Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інфокомунікацій
(повна назва)

Кафедра Інформаційно-мережної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження методів проєктування інтерфейсу вебзастосунків
(тема)

Виконав:
студент 2 курсу, групи ІМІм-22-2
Скворцов. В.Х.
(прізвище, ініціали)

Спеціальність 172 Телекомунікації та радіотехніка
(код і повна назва спеціальності)
Тип програми освітньо-наукова
Освітня програма Інформаційно-мережна інженерія
(повна назва освітньої програми)

Керівник к.т.н доц. Золотарьов В.А.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Безрук В.М.
(підпис) (прізвище, ініціали)

2024 р.

Не містить відомостей заборонених до відкритого публікування

Студент _____ / Скворцов. В.Х./

Керівник _____ / Золотарьов В.А./

Харківський національний університет радіоелектроніки

Факультет Інфокомунікацій

Кафедра Інформаційно-мережної інженерії

Рівень вищої освіти другий (магістерський)

Спеціальність 172 Телекомунікації та радіотехніка

Тип програми освітньо-наукова
(код і повна назва)

Освітня програма Інформаційно-мережна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_19_»_березня_ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Скворцову Владиславу Хусановичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження методів проєктування інтерфейсу вебзастосунків

затверджена наказом університету від 18 березня 2024 р. № 232 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 10 червня 2024 р.

3. Вихідні дані до роботи: Провести порівняльний аналіз провідних фреймворків за різними критеріями, розглянути різні архітектурні підходи до розроблення програмного забезпечення, розглянути етапи життєвого циклу програмного забезпечення та розробити вебзастосунок.

4. Перелік питань, що потрібно опрацювати в роботі: *Вступ*
1 ОГЛЯД СУЧАСНИХ ТЕХНОЛОГІЙ ВЕБРОЗРОБЛЕННЯ
2 ПРОЦЕС ТА АРХІТЕКТУРНІ ПІДХОДИ ПРОЄКТУВАННЯ ВЕБІНТЕРФЕЙСІВ
3 ВИБІР ІНСТРУМЕНТІВ ДЛЯ РОЗРОБЛЕННЯ ВЕБІНТЕРФЕЙСУ КОРИСТУВАЧА
4 РОЗРОБЛЕННЯ ІНТЕРФЕЙСУ ВЕБЗАСТОСУНКА
Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання з рішенням випускової кафедри)
Слайди у форматі Power Point – Титульний аркуш. Мета роботи. Вступ. ОГЛЯД СУЧАСНИХ ТЕХНОЛОГІЙ ВЕБРОЗРОБЛЕННЯ. ПРОЦЕС ТА АРХІТЕКТУРНІ ПІДХОДИ ПРОЄКТУВАННЯ ВЕБІНТЕРФЕЙСІВ. ВИБІР ІНСТРУМЕНТІВ ДЛЯ РОЗРОБЛЕННЯ ВЕБІНТЕРФЕЙСУ КОРИСТУВАЧА. РОЗРОБЛЕННЯ ІНТЕРФЕЙСУ ВЕБЗАСТОСУНКА. Висновки

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Ознайомлення із завданням. Уточнення ТЗ.	08.04.2024	вик.
2	Підбір літератури за темою роботи	14.04.2024	вик.
3	Виконання розділу 1	20.04.2024	вик.
4	Виконання розділу 2	02.05.2024	вик.
5	Виконання розділу 3	12.05.2024	вик.
6	Виконання розділу 4	21.05.2024	вик.
7	Оформлення презентаційного матеріалу,	09.06.2024	вик.
8	Подання атестаційної роботи на перевірку	12.06.2024	вик.

Дата видачі завдання 19 березня 2024 р.

Студент _____
(підпис)

Керівник роботи _____ к.т.н доц. Золотарьов В.А.
(підпис)

РЕФЕРАТ

Пояснювальна записка: 105 с., 58 рис., 5 табл., 25 джерел, 3 додатків.

Об'єктом дослідження є дослідження методів проєктування інтерфейсу вебзастосунків.

Мета роботи – проведення аналізу та порівняння сучасних методів і інструментів розроблення вебінтерфейсів.

У процесі дослідження проведено порівняльний аналіз провідних фреймворків за різними критеріями, розглянуто різні архітектурні підходи до розроблення програмного забезпечення та етапи життєвого циклу програмного забезпечення. Розроблено вебзастосунок.

Проведене дослідження показало, що фреймворк «React» є лідером за більшістю критеріїв і є добрим вибором для більшості проєктів, у тому числі для дослідження.

ВЕБІНТЕРФЕЙСИ; ПОРІВНЯННЯ ФРЕЙМВОРКІВ; АРХІТЕКТУРА ВЕБЗАСТОСУНКІВ; РОЗРОБКА ІНТЕРФЕЙСУ ВЕБЗАСТОСУНКА.

THE ABSTRACT

Explanatory note: 105 p., 58 figs., 5 tables, 25 sources, 3 appendices.

The object of the study is the research of methods for designing web application interfaces.

The purpose of the work is to analyze and compare modern methods and tools for developing web interfaces.

During the research process, a comparative analysis of leading frameworks based on various criteria was conducted, different architectural approaches to software development and stages of the software life cycle were considered. A web application was developed.

The conducted research showed that the "React" framework is the leader by most criteria and is a good choice for most projects, including this study.

WEB INTERFACES; FRAMEWORK COMPARISON; WEB APPLICATION ARCHITECTURE; WEB APPLICATION INTERFACE DEVELOPMENT.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

AJAX (Asynchronous JavaScript and XML) – асинхронний JavaScript і XML.

API (Application Programming Interface) – інтерфейс програмного забезпечення.

BDD (Behavior Driven Development) – розроблення з урахуванням поведінки.

CLI (Command Line Interface) – інтерфейс командного рядка.

CLS (Cumulative Layout Shift) – кумулятивний зсув макета.

CSRF (Cross-Site Request Forgery) – міжсайтова підробка запитів.

CSS (Cascading Style Sheets) – каскадні таблиці стилів.

DevOps (Development Operations) – операції та розроблення.

DOM (Document Object Model) – модель об'єктів документа.

E2E (End-to-end) – з кінця до кінця.

ECMAScript (European Computer Manufacturers Association Script) – Сценарій Європейської асоціації комп'ютерних виробників.

FCP (First Contentful Paint) – перше відображення контенту.

HTML (Hypertext Markup Language) – мова розмітки гіпертексту.

HTTP (Hypertext Transfer Protocol) – протокол передачі гіпертексту.

JSON (JavaScript Object Notation) – нотація об’єктів JavaScript.

JSX – JavaScript XML.

LCP (Largest Contentful Paint) – найбільше відображення контенту.

MVC (Model-View-Controller) – модель-представлення-контролер.

MVP (Minimum Viable Product) – мінімально цінний продукт.

PWA (Progressive Web Application) – прогресивний вебзастосунок.

SEO (Search Engine Optimization) – пошукова оптимізація.

SI (Speed Index) – індекс швидкості.

TBT (Total Blocking Time) – загальний час блокування.

TDD (Test Driven Development) – розроблення з урахуванням тестування.

UI (User Interface) – інтерфейс користувача.

URL (Uniform Resource Locator) – уніфікований покажчик ресурсу.

XML (Extensible Markup Language) – розширювана мова розмітки.

XSS (Cross-Site Scripting) – міжсайтовий скриптинг.

ВСТУП

Сучасна епоха цифрових технологій характеризується безперервним зростанням вимог до якості та функціональності вебзастосунків. Вебінтерфейси, що забезпечують інтерактивну взаємодію користувачів з системами, відіграють вирішальну роль у забезпеченні зручності користування та ефективності роботи програмних продуктів.

Актуальність даного дослідження обумовлена необхідністю забезпечення високих стандартів якості у веброзробленні, що включає продуктивність, масштабованість та зручність використання вебзастосунків. У сучасних умовах на ринку існує велика кількість JavaScript-фреймворків та бібліотек, кожен з яких має свої унікальні особливості, переваги та обмеження. Тому правильний вибір інструментів для конкретного проєкту потребує аналізу та порівняння їх характеристик.

Основною проблемою є визначення оптимального фреймворку або бібліотеки для конкретного проєкту, що забезпечить баланс між продуктивністю, зручністю використання та масштабованістю. Враховуючи швидкий розвиток технологій, розробники постійно потребують актуальної інформації щодо порівняння та оцінки сучасних інструментів для створення вебінтерфейсів. У зв’язку з цим вибір інструментів для розроблення вебінтерфейсів стає критично важливим для успішного виконання проєктів різного масштабу та призначення.

Мета даної роботи полягає у проведенні детального аналізу та порівняння сучасних методів і інструментів розроблення вебінтерфейсів.

Особливу увагу приділено аспектам тестування вебзастосунків та забезпеченню їхньої якості на всіх етапах розроблення.

Значна увага приділяється також аналізу досвіду провідних розробників, що дозволяє виявити найкращі практики та підходи у проєктування вебінтерфейсів.

У першому розділі буде проведено детальний огляд сучасних JavaScript-фреймворків та бібліотек. Розгляданні інструментів для тестування вебзастосунків, включаючи різні види тестування, такі як модульне, інтеграційне та регресійне. Також буде проаналізовано мови програмування для розробки інтерфейсів, що використовуються у сучасній веброзробці.

Другий розділ присвячений аналізу основних архітектурних підходів до проєктування вебінтерфейсів, таких як монолітний, мікросервісний та

компонентний підходи. Будуть розглянуті переваги та недоліки кожного з них, а також етапи життєвого циклу розроблення програмного забезпечення, що дозволяє більш структуровано підходити до процесу створення вебзастосунків.

У третьому розділі буде проведено порівняльний аналіз основних фреймворків для розроблення вебінтерфейсів. Визначення критеріїв для порівняння та за допомогою методу аналізу ієрархії здійснення оцінки цих фреймворків. Це дозволить визначити найбільш відповідний фреймворк для подальшого розроблення вебінтерфейсу користувача.

Четвертий розділ присвячений безпосередньо процесу розроблення інтерфейсу вебзастосунка. Розгляданню етапів розроблення, формуванню вимог до інтерфейсу, структури та організації компонентів у вебзастосунку. Розробці основних компонентів інтерфейсу та взаємодію з серверною частиною через API. Також розгляданню процесу тестування інтерфейсу, включаючи функціональні тести та тести продуктивності. У результаті буде створено ефективний та функціональний інтерфейс вебзастосунка, який відповідає сучасним вимогам.

Таким чином, робота охоплює всі ключові аспекти розробки вебінтерфейсів, від вибору технологій до реалізації та тестування готового продукту. Це дозволяє отримати цілісне уявлення про процес створення сучасних вебзастосунків та визначити оптимальні підходи для досягнення поставлених цілей.

1 ОГЛЯД СУЧАСНИХ ТЕХНОЛОГІЙ ВЕБРОЗРОБЛЕННЯ

1.1 JavaScript-фреймворки та бібліотеки

Різноманітні фреймворки та бібліотеки «JavaScript» відзначаються різною ефективністю та функціональністю, що визначається конкретними потребами користувачів. Для отримання вичерпного уявлення про сучасні тенденції у цій області, важливо проаналізувати результати дослідження використання різних фреймворків та бібліотек «JavaScript» за період між 2016 та 2022 роками [1].

Необхідно також зазначити, що дані аналізу за 2023 рік ще не доступні для загального ознайомлення.

На представленій діаграмі надано образний огляд інструментів із відсотковим відображенням їх використання, яке визначалося через анкетування респондентів. Слід відзначити, що деякі учасники дослідження, які користуються кількома інструментами з тих, що були наведені, мали можливість відзначити декілька варіантів під час опитування.

На діаграмі можна відзначити стабільне підвищення використання react, яке досягло свого піка у 2022 році та далі підтримує позитивну тенденцію росту. Angular демонструє коливання у рівні популярності, досягаючи максимуму у 2019 та 2021 роках. Популярність vue також зросла до 2021 року, але згодом виявила деяку зупинку у зростанні. Паралельно із цими трьома основними фреймворками на ринку присутні також інші платформи, такі як «Svelte», «Ember», «Lit», «Alpine», «Solid,» «Stencil, Preact» та «Quik», хоча їхня популярність виявляється менш помітною у порівнянні з фреймворками «React», «Angular» та «Vue» (рис. 1.1).

Також проаналізуємо діаграму, яка відображає порівняння позитивних та негативних відгуків. Справа від центральної осі розташовані позитивні відповіді, такі як «Хочу вивчити» та «Використовував та буду використовувати знову». Категорії задоволення від використання представлені зеленим та світло-зеленим кольорами, відповідно. Водночас червоний та світло-червоний кольори використані для позначення незадоволення від використання. Зліва розташовані негативні відгуки, наприклад «Не цікаво» та «Використовував, але більше не використовуватиму». Ширина стовпця відображає кількість респондентів, які знають про розглянуту технологію (рис. 1.2).

Аналіз доступних даних свідчить про існування різноманітних фреймворків для реалізації вебінтерфейсів користувача у сучасному ландшафті розроблення програмного забезпечення. Найбільш поширеними серед них є фреймворк «React», який відомий своєю популярністю. Svelte, хоча є новим у сфері, привертає увагу завдяки своєму інноваційному підходу до розроблення. Vue, зі свого боку, славиться своєю простотою в освоєнні та використанні, про що свідчать позитивні відгуки.

Фреймворки «Solid» та «Qwik», які раніше були менш відомими, зараз набувають популярності завдяки своїй простоті. Фреймворк «Preact» стає привабливою альтернативою react завдяки схожому API та меншому розміру фреймворку.

Фреймворк «Lit» славиться своєю простотою у створенні вебкомпонентів, що отримує позитивні відгуки. Фреймворк «Alpine» пропонує мінімалістичний підхід до сучасних функцій JavaScript, уникаючи використання великих фреймворків. Навпаки, фреймворк «Angular», попри свою повноту, може мати високий поріг входу, що призводить до критики.

Фреймворк «Stencil», хоча менш популярний, має потенціал як компілятор для створення вебкомпонентів.

Важливо враховувати, що великий розмір фреймворку, а саме кількість місця яке він займає на диску та пам'яті, може уповільнити завантаження та роботу програми, особливо на повільних мережних з'єднаннях або слабких пристроях. Тому у деяких випадках доцільно використовувати фреймворки з меншим розміром, щоб покращити продуктивність та швидкість завантаження програми.

Фреймворк «Ember» пропонує широкий спектр функцій, але може бути складним для освоєння, що відображається у негативних відгуках.

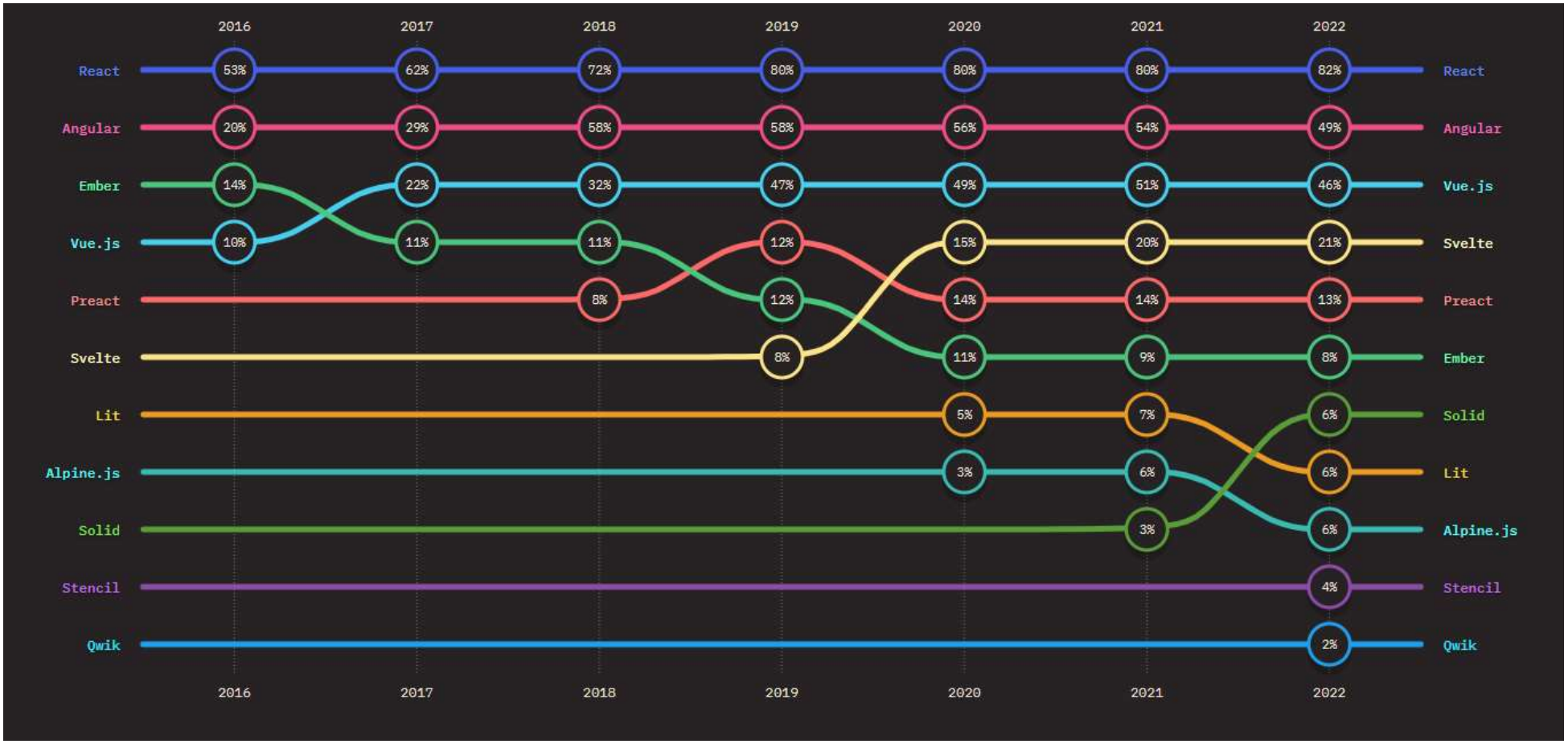

Рисунок 1.1 – Діаграма статистики використання фреймворків та бібліотек «JavaScript»

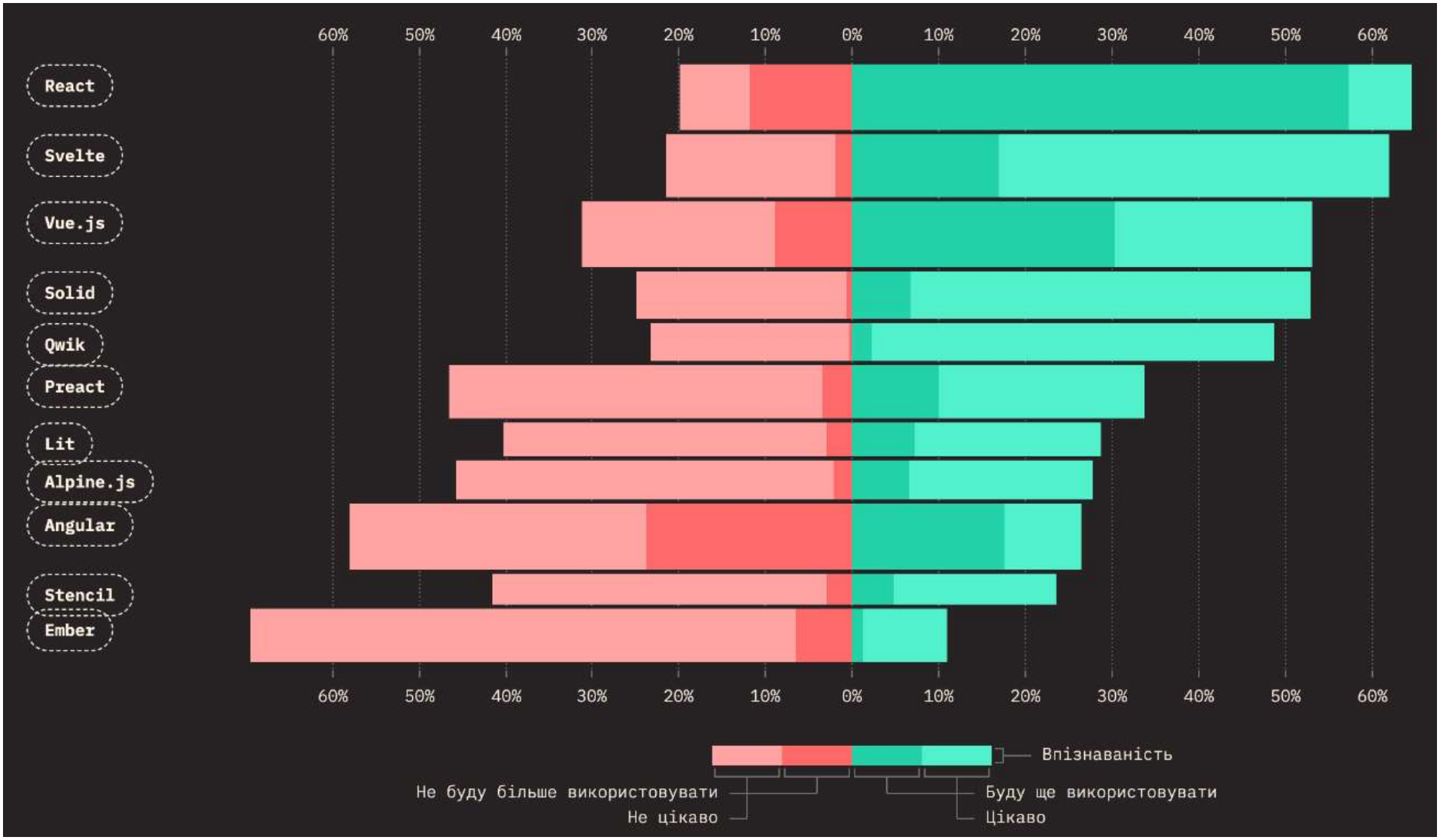

Рисунок 1.2 – Діаграма порівняння позитивних і негативних відгуків на фреймворки та бібліотеки «JavaScript»

Засновуючись на аналізі, який було проведено, можна зробити висновок, що в сучасній індустрії розроблення Front-end інтерфейсу найбільш популярними є фреймворки «React», «Vue» та «Angular».

Цей висновок підтверджується даними щодо завантаження цих інструментів [2] (рис. 1.3), де react займає впевнене лідерство у порівнянні з angular та vue.

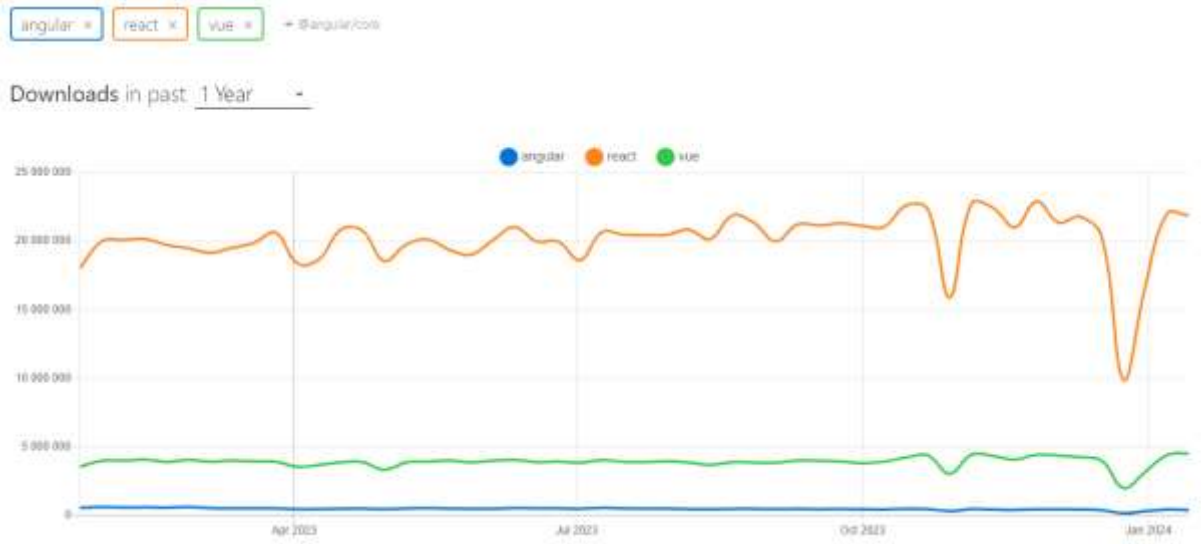

Рисунок 1.3 – Статистика кількості завантажень angular, vue та react

Слід відзначити, що існує позитивна тенденція до збільшення зацікавленості та популярності у розробників у фреймворку «Svelte». Однак, попри це, його поширеність у використанні не досягає рівня конкурентоспроможності першої трійки інструментів на сьогодні.

Після проведення аналізу зазначених вище даних, необхідно провести більш детальне дослідження функціональних характеристик інструментів «React», «Vue» та «Angular», з метою визначення найбільш відповідного для подальшого розроблення інтерфейсу вебзастосунка та досягнення запланованих цілей.

В наш час, вебсередовище відзначається високою активністю в застосуванні JavaScript-фреймворків та бібліотек, які є неодмінним елементом удосконалення процесу розроблення та вже стали стандартом у створенні вебзастосунків, що відзначаються високою продуктивністю та забезпечують користувачам інтерактивність.

1.2 Огляд інструментів тестування

1.2.1 Види тестування у веброзробленні

Вебтестування є важливим етапом в процесі оцінки функціональності вебпрограм, охоплюючи широкий спектр аспектів їхньої роботи. Цей процес містить вивчення ефективності вебзастосунка у відношенні до пошуку та виправлення можливих помилок, а також аналіз його зручності використання для користувачів. Додатково, вебтестування спрямоване на виявлення проблем безпеки, які можуть виникнути в процесі експлуатації програмного забезпечення, та на оцінку загальної продуктивності системи. Такий комплексний підхід дозволяє забезпечити надійність та оптимальну ефективність вебзастосунків у різноманітних умовах їхнього використання [3].

Некоректне тестування комп'ютерної програми може призвести до катастрофічних наслідків, виправлення яких вимагатиме значних матеріальних витрат. Правильне застосування програм для вебтестування або автоматизації вебтестування може бути виправлене наступними методами [3]:

1. Тестування функціональності та особливостей – це випробування, спрямоване на перевірку правильності роботи вебзастосунка, що містить:

- модульне тестування, яке охоплює перевірку невеликих та ізольованих частин застосунку на ранніх стадіях розроблення;
- інтеграційне тестування, спрямоване на виявлення помилок у взаємодії між компонентами програми;
- регресійне тестування, яке перевіряє вибрані тестові сценарії для виявлення областей, які можуть бути пошкоджені внаслідок змін;
- тестування користувацького інтерфейсу, що використовується для виявлення покращень в загальному дизайні та взаємодії з користувачем.

2. Тестування веб-API, цей тип тестування включає посилання запитів до кількох кінцевих точок API для перевірки відповідей.

3. Тестування безпеки, яке ставить за мету перевірку наявності захисту вебзастосунка від потенційних загроз.

4. Тестування продуктивності, спрямоване на перевірку реакції вебзастосунка при високих навантаженнях.

Для аналізу сучасних тенденцій повторно звертаємось до результатів опитування, що охоплює використання різноманітних інструментів для

тестування програмного забезпечення в період з 2016 по 2022 рік (рис. 1.4). Подання діаграми таке ж, як і в рисунку 1.1 та описано у розділі 1.1.

Після аналізу результатів багатобарвної лінійної діаграми, яка візуально відображає динаміку зміни показників використання для різних інструментів протягом часу у відсотках. Можна зробити висновок, що в цьому переліку провідні позиції займають фреймворки «Jest», «Mocha» та «Storybook».

На підставі загальновідомих фактів можна припустити, що jest набув значної популярності завдяки своїй простоті використання, швидкості та інтеграції з численними проєктами, зокрема тими, що використовують фреймворк «React».

Фреймворк «Mocha» є одним з найдавніших фреймворків для тестування JavaScript і має значну спільноту користувачів. Крім того, він пропонує велику гнучкість і сумісність з багатьма бібліотеками та інструментами, що сприяє його популярності.

«Гнучкість» в даному контексті означає здатність фреймворку або інструменту легко адаптуватися до різних умов, потреб чи вимог. Ця характеристика показує, наскільки легко можна використовувати фреймворк у різних сценаріях тестування або з різними іншими інструментами [4].

Зі свого боку, інструмент «Storybook» швидко здобуває популярність завдяки своїм можливостям для розроблення компонентів інтерфейсу. Він пропонує інтерактивне середовище, де розробники можуть ізолювати, тестувати та документувати компоненти.

Крім того, слід зауважити, що з ростом технологій та збільшенням складності вебзастосунків, використання E2E тестування стає все більш поширеним. E2E тестування являє собою процес оцінки функціональності програмного забезпечення шляхом комплексного аналізу всіх етапів робочого процесу програми від початку до завершення. Цей метод забезпечує перевірку коректності роботи програми в реальних умовах, охоплюючи всі взаємодії з інтегрованими компонентами та зовнішніми системами та інтерфейсами [5].

Успішне E2E тестування репрезентує роботу програмного забезпечення в реальному середовищі, відтворюючи типові сценарії використання користувачами та виявляючи потенційні помилки та несумісності. Це передбачає тестування всіх компонентів програмного забезпечення, включаючи інтерфейс користувача, серверні застосунки, базу даних, та будь-які зовнішні системи, які використовуються програмою.

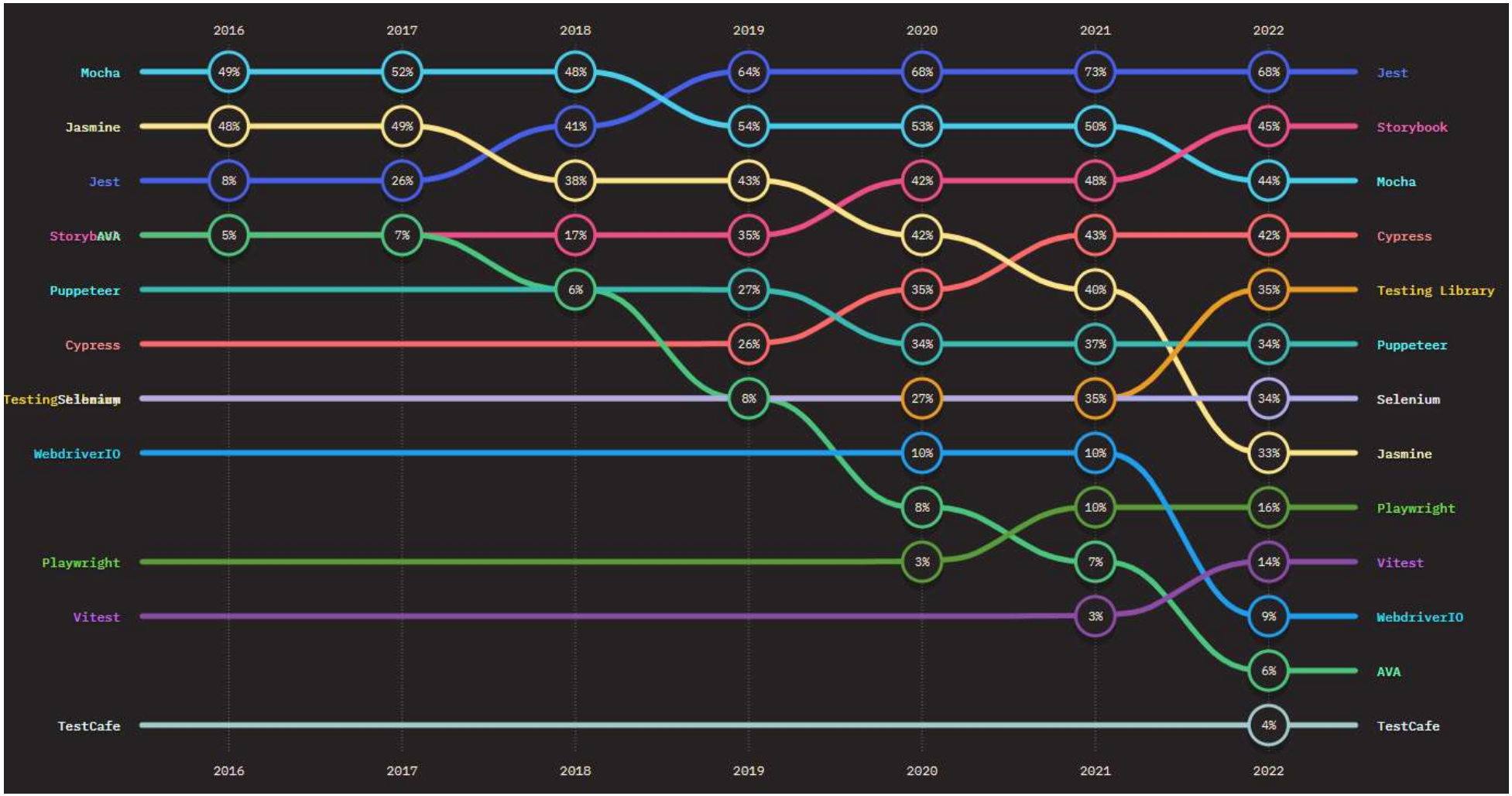

Рисунок 1.4 – Діаграма використання фреймворків та бібліотек тестування

E2E тестування може здійснюватися як вручну тестувальниками, так і за допомогою автоматизованих скриптів. Зазвичай цей вид тестування проводиться після інтеграційного тестування та перед етапом підготовки програмного продукту до випуску.

E2E тестування відрізняється від інтеграційного тестування тим, що E2E тестування оцінює весь робочий процес програми, охоплюючи всі системи та компоненти, у той час, як інтеграційне тестування зосереджено на взаємодіях між конкретними модулями або компонентами програмного забезпечення, щоб переконатися, що вони працюють разом, як передбачалося [6].

Своєю чергою інтеграційне тестування є методом перевірки програмного забезпечення, спрямованим на аналіз взаємодії та обміну даними між різними компонентами або модулями всередині програмного продукту. Основна мета цього виду тестування полягає в виявленні потенційних проблем або дефектів, які можуть виникнути під час інтеграції різних компонентів та їх взаємодії [7].

На рисунку 1.5 зображено діаграму, що ілюструє інтеграційне тестування між двома модулями (Модуль А і Модуль В). У центрі діаграми є зелена область, що показує інтеграцію цих модулів. Це графічне уявлення про те, як інтеграційне тестування перевіряє взаємодію та обмін даними між різними модулями або компонентами програми.

Інтеграційне тестування вважається важливим етапом розроблення програмного продукту, оскільки воно дозволяє виявити потенційні проблеми на ранній стадії, що допомагає уникнути серйозних проблем у подальшому та значних витрат на їх виправлення.

Маємо висновок, що наведені інструменти «Storybook», «Mocha» та «Jest», використовуються у різних сценаріях тестування і мають різноманітні переваги, що робить їх популярними серед розробників. Інструменти, необхідні для досягнення цілей та завдань наукових досліджень, включають функціональні тести та тести продуктивності.

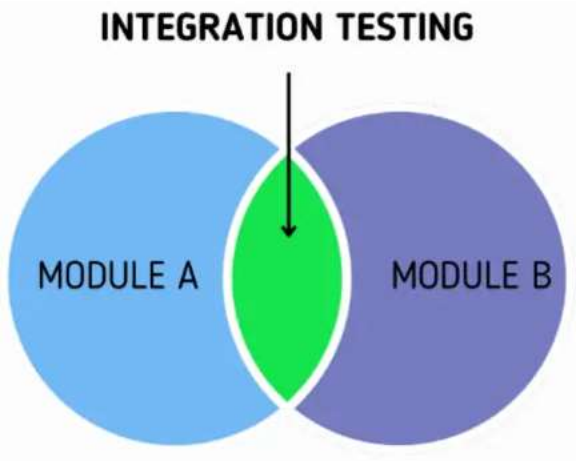

Рисунок 1.5 – Діаграма інтеграції між двома модулями

1.2.2 Функціональні тести та тести продуктивності

Функціональні тести є невіднятною частиною процесу розроблення вебзастосунків, спрямованою на перевірку коректності та функціональності окремих компонентів вебзастосунків. У цьому розділі розглянемо кілька ключових інструментів для автоматизації функціональних тестів, які дозволяють ефективно перевіряти відповідність функцій застосунків визначеним вимогам. Зокрема, буде розглянуто фреймворки та інструменти, такі як «Jest», «Mocha» та «Storybook», кожен з яких надає унікальні можливості для забезпечення якості програмного забезпечення.

Jest, розроблений компанією «Facebook», є важливим інструментом для тестування JavaScript, і відзначається своєю широкою сумісністю з різними технологічними стеками, такими як «Babel», «TypeScript», «Node», «React», «Angular», «Vue» та інші [8].

Його репутація ґрунтується на високій швидкості виконання, легкості використання та розширеній функціональності. Зосереджений на підвищенні продуктивності та надійності коду, jest пропонує кілька переваг.

Основні переваги фреймворку «Jest» полягають у його швидкості виконання, яку забезпечує паралельне виконання тестів та інтелектуальне визначення необхідних файлів для оптимізації часу, простоті конфігурації, завдяки стандартним налаштуванням, можливостям «мокування», що спрощують тестування взаємодії з іншими частинами коду, і використанні «snapshot» тестів для виявлення змін в структурі виводу компонентів чи об'єктів.

Методика тестування з використанням знімків «snapshot» є важливим інструментом у програмному тестуванні, зокрема у розробленні Front-end та аналізі користувацького інтерфейсу. Цей підхід передбачає захоплення стану компонента чи застосунку в певний момент часу та порівняння його з подальшими знімками для виявлення відмінностей. Це дозволяє ефективно виявляти непередбачені зміни та забезпечує стабільність та надійність програмного забезпечення [9].

Проте jest має деякі недоліки, серед яких можна виділити високий рівень специфічності для JavaScript, що може бути не оптимальним для розроблення на інших мовах програмування. Також у деяких сценаріях робота з асинхронним кодом може викликати труднощі, порівняно з іншими фреймворками. Крім того, деякі альтернативні фреймворки можуть надавати більше гнучкості в конфігурації, що означає, що система або програмне забезпечення мають можливість легкої зміни конфігураційних параметрів або налаштувань без необхідності внесення значних змін у вихідний код чи архітектуру системи.

Важливо відзначити, що фреймворк «Jest» виявляє тісний зв'язок з бібліотекою «React» і часто використовується для тестування програмного забезпечення, розробленого з використанням цієї технології. Jest, як і react, входить у набір інструментів, призначених для побудови сучасних вебзастосунків. Ця синергія між jest та react робить процес тестування більш ефективним і зручним. Використання фреймворку «Jest» дозволяє швидко і надійно перевіряти функціональність реактивних застосунків, забезпечуючи високу якість та стабільність програмного забезпечення.

Далі розглянемо фреймворк «Mocha», що відомий своєю широкою популярністю у сфері тестування JavaScript-застосунків, включаючи вебзастосунки. Основні характеристики mocha включають його здатність до розширення та можливість використання різних асинхронних підходів у програмуванні. Фреймворк надає користувачам зручний інтерфейс для створення різноманітних тестів та включає різні інструменти для їхнього виконання, як зазначається в джерелі [10].

Mocha, відомий тестовий фреймворк, має кілька вагомих переваг. Він пропонує гнучкість і розширюваність, що дозволяють обирати різні підходи до тестування, використовуючи різноманітні асинхронні та синхронні методи. Mocha підтримує писання тестів у стилях BDD та TDD, надаючи синтаксис залежно від вибору користувача. Також варто відзначити, що mocha генерує

докладні звіти про виконання тестів, що спрощує процес аналізу результатів тестування. Крім того, фреймворк надає можливість використовувати вбудовані засоби для роботи з асинхронним кодом, що робить тестування вебзастосунків більш зручним.

Однак, поряд з перевагами, існують і недоліки mocha. Наприклад, він не має вбудованої бібліотеки для порівняння значень, тому для цієї мети доводиться використовувати окремі бібліотеки, такі як бібліотека «Chai». Крім того, велика свобода вибору, хоча і може розглядатися як перевага, також може стати недоліком, особливо для початківців, яким важко обрати оптимальний стиль тестування та використовувати всі можливості mocha.

Додатково, цей інструмент відзначається високим рівнем адаптабельності до різноманітних стилів програмування та асинхронних підходів. «Адаптабельність» в даному контексті означає здатність програмного засобу легко пристосовуватися до різних умов, вимог, або змін у середовищі його використання.

Переваги інструмента «Storybook» полягають у декількох ключових аспектах. Він пропонує інтуїтивний та простий у використанні інтерфейс для розробки та документування компонентів інтерфейсу. Storybook дозволяє ізолювати компоненти, що дає змогу розробникам зосередитися на їх створенні та тестуванні без залежності від основного застосунку. Інтерактивне середовище Storybook дозволяє швидко переглядати різні стани компонентів, що значно спрощує їх візуалізацію та налагодження [11].

Крім того, Storybook має інструменти для автоматичного створення документації, що забезпечує високу якість та зручність у використанні компонентів іншими членами команди [11].

Всупереч цим перевагам, варто зазначити деякі недоліки інструмента Storybook. Зокрема, його налаштування та інтеграція можуть потребувати додаткового часу та зусиль, особливо для великих проєктів. Крім того, підтримка специфічних випадків використання може бути обмеженою, що потребує додаткових знань та досвіду від розробників [11].

Загалом, storybook є популярним інструментом для розроблення компонентів інтерфейсу, що сприяє підвищенню продуктивності та якості роботи команди, хоча деякі обмеження можуть вимагати додаткових зусиль для їх подолання.

Розглянуті три важливі фреймворки для функціонального тестування вебзастосунків: «Mocha», «Storybook» та «Jest». Вони мають власні характеристики та переваги. Вибір інструменту для функціонального тестування вебзастосунків буде залежати від вимог та специфіки самого проєкту.

На сьогодні продуктивність та ефективність вебзастосунків стають все більш важливими як для користувачів, так і для розробників. Це обумовлено тим, що швидкість завантаження сторінок, реакція сервера на запити, оптимізація ресурсів та масштабованість системи є ключовими факторами, які впливають на користувацький досвід.

Проведення тестів продуктивності дозволяє виявити слабкі місця вебзастосунків та прийняти необхідні заходи для їх оптимізації та підвищення ефективності. Надійні та швидкодійні вебзастосунки сприяють зменшенню навантаження на сервер шляхом зменшення запитів до нього та передачі лише необхідних даних між клієнтом та сервером. Всі ці аспекти сприяють задоволенню користувачів. Для аналізу продуктивності вебзастосунків використовуються різні інструменти, наприклад, «Lighthouse» та «WebPageTest».

Lighthouse, розроблений компанією «Google», надає докладний звіт про різні аспекти вебзастосунків, такі як швидкість завантаження, доступність та SEO.

WebPageTest, своєю чергою, дозволяє аналізувати швидкість завантаження сторінок в різних умовах та надає розширені можливості вимірювання.

Вибір належного інструменту перед створенням вебзастосунка відіграє значну роль у гарантуванні його ефективності та продуктивності.

1.3 Мови програмування інтерфейсу JavaScript та TypeScript

Важливо обрати мову програмування для розроблення вебінтерфейсів з урахуванням її впливу на ключові аспекти процесу розроблення. Правильний вибір може відобразитися на ефективності створення програмного забезпечення, якості коду, можливостях масштабування проєкту, а також на його безпеці. У сучасному програмуванні для цілей створення вебінтерфейсів найбільш поширеними є дві основні мови: «JavaScript» та «TypeScript».

JavaScript, як фундаментальна мова програмування для веброзроблення, займає центральне місце в сучасному Інтернеті. Він забезпечує динамічність і

інтерактивність вебсторінок, що дозволяє розробникам створювати складні та привабливі інтерфейси для користувачів [12].

JavaScript використовується для обробки реакцій на дії користувачів, створення анімацій, перевірки валідності даних у формах, а також для взаємодії з сервером за допомогою технології AJAX [12].

JavaScript відомий своєю розгалуженою екосистемою фреймворків та бібліотек, що сприяють ефективному розробленню вебзастосунків. Наприклад, react, angular та vue про які йшлося раніше. Вони забезпечують готові інструменти для організації коду, роблячи його більш модульним та зрозумілим. Кожен з цих фреймворків має свої особливості, дозволяючи вибрати той, який найкраще підходить для проєкту.

Всупереч широкому поширенню JavaScript, у нього є свої недоліки. Управління складними проєктами на чистому JavaScript може бути викликом, оскільки він не має вбудованої підтримки для модульності та структуризації коду. На додаток, асинхронний характер мови може створювати проблеми з управлінням потоками даних, роблячи код менш передбачуваним. Однак можна скористатися фреймворками або бібліотеками для інтерфейсу, які дозволяють писати код на вищому рівні абстракції та надають готові рішення для багатьох типових задач, таких як маніпуляція DOM або керування станом застосунку.

Альтернативою є використання мови програмування «TypeScript», що наразі виявляється дуже популярною серед розробників вебзастосунків завдяки своїм унікальним можливостям. TypeScript є суперсетом JavaScript, зберігаючи сумісність з усіма валідними програмами останнього [13].

Однією зі значущих особливостей TypeScript є його статична типізація, що дозволяє визначати типи даних для різних елементів програми, полегшуючи виявлення та виправлення помилок на етапі розроблення [13].

Крім того, TypeScript підтримує сучасні функції ECMAScript, що розширює можливості JavaScript та забезпечує зручність у використанні нововведень мови.

Всупереч перевагам, TypeScript може мати свої недоліки. Наприклад, для використання його необхідно мати компілятор, який перетворює TypeScript у звичайний JavaScript, що може збільшити час розроблення та ускладнити супровід проєкту. Крім того, впровадження TypeScript у вже наявний проєкт може зайняти час та зусилля для конвертації наявного JavaScript-коду.

Додатково, варто відзначити, що розроблення програмного забезпечення на TypeScript часто займає більше часу порівняно з JavaScript. Це пов'язано з

необхідністю дотримуватися статичної типізації та інших особливостей мови, що може вимагати додаткового часу на написання коду. Однак, всупереч цьому, переваги мови «TypeScript» у забезпеченні надійності та обслуговуваності коду можуть переважити цей недолік у випадку розроблення складних вебзастосунків.

Враховуючи переваги, які він надає, ці обмеження зазвичай вважаються дрібними у порівнянні з користю, що може забезпечити TypeScript у розвитку великих та складних вебзастосунків.

Перед початком розроблення вебзастосунка належить звернути увагу на обдуманий вибір мови програмування, що має велике значення в усьому життєвому циклі проєкту. Враховуючи, що обрана мова буде використовуватися упродовж усього процесу розроблення, його важливо аналізувати у контексті його потреб та вимог.

Також важливо, щоб обрана мова програмування відповідала основним цілям проєкту. Вона має забезпечити комфортність у розробленні, оптимальну швидкість виконання та легкість у супроводі, сприяючи досягненню успіху його реалізації.

2 ПРОЦЕС ТА АРХІТЕКТУРНІ ПІДХОДИ ПРОЄКТУВАННЯ ВЕБІНТЕРФЕЙСІВ

2.1 Основні види архітектурних підходів

У сфері веброзроблення виокремлюють три ключові стратегії в побудові архітектури: Монолітний підхід (Monolithic Approach), Мікросервісний підхід (Microservices Approach) та Компонентний підхід (Component-based Approach). Варто зазначити, що вибір конкретної стратегії залежить від вимог і цілей конкретного проєкту. Кожен з цих підходів має свої переваги та недоліки, які варто врахувати при прийнятті рішення щодо архітектурного розвитку проєкту.

2.1.1 Монолітний підхід

Монолітний підхід являє собою традиційний метод розроблення програмного забезпечення, в якому весь код програми зосереджений у єдиному монолітному проєкті. Цей підхід передбачає, що весь функціонал програми, включаючи інтерфейс користувача, серверну систему, базу даних та інші компоненти, розробляється, розвивається і розгортається в рамках одного цілісного програмного продукту [14].

Монолітний підхід до розроблення програмного забезпечення зазвичай є першим вибором завдяки його простоті. Він не потребує складних налаштувань або розгортання декількох окремих сервісів, оскільки весь функціонал концентрується в одному застосунку. Це спрощує процеси розгортання та оновлення. Однак, з ростом функціонала та навантаження можуть виникати проблеми з масштабуванням, оскільки всі компоненти програми знаходяться в єдиному моноліті.

У структурі монолітного застосунку інтерфейсу користувача використовуються HTML, CSS та JavaScript для його формування та обслуговування сервером програми. Цей інтерфейс включає різноманітні компоненти, такі як кнопки, форми, таблиці, зображення тощо, що представлені на сторінці для сприйняття користувачем. JavaScript-код відповідає за обробку подій користувача, валідацію даних, відправлення запитів на сервер та оновлення інтерфейсу відповідно до нових даних. Крім того, існує логіка, що визначає, які сторінки чи компоненти повинні бути відображені для певних URL-адрес. Також

важливо наявність механізмів для зберігання та управління станом програми на стороні клієнта, таких як контекст або сховище стану [14].

У монолітному застосунку роль інтерфейсу користувача включає обробку запитів від користувача та їх подальше направлення на серверну систему для обробки, а також відображення отриманих результатів користувачеві. Крім того, він відповідає за візуальне оформлення програми, сприяючи створенню приємного та зручного користувацького досвіду.

Цей підхід має свої переваги, зокрема, простота у розробленні та впровадженні, централізоване управління та обслуговування програми в цілому, швидке виправлення помилок і внесення змін. Однак існують певні недоліки, серед яких можна відзначити обмежену гнучкість та масштабованість, збільшення часу розроблення при додаванні нового функціоналу, ризик виникнення «спагетті-коду» та складну архітектуру, що може ускладнювати розуміння та редагування програмного коду.

2.1.2 Мікросервісний підхід

Мікросервісний підхід у сфері розроблення програмного забезпечення являє собою архітектурну стратегію, в якій програма розбивається на невеликі та незалежні компоненти, що відомі як мікросервіси. Цей підхід стоїть у протилежності до монолітної архітектури. Кожен мікросервіс являє собою окремий застосунок, що розробляється, розгортається і масштабується незалежно від інших мікросервісів [15].

На рисунку 2.1 графічна ілюстрація демонструє порівняльний розподіл компонентів у двох різних архітектурних моделях програмного забезпечення: монолітної та мікросервісної [16].

У монолітній архітектурі, зображеній зліва, всі елементи програми (інтерфейс користувача, бізнес-логіка і доступ до даних) інтегровані в єдиний блок. Це візуалізується у формі великого кола, що містить у собі три менші кола, позначених як «UI», «Бізнес-логіка» та «Доступ до даних». Ці компоненти об'єднані всередині одного загального кордону, підкреслюючи тісний внутрішній зв'язок.

У мікросервісній архітектурі, представленій праворуч, зазначені елементи розділені на індивідуальні сервіси, які взаємодіють між собою та з інтерфейсом користувача. Це відображається у вигляді великого кола, що представляє UI, з'єднаного з трьома меншими колами, що позначають окремі мікросервіси.

Кожен з цих мікросервісів містить символи, які вказують на різні аспекти бізнес-логіки або доступу до даних. Хоча ці компоненти не інтегровані тісно, вони пов'язані лініями, що є потоками комунікації між ними.

Таким чином, ця діаграма ілюструє суттєві відмінності між монолітною та мікросервісною архітектурою в контексті веброзроблення. У той час як монолітна архітектура забезпечує інтеграцію всіх компонентів програми в єдину структуру, мікросервісна архітектура дозволяє розділити застосунок на окремі, незалежні сервіси, сприяючи більш гнучкому та масштабованому розгортанню та управлінню системою.

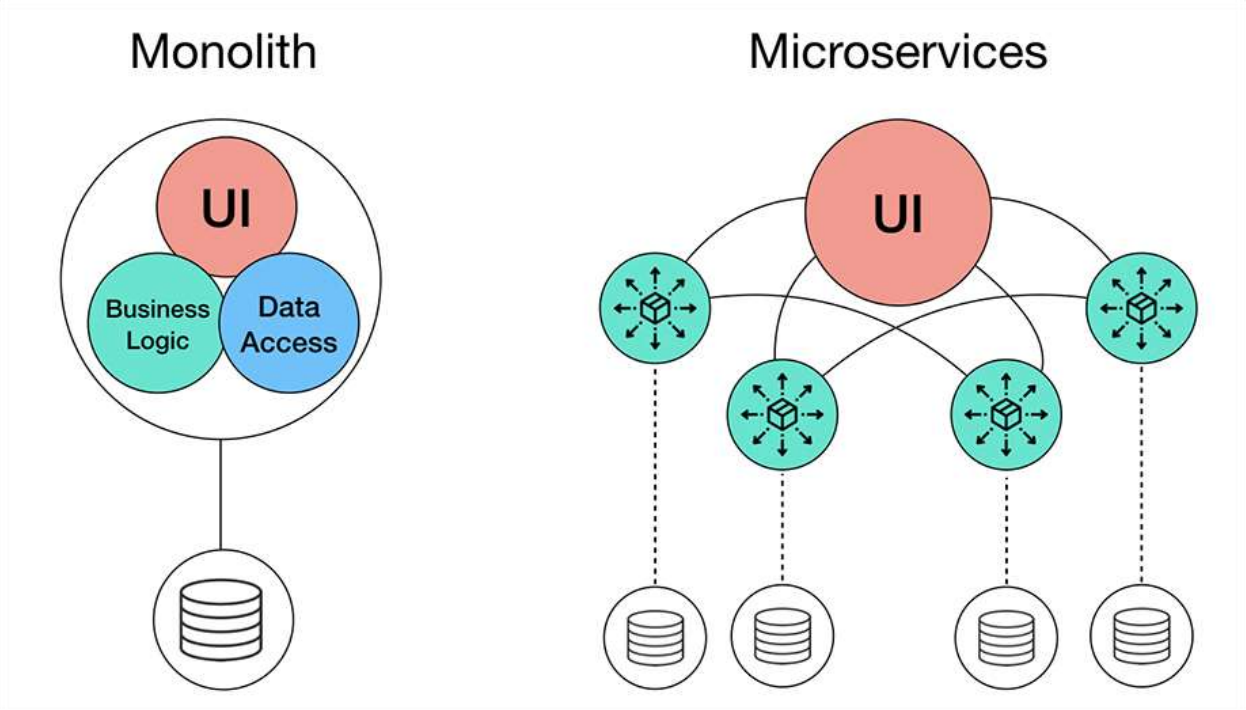

Рисунок 2.1 – Відмінності монолітної та мікросервісної архітектури

Мікросервісний підхід в архітектурі програмного забезпечення базується на розкладанні застосунку на невеликі, автономні блоки, які легко масштабуються. Кожен з цих блоків вирішує конкретне завдання чи функціональність, спрямовуючи зусилля розробників на конкретні бізнес-завдання і спрощуючи підтримку та супровід коду [15].

Центральним елементом є використання API для взаємодії між блоками, що дозволяє їм працювати незалежно та обмінюватися даними без прив'язки до внутрішньої реалізації інших блоків. Кожен мікросервіс може бути розгорнутий

та масштабований окремо, що забезпечує гнучкість та ефективність системи. Також ця архітектура дозволяє масштабувати кожен блок окремо відповідно до поточного навантаження, що забезпечує високу продуктивність і доступність системи.

Можна використовувати різні технології та інструменти для кожного мікросервісу, що сприяє оптимізації продуктивності та реагуванню на зміни у вимогах до системи. Мікросервісна архітектура також сприяє розподілу завдань між невеликими та незалежними командами, що спрощує управління проєктом та підвищує продуктивність. Проте, слід зазначити, що зі зростанням числа мікросервісів збільшується складність управління та розгортанням системи, що вимагає правильної «оркестрації» та управління для забезпечення узгодженої роботи та цілісності системи.

У мікросервісній архітектурі інтерфейс користувача розбивається на відокремлені компоненти, кожен з яких відповідає певній функціональності та мікросервісам серверної системи. Взаємодія між цими компонентами та відповідними мікросервісами забезпечує ефективну роботу програми в цілому. Наприклад, компонент, що відповідає за управління профілем користувача, взаємодіє з мікросервісом, що керує авторизацією та обліковими записами. Такий підхід до організації архітектури дозволяє забезпечити високий рівень функціональності та ефективну взаємодію різних компонентів програмної системи.

Маючи незалежні та ізольовані компоненти, можна працювати над ними незалежно один від одного, що сприяє прискоренню процесу розроблення та оновлення. Однак, необхідно забезпечити збереження єдиного стилю та дизайну інтерфейсу для всієї програми. Використання бібліотек компонентів і систем дизайну сприяє стандартизації зовнішнього вигляду і поведінки компонентів, що своєю чергою сприяє створенню єдиного досвіду користувача. Це означає, що стандартизація зовнішнього вигляду і поведінки компонентів допомагає створити єдиний досвід користувача, оскільки користувачі будуть зустрічатися зі схожими або однаковими елементами та взаємодіяти з ними однаковим чином в різних частинах вебпрограми. Це дозволяє об'єднати різні елементи інтерфейсу або сторінки в одному застосунку, які були розроблені різними командами з використанням різних фреймворків (рис. 2.2).

На зображенні представлено вебінтерфейс, який ілюструє приклад використання різних фреймворків в одному застосунку: «Angular», «React» та

«Vue». Кожен фреймворк використовується для розроблення окремого мікросервісу: «Products» на angular, «Orders» на react та «Recommendations» на vue. Це відображає концепцію мікросервісної архітектури, де кожна частина вебсторінки розроблена за допомогою різного фреймворку.

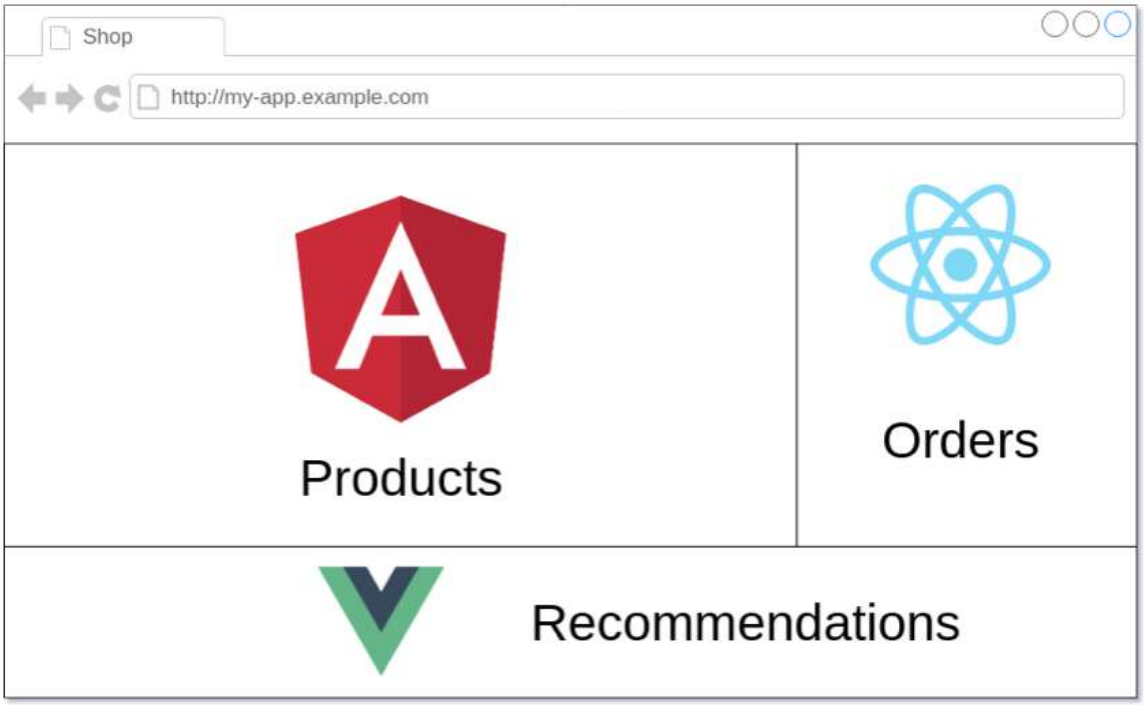

Рисунок 2.2 – Приклад використанням різних фреймворків в одному вебзастосунку

Мікросервісна архітектура відзначається низкою переваг, що визначаються її основними принципами. Гнучкість та масштабованість структури досягаються завдяки незалежності кожного мікросервісу. Це дозволяє системі ефективно адаптуватися до змін у вимогах та навантаженні, не втручаючись у функціонування інших компонентів. При цьому, легкість розроблення та оновлення відображається у можливості окремо розробляти та підтримувати кожен мікросервіс, що спрощує процес та прискорює введення продукту на ринок.

Далі, реалізація незалежних технологічних стеків дозволяє використовувати різні інструменти та технології для кожного мікросервісу, що сприяє оптимізації продуктивності та знижує витрати. Стійкість та надійність системи забезпечуються ізоляцією та незалежністю мікросервісів: збій одного

компонента не впливає на роботу інших, що мінімізує ризики та забезпечує високу доступність системи.

До інших переваг можна віднести легке масштабування команд шляхом розділення розроблення та підтримки на невеликі автономні групи, що сприяє підвищенню продуктивності та якості коду. Однак, разом з цим, існують і недоліки. Складність управління та «оркестрації» мікросервісами вимагає додаткових інструментів та ресурсів.

Розроблення та налагодження ускладнюються через необхідність керування безліччю незалежних компонентів, а також додаткові витрати на інфраструктуру для підтримки мікросервісної архітектури. Складність забезпечення узгодженості та цілісності даних потребує розроблення ефективних методів синхронізації.

Впровадження мікросервісної архітектури вимагає змін у культурі та процесах розроблення, включаючи впровадження DevOps практик та навчання персоналу [15].

В кінцевому підсумку, мікросервісний підхід до розроблення інтерфейсів вебзастосунків дозволяє створювати гнучкі, масштабовані та надійні інтерфейси, але для успішної реалізації цього підходу необхідне ретельне планування, розроблення та керування всіма компонентами системи, а також забезпечення їх ефективної інтеграції та взаємодії.

2.1.3 Компонентний підхід

Компонентна архітектура – це комбінація монолітної та мікросервісної, де програма розбита на незалежні компоненти, але вони можуть бути розгорнуті разом. Компонентна архітектура забезпечує гнучкість, оскільки кожен компонент може бути розроблений, протестований та оптимізований незалежно. Це також спрощує масштабування, оскільки окремі компоненти можуть бути масштабовані відповідно до їх індивідуальних вимог [17].

Однак варто відзначити, що, хоча компонентна архітектура може включати деякі аспекти монолітної та мікросервісної архітектур, вона не є їхньою прямою комбінацією. Натомість вона є окремим підходом до проєктування та розроблення програмного забезпечення, який може бути адаптований для задоволення конкретних вимог застосунку.

Кожен з компонентів вважається незалежною сутністю інтерфейсу, здатною працювати відокремлено від інших елементів системи. Діапазон таких

компонентів може бути від базових елементів інтерфейсу, таких як кнопки та поля введення, до складніших конструкцій, наприклад, календарів, карточок товарів або модальних вікон. Важливою особливістю кожного компонента є його самодостатність, що дозволяє йому використовуватись у різних контекстах програми, не залежно від місця його використання.

Одна з ключових переваг компонентного підходу полягає у можливості використання компонентів у різних частинах програмного забезпечення без необхідності повторного розроблення. Компоненти створюються лише один раз, що спрощує процес розроблення, економить час і зусилля розробників, а також сприяє підтримці коду надалі.

Компоненти програмного забезпечення часто впорядковані в ієрархічну структуру, де вищерівневі компоненти можуть містити в собі низькорівневі. Це підтримує модульність та організованість коду, що дозволяє розбивати складні інтерфейси на простіші, керовані частини. У кожного компонента може бути власна логіка, стан та представлення, що робить їх більш гнучкими та здатними до масштабування.

У процесі розроблення програмного забезпечення важливо враховувати можливість індивідуального розроблення та тестування кожного компонента незалежно від інших. Цей підхід сприяє уникненню взаємних залежностей між компонентами та гарантує більшу стабільність та надійність системи в цілому. Перевагою такого підходу є те, що проблеми, які виникають в одному компоненті, не мають впливу на інші, що спрощує виявлення та виправлення помилок, а також полегшує процес налагодження системи.

Підхід, заснований на компонентах, сприяє стандартизації інтерфейсу програми. Кожен компонент визначається чіткою інтерфейсною специфікацією та уявленням про його функціонування. Це спрощує процес супроводу та масштабування програмного забезпечення, оскільки новим розробникам легше розуміти структуру та взаємодію компонентів [17].

У сфері веброзроблення для втілення концепції компонентного підходу часто використовують спеціалізовані фреймворки, які розглядались у розділі 1.1, зокрема react, vue або angular. Ці фреймворки володіють набором інструментів, спрямованих на ефективне створення, організацію та управління компонентами вебзастосунка.

Підхід, що базується на компонентах, відомий своїми перевагами, однак, важливо врахувати його недоліки під час розроблення програмного забезпечення.

Серед цих недоліків можна відзначити складність управління станом, особливо при роботі з великими та складними компонентами, особливо у разі вкладених структур. Також слід звернути увагу на можливість збільшення обсягу надлишкового коду у компонентах, що може виникнути при неправильному проєктуванні та використанні. Хоча відомо, що компонентний підхід зазвичай сприяє підвищенню продуктивності, некоректне його застосування може призвести до погіршення результатів через надмірний рендеринг або складні вкладені структури. Додатковими труднощами можуть бути проблеми з навчанням та впровадженням цього підходу, особливо для тих, хто не має достатнього досвіду у використанні відповідних технологій та фреймворків. Також, слід відзначити, що складність тестування компонентів також може бути проблемою, оскільки вони можуть функціонувати незалежно від інших частин програми, що вимагає розроблення спеціальних стратегій та методів тестування.

Узагальнено, компонентний підхід являє собою ефективний та гнучкий метод для розроблення вебінтерфейсів, який сприяє створенню модульних, повторно використовуваних компонентів, що легко підтримуються. Проте успішна реалізація цього підходу вимагає дбайливого проєктування, коректного використання технологій та фреймворків, а також належного навчання та досвіду розробників проєкту.

2.1.4 Порівняння переваг та недоліків архітектур

Проведемо порівняння переваг та недоліків різних архітектур вебзастосунків. Архітектура вебпрограми відіграє ключову роль у його функціональності, продуктивності, масштабованості та загальної надійності. Розглянемо три основні архітектури, про які йшлося раніше: монолітну, мікросервісну та компонентну, досліджуючи їх основні переваги та недоліки.

Монолітна архітектура характеризується тісним зв'язком всіх компонентів програми, що може полегшити розроблення та тестування, але також може призвести до проблем з масштабованістю та гнучкістю. Мікросервісна архітектура, навпаки, передбачає поділ програми на безліч незалежних сервісів, що може покращити масштабованість та гнучкість, але також може ускладнити розроблення та тестування. Нарешті, компонентна архітектура поєднує деякі аспекти обох архітектур, дозволяючи розбивати застосунок на незалежні компоненти, які можуть бути розгорнуті разом чи окремо.

У рамках аналізу архітектурних концепцій застосунків порівняємо їх за наступними критеріями.

Розгляд кожного критерію вебзастосунків є необхідним для повного розуміння їхнього функціонування та ефективного управління ними. Архітектура програми визначає його структуру, включаючи компоненти, їх взаємозв'язки та властивості. Розмір програми належить до загального обсягу коду та ресурсів, необхідних для його функціонування. Розгортання описує процес встановлення та налаштування програми у певному середовищі, включаючи монолітне або мікросервісне розгортання. Масштабованість визначає здатність програми адаптуватися до збільшення навантаження шляхом горизонтального або вертикального масштабування. Розвиток технологій стосується постійного покращення та оновлення використовуваних технологій, таких як мови програмування та фреймворки. Відмовостійкість визначає здатність програми продовжувати працювати нормально навіть при виникненні збоїв або помилок, включаючи стратегії обробки помилок та механізми відновлення. Технічне обслуговування і планування включає підтримку програми в робочому стані, включаючи виправлення помилок, оновлення компонентів і планування майбутніх поліпшень [14, 15, 17].

При аналізі архітектурних рішень для інформаційних систем, важливо врахувати кілька критеріїв.

Першим з них є сама структура архітектури. У монолітній архітектурі всі компоненти мають тісний зв'язок та організовані на одному рівні, у той час, як архітектури мікросервісів та компонентної архітектури мають багаторівневу структуру, де компоненти можуть функціонувати незалежно.

Другий критерій – це розмір та комплексність архітектури, що визначається її масштабами. Монолітна архітектура зазвичай має великі розміри через цілісний характер зв'язків між компонентами, тоді як у мікросервісних та компонентних архітектурах розмір може бути меншим завдяки слабкій зв'язаності.

Третій критерій стосується процесу розгортання. У монолітній архітектурі розгортання зазвичай відбувається як єдине ціле, тоді як у мікросервісних та компонентних архітектурах окремі компоненти або сервіси можна розгортати незалежно.

Четвертий критерій – це масштабованість системи. У монолітній архітектурі горизонтальне масштабування може бути складним через взаємозалежність компонентів.

П'ятий критерій – удосконалення, що також впливає на обрану архітектуру. У монолітній архітектурі удосконалення може бути обмеженим строгим контекстом, тоді як у мікросервісних архітектурах воно спрощене завдяки слабкій зв'язаності.

Шостий критерій – відмовостійкість, що відображає здатність системи працювати при виникненні помилок. У монолітній архітектурі відмова одного компонента може призвести до збою всієї системи, тоді як у мікросервісних архітектурах це має менший вплив.

Сьомий критерій – це технічне обслуговування та планування, яке включає процеси підтримки та розвитку. У монолітній архітектурі цей процес може бути складним через великі розміри та взаємозалежність компонентів, тоді як у мікросервісних архітектурах він спрощений завдяки незалежності.

Надамо дані у вигляді порівняльної таблиці (табл. 2.1). Важливо відзначити, що вибір архітектури вебзастосунка має ґрунтуватися на конкретних вимогах та цілях проєкту. Немає кращої чи гіршої архітектури, кожна архітектура має свої сильні та слабкі сторони та може бути більш менш відповідною залежно від контексту. Тому важливо ретельно аналізувати вимоги та цілі проєкту перед вибором архітектури.

З урахуванням зазначених аргументів, пропонується застосування компонентного підходу у розробленні вебінтерфейсу для практичної складової дослідження. Цей підхід вважається найбільш вигідним та ефективним у більшості випадків створення вебінтерфейсів завдяки його модульності, гнучкості та можливості повторного використання компонентів. Використання компонентів сприяє структурованості коду, спрощує його обслуговування та розширення функціональності програми. Крім того, компонентний підхід сприяє стандартизації інтерфейсу.

Таблиця 2.1 – Порівняльний аналіз архітектур

Критерії	Архітектури		
	Монолітна архітектура	Мікросервісна архітектура	Компонентна архітектура
Структура	Одноріvнева структура	Багаторівнева структура	Багаторівнева структура
Розмір та комплексність	Великий, всі компоненти цільно з’єднані	Маленький, слабко з’єднані компоненти	Залежить від розміру та складності компонентів
Розгортання	Розгортається як єдине ціле	Окремі сервіси можуть розгортатися незалежно	Компоненти можуть розгортатися незалежно
Вертикальна та горизонтальна масштабованість	Горизонтальне масштабування може бути складною через залежності	Легше масштабувати по горизонталі через відсутність залежностей	Масштабування залежить від взаємодії компонентів
Удосконалення	Складне через строгий контекст	Спрощене через слабкі зв’язаності сервісів	Залежить від взаємодії та залежності компонентів
Відмовостійкість	Відмова одного компонента може призвести до збою всієї системи	Відмова одного сервісу не впливає на роботу інших	Відмова одного компонента може вплинути на роботу інших, через залежність від способу взаємодії
Технічне обслуговування та спланування	Складне через великий розмір та взаємозалежність компонентів	Спрощене через незалежність сервісів	Залежить від взаємодії та залежності компонентів

2.3 Етапи життєвого циклу розроблення програмного забезпечення

Розроблення програмного забезпечення являє собою послідовний процес, що містить в собі ряд фаз, що відображають логічний порядок дій від концепції до завершення проєкту. Незалежно від вибраної моделі розроблення, ці фази можуть варіюватися, але загальні принципи залишаються нещільно схожими. Основні фази включають:

Першою фазою є аналіз вимог, під час якого визначаються потреби та очікування користувачів стосовно програмного забезпечення, а також аналізується його функціональність та необхідні характеристики [18].

Далі йде фаза проєктування, під час якої визначається архітектура програмного забезпечення та розробляються діаграми та схеми, що відображають його структуру та взаємозв'язки між компонентами [18].

Потім наступає фаза розроблення, під час якої програмне забезпечення фактично розробляється на основі вимог та проєктування, з виробництвом коду, його тестуванням та інтеграцією компонентів [18].

Після цього йде фаза тестування, коли програмне забезпечення піддається тестуванню для перевірки його коректності, надійності та здатності відповідати вимогам користувачів, включаючи як ручне, так і автоматизоване тестування [18].

Після успішного завершення тестування настає фаза впровадження, під час якої програмне забезпечення готове до використання в реальному середовищі, включаючи установку, конфігурацію та навчання користувачів [18].

Остання фаза – це підтримка та обслуговування, під час якої програмне забезпечення продовжує підтримуватися та оновлюватися відповідно до потреб користувачів та змін у середовищі його використання [18].

Проведемо порівняння шести основних моделей розроблення програмного забезпечення: каскадної, ітеративної, спіральної, інкрементної, V-подібної, гнучкої та скрам-моделі. Кожна з цих моделей буде розглянута з погляду її переваг та недоліків, а також буде надано рекомендації щодо вибору найбільш відповідної моделі залежно від характеристик та вимог конкретного проєкту. Результат дослідження занесемо до таблиці 2.2. Розміщення інформації в осередках таблиці, розфарбованих відповідно до результату (де «Так» – це зелений, «Ні» – це червоний), сприяє більш ефективній візуалізації даних та покращенню сприйняття інформації.

Моделі управління проєктами відрізняються своїми підходами та методологіями. Каскадна модель відзначається послідовністю етапів, що робить її простою у розумінні та управлінні, зокрема, коли вимоги до проєкту чітко визначені. Однак, цей підхід може виявитися неефективним у випадку зміни вимог, яка може вимагати повернення до початкових етапів розробки [19].

Ітеративна модель натомість дозволяє команді концентруватися на невеликих частинах проєкту, спрощуючи управління та зменшуючи ризики. Проте, цей підхід може бути складним у випадку недостатнього досвіду з такою методологією [19].

Спіральна модель акцентує на ранньому виявленні та усуненні ризиків, що сприяє покращенню якості продукту. Однак, її складність та вартість управління можуть бути вирішальними факторами, особливо в масштабних проєктах [19].

Інкрементна модель дозволяє швидко отримати робочий продукт шляхом поетапного впровадження його частин, покращуючи зворотний зв’язок та клієнтське задоволення. Проте, цей підхід може бути неефективним у випадку нечітких або часто змінюваних вимог до проєкту [19].

Модель V-образного управління проєктами пропонує систему контролю на кожному етапі, що спрощує управління. Однак, якщо вимоги до проєкту змінюються, цей підхід може стати неефективним [19].

Гнучка модель надає гнучкість в управлінні змінами та акцентує на постійному вдосконаленні продукту. Проте, вона може бути викликана управлінськими труднощами у відсутність досвіду в гнучкому середовищі [19].

Scrum-модель, зі своєю гнучкою методологією, підкреслює контроль якості у всьому процесі розробки, що сприяє покращенню комунікації та співпраці. Проте, вона може виявитися складною у випадку відсутності досвіду з її впровадження та активної участі клієнта [19].

Загальний висновок з таблиці 2.2 полягає в тому, що кожна модель має свої сильні та слабкі сторони, і вибір найбільш відповідної залежить від конкретних умов проєкту, його характеристик та вимог. Це допомагає краще розуміти властивості кожної моделі, визначає їх сильні та слабкі сторони, а також надає підстави для обґрунтованого вибору моделі відповідно до конкретних умов та вимог проєкту. Наприклад, для проєктів з чіткими та стабільними вимогами каскадна модель може бути ефективною, тоді як для проєктів зі значними змінами у вимогах та швидким випуском продукту на ринок гнучка або скрам-модель може бути більш відповідною. Важливо враховувати потреби та особливості

проєкту, а також залучати команду до процесу вибору моделі для забезпечення оптимального результату.

Каскадна модель (рис. 2.3) розроблення програмного забезпечення підходить для проєктів, що характеризуються чіткими та стабільними вимогами на початку життєвого циклу проєкту. Вона є ефективною у випадках, коли вимоги можуть бути чітко сформульовані та не піддаються значним змінам протягом розроблення. Крім того, каскадна модель спрощує процес планування, контролю та управління проєктом завдяки своїй стабільній структурі та послідовності етапів.

Таким чином, вибір каскадної моделі рекомендується для проєктів, які відповідають цим характеристикам. Тобто і підходить дослідницькому проєкту. Вона забезпечить ефективний та організований підхід до розроблення програмного забезпечення, що дозволить досягти успішного результату.

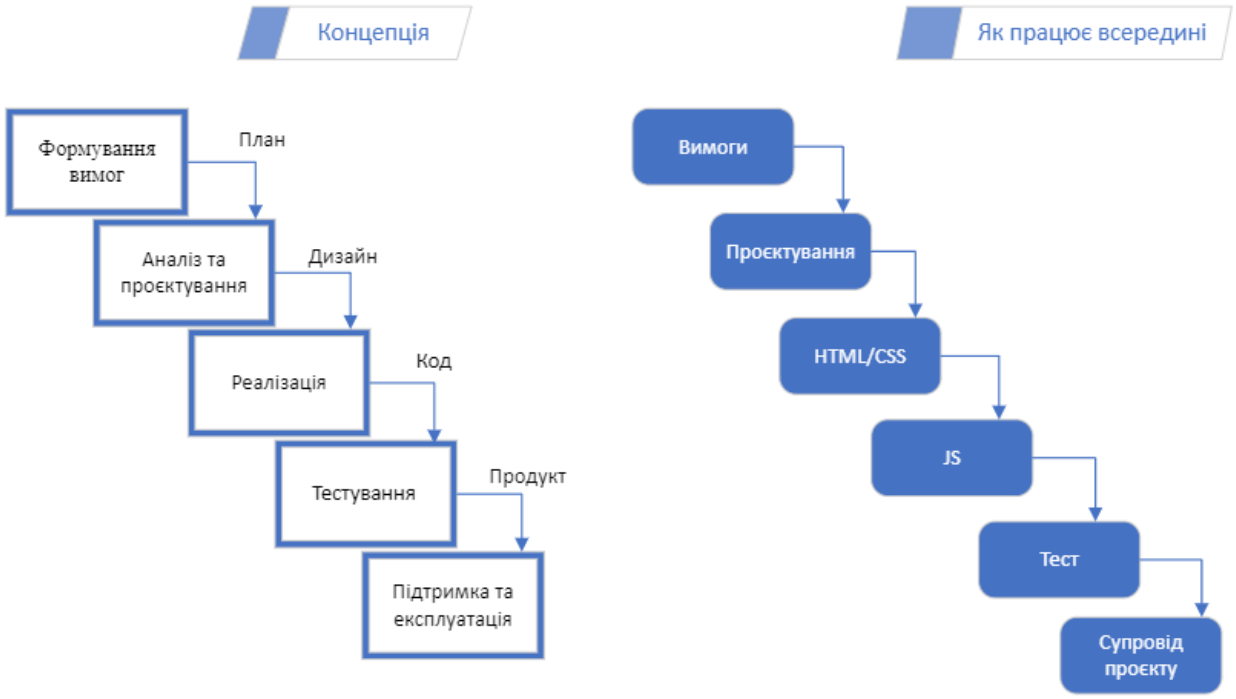

Рисунок 2.3 – Діаграма каскадної моделі

Таблиця 2.2 – Порівняльний аналіз моделей життєвого циклу розроблення програмного забезпечення

Модель / Критерії	Каскадна	Ітеративна	Спіральна	Інкрементна	V-подібна	Гнучка	Скрам
Стабільність вимог	Так	Ні	Так	Ні	Ні	Ні	Ні
Раннє виявлення конфліктів між вимогами	Ні	Так	Так	Так	Ні	Так	Так
Простота застосування	Так	Ні	Ні	Ні	Ні	Ні	Ні
Швидкий випуск MVP	Ні	Так	Так	Так	Ні	Так	Так
Моніторинг замовника	Так	Так	Так	Так	Ні	Так	Так
Розбиття великого обсягу робіт	Так	Так	Так	Так	Ні	Так	Так
Замовник може коригувати	Так	Так	Так	Так	Ні	Так	Так
Можливість нейтралізувати ризики	Так	Так	Так	Так	Так	Так	Так
Поступове звикання до нової технології	Так	Так	Так	Так	Ні	Так	Так
Сувора етапізація	Так	Ні	Ні	Ні	Так	Ні	Ні
Вдосконалений тайм-менеджмент	Ні	Ні	Ні	Ні	Так	Ні	Ні
Швидке прийняття рішень	Ні	Так	Так	Так	Ні	Так	Так
Полегшена робота з документацією	Ні	Ні	Ні	Ні	Ні	Так	Ні
Швидке додавання функціонала	Ні	Ні	Ні	Ні	Ні	Ні	Так
Можливість зміни вимог	Так	Так	Так	Так	Ні	Так	Так

3 ВИБІР ІНСТРУМЕНТІВ ДЛЯ РОЗРОБЛЕННЯ ВЕБІНТЕРФЕЙСУ КОРИСТУВАЧА

3.1 Загальний огляд фреймворків

3.1.1 React

React - це провідний JavaScript-фреймворк, що зосереджений на розробці інтерфейсів користувача для вебзастосунків. Цей фреймворк був розроблений компанією «Facebook» для створення інтерфейсу своєї соціальної мережі. На сьогодні react широко використовується в різних проєктах, від невеликих вебсайтів до великих корпоративних програм [20].

Основний принцип react полягає в компонентному підході до розроблення інтерфейсів. Весь інтерфейс користувача розбивається на невеликі, незалежні компоненти. Кожен компонент – це окрема частина інтерфейсу або функціональність, що виконує певні завдання. Компоненти react, подібно до будівельних блоків, формують основну структуру застосунку. Компонент може включати структуру HTML-елементів для відображення вмісту на сторінці, а також логіку JavaScript для обробки різноманітних дій користувача.

Компонентний підхід в react дозволяє зробити код модульним та перевикористовуваним. Це дозволяє створювати компоненти, які виконують конкретні функції, і використовувати їх у різних частинах програми, уникнувши дублювання коду. Крім того, цей підхід сприяє полегшенню підтримки коду, оскільки кожен компонент відповідає лише за свою частину інтерфейсу. Це особливо важливо для великих проєктів, де код може бути складним та об’ємним.

В основі концепції «JSX» лежить ідея об’єднання HTML-шаблонів і JavaScript-логіки в одному файлі. Це дозволяє створювати незалежні компоненти, які включають як розмітку, так і логіку візуалізації даних. Використання JSX в react має кілька суттєвих переваг. Цей підхід полегшує читання коду, адже структура інтерфейсу описується у форматі, подібному до HTML.

React відзначається великою спільнотою розробників та широкою екосистемою сторонніх бібліотек та інструментів. Ця величезна та активна спільнота дозволяє розробникам обмінюватися досвідом, ділитися знаннями та знаходити рішення для різних завдань.

Разом з react активно розвивається та розширюється екосистема сторонніх бібліотек та інструментів, спрямованих на полегшення веброзроблення. Розробники можуть скористатися безліччю готових компонентів, бібліотек станів (таких як «Redux», «MobX») та інструментів для налагодження та профілювання програм.

React підходить як для створення простих статичних вебсайтів, так і для складних динамічних програм з великим обсягом даних. Його гнучкість і масштабованість роблять його одним із найпопулярніших виборів серед розробників вебзастосунків.

3.1.2 Angular

Angular - це впливовий фреймворк для розроблення вебзастосунок, створений та підтримуваний компанією «Google». Цей фреймворк надає широкий арсенал інструментів та функцій, спрямованих на розроблення великих та складних проєктів. Ці характеристики роблять angular одним з найбільш популярних виборів серед розробників у галузі веброзроблення [21].

Основний принцип angular полягає в архітектурі Model-View-Controller (MVC), що розподіляє застосунок на три основні компоненти: модель (Model), представлення (View) та контролер (Controller). Модель в angular інкапсулює дані та бізнес-логіку за допомогою класів TypeScript, що описують дані та їх взаємозв’язки. Представлення відповідає за візуалізацію даних для користувачів і реалізується у вигляді HTML-шаблонів, які можуть містити директиви angular для динамічного відображення даних та управління інтерфейсом. Контролер, знаходячись між моделлю та представленням, керує їх взаємодією.

Використання архітектури MVC в angular забезпечує чистий та організований код, спрощує підтримку та розширення програми, покращує масштабованість проєкту, полегшує процес тестування, що своєю чергою забезпечує стабільність та надійність програмного забезпечення.

Angular використовує TypeScript, що забезпечує переваги у суворій типізації та надійності коду. TypeScript вводить статичну типізацію до JavaScript, що дозволяє виявляти помилки на етапі розроблення. Крім того, код стає більш підтримуваним та зрозумілим.

Angular, як сучасний інструмент розроблення програмного забезпечення, наділяє розробників широким спектром вбудованих функцій та інструментів, які сприяють ефективній і продуктивному розробленню програм. Серед них варто

відзначити маршрутизацію, валідацію форм, виконання HTTP-запитів, управління станом та інші.

Маршрутизація в angular відіграє ключову роль у створенні навігаційної структури застосунку, надаючи можливість налаштовувати відображення різних компонентів залежно від URL-адреси. Валідація форм в angular виконується шляхом використання вбудованого механізму, який дозволяє задавати правила валідації для кожного поля форми. HTTP-запити в angular здійснюються за допомогою інструменту «HttpClient», який надає зручні методи для посилання різних типів запитів до сервера. Управління станом в angular спрощує процес збереження та керування даними, забезпечуючи централізований підхід до їх управління.

У фреймворку «Angular» існує інструмент, відомий як Angular CLI (Command Line Interface), який призначений для автоматизації процесу розроблення, компіляції та розгортання angular-застосунків. Angular CLI забезпечує ряд команд, спрямованих на різні аспекти розроблення, такі як створення нових проєктів, додавання компонентів, тестування застосунків тощо. Це дозволяє значно спростити та прискорити процес розроблення, а також забезпечує стандартизацію процесу роботи над проєктом.

Створення великих та складних проєктів є однією з ключових переваг фреймворку «Angular», який пропонує архітектуру, ґрунтовану на принципах компонентної архітектури та ін’єкції залежностей. Ця архітектура сприяє структуруванню застосунку у вигляді компонентів, кожен з яких має визначену функціональність та відповідальність. Такий підхід сприяє упорядкуванню коду, спрощує його розуміння та підтримку, особливо у масштабних проєктах. Завдяки ефективній архітектурі та механізмам оптимізації, angular забезпечує високу продуктивність навіть у великих застосунках. Розроблено angular з урахуванням масштабованості застосунків будь-яких розмірів. Механізми поділу програми на модулі та підмодулі дозволяють ефективно керувати кодом та ресурсами при збільшенні розміру. Підтримка асинхронного завантаження модулів та лінивого завантаження в angular сприяє підвищенню продуктивності та оптимізації використання ресурсів програми. Це особливо важливо для великих застосунків, де ефективне використання ресурсів може значно покращити продуктивність та відгук користувача.

Всі ці характеристики роблять angular корисним інструментом для розроблення вебзастосунків будь-якого розміру та складності.

3.1.3 Vue

Vue - це JavaScript фреймворк, який спрямований на розроблення інтерфейсів користувача. Його простота в освоєнні та використанні робить його привабливим для різноманітних розробників. Цей фреймворк надає інтуїтивно зрозуміле та легко запам’ятовуване API для створення компонентів, керування станом та обробки подій, що базується на принципах простоти та природності. Такий підхід робить його доступним як для початківців, так і для досвідчених розробників [22].

Крім того, vue відрізняється відмінною документацією та широким набором навчальних ресурсів, включаючи офіційний сайт, підручники, відеоуроки та активну спільноту розробників. Це надає всю необхідну підтримку для вивчення та початку роботи з ним. Завдяки своїй простоті та інтуїтивності, vue дозволяє новачкам у веброзробленні швидко освоїти основні концепції та розпочати створення простих програм лише за кілька годин навчання. Нарешті, фреймворк легко інтегрується з іншими технологіями та інструментами, такими як «Webpack», «Babel» та «ESLint», що забезпечує високу гнучкість та масштабованість розроблених застосунків.

Гнучкість та розширюваність фреймворку «Vue» сприяють позитивним аспектам його використання. Існує значна кількість плагінів та розширень, що постійно розширюють можливості цього інструменту. Серед них – різноманітні компоненти, директиви, фільтри, та інші засоби, які сприяють швидкому та ефективному розширенню функціональності програм. Vue був спроєктований з підвищеною увагою до гнучкості та можливості розширення, спрощуючи процес налаштування та використання. Інтеграція плагінів та розширень у проєкти відбувається безпроблемно, а також відкривається можливість створення власних компонентів та утиліт для вирішення конкретних завдань. Завдяки високій сумісності з іншими бібліотеками та JavaScript-фреймворками, vue може легко інтегруватися з різними технологіями для створення складних програм. Це робить можливим використання vue у поєднанні з бібліотеками управління станом, такими як «Vuex», або фреймворками для створення односторінкових програм, наприклад, «Nuxt.js». Модульна структура vue дозволяє використовувати лише необхідні складові фреймворку, що забезпечує легкість заміни або розширення функціональності за потреби. Такий підхід до

конструювання робить vue адаптивним до різноманітних вимог та умов використання.

Реактивність та компонентний підхід є основними принципами, що лежать в основі архітектури vue. Основою цієї архітектури є реактивна модель, що автоматично оновлює інтерфейс користувача у відповідь на зміни даних. Це досягається завдяки системі спостереження за даними, яка постійно відстежує їх стан та автоматично оновлює відповідні частини інтерфейсу. Саме такий підхід значною мірою спрощує процес розроблення та робить програми більш реактивними.

Цей фреймворк володіє компактним розміром, що робить його привабливим вибором для вебзастосунків, де критичними факторами є ефективне використання ресурсів та швидкість завантаження. Мінімізація обсягу даних, що передаються через мережу, та оптимізація часу завантаження сприяють загальній продуктивності програми.

Vue використовує ефективну віртуальну DOM для оновлення інтерфейсу користувача, що істотно зменшує навантаження на браузер і покращує час оновлення інтерфейсу. Завдяки цьому підходу vue може гарантувати мінімальні затримки при оновленні інтерфейсу користувача, роблячи програми, розроблені на його основі, чуттєвими та плавними, навіть при обробці великого обсягу даних або складних інтерфейсів користувача.

Всі ці характеристики роблять vue відмінним вибором для розроблення вебзастосунків, що вимагають легкості в використанні.

3.2 Порівняльний аналіз react, angular та vue

3.2.1 Визначення критеріїв для порівняльного аналізу

Для аналізу та порівняння фреймворків необхідно визначитися з основними критеріями, які будуть розглядатися за шкалою оцінок. Результати оцінок та будуть представлені у вигляді таблиці. Таблиця міститиме загальне уявлення про переваги наданих інструментів у вигляді оцінки. Далі на основі цих показників деталізуємо переваги та оцінку окремих критеріїв залежно від їх вагомих коефіцієнтів для дослідницького проєкту.

Для створення таблиці порівняння фреймворків були розглянуті наступні критерії:

1. Продуктивність – це ключовий критерій, що визначає, наскільки швидко та ефективно вебзастосунок реагує на дії користувача. Висока продуктивність означає менший час завантаження сторінок, швидкі відповіді на запити та плавну роботу інтерфейсу навіть при великих обсягах даних або складних операціях [23].

2. Зворотна сумісність – це властивість, що дозволяє забезпечити працездатність програми на різних версіях браузерів або інших програмних засобів. Це дозволяє користувачам продовжувати користуватися програмою навіть після оновлення програмного забезпечення [23].

3. Швидкість розроблення – це швидкість створення нового функціоналу або випуску оновлень програми. Більша швидкість розроблення дозволяє компаніям оперативно реагувати на зміни у ринкових умовах чи вимогах користувачів, а також швидше запускати нові продукти чи функції до ринку [23].

4. Відповідність стандартам – це забезпечення дотримання встановлених стандартів веброзроблення, таких як HTML, CSS та JavaScript, а також принципів, таких як доступність вихідного коду, відкриті стандарти та протоколи [23].

5. Розширюваність – це здатність адаптуватися до різних вимог проєкту та легко розширювати функціональність програми, додаючи нові компоненти або модулі без значних змін в основному коді. Це дозволяє розробникам швидко реагувати на зміни у вимогах користувачів чи бізнесу [23].

6. Зручність використання – це важливий критерій для розробників, що визначає, наскільки легко та швидко вони можуть освоїти фреймворк та почати використовувати його для створення програм. Простий та інтуїтивно зрозумілий інтерфейс, хороша документація та широка спільнота розробників можуть значно полегшити процес розроблення [23].

7. Екосистема інструментів – це набір бібліотек, інструментів, плагінів та ресурсів, доступних для використання з фреймворком. Добре розвинена екосистема забезпечує підтримку різних сценаріїв розроблення, покращує процес розроблення та забезпечує зростання спільноти розробників [23].

8. Безпека – це забезпечення захисту від уразливостей та забезпечення стабільної роботи програми у різних умовах використання. Безпека включає захист від атак і витоків даних, а також забезпечення відповідності стандартам безпеки [23].

9. Сумісність – це здатність програми працювати на різних платформах, пристроях та браузерах [23].

10. Доступність – це здатність програми в забезпеченні доступності для всіх користувачів, включаючи людей з обмеженими можливостями [23].

11. Підтримка мобільного розроблення – це можливість використання фреймворку для створення мобільних програм або адаптації вебзастосунка для роботи на мобільних пристроях. Це дозволяє компаніям забезпечити доступ до своїх продуктів для користувачів мобільних пристроїв та розширити аудиторію програми [23].

Проведений аналіз базується на думках експертів, які працювали з цими фреймворками, особистому досвіді та офіційних джерелах документації кожного фреймворку. За переліченими критеріями аналіз буде проведений для трьох основних фреймворків для веброзроблення: «React», «Vue» та «Angular».

Почнемо з продуктивності. У цьому аспекті react і vue виявилися переважними над angular. Зокрема, react демонструє найвищий рейтинг продуктивності – 82, на другому місці vue з 81, а на останньому – angular з 79 [24].

Щодо зворотної сумісності, react має найкращу зворотну сумісність, оскільки він надає пристосовуваність для інтеграції з наявними проєктами [20].

Angular має сувору політику зворотної сумісності, але не на такому ж рівні пристосованості, як у react [21, 24].

Vue може вимагати більше зусиль для інтеграції з наявними проєктами [22].

Швидкість розроблення є ще одним важливим критерієм порівняння. З огляду на швидкість розроблення, vue вирізняється своїм інтуїтивним підходом, що сприяє швидкому розробленню вебзастосунків. Водночас react, хоча і дозволяє швидше розробляти вебзастосунки, вимагає більше часу на вивчення, порівняно з vue. Angular, зі свого боку, може виявитися складнішим для освоєння та розробляння порівняно з vue та react.

У відношенні до відповідності стандартам, angular використовує TypeScript, що надає покращені інструменти та полегшує підтримку, щоб забезпечити кращий досвід розробника [21].

Навпаки, vue використовує JavaScript як стандарт розробки, але не має таких розширених інструментів, як TypeScript в angular. React використовує JSX, що є розширенням JavaScript, але може бути менш зручним для розробників, порівняно з TypeScript в angular [20, 22].

У плані розширюваності, react видається найбільш розширюваним завдяки можливості створювати повторно використовувані компоненти інтерфейсу користувача [20].

Vue також пропонує цю можливість, але може вимагати більше зусиль для інтеграції з вже наявними проєктами. Angular, всупереч багатофункціональності своїх бібліотек, може зустріти труднощі у процесі їх інтеграції порівняно з react.

Зручність використання також варіюється. Vue відзначається своїм простим API та документацією, що сприяє зручності використання. React також має простий API, хоча вимагає більше часу для вивчення порівняно з vue. Angular може виявитися складнішим для вивчення та використання порівняно з vue.

Далі екосистема інструментів, де angular має різноманітну спільноту з понад 1,7 мільйонами розробників, що сприяє активному обміну досвідом та розробкою різноманітних бібліотек і інструментів [21].

React також має активну спільноту, хоча менш різноманітну, ніж angular, але пропонує значну кількість корисних бібліотек і інструментів. Vue, хоча має також активну спільноту, але менш різноманітну, ніж angular та react.

У сфері безпеки, angular відрізняється вбудованими механізмами безпеки, що запобігають атакам, таким як XSS та CSRF, також він отримує підтримку від компанії «Google» та проходить суворі тестування [21].

Vue та react також мають вбудовані механізми безпеки, але вони менш розширені порівняно з angular.

У розгляді сумісності з різноманітними браузерами, react, angular і vue демонструють загальну сумісність з більшістю сучасних веббраузерів. Однак react відзначається вищим ступенем сумісності, порівняно з іншими фреймворками. Хоча vue, з іншого боку, не виявляє таких високих показників у цьому контексті [24].

Angular має вищий рівень доступності серед конкурентів. Проте, react також представлений як відкрите програмне забезпечення яке надає такі можливості. Vue своєю чергою має гірші показники за цим параметром [24].

Щодо можливостей розроблення мобільних застосунків, angular може використовуватися у поєднанні з фреймворком «Ionic» та іншими інструментами для створення мобільних застосунків й вони показують кращі показники [21, 24].

Водночас react співпрацює з «React Native», що дозволяє йому також розробляти мобільні застосунки [20].

А vue може використовуватися з додатковими інструментами, такими як «Weex» або «NativeScript», для створення мобільних застосунків [22].

3.2.2 Порівняння та розрахунок провідних фреймворків методом аналізу ієрархії

На основі представлених дослідницьких даних проведемо оцінку різних критеріїв, які включені до аналізу. Оцінки цих критеріїв будуть занесені до таблиці 3.1 згідно з встановленою шкалою, де оцінка 5 – відмінно, відповідає найкращому показнику серед досліджуваних фреймворків, оцінка 4 бали – добре, оцінка 3 бали – задовільно. А оцінка 2 відповідає рівню – незадовільно, відповідно 1 бал – відсутність даного критерію.

Таблиця 3.1 – Встановлення оцінок для аналізу фреймворків «React», «Vue» та «Angular»

Критерії		React	Vue	Angular
1	Продуктивність	5	4	3
2	Зворотна сумісність	5	3	4
3	Швидкість розроблення	4	5	3
4	Відповідність стандартам	3	4	5
5	Розширюваність	5	4	3
6	Зручність використання	4	5	3
7	Екосистема інструментів	4	3	5
8	Безпека	3	4	5
9	Сумісність	5	3	4
10	Доступність	3	4	5
11	Підтримка мобільного розроблення	4	3	5
Разом		45	42	45

Результати вказують, що react та angular мають однакову загальну оцінку у 45 балів, тоді як vue набирає 42 бали.

Для розв'язання багатокритеріальних завдань, широко застосовується метод аналізу ієрархії. Сутність цього методу полягає в тому, щоб розкласти складну проблему на менші компоненти для подальшого аналізу. Такий підхід дозволяє систематизувати велику кількість критеріїв та враховувати їх зв'язок.

Також такий підхід забезпечує більш об'єктивні результати та полегшує прийняття оптимальних рішень в умовах складних ситуацій.

Для аналізу та систематизації даних створимо таблицю у формі матриці (рисунок 3.1). Ця матриця буде служити інструментом для розрахунку.

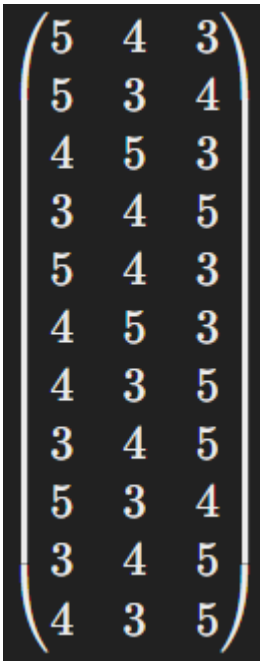

Рисунок 3.1 – Матричне подання таблиці оцінок для порівняння

Після побудови матриці визначення пріоритету кожного окремого об’єкту в ієрархії здійснюється шляхом оцінки відповідного нормованого головного власного вектора даної матриці. Точне визначення власного вектора матриці пріоритетів являє собою завдання, що стоїть перед певними труднощами. У світлі цього на практиці запропоновано застосовувати наступний підхід: спочатку елементи кожного стовпця матриці нормалізуються шляхом ділення на суму всіх елементів у цьому стовпці, а потім елементи кожного отриманого рядка сумуються, а ця сума ділиться на кількість елементів у рядку [25].

Для розроблення вебзастосунка визначимо вагомі критерії проєкту: продуктивність, швидкість розроблення, зручність використання та екосистема інструментів.

Використаємо наступну формулу для розрахунків:

$$d_j = \sum_{i=1}^{n_w} \left(\frac{a_{ij}}{\sum_{j=1}^{m} a_{ij}} * \frac{\sum_{i=1}^{n} \left(\frac{a_{ij}}{\sum_{j=1}^{m} a_{ij}} \right)}{n} \right), \quad (3.1)$$

де d_j – загальна оцінка для альтернативи j;

a_{ij} – оцінка критерію i для альтернативи j;

n – кількість критеріїв;

n_w – кількість вагомих критеріїв;

m – кількість альтернатив.

Отримані за розрахунком значення коефіцієнта кожного критерію для кожної альтернативи занесемо до таблиці 3.2. А також занесемо у таблицю 3.3 виважений результівний бал для порівняних фреймворків. Найкращі результати позначено зеленим кольором, а найгірші – червоним.

Для розробки вебзастосунка визначимо важливі проєкту критерії: продуктивність, швидкість розроблення, зручність використання та екосистема інструментів.

Проведемо розрахунок для react:

$$d_{react} = \left(\frac{5}{12} * \frac{3.75}{11}\right) + \left(\frac{4}{12} * \frac{3.75}{11}\right) + \left(\frac{4}{12} * \frac{3.75}{11}\right) + \left(\frac{4}{12} * \frac{3.75}{11}\right) = 0.48.$$

Проведемо розрахунок для vue:

$$d_{vue} = \left(\frac{4}{12} * \frac{3.5}{11}\right) + \left(\frac{5}{12} * \frac{3.5}{11}\right) + \left(\frac{5}{12} * \frac{3.5}{11}\right) + \left(\frac{3}{12} * \frac{3.5}{11}\right) = 0.45.$$

Проведемо розрахунок для angular:

$$d_{angular} = \left(\frac{3}{12} * \frac{3.75}{11}\right) + \left(\frac{3}{12} * \frac{3.75}{11}\right) + \left(\frac{3}{12} * \frac{3.75}{11}\right) + \left(\frac{5}{12} * \frac{3.75}{11}\right) = 0.39.$$

Таблиця 3.2 – Коефіцієнти критеріїв для кожної альтернативи

Критерії	React	Vue	Angular
Продуктивність	0,42	0,33	0,25
Зворотна сумісність	0,42	0,25	0,33
Швидкість розроблення	0,33	0,42	0,25
Відповідність стандартам	0,25	0,33	0,42
Розширюваність	0,42	0,33	0,25
Зручність використання	0,33	0,42	0,25
Екосистема інструментів	0,33	0,25	0,42
Безпека	0,25	0,33	0,42
Сумісність	0,42	0,25	0,33
Доступність	0,25	0,33	0,42
Підтримка мобільного розроблення	0,33	0,25	0,42

Таблиця 3.3 – Результат порівняння фреймворків з урахуванням вагових критеріїв

Фреймворки	Результат
React	0.48
Vue	0.45
Angular	0.39

Після аналізу з урахуванням вагових критеріїв, можна зробити наступні висновки. React виявився лідером з найвищою зваженою оцінкою, яка становить 0.48, що свідчить про його перевагу порівняно з іншими інструментами. Vue також показав високий результат з оцінкою 0.45, хоча він трохи поступається react. З іншого боку, angular має найнижчу зважену оцінку, яка становить 0.39, що вказує на його нижчу оцінку порівняно з іншими інструментами, враховуючи коефіцієнт вагомості. Таким чином, проведений аналіз дозволяє зробити висновок про перевагу кожного інструменту, враховуючи важливість встановлених критеріїв. Отже, вибір зроблено на користь фреймворку «React».

Хоча angular оцінюється нижче за react та vue, він все ще є ефективним фреймворком, який пропонує відносно високу продуктивність, відповідність

стандартам, широкий вибір інструментів, безпеку, а також підтримку мобільного розроблення. Це робить angular також цінним вибором для різних сценаріїв розроблення.

У розділі 1.2.2 було згадано про взаємозв'язок між jest-тестуванням та застосунками на react. Це пов'язано з тим, що jest, який був створений компанією «Facebook», спочатку був призначений для тестування застосунків на react. Він має вбудовану підтримку тестування компонентів react, що робить його ідеальним вибором для тестування застосунків на react.

Щодо тестування продуктивності, оберемо інструмент «Lighthouse», про який йшлося в розділі 1.2.3. Інструмент тестування «Lighthouse», хоча і не прямо пов'язаний з react і jest, надає важливу інформацію про продуктивність, доступність, завантаження контенту та інші аспекти вебзастосунків, включаючи ті, які розробляються з використанням react і тестуються за допомогою jest.

Lighthouse проводить аналіз продуктивності та доступності вебпрограми, що є особливо важливим при розробленні застосунків reac». Він інтегрований з інструментами розроблення, такими як інструмент «Google Chrome DevTools», що полегшує процес тестування та аналізу. Крім виявлення проблем, lighthouse пропонує рекомендації щодо покращення продуктивності та доступності вебзастосунка.

Отже, використання інструменту «Lighthouse» разом з фреймворком «Jest» для тестування застосунків react дозволяє отримати повне уявлення про продуктивність, доступність та інші важливі аспекти програми.

4 РОЗРОБЛЕННЯ ІНТЕРФЕЙСУ ВЕБЗАСТОСУНКА

4.1 Етапи розроблення вебзастосунка

Розроблення вебзастосунка відбувається в рамках каскадної моделі життєвого циклу програмного забезпечення, яка була розглянута в розділі 2.3. Цей процес можна розглядати як послідовність етапів (рис. 4.1).

Рисунок 4.1 – Етапи розроблення вебзастосунка

На початковому етапі формуються вимоги та завдання проєкту. Ключовим є уточнення та документування вимог до функціональності та характеристик системи.

Наступний етап включає проєктування структури та організації компонентів у вебзастосунку. Вирішуються питання архітектурних рішень та розподілу функціональності між компонентами системи.

Потім відбувається реалізація компонентів програми відповідно до концепції компонентної архітектури. Важливою задачею є розроблення та інтеграція окремих модулів програми.

Останній етап передбачає проведення функціонального і продуктивного тестування. Це спрямовано на визначення відповідності розробленого програмного забезпечення вимогам та його стабільності в умовах реального використання.

4.2. Формування вимог для розроблення інтерфейсу вебзастосунка

При розробленні вебзастосунка важливо визначити його основні функціональні вимоги. Однією з головних може бути, наприклад, можливість швидкого та ефективного пошуку даних про країни, їх регіони та столиці. Користувачам може знадобитися доступ до деталізованої інформації про країни з можливістю фільтрації за різними критеріями, такими як регіон або назва столиці.

Враховуючи, що для проєкту важлива швидкість розроблення, буде використовуватись мова програмування «JavaScript» замість «TypeScript». Переваги у швидкості розробляння якої зазначались у розділі 1.3.

Враховуючи обмеження на вибір постачальника API для отримання даних, важливо забезпечити можливість пошуку інформації виключно англійською мовою. Це допоможе забезпечити єдність інтерфейсу та зручність використання для користувачів з різних країн та мовних середовищ.

Вебзастосунок буде використовувати безплатний сервіс «REST Countries» для отримання API. Це сервіс, який надає вільний доступ до даних про країни з усього світу через простий API.

Крім того, функціонал пошуку може бути розширений за допомогою можливості проведення пошуку за окремими літерами, складами або неповною назвою регіону, столиці або країни. Це дозволить користувачам здійснювати більш гнучкий та точний пошук, особливо коли вони мають обмежену інформацію про об’єкт пошуку або потребують швидкого доступу до результатів.

Під час розроблення вебзастосунка слід також враховувати можливості взаємодії з користувачем. Для забезпечення зручності та ефективності використання застосунку можна розглянути включення додаткових функцій для подальшого розвитку, таких як сортування результатів, візуалізація даних, а також можливість збереження результатів пошуку для подальшого використання або аналізу.

Також, при розробці інтерфейсу користувача варто приділяти увагу його зручності та інтуїтивності. Важливо забезпечити простий та зрозумілий інтерфейс, що дозволить користувачам легко зорієнтуватися у функціоналі застосунку та ефективно використовувати його для своїх потреб.

Такий підхід до розроблення вебзастосунка дозволить створити продукт, який відповідає потребам користувачів та забезпечить їм зручний та ефективний інструмент для проведення досліджень та аналізу даних про країни.

4.3 Структура та організація компонентів у вебзастосунках

При створенні вебзастосунка ключовими є ефективна структура та організація компонентів. Компоненти є фундаментальними елементами вебзастосунків, а їхнє правильне розташування впливає на чистоту коду, його розширюваність та підтримку. Добре організовані компоненти сприяють гармонійній роботі в команді, полегшують додавання нового функціоналу та розв'язування майбутніх проблем.

Каталог компонентів є важливим елементом організації компонентів у вебпроєкті. Це спеціальне місце, де знаходяться всі компоненти, що використовуються в застосунку. Ефективне структурування цього каталогу відіграє важливу роль у підтримці порядку та зрозумілості в проєкті.

Організація компонентів у вигляді окремих каталогів є важливою для збереження систематизованості в рамках проєкту. Кожен компонент має свою власну теку або файл, що сприяє не лише легкості їх виявлення, але й полегшує їх локалізацію та управління.

Крім того, консолідація всіх компонентів у єдиному каталозі забезпечує зручний доступ до них, сприяючи оперативному використанню та аналізу. На додаток, каталог компонентів може легко розширюватися шляхом додавання нових тек чи файлів, що сприяє масштабуванню проєкту та інтеграції нового функціоналу.

Створення підкаталогів за функціональністю є важливим для ефективної організації компонентів у вебпроєкті. Цей підхід сприяє логічному групуванню компонентів згідно з їх функціональним призначенням, що полегшує розуміння структури проєкту та спрощує навігацію.

Підкаталоги за функціональністю дозволяють об’єднувати компоненти, які виконують подібні функції або мають спільну функціональність, у зручних групах. Це сприяє утворенню логічної структури проєкту, де різні компоненти розташовані відповідно до їхніх функціональних аспектів. Кожен підкаталог може представляти певний аспект функціональності застосунку, що спрощує розуміння та навігацію для розробників. Групування компонентів за типом у

відповідних підкаталогах допомагає уніфікувати код та полегшує співпрацю в команді.

У практичному плані, для успішної роботи з каталогом компонентів важливо встановлювати стандарти іменування тек та файлів, групувати схожі компоненти разом та використовувати підкаталоги для подальшого поділу компонентів на більш дрібні групи. Необхідно також враховувати потреби конкретного проєкту та його архітектуру при організації каталогів та підкаталогів.

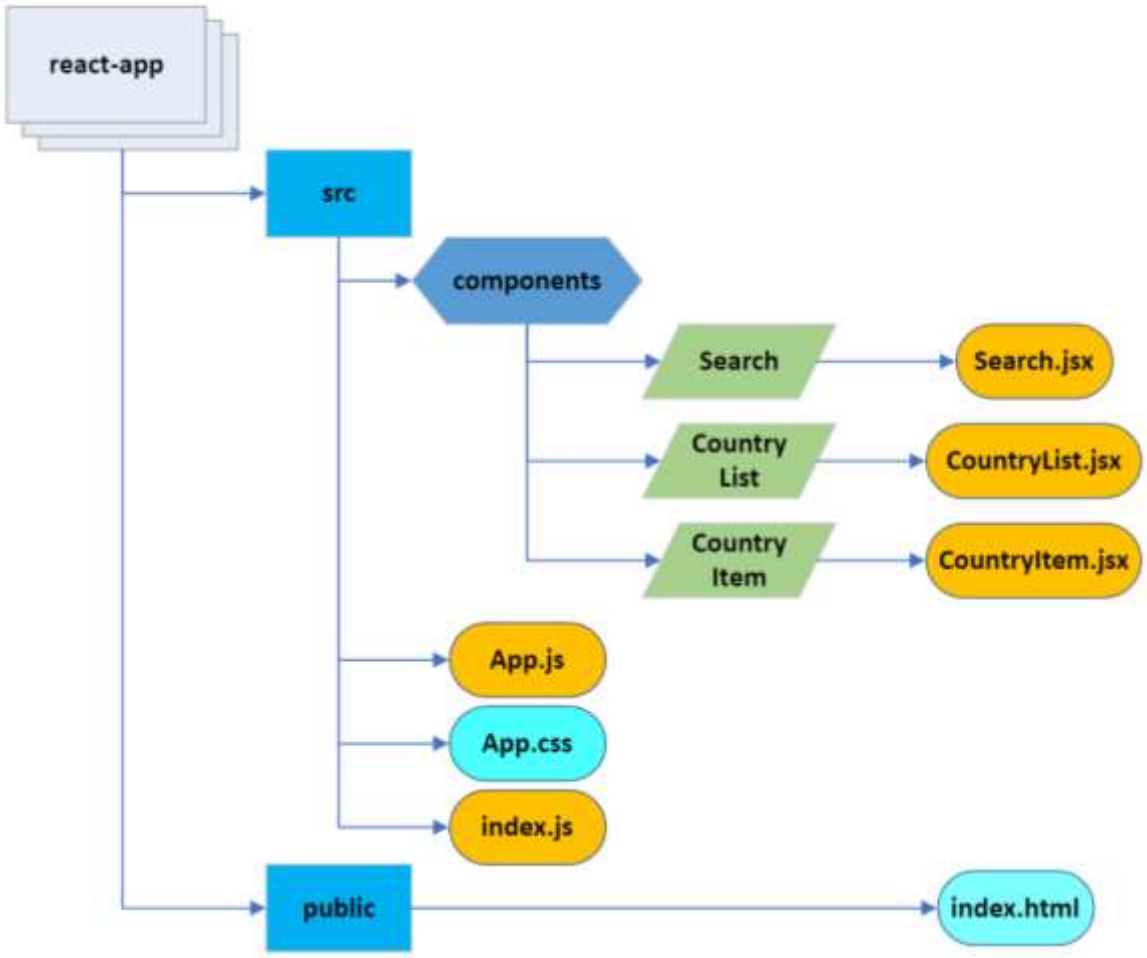

Рисунок 4.2 – Структура застосунку для пошуку інформації про країни

При створенні вебзастосунка для пошуку інформації про країни, розробимо ефективну структуру проєкту (рис. 4.2). Основними елементами структури є директорія «src», яка містить весь вихідний код програми, та директорія «components», де зберігаються всі компоненти програми.

У директорії «components» можна виділити піддиректорії, такі як «Search», «CountryList» та «CountryItem». Вони містять компоненти, що відповідають за відображення пошуку, списку країн та елементу списку країн відповідно.

В корені проєкту розміщуються файли «App.js», «App.css», «index.js» та «index.html». Файл «App.js» є основним компонентом програми, файл «App.css» містить стилі для головного компонента, файл «index.js» є точкою входу програми, де відбувається рендеринг основного компонента «App» в DOM, а файл «index.html» є HTML файлом програми.

Кожен компонент має власну теку, всередині якої знаходяться файли JSX (або JavaScript) і CSS (якщо використовується). Це дозволяє забезпечити чистоту та структуру проєкту, особливо у разі великої кількості компонентів.

4.4 Розроблення компонентів для інтерфейсу та взаємодії з серверною частиною через API

4.4.1 Компонент «App»

Компонент «App» є центральною частиною застосунку, відповідальною за відображення та координацію основного інтерфейсу користувача (рис. А.1). Він містить ряд підкомпонентів, таких як «Search» і «CountryList», і використовує стан для управління взаємодією користувача та відображенням даних.

Компонент «App» встановлює початкові значення стану, включаючи порожній список країн, порожнє значення для рядка пошуку, а також значення фільтра за замовчуванням – «name». Крім того, він встановлює початковий текст для підказки в полі пошуку.

Цей компонент використовує ефект «useEffect», щоб відстежувати зміни у введенні користувача та обраному фільтрі, і автоматично оновлювати відображення країн відповідно до змін. Також, він має функції обробників подій для взаємодії з компонентом «Search», які відповідають за оновлення значень стану при зміні введення користувача та обраного фільтра.

Заголовок та інтерфейс компонента «App» є ключовими елементами користувацького досвіду, і вони відображаються на початку сторінки. Компонент «CountryList» відображає список країн у форматі карточок, які можна прокручувати, що дозволяє користувачеві легко переглядати інформацію про кожну країну.

4.4.2 Компонент «Search»

Компонент «Search» відповідає за створення інтерфейсу користувача для пошуку країн з використанням різних фільтрів (рис. А.2). Його функціональність базується на введенні користувачем запиту в текстове поле та виборі одного з варіантів фільтрації зі списку що випадає.

Компонент має форму, яка містить текстове поле для введення запиту та список параметрів, де користувач може вибрати, за яким критерієм шукати країни. Підказка в полі введення тексту динамічно змінюється відповідно до обраного фільтра, надаючи користувачеві відповідну інформацію про те, за яким параметром проводиться пошук.

Компонент реалізований з використанням тегів HTML «<form>», «<input>» та «<select>», що забезпечує стандартний інтерфейс для введення тексту та вибору опцій. Поле введення тексту «<input>» призначене для введення пошукового запиту, а список «<select>» дозволяє вибрати тип пошуку: за ім'ям країни, регіоном або столицею. Параметри компонента, такі як функції зміни значень та плейсхолдера, передаються через властивості.

При зміні значення в текстовому полі або обранні іншого фільтра компонент «Search» викликає відповідні функції зворотного виклику, які передають інформацію про нові параметри пошуку батьківському компоненту. Це дозволяє компоненту «App» оновлювати відображення країн відповідно до нових критеріїв пошуку.

Отже, компонент «Search» забезпечує зручний та ефективний інтерфейс для користувача, який дозволяє шукати країни з використанням різних фільтрів і зручною системою введення запитів.

4.4.3 Компонент «CountryList»

Компонент «CountryList» є частиною інтерфейсу застосунку, призначеною для відображення списку країн (рис. А.3). Його функція полягає у створенні візуального представлення даних про країни у вигляді карточок, які відображаються на сторінці.

Призначення компонента полягає у тому, щоб створити контейнер для відображення інформації про кожну країну. Він отримує список країн у вигляді масиву ітерується через цей масив, створюючи карточку для кожної країни. Карточка країни містить інформацію, таку як назва країни, прапор, населення, регіон та столиця.

Кожна карточка країни відображається у відповідному стилі, що робить їх привабливими та легкими для сприйняття користувачем. Кожна карточка має зображення прапору країни, а також текстову інформацію про назву країни, її населення, регіон та столицю.

Крім того, компонент «CountryList» має механізм перевірки вхідних даних: перевіряється, чи є вхідний список країн масивом. Якщо він не є масивом, компонент повертає «null», що допомагає уникнути помилок під час відображення даних.

Узагальнюючи, компонент «CountryList» відповідає за створення і відображення карточок для кожної країни у вигляді списку, що сприяє зручному та естетичному перегляду інформації про країни.

4.4.4 Компонент «CountryItem»

Компонент «CountryItem» відповідає за відображення окремої карточки країни у списку країн (рис. А.4). Він приймає на вхід об'єкт «country», який містить інформацію про країну, і рендерить цю інформацію у вигляді карточки з відповідною графічною символікою та текстовими даними.

Назва країни відображається у вигляді заголовка карточки з відповідною назвою. Під назвою країни розміщені текстові блоки з іншими важливими характеристиками. Кожен з цих текстових блоків має відповідний заголовок, що підкреслює його значення.

Зображення прапора відображається зверху від текстових даних інформації про країну. Текстові дані, які включають назву, кількість населення, регіон і столицю, відображаються у відповідних блоках, розташованих знизу від зображення прапора.

Компонент «CountryItem» виконує роль забезпечення коректного та естетичного відображення даних про країну для зручного користування користувачем. Він є складовою частиною інтерфейсу користувача, що спрощує процес огляду та аналізу інформації про країни у списку.

4.4.5 Функція для взаємодії з серверною частиною через API

Функція «fetchData» є елементом застосунку, відповідальним за отримання даних про країни з віддаленого джерела за допомогою HTTP-запитів (рис. А.5). Після отримання відповіді від сервера, ця функція обробляє дані та оновлює стан застосунку згідно з отриманим даними.

При виклику цієї функції передаються три параметри: «searchValue» (значення для пошуку), «filter» (тип фільтрації) і «setCountries» (функція для оновлення списку країн).

На основі параметрів «searchValue» і «filter», функція спочатку формує URL-адресу запиту до API, враховуючи обраний тип фільтрації та введений користувачем пошуковий запит. Потім вона виконує асинхронний HTTP-запит за допомогою функції «fetch», передаючи згенеровану URL-адресу. Після отримання відповіді від сервера в форматі JSON, функція асинхронно розпаковує її за допомогою функції «response.json()» передає отримані дані функції «setCountries». Після цього вона оновлює стан застосунку, передаючи отримані дані у вигляді списку країн.

У випадку помилки під час виконання запиту, функція «fetchData» перехоплює виняток і виводить повідомлення про помилку у консоль. Це дозволяє налагоджувати та виправляти проблеми з отриманням даних, забезпечуючи стабільну та надійну роботу застосунку. Функція забезпечує надійну та ефективну інтеграцію з віддаленим джерелом даних, що є важливим аспектом розроблення вебзастосунків.

Отже, функція «fetchData» відповідає за виконання запиту до API, отримання та обробку даних про країни на основі введеного користувачем запиту та обраного типу фільтрації.

4.5 Тестування інтерфейсу вебзастосунка

4.5.1 Функціональні тести інтерфейсу вебзастосунка

У процесі валідації тестового набору для застосунку проводиться аналіз різноманітних аспектів його функціональності (рис. А.6 – А.7).

Перший тест спрямований на визначення наявності головного заголовка застосунку «Where in the world?» на відповідній сторінці. Це дозволяє переконатися, що користувач вірно ідентифікує місце призначення програми.

Другий тест перевіряє можливість здійснення пошуку країни за її назвою шляхом введення відповідного значення у відповідне поле. Очікується, що введення здійснюється ефективно та результати пошуку відображаються правильно.

Третій тест перевіряє наявність поля пошуку з відповідним плейсхолдером та вибраним фільтром, забезпечуючи коректну функціональність цієї опції.

Четвертий тест аналізує правильність відображення інформації про країну, включаючи її назву, населення, регіон та столицю. Це дозволяє перевірити відповідність даних, які відображаються для користувача.

П'ятий тест перевіряє відображення списку країн з правильними даними для кожної країни у списку, забезпечуючи повноту та точність інформації.

Шостий тест аналізує коректність виклику правильного URL та встановлення даних у стані компонента під час виконання запиту на сервер, забезпечуючи правильну взаємодію з сервером.

Сьомий тест перевіряє правильну обробку зміни значення пошукового запиту в компоненті пошуку, що забезпечує коректне функціонування цієї функції.

Восьмий тест перевіряє коректну обробку зміни вибраного фільтру в компоненті пошуку, забезпечуючи правильну фільтрацію даних.

Всі ці тести використовують можливості jest для тестування react, щоб гарантувати якість та стабільність функціональності застосунку.

На рисунку 4.3 спостерігаються результати успішного функціонального тестування на jest. Ці результати свідчать про ефективність виконання тестів, розроблених з використанням фреймворку «Jest». Успішне проходження тестів без помилок підтверджує правильність реалізації функціональності програмного забезпечення. Зазначений результат також вказує на те, що розроблений програмний продукт відповідає встановленим вимогам.

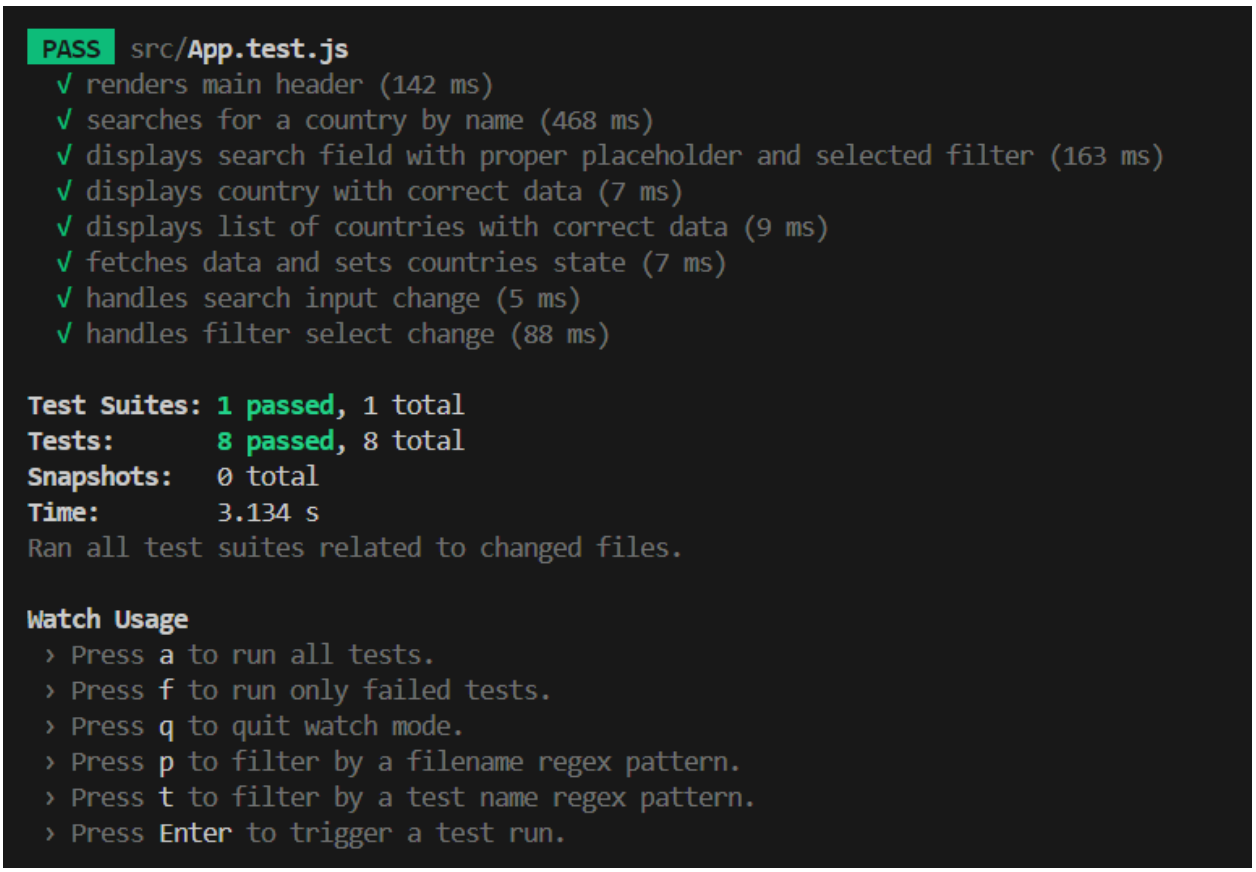

Рисунок 4.3 – Інтерфейс консолі «Результат успішного функціонального Jest-тестування»

4.5.2 Тести продуктивності інтерфейсу вебзастосунка

Проведений тест продуктивності інтерфейсу вебзастосунка зображено рисунку 4.4.

Загальна оцінка продуктивності вебзастосунка становить 81 бал, вона формується на основі п'яти ключових метрик: FCP (First Contentful Paint), SI (Speed Index), LCP (Largest Contentful Paint), TBT (Total Blocking Time) та CLS (Cumulative Layout Shift). Кожна з цих метрик має певну вагу в загальній оцінці, що визначає їхній вплив на кінцевий результат.

Метрика FCP вимірює час, за який перший елемент контенту стає видимим для користувача. Швидкий FCP покращує користувацький досвід, створюючи враження швидкого завантаження сторінки. Вага цієї метрики в загальній оцінці становить 10%.

Метрика SI показує, наскільки швидко контент сторінки стає видимим. Низький SI свідчить про швидке завантаження контенту, що позитивно впливає на загальну оцінку. Вага цієї метрики також складає 10%.

Метрика LCP вимірює час, за який найбільший елемент контенту стає видимим. Це критичний показник, оскільки великі елементи зазвичай є важливими частинами контенту сторінки. Швидкий LCP значно покращує загальну оцінку. Вага LCP становить 25%, що відображає її важливість.

Метрика TBT показує загальний час, протягом якого основний потік був заблокований під час завантаження сторінки. Менший TBT вказує на кращу продуктивність і здатність сторінки швидко реагувати на введення користувача. Це найбільш вагома метрика в загальній оцінці. Це найбільш вагома метрика в загальній оцінці з вагою 30%.

Метрика CLS вимірює стабільність макета сторінки, визначаючи сумарний обсяг зміщень під час завантаження. Низький CLS свідчить про стабільний макет, що позитивно впливає на користувацький досвід і підвищує загальну оцінку. Вага CLS становить 25%.

Загальна оцінка вебзастосунка визначається шляхом зважених середніх значень цих п'яти метрик. Метрики з більшою вагою (LCP, TBT, CLS) мають більший вплив на загальну оцінку, ніж метрики з меншою вагою (FCP, SI). Таким чином, для досягнення високої загальної оцінки необхідно забезпечити високі показники продуктивності за всіма п'ятьма метриками, приділяючи особливу увагу тим, що мають більшу вагу в загальній оцінці.

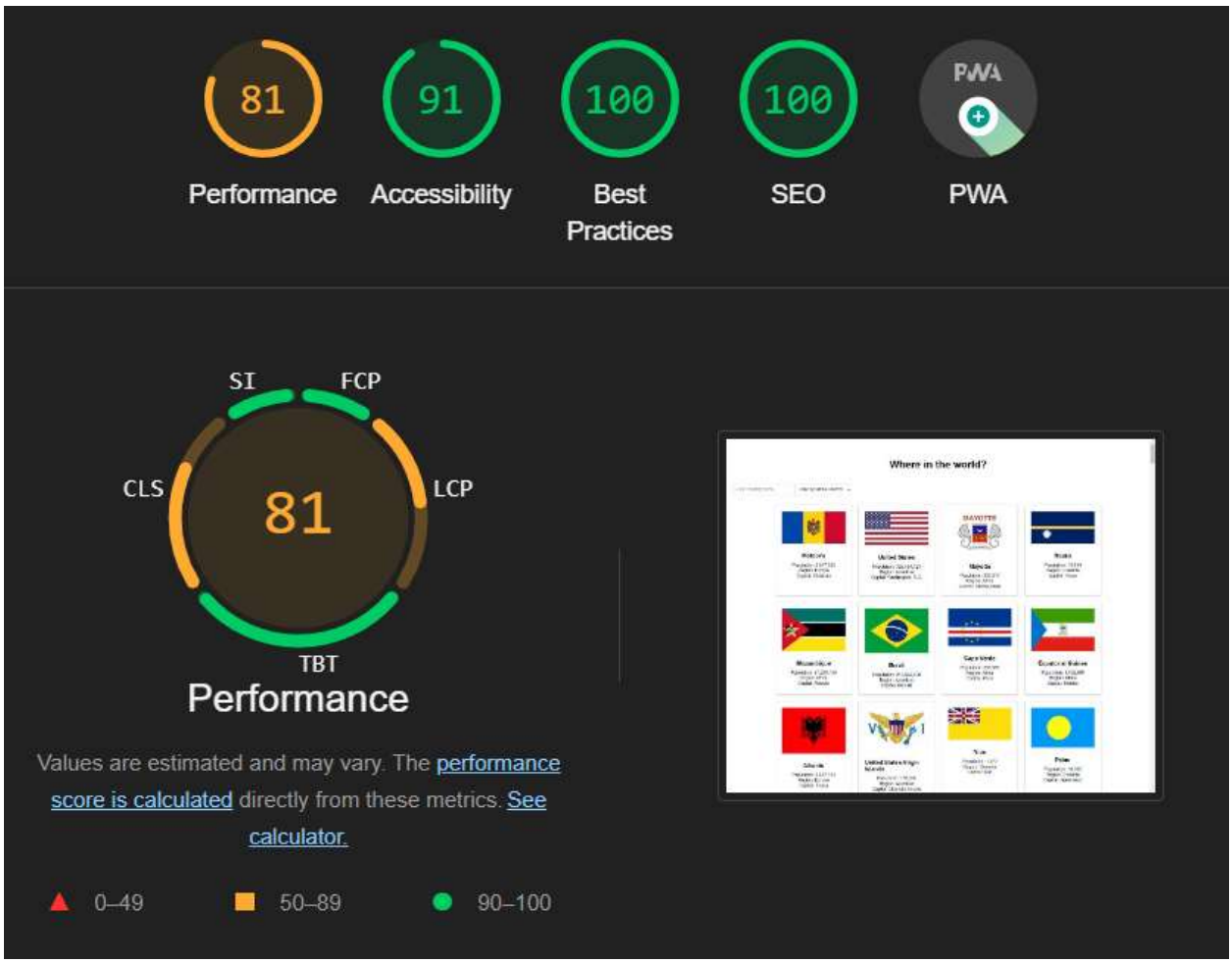

Рисунок 4.4 – Інтерфейс інструменту Lighthouse «Результат тесту продуктивності інтерфейсу вебзастосунка»

Оцінка доступності складає 91 бал, що вказує на високий рівень доступності програми для користувачів з обмеженими можливостями. Все ж таки, можливостей для поліпшення все ще залишається.

В області кращих практик застосунок отримав максимальні 100 балів, що свідчить про його відповідність до сучасних стандартів веброзроблення.

Оцінка оптимізації для пошукових систем також становить 100 балів, що вказує на те, що програма оптимізована для пошукових систем і має гарну видимість у них.

Застосунок також отримав позначку відповідно до критеріїв прогресивного вебзастосунка (PWA), що підтверджує його надійність, швидкодію та якісну взаємодію з користувачем.

Аналіз поданих даних підтверджує, що react-застосунок виявляє помірні показники продуктивності, високий рівень доступності та відповідність

сучасним стандартам веброзроблення. Його привабливість для користувачів доповнюється оптимізацією для пошукових систем та підтримкою PWA. Однак, потрібна увага до областей, де є потенціал для покращення, включаючи, але не обмежуючись, аспектами продуктивності та доступності.

4.6 Результат розроблення інтерфейсу вебзастосунка

Вебзастосунок демонструє значну функціональність та корисність для користувачів, надаючи можливість швидкого та зручного доступу до інформації про різні країни світу. Його простий і зрозумілий інтерфейс дозволяє здійснювати пошук за різними критеріями, такими як назва країни, регіон чи столиця. Результати розроблення вебінтерфейсу можна побачити на рисунку 4.5.

Крім того, застосунок успішно пройшов тестування, оскільки було розроблено набір тестів для перевірки різних аспектів його функціональності. Це забезпечує впевненість у тому, що він працює правильно і без помилок у різних сценаріях використання.

Загалом, цей вебзастосунок є ефективним інструментом для отримання інформації про країни світу і підтверджує ефективність обраних інструментів для його реалізації.

Він може бути корисним для широкого кола користувачів, які цікавляться географією та хочуть дізнатися більше про різні країни. Цей проєкт також може слугувати прикладом для порівняння в майбутніх дослідженнях у сфері розроблення інтерфейсів вебзастосунків.

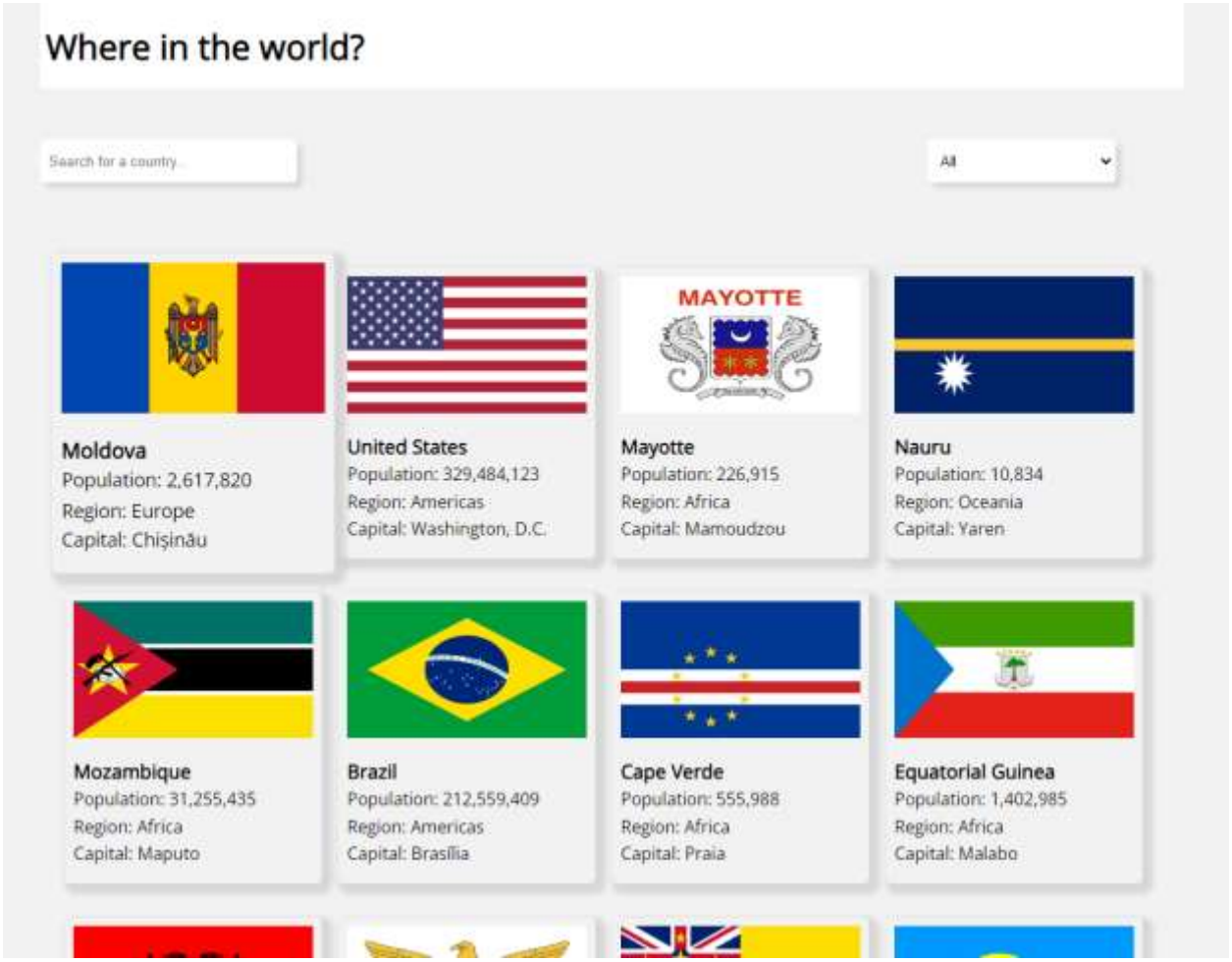

Рисунок 4.5 – Інтерфейс розробленого вебзастосунка

ВИСНОВКИ

На основі проведеного дослідження було встановлено, що вибір інструментів для розробки вебінтерфейсів є критичним етапом у створенні ефективних та високоякісних вебзастосунків. Проведений аналіз сучасних JavaScript-фреймворків, таких як react, angular та vue, дозволив виявити їхні ключові особливості, переваги та обмеження. Кожен з розглянутих фреймворків має унікальні характеристики, що робить їх придатними для різних типів проєктів.

У першому розділі було проведено детальний огляд сучасних JavaScript-фреймворків та бібліотек. Розглянуто інструменти для тестування вебзастосунків, включаючи різні види тестування. Проаналізовано мови програмування для розробки інтерфейсів, зокрема JavaScript та TypeScript, що використовуються у сучасній веброзробці.

Другий розділ присвячений аналізу основних архітектурних підходів до проєктування вебінтерфейсів, таких як монолітний, мікросервісний та компонентний підходи. Розглянуто переваги та недоліки кожного з них, а також етапи життєвого циклу розроблення програмного забезпечення, що дозволяє більш структуровано підходити до процесу створення вебзастосунків.

У третьому розділі проведено порівняльний аналіз основних фреймворків для розроблення вебінтерфейсів: react, angular та vue. Визначено критерії для порівняння та за допомогою методу аналізу ієрархії здійснено оцінку цих фреймворків. Це дозволило визначити найбільш відповідний фреймворк для подальшої розробки вебінтерфейсу користувача.

Четвертий розділ присвячений безпосередньо процесу розроблення інтерфейсу вебзастосунка. Розглянуто етапи розроблення, формування вимог до інтерфейсу, структура та організація компонентів у вебзастосунку. Описано розробку основних компонентів інтерфейсу та взаємодію з серверною частиною через API. Розглянуто процес тестування інтерфейсу, включаючи функціональні тести та тести продуктивності. У результаті було створено ефективний та функціональний інтерфейс вебзастосунка, який відповідає сучасним вимогам.

React виявився лідером за багатьма критеріями, включаючи продуктивність, зворотну сумісність та екосистему інструментів, що робить його оптимальним вибором для великих корпоративних проєктів. Його компонентний підхід

дозволяє створювати складні та гнучкі інтерфейси, забезпечуючи високу продуктивність та масштабованість.

Angular, хоча і має складнішу архітектуру, демонструє високий рівень структурованості та підтримки великих застосунків. Його використання виправдане у випадках, коли потрібна чітка архітектура та дотримання строгих принципів програмування, що сприяє підтримці та розширюваності великих проєктів.

Vue відзначається своєю простотою та інтуїтивністю, що робить його ідеальним для швидкого старту проєктів та розробки менш складних застосунків. Його гнучкість та легкість у навчанні забезпечують швидку адаптацію розробників та зручність використання.

У результаті проведеного дослідження був розроблений інтерфейс вебзастосунка, який демонструє практичне застосування теоретичних знань та обраних інструментів. Цей інтерфейс слугує прикладом оптимального поєднання технологій для досягнення високої продуктивності та зручності використання. Таким чином, дана робота спрямована на надання комплексного огляду та рекомендацій щодо вибору та використання сучасних інструментів для створення високоякісних вебзастосунків.

Розроблений застосунок демонструє високу функціональність і зручність використання, що забезпечує користувачам швидкий доступ до необхідних даних. Простий та інтуїтивно зрозумілий інтерфейс дозволяє здійснювати пошук за різними критеріями, такими як назва країни, регіон чи столиця, що робить його корисним інструментом для широкого кола користувачів.

Важливим досягненням є успішне тестування застосунку, що підтвердило його правильну роботу в різних сценаріях використання. Це було досягнуто завдяки розробці набору тестів, що охоплюють різні аспекти функціональності. Такий підхід забезпечив впевненість у надійності та стабільності роботи застосунку, що є критично важливим для його подальшого використання.

Дослідження також показало ефективність обраних інструментів та технологій для реалізації вебзастосунка, підтвердивши правильність обраного підходу. Використання сучасних технологій забезпечило високу продуктивність та відповідність застосунку сучасним стандартам веброзробки, включаючи оптимізацію для пошукових систем та підтримку прогресивних вебзастосунків (PWA).

Перспективи подальших досліджень включають оптимізацію продуктивності застосунку, розширення функціональності та інтеграцію з іншими системами. Дослідження в цьому напрямку можуть також включати покращення користувацького досвіду та впровадження нових технологій для забезпечення більшої інтерактивності та зручності використання.

Таким чином, проведене дослідження зробило внесок у розвиток веброзробки, надавши корисні інструменти та методики для створення ефективних та надійних вебзастосунків. Результати цієї роботи можуть бути використані для подальших досліджень і розробок у цій галузі, сприяючи розвитку сучасних вебтехнологій та покращенню якості вебсервісів.

ПЕРЕЛІК ПОСИЛАНЬ

1. State of JS 2022. 05.02.2024 [Електронний документ] - Режим доступу: https://2022.stateofjs.com/en-US
2. Порівняння Angular, React та Vue за статистикою завантажень NPM. 08.02.2024 [Електронний документ] - Режим доступу: https://npmtrends.com/angular-vs-react-vs-vue
3. Types of Web Application Testing. 09.02.2024 [Електронний документ] - Режим доступу: https://katalon.com/resources-center/blog/types-of-web-testing
4. 20 Best Software Testing Tools for QA. 09.02.2024 [Електронний документ] - Режим доступу: https://www.lambdatest.com/blog/software-testing-tools/
5. What Is End-to-End Testing? (With How-To and Example). 10.02.2024 [Електронний документ] - Режим доступу: https://www.indeed.com/career-advice/career-development/end-to-end-testing
6. End-to-End (E2E) vs Integration Testing: Key Differences. [Електронний документ] - Режим доступу: https://www.lambdatest.com/blog/end-to-end-testing-vs-integration-testing/
7. What is Integration Testing. [Електронний документ] - Режим доступу: https://www.hypertest.co/integration-testing/what-is-integration-testing
8. Офіційний сайт Jest. [Електронний документ] - Режим доступу: https://jestjs.io/
9. Snapshot Testing. [Електронний документ] - Режим доступу: https://jestjs.io/docs/snapshot-testing
10. Офіційний сайт Mocha. [Електронний документ] - Режим доступу: https://mochajs.org/
11. Офіційний сайт Storybook. [Електронний документ] - Режим доступу: https://storybook.js.org/?ref=border.ghost.io
12. A detailed guide on JavaScript Web Development. [Електронний документ] - Режим доступу: https://www.browserstack.com/guide/javascript-web-development
13. The Rise of TypeScript: A Strongly Typed JavaScript Superset. [Електронний документ] - Режим доступу: https://dev.to/saloman_james/the-rise-of-typescript-a-strongly-typed-javascript-superset-5b27

14. Monolithic Architecture. [Електронний документ] - Режим доступу: https://www.geeksforgeeks.org/monolithic-architecture/

15. Monolithic vs Microservices Architecture. [Електронний документ] - Режим доступу: https://www.geeksforgeeks.org/monolithic-vs-microservices-architecture/

16. Мікросервісний підхід у веброзробці: micro frontends. [Електронний документ] - Режим доступу: https://dou.ua/lenta/articles/micro-frontend/

17. React's Component-Based Architecture: A Case Study. [Електронний документ] - Режим доступу: https://appmaster.io/blog/react-component-based-architecture

18. The Software Development Life Cycle (SDLC). [Електронний документ] - Режим доступу: https://theproductmanager.com/topics/software-development-life-cycle/

19. A Complete Guide to Software Development Models. [Електронний документ] - Режим доступу: https://binariks.com/blog/software-development-models/

20. Офіційний сайт React. [Електронний документ] - Режим доступу: https://react.dev/

21. Офіційний сайт Angular. [Електронний документ] - Режим доступу: https://angular.io/

22. Vue. [Електронний документ] - Режим доступу: https://vuejs.org/

23. Web Development. [Електронний документ] - Режим доступу: https://www.geeksforgeeks.org/web-development/

24. Angular vs. React vs. Vue.js: Comparing performance. [Електронний документ] - Режим доступу: https://blog.logrocket.com/angular-vs-react-vs-vue-js-comparing-performance/

25. Метод аналізу ієрархій. [Електронний документ] - Режим доступу: https://dss.tg.ck.ua/ahp-help