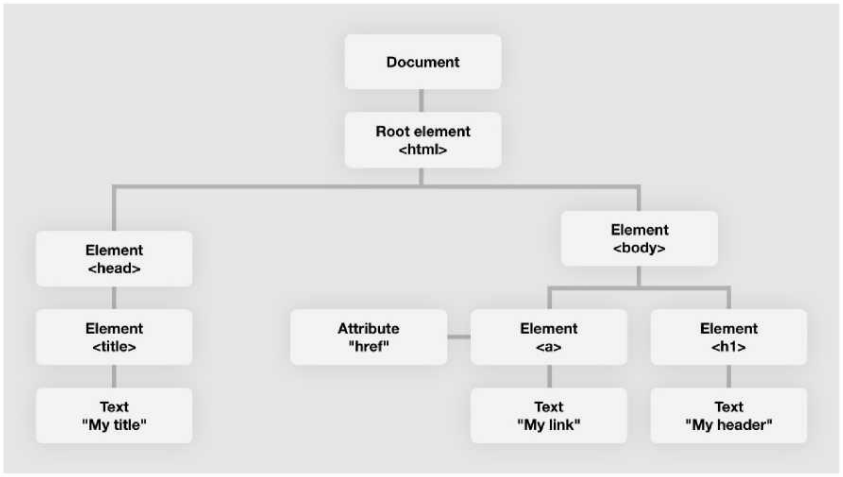


Рисунок 2.8 -Деревоподібна структура об’єктної моделі документа.

Проблеми

У наш час у будь-якій веб-програмі дерева DOM величезні та складні. Оскільки розробники все більше наполягають на динамічних веб-додатках (SPA), тому дерево DOM безупинно вносить багато модифікацій. Це погано з точки зору продуктивності та розвитку. Якщо на веб-сторінці є список із кількох елементів і під час будь-якої взаємодії користувача, він має оновлювати кожен із них, весь реальний DOM буде повторно відтворено. Це в рази більше зусиль, ніж вимагається. Це не лише впливає на будь-яку іншу поточну взаємодію користувача на веб-сторінці, але також має недоліки щодо пропускної здатності. Враховуючи, що DOM складається із сотень елементів

div. І є багато методів, які обробляють події: клацання, надсилання, введення тощо. Типовий обробник подій JavaScript спочатку націлюється на кожен необхідний вузол у події, а потім оновлює його, якщо потрібно. Раніше це був метод доступу до DOM, але це породжує дві проблеми:

- Важко підтримувати. Наприклад, якщо є обробник події, який потрібно змінити. Якщо розробники, які займаються обслуговуванням, втратили контекст, їм доведеться глибоко зануритися в кодову базу, щоб навіть зрозуміти, що ці рядки коду мали бути. І ризиковано, і займає багато часу.
- Це неефективно. Такі дії вимагають більше ручної роботи. Те, як це повинно робитися, - це заздалегідь повідомляти, які вузли потрібно оновити.

Приклад і рішення: віртуальний DOM У випадку, якщо будь-яка програма, яка виконує багато таких операцій у фоновому режимі, відображатиметься погано. Для таких веб-сайтів, як соціальні мережі, яким потрібно часто отримувати та надсилати дані, DOM повинні мати можливість надзвичайно швидко оновлюватися, щоб дані, що відображаються в елементах інтерфейсу користувача, синхронізувалися з базою даних. Наприклад, використовуючи мобільний сайт Twitter або Quora, коли користувачі прокручують сторінку вниз, їм буде показана кнопка «показати новіші канали». Після того, як користувач почне прокручувати деякий час, у нього будуть додані тисячі вузлів до DOM. Ефективна взаємодія з цими вузлами є величезною проблемою. Наприклад, якщо хтось намагається перемістити 1000 'div' на 5 пікселів вліво, це, ймовірно, займе більше секунди. Зважаючи на продуктивність, це багато часу для сучасного веб-додатку. Розробники можуть оптимізувати сценарій і застосувати деякі хитрощі, але в кінцевому підсумку і в довгостроковій перспективі працювати з величезними сторінками та динамічним інтерфейсом користувача буде просто безлад. Подібність видно на Facebook, реклама, стрічки новин, історії тощо змінюються з часом, коментарі оновлюються майже миттєво, щоб задовольнити взаємодію між усіма користувачами. Зайве говорити, що за сценою виконується безліч операцій DOM. Незважаючи на те, що сама DOM принесла багато проблем, у

розробників також є методи вирішення проблеми. Зараз група W3C працює над тим, що називається Shadow DOM. Тіньовий DOM є робочим стандартом W3C. Ця специфікація створює спосіб об'єднання багатьох деревовидних структур DOM в одну ієрархію та керування тим, як ці дерева взаємодіють одне з одним у документі, отже, це забезпечує кращу композицію DOM. Інший варіант - віртуальний DOM. Віртуальний DOM є абстрактним поняттям DOM. Віртуальні рішення DOM побудовані на основі стандартного DOM. Зрештою вони все ще використовують DOM, але роблять це якомога рідше та дуже ефективно. Він легкий, і для простоти визначення Virtual DOM є локальною та спрощеною версією React фактичного HTML DOM. На відміну від DOM і тіньового DOM, віртуальний DOM не є офіційною специфікацією, а новим методом зв'язку з DOM.

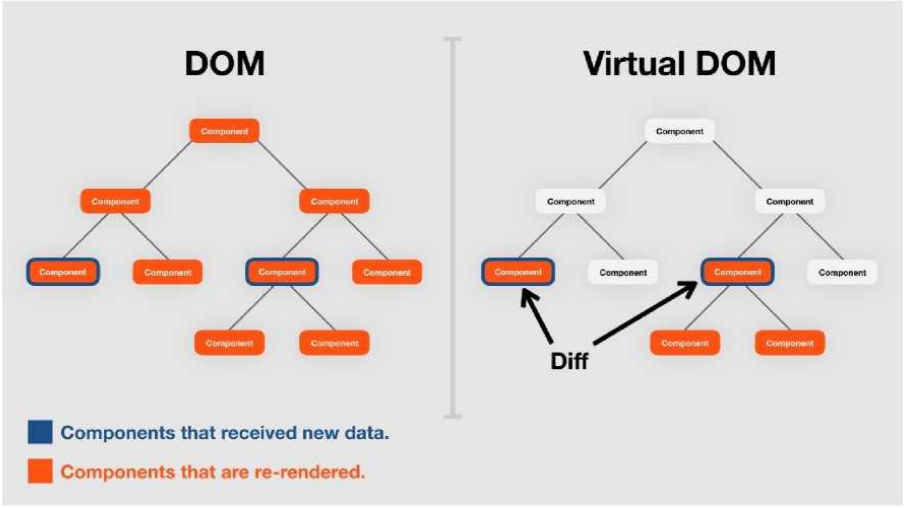


Рисунок 2.9 - Як DOM і Virtual DOM повторно візуалізують дерево компонентів.

На рисунку 2.9 показано різницю між тим, як DOM і Virtual DOM повторно відтворюють оновлені компоненти. Замість повторного відтворення всього дерева DOM, коли компоненти отримати нові дані, Virtual DOM повторно візуалізує лише компоненти, які потрібно оновити. Коли сторінка відображається за допомогою Virtual DOM, стан структури/ієрархії дерева DOM зберігається, і коли потрібно внести будь-які оновлення в інтерфейс

користувача, замість того, щоб писати ціле нове дерево, він виконує алгоритм «відмінювання». Бібліотека React наразі використовує Virtual DOM, що дозволяє їй виконувати обчислення в цьому домені та пропускати реальні операції DOM. Кожна частина інтерфейсу є компонентом у React, і кожен компонент має стан. React.js спочатку прослуховуватиме та спостерігатиме за змінами стану. Коли компонент змінює свій стан, React оновлює дерево Virtual DOM, а потім порівнює поточну версію з попередньою версією Virtual DOM. Цей процес порівняння застосовує алгоритм «різниці». Таким чином зменшується кількість операцій/оновлень DOM, що значно покращує продуктивність. Весь процес називається узгодженням [4]/

Альтернативи. Virtual DOM Існує два типи DOM: віртуальний DOM і справжній DOM. Існує багато інтерфейсних фреймворків, які використовують ці два підходи як свій принцип. React застосовує концепцію Virtual DOM для оновлення програми, тоді як Angular виконує це інакше, використовуючи реальну DOM. Кожен компонент в Angular має відповідний детектор змін, який генерується під час запуску програми. Коли викликається виявлення змін, для кожної функції, що використовується в компоненті, порівнюється різниця між поточним і попереднім значенням. Справжній DOM буде оновлено, якщо буде знайдена різниця. На даний момент Vue і React є двома добре відомими фреймворками, які застосовують той самий процес візуалізації під назвою Virtual DOM. З іншого боку, Angular і Svelte поділяють ту саму філософську техніку, вони змінюють дерево DOM, звертаючись безпосередньо до самого DOM.

2.3 Модулярність

Модулярність - це процес поділу комп’ютерної програми на менші окремі підпрограми. Модуль – це окрема програмна складова. Його часто можна використовувати в різноманітних програмах і функціях з іншими частинами системи. Пов’язані функції класифікуються в одній одиниці

програмного коду, а непов’язані функції розробляються як окремі одиниці коду, щоб код можна було повторно використовувати іншими програмами.

ECMAScript є стандартом для мов сценаріїв, а JavaScript є реалізацією ECMAScript. Акронім ES означає ECMAScript. ECMA - абревіатура від European Computer Manufacturer's Association; однак технічно ES і JavaScript - це практично одне й те саме, за винятком того, що JavaScript може надавати деякі додаткові функції (Ecma International, nd). JavaScript почався як клієнтська мова сценаріїв для розробки веб-додатків. На сьогодні ES опублікував десять версій. Перші три версії - ES1, ES2, ES3 були щорічними оновленнями, тоді як ES4 так і не була випущена через політичні розбіжності. Однак його попередня версія – ES5 містила обмеження, які не дозволяли розробникам створювати класи, імпортувати модулі тощо. ES5 називають першою стандартизованою версією JavaScript. Однак ES5 не підтримував модулі. Пізніша версія була представлена як ES6/ECMAScript - 2015/ES2015, це було перше оновлення мови з 2009 року. З випуском цієї версії JavaScript досяг великого успіху, надавши більше синтаксису та зменшивши навантаження для розробників, і частково виправдав очікування сучасної мови програмування. Останні версії JavaScript: ES7, ES8, ES9, ES10 також принесли оновлення мови програмування, однак, для ясності та обсягу теми, ця частина не включатиме їх тут. Щоб зрозуміти важливість цього, потрібне повне розуміння потреби в модулях і того, що вони забезпечують (Ecma International, nd).

Проблеми. JavaScript розглядався як мова для створення невеликих веб-сайтів; однак у наш час веб-сайти еволюціонують до веб-додатків; база коду та функціональність стають величезними. Розробники вимагали певного способу розподілу функціональності та одночасного визначення залежностей. Мови програмування, такі як C++ і Java, покладаються на зовнішні бібліотеки, наприклад, математичні бібліотеки, щоб виконувати обчислення за допомогою базових математичних рівнянь, не перевизначаючи та не переписуючи їх. Відповідно, бібліотеки дозволяють мовам програмування не копіювати певні коди. Подібність проявляється в тому, які модулі вносять у JavaScript. Модулі

дозволяють програмістам розділяти функціональні можливості та точніше організовувати кодову базу. Однією з найбільших переваг модулів є те, що глобальний простір імен JavaScript, безсумнівно, легко забруднити. У попередній версії функції були єдиним способом у JavaScript, який генерує область видимості, тому все, що не оголошено всередині функції, належить до глобального простору імен.

Отже, кожна функція, оголошена в JavaScript, доступна глобально, тому її ім'я не можна оголошувати повторно. Як наслідок, це призведе до забруднення простору імен, де імена та код буде важко знайти та повторно використати. Однак глобальна доступність функції не завжди бажана. Модуль з'являється на основі ідеї зробити функції у файлі JavaScript доступними для інших файлів JavaScript і навпаки. Модулі дозволяють розробникам розділяти функціональні можливості в програмах, це допомагає програмістам ефективніше керувати залежностями. Це забезпечує повторне використання коду, рефакторинг коду та, крім того, розширюваність коду. ES5 є основою кожної веб-програми та підтримується майже всіма поточними браузерами. Недоліком є те, що, як було зазначено раніше, ES5 не підтримує модулі. З цієї причини було запропоновано різні обхідні шляхи для підтримки модульності. У Vanilla ES5 є два загальних шаблони для реалізації модулів: шаблон Singleton (IIFE - негайно викликані вирази функції) і шаблон конструктора. Крім Vanilla ES5, замість того, щоб покладатися на нього, сторонні інструменти принесли переваги, яких ES5 не міг. ES6 додав різні функції та усунув обмеження ES5. Наразі ES6 має синтаксис «класу», що робить об'єктно-орієнтоване програмування простішим у реалізації. «Let» і «const» - нові ключові слова, які замінять «var». "Let" можна перепризначити, а "const" можна призначити лише одному значенню один раз. І, нарешті, ES6 додала підтримку модулів. Хоча ES6 має деякі нові функції, багато з них несумісні з більшістю сучасних веб-браузерів.

Тому розробники винайшли методи надання функцій, які вони бажали: створюються інструменти, так звані «trans-pilers», щоб компілювати версії коду

ES6 у старіші версії, такі як ES5, щоб браузери могли зрозуміти цей новостворений синтаксис. Більшість сучасних веб-додатків використовують ці компілятори, щоб код був сумісний із ES6, коли він буде випущений, але також сумісний із поточними веб-браузерами. Як і в інших випадках використання модулів, ES6 надає можливість експортувати модулі, щоб їх можна було вказати як залежності від інших модулів. З огляду на традиційний веб-проект, який використовує лише Vanilla HTML, CSS і JavaScript, усі файли зазвичай розміщуються у відповідній папці, названій відповідно до їхнього формату, але немає офіційної особи, яка б надала офіційні вказівки щодо структури проекту веб-розробки, подібної до тієї, що надається мобільними або настільними платформами, коли розробка програмних додатків для них. Це призводить до відсутності конвенцій і хороших практик, коли мова заходить про структурування веб-проекту. Структура традиційного веб-проекту різна, і відсутність узгодженості погіршує ситуацію. Це призводить до того, що різні групи розробників по-різному структурують веб-проект. Для нових членів команди потрібно багато часу, щоб адаптуватися до способу роботи в команді.

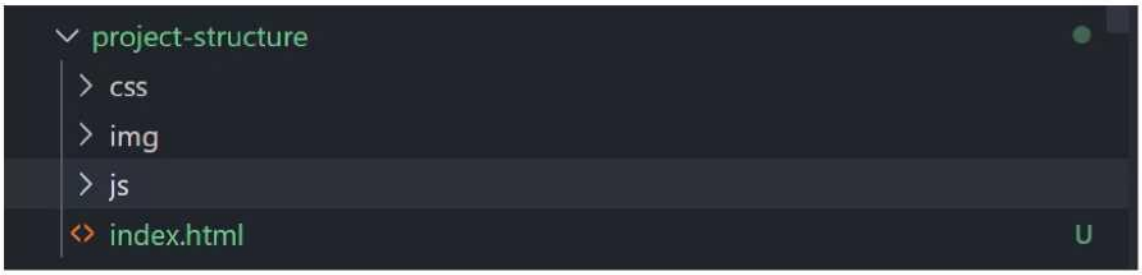


Рисунок 2.10 - Групування файлів за їх форматами.

Як показано на рисунку 2.10, один з поширених підходів до модульної структури веб-проекту полягає в тому, щоб розмістити файли, які мають однаковий формат, у відповідному каталозі.

Рішення та приклад. React представлено в прикладі того, як React вирішує проблему з повторюваними кодами HTML. У цьому розділі буде наведено більше прикладів, щоб продемонструвати, як React організовує та

модульує зовнішній веб-проект. Порівняно з традиційним способом організації файлів на основі їхніх форматів, React.js пропонує більш розумний підхід. У React усі елементи веб-сторінки розбиваються на компоненти, які динамічно оновлюються на основі їх стану. Коди для кожного компонента розміщуватимуться у відповідному файлі чи папці, а компоненти впорядковано на основі їхніх батьківських і дочірніх зв'язків.

На рисунку 2.11 у кореневому каталозі проекту є дві папки «public» і «src». Перша містить шаблон HTML та інші нединамічні ресурси, такі як піктограми, а друга містить усі вихідні коди та ресурси, які будуть імпортовані вихідними кодами. У корені каталогу 'src' можна знайти файл входу 'index.js',

який містить усі компоненти. У папці «components» «App.js» є батьківським для всіх дочірніх компонентів, а його дочірній компонент «FriendList» має папку з відповідною назвою. Що стосується «Friend», дочірній компонент «FriendList» розміщується поруч із «index.js» у каталозі «FriendList».

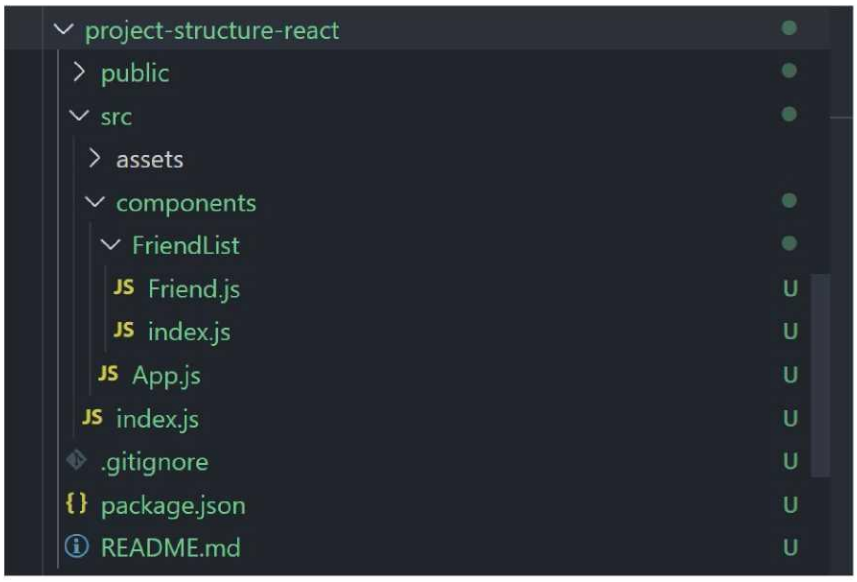


Рисунок 2.11 - Структура типового проекту React.

Наслідки прикладу. React Оскільки всі розробники, які навчалися писати в React, вивчать цю філософію структурування проекту, будь-яким командам веб-розробників, які використовують React, також легше найняти нового члена. До епохи веб-бібліотек і фреймворків веб-розробник не знає, чого

очікувати, подаючись на таку посаду, оскільки веб-проект можуть бути структуровані та організовані різними способами. У деяких випадках файли можуть бути розміщені в папках, які відповідають їхньому формату, в інших випадках команда проекту могла розробити свою внутрішню структуру для модульної побудови проекту, оскільки філософія методів модульної побудови веб-бібліотек і фреймворків часто універсальна (наприклад, розбиття елементів інтерфейсу користувача на компоненти). Таким чином, неважко зробити висновок, що нещодавно приєднані члени команди веб-розробників потребуватимуть додаткового часу в минулому, щоб ознайомитися з проектом на відміну від теперішнього часу. Якщо команда використовує бібліотеки, такі як React, структура проекту набагато більш передбачувана, як шаблон HTML, і статичні ресурси будуть розміщені в каталозі «public», а всі вихідні коди та динамічні ресурси будуть у каталозі «src» тощо.

Альтернативи React . Окрім React, те, як інші бібліотеки та фреймворки модульують зовнішній веб-проект, має багато спільного з наведеним вище прикладом. Наприклад, у бібліотеках і фреймворках, таких як Vue, Svelte, елементи інтерфейсу також розбиваються на компоненти (You, 2019). Подібним чином ці підходи сприяють певному способу організації файлів як найкращій практиці, щоб забезпечити керівництво для тих, хто їх прийняв, але вони не змушують розробників сліпо дотримуватися його. Таким чином, замість того, щоб завжди групувати файли, які пов’язані з одним компонентом, в одному каталозі, також прийнято розміщувати файли, які відповідають одному маршруту чи функції, в одній папці.

Принцип поділу проблем (SoC) для поділу комп’ютерної програми на окремі розділи, щоб кожен розділ розглядав окрему проблему (Elliott, 2014). Розподіл завдань визначається так, що кожен модуль або рівень на веб-сайті чи програмі має відповідати лише за певну річ і, таким чином, не повинен містити кодів, які обробляють інші функції. Розділення проблем зменшує складність коду, розбиваючи велику програму на менші одиниці інкапсульованої функціональності.

Проблеми. Розділення презентації та логіки під час створення веб-додатків дозволяє розробникам легше та швидше створювати веб-сайти з динамічним вмістом. Це також дозволяє веб-дизайнерам, які не мають особливого досвіду в розробці програм, легко маніпулювати зовнішнім виглядом веб-сайту. Для веб-сайтів із вмістом, який потребує регулярних змін, перевага означає, що оновлення можна досягти набагато швидше, тому вміст буде надано відвідувачам веб-сайту швидше. Ранні веб-додатки були нескладними і часто містили як презентацію, так і бізнес-логіку. Це може призвести до проблем з техобслуговуванням у разі внесення змін до будь-якого з них. Розділення двох компонентів спрощує технічне обслуговування та дозволяє швидше та легше оновлювати.

Рішення та приклад: Redux Проста архітектура програми в контексті невеликої демонстрації покаже, як досягти цього розділення та швидше розгортати та змінювати веб-програми.

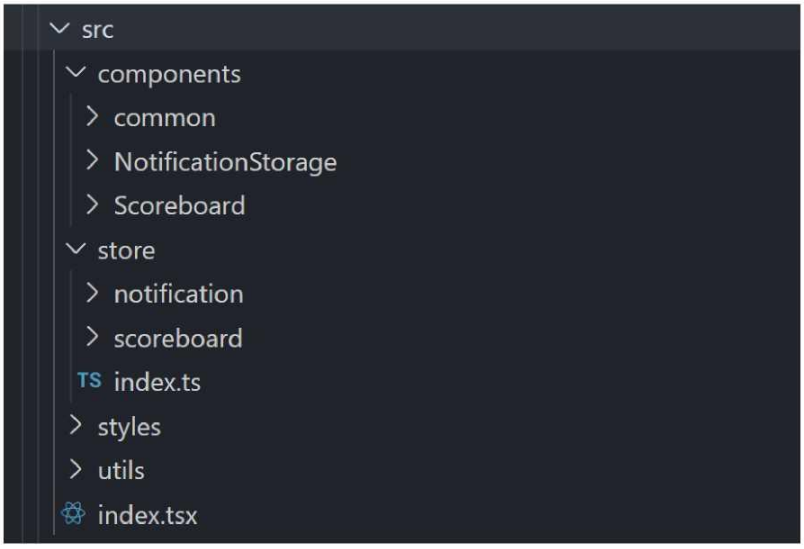


Рисунок 2.12 - Приклад структури проекту React-Redux.

Як видно на рисунку 2.12, у типовому проекті React Redux структура проекту здебільшого така ж, як і в проекті React. Різниця полягає в тому, що в каталозі «src» бізнес-логіка, яка зазвичай включає коди, що стосуються того, як дані витягуються та маніпулюють, розміщена в каталозі «store», тоді як

компоненти в каталозі «components» тепер відповідають лише за представлення даних у візуально привабливий спосіб. У такому проекті бізнес-логіка обробляється Redux, а презентації – React. Використання React разом із Redux почне показувати свої переваги, коли проекти збільшаться в розмірі, оскільки різні частини проекту будуть оброблятися розробником, який спеціалізується на певній галузі. Наприклад, оскільки компоненти React тепер стурбовані лише тим, як представлені елементи інтерфейсу користувача, тому розробники, які спеціалізуються на дизайні та візуальному аспекті Інтернету, можуть працювати над компонентами React, не турбуючись про те, як керуються станами та як обробляються дані.

Альтернативи Redux

Подібна концепція вже була широко прийнята в розробці програмного забезпечення, і вона називається парадигмою). Бізнес-логіка часто розбивається на два компоненти в парадигмі модель-представлення-контролер (MVC) - модель, яка визначає, як дані будуть зберігатися та витягуватися; і контролер, який керує взаємодією між моделлю та представленням. Для простоти та практичності роботане буде фокусуватися на цьому розрізненні. У багатьох випадках розрізнення непотрібне, оскільки шаблон потоку керування моделі та контролера надзвичайно подібний і задуманий одночасно. Логіка представлення означає, як вона відображає ці об’єкти користувачеві. Це часто називають представленням у парадигмі MVC. Часто важко визначити, де закінчується бізнес-логіка і починається презентаційна логіка. Те, як розробники моделюють свою програму, часто визначає, що можливо в інтерфейсі користувача. Іноді бізнес-логіка та логіка презентації настільки заплутані й залежать одна від одної, що вони переплітаються одна з одною. Іноді зазвичай складно або іноді неможливо чітко розділити ці два аспекти. У більшості випадків розділення цих двох — це лише питання дисципліни.

2.4 Висновки до розділу

Перехід до сучасних бібліотек і препроцесорів є логічним кроком для підвищення ефективності розробки.

Управління DOM є критично важливим аспектом продуктивності веб-додатків; завдяки використанню Virtual DOM можна значно зменшити кількість оновлень справжнього DOM, що підвищує швидкість та ефективність роботи інтерфейсу.

Текст добре описує розвиток модулярності у JavaScript та впровадження структурованих підходів у сучасних фреймворках, зокрема React і Redux. Він охоплює як історичні аспекти, так і практичні приклади організації коду, що демонструє еволюцію від ES5 до ES6+ та важливість поділу логіки й представлення.

РОЗДІЛ 3. ІНСТРУМЕНТИ ТА ПРОЦЕСИ ВЕБ-РОЗРОБКИ

3.1 Середовище розробки

У розробці програмного забезпечення середовище розробки - це сукупність процесів і інструментів, які використовуються для розробки вихідного коду або програми. Цей термін використовується як синонім інтегрованого середовища розробки (IDE), яке є інструментом розробки програмного забезпечення, який використовується для написання, створення, тестування та налагодження програми. Він також надає розробникам загальний інтерфейс користувача (UI) для розробки та налагодження в різних режимах. Загально кажучи, термін «середовище розробки» застосовуватиметься до всього середовища, тоді як IDE стосується лише локальної програми, яку розробники використовують для кодування. Існує багато збігів, оскільки вони використовують IDE для налагодження так само, як використовують сервер розробки для тестування.

Проблеми

Ключові технології веб-розробки переднього плану не мають належного середовища розробки з самого початку, оскільки HTML є мовою розмітки, CSS є мовою таблиць стилів, а JavaScript є мовою сценаріїв. Це означає, що, незважаючи на те, що браузерам потрібно скомпілювати та виконати їх під капотом, час, необхідний настільки мінімальний, що вони більше схожі на роботу в реальному часі з практичної точки зору. Отже, порівняно з традиційною скомпільованою мовою програмування, як-от C, їх не потрібно компілювати та створювати в середовищі розробки перед розгортанням на веб-сервері чи браузері. Що ще гірше, на відміну від розробки власних програмних додатків для настільних комп’ютерів і мобільних платформ, таких як Windows або Android, не існує авторитетної особи, яка б надала офіційне середовище розробки. Наприклад, C# в основному використовується для платформи Windows, яка підтримується Microsoft, тому Microsoft надає офіційну IDE під

назвою Visual Studio, тоді як Objective-C або Swift мають офіційну IDE під назвою Xcode, яка підтримується та розробляється Apple.

Рішення: інструменти сторонніх виробників Таким чином, багато приватних компаній або індивідуальних забудовників взяли справу в свої руки. Багато редакторів, як-от Visual Studio Code, Atom і Subline Text, створені, щоб заповнити цю порожнечу, хоча технічно вони не є IDE, але за допомогою різноманітних розширень і плагінів вони функціонально схожі на традиційні IDE, такі як Visual Studio або Eclipse. Однак навіть із такими інструментами, як Visual Studio Code, багато важливих компонентів усього процесу розробки все ще відсутні, наприклад менеджер залежностей.

Проблеми Щоб налаштувати середовище розробки, якого немає в шаблоні Vanilla HTML, CSS і JavaScript, потрібен інструмент керування, який дозволить розробникам швидко виконувати базову взаємодію з усіма залежностями. Незважаючи на те, що менеджер пакунків із графічним інтерфейсом забезпечить більшу зручність для користувача, для його створення та підтримки знадобиться набагато більше зусиль, часу та ресурсів, отже менеджер пакунків, який керується за допомогою командних рядків, є раціональним вибором. Оскільки жоден із HTML, CSS або JavaScript не може працювати в середовищі командного рядка, потрібно щось заповнити цю прогалину.

Рішення: Node.js і NPM Node.js дозволяє JavaScript запускати в командному рядку, а пізніше Node Package Manager (NPM) стає одним із найпоширеніших менеджерів залежностей. Node.js дає можливість писати програми на JavaScript на сервері. Він побудований на середовищі виконання V8 JavaScript і написаний на C++ - тому швидкість забезпечена. Спочатку він був призначений як серверне середовище для додатків, однак розробники почали використовувати його для створення інструментів, які допомагають їм у локальній автоматизації завдань. З тих пір ціла нова екосистема ¬інструментів на основі Node, таких як Grunt, Gulp і Webpack, розвинулася, щоб змінити вигляд інтерфейсної розробки. Щоб використовувати ці інструменти (або

пакети) у Node.js, необхідно вміти встановлювати їх і керувати ними у корисний спосіб. Ось де NPM стає корисним. Він встановлює пакети, які хочуть використовувати розробники, і надає інтерфейс користувача для роботи з ними. Ці приклади продемонструють, як працює NPM і його «package.json».

Рисунок 3.1 - Пакет 'package.json', згенерований командою 'create-react-app'.

Як показано на рисунку 3.1, проект, який використовує Node.js, потребує файл 'package.json', який зазвичай генерується бібліотекою або фреймворком. У найпростішому випадку, 'package.json' файл можна представити як маніфест проекту, який містить усі необхідні пакети та програми, інформацію про

джерело керування та метадані, такі як назва проекту, опис, автор тощо. Усередині «package.json» метадані майже завжди містяться специфічні для проекту - незалежно від того, чи це веб-програма, модуль Node.js чи навіть проста бібліотека JavaScript. Ці метадані забезпечують ідентифікацію проекту та служать базовою лінією для користувачів і учасників, щоб отримати інформацію про проект. Файл «package.json» завжди створюється у форматі JSON, що дозволяє легко читати його як метадані та аналізувати. Іншим важливим аспектом 'package.json' є те, що він містить набір залежностей будь-якого проекту. Ці залежності є модулями, від яких залежить належне функціонування проекту.

Наявність залежностей у файлі «package.json» проекту дозволяє проекту встановлювати версії модулів, від яких він залежить. Виконуючи команду встановлення всередині проекту, розробники можуть інсталювати всі залежності, перелічені в package.json проекту, тобто ніколи не повинні бути пов’язані з самим проектом. Це також дозволяє відокремити залежності, необхідні для виробництва, і залежності, необхідні для розробки. У виробництві, ймовірно, не знадобиться інструмент для перегляду файлів CSS на наявність змін і оновлення програми, коли вони змінюються. Але як у виробництві, так і в розробці розробники хочуть мати залежності, які дозволять їм виконати певне завдання. Якщо необхідність відформатувати файл «package.json» вручну, щоб запустити проект, здається трохи складною, є зручна команда, яка автоматично створить базовий файл «package.json», який є командою «npm init».

Команда 'npm init' - це покроковий інструмент для розробки проекту. Він запропонує ввести кілька аспектів проекту в такому порядку:

- Назва проекту
- Початкова версія проекту
- Опис проекту
- проекту (мається на увазі головний файл проекту)
- Команда проекту test (щоб ініціювати тестування з чимось на зразок

Standard)

- Git-репозиторій проекту (де можна знайти джерело проекту)
- Ключові слова проекту (загалом теги, пов’язані з проектом)
- Ліцензія проекту (за замовчуванням це ISC - більшість проектів Node.js з відкритим кодом є MIT)

Після виконання кроків 'npm init', наведених вище, файл package.json буде згенеровано та розміщено в поточному каталозі.

3.2 Компіляція з вихідного коду

Проблеми з використанням останньої версії ECMAScript Хоча HTML, CSS і JavaScript постійно розвиваються, браузери користувачів можуть не підтримувати останні версії цих технологій, якщо користувачі ще не оновили свої браузери. Це створює дилему: з одного боку, нові функції, представлені в останній версії HTML, CSS і JavaScript, значно покращують досвід розробників під час роботи над проектом, але з іншого боку, програмне забезпечення користувачів може ще не підтримувати ці нові функції.

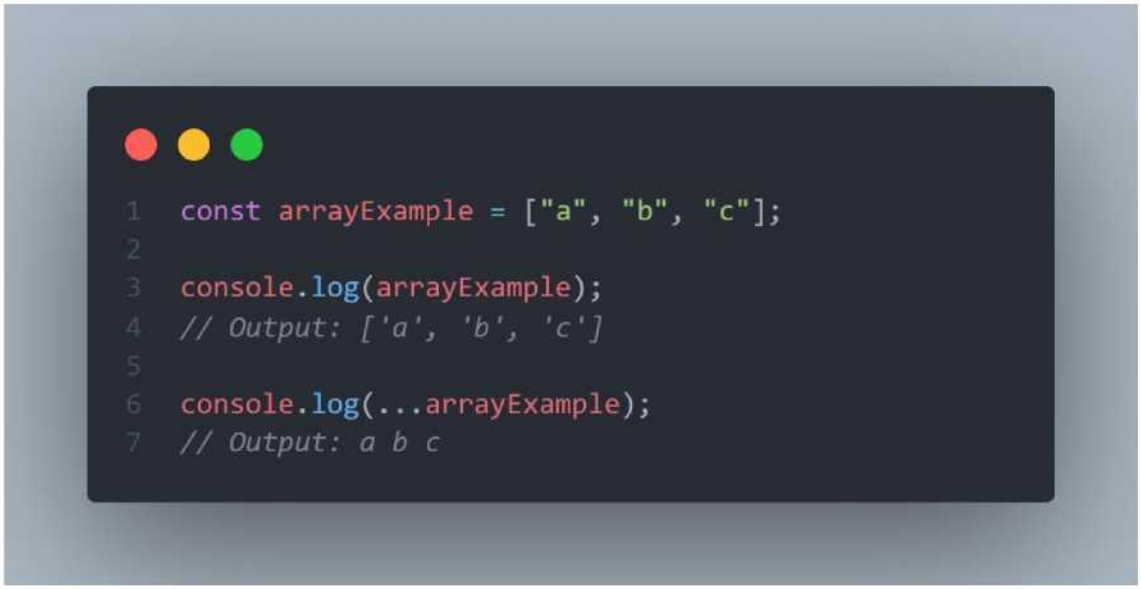


Рисунок 3.2 - Оператор поширення, представлений в ECMAScript 6.

Наприклад, як показано на рисунку 3.2, оператор поширення з ECMAScript 6 допомагає розробникам повторювати та клонувати масив або рядок, просто розміщуючи три крапки на початку масиву чи рядка без необхідності писати довгу налаштовану функцію або встановлювати сторонні бібліотеки утиліт, такі як Lodash (Ecma International TC39, 2014).

Рішення: компіляція від джерела до джерела

Щоб дозволити розробникам використовувати ці зручні нові синтаксиси без шкоди для сумісності продукту, компіляція вихідного коду є рішенням. Компілятор типу «джерело-до-джерела» — це різновид компілятора, який отримує код програми, написаний однією мовою, як вхідні дані, і повертає аналогічний вихідний код на тій самій або іншій мові на свій вихід. Компілятор «джерело-до-джерела», який інтерпретує мову програмування, має однаковий рівень абстракції [11]. JavaScript постійно розвивається. Було реалізовано численні версії JavaScript, які додали оновлення синтаксису та функціональності мови. Ці оновлення роблять JavaScript зручнішим для кодування та зменшують труднощі розробників загалом. Тим не менш, ці функції JavaScript існують уже кілька років і постійно застосовуються майже розробниками, тому браузери також потребують оновлення, щоб відповідати цим змінам. Якщо нова версія коду JavaScript виконується в непідтримуваному браузері (Internet Explorer), код не працюватиме належним чином. Однією з проблем, з якою постійно стикається кожен веб-розробник, є те, що багато нових функцій останньої версії JavaScript, які економлять час, часто підтримуються лише останніми версіями браузера. Наприклад, численні функції стандарту ECMAScript 6 не підтримуються застарілими браузерами, такими як Internet Explorer 11, або спрощеними версіями браузерів, такими як Opera Mini [18].

Практичний приклад:

Babel і ECMAScript Щоб дозволити розробникам використовувати новітні функції мови JavaScript, зберігаючи при цьому сумісність із застарілими браузерами, потрібен компілятор джерела до джерела. Серед компіляторів типу

«джерело-до-джерела» для JavaScript одним із найвідоміших є Babel. Для ілюстрації, синтаксис оператора поширення з ECMAScript 6, згаданий у попередньому розділі, буде використано як вхідні дані для Babel.

Рисунок 3.3 - Передача кодів, які використовують оператор поширення, у Babel.

Як показано на рисунку 3.3, три літери «a», «b», «c» призначаються масиву під назвою «abc», а потім масив «abc» об'єднується з новим алфавітом «d», щоб сформувати новий масив під назвою «abcd». Використання тут оператора розширення дозволяє розробникам інтуїтивно об'єднувати масиви та рядки, просто додаючи кому між масивом і новим елементом, а також гарантує, що вихідний масив «abc» ніколи не буде змінено.

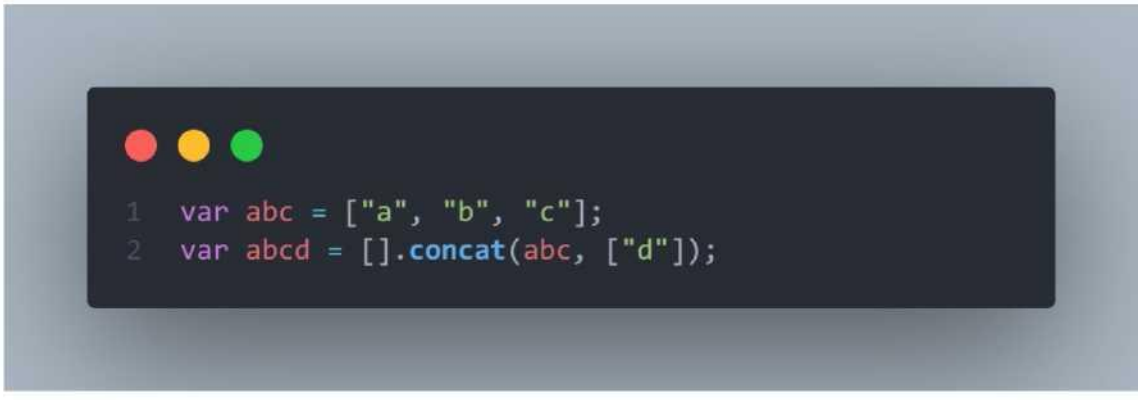


Рисунок 3.4 - Babel перекладає новий синтаксис на старий.

Як показано на рисунку 3.3, після того, як Babel отримав коди на рисунку 3.4, перша помітна зміна відбулася в 'const', оскільки тепер вони

замінені старішим синтаксисом [2] Що стосується оператора поширення, то в цьому випадку він використовується для конкатенації, тому Babel використовує його еквівалент у старішій версії JavaScript 'concat', щоб досягти того самого ефекту, зберігаючи при цьому коди сумісними зі старими браузерами [10]/

Приклад: Babel і JSX

Окрім перекладу синтаксису з останнього ECMAScript, Babel також можна використовувати для компіляції мовних розширень для JavaScript, таких як JSX і TypeScript. JSX є розширенням синтаксису мови JavaScript. Подібний за зовнішнім виглядом до HTML, JSX надає спосіб структурувати відтворення компонентів за допомогою синтаксису, знайомого багатьом розробникам. Компоненти React зазвичай пишуться за допомогою JSX, хоча це не обов'язково (компоненти також можуть бути написані за допомогою чистого JavaScript).

```
// Select .container
const container = document.querySelector(".container");

// Create div
const div = document.createElement("DIV");

// Create h1 and append it to div
const heading1 = document.createElement("H1");
heading1.innerHTML = "Hello!";
container.appendChild(heading1);

// Create h2 and append it to div
const heading2 = document.createElement("H2");
heading1.innerHTML = "Good to see you here.";
container.appendChild(heading2);

// Append div to .container
container.appendChild(div);
```

Рисунок 3.5 - Маніпуляції DOM у Vanilla JavaScript.

На рисунку 3.5 показано, як створити елемент « », який містить «<h1>» і «<h2>». Це чітко показує, що без інтуїтивно зрозумілого синтаксису HTML коди, необхідні для створення цих простих елементів, важко читати та підтримувати. За допомогою React JSX елемент HTML можна призначити безпосередньо змінній JavaScript, зберігаючи інтуїтивно зрозумілий синтаксис HTML і правильне підсвічування синтаксису. Щоб розмістити цей HTML-елемент у призначеному місці, потрібно лише загорнути змінну, якій він був призначений, у дужки, а потім помістити змінну безпосередньо в HTML. Під час компіляції проектів React Babel перекладатиме всі коди на Vanilla JavaScript.

Рисунок 3.6 - Маніпуляції DOM у React JSX.

Як видно на рисунку 3.6, створення елемента « » з «<h1>» і «<h2>» набагато простіше в React, оскільки React JSX дозволяє використовувати синтаксис HTML разом із синтаксисом JavaScript.

Проблеми з відсутністю системи зв'язування в JavaScript

Під час розробки JavaScript команда розробників представила JavaScript як мову програмування на стороні клієнта. Однак, використовуючи JavaScript,

розробники дізнаються, що його можна також використовувати як мову програмування на стороні сервера. Однак, коли мова почала розвиватися, код JavaScript став складним і важким. Через це JavaScript не був готовий виконати вимоги об’єктно-орієнтованої мови програмування. Це обмежує можливості

JavaScript для досягнення корпоративного рівня як серверної технології.

Рішення: компіляція з вихідного коду та TypeScript

TypeScript був розроблений командою розробників, щоб подолати цю прогалину [7], і це стане можливим без підтримки компіляції від джерела до джерела. TypeScript використовується для створення великих програм. Коли Microsoft вперше створила TypeScript, він мав девіз: «JavaScript, який масштабується». За допомогою TypeScript простіше керувати великою кодовою базою, оскільки він спрощує код JavaScript. TypeScript також містить інструменти для підвищення ефективності розробників і має потужну систему типів. TypeScript дуже корисний для налагодження. За допомогою JavaScript пошук помилок займає багато часу, оскільки це інтерпретована мова, тобто розробникам доводиться фактично запускати код, щоб виявити помилки, що потребує багато часу. Але всередині TypeScript є компілятор, який автоматично перевіряє код на наявність проблем. Якщо він побачить будь-які помилки, він створить повідомлення про помилку компіляції, що допоможе програмістам вивчити помилки перед виконанням коду. JavaScript типізується динамічно. Це означає, що JavaScript не знає, якого типу є змінна, доки її екземпляр не буде створено під час виконання. TypeScript додає підтримку типів у JavaScript. Помилки, викликані хибними припущеннями, що певна змінна має певний тип, можна повністю викорінити (наскільки суворо розробники вводять коди, залежить від них).

Практичний приклад: TypeScript

TypeScript робить процес введення тексту простішим і менш явним за допомогою визначення типу. "Тип" у TypeScript напряму посилається на його використання. Навіть якщо він не вводить типи явно, вони все одно існують,

щоб утримати розробників від дій, які, як наслідок, призведуть до помилки під час виконання.

Рисунок 3.7 - JavaScript не надає системи типів.

Випадок, показаний на рисунку 3.7, здається, не має такого великого значення, оскільки існує лише чотири рядки коду, але в міру того, як проект зростатиме, сотні тисяч змінних будуть додані до проекту, і вони будуть використовуватися іншими людьми, а не їхніми творцями. Коли один розробник неправильно використовує змінну, найгірший сценарій полягає в тому, що всі елементи інтерфейсу користувача можуть відтворюватися без помилок, але деякі з них можуть відтворюватися не так, як це повинно бути. У цьому випадку буде надзвичайно важко налагодити, який конкретний рядок коду є помилковим, оскільки JavaScript видаватиме помилки лише тоді, коли з компіляцією щось не так. Щоб уникнути цього, TypeScript привніс у JavaScript статичну систему набору тексту.

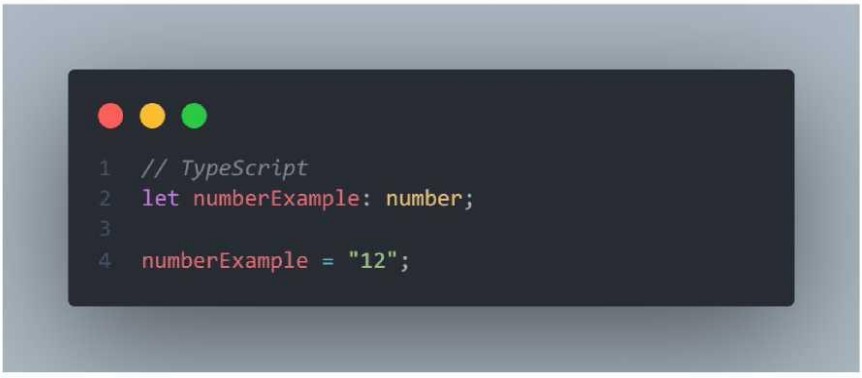


Рисунок 3.8 - TypeScript видає помилки, якщо змінна використовується неправильно

Якщо 'numberExample' присвоєно рядку, як показано на рисунку 3.8, TypeScript негайно попередить розробників про цю помилку. Оскільки, просто додавши ': number' після оголошення змінної 'numberExample', TypeScript запам'ятає, що ця змінна має бути числом.

Компіляція від джерела до джерела надає інші можливості

Нарешті, окрім створення TypeScript або використання найновішого синтаксису ECMAScript, компіляція від джерела до джерела також використовується в інших частинах зовнішньої веб-розробки, таких як CSS. Переваги препроцесора CSS, як-от Sass, уже обговорювалися в попередніх розділах. Подібно до Babel і JavaScript, ці препроцесори також переводитимуть коди, написані в розширеннях мови CSS, наприклад SCSS, у Vanilla CSS під час компіляції проекту.

3.3 Лінтування та форматування коду

Проблеми з узгодженістю та точністю кодів

Оскільки розробники можуть використовувати різні редактори для роботи над одним проектом, особливо якщо проект є відкритим кодом, потрібне рішення, яке б гарантувало, що незалежно від того, які текстові редактори чи IDE використовує член команди, їхні коди завжди будуть належним чином перевірені та відформатовані щоразу, коли вони надсилають коди до системи контролю версій. Для цього потрібні інструменти для лінінгування та форматування коду.

Рішення: Лінтер і форматер

Лінтинг або лінтер - це інструмент, який перевіряє вихідний код для позначення потенційних програмних помилок, таких як помилки, стилістичні помилки та незвичайні конструкції. Цей термін спочатку походить від утиліти

Unix, яка вивчала вихідний код мови C [16] Форматування коду або Formatter допомагає додатку видалити весь оригінальний вхідний код зі стилями та гарантувати, що всі виходи дотримуються узгодженого стилю, а також підвищити читабельність [20].

Приклад: ESLint

Наступне прикладне дослідження продемонструє, як певні технології підходять до лінтування та форматування для веб-проекту. Лінтинг для певної мови програмування простий, але коли використовуються різні технології, все стає набагато складніше. Наприклад, у цьому конкретному прикладі використовуються React, TypeScript і Sass. Одним із найпоширеніших лінтерів для JavaScript на ринку в даний час є ESLint, і приклад використання TypeScript, мовного розширення для JavaScript і разом з React, який має його синтаксис, як JSX. На щастя, ESLint надає плагіни, які дозволяють усім цим технологіям працювати разом. Усередині 'package.json', файлу, який дозволяє NPM або його альтернативам відстежувати, яка залежність потрібна, є чотири залежності.

Рисунок 3.9 - Плагіни ESLint як «devDependencies» у «package.json».

На рисунку 3.9 показано плагіни, які використовує ESLint для підтримки React JSX і TypeScript. Обидва '@typescript-eslint' призначені для того, щоб

дозволити ESLint підтримувати TypeScript, а 'eslint-plugin-react' є плагіном ESLint, який додає спеціальні правила лінінгування React для ESLint. Файл конфігурації за замовчуванням для ESLint називається '.eslintrc', можна додати спеціальні правила та параметри, щоб налаштувати ESLint відповідно до конкретних потреб проекту.

Рисунок 3.10 - Файл конфігурації для ESLint за замовчуванням називається '.eslintrc'.

На рисунку 3.10 показано деякі конфігурації, які підтримує ESLint. Синтаксичний аналізатор із '@typescript-eslint' використовується для інтерпретації синтаксису TypeScript, а в параметрах аналізатора для JSX встановлено значення true, щоб підтримувати лінтування для React. Усередині «extends» визначено правила лінінгу, які дозволяють ESLint перевіряти всі коди JavaScript, TypeScript і JSX на основі певних уподобань. Два плагіни ESLint, які обговорювалися раніше, '@typescript-eslint' і 'react', указані в розділі 'плагіни'.

Об’єкт «параметри» дозволяє додавати спільні параметри, які надаватимуться до кожного правила, визначеного в об’єкті «розширює». Нарешті, всередині об’єкта «правил» можна перезаписати деякі правила, які не відповідають проекту.

Практичний приклад: StyleLint У той час як ESLint використовується для JavaScript, TypeScript і JSX, у цьому прикладі для підтримки лінтування для всіх

таблиць стилів, таких як CSS, Sass, Less тощо, лінтер під назвою StyleLint використовується. Подібно до файлу конфігурації ESLint, для StyleLint він називається «.stylelintrc».

Рисунок 3.11 - Файл конфігурації для StyleLint за замовчуванням називається '.stylelintrc'

На рисунку 3.11 зображено деякі конфігурації, які підтримує StyleLint. Плагін StyleLint 'stylelint-scss' дозволяє StyleLint працювати з SCSS, або більш відомим як Sass, мовним розширенням CSS. Усередині «extends» визначено спеціальне правило лінінгу, яке запитує StyleLint про перевірку всіх кодів SCSS на основі конкретних уподобань.

Лінтери та форматери також підтримують правила ігнорування

Усі інструменти лінінгу та форматування підтримують правила ігнорування, подібно до Git. Усередині цих файлів «.ignore» до правил ігнорування можна додати каталоги та файли, щоб інструменти їх не торкалися.

```
# Dependencies
node_modules
bower_components
jspm_packages

# Testing
coverage

# Production (build, distribution)
dist
build

# Optional eslint cache
.eslintcache

# Optional npm cache directory
.npm

# Yarn Integrity file
.yarn-integrity

# dotenv environment variables file
.env
.env.test

# Logs
logs
*.log
npm-debug.log*
yarn-debug.log*
yarn-error.log*
```

Рисунок 3.12 - Правила ігнорування можна визначити в цих файлах «.ignore».

На рисунку 3.12 показано приклад файлів «.ignore». Найчастіше ігноруються каталоги, які містять залежності, результати тестування, файли конфігурації та файли збірки. Не є винятком у цьому прикладі 'node_modules' і «bower_components» - це папки, у яких зберігаються залежності, «coverage» для результатів тестування від Jest, dist» і «build», а dist» містить файли збірки, згенеровані Webpack, які готові до надання клієнтам. Стандартні регулярні вирази також підтримуються в цих файлах '.ignore', як можна побачити в останній частині фрагмента коду. У розділі «#Config» «webpack.*.js» дозволяє

інструментам ігнорувати всі файли, які починаються з «webpack», і те саме правило також застосовується до «jest.*.js». Що стосується '*.config.js', це означає, що всі файли, які закінчуються на 'config.js', будуть проігноровані.

Проблеми із запуском linter і formatter для кожного файлу

Після того як усі інструменти лінінгу та форматування правильно налаштовано, у цьому проекті з підтвердження концепції все ще бракує однієї речі - автоматично запускати всі поетапні файли проти лінтерів і форматувальників під час кожного коміту коду. Причина перевіряти лише поетапні файли замість виконання повної команди linting або форматування полягає в тому, що зі збільшенням масштабу проектів кількість часу, необхідна для перевірки всіх файлів, також масштабується разом із цим. Однак, щоб досягти цього, Git або інші системи контролю версій повинні мати можливість спілкуватися з усіма інструментами linting і форматування.

Рішення та практичні приклади: Git Hooks, Husky та Lint-staged

На щастя, Git Hooks може зробити саме це. У цьому прикладі встановлено перехоплювачі Git під назвою "Husky", щоб уможливити запуск користувальницьких сценаріїв для кожної команди Git, тоді як "lint-staged" використовується для вилучення імен зі списку проміжних файлів, які контролюються Git, і передачі імен усім інструментам linting і форматування.

Рисунок 3.13 - «lint-staged» запускатиметься під час кожного «git-коміту».

Рисунок 3.13 демонструє, як дозволити Git Hooks запускати Lint-staged для кожного коміту Git. Однак різні командні команди командного рядка для лінінгу та форматування часто підтримують різні типи шляхів до файлів, тому необхідно налаштувати шляхи, отримані з Git, щоб відповідати вимогам конкретного інтерфейсу командного рядка для лінінгування чи форматування. 'lint-staged' взяв це до уваги, тому підтримує написання функцій JavaScript для налаштування шляхів.

На рисунку 3.14 показано, як Lint-staged можна налаштувати для передачі відносних або абсолютних шляхів залежно від вимог інструменту.

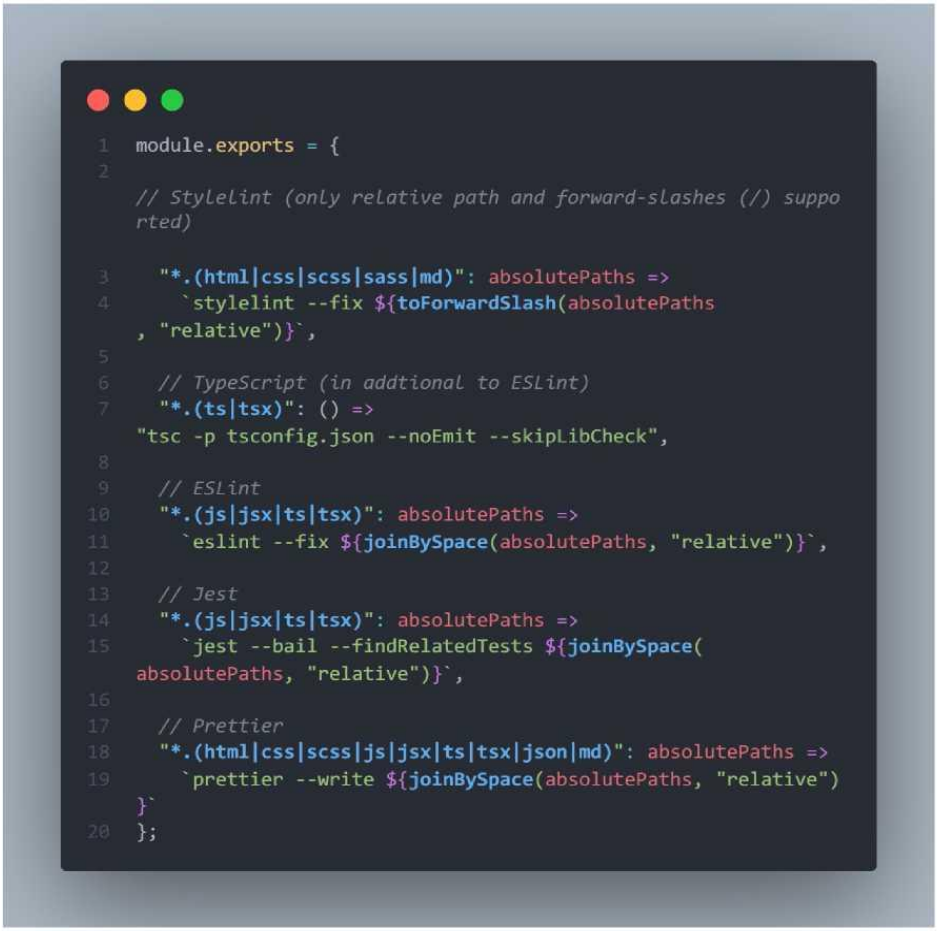


Рисунок 3.14 - 'module.exports' файлу конфігурації 'lint-staged'.

3.4 Тестування та налагодження

Проблеми. Під час написання скомпільованих мов програмування потрібна IDE, щоб мати можливість запускати коди. Такі IDE часто постачаються з вбудованими платформами тестування та налагоджувачами як

вбудовані NUnit і Xunit Microsoft Visual Studio для тестування C# ([25]. Як обговорювалося на початку розділу «Середовище розробки», відсутність відповідних IDE для веб-розробки призводить до появи великої кількості наборів інструментів сторонніх розробників, це не є винятком, коли справа стосується тестування та налагодження. Для інтерфейсної веб-розробки традиційно тестування значною мірою покладалося на наскрізне тестування, яке вимагало від розробників тестування всіх елементів і функцій усіх рівнів інтерфейсу користувача вручну. Що стосується налагодження, то воно часто робилося за допомогою вбудованих інструментів перевірки браузерів. З появою численних бібліотек і фреймворків для зовнішнього веб-сайту також розроблено налаштовані тестові фреймворки та налагоджувачі, спеціально призначені для роботи з кожною з цих бібліотек або фреймворків [19].

Види тестуванняю. Веб-тестування - це метод тестування програмного забезпечення для перевірки веб-додатків на потенційні помилки. Будь-які веб-додатки повинні пройти це тестування перед тим, як запускати їх у виробництво. Впроваджуючи тестування веб-сайтів, організації можуть переконатися, що система функціонує точно та може використовуватися користувачами в режимі реального часу [1]. Модульне тестування - це тип тестування програмного забезпечення, де тестуються окремі одиниці або компоненти програмного забезпечення. Зазвичай цей процес виконується на стадії розробки програми. Модульне тестування ізолює сегмент кодової бази та перевіряє його правильність. Модульне тестування - це найбільш фундаментальне тестування, яке зазвичай виконується розробником . Наскрізне тестування – це метод, який використовується для оцінки потоку програми від початку до кінця для спостереження за поведінкою веб-програми. Метою цього

процесу є забезпечення збереження цілісності даних і виявлення системних залежностей у кількох системних компонентах і системах. Ціла програма перевіряється на ключові функції, такі як зв’язок з іншими інтерфейсами, базами даних, системами, мережами та іншими програмами.

Рішення: сторонні фреймворки тестування та налагоджувачі У наступному прикладі Jest і Enzyme будуть використані для написання тестів для React, Redux, TypeScript і JavaScript загалом. Для налагодження використовуються React Developer Tools і Redux DevTools разом із вбудованими інструментами перевірки браузерів.

Практичний приклад: Jest - це фреймворк для тестування JavaScript, створений розробниками з Facebook, який надає прості у використанні API та функції, такі як тестування знімків [26]. Наразі Jest є найпоширенішою платформою для тестування JavaScript у галузі [21]. Подібно до інших бібліотек і фреймворків, Jest також підтримує використання файлів конфігурації для налаштування параметрів і функцій. У цьому випадку «jest.config.js» може бути автоматично згенерований, спочатку глобально встановивши Jest CLI, а потім запустивши «jest -init» у терміналі.

На рисунку 3.15 показано деякі конфігурації, які підтримує Jest. За замовчуванням 'jest.config.js' матиме всі параметри Jest, доступні у файлі у вигляді коментарів. Щоб змінити певний параметр, просто видаліть коментарі з рядка кодів і змініть його відповідно. Файл «jest.config.js» із усіма видами корисних коментарів можна створити, запустивши «jest --init» у терміналі. Усередині 'package.json' можна додавати спеціальні сценарії. У цьому випадку запуск «npm test» (або «yarn test», якщо використовується Yarn) у терміналі виконає «jest -coverage». Позначка «- coverage» є одним із параметрів Jest CLI, який створює докладний звіт про покриття тестування проекту.

```
module.exports = {
  clearMocks: true,

  coverageDirectory: "coverage",

  roots: ["<rootDir>"],

  setupFilesAfterEnv: ["<rootDir>/src/utils/testSetup.ts"],


  snapshotSerializers: ["enzyme-to-json/serializer"],

  testEnvironment: "jsdom",

  testMatch: ["**/__tests__/**/*.[jt]s?(x)",
"**/?(*.)+(spec|test).[tj]s?(x)"],

  transform: {
    "^.+\\.(t|j)sx?$": "ts-jest"
  },

  verbose: true
};
```

Рисунок 3.15 - Файл конфігурації для Jest.

```
{
  "scripts": {
    "test": "jest --coverage"
  }
}
```

Рисунок 3.16 - Налаштовані сценарії можна визначити всередині об’єкта «сценарії».

Як видно на рисунку 3.16, запуск «npm test» виконає «jest --coverage», тому Jest не потрібно встановлювати глобально. '--coverage' створить папку 'coverage' у кореневому каталозі проекту, яка містить такі файли, як HTML, які можна відкрити, щоб перевірити, скільки рядків коду охоплено тестами.

Кейсове дослідження: Фермент Крім Jest, Enzyme - це утиліта для тестування, яка часто використовується разом з іншими фреймворками тестування для тестування виходу компонента React. Enzyme описується як

неперевірений, коли мова заходить про роботу з бандлерами та бібліотеками тверджень, тому його можна використовувати разом із більшістю популярних залежностей веб-проектів, таких як Webpack Mocha, Browserify тощо (Airbnb, Inc., 2020).

Рисунок 3.17 - Фермент і адаптер буде імпортовано та налаштовано під час кожного тестового запуску.

На рисунку 3.17 показано, як автоматично імпортувати Enzyme та його адаптер під час кожного тестового запуску. За замовчуванням потрібно імпортувати фермент, адаптер і налаштувати адаптер для кожного написаного тесту. На щастя, опція «setupFilesAfterEnv» із «jest.config.js» може вирішити цю проблему, просто перемістивши повторювані коди, необхідні для кожного тесту, в окремий файл, а потім додайте шлях до цього файлу в опцію «setupFilesAfterEnv». У цьому випадку створюється «testSetup.js» для обробки ферменту та його адаптера. Перенесення повторюваних кодів, необхідних для кожного тесту, в окремий файл, щоб у разі необхідності змінити ці рядки коду нам не потрібно було використовувати такі функції IDE, як пошук і заміна. Натомість ми можемо просто змінити їх лише один раз.

Як показано на рисунку 3.18, додавання шляху до файлу до параметра 'setupFilesAfterEnv' дозволить Jest запускати коди в цьому файлі перед кожним тестом.

Рисунок 3.18 - Додавання шляху до файлу вище до конфігурації Jest.

Рисунок 3.19 - Порожній модуль експортовано для представлення активів

На рисунку 31 показано, як налаштувати Jest пропускати ресурси під час виконання тестів, оскільки такі активи, як зображення та інші формати, такі як CSS, Jest не розпізнає. Таким чином, для імітації поведінки цих файлів потрібно використовувати заглушку. У цьому випадку, оскільки самі файли необхідно перевірити, а стиль елементів інтерфейсу користувача не стосується Jest або Enzyme, для представлення цих файлів експортується порожній модуль.

Ферментні та React-Redux хуки. Незважаючи на те, що Jest і Enzyme працюють з React, Redux, JavaScript або TypeScript досить прямо, нові хуки, такі як React Hooks і React-Redux Hooks, внесли додаткову складність. Без хуків Jest і Enzyme можуть отримати прямий доступ до станів компонента, щоб перевірити, як він змінюється та маніпулюється, а використання Redux означає,

що деякі стани компонентів зберігаються в сховищі Redux, а деякі – у компонентах. Хоча використання хуків React-Redux не заважає нам отримати доступ до станів усередині сховища Redux, усі внутрішні стани компонента залишаються недоступними. Таким чином, замість того, щоб перевіряти стани самих компонентів, тепер тести мають зосереджуватися на тому, як змінилися елементи інтерфейсу користувача компонента, якщо стани оновлюються, і щоб перевірити, як оновлюються стани, нам потрібно змоделювати такі взаємодії, як подія «onClick», щоб ініціювати оновлення. Іншими словами, ми повинні відійти від традиційного модульного тестування, відомого як тестування окремі функціональні можливості, щоб перевірити, як користувачі взаємодіють з елементами інтерфейсу користувача та що ці взаємодії означають для станів компонента.

Як видно на рисунку 3.20, оскільки деякі стани зберігаються в Redux, тому для полегшення процесу написана допоміжна функція для налаштування фіктивного сховища (сховища, яке імітує реальне). Функція для налаштування фіктивного сховища Redux для імітації справжнього; користувач цієї функції може передавати необов’язкові аргументи для зміни налаштувань сховища за замовчуванням.

```
// Setup a store with some fake states for testing
export function setupStore({
  playerList = [
    {
      id: "1654A",
      name: "Alpha",
      score: 10,
      created: "11/21/2016",
      updated: "06/12/2018",
      isSelected: false
    },
    {
      id: "1654B",
      name: "Beta",
      score: 4,
      created: "11/8/2016",
      updated: "11/9/2017",
      isSelected: false
    }
  ],
  isRunning = false,
  previousTime = 0,
  elapsedTime = 0,
  isLightMode = false,
  visibility = false
}: setupStorePars = {}): Store {
  return createStore(rootReducer, {
    notification: {
      storage: {
        visibility
      }
    },
    scoreboard: {
      player: {
        playerList
      },
      stopwatch: {
        isRunning,
        previousTime,
        elapsedTime
      },
      theme: {
        isLightMode
      }
    }
  });
}
```

Рисунок 3.20 - Допоміжна функція, яка налаштувала імітований магазин Redux.

```
function setupWrapper(mockStore = setupStore(), props =
 {}): ReactWrapper {
  return mount(
    <Provider store={mockStore}>
      <Scoreboard {...props} />
    </Provider>
  );
}
```

Рисунок 3.21 - Як використовується функція налаштування магазину Redux

На рисунку 3.22 показано, як можна використовувати коди з рисунка 3.21. Імітаційне сховище Redux спочатку передається постачальнику, а потім постачальник передає вміст цього сховища компоненту.

```
describe("Scoreboard", () => {
  let wrapper: ReactWrapper;

  // Setup wrapper
  beforeEach(() => {
    wrapper = setupWrapper();
  });

  test("compare to the last snapshot", () => {
    expect(wrapper).toMatchSnapshot();
  });

  it("should call watchForHover()", () => {
    expect(watchForHover).toHaveBeenCalledTimes(1);
  });
});
```

Рисунок 3.22 - Приклад тестування компонента React.

На рисунку 3.22 показано, як компонент React тестується в Jest і Enzyme, у якому використовується тестування знімків, щоб гарантувати, що UI не зміниться неочікувано без необхідності писати тест для кожного елемента UI .

Рисунок 3.23 - Приклад того, як перевіряються функції, пов’язані з Redux.

На рисунку 3.23 під час тестування пов’язаних із Redux функцій компонента перевіряється, як ці функції впливають на фактичні елементи інтерфейсу користувача, щоб краще відобразити, як користувачі насправді взаємодіятимуть із програмою.

Два способи організації тестових файлів

Загалом, є два способи організації тестових файлів. Один із методів полягає в тому, щоб розмістити тестовий файл поруч із файлом, який він має тестувати. Цей підхід підкреслює, що тести мають бути розширенням кодів, які вони тестують. Інший поширений метод полягає в тому, щоб помістити всі тести в папку «test» і організувати тести так, щоб вони віддзеркалювали фактичну структуру проекту.

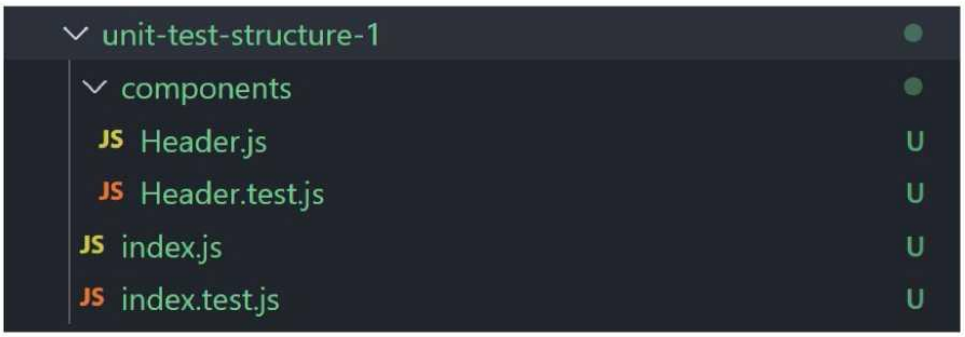


Рисунок 3.24 - Тестові файли розміщуються поруч із файлом, який вони тестують.

Як показано на рисунку 3.24, одним із методів упорядкування тестових файлів є розміщення тестового файлу поруч із файлом, який він має тестувати. Цей підхід означає, що коди тестування є розширенням кодів, які вони тестують.

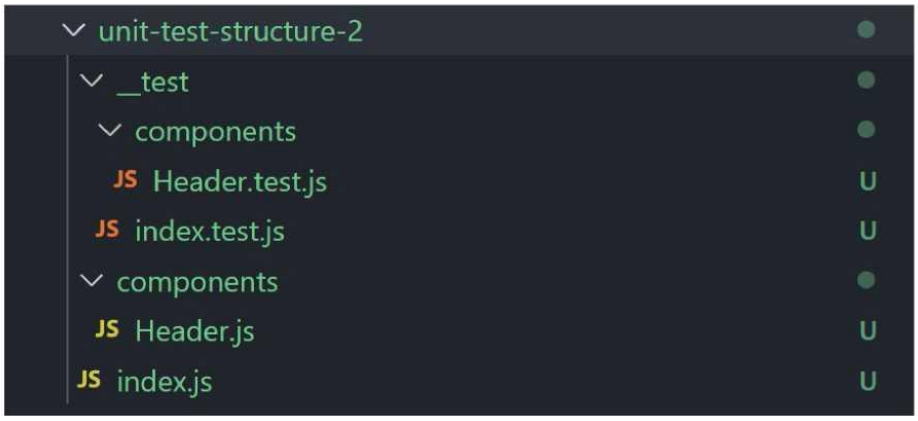


Рисунок 3.25 - Усі тестові файли розміщено в папці «test».

Як показано на рисунку 3.25, іншим методом є розміщення всіх тестів у папці «test» і організуйте тести, щоб відобразити фактичну структуру проекту, цей підхід наголошує на тому, щоб зробити кожен модуль проекту менш перевантаженим і легше переміщатися по всіх каталогах. Відокремлюючи тести від фактичних кодів, папка, яка містить модуль, міститиме лише файли, які стосуються функціональності модуля.

3.5 Оптимізація для проблем розгортання

Перш ніж інтерфейсний веб-проект буде готовий до розгортання, необхідно врахувати кілька оптимізацій, інакше продуктивність постраждає, і, зрештою, користувацький досвід буде гіршим. Оскільки, знову ж таки, немає авторитетної особи, яка б надала офіційні інструменти розробки, для вирішення кожної частини головоломки потрібні різні інструменти сторонніх розробників.

Рішення та кейс: мініфікація

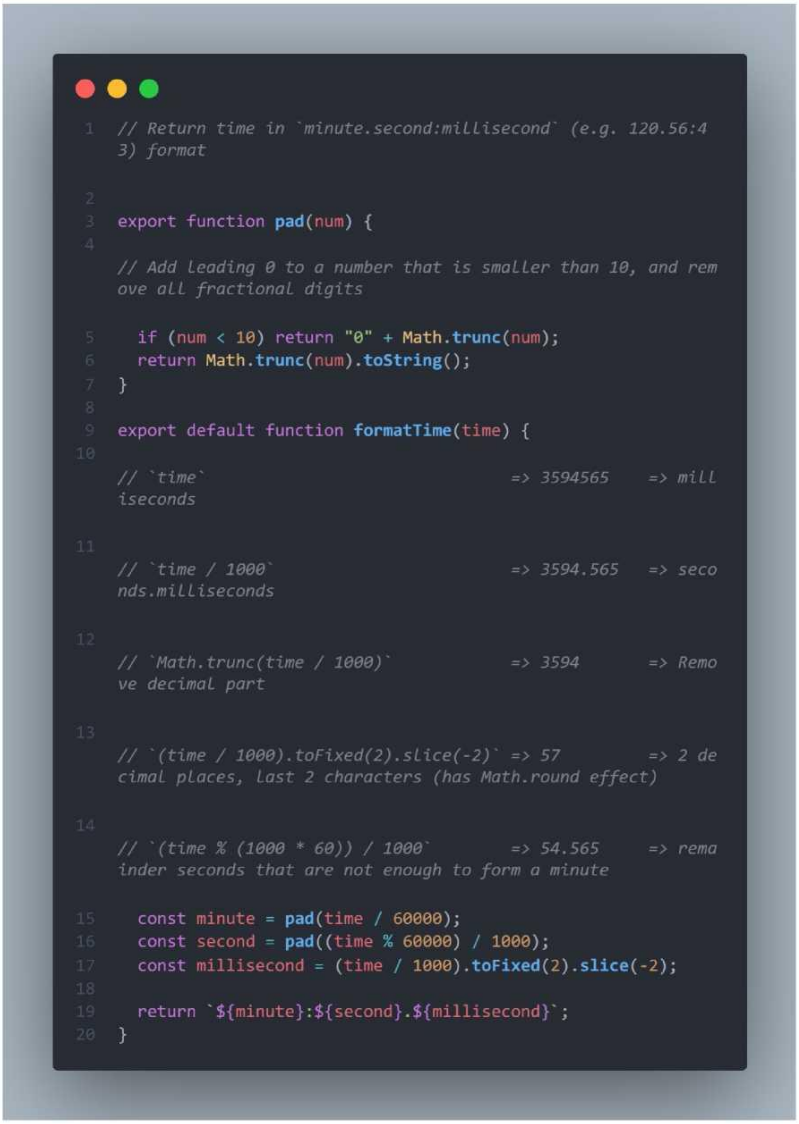
Почнемо з того, що мініфікація — це процес усунення коментарів,

непотрібних пробілів і подібних надмірностей у вихідних кодах для зменшення розміру файлу та покращення швидкості завантаження (Google LLC, 2018).

Рисунок 3.26 - Вихідні коди JavaScript після мініфікації.

Як показано на рисунку 3.26, вихідні коди JavaScript перед мінімізацією зазвичай складаються з багатьох порожніх місць і коментарів для покращення читабельності.

На рисунку 3.27 показано, як коди з рисунка 3.28 виглядають після мініфікації. Як можна помітити, порівнюючи рисунок 3.28 і рисунок 3.27, після мінімізації вихідних кодів вони більше не читаються людиною, але різниця в розмірі файлу є значною. Розмір файлу вихідного коду на рисунку 36 становить 1,28 КБ (1318 байт), тоді як розмір коду на рисунку 37 становить лише 188 байт (188 байт), різниця між ними становить 1130 байт або 86%. Чим більший проект, тим більший прибуток від мініфікації. У дослідженні прикладу використовувався збірник Webpack, який за замовчуванням використовуватиме «TerserWebpackPlugin» для мінімізації вихідних кодів JavaScript у робочому режимі (Webpack, 2020).



```
// Return time in `minute.second:millisecond` (e.g. 120.56:43) format

export function pad(num) {

// Add leading 0 to a number that is smaller than 10, and remove all fractional digits

  if (num < 10) return "0" + Math.trunc(num);
  return Math.trunc(num).toString();
}

export default function formatTime(time) {

// `time`                              => 3594565    => milliseconds

// `time / 1000`                       => 3594.565   => seconds.milliseconds

// `Math.trunc(time / 1000)`           => 3594       => Remove decimal part

// `(time / 1000).toFixed(2).slice(-2)` => 57        => 2 decimal places, last 2 characters (has Math.round effect)

// `(time % (1000 * 60)) / 1000`       => 54.565     => rema
inder seconds that are not enough to form a minute

  const minute = pad(time / 60000);
  const second = pad((time % 60000) / 1000);
  const millisecond = (time / 1000).toFixed(2).slice(-2);

  return `${minute}:${second}.${millisecond}`;
}
```

Рисунок 3.27 - Вихідні коди JavaScript до мініфікації

```
module.exports = {
  // "development" will enable debug and etc.
  // "production" will enable minifier and etc.
  mode: "production"
};
```

Рисунок 3.28 - Встановлення робочого режиму в 'webpack.config.js' для ввімкнення мініфікації.

Як показано на рисунку 3.28, встановивши для «mode» значення «production», Webpack увімкне вбудовану мініміфікацію. Однак Webpack не підтримує мінімізацію для CSS за замовчуванням, тому для цього потрібні плагіни.

На рисунку 3.29, щоб дозволити Webpack мінімізувати коди CSS, до процесу створення Webpack можна додати такі плагіни, як «mini-css-extract-plugin».

Рисунок 3.29 - Як додати мініміфікацію CSS для Webpack.

Рішення та приклад: стиснення. Щоб ще більше зменшити розмір файлу, алгоритми стиснення, такі як Gzip, можна інтегрувати в проект для створення стисненої версії того самого файлу, зберігаючи оригінальний файл, щоб веб-сервер, який не підтримує стиснення, міг обслуговувати оригінальні файли. Gzip — це програмний додаток, створений Жаном-Лу Ґейлі та Марком Адлером для стиснення та депресії даних [27].

Як видно на рисунку 3.30, щоб дозволити Webpack увімкнути стиснення для всіх файлів, які не є активами, потрібно встановити «compression-webpack-plugin».

Рішення та приклад: розділення коду. JavaScript часто розглядається як інтерпретована мова, і його коди можна запускати без компіляції. Традиційно, коли браузер отримує вихідні коди JavaScript, вони зазвичай передаються інтерпретатору браузера, де інтерпретується кожен рядок коду. Зараз є більш сучасні браузери, такі як Google Chrome реалізував компілятор JIT (Just-in-time) у своєму механізмі для підвищення продуктивності під час виконання кодів JavaScript [22]. Однак, незалежно від того, чи браузер інтерпретує чи компілює вихідні коди JavaScript, йому все одно потрібно спочатку пройти всі коди, перш ніж буде відображено будь-який значущий вміст, і це особливо вірно для SPA, відтворених на стороні клієнта. Таким чином, якщо веб-сайт містить усі вихідні коди JavaScript в одному файлі, і користувач відвідує сайт уперше, пройде багато часу, перш ніж користувачі зможуть побачити щось значуще або взаємодіяти з будь-якими елементами на сторінці. Ось чому поділ коду тепер став основною частиною зовнішньої веб-розробки. Розбиття коду, як випливає з назви, - це процес поділу вихідного коду JavaScript на менші частини на основі певних принципів. Один із поширених способів розбиття коду - помістити всі коди зі сторонніх бібліотек і фреймворків в один файл, зберігаючи коди в іншому [22].

```
const MiniCssExtractPlugin = require("mini-css-extract-plugin");
const CompressionPlugin = require("compression-webpack-plugin");

module.exports = {
  // "development" will enable debug and etc.
  // "production" will enable minifier and etc.
  mode: "production",

  // "source-map" for "production"
  // "inline-source-map" for "development"
  devtool: "source-map",

  plugins: [
    // Extracts CSS into separate files
    new MiniCssExtractPlugin({

// [contenthash] genereates hash based on the content of individua
l file, expected to be default in Webpack 5

      filename: "[name].[contenthash].css",
      chunkFilename: "[name].[contenthash].css",
      // Enable/diable ignoring warnings about conflicting order
      ignoreOrder: false
    }),

    // Gzip
    new CompressionPlugin()
  ]
};
```

Рисунок 3.30 - Як додати стиснення файлу для Webpack.

```
optimization: {
  // Code splitting
  splitChunks: {
    cacheGroups: {
      commons: {
        test: /[\\/]node_modules[\\/]/,

// Place all codes from 3rd-party inside `vendor.js`
        name: "vendor",

// `all`, `async`, or `initial`, see https://webpack.js.o
rg/plugins/split-chunks-plugin/

        chunks: "all"
      }
    }
  }
}
```

Рисунок 3.31 - Webpack підтримує розділення коду за замовчуванням

Як показано на рисунку 3.31, Webpack підтримує розділення коду з коробки, щоб увімкнути це, просто додайте об’єкт «splitChunks» в об’єкт «optimization».

Рішення та приклад: Кешування

Під час використання такого браузера, як Google Chrome, він зберігатиме деякі дані, завантажені з веб-сайту, у свій локальний кеш [17]. Іноді це викликає проблеми, коли веб-сайт оновив свої файли, але оскільки імена цих файлів залишилися незмінними, браузери просто завантажуватимуть старі файли з локального кешу. Щоб усунути цю проблему, певний механізм автоматичного іменування для іменування файлів на основі їх вмісту буде рішенням, і Webpack підтримує саме це.

Як показано на рисунку 3.32, додавши '[contenthash]' до 'filename' і 'chunkFielname' всередині об'єкта 'output', хеш-код буде автоматично згенерований Webpack на основі вмісту кожного окремого файлу.

Рисунок 3.32 - Webpack підтримує генерування хешу на основі вмісту файлу.

Загалом аналіз показав, що кожна окрема бібліотека, фреймворк і утиліта, створена для зовнішньої веб-розробки, призначена для вирішення однієї або кількох проблем, спричинених дизайном HTML, CSS і JavaScript. Хоча створення цілої бібліотеки виключно для вирішення конкретної проблеми часом здається надмірним, однак це результат відсутності авторитетної особи, щоб забезпечити ретельно розроблену IDE для зовнішньої веб-розробки. Починаючи з вирішення проблеми повторення, успадкованої від дизайну HTML і CSS, а потім розширюючи до вдосконалення способу взаємодії JavaScript з елементами інтерфейсу користувача, створено такі інструменти, як React, Sass тощо. Оскільки зовнішня частина веб-розробки стає дедалі складнішою, все більше і більше розробників традиційного програмного забезпечення приєднуються до неї та привносять із собою багато перевірених концепцій у зовнішню веб-розробку. Оскільки дизайн HTML, CSS і JavaScript часто суперечить цим шаблонам дизайну більш зрілих мов програмування, кількість інтерфейсних веб-інструментів, створених для вирішення всіх цих проблем, зростає ще більше. Оскільки вимоги до зовнішньої веб-розробки дедалі зростають, три стандарти HTML, CSS і JavaScript також постійно розвиваються в надії вирішити всі проблеми без необхідності використання будь-яких сторонніх веб-бібліотек, фреймворків і утиліт. Однак швидкість, з

якою розвиваються стандарти, недостатньо швидка, щоб наздогнати зростання Інтернету, і причина цього полягає в тому, що всі нові специфікації покращили веб-інтерфейс загалом, вони не вирішують найважливішу проблему веб-розробки інтерфейсу, а саме відсутність належної IDE. Те, що надали ці нові функції з пізніших ітерацій веб-стандартів, насправді дозволило створювати потужніші бібліотеки, фреймворки та утиліти. Підводячи підсумок, у цій роботі було виявлено, що фундаментальні проблеми, пов'язані з розробкою HTML, CSS і JavaScript, у поєднанні з відсутністю авторитетної особи у зовнішній веб-розробці, як-от у мобільних чи настільних платформах, зрештою призводять до створення великої кількості зовнішніх веб-бібліотек, фреймворків, мовних розширень тощо сторонніх розробників.

3.6 Висновки до розділу

Сучасне середовище веброзробки ґрунтується на сторонніх інструментах, таких як Visual Studio Code, Node.js та NPM, які компенсують відсутність офіційного IDE, забезпечуючи повноцінну інфраструктуру для керування залежностями та автоматизації процесів. Завдяки цьому розробники отримують гнучке та масштабоване середовище для ефективного створення вебдодатків.

Компіляція з вихідного коду є ключовим рішенням для забезпечення сумісності сучасного синтаксису JavaScript і CSS з застарілими браузерами, що дозволяє розробникам використовувати новітні можливості мов без шкоди для працездатності застосунків. Інструменти на кшталт Babel і TypeScript значно підвищують продуктивність і стабільність розробки, забезпечуючи перевірку типів та зворотну сумісність.

Отже, використання інструментів лінтування, форматування, тестування та налагодження є невід'ємною частиною сучасної веброзробки, що сприяє підвищенню якості коду та ефективності командної роботи. Це дозволяє уникнути помилок на ранніх етапах та забезпечити стабільність застосунку

Оптимізація вебзастосунків перед розгортанням — це критично важливий

етап, що включає мініфікацію, стиснення, розділення коду та кешування, які разом значно покращують продуктивність та досвід користувача. Водночас відсутність централізованої екосистеми стимулює розвиток численних сторонніх інструментів, покликаних компенсувати недоліки стандартів HTML, CSS та JavaScript.

ВИСНОВКИ

Роботаспочатку розглядає проблеми, які зараз присутні у сфері веб-розробки, від основ до досягнень. Ці проблеми залишаються в веб-індустрії протягом дуже тривалого часу. Щоб вирішити всі ці проблеми, спільноти розробників створили велику кількість веб-бібліотек, фреймворків тощо, як показано на рисунку 45. Крім того, роботатакож відповідає на питання, чому в веб-сфері існує так багато веб-фреймворків. Можна зробити висновок, що хоча кількість інтерфейсних веб-бібліотек і фреймворків надзвичайно велика, це не повинно бути перешкодою для розробників. Якби організація, відповідальна за розповсюдження стандартів, не була в змозі слідкувати за процвітанням цієї галузі, що швидко змінюється, веб-розробка не розвивалася б суттєво без створення цих фреймворків. Автори вважають, що це корисно, оскільки надає більше альтернативних рішень для розробників і допомагає створити більш відкриту та децентралізовану мережу.

Дослідження в цій роботі показало, що кількість фреймворків, бібліотек та їхніх альтернатив величезна, але це є необхідністю для розвитку цієї галузі. Це обмеження, але це, безсумнівно, перевага. Багато фреймворків, які існують на ринку, поділяють ту саму філософію та націлені на той самий результат. Хоча шляхи, які обирає кожен фреймворк, різні, вони завжди досягають одного напрямку й цілі: створювати багатофункціональні та потужні веб-додатки. Вибір інструментів для застосування є складним рішенням як для молодших, так і для досвідчених розробників. Відповідно, сьогодні веб-розробники повинні глибоко розуміти або принаймні мати досвід роботи з більш ніж одним або двома фреймворками та бібліотеками, а не просто вивчати певну. Хоча це твердження не зовсім точне, різноманітність знань завжди важлива. Наразі ринок відповів кількома обмеженими діями. Розробники веб-сайтів і CMS все частіше звертаються до низькокваліфікованих програмістів або розробників. Wix, WordPress, Microsoft Power Apps, Google App Maker – кілька прикладів. Такі рішення далеко не повне вирішення того, що лежить за сценою. Це швидкі

рішення для невеликих стартапів або навіть персональних веб-сайтів, але не для підприємств.

Тому потреба в досвідчених веб-розробниках все ще потрібна. Для ширшого погляду, за останні кілька років спостерігався сплеск різноманітності споживчих пристроїв. Це дуже важливо для галузі веб-розробки. Більша різноманітність пристроїв пов’язана з більшою різноманітністю платформ, типів і форм входів і виходів як для технологічної галузі в цілому, так і для веб-розробки окремо. Досягти максимального використання веб-додатків важко, розробники сьогодні повинні планувати та робити їх доступними для широкого та непередбачуваного діапазону пристроїв, тобто створювати їх таким чином, щоб веб-розробники могли ефективно адаптувати їх до будь-яких поточних пристроїв, а також бути готовими до майбутнього.

СПИСОК ПОСИЛАНЬ НА ДЖЕРЕЛА

1. Addison, P. *Modern Web Development with JavaScript.* — Packt Publishing, 2024. — 432 с. — Режим доступу: https://www.packtpub.com/product/modern-web-development-with-javascript/9781803243603

2. Amazon Web Services. AWS Amplify: Building Modern Web Applications. *aws.amazon.com*, 2024. — Режим доступу: https://aws.amazon.com/amplify

3. Andrew, R. *The New CSS Layout.* — A Book Apart, 2017. — 144 с. — Режим доступу: https://abookapart.com/products/the-new-css-layout

4. Banks, A., & Porcello, E. *Learning React: Functional Web Development with React and Redux.* — 2nd ed. — O'Reilly Media, 2020. — 310 с. — Режим доступу: https://www.oreilly.com/library/view/learning-react/9781492051718

5. Cederholm, D. *CSS3 for Web Designers.* — A Book Apart, 2015. — 136 с. — Режим доступу: https://abookapart.com/products/css3-for-web-designers

6. Coyier, C. *Practical SVG.* — A Book Apart, 2016. — 112 с. — Режим доступу: https://abookapart.com/products/practical-svg

7. Duckett, J. *HTML and CSS: Design and Build Websites.* — Wiley, 2011. — 512 с.

8. Eich, B. JavaScript: The definitive guide to the language's past, present, and future. *ACM Computing Surveys*, 2023, 55(9), 1–25. — Режим доступу: https://doi.org/10.1145/3604908

9. Flanagan, D. *JavaScript: The Definitive Guide.* — 7th ed. — O'Reilly Media, 2020. — 704 с. — Режим доступу: https://www.oreilly.com/library/view/javascript-the-definitive/9781491952016

10. Freeman, A. *Pro Angular 16.* — Apress, 2024. — 772 с. — Режим доступу: https://www.apress.com/gp/book/9781484299975

11. Google. Web Fundamentals: Modern Web Development Practices. *web.dev*, 2024. — Режим доступу: https://web.dev

12. Haverbeke, M. *Eloquent JavaScript: A Modern Introduction to Programming.* — 3rd ed. — No Starch Press, 2018. — 472 с. — Режим доступу: https://eloquentjavascript.net

13. Hogan, L. *Designing for Performance: Weighing Aesthetics and Speed.* — O’Reilly Media, 2014. — 180 с. — Режим доступу: https://www.oreilly.com/library/view/designing-for-performance/9781491903728

14. Keith, J., & Andrew, R. *HTML5 for Web Designers.* — 2nd ed. — A Book Apart, 2016. — 96 с. — Режим доступу: https://abookapart.com/products/html5-for-web-designers

15. Lawson, B., & Sharp, R. *Introducing HTML5.* — 2nd ed. — New Riders, 2011. — 312 с.

16. MacDonald, M. *Node.js: Novice to Ninja.* — SitePoint, 2022. — 408 с. — Режим доступу: https://www.sitepoint.com/premium/books/node-js-novice-to-ninja

17. Marcotte, E. *Responsive Web Design.* — 2nd ed. — A Book Apart, 2014. — 160 с. — Режим доступу: https://abookapart.com/products/responsive-web-design

18. **MDN Web Docs. Web APIs and Frameworks. *developer.mozilla.org*, 2025. — Режим доступу: https://developer.mozilla.org/en-US/docs/Web

19. Microsoft. TypeScript Handbook: Building Scalable Web Applications. *typescriptlang.org*, 2024. — Режим доступу: https://www.typescriptlang.org/docs

20. Osmani, A. *Learning JavaScript Design Patterns.* — 2nd ed. — O’Reilly Media, 2023. — 512 с. — Режим доступу: https://www.oreilly.com/library/view/learning-javascript-design/9781098139865

21. Powers, S. *Learning Node: Moving to the Server-Side.* — 2nd ed. — O’Reilly Media, 2016. — 288 с. — Режим доступу: https://www.oreilly.com/library/view/learning-node/9781491943113

22. React. Official Documentation: Building User Interfaces. *react.dev*, 2024. — Режим доступу: https://react.dev

23. Resig, J. *Secrets of the JavaScript Ninja*. — 2nd ed. — Manning Publications, 2016. — 464 с. — Режим доступу: https://www.manning.com/books/secrets-of-the-javascript-ninja-second-edition

24. Vue.js. Official Documentation: The Progressive JavaScript Framework. *vuejs.org*, 2024. — Режим доступу: https://vuejs.org

25. W3C. Web Accessibility Initiative: Accessible Web Development. *w3.org*, 2023. — Режим доступу: https://www.w3.org/WAI

26. Wilson, C. *Modern Web Development with Svelte and Vite*. — Packt Publishing, 2023. — 380 с. — Режим доступу: https://www.packtpub.com/product/modern-web-development-with-svelte-and-vite/9781803238425

27. Zak, M. Progressive web apps (PWA) in 2025: Trends and technologies. *IEEE Software*, 2025, 42(1), 34–41. — Режим доступу: https://doi.org/10.1109/MS.2024.3467890

28. Zakas, N. C. *Understanding ECMAScript 6: The Definitive Guide for JavaScript Developers*. — No Starch Press, 2016. — 352 с. — Режим доступу: https://leanpub.com/understandinges6

БІБЛІОГРАФІЧНА ДОВІДКА

Тема бакалаврської роботи: " Порівняльний аналіз технологій та фреймворків ВЕБ-розробки"

Обсяг пояснювальної записки: 70 аркушів

Дата закінчення дипломної роботи 10 червня 2024р.

Підпис студента _____