Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

ДОСЛІДЖЕННЯ ТА ПОРІВНЯЛЬНИЙ АНАЛІЗ JAVASCRIPT-ФРЕЙМВОРКІВ ДЛЯ РОЗРОБЛЕННЯ ВЕБЗАСТОСУНКІВ
(тема)

Виконав:
студент 2 курсу, групи ІНФм-22-3

Ткачов В.А.
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник доц. ТітоваО.В.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

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

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Ткачову Володимиру Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження та порівняльний аналіз JavaScript-фреймворків для розроблення вебзастосунків

затверджена наказом по університету від 3 листопада 2023 року № 1280Ст

2. Термін подання студентом роботи до екзаменаційної комісії _1_ січня 2023 р.

3. Вихідні дані до роботи офіційні документації фреймворків: статистичні дані з аналітичних платформ, репозиторії з прикладами коду та проектами-демонстраціями.

4. Перелік питань, що потрібно опрацювати в роботі

1. Визначити ключові критерії оцінювання фреймворків та розробити методологію для порівняльного аналізу.
2. Провести аналіз фреймворків шляхом створення тестових вебзастосунків.
3. Порівняти фреймворки за встановленими критеріями, включаючи практичне застосування фреймворків у реальних проєктах.
4. Надати рекомендації щодо вибору фреймворку в залежності від конкретних потреб проєкту.
5. Сформулювати пропозиції щодо можливих напрямків подальших досліджень у сфері веброзробки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) актуальність проблеми вибору фреймворку для побудови вебзастосунків, постановка задачі, тестові зображення.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	03.11.2023	
2	Аналіз завдання, підбір літератури	04.11.23-10.11.23	
3	Аналіз літератури з досліджуваної проблеми	11.11.23-20.11.23	
4	Вибір параметрів для аналізу	21.11.23-30.11.23	
5	Розробка методів порівняльного аналізу	01.12.23-09.12.23	
6	Програмна реалізація	10.12.23-20.12.23	
7	Оформлення пояснювальної записки	20.12.23-23.12.23	
8	Перевірка на плагіат	24.12.2023	
9	Рецензування	25.12.2023	
10	Підготовка презентації та доповіді	26.12.2023	
11	Занесення роботи в електронний архів	10.01.2024	
12	Попередній захист кваліфікаційної роботи	10.01.2024	

Дата видачі завдання 3 листопада 2023 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Тітова О.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 60 с., 4 табл., 5 рис. та 34 джерела.

JAVASCRIPT-ФРЕЙМВОРКИ, REACT, ANGULAR, VUE.JS, ВЕБЗАСТОСУНКИ, ПОРІВНЯЛЬНИЙ АНАЛІЗ, ЕКОСИСТЕМА, РОЗРОБКА ВЕБЗАСТОСУНКІВ, ВИБІР ФРЕЙМВОРКУ.

Об’єктом дослідження є популярні JavaScript фреймворки, такі як React, Angular, Vue.js та Ember.js.

Метою дослідження є розробка рекомендацій щодо застосування JavaScript-фреймворків для розроблення вебзастосунків.

Результати аналізу можуть допомогти розробникам і компаніям вибрати найбільш підходящий фреймворк для своїх проєктів в залежності від конкретних вимог та завдань. В роботі проведено оцінку основних переваг і недоліків, продуктивності, легкості вивчення та інших аспектів популярних JavaScript-фреймворків.

Загалом, дослідження та порівняльний аналіз JavaScript-фреймворків мають велике значення для розробників, оскільки вони допомагають зрозуміти переваги та обмеження кожного фреймворку і прийняти обгрунтоване рішення при розробці вебзастосунків.

JAVASCRIPT FRAMEWORKS, REACT, ANGULAR, VUE.JS, WEB APPLICATIONS, BENCHMARKING, ECOSYSTEM, WEB APPLICATION DEVELOPMENT, FRAMEWORK CHOICE.

The object of research is popular JavaScript frameworks such as React, Angular, Vue.js and Ember.js.

The purpose of the study is to develop recommendations for the use of JavaScript frameworks for the development of web applications.

The results of the analysis can help developers and companies choose the most suitable framework for their projects, depending on specific requirements and tasks. The work evaluates the main advantages and disadvantages, productivity, ease of learning and other aspects of popular JavaScript frameworks.

Overall, JavaScript framework research and benchmarking is of great value to developers as it helps them understand the strengths and limitations of each framework and make an informed decision when developing web applications.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

JS — JavaScript (джаваскрипт)

HTML — HyperText Markup Language (мова розмітки гіпертексту)

CSS — Cascading Style Sheets (каскадні таблиці стилів)

SPA — Single Page Application (односторінковий застосунок)

DOM — Document Object Model (об’єктна модель документа)

API — Application Programming Interface (інтерфейс програмування застосунків)

MVC — Model-View-Controller (модель-вид-контролер)

AJAX — Asynchronous JavaScript and XML (асинхронний JavaScript та XML)

JSON — JavaScript Object Notation (нотація об’єктів JavaScript)

HTTP — HyperText Transfer Protocol (протокол передачі гіпертексту)

HTTPS — HyperText Transfer Protocol Secure (безпечний протокол передачі гіпертексту)

URL — Uniform Resource Locator (єдиний локатор ресурсів)

UI — User Interface (інтерфейс користувача)

UX — User Experience (досвід користувача)

SSR — Server-Side Rendering (рендеринг на стороні сервера)

CSR — Client-Side Rendering (рендеринг на стороні клієнта)

CLI — Command Line Interface (інтерфейс командного рядка)

IDE — Integrated Development Environment (нтегроване середовище розробки)

GIT — Distributed Version Control System (розподілена система контролю версій)

GPU — Graphics Processing Unit (графічний процесор)

CPU — Central Processing Unit (центральний процесор)

RAM — Random Access Memory (оперативна пам’ять)

VDOM — Virtual DOM (віртуальний DOM)

OOP — Object-Oriented Programming (об’єктно-орієнтоване програмування)

SASS/SCSS — Syntactically Awesome Stylesheets/Sassy CSS (синтаксично чудові таблиці стилів/синтаксично чудовий CSS)

LESS — Leaner Style Sheets (таблиці стилів з меншою кількістю коду)

Webpack — Module Bundler (збірник модулів)

TypeScript — Superset of JavaScript (підмножина JavaScript)

ВСТУП

У сучасному світі веброзробки, де потреби та очікування користувачів до швидкості, ефективності та зручності вебзастосунків невпинно зростають, вибір відповідного JavaScript-фреймворку стає ключовим рішенням для успішної розробки. JavaScript-фреймворки, такі як React, Angular, Vue.js та Ember.js, пропонують унікальні підходи до розроблення вебзастосунків, кожен з яких має свої переваги та недоліки. Розробники стикаються з викликом обрати фреймворк, який найкраще відповідає специфічним вимогам проєкту, беручи до уваги чинники такі як продуктивність, гнучкість, легкість навчання, величина спільноти та доступні ресурси.

Ця кваліфікаційна робота спрямована на дослідження та порівняльний аналіз вищезазначених JavaScript-фреймворків для розроблення вебзастосунків. Вона покликана оцінити кожен фреймворк з різних аспектів, включаючи архітектуру, продуктивність, підтримку спільноти, вартість навчання та інтеграції, та розглянути реальні сценарії використання, де кожен з них може виявитися найбільш вигідним.

Актуальність дослідження полягає у необхідності глибокого аналізу технічних характеристик та властивостей популярних JavaScript-фреймворків, а також у визначенні їхньої придатності до вирішення різних задач веброзробки. Із швидким розвитком технологій, поява нових версій фреймворків, і, як наслідок, зміна їхніх можливостей, зростає важливість постійного оновлення знань розробників. Таке дослідження дозволяє не тільки систематизувати існуючі дані, а й пролити світло на перспективи розвитку вебтехнологій, сприяти формуванню об’єктивного бачення сучасного стану веброзробки, та надати практичні рекомендації щодо вибору інструментарію для нових проєктів.

1 ТЕОРЕТИЧНІ ОСНОВИ ВЕБРОЗРОБКИ

1.1 Історія та розвиток JavaScript-фреймворків

Історія та розвиток JavaScript-фреймворків є захоплюючою темою, яка відображає еволюцію вебтехнологій та постійне прагнення розробників до покращення продуктивності, доступності та ефективності вебзастосунків. Давайте розглянемо цю тему детальніше.

JavaScript, спочатку створений для додавання інтерактивності на вебсторінки, швидко став популярним завдяки своїй гнучкості та простоті використання. Однак з розвитком вебтехнологій з’явилася потреба в більш структурованих підходах до розробки вебзастосунків, що призвело до появи перших фреймворків [1].

jQuery, запущений у 2006 році, став революційним інструментом у розробці вебзастосунків, надаючи простий та зручний спосіб для взаємодії з HTML DOM, обробки подій, анімації та Ajax-запитів. Його легкість у використанні та висока сумісність з різними браузерами зробили jQuery дуже популярним серед розробників. Фреймворк значно спростив маніпуляції з DOM, дозволяючи розробникам легко змінювати елементи, стилі та вміст сторінки. jQuery також впровадив систему плагінів (pluggins), що дозволила спільноті створювати та ділитися різноманітними розширеннями. Незважаючи на свою популярність у минулому, сьогодні використання jQuery зменшилося через появу сучасних JavaScript-фреймворків і поліпшення нативних можливостей JavaScript та CSS. Однак jQuery все ще використовується в багатьох існуючих проектах і залишається важливою частиною веброзробки [2].

MooTools (My Object-Oriented Tools) і Prototype є двома з ранніх JavaScript-бібліотек, які були популярними у середині 2000-х років, перш ніж виникли сучасні JavaScript-фреймворки.

MooTools є об’єктно-орієнтованою бібліотекою, яка розширює JavaScript, надаючи додаткові функції та утиліти. Вона пропонує покращену роботу з DOM, анімації, Ajax-запити та більш потужну обробку подій. MooTools також включає систему класів, яка дозволяє використовувати об’єктно-орієнтовані патерни в JavaScript. Однак з часом популярність MooTools знизилася через зростання інших фреймворків та стандартизацію JavaScript.

Prototype є іншою ранньою JavaScript-бібліотекою, яка надавала розширені можливості для роботи з DOM, обробки подій і Ajax-запитів. Prototype відома своїм впровадженням функціональності класів у JavaScript та розширенням вбудованих JavaScript-об’єктів. Вона значно спрощувала розробку JavaScript, але, як і MooTools, згодом поступилася місцем сучаснішим фреймворкам та змінам у стандартах веброзробки.

Обидві ці бібліотеки мали значний вплив на ранній розвиток JavaScript як мови для клієнтської сторони, але з часом їх роль у веброзробці стала менш помітною.

Angular був створений і випущений командою розробників з Google у 2010 році. Перша версія Angular, відома як AngularJS, була розроблена Мішком Хевері та Адамом Абронсом. Ідея полягала в тому, щоб спростити розробку вебзастосунків, використовуюч-и шаблони HTML з вбудованим JavaScript-кодом. AngularJS запровадив ряд інноваційних функцій, таких як двостороннє зв’язування даних, впровадження залежностей та модульність, які відразу знайшли відгук серед розробників.

З часом AngularJS зазнав значних змін, і у 2016 році була представлена Angular 2, яка була повністю переписана з використанням новітніх стандартів та підходів. Angular 2+ (часто просто називаний Angular) використовує TypeScript замість JavaScript, що забезпечує більшу стабільність і безпеку типів. Нова версія також відмовилася від концепції $scope та контролерів, замінивши їх компонентами та сервісами. З того часу Angular продовжує розвиватися, з кожним релізом вносячи покращення в продуктивність,

удосконалюючи API і додаючи нові функції для спрощення розробки великих і складних вебзастосунків. Його стабільність, потужність і підтримка від Google зробили Angular одним з ведучих рішень для професійної веброзробки [3].

Backbone.js, фреймворк, який зіграв ключову роль у розвитку підходів до фронтенд-розробки, був створений Джеремі Ешкенасом і вперше випущений у 2010 році. Цей фреймворк став одним з перших, що надав структуру для розробки складних односторінкових застосунків (SPA), використовуючи модель MVC (Model-View-Controller). Backbone.js вирізнявся своєю легкістю та гнучкістю, дозволяючи розробникам легко створювати застосунки з чітко визначеною структурою.

Backbone.js запровадив концепцію моделей для управління даними, видів для управління інтерфейсом користувача та колекцій для роботи з групами моделей. Це дозволило розробникам відокремлювати бізнес-логіку від інтерфейсу користувача, покращуючи організацію коду та полегшуючи його розширення та підтримку. Backbone також надав механізми для спрощення роботи з RESTful API, що сприяло взаємодії між фронтендом та бекендом [4].

Хоча сьогодні Backbone.js не є настільки популярним, як новіші фреймворки, він мав значний вплив на фронтенд-розробку та надихнув багато ідей, які були реалізовані у сучасних фреймворках і бібліотеках. Використання Backbone.js в проектах дозволило розробникам краще зрозуміти важливість структури та організації у великих вебзастосунках.

React був створений інженером Facebook Джорданом Валке. Його вперше було представлено у 2013 році як відповідь на потребу у більш ефективному та гнучкому способі побудови інтерфейсів користувача, особливо в умовах динамічних вебзастосунків. React пропонував інноваційний підхід до створення користувацьких інтерфейсів через компонентну архітектуру, що дозволяє розробникам створювати взаємодіючі UI-елементи, які можуть бути легко перевикористані та тестовані.

Однією з ключових особливостей React став віртуальний DOM, що дозволяє оптимізувати взаємодію з реальним DOM браузера, підвищуючи продуктивність та реактивність застосунків. React також впровадив JSX, синтаксис, який поєднує HTML та JavaScript, роблячи код більш зрозумілим та читабельним.

З часом React набув величезної популярності та визнання в спільноті розробників завдяки своїй гнучкості, продуктивності та широкій екосистемі. Розвиток React сприяв появі таких проектів, як React Native для розробки мобільних застосунків та Redux як шаблон управління станом, що додатково розширило можливості використання React. Сьогодні React залишається одним з провідних інструментів для створення динамічних та відгукових вебзастосунків [5].

Vue.js був створений Еваном Ю в 2014 році. Еван працював у Google над AngularJS-проектами та хотів створити щось легше і більш гнучке. Він започаткував Vue як легковаговий фреймворк, спрямований на поєднання кращих аспектів Angular (декларативні шаблони) та React (реактивні компоненти).

Vue швидко здобув популярність завдяки своїй простоті, гнучкості та інтуїтивно зрозумілому API. Він пропонував легкий спосіб створення компонентів, реактивність та двостороннє зв’язування даних, що робило його ідеальним для розробки прогресивних вебзастосунків [6].

Vue не вимагає використання всього фреймворку, що дозволяє легко інтегрувати його у існуючі проекти. Крім того, Vue.js має активну міжнародну спільноту та широку екосистему, яка включає інструменти та бібліотеки, такі як Vuex для управління станом та Vue Router для маршрутизації.

Завдяки своїй легкості, деталізованій документації та підтримці спільноти, Vue.js став одним з найпопулярніших фреймворків для фронтенд-розробки.

Ember.js, фреймворк для розробки вебзастосунків, був створений Яхудою Кацем та Томом Дейлом та вперше випущений у 2011 році. Ember

зародився з проекту SproutCore 2.0, від якого він успадкував концепцію розширеного використання шаблонів та автоматичного оновлення інтерфейсу користувача відповідно до змін у моделях даних.

З самого початку Ember був спрямований на надання повного набору інструментів для розробки, включаючи двостороннє зв’язування даних, автоматичне відстеження залежностей, шаблонізацію та систему маршрутизації. Це дозволяло створювати масштабовані та структуровані вебзастосунки.

Ember відрізнявся своїм підходом «конвенція над конфігурацією», забезпечуючи стандартні патерни та практики, які зменшували кількість рутинного коду та спрощували процес розробки. Фреймворк також включав Ember CLI, потужний інструмент командного рядка для автоматизації процесів збірки та розгортання застосунків.

З часом Ember.js продовжив розвиватися, фокусуючись на продуктивності, сучасних вебстандартах та зручності розробників. Його спільнота активно підтримує проект, вносячи оновлення та покращення, що забезпечують його актуальність у світі сучасної веброзробки [7].

З’явилися також нові інструменти, як Svelte, новаторський фреймворк для веброзробки, був створений Річем Харрісом і вперше представлений у 2016 році. Він виник як альтернатива традиційним фреймворкам, таким як React та Vue, з ідеєю значно зменшити кількість коду, який потрібно писати та відправляти браузеру.

Особливістю Svelte є те, що він не використовує віртуальний DOM. Замість цього, Svelte компілює компоненти в оптимізований ванільний JavaScript на етапі збірки, що забезпечує високу продуктивність та меншу вагу кінцевого застосунку. Такий підхід дозволяє Svelte здійснювати реактивність на рівні мови, не потребуючи додаткових бібліотек чи фреймворків.

Svelte забезпечує розробникам можливість створювати вебінтерфейси з меншими зусиллями, оскільки він автоматично управляє всіма оновленнями

DOM відповідно до змін у стані застосунку. Це робить код більш декларативним та легким для читання та підтримки.

З часом Svelte набув популярності серед розробників завдяки своїм інноваційним особливостям, мінімалістичному підходу та здатності створювати швидкі вебзастосунки з меншим обсягом коду. Розвиток Svelte показує тенденцію у веброзробці до створення більш ефективних і легких застосунків.

Ця еволюція відображає непостійність у сфері веброзробки та постійну потребу адаптуватися до нових вимог, високої продуктивності, зручності користування та оптимізації процесів.

1.2 Основні концепції сучасних JavaScript-фреймворків

Сучасні JavaScript-фреймворки надають розробникам різноманітні інструменти та абстракції для ефективної та швидкої розробки вебзастосунків. Нижче представлено деякі з ключових концепцій, які є фундаментом більшості сучасних JavaScript-фреймворків:

– компонентний підхід у сучасних JavaScript-фреймворках передбачає розбиття інтерфейсу на відокремлені, перевикористовувані блоки, відомі як компоненти. Кожен компонент капсулює власну HTML-структуру, стилі CSS та логіку JavaScript. Це дозволяє розробникам управляти складними інтерфейсами як набором незалежних частин, що спрощує розробку та підтримку коду. Компоненти можуть містити інші компоненти, утворюючи ієрархічні структури, що відображають взаємозв’язки та логіку інтерфейсу. Це дозволяє створювати складні, але добре структуровані інтерфейси користувача. Компонентний підхід також сприяє перевикористанню коду, оскільки розробники можуть використовувати одні й ті ж компоненти у різних частинах застосунку або навіть у різних проектах. У різних фреймворках концепція компонентів реалізована по-різному, але основна ідея залишається

однаковою: створення інтерфейсу користувача як композиції взаємодіючих частин, що забезпечує модульність, легкість тестування та підтримки;

– декларативне програмування у сучасних JavaScript-фреймворках полягає в описі того, що має бути зроблено, а не як це має бути зроблено. Ця концепція фокусується на описі логіки інтерфейсу користувача без вказівки детальних кроків для його досягнення. Фреймворки використовують декларативні шаблони, які автоматично оновлюють інтерфейс користувача при зміні стану застосунку. Це забезпечує більш чистий та зрозумілий код, зменшуючи ймовірність помилок і спрощуючи розробку. Декларативне програмування дозволяє розробникам абстрагуватися від низькорівневих маніпуляцій DOM і зосередитися на визначенні вищих рівнів логіки за допомогою шаблонів і компонентів. Це сприяє створенню більш підтримуваного та масштабованого коду, а також підвищує продуктивність розробників;

– реактивність та двостороннє зв’язування в сучасних JavaScript-фреймворках є ключовими концепціями, які забезпечують динамічну взаємодію між даними та інтерфейсом користувача. Реактивність означає, що фреймворки автоматично відстежують зміни в даних та оновлюють інтерфейс, не потребуючи від розробників явного втручання для оновлення DOM. Двостороннє зв’язування даних, яке є важливою особливістю Angular та використовується в деяких інших фреймворках, дозволяє синхронізувати дані між моделлю (JavaScript-об’єктом) та представленням (шаблоном HTML). Коли дані в моделі змінюються, представлення автоматично оновлюється, і навпаки, коли користувач взаємодіє з формою чи інтерфейсом, модель оновлюється відповідно. Ці концепції сприяють створенню інтерактивних, динамічних вебзастосунків, де стан застосунку постійно синхронізується між користувачем та сервером. Реактивність та двостороннє зв’язування даних спрощують управління складними інтерфейсами, роблячи код більш інтуїтивно зрозумілим та легшим для підтримки;

– віртуальний DOM (VDOM) є фундаментальною концепцією в сучасних JavaScript-фреймворках, особливо в React. Віртуальний DOM — це легковагова копія реального DOM, яка дозволяє оптимізувати взаємодію з браузером та покращити продуктивність застосунку. Коли стан застосунку змінюється, замість того, щоб відразу оновлювати реальний DOM, фреймворк спочатку робить зміни у віртуальному DOM (рис 1.1).

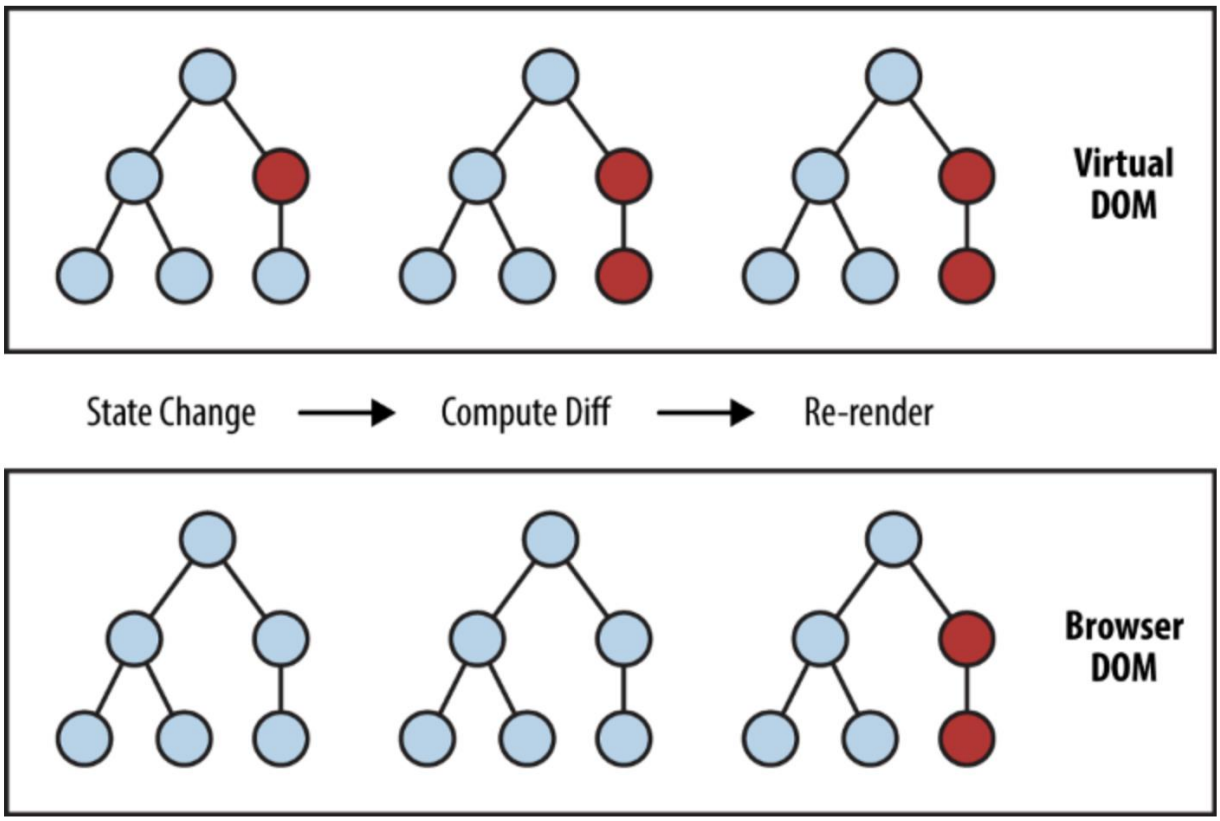

Рисунок 1.1 – Принцип роботи віртуального DOM

Потім, використовуючи алгоритм порівняння, фреймворк визначає різницю між попереднім та поточним станами віртуального DOM. На основі цієї різниці фреймворк оновлює тільки ті частини реального DOM, які змінилися, замість того, щоб оновлювати весь DOM цілком. Цей підхід забезпечує значну ефективність, оскільки маніпуляції з реальним DOM є відносно повільними та витратними з точки зору продуктивності. Віртуальний DOM дозволяє мінімізувати кількість маніпуляцій з DOM та знизити вплив на

продуктивність, особливо при роботі з великими та складними інтерфейсами користувача;

– односторонній потік даних у сучасних JavaScript-фреймворках є ключовою концепцією, що сприяє зрозумілості та передбачуваності архітектури застосунків. Цей підхід означає, що дані в застосунку переміщуються в одному напрямку - від батьківських компонентів до дочірніх. У такій архітектурі стан застосунку зберігається в певних «джерелах правди» (наприклад, у кореневих компонентах або в спеціалізованих стейт-менеджерах (state managers), як Redux). Ці джерела передають дані вниз до дочірніх компонентів через пропси (props). Дочірні компоненти можуть змінювати ці дані, але замість безпосереднього оновлення стану вони використовують колбеки (callbacks) чи події, щоб повідомити батьківські компоненти про необхідність змін. Це дозволяє створювати більш передбачувані та легкокеровані застосунки, оскільки знижується складність відстеження змін у стані та взаємодій між компонентами. Крім того, односторонній потік даних сприяє більш ефективному розподілу відповідальності між компонентами та спрощує процес тестування та дебагінгу (debugging) [8];

– TypeScript як надмножина JavaScript став важливою частиною сучасних JavaScript-фреймворків завдяки введенню строгої типізації та об’єктно-орієнтованих можливостей. Це забезпечує розробникам інструменти для створення більш надійного, підтримуваного та масштабованого коду. Використання TypeScript дозволяє виявляти помилки пов’язані з типами даних на ранніх етапах розробки, покращує автодоповнення коду і рефакторинг. Інтеграція TypeScript з фреймворками, як Angular, де він використовується як основна мова, а також його підтримка в React і Vue, робить його ідеальним вибором для розробки сучасних вебзастосунків. TypeScript сприяє створенню більш структурованого та легко читабельного коду, що особливо важливо у великих проектах та командах [9];

– модульність у сучасних JavaScript-фреймворках є ключовою концепцією, яка дозволяє розбивати великі застосунки на менші, незалежні

частини, звані модулями або компонентами. Ця концепція допомагає розробникам утримувати код організованим, зробити його більш читабельним, легшим для тестування та підтримки.

Кожен модуль або компонент у фреймворках, як React, Vue, чи Angular, ізолює певну функціональність або логіку застосунку, містячи відповідний HTML, CSS та JavaScript. Це сприяє перевикористанню коду, оскільки модулі можуть бути легко імпортовані та використані у різних частинах застосунку. Модульність також полегшує управління залежностями у застосунку, дозволяючи розробникам імпортувати лише ті функції чи бібліотеки, які необхідні для конкретного модуля. Це зменшує розмір загального застосунку та підвищує його продуктивність. У сучасних фреймворках модульність також підтримується на рівні інструментів збірки та розгортання, які дозволяють оптимізувати застосунки, видаляючи невикористані модулі та забезпечуючи ефективну загрузку необхідних частин.Таким чином, модульність у сучасних JavaScript-фреймворках є важливою для створення ефективних, легко підтримуваних та масштабованих вебзастосунках;

– Server-Side Rendering та Static Site Generation: SSR і SSG є важливими концепціями у сучасних JavaScript-фреймворках, що підвищують продуктивність та оптимізують SEO (пошукову оптимізацію). SSR полягає у генерації повного HTML-коду сторінки на сервері перед тим, як він буде відправлений до клієнта. Це особливо корисно для поліпшення швидкості першого відображення контенту та оптимізації для пошукових систем, оскільки пошукові роботи легше індексують сторінки, що вже містять увесь необхідний контент. SSR часто використовується у фреймворках, як Next.js для React або Nuxt.js для Vue. SSG передбачає генерацію статичних HTML-сторінок під час збірки проекту. Відмінність від SSR полягає у тому, що сторінки генеруються заздалегідь, а не при кожному запиті користувача. Це забезпечує ще вищу швидкість завантаження та ефективність, оскільки серверу потрібно лише відправити вже готовий HTML. SSG ідеально підходить для блогів, документації та інших проектів, де контент не

змінюється динамічно. Обидві ці техніки дозволяють розробникам оптимізувати застосунки для кращого досвіду користувачів та пошукової оптимізації, вибираючи найбільш підходящий підхід в залежності від специфіки проекту [10];

– Client-Side Rendering є основною концепцією у сучасних JavaScript-фреймворках. Ця техніка передбачає, що весь контент вебсторінки генерується в браузері користувача за допомогою JavaScript. В CSR, коли користувач запитує сторінку, сервер надсилає лише мінімальний HTML-скелет разом з JavaScript-файлами. Після завантаження цих файлів, JavaScript виконується у браузері, генеруючи вміст сторінки і динамічно взаємодіючи з DOM. Це означає, що вся важка робота з рендерингу переноситься на клієнтську сторону. Переваги CSR включають гнучкість та потужність динамічного оновлення контенту, здатність створювати швидкі та плавні користувацькі інтерфейси, а також зниження навантаження на сервер. Однак, це також може призвести до повільнішого завантаження першого відображення сторінки, оскільки потрібен час на завантаження та виконання JavaScript-коду, а також може створити проблеми з SEO, оскільки пошукові роботи можуть мати проблеми з індексацією динамічно генерованого контенту. CSR є важливою частиною багатьох сучасних вебзастосунків, пропонуючи високий рівень інтерактивності та користувацького досвіду, характерного для «багатих» клієнтських застосунків;

– оптимізація продуктивності в сучасних JavaScript-фреймворках зосереджується на забезпеченні швидкого та ефективного виконання вебзастосунків. Це досягається через ряд стратегій та практик. Мінімізація часу завантаження досягається завдяки мініфікації файлів, оптимізації зображень та використанню Content Delivery Network. Ліниве завантаження дозволяє завантажувати компоненти лише за потреби, зменшуючи початкове навантаження. Віртуальний DOM оптимізує взаємодію з реальним DOM, що покращує продуктивність. Код-спліттінг (code splitting) ділить застосунок на менші частини, які завантажуються за потреби, зменшуючи початковий обсяг

завантаження. Важливо також оптимізувати управління пам’яттю та ресурсами для запобігання витоків пам’яті та забезпечення ефективного використання ресурсів. Всі ці методи сприяють створенню швидких та ефективних вебзастосунків.

Ці концепції демонструють сучасний тренд до створення більш ефективних, інтуїтивно зрозумілих та масштабованих вебзастосунків. Розуміння та правильне застосування цих концепцій є ключовими для успішної розробки в епоху сучасних вебтехнологій.

1.3 Критерії вибору JavaScript-фреймворку

Критерії вибору JavaScript-фреймворку залежать від специфіки проєкту, команди розробників та вимог до функціональності й продуктивності застосунку. Ось основні аспекти, які слід враховувати при виборі фреймворку:

– специфіка проекту впливає на вибір JavaScript-фреймворку, оскільки різні проекти мають різні технічні вимоги та цілі. Для складних односторінкових застосунків, які потребують багатого на функції інтерактивного користувацького інтерфейсу, підходять фреймворки, що пропонують детальне управління станом і маршрутизацією, такі як Angular або React. Якщо проект має бути більш простим, з меншою кількістю інтерактивності, можна вибрати легший фреймворк, наприклад Vue. Також важливо враховувати, чи проект буде масштабуватися в майбутньому, адже це впливає на вибір фреймворку, здатного підтримувати розширення функціональності без втрати продуктивності. Якщо проект має унікальні вимоги до взаємодії з сервером або інтеграції з іншими системами, це також впливає на вибір фреймворку, що найкраще підходить для таких завдань. Важливо також врахувати часові рамки розробки та ресурси, доступні для проекту, оскільки деякі фреймворки можуть вимагати більше часу на навчання та розробку;

– зрілість і стабільність фреймворку є важливими чинниками при виборі технологій для проекту. Зрілість означає, що фреймворк має довгу історію розвитку, в ході якої були виправлені помилки, додані нові можливості та оптимізовані існуючі функції. Це також свідчить про наявність великої спільноти розробників та користувачів, що може забезпечити підтримку та велику кількість ресурсів для навчання. Стабільність фреймворку підкреслює його надійність та прогнозованість. Стабільний фреймворк має менше ризиків внесення руйнівних змін у майбутніх оновленнях, що забезпечує безперебійну роботу існуючих застосунків. Також стабільність важлива для довгострокового планування, оскільки дає впевненість у тому, що використовувана технологія не стане застарілою або не підтримуваною в найближчому майбутньому. Вибір зрілого та стабільного фреймворку є особливо важливим для великих проектів з довгим життєвим циклом, де необхідна надійність та можливість підтримки на протязі тривалого часу. Однак, слід також враховувати, що деякі молодші технології можуть пропонувати інноваційні підходи та переваги, які варто розглянути залежно від специфіки проекту;

– продуктивність є ключовим фактором при виборі JavaScript-фреймворку, оскільки це безпосередньо впливає на швидкість роботи застосунку та досвід користувача. Важливо оцінити, наскільки ефективно фреймворк може обробляти DOM-операції, управляти станом застосунку та реагувати на користувацькі дії.

Фреймворки, які використовують віртуальний DOM забезпечують більш високу продуктивність, особливо у складних інтерфейсах. Також важливою є ефективність управління станом та реактивність даних, оскільки це впливає на швидкість відгуку застосунку на дії користувачів.

Фреймворки з механізмами для лінивого завантаження (lazy loading) та код-спліттінгу (code splitting) дозволяють зменшити початкове навантаження, завантажуючи лише необхідний код, що важливо для швидкості завантаження

сторінки. Іншим важливим аспектом є оптимізація збірки застосунку, включаючи мініфікацію та усунення невикористаного коду.

Продуктивність також залежить від специфіки проекту. Наприклад, для проектів з високою кількістю динамічних взаємодій користувача може знадобитися фреймворк із сильними реактивними можливостями, тоді як для простіших вебсайтів може підійти легший фреймворк.

Вибір фреймворку залежить від балансу між продуктивністю та іншими необхідними характеристиками, які важливі для конкретного проекту;

– гнучкість та масштабованість фреймворку важливі для адаптації до зростаючих та змінних вимог проекту. Гнучкість означає можливість фреймворку інтегруватися з різними бібліотеками та інструментами, а також його спроможність використовуватися в широкому спектрі сценаріїв розробки. Масштабованість відноситься до здатності фреймворку підтримувати розширення функціональності та користувацької бази без втрати продуктивності чи якості застосунку.

Фреймворки з високою гнучкістю дозволяють розробникам вибирати найкращі інструменти для кожної задачі, забезпечуючи оптимальне рішення для конкретних потреб проекту. Це також включає можливість легко адаптувати архітектуру застосунку згідно зі змінами у вимогах або технологіях. Масштабованість забезпечує, що застосунок може рости та розвиватися, збільшуючи кількість користувачів, даних або транзакцій, без необхідності повної переробки чи внесення значних змін у код. Важливо, щоб фреймворк міг ефективно обробляти збільшення навантаження та володів гнучкою архітектурою, що дозволяє розширювати та модифікувати компоненти застосунку.

Вибір фреймворку, який забезпечує як гнучкість, так і масштабованість, є критичним для довгострокового успіху проекту, особливо в динамічному світі веброзробки, де технології та вимоги користувачів швидко змінюються;

– екосистема фреймворку є критично важливою для ефективності розробки та можливостей масштабування проекту. Екосистема включає в себе

наявні бібліотеки, плагіни (pluggins), інструменти, розширення та інтеграції, які забезпечують додаткову функціональність та полегшують процес розробки.

Багата екосистема означає, що розробникам часто не потрібно писати все з нуля, оскільки вони можуть використовувати існуючі рішення для загальних задач. Це може включати утиліти для роутингу, управління станом, зв’язування даних, тестування, а також інтеграцію з іншими системами та API.

Фреймворки з активною та великою спільнотою зазвичай мають розвинену екосистему з великою кількістю доступних ресурсів. Це також включає наявність добре документованих бібліотек та рішень, які розробники можуть використовувати для прискорення процесу розробки.

Така екосистема також забезпечує більше можливостей для навчання та підтримки, оскільки розробники можуть легко знайти документацію, підручники, курси та спільноти, де вони можуть отримати допомогу або обговорити проблеми.

Вибір фреймворку з багатою та активною екосистемою може значно вплинути на швидкість розвитку проекту, його якість та легкість підтримки в майбутньому;

– легкість навчання фреймворку важлива для швидкої адаптації розробників та ефективного впровадження технології в проект. Фреймворки з низьким порогом входження та простими концепціями дозволяють розробникам швидше почати роботу над проектом, навіть якщо вони не мають глибоких знань у цій технології.

Легкість навчання залежить від інтуїтивності API та ясності концепцій, які лежать в основі фреймворку. Наприклад, Vue.js часто вважається інтуїтивно зрозумілим, оскільки його дизайн та архітектура зроблені максимально простими та зрозумілими.

Наявність чіткої, добре структурованої документації та навчальних матеріалів сприяє легшому вивченню фреймворку. Це включає докладні посібники, підручники, зразки коду та навчальні відео.

Для розробників, які вже знайомі з іншими JavaScript-фреймворками або бібліотеками, важливо, наскільки легко їм буде перейти на новий фреймворк. Наприклад, якщо розробник знає React, вивчення Vue або Angular може бути спрощеним завдяки схожим концепціям.

Легкість навчання фреймворку має значний вплив на швидкість розвитку проекту та здатність швидко адаптувати нові технології;

– вбудовані функції та можливості: автоматизація звичайних завдань (наприклад, маршрутизація, управління станом), вбудовані інструменти для тестування та відлагодження;

– технічні характеристики: модель даних і зв’язування, підтримка серверного рендерингу (SSR) і статичної генерації сторінок (SSG), використання TypeScript або інших статично-типізованих мов;

– ліцензування фреймворку є важливим аспектом, який слід враховувати при його виборі, оскільки це визначає, як можна використовувати фреймворк та розповсюджувати продукти, створені з його допомогою.

Фреймворки зазвичай розповсюджуються під відкритими ліцензіями, такими як MIT або Apache 2.0, які дозволяють використовувати, змінювати та розповсюджувати фреймворк без обмежень та без вимоги плати за ліцензію. Ці ліцензії забезпечують високий рівень гнучкості та сприяють широкому використанню фреймворку у комерційних та некомерційних проектах.

Важливо розуміти умови ліцензії фреймворку, особливо в контексті комерційного використання. Наприклад, деякі ліцензії можуть вимагати збереження авторських прав та посилань на авторів у вихідному коді або документації продукту.

Крім того, у разі використання фреймворків з відкритим кодом в комерційних продуктах, важливо звернути увагу на сумісність ліцензій фреймворку з іншими використовуваними бібліотеками та інструментами, щоб уникнути юридичних конфліктів.

Таким чином, ліцензування може мати значний вплив на вибір фреймворку, особливо у випадках, коли розробляються продукти, що

потенційно можуть бути комерціалізовані або використовуватися у великих масштабних проектах.

Ці критерії допоможуть обрати найбільш підходящий JavaScript-фреймворк, відповідно до специфічних потреб проєкту та можливостей команди розробників.

1.4 Постановка задачі дослідження

Ціль дослідження полягає у проведенні глибокого аналізу та порівняльному оцінюванні популярних JavaScript-фреймворків, таких як React, Angular, Vue.js та Ember.js, для визначення їх придатності до розроблення вебзастосунків різної складності та обсягу. Аналіз має базуватися на критичному розгляді основних характеристик кожного фреймворку, включаючи продуктивність, гнучкість, масштабованість, спільноту та підтримку, екосистему, легкість навчання та інші ключові параметри.

Об’єктом дослідження є популярні JavaScript фреймворки, такі як React, Angular, Vue.js та Ember.js.

Метою дослідження є розробка рекомендацій щодо застосування JavaScript-фреймворків для розроблення вебзастосунків.

Для досягнення мети необхідно вирішити такі завдання:

– проаналізувати існуючу літературу, джерела та дослідження щодо застосування та особливостей кожного з розглянутих фреймворків;

– визначити ключові критерії оцінювання фреймворків та розробити методологію для порівняльного аналізу;

– провести аналіз фреймворків шляхом створення тестових вебзастосунків або аналізу існуючих проєктів;

– порівняти фреймворки за встановленими критеріями, включаючи практичне застосування фреймворків у реальних проєктах;

– визначити сильні та слабкі сторони кожного фреймворку в контексті різних типів вебзастосунків;

– надати рекомендації щодо вибору фреймворку в залежності від конкретних потреб проєкту;

– сформулювати пропозиції щодо можливих напрямків подальших досліджень у сфері веброзробки.

Результати цього дослідження мають на меті слугувати путівником для вибору найбільш ефективного інструменту для розробки вебзастосунків та допомогти розробникам у прийнятті обґрунтованих рішень під час вибору технологічного стеку.

2 ОГЛЯД JAVASCRIPT-ФРЕЙМВОРКІВ ДЛЯ РОЗРОБЛЕННЯ ВЕБЗАСТОСУНКІВ

2.1 React

2.1.1 Архітектура та основні характеристики

React є відкритим JavaScript-фреймворком від Facebook, призначеним для створення інтерфейсів користувача, особливо для односторінкових застосунків. Він дозволяє розробникам створювати великі вебзастосунки, що можуть змінювати дані без перезавантаження сторінки.

Його ключова особливість – віртуальний DOM, що дозволяє оптимізувати взаємодію з реальним DOM і покращує продуктивність застосунків. Віртуальний DOM (VDOM) в React – це один з ключових елементів, який сприяє ефективності та швидкості фреймворку. Це концепція, яка дозволяє оптимізувати оновлення інтерфейсу користувача, мінімізуючи взаємодію з реальним DOM браузера, яка є відносно повільною. Кожного разу, коли стан компоненту в React змінюється, замість безпосередньої зміни DOM, React створює нове віртуальне дерево DOM. Це нове дерево порівнюється з попередньою версією віртуального DOM, щоб визначити, які саме зміни потрібно внести в реальний DOM. Цей процес називається «reconciliation» [11]. React використовує алгоритм визначення різниці, який ефективно визначає змінені елементи. Після цього React оновлює тільки ті частини реального DOM, які зазнали змін, замість того, щоб перерендерювати весь DOM. Це значно знижує кількість маніпуляцій, які потрібно виконати, і підвищує продуктивність вебзастосунка, особливо при роботі з великими або складними інтерфейсами.Віртуальний DOM також спрощує процес розробки, оскільки розробники можуть працювати у високорівневому абстрактному API, не турбуючись про оптимізацію маніпуляцій з DOM та їхнього впливу на

продуктивність [12]. React бере на себе відповідальність за оптимальне управління цими процесами.

React використовує компонентний підхід, де інтерфейс розбивається на незалежні, перевикористовувані частини, які управляються їх власним станом. Це додає модульності та спрощує управління кодом. Кожен компонент має свій власний стан і життєвий цикл, і він відповідає за рендеринг частини UI. React використовує JSX для опису UI, що змішує HTML з JavaScript, роблячи синтаксис більш зрозумілим [13]. Компоненти приймають «пропси (props)» як параметри та використовують «стейт (state)» для управління даними, які змінюються у часі. Цей підхід полегшує управління складними застосунками, сприяє чистоті коду та його модульності. Компоненти React можуть бути функціональними або класовими, з хуками, що надають більше функціональності компонентам. Розробники використовують компонентний підхід для побудови масштабованих і ефективних вебзастосунків.

Життєвий цикл компонентів у React дозволяє виконувати код на різних етапах їх існування. Починається з монтування, коли компонент створюється і вставляється у DOM. У цей момент викликаються методи constructor для ініціалізації стану, getDerivedStateFromProps для оновлення стану перед рендерингом, render для відображення компонента та componentDidMount для виконання операцій після монтування, як от запити до сервера [14]. Під час оновлення компонента, що відбувається при зміні пропсів або стану, знову викликається getDerivedStateFromProps, потім shouldComponentUpdate вирішує, чи потрібно перерендерити компонент. Після render викликається getSnapshotBeforeUpdate перед оновленням DOM і componentDidUpdate після оновлення. Коли компонент видаляється з DOM, викликається componentWillUnmount, де можна виконати прибирання, наприклад, видаляти таймери або скасовувати запити до мережі [15]. У функціональних компонентах з хуками життєвий цикл можна емулювати за допомогою useEffect, який об’єднує можливості componentDidMount, componentDidUpdate та componentWillUnmount.

Спільнота React є однією з найбільших у світі веброзробки, що забезпечує широкий спектр плагінів (pluggins), інструментів та компонентів, готових до використання. Також існує велика кількість освітніх ресурсів, включаючи офіційну документацію, туторіали, курси та спільноти, які полегшують освоєння фреймворку.

React постійно розвивається, впроваджуючи нові функції, що сприяють більшій продуктивності та гнучкості. Хуки, введені у версії 16.8, дозволяють використовувати стан та інші реактивні функції у функціональних компонентах, зменшуючи потребу у класових компонентах. Конкурентний режим пропонує покращене управління рендерингом для забезпечення плавнішої роботи інтерфейсів. Хуки Suspense та Lazy Loading допомагають оптимізувати завантаження та відображення компонентів, знижуючи час завантаження сторінок [16]. React також фокусується на поліпшенні розробки з інструментами для дебагінгу (debugging) та профілювання. Зміни в архітектурі з Fiber покращують здатність React до обробки великої кількості даних та складних UI. Усе це робить React дуже гнучким і потужним інструментом для створення сучасних вебзастосунків. Це робить React сучасним інструментом, який відповідає потребам сучасної веброзробки і дозволяє створювати динамічні, швидкі та відгукові вебзастосунки.

2.1.2 Спільнота та екосистема

Спільнота та екосистема React мають ключове значення для успіху цього фреймворку. Ще з моменту його створення компанією Facebook, React швидко набув популярності серед розробників. Сила і підтримка спільноти React полягають у наступному:

– React є відкритим кодом, що дозволяє розробникам з усього світу вносити свій вклад у його розвиток, виявляти та виправляти помилки, а також розширювати його можливості;

– завдяки високій продуктивності та гнучкості React, багато відомих компаній, таких як Airbnb, Netflix і, звісно, Facebook, використовують його для своїх вебзастосунків;

– сильна база користувачів забезпечує великий обсяг знань та ресурсів для навчання та підтримки нових розробників;

– існує безліч ресурсів для навчання React, включаючи офіційну документацію, відеокурси, спеціалізовані книги, блоги, подкасти та інші матеріали;

– форуми, такі як Stack Overflow, Reddit, та спеціалізовані групи в Discord та Slack, забезпечують місця для обговорення, вирішення проблем та обміну досвідом;

– регулярні зустрічі та конференції, такі як React Conf, допомагають розробникам залишатися в курсі останніх новин і найкращих практик;

– існує безліч сторонніх бібліотек і розширень для React, які покращують його функціональність і спрощують розробку, включаючи стан-менеджери (Redux, MobX), роутери (React Router), інтерфейсні кіті (Material-UI, Ant Design) та інші;

– застосунки для створення проєктів, такі як Create React App та Next.js, спрощують налаштування та оптимізацію проєктів;

– React сприяє культурі співпраці та спільного використання коду, де розробники можуть ділитися компонентами та рішеннями.

Спільнота та екосистема React не лише допомагають утримувати його на передовій веброзробки, але й гарантують, що фреймворк буде продовжувати розвиватися та адаптуватися до змін у вимогах та технологіях.

2.2 Angular

2.2.1 Архітектура та основні характеристики

Angular — це потужний та гнучкий фреймворк для розробки односторінкових застосунків, створений та підтримуваний Google. Він написаний на TypeScript та пропонує розробникам строгу типізацію та об’єктно-орієнтовані можливості. Angular використовує компонентний підхід, де застосунки будуються із використанням взаємопов’язаних компонентів, кожен з яких має свій шаблон, клас та стилі. Особливістю Angular є його інтегрованість: він містить готові рішення для багатьох завдань, включаючи маршрутизацію, управління формами, клієнтські запити до сервера та багато іншого. Фреймворк використовує двостороннє зв’язування даних, що дозволяє автоматично синхронізувати моделі та вигляд, значно спрощуючи розробку [17]. Модульна структура Angular сприяє організації коду та його перевикористанню. Angular CLI, потужний інструмент командного рядка, спрощує процес розробки, дозволяючи швидко генерувати компоненти, сервіси та інші елементи застосунка, а також забезпечує оптимізацію та розгортання застосунків.

В Angular ієрархічна структура компонентів організовується у деревоподібну модель, де кожен компонент може включати в себе інші компоненти, створюючи вкладену структуру. У корені цієї ієрархії зазвичай знаходиться головний компонент, який відповідає за основний лейаут (layout) застосунку. Від цього кореневого компонента відгалужуються дочірні компоненти, кожен з яких може мати свої власні дочірні компоненти. Ця ієрархія дозволяє локалізувати функціональність та стан в окремих частинах застосунку, спрощуючи управління та перевикористання коду. Компоненти взаємодіють між собою через входи (inputs) для передачі даних вниз по ієрархії та виходи (outputs) для сповіщення про події вгору по ієрархії [18]. Ця структура також сприяє чіткому відділенню відповідальності між

компонентами, роблячи архітектуру застосунку більш зрозумілою та легкою для тестування.

Angular активно використовує TypeScript, мову програмування розроблену Microsoft, яка є надбудовою над JavaScript. TypeScript додає строгу типізацію і об’єктно-орієнтовані можливості, що сприяє більш чіткій структуризації коду і покращує його надійність та масштабованість. У Angular TypeScript використовується для опису компонентів, сервісів, директив, модулів та інших елементів застосунку. Строга типізація дозволяє виявляти помилки на ранніх етапах розробки, що спрощує дебагінг (debugging) і знижує кількість помилок у продакшн-версії застосунку. Також TypeScript покращує автодоповнення коду та навігацію, що робить розробку в Angular більш зручною і ефективною. Використання TypeScript у Angular сприяє розробці більш структурованого та легко підтримуваного коду.

Angular використовує декларативні шаблони для визначення інтерфейсу користувача, дозволяючи розробникам описувати, яким чином відображається UI в залежності від логіки та даних застосунку [19]. Шаблони у Angular написані у HTML з додаванням Angular-специфічних директив, які розширюють можливості HTML. Ці директиви включають умовний рендеринг, цикли, обробку подій та двостороннє зв’язування даних. Двостороннє зв’язування, зокрема, дозволяє автоматично синхронізувати значення в шаблоні з відповідними даними в компоненті. Декларативність шаблонів спрощує розуміння та управління UI, забезпечуючи чітке відокремлення візуальної структури від бізнес-логіки. Це робить шаблони Angular потужним інструментом для створення динамічних та інтерактивних вебінтерфейсів.

У Angular впровадження залежностей є ключовою частиною архітектури, що дозволяє створювати модульний та легко тестований код. Воно базується на використанні сервісів, які можуть бути інжектовані в компоненти та інші сервіси. Залежності визначаються через конструктори компонентів, де Angular автоматично забезпечує необхідні екземпляри

сервісів. Це розділяє відповідальність між різними частинами застосунку, сприяє використанню одного екземпляру сервісу у всьому застосунку і знижує залежність між компонентами. Впровадження залежностей у Angular використовує ієрархічний інжектор, який дозволяє визначати залежності на різних рівнях застосунку, від кореневого модуля до окремих компонентів, надаючи гнучкість у керуванні залежностями та їх областю видимості.

У Angular директиви є особливими класами, які додають поведінку до елементів у Angular-шаблонах. Існують три основні типи директив: компоненти, структурні директиви та атрибутивні директиви. Компоненти визначають шаблони та логіку інтерфейсу, вони є найбільш розповсюдженим типом директив. Структурні директиви змінюють макет, додаючи, видаляючи чи замінюючи елементи DOM, прикладами є *ngIf та *ngFor. Атрибутивні директиви змінюють вигляд чи поведінку елементів DOM, наприклад, NgStyle та NgClass [20]. Директиви в Angular дозволяють розробникам маніпулювати елементами та компонентами на сторінці більш ефективно та гнучко, забезпечуючи потужний інструментарій для створення динамічних інтерфейсів.

У Angular сервіси — це синглтони, які забезпечують можливість виконувати перевикористовувану бізнес-логіку та управління даними по всьому застосунку [21]. Вони відокремлюють логіку від компонентів, що сприяє чистішому коду та легшій підтримці. Сервіси можуть виконувати різні завдання, включаючи роботу з API для отримання та відправлення даних, управління станом застосунку, логіку маршрутизації та інше. Вони інжектуються у компоненти чи інші сервіси через механізм впровадження залежностей Angular, що дозволяє легко доступатися до функціональності сервісу без необхідності створювати новий екземпляр. Управління даними в Angular часто включає використання сервісів для роботи з HTTP-клієнтом для взаємодії з серверними API, обробки помилок, роботи з асинхронними даними через RxJS та інші реактивні підходи. Це забезпечує централізований та ефективний контроль над даними в застосунку.

Маршрутизація в Angular дозволяє управляти навігацією між різними сторінками та компонентами в односторінкових застосунках. Вона використовує Angular Router для визначення маршрутів, що асоціюються з компонентами. Конфігурація маршрутів визначається у модулі маршрутизації, де кожному маршруту присвоюється шлях та відповідний компонент. Angular Router дозволяє виконувати ліниве завантаження модулів для ефективнішої роботи застосунку, а також підтримує захищені маршрути з використанням Guard. Перехід між сторінками здійснюється без перезавантаження всього застосунку, що забезпечує швидку та плавну навігацію. Angular Router також дозволяє управляти історією навігації та параметрами маршрутів, надаючи додаткові можливості для кастомізації поведінки навігації в застосунку.

У Angular існують два основних підходи до роботи з формами: реактивні форми та шаблонні форми. Реактивні форми забезпечують більш гнучкий та масштабований підхід, де управління станом форми та її валідація здійснюються за допомогою класів та реактивних потоків. Це дає розробникам більший контроль та спрощує тестування. Шаблонні форми пропонують більш простий та декларативний підхід, де логіка форми описується безпосередньо у шаблоні HTML з використанням директив Angular. Це робить їх зручними для простих сценаріїв. Обидва типи форм підтримують двостороннє зв’язування даних, валідацію та групування полів, а також дозволяють розробникам легко отримувати та обробляти дані користувача.

HTTP-клієнт у Angular дозволяє виконувати HTTP-запити до серверів, що є необхідним для взаємодії з API та обміну даними. Він забезпечує потужні та гнучкі можливості для відправки запитів, отримання відповідей, обробки помилок та конфігурації параметрів запитів. Використовуючи RxJS, Angular HTTP-клієнт пропонує реактивний підхід для обробки асинхронних запитів, що дозволяє легко впроваджувати складні сценарії, такі як повторні спроби, дебаунсинг або відстеження стану завантаження. Цей клієнт також підтримує інтерцептори для додавання глобальної логіки обробки запитів та відповідей,

як от авторизація або логування, що робить його гнучким інструментом для управління HTTP-запитами в Angular-застосунках.

2.2.2 Спільнота та екосистема

Спільнота та екосистема Angular мають важливе значення для успіху та поширеності цього фреймворку. Вони включають як розробників і користувачів, так і різноманітні інструменти, бібліотеки та ресурси, що підтримують і розвивають Angular-екосистему.

– Angular активно підтримується та розвивається командою розробників у Google, що гарантує регулярні оновлення та довготривалу підтримку;

– по всьому світу регулярно проводяться конференції та зустрічі, присвячені Angular, такі як ng-conf, AngularConnect та локальні ng-meetups, що дозволяє розробникам ділитися знаннями та досвідом;

– існує велика кількість навчальних ресурсів, таких як книги, онлайн-курси, блоги та офіційна документація, які спрощують вивчення Angular;

– спільноти на таких платформах, як Stack Overflow, GitHub, Reddit, та Gitter, дозволяють розробникам отримувати відповіді на свої запитання та обговорювати різні аспекти Angular;

– Angular CLI (Command Line Interface) - інструмент для автоматизації процесів створення, тестування, та розгортання Angular-застосунків;

– розширення для редакторів коду (наприклад, Angular Language);

– Angular Material - бібліотека компонентів UI з високою інтеграцією з Angular;

– RxJS - бібліотека для реактивного програмування з використанням спостерігачів (Observables), що є ключовою частиною Angular для асинхронного програмування;

– NgRx - бібліотека для управління станом за допомогою патернів Redux у Angular-застосунках;

– Angular Universal для серверного рендерингу (SSR) та підвищення швидкості завантаження сторінок;

– інтеграція з мобільними фреймворками, такими як Ionic, для створення мобільних застосунків;

– професійні інструменти для тестування, включаючи Karma, Jasmine, та Protractor, які сприяють розробці надійних та легко тестованих застосунків.

Angular-спільнота та екосистема відрізняються високим рівнем організації та активності, що сприяє швидкому розв’язанню проблем, неперервному навчанню та інноваціям у розробці вебзастосунків.

2.3 Vue.js

2.3.1 Архітектура та основні характеристики

Vue.js є прогресивним JavaScript-фреймворком для створення інтерфейсів користувача, відомим своєю легкістю інтеграції та простотою використання. Його реактивна система даних дозволяє автоматично оновлювати інтерфейс при зміні стану застосунку. Vue використовує декларативний підхід для побудови шаблонів з HTML, CSS та JavaScript. Фреймворк пропонує компонентну архітектуру, що робить код перевикористовуваним та легким для тестування. Vue забезпечує деталізовану документацію та велику спільноту, що робить його доступним для новачків, при цьому має достатньо функціональності для складних застосунків. Він підтримує односторінкові застосунки (SPA) через Vue Router для маршрутизації та Vuex для управління глобальним станом. Його екосистема включає численні плагіни та інтеграції, роблячи Vue гнучким рішенням для широкого спектра проектів веброзробки. Vue.js відзначається високою швидкістю та оптимізацією продуктивності, а також простотою переходу для розробників, які вже знайомі з HTML, CSS та JavaScript.

Реактивність у Vue є фундаментальною характеристикою, яка дозволяє застосункам автоматично оновлюватися у відповідь на зміни стану даних. Це досягається за допомогою системи реактивності, яка відстежує зміни у властивостях даних та автоматично оновлює компоненти. Коли властивості даних змінюються, Vue оновлює DOM, відображаючи нові значення, без необхідності явного втручання розробника. Реактивність у Vue реалізована за допомогою геттерів (getters) і сеттерів (setters) JavaScript, що дозволяє фреймворку автоматично відстежувати залежності між даними та шаблонами. Це сприяє створенню інтуїтивно зрозумілих та декларативних шаблонів, зменшуючи кількість коду, необхідного для управління станом інтерфейсу. Реактивність у Vue робить розробку більш ефективною і зменшує ймовірність помилок, пов’язаних з синхронізацією стану застосунку та його відображенням.

Компонентний підхід у Vue полягає у створенні інтерфейсу користувача через самостійні та перевикористовувані компоненти. Кожен компонент у Vue містить свою HTML-структуру, CSS-стилі та JavaScript-логіку, організовані в одному файлі з розширенням .vue. Це забезпечує модульність та легкість підтримки коду. Компоненти можна вкладати один в одного, створюючи ієрархічну структуру, що відображає структуру інтерфейсу застосунку. Vue використовує систему пропсів (props) для передачі даних від батьківських до дочірніх компонентів, а також події для комунікації у зворотному напрямку. Цей підхід дозволяє створювати високо змінювані та легко перевикористовувані інтерфейси, покращуючи якість та швидкість розробки.

Декларативний рендеринг у Vue дозволяє розробникам описати, як UI повинен виглядати в залежності від стану застосунку, замість того, щоб керувати його змінами вручну. У Vue це досягається за допомогою шаблонів, написаних у HTML, які використовують спеціальний синтаксис для вбудовування динамічних даних. Змінні та вирази JavaScript вставляються безпосередньо в шаблон, прив’язуючись до даних компонента. Коли стан компонента змінюється, Vue автоматично оновлює відповідні частини DOM,

відображаючи нові значення. Цей підхід забезпечує інтуїтивно зрозуміле та ефективне управління інтерфейсом, мінімізуючи помилки та зменшуючи обсяг коду, необхідного для управління динамічними даними.

Vue має вбудовану підтримку для перехідних ефектів, що дозволяє легко додавати анімації та переходи при відображенні чи приховуванні елементів або при зміні компонентів. Використовуючи спеціальні компоненти `<transition>` та `<transition-group>`, розробники можуть огортати елементи чи компоненти, які потребують анімації [22]. Vue автоматично виявляє, коли елемент або компонент вводиться чи видаляється, та застосовує відповідні класи CSS для анімації. Розробники можуть використовувати стандартні CSS-переходи та анімації або інтегрувати JavaScript-хуки для більш складних ефектів. Цей механізм забезпечує гладкі та привабливі переходи в інтерфейсах користувача, підвищуючи візуальну привабливість застосунків на Vue.

Vue надає потужні інструменти для розробників, які полегшують процес розробки та дебагінгу (debugging). Основним інструментом є Vue DevTools, розширення для браузерів Chrome і Firefox, що дозволяє інспектувати та відлагоджувати Vue-застосунки в режимі реального часу. Цей інструмент надає можливість переглядати та редагувати стан компонентів, слідкувати за змінами даних, відстежувати події та аналізувати залежності. Крім того, Vue CLI, потужний інструмент командного рядка, надає шаблони проектів, полегшує налаштування та оптимізує процес розробки та розгортання застосунків [23]. Vue CLI включає функції гарячого перезавантаження, лінтингу, тестування та інші корисні утиліти. Разом ці інструменти роблять процес розробки на Vue ефективним та зручним, надаючи широкі можливості для швидкого прототипування та високоякісної розробки [24].

Легкість інтеграції є однією з ключових переваг Vue.js, завдяки чому фреймворк здобув популярність серед розробників. Vue може бути легко інтегрований у існуючі проекти, оскільки не вимагає великих змін у структурі застосунку [25]. Це дозволяє додавати Vue поступово, наприклад, використовуючи його лише для певних частин інтерфейсу або компонентів.

Мала вага та гнучкість фреймворку роблять його ідеальним вибором для розробки простих вебсайтів, а також для впровадження в складні, багатосторінкові застосунки. Vue також пропонує простий синтаксис та шаблонізацію, що спрощує розробку та інтеграцію для розробників, навіть з обмеженим досвідом у JavaScript або фронтенді. Крім того, Vue є сумісним з різноманітними бібліотеками та інструментами, що дозволяє легко інтегрувати його з іншими технологіями та використовувати в різних контекстах веброзробки.

2.3.2. Спільнота та екосистема

Vue.js пишається активною та дружньою спільнотою. Хоча він не має підтримки великої корпорації, як Angular чи React, його екосистема швидко зростає завдяки ентузіазму й підтримці розробників.

Особливості спільноти та екосистеми Vue.js:

– Vue має масивну підтримку у формі форумів, чат-груп, конференцій та онлайн-ресурсів для навчання;

– розробники Vue можуть користуватися численними інструментами та плагінами, що робить розробку гнучкою та ефективною;

– до екосистеми Vue належать офіційні бібліотеки, такі як Vue Router для маршрутизації та Vuex для управління станом, а також велика кількість бібліотек і плагінів (pluggins) від спільноти;

– Vue.js відомий своєю низькою пороговою входження, що робить його привабливим для новачків у сфері веброзробки.

2.4 Ember.js

2.4.1 Архітектура та основні характеристики

Ember.js є відкритим фреймворком для розробки вебзастосунків, який славиться своєю конвенцією над конфігурацією. Ember використовує компонентний підхід для побудови інтерфейсу користувача, пропонуючи вбудовані шаблони, які спрощують процес розробки. Фреймворк має строгу організацію та вимоги до структури проекту, що робить застосунки на Ember легкими для масштабування та підтримки. Завдяки використанню двостороннього зв’язування даних, Ember забезпечує ефективну синхронізацію між моделлю та виглядом. Фреймворк включає в себе роутер для управління навігацією в застосунку, а також власний механізм для управління станом. Ember CLI, інструмент командного рядка, покращує процес розробки, надаючи можливості для швидкого створення компонентів, сервісів та інших елементів застосунку, а також спрощує процес збірки та розгортання. Ember має активну спільноту розробників і регулярно оновлюється, забезпечуючи підтримку сучасних стандартів веброзробки. Хоча Ember може мати крутіший кривий навчання порівняно з іншими фреймворками, його продумана архітектура та комплексність роблять його відмінним вибором для розробки великих та складних вебзастосунків.

Конвенція над конфігурацією у Ember є ключовим принципом, що означає, що фреймворк має встановлені стандарти та практики, якими слід керуватися під час розробки, замість детальної настройки кожного аспекту застосунку. Цей підхід спрощує процес розробки, оскільки розробники можуть сконцентруватися на побудові застосунку, а не на налаштуванні інфраструктури. Ember автоматично визначає, як повинні взаємодіяти компоненти, моделі, роутери та інші елементи застосунку на основі їх назв та розташування у структурі проекту. Це робить застосунки на Ember більш передбачуваними та зменшує кількість потенційних помилок, що особливо корисно при роботі великих команд або над великими проектами.

Ember CLI є командним інтерфейсом, який служить як основний інструмент для створення, розробки та розгортання застосунків Ember. Він автоматизує багато аспектів розробки, надаючи шаблони для швидкого створення компонентів, маршрутів, сервісів та інших елементів застосунку. Ember CLI підтримує гаряче перезавантаження, що дозволяє розробникам бачити зміни в браузері миттєво без перезавантаження сторінки. Також інструмент включає потужні засоби для тестування, включаючи єдиний інтерфейс для модульних, інтеграційних та приймальних тестів. Ember CLI оптимізує процес збірки застосунку, забезпечуючи мініфікацію, конкатенацію файлів та інші покращення продуктивності. Інструмент також спрощує процес розгортання застосунків за допомогою вбудованих рішень або плагінів (pluggins). Ember CLI значно підвищує продуктивність розробників та є ключовою складовою екосистеми Ember.

Glimmer Engine в Ember — це легковаговий рендеринговий двигун, який забезпечує високу продуктивність рендерингу компонентів. Використовуючи сучасні технології та оптимізації, Glimmer покращує швидкість рендерингу та зменшує навантаження на браузер, особливо у великих та складних інтерфейсах. Glimmer використовує компоненти як основні будівельні блоки інтерфейсу, пропонуючи ефективне деревоподібне оновлення DOM та розумне керування ресурсами. Завдяки цьому двигуну Ember застосунки можуть досягати кращої продуктивності та швидкості відгуку, що позитивно впливає на загальний досвід користувачів. Glimmer також підтримує розширені функції, такі як Server-Side Rendering (SSR) та Progressive Web Apps (PWA), роблячи Ember могутнім інструментом для розробки сучасних вебзастосунків.

Routing у Ember забезпечує управління навігацією і рендерингом сторінок у односторінкових застосунках. Він дозволяє визначити маршрути, кожен з яких асоціюється з конкретним компонентом або шаблоном. Ember Router використовує шляхи URL для визначення стану застосунку, роблячи URL-адреси основою для навігації та стану. Це забезпечує зручну навігацію

користувачів, можливість використання закладок та підтримку історії перегляду. Ember реалізує логіку маршрутизації через ієрархічну структуру, що дозволяє розробникам структурувати застосунок у вигляді вкладених маршрутів і компонентів. Router також підтримує динамічні параметри маршрутів, запити на сервер, ліниве завантаження коду та захистені маршрути з використанням системи Guard. Це робить маршрутизацію в Ember гнучкою і потужною частиною фреймворку для розробки комплексних вебзастосунків.

Data Layer у Ember забезпечується за допомогою Ember Data, потужного рішення для управління даними у застосунках. Ember Data допомагає в роботі з моделями даних, надаючи уніфікований спосіб завантаження, зберігання та маніпулювання даними з сервера. Воно використовує концепцію моделей для представлення даних, адаптерів для взаємодії з серверним API та серіалізаторів для перетворення даних при передачі між клієнтом та сервером. Ember Data підтримує автоматичне кешування запитів, що зменшує навантаження на мережу та забезпечує більш швидку відповідь застосунку. Воно також дозволяє легко управляти залежностями між моделями через відносини, такі як «belongsTo» та «hasMany». Завдяки цьому, Data Layer в Ember спрощує роботу з складними структурами даних та забезпечує ефективну взаємодію між фронтендом та бекендом.

2.4.2. Спільнота та екосистема

Ember.js має стабільну та досвідчену спільноту розробників. Завдяки своєму довгому історичному розвитку, Ember заслужив репутацію надійного фреймворку з багатим набором функцій.

Особливості спільноти та екосистеми Ember.js:

– Ember.js знаменитий своїм політиками довгострокової підтримки, що робить його вибором для застосунків, що потребують довготривалого обслуговування;

– спільнота активно підтримує фреймворк, регулярно випускаючи оновлення та вдосконалення;

– велика кількість ресурсів, доступно багато туторіалів, документації та книг, які покривають Ember.js;

– екосистема Ember.js містить велику кількість застосунків та плагінів (pluggins), які розширюють функціональність основного фреймворку;

– конференції, зустрічі та хакатони часто проводяться по всьому світу, що дозволяє розробникам спілкуватися, обмінюватися ідеями та навичками.

3 МЕТОДОЛОГІЯ ПОРІВНЯЛЬНОГО АНАЛІЗУ

3.1 Вибір параметрів для аналізу

Вибір параметрів для аналізу є критичним етапом в порівняльному дослідженні JavaScript-фреймворків, адже саме вони формують основу для об’єктивного та всебічного оцінювання. Для ефективного порівняння React, Angular, Vue.js та Ember.js було обрано наступні параметри:

– швидкість роботи та продуктивність включає аналіз часу завантаження сторінки, часу відгуку на дії користувача, а також ефективність використання ресурсів (CPU та пам’ять) [26];

– легкість освоєння та розробки, оцінка часу та зусиль, необхідних для новачків, щоб почати працювати з фреймворком, а також зручність розробки і підтримки застосунків [27];

– гнучкість та масштабованість, оцінка спроможності фреймворків адаптуватися до змінних вимог проєкту та їх здатність підтримувати великі та складні застосунки;

– безпека, аналіз вбудованих механізмів безпеки та здатність фреймворків захищати застосунки від поширених вразливостей [28];

– тестування та дебагінг (debugging), дослідження інструментів та можливостей, які фреймворки надають для тестування та виправлення помилок у застосунках;

– спільнота та підтримка, оцінка активності та розміру розробницької спільноти, наявності навчальних матеріалів, та доступності підтримки;

– сумісність та інтеграція, аналіз здатності фреймворків працювати разом з іншими бібліотеками та інструментами, а також їх інтеграція з існуючими системами [29];

– екосистема та доступні ресурси, дослідження наявності готових рішень, плагінів (pluggins), компонентів та інших ресурсів, що можуть полегшити та прискорити розробку.

Ці параметри допоможуть сформувати об’єктивне бачення кожного фреймворку, оцінюючи їх не тільки за технічними характеристиками [29], а й за зручністю у використанні, підтримкою спільноти та ефективністю застосування в реальних проєктах [30].

3.2 Методи збору та аналізу даних

Розділ, присвячений методам збору та аналізу даних, зосереджується на підходах та інструментарії, які були застосовані для забезпечення достовірності та відтворюваності результатів порівняльного аналізу JavaScript-фреймворків [31]. Здійснення глибокого та всебічного огляду кожного з фреймворків вимагало використання комбінованого підходу, який включав:

– документальний аналіз, огляд офіційної документації, руководств для розробників, та технічних специфікацій фреймворків для вивчення їх можливостей та особливостей [32];

– бенчмаркінг (benchmarking), проведення стандартизованих тестів продуктивності за допомогою інструментів, таких як Lighthouse та WebPageTest для вимірювання швидкості завантаження, часу відклику на взаємодії користувача та використання системних ресурсів;

– аналіз ринкової частки, використання даних з різних аналітичних джерел, таких як Google Trends, Stack Overflow Insights та GitHub, для визначення популярності та активності спільноти навколо кожного фреймворку;

– розробка декількох тестових проєктів [33], щоб визначити гнучкість, масштабованість, та інтеграцію фреймворків у різних сценаріях використання;

Вибір методів збору та аналізу даних був спрямований на мінімізацію упередженості [34] та підвищення об’єктивності оцінювання, а також забезпечення повноти та глибини дослідження.

3.3. Процес оцінки фреймворків

3.3.1 Розробка програми

Для порівняльного аналізу фреймворків React, Angular, Vue.js та Ember.js було розроблено програму «To Do List» на кожному з них. Цей проект обрано через його універсальність та відображення типових завдань веброзробки, включаючи управління станом, взаємодію з користувачем та рендеринг інтерфейсу.

Наступним кроком була розробка дизайну інтерфейсу (рис. 3.1). Було створено простий, але функціональний UI, який був однаковий для всіх фреймворків, забезпечуючи справедливе порівняння візуального представлення.

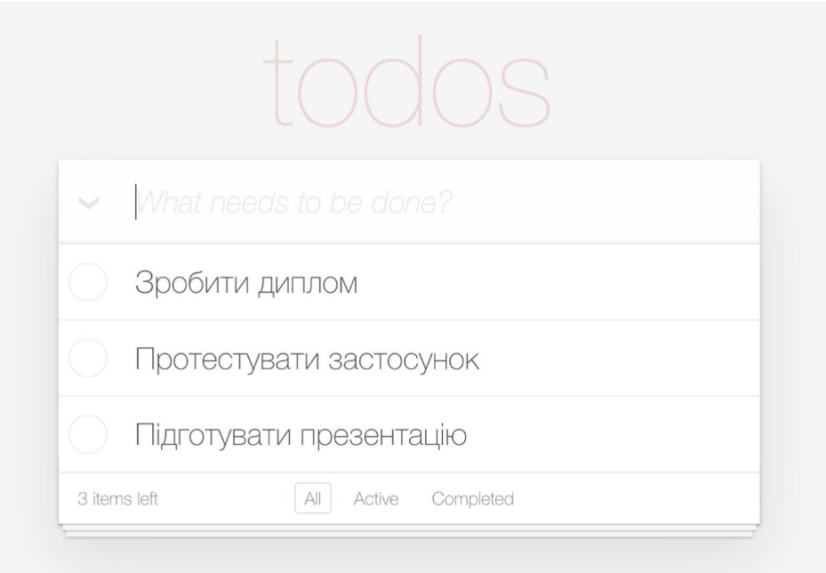

Рисунок 3.1 – Дизайн інтерфейсу

Процес розробки почався з визначення функціональних вимог до програми, що включали можливість додавати завдання, видаляти їх (рис. 3.2) та відмічати виконані (рис. 3.3). Це забезпечило однакові умови для кожного фреймворку.

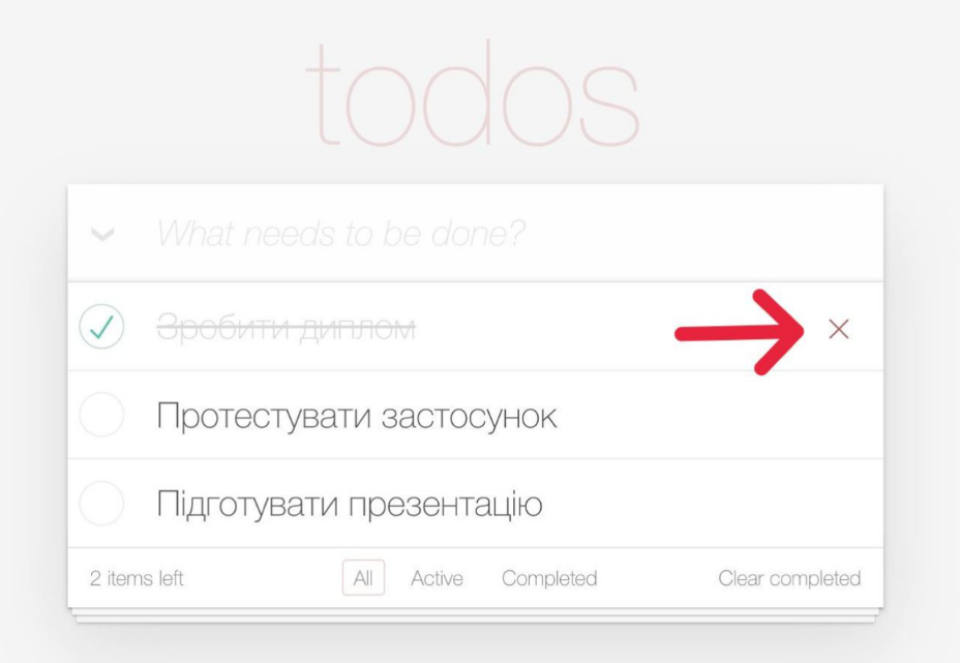

Рисунок 3.2 – Видалення завдань

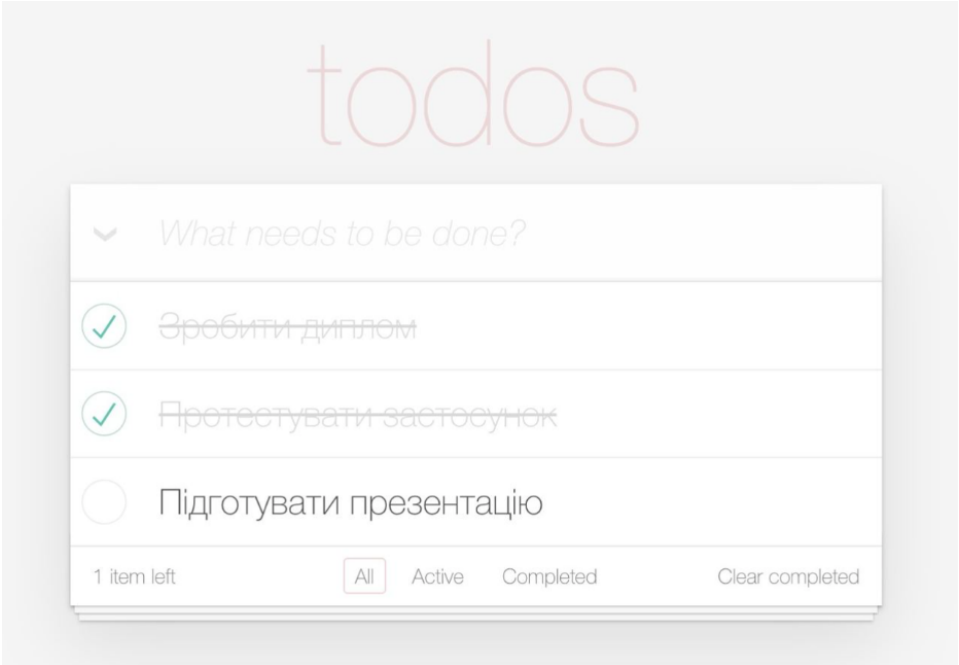

Рисунок 3.3 – Відмічення виконаних завдань

Для кожного фреймворку було послідовно розроблено версію «To Do List», дотримуючись рекомендованих практик та архітектурних патернів кожного інструменту [27]. Після реалізації основного функціоналу, було проведено оптимізацію коду для кращої продуктивності.

3.3.2 Проведення тестів продуктивності

Для проведення тестів продуктивності візуально однакових сайтів на різних фреймворках було використано застосунок Lighthouse, який є частиною Chrome DevTools у браузері Google Chrome. Відкриваємо кожен сайт у Chrome і запускаємо Chrome DevTools. У вкладці Lighthouse вибираємо потрібні параметри для тестування, наприклад, емуляцію мобільного пристрою та тип мережі. Після цього запускаємо тест, натискаючи на кнопку «Generate report». Після завершення тесту Lighthouse створює звіт з оцінками та рекомендаціями щодо поліпшення продуктивності сайту. Цей процес повторюємо для кожного сайту, розробленого на різних фреймворках. Порівнюємо звіти Lighthouse для кожного фреймворку, аналізуючи такі показники, як швидкість завантаження сторінки, час відгуку на дії користувача та ефективність використання ресурсів. На основі отриманих даних робимо висновки про продуктивність кожного фреймворку, що дає змогу оцінити їхні сильні та слабкі сторони у контексті реальних умов використання.

У таблиці 3.1 показані результати порівняння сайтів на мобільних пристроях.

Таблиця 3.1 – Порівняння сайтів на мобільних пристроях за допомогою Lighthouse

Фреймворк	**Продуктивність (Performance)**	**Доступність (Accessibility)**	**Найкращі практики (Best Practices)**	**SEO**
Angular	75%	88%	95%	64%
Vue	95%	78%	100%	64%
Ember	96%	78%	100%	82%
React	82%	87%	100%	64%

У таблиці 3.2 показані результати порівняння сайтів на десктопних пристроях.

Таблиця 3.2 – Порівняння сайтів на десктопних пристроях у Lighthouse

Фреймворк	**Продуктивність (Performance)**	**Доступність (Accessibility)**	**Найкращі практики (Best Practices)**	**SEO**
Angular	99%	88%	95%	78%
Vue	100%	78%	100%	78%
Ember	100%	78%	100%	89%
React	99%	87%	100%	78%

На основі даних з таблиць можна зробити такі висновки:

Angular виявляє високу продуктивність на десктопних пристроях, але випадає в мобільному продуктивністному рейтингу та показує середні результати в SEO.

Vue має вражаючі показники продуктивності в усіх сферах, але показує найнижчу оцінку доступності серед усіх фреймворків.

Ember вражає на мобільних пристроях з продуктивністю та SEO, але має відносно низьку доступність.

React демонструє стабільно високу продуктивність на десктопних пристроях і добрі показники за найкращими практиками, однак SEO залишається середнім без змін на різних пристроях.

Наступним етапом було використання застосунку WebPageTest. Завершення тесту супроводжується генерацією звіту з ключовими метриками швидкості та візуальними показниками завантаження сторінки. Графік завантаження ресурсів надає детальний розгляд кожного запиту і часу його обробки. Звіт також включає порівняльну аналітику і рекомендації для оптимізації. На основі цих даних вносяться вдосконалення в сайт, після чого можливе повторне тестування для оцінки змін у продуктивності. Результати тесту показані у таблиці 3.3

Таблиця 3.3 – Порівняння сайтів за допомогою WebPageTest

Фреймворк	**Time to First Byte (TTFB)**	**Start Render**	**First Contentful Paint (FCP)**	**Speed Index**	**Largest Contentful Paint (LCP)**	**Cumulative Layout Shift (CLS)**	**Total Blocking Time (TBT)**	**Page Weight**
Vue	0,743 s	1,100 s	1,070 s	1,149 s	1,451 s	0,033	0,302 s	194k

Продовження таблиці 3.3

Фреймворк	**Time to First Byte (TTFB)**	**Start Render**	**First Contentful Paint (FCP)**	**Speed Index**	**Largest Contentful Paint (LCP)**	**Cumulative Layout Shift (CLS)**	**Total Blocking Time (TBT)**	**Page Weight**
Ember	0,731 s	2,300 s	2,295 s	2,300 s	2,295 s	0	0,000 s	172k
Angular	0,757 s	2,000 s	1,953 s	2,035 s	1,953 s	0	0,395 s	405k
React	0,731 s	1,100 s	1,066 s	1,246 s	1,896 s	0,014	0,770 s	382k

За даними тестів, Vue виявляється лідером з швидкості відповіді сервера (Time to First Byte) та швидкості рендерингу (Start Render), що свідчить про ефективність ініціалізації застосунку. Ember показує найповільніший час початку рендерингу та First Contentful Paint, що може свідчити про відносно повільніше завантаження контенту. Angular має високий Total Blocking Time, що вказує на потенційні затримки в обробці подій через зайнятість основного потоку. React демонструє найкращий показник для Largest Contentful Paint, що свідчить про швидкість завантаження найбільшого елементу контенту, але має відносно високий Total Blocking Time. Найменша вага сторінки у Vue та Ember може сприяти швидшому завантаженню сторінок, в той час як вища вага сторінки у Angular і React може бути пов’язана з більш великими бібліотеками або некомпактним кодом.

3.3.3 Порівняльний аналіз популярності фреймворків

Спочатку було проведено порівняння у Google Trends. Для проведення аналізу на Google Trends було введено назву першого фреймворку для порівняння у поле пошуку. Використав опцію «Додати запит» для включення додаткових фреймворків у порівняння. Був встановлений часовий діапазон на 5 років для аналізу трендів. Результати показано на графіках на рисунку 3.4. Отримані дані було використано для оцінки загальної популярності та інтересу до кожного фреймворку.

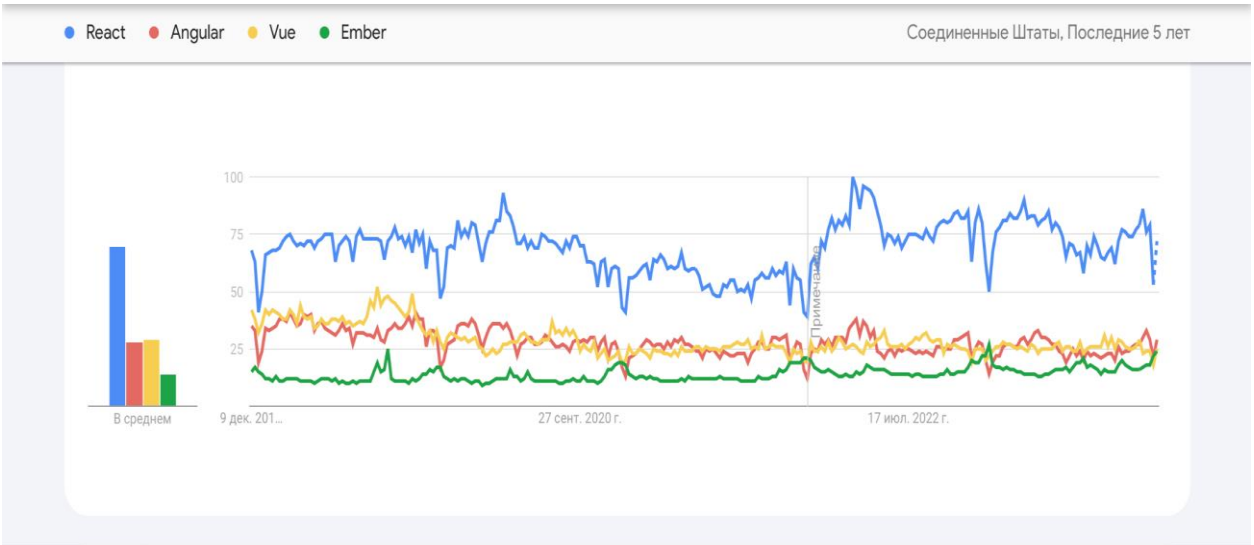

Рисунок 3.4 – Порівняння у Google Trends

За даними Google Trends, React має найвищий рівень інтересу, що стабільно зберігається протягом останніх 5 років, з кількома піками, що можуть вказувати на особливі події або релізи. Angular утримує другу позицію з суттєвим відставанням від React, вказуючи на меншу, але все ж високу популярність. Vue і Ember показують більш скромні показники і обидва демонструють більш стабільний, але нижчий рівень інтересу порівняно з React та Angular. Ember має найменший рівень інтересу серед цих фреймворків.

Наступним етапом буде порівняння через Stack Overflow Insights. Stack Overflow, будучи провідною платформою для професіоналів у сфері

програмування, забезпечує багатий набір даних про тенденції у запитах, відповідях, мітках та інших взаємодіях, що відображають реальний інтерес та виклики, з якими стикаються розробники щоденно.

Аналізуючи ці дані, ми можемо отримати глибоке розуміння популярності та прийняття фреймворків в порівнянні з іншими доступними інструментами. Також це надасть нам інформацію про рівень залученості спільноти та їхню готовність ділитися знаннями та досвідом, що є критично важливим для новачків та досвідчених розробників, які вибирають фреймворк для своїх майбутніх проектів. Завдяки цьому буде отримано цінну інформацію, яка допоможе у формуванні виваженого погляду на кожний фреймворк, його місце в екосистемі сучасної веброзробки та його потенціал у майбутньому.

Дані за 2023 рік за популярністю фреймворків серед професійних розробників було виведено у таблицю 3.4 [35].

Таблиця 3.4 – Порівняння фреймворків на Stack Overflow Insights

Фреймворк	**Відсоток використання**
React	42.87%
Angular	19.89%
Vue.js	17.64%
Ember	6.01%

За даними таблиці, React є найпопулярнішим фреймворком серед розглянутих на Stack Overflow Insights, займаючи майже половину ринкової частки. Angular і Vue.js також мають значну популярність, але відстають від React, з меншою кількістю використання серед розробників. Ember, хоча й

включений до порівняння, має значно менший відсоток використання, що вказує на меншу популярність у порівнянні з іншими фреймворками.

3.3.4 Рекомендації щодо вибору фреймворку в залежності від конкретних потреб проєкту

При виборі фреймворку для проєкту важливо врахувати його масштаб, вимоги до продуктивності, складність інтерфейсу та наявність розробників. React підходить для динамічних вебзастосунків із великою кількістю інтерактивних елементів, він пропонує велику гнучкість та має велику спільноту. Angular є оптимальним вибором для enterprise-level застосунків, що вимагають строгої структури та готових рішень «з коробки». Vue.js пропонує спрощений та прогресивний підхід, що робить його ідеальним для швидкої розробки та високої продуктивності у менших або середніх проектах. Ember може бути корисним для тих, хто шукає конвенційно орієнтований фреймворк із суворим набором конвенцій та вбудованими рішеннями для швидкого старту проекту. Розглядаючи ці особливості та потреби проекту, можна зробити обґрунтований вибір фреймворку.

3.3.5 Пропозиції щодо подальших досліджень

Для подальших досліджень у сфері веброзробки можна розглянути адаптацію сучасних вебAPI та прогресивних вебзастосунків, вивчення впливу технологій штучного інтелекту на автоматизацію розробки, аналіз ефективності serverless архітектур та контейнеризації, дослідження стратегій для покращення продуктивності вебзастосунків, включаючи оптимізацію асетів (assets) і використання CDN, а також оцінка безпеки вебвебзастосунків і розробка нових методів захисту від загроз. Також варто звернути увагу на

розвиток та інтеграцію кросплатформних рішень, вдосконалення доступності вебконтенту згідно з WCAG та розробку інструментів для забезпечення якості та сумісності коду.

ВИСНОВКИ

У рамках цієї кваліфікаційної роботи було проведено дослідження та порівняльний аналіз чотирьох популярних JavaScript-фреймворків для розроблення вебзастосунків: React, Angular, Vue.js та Ember.js.

Кожен з розглянутих фреймворків має свою унікальну специфіку, що робить його підходящим для певних типів проєктів. Наприклад, React пропонує гнучкість і широке застосування завдяки своїй компонентній архітектурі, в той час як Angular надає повноцінне рішення з багатим набором функцій для комплексних застосунків;

В продуктивності фреймворків багато що залежить від конкретної реалізації і оптимізації коду. Сучасні фреймворки пропонують механізми для підвищення продуктивності, такі як віртуальний DOM в React та компіляція Ahead-of-Time в Angular;

Наявність великої та активної спільноти є великим плюсом для фреймворків, адже це забезпечує кращу документацію, більшу кількість плагінів (pluggins) та інструментів, а також сприяє швидкому вирішенню проблем;

Деякі фреймворки, як-от Angular, надають суворіші конвенції та патерни, що може спростити масштабування застосунків і підтримку коду на великих проєктах;

Швидкість навчання та впровадження також важлива для розробників. Наприклад, Vue.js часто вважають більш простим для вивчення через його лаконічний синтаксис і підхід, орієнтований на HTML;

Важливо звернути увагу на траєкторію розвитку фреймворку і відкритість до нових інновацій. Svelte, наприклад, впроваджує новаторський підхід до побудови застосунків, що може змінити парадигму розробки в майбутньому.

Враховуючи всі ці аспекти, розробники повинні оцінювати JavaScript-фреймворки на основі конкретних потреб проєкту, ресурсів та переваг команди. Важливо також пам’ятати, що технологічний ландшафт постійно змінюється, і нові тенденції можуть з’являтися швидко, тому необхідно бути відкритими до навчання і адаптації нових інструментів і практик.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Osmani, A. (2023). *Learning JavaScript design patterns*. " O'Reilly Media, Inc.".

2. Su, Y., Chen, G., Li, M., Shi, T., & Fang, D. (2021). Design and implementation of web multimedia teaching evaluation system based on artificial intelligence and jQuery. *Mobile Information Systems*, *2021*, 1-11.

3. AngularJS - Quick Guide. What is AngularJS? URL: https://www.tutorialspoint.com/angularjs/angularjs_quick_guide.htm (дата звернення 29.10.2023).

4. Ferreira, F., Borges, H. S., & Valente, M. T. (2022). On the (un-) adoption of JavaScript front-end frameworks. *Software: Practice and Experience*, *52*(4), 947-966.

5. Chęć, D., & Nowak, Z. (2019). The performance analysis of web applications based on virtual DOM and reactive user interfaces. In *Engineering Software Systems: Research and Praxis* (pp. 119-134). Springer International Publishing.

6. Kumpulainen, T. (2021). Web application development with Vue. js.

7. bin Uzayr, S. (2023). Conquering JavaScript: The Practical Handbook.

8. Thakkar, M. (2020). *Building React Apps with Server-Side Rendering: Use React, Redux, and Next to Build Full Server-Side Rendering Applications*. Apress.

9. Cherny, B. (2019). *Programming TypeScript: making your JavaScript applications scale*. O'Reilly Media.

10. Iskandar, T. F., Lubis, M., Kusumasari, T. F., & Lubis, A. R. (2020, May). Comparison between client-side and server-side rendering in the web development. In *IOP Conference Series: Materials Science and Engineering* (Vol. 801, No. 1, p. 012136). IOP Publishing.

11. Banks, A., & Porcello, E. (2017). *Learning React.* O'Reilly Media.

12. Bertoli, M. (2016). *React Design Patterns and Best Practices.* Packt

Publishing.

13. Ranjan, A., Sinha, A., & Battewad, R. (2020). *JavaScript for modern web development: building a web application using HTML, CSS, and JavaScript*. BPB Publications.

14. Madsen, M., Lhotak, O., & Tip, F. (2020, January). A Semantics for the Essence of React. In *European Conference on Object-Oriented Programming*.

15. Struhár, V., Craciunas, S. S., Ashjaei, M., Behnam, M., & Papadopoulos, A. V. (2021, September). React: Enabling real-time container orchestration. In *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)* (pp. 1-8). IEEE.

16. Boduch, A., & Derks, R. (2020). *React and React Native: A complete hands-on guide to modern web and mobile development with React. js*. Packt Publishing Ltd.

17. Murray, N., Lerner, A., & Coury, F. (2018). *ng-book: The Complete Guide to Angular.* Fullstack.io.

18. Wilken, J. (2016). *Angular in Action.* Manning Publications.

19. O'Hanlon, P. (2019). *Advanced TypeScript Programming Projects: Build 9 Different Apps with TypeScript 3 and JavaScript Frameworks Such as Angular, React, and Vue*. Packt Publishing Ltd.

20. Bijoura, A., & Murugan, K. (2021). Comparative Analysis of Web Frameworks (Angular JS, React JS, Vue JS). *Solid State Technology*, *64*(2), 5130-5140.

21. Singh, V. P., Sondhi, B., & Kanji, R. (2021). E-commerce Web Application Using Angular.

22. Macrae, C. (2018). *Vue.js: Up and Running.* O'Reilly Media.

23. Gore, A. (2018). *Full-Stack Vue.js 2 and Laravel 5.* Packt Publishing.

24. Ogelsteller, F., & Strack, I. (2014). *Meteor: Full-Stack Web Application Development.* Packt Publishing.

25. Accomazzo, A., & Murray, N. (2017). *Fullstack React.* Self-published.

26. Оцінка юзабіліті освітніх сайтів: методи і технології / С. В. Тітов, О. В. Тітова // Вісник ХДАК / Зб. наук. праць. – Х: ХДАК., 2015. – Вип. 47. – С. 127-134.

27. Web-сайти органів місцевого самоврядування як складова впровадження е-урядування в Україні [Електронний ресурс] / С. В. Тітов, О. В. Тітова // Вісник Харківської державної академії культури . - 2013. - Вип. 39. - С. 146-155. - Режим доступу: http://nbuv.gov.ua/j-pdf/hak_2013_39_20.pdf

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

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

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

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

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

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

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

35. Web frameworks and technologies. URL: https://survey.stackoverflow.co/2023/#most-popular-technologies-webframe-prof (дата звернення 14.10.2023)