БАКАЛАВРСЬКА РОБОТА

БР. ІП - 59.00.00.000 ПЗ

Група ІП-22-3

Банцур Денис

2026

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Банцур Денис Андрійович

(прізвище, ім'я, по батькові)

УДК 004.942
(індекс)

БАКАЛАВРСЬКА РОБОТА

Вплив вибору фреймворків на швидкість створення веб-додатків

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

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

Здобувач освітнього ступеня _____ Д.А. Банцур
(підпис, ініціали та прізвище здобувача)

Науковий керівник ____Храбатин Роман Ігорович, к.ф.м.н, доцент_____
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри

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

Івано-Франківськ – 2026

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доцент. В.В. Бандура
“ ____ ” _____2026 р.

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Банцур Денису Андрійовичу

(прізвище, ім’я, по-батькові)

1. Тема проекту (роботи) "Вплив вибору фреймворків на швидкість створення веб-додатків "

керівник проекту (роботи) Храбатин Роман Ігорович, к.ф.м.н, доцент

затверджені наказом вищого навчального закладу від “ 21 ” квітня 2026 р. № 179/7

2. Строк подання студентом проекту (роботи) 10 червня 2026 р.

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

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

1. Аналіз вимог до програмного забезпечення
2. Аналіз вимог і постановка задачі
3. Проектування системи
4. Реалізація проекту
5. Тестування та впровадження

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

1. Мережевий потік - Android (рис. 1.15, ст.27)
2. Програма чату – потік графічного інтерфейсу (рис.2.2, ст.33)
3. Rails – канал мовлення (рис. 2.5, ст.36)
4. Критерії розгортання застосунку (рис. 3.1, ст.51)
5. Веб-застосунки аналіз TCP (рис. 3.22, ст.68)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

2. Дата видачі завдання **2026 р.**

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Визначення та обгрунтування теми роботи	17.01.2026	виконано
2	Огляд існуючих концепцій, рішень та сервісів в даній області	21.02.2026	виконано
3	Побудова моделі або алгоритму власного рішення	01.03.2026	виконано
4	Документування реалізації власного оригінального рішення вибраними засобами	21.04.2026	виконано
5	Оформлення пояснювальної записки бакалаврської роботи	02.05.2026	виконано

Студент _____Банцур А.І.

Керівник роботи _____Храбатин Р.І.
(підпис)

АНОТАЦІЯ

Бакалаврська робота містить 75 сторінок, 73 рисунки, 3 таблиці, список використаних джерел із 40 найменувань.

Метою роботи є дослідження впливу вибору фреймворків на швидкість створення веб-додатків та аналіз підходів до підвищення ефективності процесу розробки сучасних веб-систем.

Об’єкт дослідження: процес розробки веб-додатків із використанням сучасних фреймворків.

Предмет дослідження: фреймворки веб-розробки, зокрема React.js, Vue.js, Ruby on Rails та Laravel, а також методи оцінювання їх впливу на швидкість розробки програмного забезпечення.

Результати дослідження: проведено аналіз сучасних веб-фреймворків, визначено їх особливості та вплив на продуктивність розробки, а також виконано порівняльний аналіз швидкості створення додатків.

У першому розділі розглянуто теоретичні основи використання фреймворків у веб-розробці, їх роль у побудові програмних систем та підходи до оцінювання ефективності розробки.

У другому розділі досліджено методи та інструменти створення веб-додатків, реалізовано чат-застосунок із використанням різних фреймворків та проведено аналіз їх функціональних можливостей.

У третьому розділі розглянуто моделі порівняння фреймворків, виконано аналіз продуктивності та швидкості розробки, а також узагальнено результати дослідження.

Висновок: вибір фреймворку суттєво впливає на швидкість створення веб-додатків, зручність розробки та ефективність підтримки програмного забезпечення.

КЛЮЧОВІ СЛОВА: ВЕБ-РОЗРОБКА, ФРЕЙМВОРКИ, REACT, VUE, LARAVEL, RUBY ON RAILS, ПРОДУКТИВНІСТЬ, ШВИДКІСТЬ РОЗРОБКИ, ВЕБ-ДОДАТКИ.

ANNOTATION

The bachelor's thesis contains of 75 pages, 73 figures, 3 tables, and a reference list with 40 sources.

The aim of the work is to study the impact of framework selection on the speed of web application development and to analyze approaches to improving the efficiency of modern web system development processes.

Research object: the process of web application development using modern frameworks.

Research subject: web development frameworks, including React.js, Vue.js, Ruby on Rails, and Laravel, as well as methods for evaluating their impact on software development speed.

Research results: an analysis of modern web frameworks was conducted, their features and impact on development productivity were identified, a test web application (chat system) was developed using different technologies, and a comparative analysis of development speed was performed.

The first section describes the theoretical foundations of using frameworks in web development, their role in building software systems, and approaches to evaluating development efficiency.

The second section analyzes methods and tools for web application development, implements a chat application using different frameworks, and evaluates their functionality.

The third section presents comparison models of frameworks, analyzes performance and development speed, and summarizes the research results.

Conclusion: the choice of framework significantly affects the speed of web application development, ease of development, and maintainability of software systems.

KEY WORDS: WEB DEVELOPMENT, FRAMEWORKS, REACT, VUE, LARAVEL, RUBY ON RAILS, PRODUCTIVITY, DEVELOPMENT SPEED, WEB APPLICATIONS.

ЗМІСТ

ВСТУП

У сучасному світі інформаційні технології відіграють ключову роль у розвитку бізнесу, науки та суспільства загалом. Особливо стрімко розвивається сфера веб-розробки, яка забезпечує створення інтерактивних, масштабованих і доступних програмних систем. Зростання вимог до швидкості розробки, якості програмного забезпечення та скорочення часу виходу продукту на ринок (time-to-market) обумовлює необхідність використання ефективних інструментів і технологій. Одним із таких інструментів є фреймворки, які значно спрощують процес розробки веб-додатків, надаючи готові рішення, шаблони та архітектурні підходи.

Актуальність теми зумовлена тим, що вибір фреймворку безпосередньо впливає на швидкість створення веб-додатків, продуктивність розробників, складність підтримки програмного забезпечення та загальну ефективність проєкту. У сучасних умовах існує велика кількість фреймворків, таких як React, Vue, Laravel, Ruby on Rails та інші, кожен з яких має свої особливості, переваги та обмеження. Неправильний вибір технології може призвести до збільшення часу розробки, ускладнення масштабування системи та зниження її продуктивності.

Метою роботи є дослідження впливу вибору фреймворків на швидкість створення веб-додатків, а також аналіз ефективності їх використання в сучасних умовах розробки програмного забезпечення.

Завданнями дослідження є аналіз теоретичних основ використання фреймворків у веб-розробці, визначення критеріїв оцінювання швидкості розробки, дослідження сучасних фреймворків та їх функціональних можливостей, реалізація веб-додатку (чат-системи) з використанням різних технологій, проведення порівняльного аналізу продуктивності та швидкості розробки, а також формулювання висновків щодо доцільності використання конкретних фреймворків у різних умовах.

Об’єктом дослідження є процеси розробки веб-додатків із використанням

сучасних фреймворків, що включають етапи проєктування, реалізації, тестування та впровадження програмного забезпечення.

Предметом дослідження є методи, підходи та інструменти, що впливають на швидкість створення веб-додатків, зокрема особливості використання фреймворків React, Vue, Laravel та Ruby on Rails, їх архітектурні рішення та можливості оптимізації процесу розробки.

Методи дослідження включають аналіз наукових і технічних джерел для вивчення сучасних підходів до веб-розробки, порівняльний аналіз фреймворків, моделювання процесів створення програмного забезпечення, експериментальний метод для реалізації тестового додатку, а також аналіз отриманих результатів з метою оцінювання ефективності використання різних технологій.

У рамках роботи передбачається розробка веб-додатку (чат-системи) з використанням різних фреймворків, що дозволить на практиці оцінити їх вплив на швидкість створення програмного забезпечення. Особлива увага приділяється продуктивності, зручності розробки, повторному використанню коду та можливостям масштабування. Очікується, що результати дослідження дозволять визначити найбільш ефективні підходи до вибору фреймворків залежно від вимог проєкту.

Бакалаврська робота містить 75 сторінок, 73 рисунків, 3 таблиці, 3 розділи, список використаних джерел із 40 найменувань.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ВИБОРУ ФРЕЙМВОРКІВ У ВЕБ-РОЗРОБЦІ

1.1 Сутність фреймворків та їх роль у розробці веб-додатків

Ранні веб-сайти використовували HTML для обміну інформацією та даними. Хоча веб-сайти на основі HTML можуть легко передавати статичний контент, такий як текстові документи, фотографії та блоги, його застосування для складніших моделей досить обмежене. HTML не може задовольнити вимоги до інтеграції додатків, гнучких моделей та портативних середовищ, що призвело до впровадження трирівневої архітектури, яка дозволяє розділити логічні рівні в додатку. Таким чином, ця покращена трирівнева архітектура включає рівень презентації, бізнес-рівень та спеціалізований рівень бази даних. Рівень презентації підтримує інтерфейс користувача та веб-додатку, а бізнес-рівень обробляє пошук та публікацію даних, перевірку та забезпечення дотримання правил. Нарешті, рівень даних підключається до бази даних за допомогою заданих модулів, класів та моделей. Загалом, ця архітектура дозволяє веб-додаткам працювати зі складними моделями та структурами, одночасно відокремлюючи рівень бази даних та поточний рівень інтерфейсу користувача від бізнес-рівня. Окрім загальних переваг модульного програмного забезпечення, ця архітектура була розроблена таким чином, щоб дозволити будь-який із трьох рівнів незалежно оновлювати або змінювати, якщо змінюється будь-яка технологія або вимога [1].

Завдяки цій архітектурі з’явилося безліч фреймворків, а згодом багато компаній розробили технології для доповнення кожного з цих рівнів. Наприклад:

Рівень інтерфейсу користувача : HTML, CSS та JavaScript.

Рівень серверної частини : Perl, PHP, C# та ASP.

Рівень бази даних : SQL, mySQL та Oracle.

Рівень зв'язку між браузером та сервером : XML та AJAX.

Створення веб-сайту або розміщення сервера за допомогою цих технологій – це виснажливий процес. Веб-фреймворк усуває необхідність

програмування та налаштування базового коду з нуля. Тому впровадження фреймворків зробило ці згадані технології застарілими.

В сучасну епоху веб-технологій існує три основні типи застосунків: нативні, веб- та прогресивні веб-застосунки. Нативні застосунки залежать від платформи та потребують спеціалізованих мов програмування та комплектів розробки, тоді як веб-застосунки – це платформонезалежні веб-сторінки, які в багатьох аспектах виглядають як нативні застосунки. Нативні застосунки отримують доступ до апаратного забезпечення пристрою, тоді як веб-застосунки мають обмежений доступ. Однак обидві програми мають свої переваги та недоліки залежно від використовуваного фреймворку та платформи. З іншого боку, прогресивні веб-застосунки (PWA) розробляються з використанням певного набору стандартних шаблонів, що дозволяє цим застосункам працювати в нативному середовищі, зберігаючи при цьому спільну веб-архітектуру.

Сьогодні веб-додатки розробляються з використанням шаблонів та методологій PWA для покращення взаємодії з користувачем та зменшення навантаження на сервер [2]. Крім того, використання CMS значно спрощує розробку додатків та забезпечує кращу інтеграцію та стабільність. Замість написання вихідного коду, CMS дозволяє розробнику створювати, редагувати та керувати полями даних, такими як текст, зображення, аудіо та відео.

Дослідження використовується як основа для вивчення реального випадку веб-розробки та впровадження чат-додатку на різних платформах і пристроях.

Структура документа поділена на три розділи. Перша частина надає глибокий огляд веб- та мобільних фреймворків. У другій представлено порівняння різних моделей фреймворків додатків для кращого розуміння їхніх характеристик. Також обговорюються методи, обрані для оцінки ефективності кожного фреймворку. Після цього аналізується реалізація одного додатку з використанням різних фреймворків. Після цього оцінюється відносна продуктивність фреймворків та узагальнюються всі висновки.

Дослідницька проблема

Останнім часом у сфері веб-розробки було розроблено різноманітні веб-фреймворки. Серед них такі фреймворки, як Angular, React, Swift, Laravel, Express та Flask, здобули успіх. Ці веб-фреймворки надають різні можливості, які підходять для різних типів застосунків. Як результат, існує кілька порівняльних досліджень щодо веб-фреймворків, таких як « *Порівняння мобільних веб-фреймворків»* та *«Модель порівняння для гнучких веб-фреймворків»*. Ці порівняльні дослідження розглядають фреймворки та проводять суб'єктивний аналіз їхніх можливостей.

Не існує моделі порівняння, що базується на аналізі продуктивності та мережі застосунків. Тому дослідження порівняння веб-фреймворків буде зосереджено на аналізі веб-фреймворків з використанням різних критеріїв порівняння, мережевого аналізу та аналізу продуктивності.

Метою роботи є дослідження продуктивності відповідних фреймворків шляхом розробки однакового чат-додатку для популярних веб- та мобільних фреймворків. Визначена продуктивність використовується для проведення перехресного порівняння між усіма відповідними фреймворками.

Метод дослідження

В аналізі було використано комбінації фреймворків для веб-розробки, такі як React та Rails, Angular та Flask, React Native та Express, Vue та Laravel, а також Swift. Тестовий чат-застосунок було реалізовано з використанням усіх цих комбінацій фреймворків. Цей застосунок дозволить провести поглиблений порівняльний аналіз фреймворків.

Для порівняльного аналізу враховуються такі критерії, як розробка, простота розгортання, простота модифікації, продуктивність фреймворку. Аналогічно, мережевий аналіз фреймворків включає аналіз пакетів HTTP, TCP та WebSocket.

Впроваджений чат-додаток працював протягом 10 хвилин і часто оновлювався для генерації унікального трафіку. Цей трафік потім фіксувався для детального аналізу.

1.2 Концептуальні підходи до оцінювання впливу фреймворків на швидкість розробки

З появою нових технологій веб-розробки щороку розробники повинні аналізувати кожен фреймворк додатків, придатний для проекту [3]. Аналіз базується на технічних перспективах, таких як мережевий аналіз додатку. Нещодавно випущений фреймворк може пропонувати нові переваги та цілісність, але він може бути доступний лише для певних платформ та операційних систем. Тому стає необхідністю вибирати фреймворки, які добре підходять для проекту. Це підказує кілька важливих питань, таких як:

1) Що таке фреймворк?
2) Як виконується мережевий аналіз фреймворку?
3) Чи існує модель порівняння для фреймворків?

Відповідно, у наступних розділах буде визначено концепцію мережевого аналізу, визначення та класифікацію сучасних фреймворків, детальну інформацію про фреймворки та модель порівняння для фреймворків. Маючи попередні знання про порівняльні результати різних фреймворків, легше вибрати фреймворк, який найкраще працюватиме з необхідною методологією управління проектами та операційною платформою.

Мережевий трафік

Мережеві пакети складають мережевий трафік. Мережевий аналіз, також відомий як аналіз мережевого трафіку, можна описати як метод аналізу мережевих пакетів. Він включає моніторинг різних мережевих дій, таких як HTTP, TCP та WebSocket, збір відповідної інформації та її аналіз у режимі реального часу. Його можна використовувати для різних цілей, таких як виявлення несправностей у мережі, вивчення поведінки мережі та виявлення шкідливої діяльності. Крім того, він також корисний для виявлення вразливих протоколів, які можуть стати ціллю будь-якої шкідливої діяльності в майбутньому. Хоча це може не бути дуже корисним на особистому рівні, це надзвичайно важливо на корпоративному рівні, де тисячі машин підключені до різних ланцюгів серверів та

мережевих пристроїв. Ці пристрої генерують величезний обсяг трафіку. Отже, ці сервери є найбільш підходящим вибором для більшості зловмисників, тому мережевий аналіз забезпечує значну ідентифікацію для подолання цих загроз.

Основні мережеві запити в застосунку обробляються через протоколи передачі, а саме: протокол керування передачею (TCP), протокол передачі гіпертексту (HTTP) та WebSocket (WS).

TCP – це протокол, орієнтований на з'єднання, який визначає, як комп'ютери надсилають пакети даних один одному. TCP встановлює з'єднання та підтримує його, доки обмін пакетами не завершиться. Він використовується для безпечної організації пакетів даних, щоб забезпечити належну передачу. TCP відіграє важливу роль у встановленні правил та стандартних процедур для здійснення зв'язку через Інтернет.

TCP (Рисунок 1.1) об'єднаний у концептуальну модель обміну даними, яка є моделлю стека TCP/IP. Модель стека TCP/IP перепаковує дані на кожному рівні на основі функціональності та транспортних протоколів.

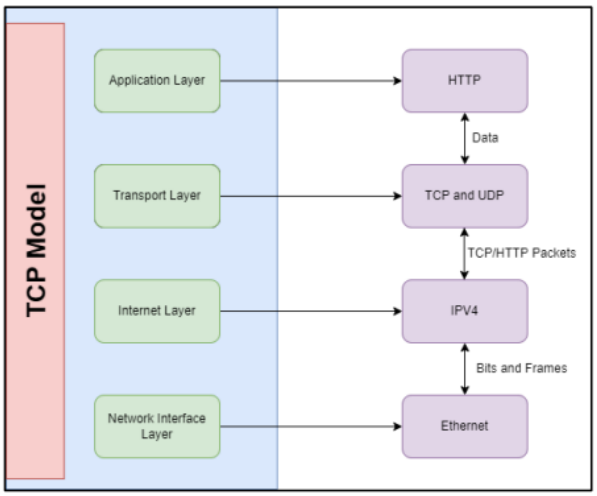


Рисунок 1.1 - Модель TCP.

HTTP (Рисунок 1.2) – це протокол, який використовує послуги TCP-портів для полегшення передачі документів через Інтернет [4]. На відміну від інших з’єднань, HTTP використовує лише одне TCP-з’єднання, яке називається «Data

Link», для передачі. Це клієнт-серверний протокол, який отримує HTML-документи для обміну інформацією в Інтернеті. Клієнтська сторона доставляє повідомлення запиту на веб-сервер, який надсилає запитуваний контент назад клієнту. Під час запиту та відповіді HTTP використовує Secure Socket Layer (SSL) для захисту всього зв’язку.

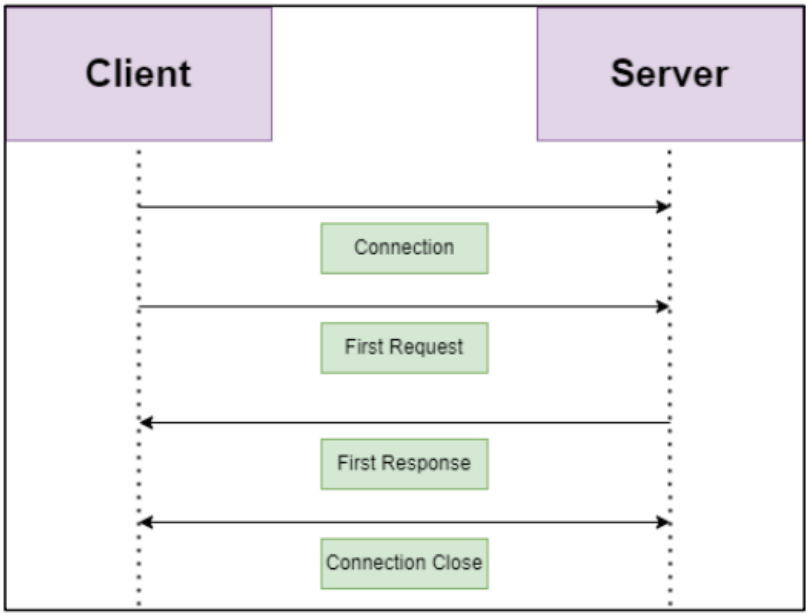


Рисунок 1.2 - Модель HTTP

WebSocket (Рисунок 1.3) – це протокол зі збереженням стану, що означає, що він надсилає запит на сервер і очікує відповіді. Якщо відповідь не отримано, протокол зі збереженням стану повторно надсилає запит. WebSocket підтримує з'єднання між клієнтом і сервером, доки воно не буде перервано будь-якою стороною. Дані передаються між клієнтом і сервером безперервно, що виключає час опитування запитів. Серверу не потрібно чекати стану клієнта, перш ніж відповісти на його запит. На відміну від HTTP-запитів, WebSocket підтримує двонаправлене з'єднання з клієнтом і сервером. Після того, як клієнт...

WebSocket служить головною метою передачі інформації в режимі реального часу від сервера до клієнта швидше, ніж HTTP [5]. Отже, торгові програми, ігрові програми та програми чату використовують WebSocket.

На рисунку 1.3 показано, як виконується рукостискання HTTP-запиту між клієнтом і сервером. Після цього рукостискання встановлюється двонаправлений зв'язок, а потім з'єднання розривається.

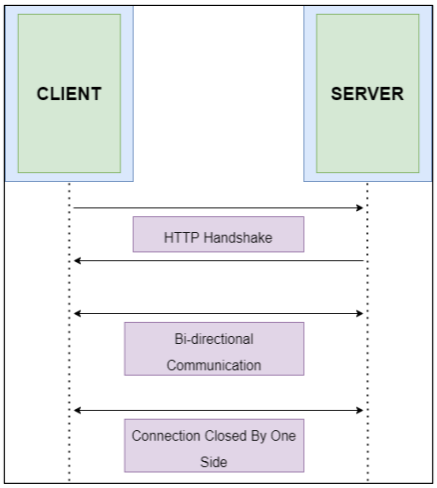


Рисунок 1.3 - З'єднання WebSocket.

Вебсайт по суті поділяється на дві основні частини: фронтенд та бекенд . Ці основні частини стосуються поділу шару інтерфейсу користувача та шару даних.

Фронтенд (Рисунок 1.4) веб-сайту можна описати як інтерфейс веб-сайту або веб-застосунку, доступний користувачеві.

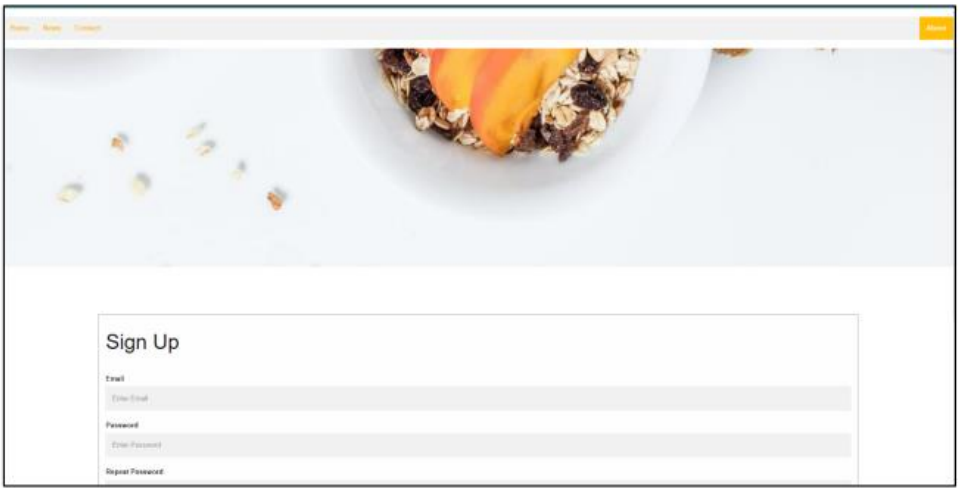


Рисунок 1.4 - Приклад інтерфейсу

Він також відомий як клієнтська частина веб-сайту або веб-застосунку. Фронтенд створюється за допомогою таких технологій, як HTML, CSS та JavaScript. Вся взаємодія із сервером здійснюється завдяки фронтенду [6]. Фронтенд відповідає за архітектуру веб-сайту, забезпечуючи простий, але елегантний користувацький досвід без шкоди для будь-якої функціональності.

Бекенд або серверна частина веб-сайту обробляє функціональність сервера. Усі запити та відповіді обробляються сервером веб-сайту або веб-застосунку. Це може включати отримання даних, публікацію інформації, оновлення даних з баз даних та до них. Він також обробляє всю логіку, необхідну для веб-сайту або веб-застосунку. Після виконання запитів бекенд надсилає відповідь на фронтенд, щоб він міг відобразити вигляд для користувача.

Коротка історія веб-фреймворків

Веб-фреймворки вважаються найважливішою частиною будь-якого веб-застосунку. Веб-фреймворк – це набір інструментів, які допомагають створювати веб-сайт, уникаючи помилок та заощаджуючи час. Фреймворки можуть використовуватися як на статичних, так і на динамічних веб-сторінках. Відповідно, фреймворк веб-застосунку можна визначити як багаторазовий набір бібліотек коду мови програмування та інструментів, призначених для підтримки розробки веб-застосунку. Перевагою використання веб-фреймворків є комплексна ефективність та організація коду.

Існує безліч типів веб-фреймворків (рис. 1.5), але всі вони належать до категорії прогресивних, нативних або гібридних. Ці фреймворки створені із загальною метою додавання більшої кількості веб-компонентів до оригінальної мови програмування.

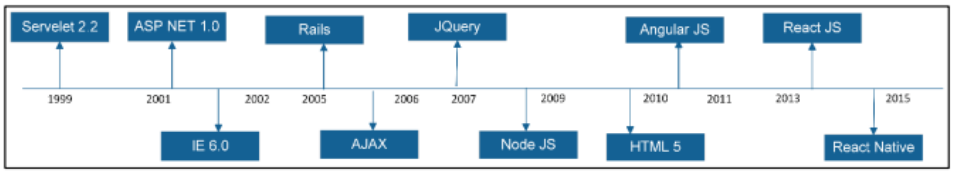


Рисунок 1.5 - Еволюція веб-фреймворків

Вебфреймворки зростали в популярності, оскільки JavaScript ставав більш функціональним у браузерах. У 2004 році розробка техніки Asynchronous JavaScript and XML (AJAX) (Рисунок 1.6) стала затребуваною для створення динамічних вебсайтів без необхідності повного оновлення сторінки. Попит на веброзробку з використанням JavaScript зобов'язав W3C запровадити новий набір правил [7]. Ці правила вимагали від постачальників браузерів впровадження технологій, що сприяють стандартизації вебтехнологій, таких як HTML, JavaScript/ECMAScript та CSS.

Щоб подолати поганий користувацький досвід і водночас полегшити маніпуляції з DOM, веб-фреймворки почали розвиватися. Наступне покоління фреймворків забезпечувало гарну клієнт-серверну архітектуру, відповідало сучасним веб-стандартам і було досить стабільним, що пропонувало розширений користувацький досвід. Поява фреймворків AngularJS та Backbone довела, що створення веб-застосунків, які могли б працювати безпосередньо в браузері без потреби в надзвичайно швидких комп'ютерах, стало життєздатним.

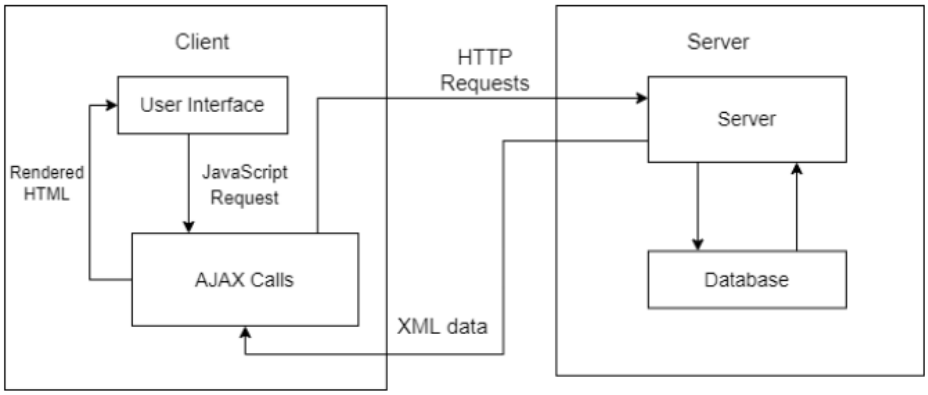


Рисунок 1.6 - Запити AJAX

Еволюція веб-фреймворків. Рання веб-розробка

До появи веб-фреймворків веб-сайти мали просту програмну архітектуру. Вона складається з: рівня презентації, бізнес-рівня та рівня персистенції. Концепція ізоляції кожного шару призводить до того, що кожен шар працює незалежно

від іншого. Жоден шар не залежить від заломлення світла іншого шару. Це ізолює зміни та полегшує подальший розвиток цього шару (Рисунок 1.7).

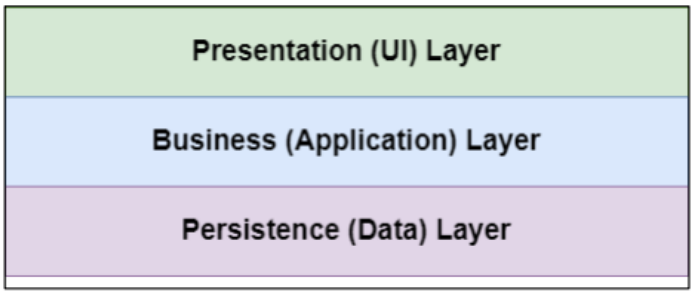


Рисунок 1.7 - Трирівнева архітектура.

Рівень презентації відомий як рівень інтерфейсу користувача (UI). Це найвищий рівень старої архітектури програмного забезпечення. Цей рівень надає послуги презентації, які включають демонстрацію контенту кінцевому користувачеві за допомогою графічного інтерфейсу користувача (GUI). Доступ до рівня можна отримати через різні клієнтські пристрої, такі як ноутбук, настільний комп'ютер, мобільний телефон і планшет. Рівень представляє контент, взаємодіючи з іншими рівнями в моделі архітектури програмного забезпечення.

Бізнес-рівень

Бізнес-рівень також відомий як прикладний рівень, який є середнім рівнем архітектури програмного забезпечення. Цей рівень дотримується набору правил, необхідних для роботи програми [8]. Отже, цей рівень складається з бізнес-логіки, яка зазвичай працює на одному або кількох серверах програм.

Шар стійкості

Рівень збереження, який описується як рівень даних. Це найнижчий рівень моделі програмного забезпечення. Цей рівень відповідає за зберігання та отримання даних програми з бази даних, файлового сервера або будь-якого іншого носія інформації.

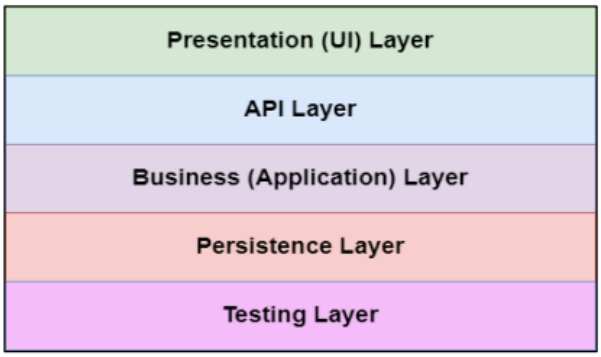


Рисунок 1.8 - Додані рівні — трирівнева архітектура.

Великі компанії дотримувалися того ж набору методів розробки адаптивних веб-сайтів, використовуючи цю програмну архітектуру. Хоча деякі веб-сайти вимагають високої зв'язності, коли фрагменти коду пов'язані один з одним. Тому, щоб вирішити цю проблему, до архітектури програмного забезпечення додаються деякі додаткові рівні (Рисунок 1.8).

Під час розвитку веб-розробки у 2000-х роках багато інтернет-сайтів покладалися на сервери для відображення вигляду на стороні клієнта. Браузер діяв як засіб перегляду, який відображав весь згенерований результат. Цей результат включав динамічно згенеровані сервером сторінки HTML та CSS. Ці веб-сайти дотримувалися стандартної структури Model-View-Controller або структури MVC.

Модель-Вид-Контролер

Контролер представлення моделі або MVC (рис. 1.9) – це веб-архітектура, яка відповідає стандартній архітектурі програмного забезпечення. Рівень моделі складається з рівня даних, який обробляє всю логіку модифікації. Аналогічно, рівень контролера обробляє бізнес-логіку та забезпечує зв'язок між виглядом та рівнем моделі. Нарешті, рівень представлення складається з рівня представлення та обробляє весь інтерфейс користувача, доступний користувачеві [9].

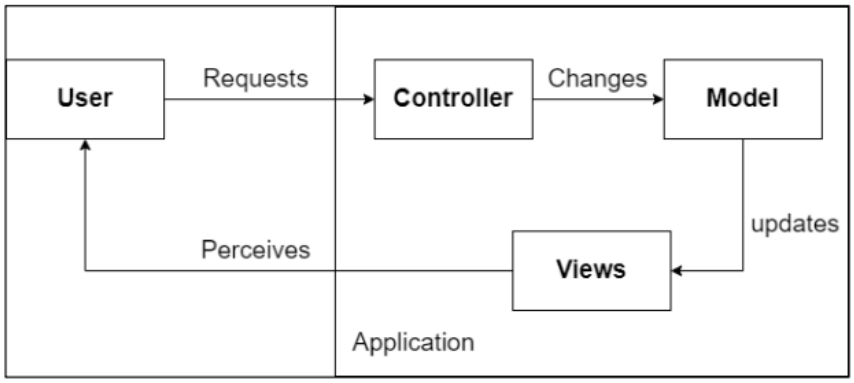


Рисунок 1.9 - Структура MVC.

Усі запити та відповіді потім можуть оброблятися jQuery або AJAX, щоб зробити веб-сторінку інтерактивною. Хоча ця архітектура стикається з багатьма складнощами, такими як поєднання компонентів, інтеграція фронтенду (представлення) та серверної частини (модель та контролер), що призводить до недостатньої архітектури.

У 2000-х роках багато компаній зіткнулися з тією ж проблемою зв'язку з архітектурою MVC, що змусило розробників використовувати відокремлені API від сервера. Хоча це вимагало від програми роботи з плагінами браузера, які потребували окремого встановлення. Ці плагіни часто були застарілими та мали різні вразливості безпеки, що робило їх застарілими для запуску веб- та нативних додатків.

Архітектура моделі-вигляду-вигляду

На початку 2000-х років ці проблеми були вирішені шляхом запровадження клієнтської архітектури Model-View-ViewModel (MVVM) з REST API (Representational State Transfer).

REST — це інтерфейс прикладного програмування або API, який дозволяє кільком клієнтам взаємодіяти із сервером. Він забезпечує розробникам велику гнучкість, оскільки усуває залежність від бібліотек коду для доступу до вебсервісів. Він обробляє запити у формі параметрів. Ці параметри включають кінцеву точку API, метод API та дані, що передаються за допомогою API.

Кінцева точка: Кінцева точка – це унікальна URL-адреса, яка представляє

об’єкт даних. HTTP-запити спрямовуються до кінцевої точки для взаємодії з усіма ресурсами даних. Дані також називають тілом запиту, яке представляє ресурс. Дані містять необхідну інформацію про ресурс.

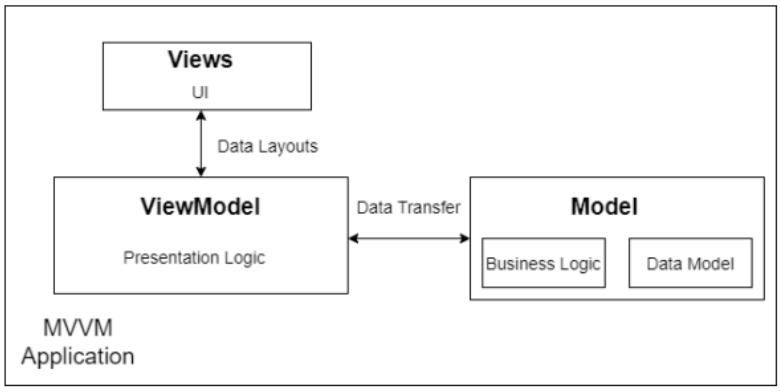


Рисунок 1.10 - Структура MVVM.

MVVM (Рисунок 1.10) розділяє рівні представлення, логіки та моделі, що значно спрощує розробку графічного інтерфейсу порівняно з архітектурою MVC. Рівень ViewModel цієї архітектури взаємодіє з Rest API, щоб надати запитуваний ресурс клієнту та представити його за допомогою рівня View.

Однак, як клієнтська, так і серверна сторони реалізують власні рівні архітектури програмного забезпечення, що ускладнює архітектуру.

Ранні веб-фреймворки, такі як Backbone та AngularJS, мали труднощі з розробкою веб-сайтів через погано продуману архітектуру [9]. У деяких випадках API повертали моделі даних, які надавали доступ до реляційних моделей даних веб-застосункам, що створювало нову вразливість безпеки. Крім того, погано відображена структура даних створювала неконтрольований зв'язок між шарами архітектури, яка створювала безумовні складності. Отже, виникла потреба в нових веб-фреймворках.

Сучасна веб-розробка

Сучасна веб-розробка дотримується сучасних фреймворків. Ці фреймворки забезпечують належну архітектуру, відповідають сучасним веб-стандартам, забезпечують масштабованість та стабільність.

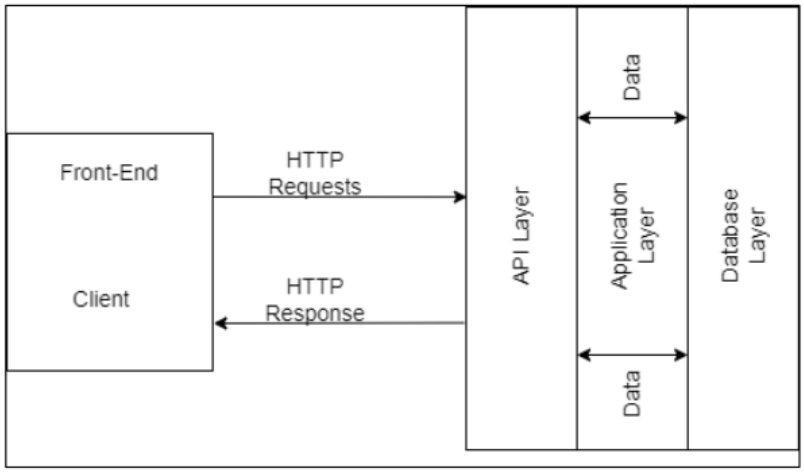


Рисунок 1.11 - Сучасна структура запиту API.

Сучасні додатки (рис. 1.11) використовують шари API для вирівнювання моделі даних перед надсиланням відповіді клієнту, що робить передачу даних оптимальною та усуває потребу в спеціалізованих API.

Веб-додатки

У термінології веб-розробки веб-застосунок можна описати як клієнтський та серверний програмний застосунок, у якому клієнт працює та запитує дані у веб-браузері. Прикладами таких веб-застосунків є служби обміну повідомленнями, веб-сайти роздрібної торгівлі, поштові клієнти та онлайн-форми.

У таблиці 1.1 перераховано деякі сучасні фреймворки веб-застосунків, класифіковані за мовами програмування back-end та front-end.

Таблиця 1.1

Структура на основі мови програмування.

Мова програмування	**Front-End фреймворк**	**Back-End фреймворк**
JavaScript	React, Vue, Angular	Express (Node JS), Next.js
Python	Turbogears, Django	Django, Flask
Ruby	Ruby	Rails
PHP	Відсутні (Nil)	Laravel, Symfony

Для роботи веб-застосунку (рис. 1.12) потрібен веб-сервер, сервер застосунків та база даних. Усі запити від клієнта обробляються веб-серверами, а запитуване завдання виконує сервер застосунків. Вся необхідна інформація

зберігається в базі даних.

До цих програм можна отримати доступ через кілька браузерів, до них можуть отримати доступ кілька користувачів одночасно, і вони не потребують завантаження [10]. Отже, ці програми є ідеальним кандидатом для доставки контенту користувачам.

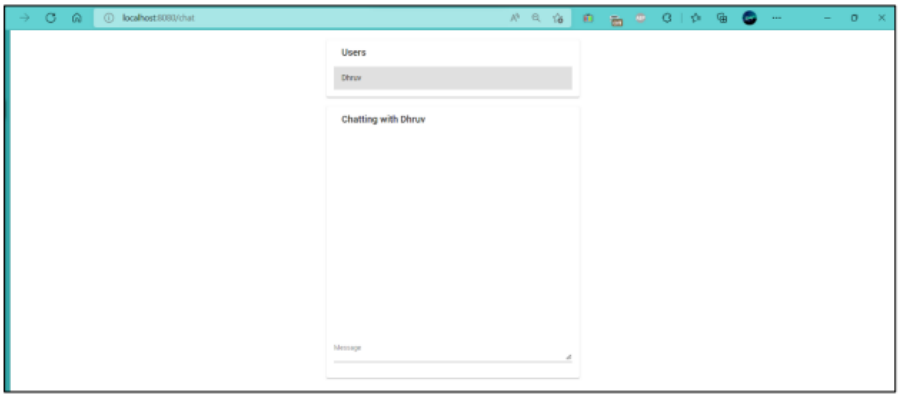


Рисунок 1.12 - Приклад — веб-додаток

Нативні та кросплатформні додатки

Сучасна веб-розробка визначає нативні додатки як програми, написані для роботи на певній платформі. Ці нативні додатки (Рисунок 1.13) розроблені для платформ iOS та Android. Ці додатки працюють з ОС мобільного пристрою, щоб швидше доставляти контент запиту та забезпечувати більшу гнучкість, ніж альтернативні додатки.

Кросплатформні фреймворки працюють на ідеї розробки коду повторного використання для створення додатків для різних ОС. Наприклад, той самий код для Android-додатку можна використовувати для розробки iOS-додатку з іншою архітектурою.

Ці програми можуть отримати доступ до таких функцій пристрою, як мікрофон, гіроскоп або push-сповіщення [11]. Таким чином, практичне використання цих програм варіюється від простих музичних програм, таких як «Black Player Ex», до соціальних додатків, таких як «Twitter» або «Facebook».

Нижче наведено нативні та кросплатформні фреймворки додатків,

класифіковані за ОС пристрою:

Android : React-Native (JS), Native Scripts (JS), Flutter та Java

iOS : Swift (Objective-C), Flutter, React-Native (JS) та Xamarin (C#)

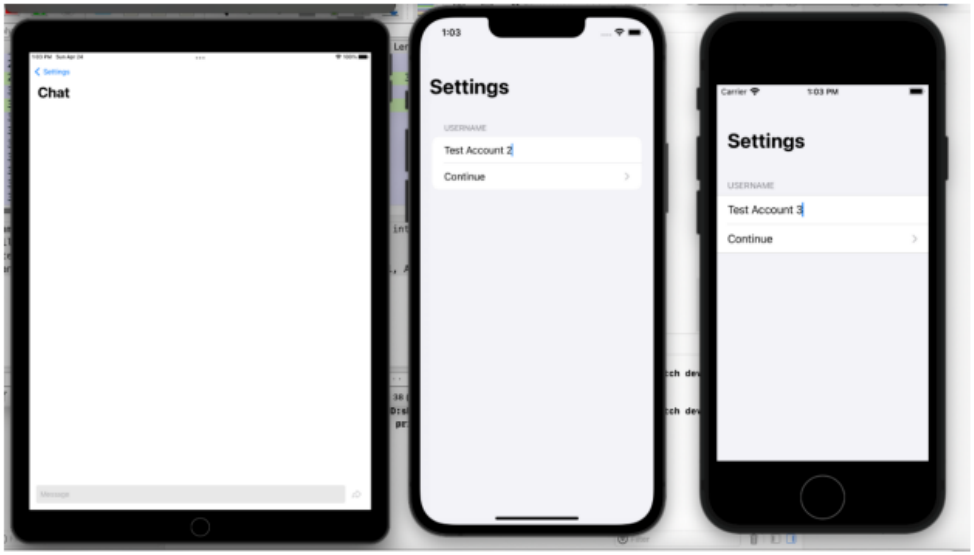


Рисунок 1.13 - Приклад — нативний додаток.

1.3 Дані та методи дослідження ефективності використання фреймворків

Метою цього дослідження є аналіз продуктивності різних фреймворків, що запускають один і той самий додаток на одному сервері. Для отримання необхідного набору даних для різних фреймворків було використано методологію захоплення мережі. Найважливішою частиною захоплення мережі було ізолювання інших фонових мережевих дій від мережевих запитів сервера чату (Рисунок 14).

Отримання даних для вебзастосунків легше порівняно з нативними застосунками, оскільки фоновий зв'язок ускладнює отримання необхідного набору даних. Тому важливо ізолювати зовнішні мережі від основного сервера чату.

Генерація даних. Нативні та кросплатформні додатки.

Нативні та кросплатформні додатки поділяються на дві категорії: Android

та iOS [12]. Існують різні пристрої та інструменти для категоризації згенерованих даних, які детально пояснюються в наступних розділах:

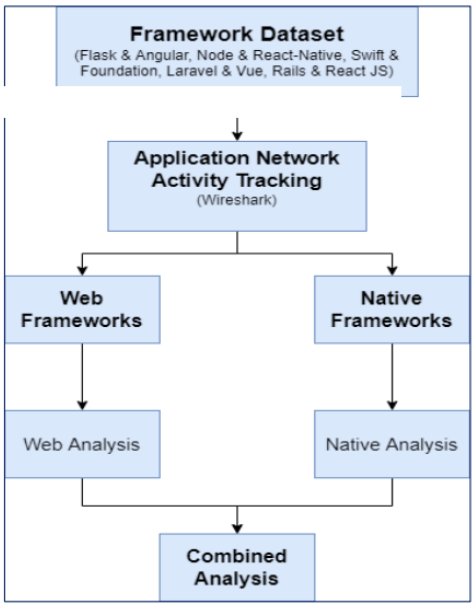


Рисунок 1.14 - Схема дослідження.

Андроїд

Для Android-додатків для створення набору даних використовується емулятор Android. Як емулятори використовуються Pixel 4 (версія Android 11), Pixel 2 (версія Android 10) та фізичний портативний пристрій One Plus 8T з Android 12. На цих пристроях встановлено додаток «Packet Capture» для блокування зовнішніх даних та захоплення необхідних пакетів. Найкращий спосіб ізолювати мережеву активність додатка — це блокувати всі непотрібні з'єднання та, таким чином, приймати лише необхідні запити від виділеного сервера чату. Таким чином можна уникнути небажаного трафіку. Вбудований фреймворк використовує пакети WebSocket та TCP для надсилання запитуваних даних та отримання відповіді. Згенеровані дані були зібрані за допомогою мережевих пакетів у форматі PCAP. Стек мережевих протоколів на стороні сервера був захоплений за допомогою додатка Wireshark для ізоляції пакетів TCP, HTTP та WebSocket.

«Захоплення пакетів» використовує вбудовану функцію VPN Android для запису всього мережевого трафіку, що спрощує класифікацію для різних

програм. Для запуску вбудованої програми чату було використано «Expo Go». «Захоплення пакетів» встановлює мережевий інтерфейс, який потім налаштовує правила маршрутизації та надає програмі дескриптор. Кожен дескриптор забезпечує інформативний аналіз кожного вхідного та вихідного пакета на мережевому інтерфейсі [13]. Програма встановлює локальне VPN-з'єднання з головним сервером через тунель (Рисунок 1.15). Захоплені пакети далі надсилаються до зовнішньої мережі, після чого може бути проведений подальший аналіз.

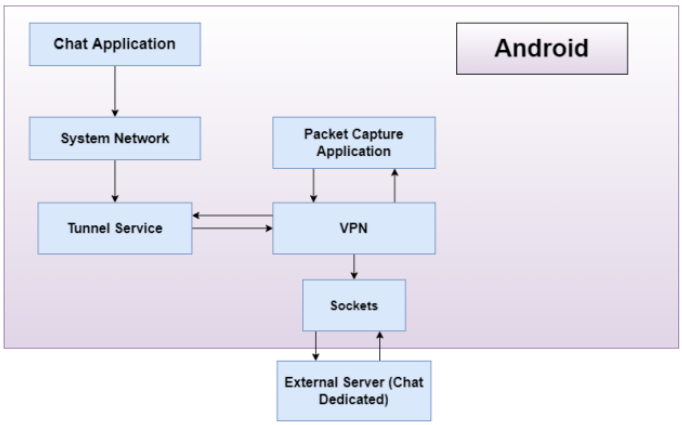


Рисунок 1.15 - Мережевий потік — Android.

Пакети фіксуються у форматі PCAP, а потім аналізуються в застосунку Network-Analysis (Карстенс, nd). Пакети представлені графічно для кращого аналізу фреймворку (Рисунок 1.16).

Рисунок 1.16 - Деталі перехопленого пакета — Android

Для iOS-додатку для захоплення мережі використовуються емулятори: iPad (9-го покоління з iOS 15.4), iPod touch (7-го покоління з iOS 15.4) та iPhone 13 pro (з iOS 15.4). Оскільки всі запуски виконуються в емуляторах, зовнішня мережа не потребує фільтрації. Усі непотрібні запити фільтруються шляхом призначення простого проксі-сервера, яким у цьому випадку є «Localhost» (Рисунок 1.17).

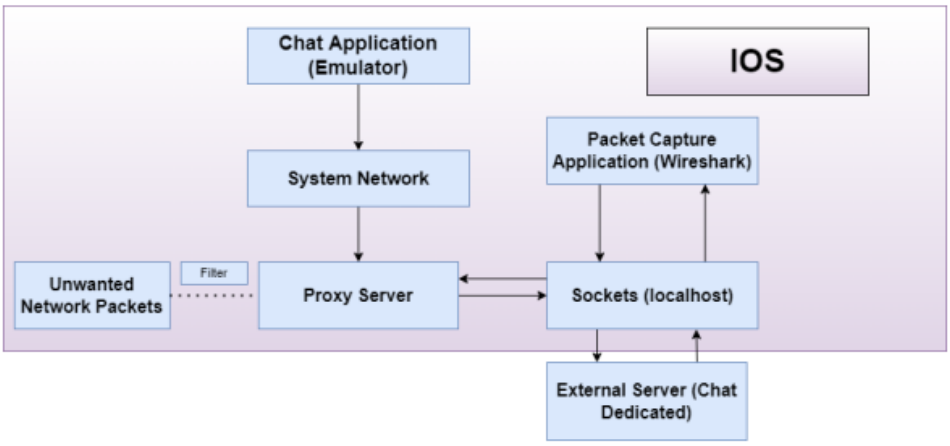


Рисунок 1.17 - Мережевий потік — iOS

Усі необхідні мережеві пакети (HTTP, WebSocket та TCP) були захоплені та відстежені за допомогою Wireshark, після чого були згенеровані мережеві графіки.

Веб-додатки

Аналіз мережевого трафіку веб-застосунків є менш складним порівняно з нативними застосунками. Для збору мережевих даних веб-застосунків на Windows та Mac використовується мережевий адаптер зворотного зв'язку для реєстрації всіх відповідних пакетів [14]. Таким чином, весь непотрібний фоновий мережевий трафік обмежується.

Тестування програм у середовищі Mac відрізняється від тестування у Windows. Щоб вирішити цю проблему, для середовища Mac було визначено спеціальний проксі-сервер.

Проксі-сервер працює на сервері "Localhost", що фільтрує непотрібні

мережеві пакети.

Для тестування застосунків Windows затримкою, яка витрачається на постійний зв'язок між протоколом простого виявлення послуг (SSDP) та локальним хостом, не враховується.

Наступне рівняння визначає співвідношення між загальним часом, витраченим програмою, та часом, витраченим SSDP під час зв'язку.

Рівняння 1. Затримка на сервері — Windows.

$$\text{totalTime}_{\text{app}_1} = \text{time}_{\text{app}_1} + \text{time}_{\text{communication}_{\text{SSDP}_1}} \quad (1.1)$$

Рівняння 2. Часова затримка – Windows.

$$\text{totalTime}_{\text{app}_2} = \text{time}_{\text{app}_2} + \text{time}_{\text{communication}_{\text{SSDP}_2}} \quad (1.2)$$

У всіх сценаріях час, витрачений SSDP, залишається постійним, тому ним можна знехтувати, враховуючи кінцеву різницю в часі. Це дає точне вимірювання різниці в часі між програмами, що призводить до кращого аналізу.

$$\text{time}_{\text{communication}_{\text{SSDP}_1}} = \text{time}_{\text{communication}_{\text{SSDP}_2}} \quad (1.3)$$

Рівняння 3. Отримана різниця в часі на серверах.

$$\Rightarrow \text{time}_{\text{difference}} = \text{time}_{\text{app}_1} - \text{time}_{\text{app}_2} \quad (1.4)$$

Для вилучення необхідних мережевих пакетів було використано застосунок Wireshark. Застосунок налаштовано для роботи через мережу чат-сервера для

захоплення WebSocket та HTTP-трафіку. Він захоплює мережеві пакети у вигляді "бітів" з локального Ethernet-адаптера системи та представляє їх відповідно до стандартної еталонної моделі OSI.

Рівень 7, або прикладний рівень моделі OSI, відповідає за ініціювання запитів, тоді як рівень 4 (транспортний) та рівень 3 (мережевий) обробляють усі мережеві запити за допомогою пакетів TCP, HTTP та WebSocket (рис. 1.18).

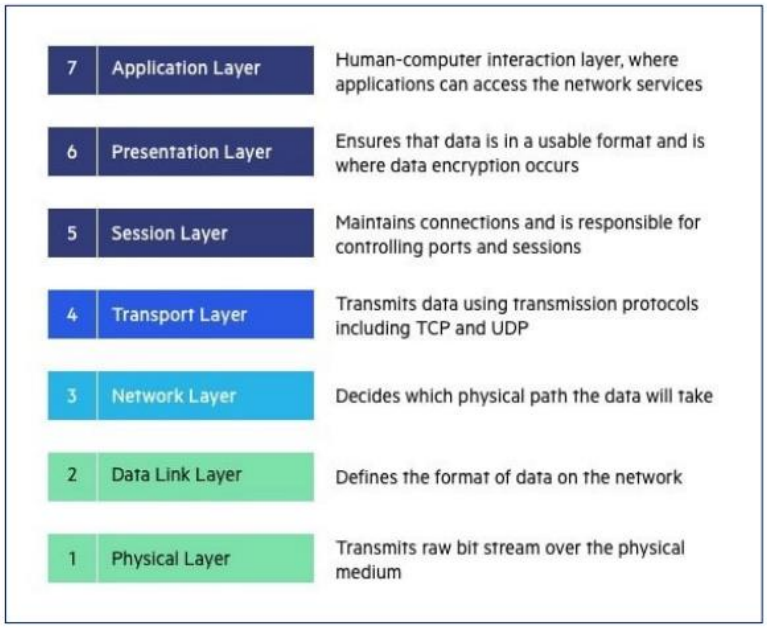


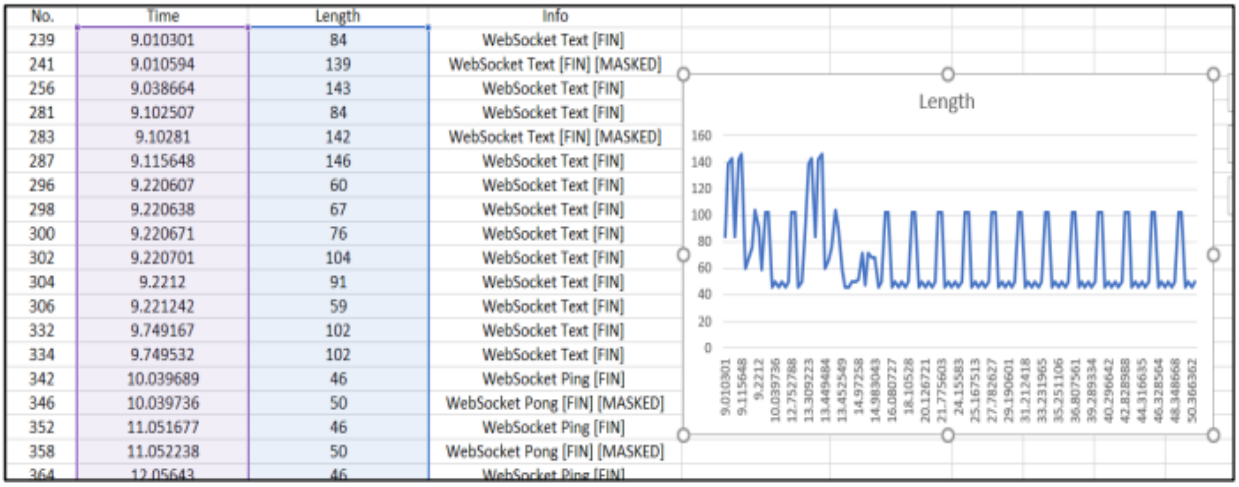
Рисунок 1.18 - Модель OSI

Методи

Wireshark використовується для відкриття файлу PCAP, який містить згенеровані пакети, а вибрані відображені поля витягуються у файл значень, розділених комами (CSV) [15]. Файли CSV класифікуються на запити TCP, WebSocket та HTTP. Усі ці файли CSV містять «час, витрачений на запити джерела», «час, витрачений на запити призначення», «довжину пакетів» та «кількість пакетів на кожен запит».

CSV-файли містять інформацію про кількість та довжину пакетів, згенерованих протягом певного періоду часу [16]. Детальні CSV-файли відображаються у вигляді графіка, щоб мати чітке розуміння порівняння різних

пакетів для різних програм, включаючи нативні та веб-програми (Рисунок 1.19).



No.	Time	Length	Info
239	9.010301	84	WebSocket Text [FIN]
241	9.010594	139	WebSocket Text [FIN] [MASKED]
256	9.038664	143	WebSocket Text [FIN]
281	9.102507	84	WebSocket Text [FIN]
283	9.10281	142	WebSocket Text [FIN] [MASKED]
287	9.115648	146	WebSocket Text [FIN]
296	9.220607	60	WebSocket Text [FIN]
298	9.220638	67	WebSocket Text [FIN]
300	9.220671	76	WebSocket Text [FIN]
302	9.220701	104	WebSocket Text [FIN]
304	9.2212	91	WebSocket Text [FIN]
306	9.221242	59	WebSocket Text [FIN]
332	9.749167	102	WebSocket Text [FIN]
334	9.749532	102	WebSocket Text [FIN]
342	10.039689	46	WebSocket Ping [FIN]
346	10.039736	50	WebSocket Pong [FIN] [MASKED]
352	11.051677	46	WebSocket Ping [FIN]
358	11.052238	50	WebSocket Pong [FIN] [MASKED]
364	12.05643	46	WebSocket Ping [FIN]

Рисунок 1.19 - CSV – Згенерована діаграма.

1.4 Висновки по розділу

У першому розділі було розглянуто теоретичні основи використання фреймворків у веб-розробці та їх вплив на швидкість створення веб-додатків. Визначено сутність фреймворків, їх ключові переваги та роль у стандартизації процесу розробки програмного забезпечення. Проаналізовано концептуальні підходи до оцінювання ефективності фреймворків, зокрема критерії продуктивності, масштабованості, швидкості розробки та зручності підтримки проєктів. Також досліджено сучасні методи аналізу ефективності використання фреймворків і встановлено, що правильний вибір технологічного стеку суттєво впливає на терміни реалізації, якість програмного продукту та ефективність командної роботи.

РОЗДІЛ 2. МЕТОДИ ТА ІНСТРУМЕНТИ РОЗРОБКИ ВЕБ-ДОДАТКІВ ІЗ ВИКОРИСТАННЯМ ФРЕЙМВОРКІВ

2.1 Реалізація веб-додатку з використанням сучасних фреймворків

Мета цього підрозділу — надати уявлення про реалізацію чат-застосунку з використанням розглянутих фреймворків: React JS та Rails, Angular та Flask, Vue та Laravel, Swift з Objective-C та React-Native з Express. Кожен чат-застосунок базується лише на одному шаблоні, який включає аналогічно розроблений фронтенд, бекенд та модель бази даних. Таким чином, застосунок є подібним на всіх фреймворках.

Додаток для чату обробляє текстове спілкування через мережу за допомогою веб-сокетів. У додатку можна налаштувати текстове спілкування в чаті між двома або більше користувачами. Кімната чату також дозволяє кільком користувачам обмінюватися повідомленнями та взаємодіяти одночасно [17].

Програма поділена на два модулі, які включають базовий екран входу та чат. Це детальніше пояснюється в графічному інтерфейсі програми. Графічний інтерфейс – це візуальне представлення шляху, який користувач проходить під час використання програми.

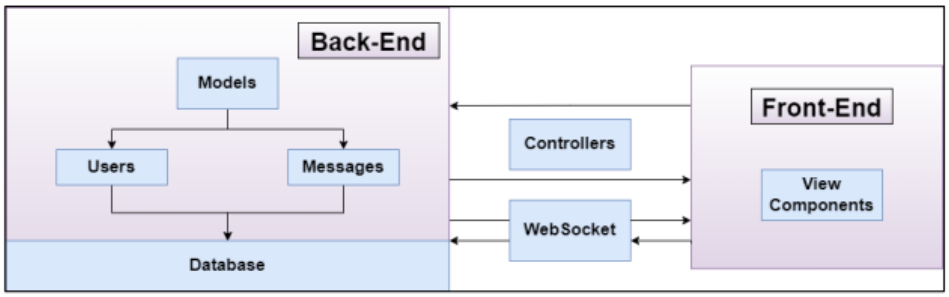


Рисунок 2.1 - Структура чат-застосунку.

Усі дані обробляються моделями, що працюють у бек-енді, тоді як фронтенд має контролери, які отримують інформацію для відображення користувачеві

(Рисунок 2.1).

Фронтенд підтримує безперервне з'єднання із сервером за допомогою веб-сокетів. З'єднання триває, доки його не закриє будь-яка сторона, яка бере участь у зв'язку. Зв'язок вважається завершеним, коли всі пакети WebSocket передано через порт TCP і не залишилося жодного запиту чи відповіді.

Потік графічного інтерфейсу

Додаток поділено на два модулі. Перший екран – це проста сторінка входу, де користувачеві потрібно лише ввести ім’я користувача. Другий екран, який є екраном чату, доступний лише після екрана входу.

Екран чату дозволяє спілкування або між двома користувачами, або між усіма користувачами в кімнаті чату (Рисунок 2.2).

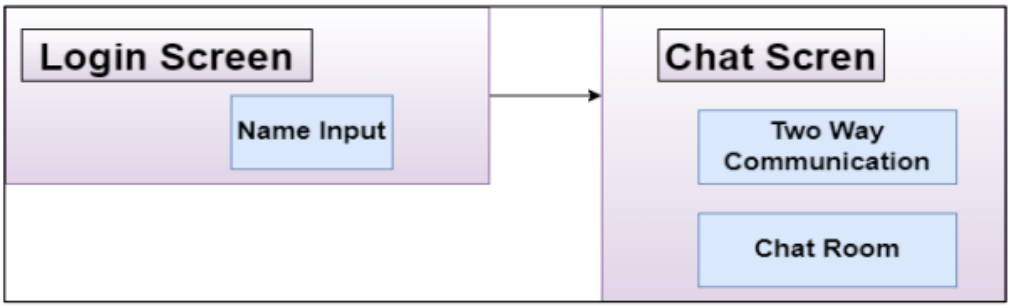


Рисунок 2.2 - Програма чату – потік графічного інтерфейсу.

Моделі

Як і потоки графічного інтерфейсу, модель домену також поділена на два класи: Користувач та Повідомлення. Модель користувача містить схему користувача, яка визначає, як користувач додається та змінюється в базі даних. Аналогічно, схема повідомлення чату визначає додавання та представлення повідомлення чату (Рисунок 2.3).

Модель повідомлення чату

На рисунку 2.3 чітко видно, що модель повідомлення чату містить певні параметри, такі як:

Вміст: Це основна частина повідомлення чату. Це повідомлення, яке

користувач надсилає іншому користувачеві або до кімнати чату для прочитання іншими користувачами.

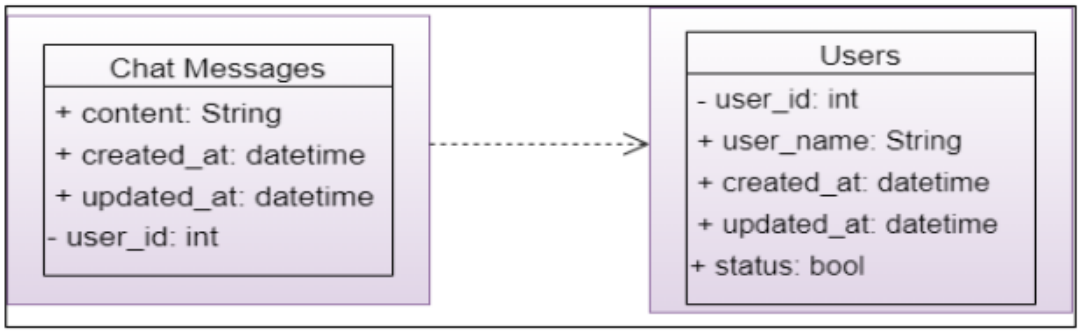


Рисунок 2.3 - Додаток чату – Моделі даних.

Створено/Оновлено о: Ці параметри визначають дату та час повідомлення. Ці параметри включають час створення повідомлення користувачем та час оновлення повідомлення.

Ідентифікатор користувача: Цей параметр визначає користувача повідомлення. Він безпосередньо пов'язаний з моделлю користувача, щоб ім'я користувача відображалося разом із надісланим повідомленням у чаті.

Модель користувача

Як і модель чату, модель користувача містить параметри, які пояснюються так:

Ідентифікатор користувача: це параметр «ID» для користувачів. Він пов’язаний з моделлю чату, діючи як зовнішній ключ [18]. Використовується для відображення повідомлень, пов’язаних з користувачем.

Ім'я користувача : Цей параметр визначає ім'я користувача. Ім'я користувача відображається в чаті щоразу, коли користувач надсилає повідомлення.

Статус: Цей параметр визначає статус користувача. Він повідомляє іншим користувачам, чи активний цей користувач.

Створено/Оновлено о: Ці параметри визначають дату та час повідомлення. Ці параметри описують позначку часу повідомлення, тобто час створення

повідомлення та час оновлення повідомлення користувачем.

Веб-додатки

Усі застосунки веб-чату дотримуються схожої ідеї реалізації. Окрім залежностей фреймворку, кожен застосунок було встановлено із залежністю WebSocket та залежністю CORS (Cross-Origin Resource Sharing). CORS – це заголовок на основі протоколу HTTP [19]. Це дозволяє серверу завантажувати ресурси до будь-якого зовнішнього домену, щоб завантажувати ресурси, відмінні від дозволеного локального домену.

2.2 Аналіз використання React.js та Ruby on Rails у веб-розробці

Перший чат-застосунок розроблено з використанням фронтенд-фреймворку React JS та бекенд-фреймворку Rails. React базується на мові програмування JavaScript, тоді як Rails виник на основі мови Ruby. Подібно до інших реалізованих застосунків, цей також базується на графічному інтерфейсі, показаному на рисунку 2.2, та моделі бази даних, показаній на рисунку 2.3.

Для згаданих технологій, що розглядаються, основний застосунок розділений на кілька модулів [20]. Перед складанням застосунку реалізовано навігацію між кількома модулями.

Після тестування навігації, застосунок розділено на кілька компонентів. Ці компоненти включають: модуль чату, модуль повідомлень, модуль списку користувачів та модуль користувача. Всі ці компоненти працюють разом, формуючи стек навігації для створення інтерфейсу користувача застосунку, який сприймається користувачем.

Як пояснено на рисунку 2.4, він чітко показує, що користувач повинен спочатку увійти в систему, перш ніж приєднатися до чату. Компонент 'currentUserLoggedIn' визначає доступність екрана чату.

Тільки авторизовані користувачі мають доступ до навігації екраном чату та повідомлень, де вони також можуть спілкуватися та взаємодіяти один з одним. Після екрана входу користувач переходить на екран чату. Екран чату дозволяє

спілкуватися між користувачами.

```
{currentUserLoggedIn &&
  <div className="grid grid-cols-8 gap-1 rounded-md">
    <div className="col-span-1 bg-ill-main min-h-screen" >
      <Users />
    </div>
    <div className="col-span-7 flex flex-col justify-between">
      <div classNames="">
        <Messages />
      </div>
      <div>
      <Entry />
      </div>
    </div>
  </div>
```

Рисунок 2.4 - React JS – Вхід користувача.

Бекенд

Фронтенд та бекенд з'єднані за допомогою WebSocket. Усі запити та відповіді клієнта надсилаються/отримуються через WebSockets. Ці запити потім передаються на сервер, де зберігаються в базі даних. У цьому випадку як база даних використовується PostgreSQL.

Модуль «ApplicationCable» на рисунку 2.5 допомагає інтегрувати WebSocket у за стосунок [21]. З'єднання також транслюється та транслюється у застосунку, що дозволяє кільком користувачам взаємодіяти без будь-яких проблем із портами.

```
cation.rb   Gemfile   Untitled-1   config.ru   user_s
app > channels > messages_channel.rb
1  class MessagesChannel < ApplicationCable::Channel
2    def subscribed
3      stream_from 'message_channel'
4    end
5
6    def unsubscribed; end
7  end
```

Рисунок 2.5 - Rails – канал мовлення.

Робочий приклад

На екрані чату відображаються активні користувачі та канал зв'язку (Рисунок 2.6).

Рисунок 2.6 React — робочий додаток чату

Сервер також входить у консоль, що допомагає налагоджувати з'єднання між клієнтом і сервером. Консоль відображає кожне повідомлення, починаючи від реєстрації користувача і закінчуючи повідомленнями (Рисунок 2.7).

```
ANSACTION (0.2ms)  BEGIN
app/controllers/users_controller.rb:16:in `add_message'
ssage Create (0.8ms)  INSERT INTO "messages" ("content", "created_at", "updated_at", "user_id") VALUES ($1, $2, $3, $4) RETURNING "id"  [
ated_at", "2022-05-17 06:48:22.635119"], ["user_id", 1]]
app/controllers/users_controller.rb:16:in `add_message'
ANSACTION (11.5ms)  COMMIT
app/controllers/users_controller.rb:16:in `add_message'
ionCable] Broadcasting to message_channel: #<Message id: 1, content: "Dhruv sends: Me Alone Right now!", created_at: "2022-05-17 06:48:22
leted 200 OK in 60ms (ActiveRecord: 15.5ms | Allocations: 13631)
```

Рисунок 2.7 - Журнали сервера React — Chat Application.

Тестування

Тестування включає два тести. Тестовий випадок «Звіт про з’єднання» перевіряє статус, який має бути 200. Цей статус означає, що відповідь сервера «ОК».

Тест «Додавання відповіді» перевіряє, чи отримано надіслані дані сервером без будь-яких помилок даних.

З рисунка 2.8 видно, що застосунок пройшов обидва тести в тестовому застосунку сервера (у цьому випадку postman).

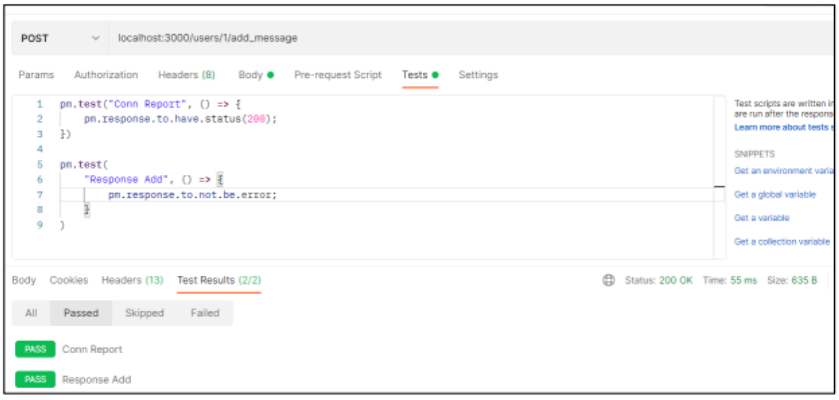


Рисунок 2.8 - React — тестування застосунку для чату «Postman».

Angular JS та Flask

Другий чат-застосунок було розроблено за допомогою фронтенд-фреймворку Angular та бекенд-фреймворку Flask. Застосунок відповідає графічному інтерфейсу, показаному на рисунку 2.2, та моделі бази даних, показаній на рисунку 2.3.

Фронтенд. Далі виконуються ті ж кроки ініціалізації, що й у попередньому випадку [22]. Як видно на рисунку 2.9, шлях «chat» у розділі «Маршрути» має параметр «canActivate». Цей параметр має значення AuthGuard, яке змушує користувача увійти в програму, перш ніж вона зможе використовувати функції чату.

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { Routes, RouterModule } from '@angular/router'; // CLI imports router
import { LoginComponent } from './login/login.component';
import { AuthGuard } from './auth.guard';
import { ChatComponent } from './chat/chat.component';


const routes: Routes = [
  { path: '', component: LoginComponent },
  { path: 'chat', component: ChatComponent, canActivate: [AuthGuard] },
  { path: '**', component: LoginComponent },
```

Рисунок 2.9 - Angular – Вхід.

Бекенд

У цьому випадку серверна частина застосунку написана за допомогою Flask. На початковому етапі готується віртуальне середовище для запуску та тестування застосунку. Після підготовки середовища невдовзі встановлюються залежності застосунку.

Для підключення до фронтенду потрібне стабільне з'єднання за допомогою WebSocket [23]. Запити та відповіді відповідають стану WebSocket. Якщо стан WebSocket підключений, то з'єднання між клієнтом і сервером стабільне (Рисунок 2.10).

```
users = {}
@websocket.on('connect')
def on_connect():
    print('Client connected')
    websocket.emit('my response', {'data': 'Connected'})

@websocket.on('disconnect')
def on_disconnect():
    users.pop(request.sid,'No user found')
    websocket.emit('current_users', users)
    print("User disconnected!\nThe users are: ", users)
```

Рисунок 2.10 - З’єднання Flask з WebSocket.

Програма також відстежує запити та відповіді, отримані й надіслані сервером. Консольні повідомлення допомагають у налагодженні програми, водночас надаючи уявлення про запити програми.

Робочий приклад

На екрані чату відображаються активні користувачі та можливість спілкування з ними (Рисунок 2.11).

Повідомлення консолі надають візуальне уявлення про функціональність серверної частини. Перша подія описує стабілізоване з'єднання між клієнтом і сервером. Після встановлення з'єднання на сервері реєструється новий користувач. Консоль відображає ідентифікатор та ім'я користувача.

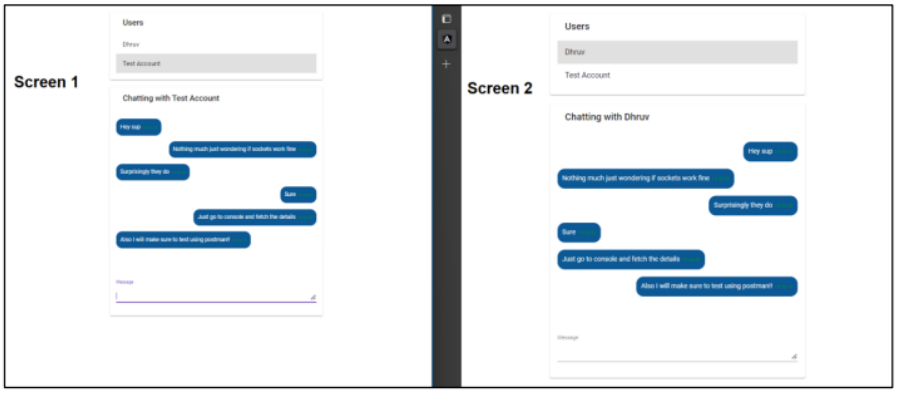


Рисунок 2.11 - Angular – Робочий додаток чату.

Перед відключенням від сервера повідомлення користувача також відображаються в консолі [24]. Це надає уявлення про вміст повідомлення користувача та інші атрибути, пов'язані з повідомленням, такі як дата, час, відправник та одержувач (Рисунок 2.12).

Рисунок 2.12 - Журнали сервера Angular – Chat Application.

Тестування

Тестування включає два тести: «Додавання відповіді» та «Звіт про підключення». Перший перевіряє наявність помилок у запитах та відповідях програми, а другий – стан програми. Програма пройшла обидва тести з відповіддю «OK».

2.3 Аналіз використання Vue.js та Laravel у створенні веб-додатків

Третій чат-застосунок реалізовано за допомогою фронтенд-фреймворку Vue JS та бекенд-фреймворку Laravel. Як і інші фронтенд-фреймворки, Vue також базується на мові програмування JavaScript, тоді як Laravel розроблено на мові PHP. Як і інші реалізовані застосунки, цей також базується на моделі графічного інтерфейсу, зображеній на рисунку 2.2, та моделі бази даних, зображеній на рисунку 2.3.

Фронтенд

Після ініціалізації застосунку як проєкту, його також поділяють на кілька різних компонентів [25]. Ці компоненти включають компонент чату та головний компонент. Першим кроком є реалізація головного екрана входу для користувачів, а потім починається розробка екрана чату.

Перед ініціалізацією будь-якої іншої функціональності розробляються компоненти навігації. Ці компоненти допоможуть користувачеві перейти від екрана входу до екрана чату.

Як видно на рисунку 2.13, компоненти chat містять два компоненти: 'mainLoginComponent' та 'ChatsComponent'. Ці компоненти описують маршрути таким чином, що екран чату недоступний без попереднього входу в програму.

```
// const files = require.context('./', true, /\.vue$/i)
// files.keys().map(key => Vue.component(key.split('/').pop().split('.')[0], files(key).default))
Vue.component('chats', require('./components/mainLoginComponent.vue'));
Vue.component('chats', require('./components/ChatsComponent.vue'));
```

Рисунок 2.13 - Vue – компоненти входу.

Бекенд

Бекенд цієї програми написано за допомогою Laravel. Першим кроком є підготовка локального сервера для запуску програми. Локальний сервер готується за допомогою програми Xampp (Рисунок 2.14).

Рисунок 2.14 - Xampp - сервер Apache та база даних SQL

На рисунку 2.14 увімкнено служби Apache та MySQL. Ці служби надають можливість запускати сервер та базу даних локально.

Модуль API pusher (рис. 2.15) у Laravel допомагає ініціалізувати та транслювати повідомлення за допомогою WebSocket. Контролери налаштовані на трансляцію каналу чату з будь-якого джерела, таким чином вирішуючи проблему CORS. Повідомлення та користувачі потім зберігаються в базі даних MySQL, яка використовує ту саму модель даних, що обговорювалася раніше.

```
'apps' => [
    [
        'id' => env('PUSHER_APP_ID'),
        'name' => env('APP_NAME'),
        'key' => env('PUSHER_APP_KEY'),
        'secret' => env('PUSHER_APP_SECRET'),
        'enable_client_messages' => true,
        'enable_statistics' => true,
        'encrypted' => true,
    ],
],
```

Рисунок 2.15 - Laravel — API штовхача.

Усі дії, включаючи запити та відповіді, реєструються в консолі, що допомагає відстежувати з'єднання WebSocket. Дані консолі включають створення

користувачів, події користувачів (отримання та надсилання повідомлень) та деталі користувача (Рисунок 2.16).

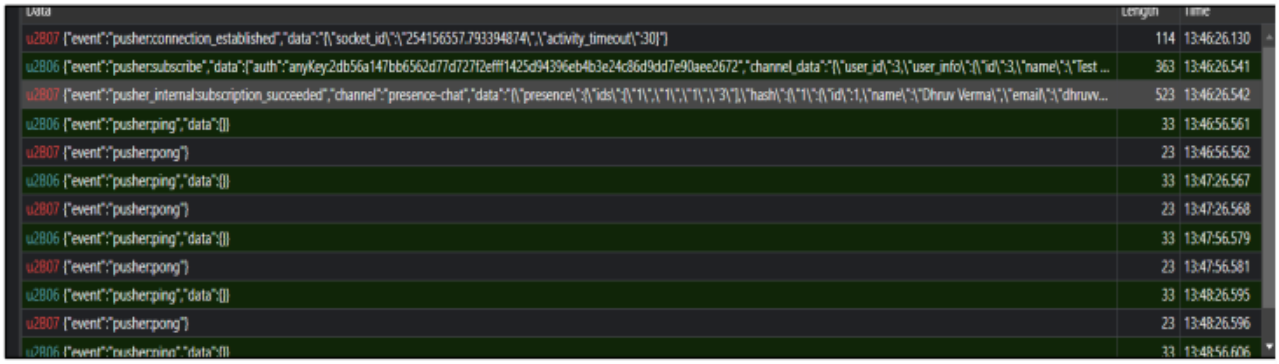


Рисунок 2.16 - Журнали сервера застосунків Vue Chat.

На екрані чату відображаються активні користувачі та можливість спілкування з ними (Рисунок 2.17).

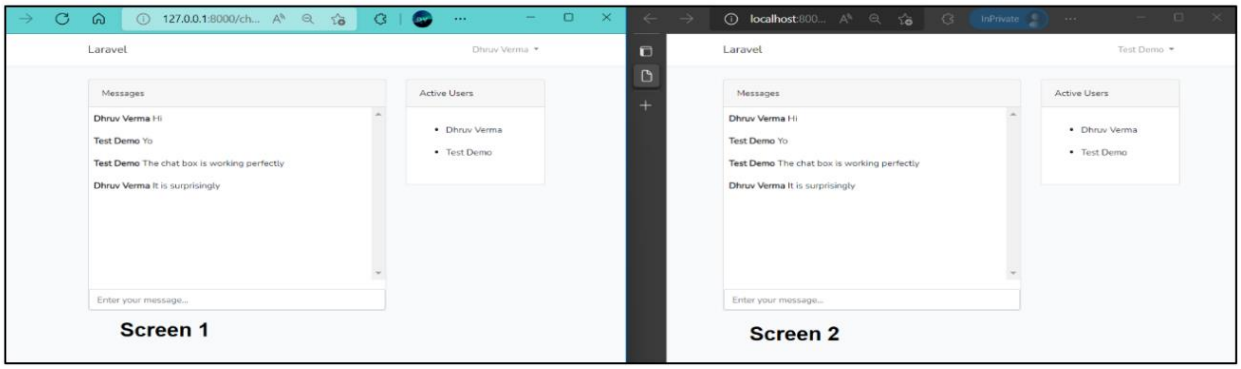


Рисунок 2.17 - Vue Робоча програма чату.

Тестування

Тестування включає два тести. Запит на публікацію надсилається на сервер із повідомленням у форматі JSON. Тест «Conn Report» перевіряє, чи статус відповіді дорівнює 200, що перекладається як відповідь «OK». Застосунок отримав відповідь зі статусом «OK».

Нативні та кросплатформні програми

Нативні додатки реалізуються виключно для мобільних пристроїв залежно від ОС. Однак кросплатформні додатки розробляються відповідно для роботи на різних платформах, таких як iOS та Android [26]. Ці чат-додатки дотримуються тієї ж ідеології спілкування за допомогою WebSocket. Окрім залежностей фреймворку, ці реалізовані додатки були встановлені із залежністю WebSocket та залежністю інтерфейсу користувача для клієнтської частини.

Четвертий чат-застосунок розроблено з використанням бібліотек фреймворку Swift. Базований на мові програмування Objective-C, цей фреймворк проектує застосунки для платформи iOS. Фронтенд розроблено з використанням бібліотеки SwiftUI для забезпечення зручного взаємодії з користувачем. Бекенд базується на бібліотеці Foundation фреймворку Swift. Подібно до веб-застосунків, цей нативний застосунок також реалізовано з використанням моделі, зображеної на рисунку 2.2, та моделі бази даних, зображеної на рисунку 2.3.

Фронтенд

Подібно до попередніх випадків, проєкт, розроблений у цьому випадку, також поділено на два компоненти, які включають: компонент входу та компонент чату. Спочатку розробляється головний екран, де користувач може увійти. Після цього реалізується посилання навігації.

```
//
//  Created by Dhruv Verma
//

import SwiftUI

struct SettingsScreen: View {
    @EnvironmentObject private var userInfo: UserInfo

    private var isUsernameValid: Bool {
        !userInfo.username.trimmingCharacters(in: .whitespaces).isEmpty
    }

    var body: some View {
        Form {
            Section(header: Text("Username")) {
                TextField("E.g. Dhruv Verma", text: $userInfo.username)

                NavigationLink("Continue", destination: ChatScreen())
                    .disabled(!isUsernameValid)
            }
        }
```

Рисунок 2.18 - Компоненти інтерфейсу користувача Swift.

Компонент «navigationLink» (Рис. 2.18) переводить користувача з екрана входу на екран чату. Перед реалізацією компонента чату, представлення ініціалізується для тестування навігації.

Бекенд

Першим кроком є підготовка сокетного з’єднання для запуску програми. Сокетне з’єднання регулярно пінгується кожні 10 секунд, щоб перевірити стабільність з’єднання між клієнтом і сервером. (Рисунок 2.19)

```swift
// Defining websockets connection
var clientConnections = Set<WebSocket>()

// Pinging to Sockets in order to make sure the connection is not closed
app.webSocket("chat") { req, client in
    client.pingInterval = .seconds(10)

    clientConnections.insert(client)

    client.onClose.whenComplete { _ in
        print("Disconnected:", client)
        clientConnections.remove(client)
    }

    client.onText { ws, text in


        do {
            guard let data = text.data(using: .utf8) else {
                return
            }

            let incomingMessage = try JSONDecoder().decode(SubmittedChatMessage.self, from: data)
```

Рисунок 2.19 - Swift підключення до WebSocket

Дані зберігаються в локальній базі даних після визначення моделей для компонентів чату та користувача. Для захисту застосунку всі дані хешуються (Рис. 2.20) перед передачею через з'єднання WebSocket до бази даних.

```swift
import Foundation
import WebSocketKit

extension WebSocket: Hashable {
    public static func == (lhs: WebSocket, rhs: WebSocket) -> Bool {
        ObjectIdentifier(lhs) == ObjectIdentifier(rhs)
    }

    public func hash(into hasher: inout Hasher) {
        hasher.combine(ObjectIdentifier(self))
    }
}
```

Рисунок 2.20 - Swift Хешування даних.

Робочий приклад

Запити користувачів та відповіді сервера реєструються для коректного тестування та налагодження [27]. Ці журнали допомагають відстежувати дані у WebSocket-з'єднанні. Термінал виводить дані, які включають вхід користувача та повідомлення чату користувача (Рисунок 2.21).

```
ReceivingChatMessage(date: 2022-04-24 10:04:33 +0000, id: C22FB709-0E64-4A9E-8994-3EF6918275C3, message: "Join the
    server", user: "Test Account 1", userID: 083EEB1D-C914-4C5A-88FC-29E79B7177CC)
ignoring singular matrix: ProjectionTransform(m11: 5e-324, m12: 0.0, m13: 0.0, m21: 0.0, m22: 5e-324, m23: 0.0, m31:
    0.0, m32: 0.0, m33: 1.0)
ReceivingChatMessage(date: 2022-04-24 10:04:46 +0000, id: 434DEDBE-F4A6-4A0E-A62E-589841FAA2D2, message: "Which CS or
    Halo", user: "Test Account 2", userID: 16F44E75-C4B9-4779-9536-F61B70D9A126)
ignoring singular matrix: ProjectionTransform(m11: 5e-324, m12: 0.0, m13: 0.0, m21: 0.0, m22: 5e-324, m23: 0.0, m31:
    370.0, m32: 0.0, m33: 1.0)
ignoring singular matrix: ProjectionTransform(m11: 5e-324, m12: 0.0, m13: 0.0, m21: 0.0, m22: 5e-324, m23: 0.0, m31:
    370.0, m32: 0.0, m33: 1.0)
ReceivingChatMessage(date: 2022-04-24 10:04:53 +0000, id: CED8C603-284F-4307-ADC6-AE82CE1E5DC9, message: "I vote for
    CS Go", user: "Test Account 3", userID: 3FD56254-BEF1-45C6-8999-81797A3A9C54)
ignoring singular matrix: ProjectionTransform(m11: 5e-324, m12: 0.0, m13: 0.0, m21: 0.0, m22: 5e-324, m23: 0.0, m31:
    0.0, m32: 0.0, m33: 1.0)
ReceivingChatMessage(date: 2022-04-24 10:05:15 +0000, id: 00B9D82F-536A-42CD-A327-2DB8E35391E3, message: "Sure… CS go
    and then black Ops Cod", user: "Test Account 1", userID: 083EEB1D-C914-4C5A-88FC-29E79B7177CC)
ignoring singular matrix: ProjectionTransform(m11: 5e-324, m12: 0.0, m13: 0.0, m21: 0.0, m22: 5e-324, m23: 0.0, m31:
    0.0, m32: 0.0, m33: 1.0)
```

Рисунок 2.21 - Swift журнали сервера застосунків чату.

На екрані чату відображаються активні користувачі та надається можливість спілкуватися в чаті. Робоча демоверсія реалізована на трьох різних симуляторах iOS для перевірки сумісності нативного додатка з різними пристроями (рис. 2.22).

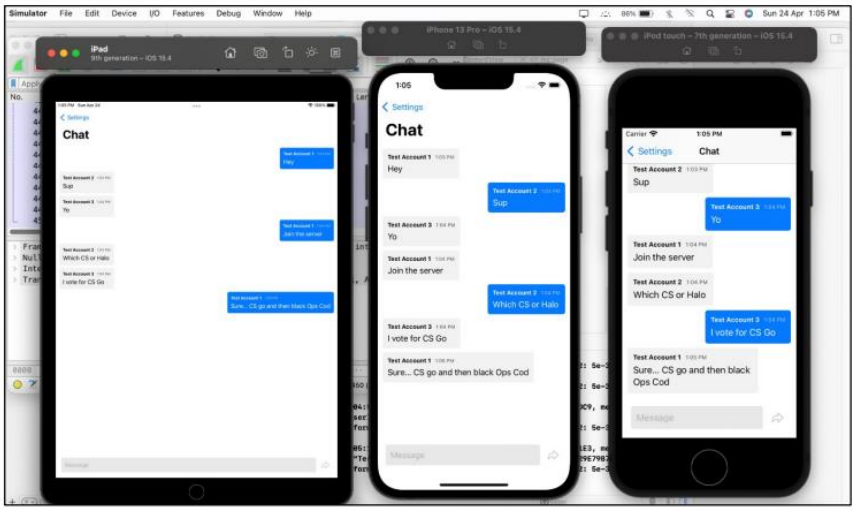


Рисунок 2.22 - Swift Робоча програма чату.

Тестування

Тестування нативного застосунку відрізняється від інших веб-застосунків. Фреймворк Swift не дозволяє доступ до локального сервера за допомогою сторонніх інструментів, таких як Postman [28]. Тому з'єднання було перевірено вручну за допомогою консолі браузера Safari (Рисунок 2.23).

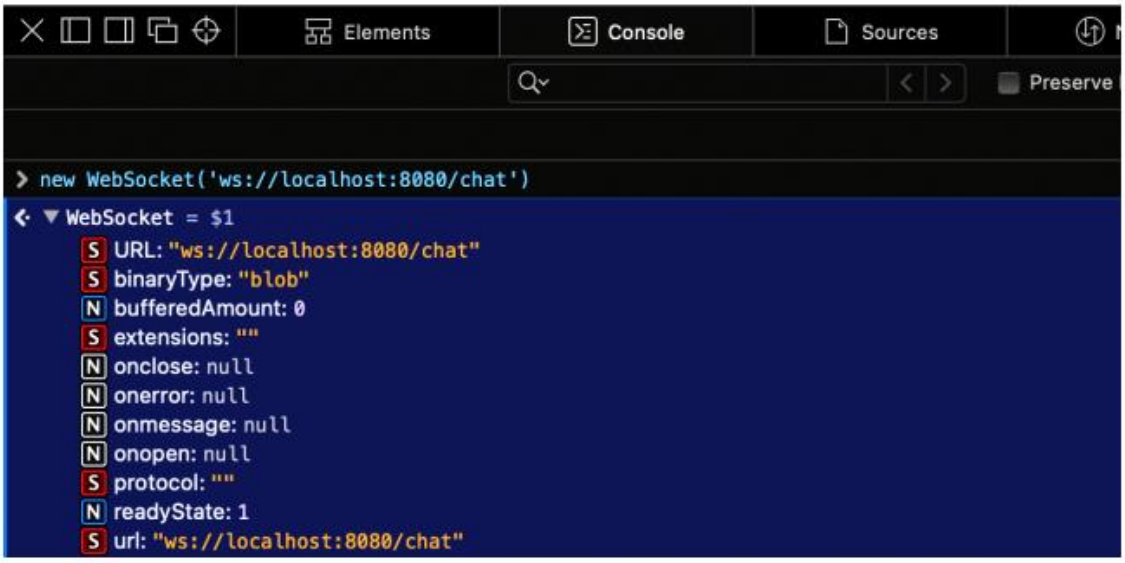


Рисунок 2.23 - Swift Ручна перевірка з’єднання.

React-Native та Express

Зрештою, цей застосунок розроблено з використанням фронтенд-фреймворку React-Native та бекенд-фреймворку Express. Як фронтенд, так і бекенд-фреймворки базуються на JavaScript. Як і інші реалізовані застосунки, цей також дотримується моделі інтерфейсу користувача, як показано на рисунку 21, та моделі бази даних, як показано на рисунку 2.3.

Фронтенд

Після ініціалізації проєкту його також розділяють на два компоненти: компонент «Користувач» та компонент «Чат». Компонент «Користувач» відповідає за функціонал входу в систему, а компонент «Чат» — за функціонал обміну повідомленнями. У цьому випадку головний екран розробляється разом із стеком навігації. Потім головний екран ініціалізує компонент входу в систему, а після

цього — компонент «Чат» (малюнок 2.24).

Рисунок 2.24 - Стек навігації React-Native.

Бекенд

Першим кроком для створення сервера для застосунку є реалізація WebSocket-з'єднання [29]. Сокетне з'єднання перевіряється на стабільність. На рисунку 2.25 показано код для з'єднання між клієнтською та серверною сторонами.

Рисунок 2.25 - Підключення React-Native WebSocket

Робочий приклад

Повідомлення користувачів записуються в консоль. Ці журнали допомагають відстежувати дані у WebSocket-з'єднанні. Термінал виводить дані, які включають ім'я користувача та повідомлення чату користувача. (Рисунок 2.26)

```
F:\Thesis\r-native-app-final\server>node index.js
Listening to port 8080
TestAccount1 : Hello false
Test New  : Hi false
Test New  : I am bored false
TestAccount1 : Same here bro false
```

Рисунок 2.26 - Журнали сервера застосунків React-Native Chat.

Як видно на рисунку 2.27, зрозуміло, що два користувачі приєднуються до чату шляхом входу в систему. Усі спільні повідомлення та відповідні імена користувачів відображаються в консолі. Текстові повідомлення обмінюються між підключеними користувачами.

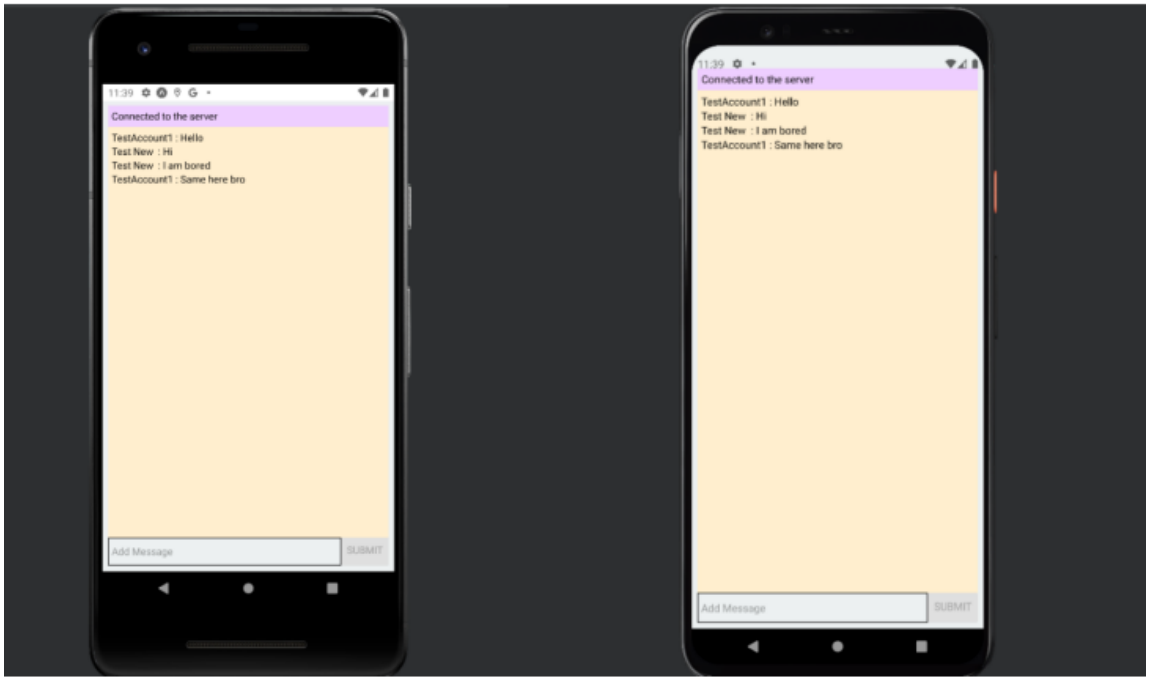


Рисунок 2.27 - Робочий додаток чату на React-Native

Тестування застосунку для Android є відносно простішим порівняно з тестуванням для iOS [30]. Платформа Windows дозволяє емуляторам Android запускати тести Postman на стороні сервера. Подібно до інших фреймворків, застосунок, розроблений на React Native, також отримував відповідь зі статусом «OK».

2.4 Висновки по розділу

У другому розділі було досліджено методи та інструменти розробки вебдодатків із використанням сучасних фреймворків. Проведено аналіз особливостей реалізації веб-застосунків на основі React.js, Ruby on Rails, Vue.js та Laravel, визначено їх сильні та слабкі сторони у контексті швидкості розробки. Встановлено, що сучасні фреймворки забезпечують високий рівень повторного використання компонентів, спрощують організацію архітектури застосунків та прискорюють процес створення функціональності. Крім того, було визначено, що вибір фреймворку залежить від вимог проєкту, складності системи, досвіду команди та потреб у масштабуванні програмного забезпечення.

-РОЗДІЛ 3. ОЦІНКА ВПЛИВУ ФРЕЙМВОРКІВ НА ШВИДКІСТЬ РОЗРОБКИ

3.1 Модель порівняння ефективності фреймворків

Оскільки кожен проект веб-розробки має різні вимоги порівняно з іншими, нелегко визначити правильне порівняння під час вибору веб-фреймворку. Тому для цього дослідження було зроблено спробу побудувати відповідну модель, яка намагається врахувати деякі важливі аспекти фреймворків.

Цей підрозділ стосується порівняння всіх фреймворків, які використовувалися для реалізації чат-застосунків. Результати будуть обговорені в наступному розділі. Усі веб- та нативні додатки було порівняно відповідно до наступних бенчмарків. Ці параметри були враховані з відомого бенчмаркінгу, проведеного TechEmpower.

Розробка та легкість модифікації

Дуже важливо враховувати загальний час, витрачений на розробку, та легкість модифікації додатків [31]. Опитування «веб-розробки» надає чіткі контрольні показники для всіх фреймворків. Ці показники включають створення, налагодження та тестування проекту.

Легкість розгортання

Рисунок 3.1 - Критерії розгортання застосунку

Це творіння описує легкість, з якою застосунок можна розгорнути на сервері (включаючи фронтенд та серверну частини). Цей бенчмарк залежить від компонентів, логіки та середовища бази даних застосунку. Деякі фреймворки потребують мало роботи для розгортання, тоді як інші можуть зайняти багато часу, тому важливо враховувати це під час порівняння фреймворків (Рисунок 3.1).

Згенерована HTML-структура

Обсяг згенерованого HTML-коду відображає гнучкість та швидкість фреймворків. Великий обсяг згенерованої HTML-структури означає, що застосунок повинен відображати великий обсяг даних та структури, що знижує продуктивність. Отже, обсяг згенерованого HTML-коду веб-застосунком обернено пропорційний часу, витраченому на очікування завантаження веб-сторінки.

Згенерована HTML-структура застосунку зазвичай містить різні JavaScript-кодів та метадані. Ці значення унікальні для кожного фреймворку; тому розмір згенерованого HTML-коду різниться. Розмір структури вимірюється в кілобайтах (КБ) на сторінку.

Продуктивність фреймворку

Цей критерій описує продуктивність фреймворку та його інтерфейсу командного рядка (CLI). Продуктивність включає час запуску, час запиту та відповіді, а також час ініціалізації WebSocket програми. Чим вищий час роботи фреймворку, тим нижчий його показник продуктивності. Якщо продуктивність фреймворку низька, то програми, побудовані на цих фреймворках, працюють повільніше. Оскільки програми не працюють добре при стабільному з'єднанні, ресурси браузера витрачаються даремно, і завантаження веб-сторінок займає багато часу [32].

Відповідає сучасним стандартам

Дуже важливо, щоб вебсайти відповідали сучасним стандартам розробки вебсайтів. Ці стандарти включають стандарти кодування HTML 5, CSS 3 та JavaScript (стандарт кодування ES6). Сучасні браузери підтримують ці стандарти. Тому ці фреймворки також повинні дотримуватися цих стандартів для кращої сумісності.

Якщо стандарти розробки не дотримуються належним чином, це призведе до слабкої згенерованої структури HTML, що, у свою чергу, уповільнить роботу програми та, отже, знизить продуктивність.

Усі розглянуті критерії бенчмарків відсортовані за пріоритетом. Пріоритети встановлені відповідно до поняття попередньо визначених концепцій [33]. Усі бенчмарки в сумі дають 100 балів. Що стосується технологій веб-розробки, два основні критерії, такі як «Розробка та модифікація» та «Продуктивність фреймворку», мають пріоритет понад усі інші, кожен з яких має 45 та 30 балів відповідно. Повні критерії бенчмарків визначені в таблиці 3.1

Таблиця 3.1

Структура – критерії порівняння (у відсотках).

Критерій вимірювання (Measurement Criteria)	**Виміряна важливість (у відсотках)**
Розробка та легкість модифікації (Development and Ease of Modification)	45%
Продуктивність фреймворку (Framework Performance)	30%
Легкість розгортання та впровадження (Ease of Deployment and Implementation)	15%
Відповідність сучасним стандартам (Corresponding to today's Standards)	10%

Фреймворки Angular JS та Flask. Безсерверні платформи та екосистема хмарних сервісів. Аналіз мережі. Мережевий аналіз цього фреймворку включає відстеження запитів/відповідей HTTP, TCP та WebSocket.

Відстеження HTTP-пакетів : На рисунку 48 показано кількість HTTP-пакетів, надісланих та отриманих протягом інтервалу 100 секунд. Великий обсяг потоку HTTP-трафіку призводить до постійного зв'язку між клієнтом(ами) та сервером. Це означає, що клієнт повинен постійно відображати HTML та JavaScript, не розриваючи з'єднання. Загальна кількість HTTP-пакетів, отриманих на 100-й секунді, становить близько 1630.

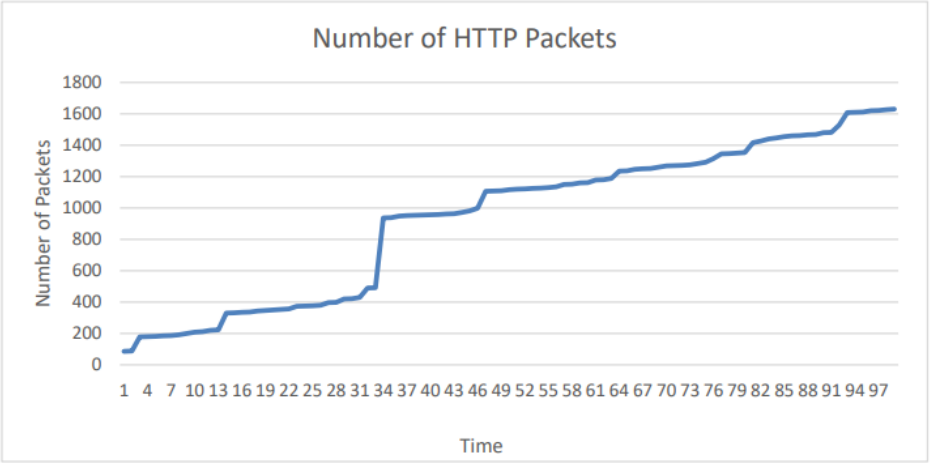


Рисунок 3.2 - Номери HTTP-пакетів Angular-Flask

На рисунку 3.3 показано довжину HTTP-пакетів, надісланих та отриманих за заданий проміжок часу (100 секунд). Довжина HTTP-пакетів вимірюється в байтах. Можна зробити висновок, що розмір відображених HTML-сторінок досить великий, оскільки максимальна довжина записаного HTTP-пакета становить близько 61006.

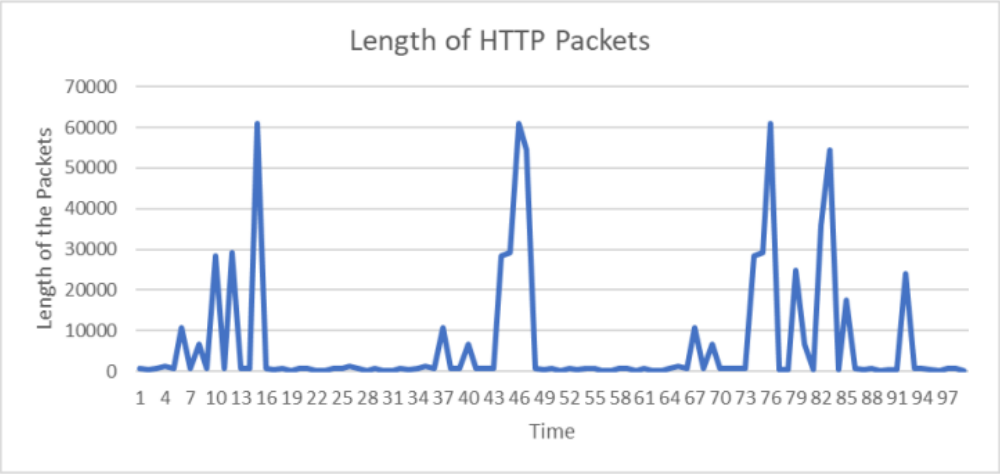


Рисунок 3.3 - Довжина HTTP-пакета Angular-Flask.

Відстеження TCP-пакетів: Цифра 50 показує кількість надісланих та отриманих TCP-пакетів протягом 100 секунд. Велика кількість отриманих TCP-пакетів свідчить про слабку мережеву структуру фреймворку. Максимальна

кількість зареєстрованих TCP-пакетів становить 540.

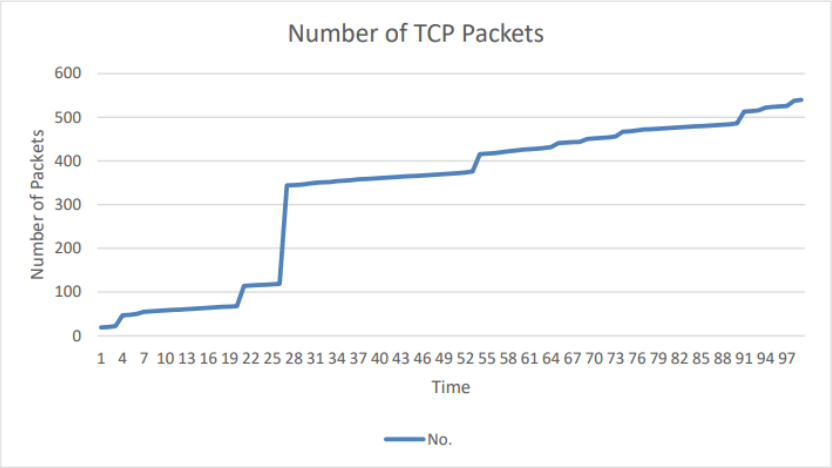


Рисунок 3.4 - Angular-Flask – Кількість TCP-пакетів.

На рисунку 3.5 зображено довжину TCP-пакетів (виміряну в байтах), надісланих та отриманих до 100-ї секунди. Довжина TCP-пакетів відображає потік даних у програмі. Постійне зменшення масштабування TCP-пакетів призводить до стабільного трафіку [34]. Максимальна довжина TCP-пакету, досягнута в цій програмі, становить 253 байти.

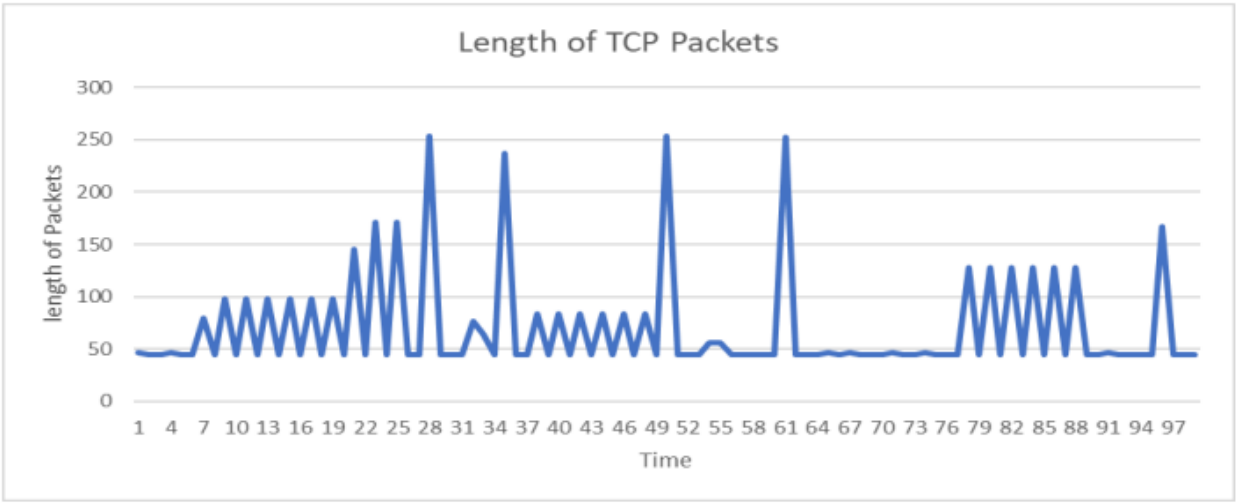


Рисунок 3.5 - Angular-Flask - Довжина TCP-пакетів

Спостерігається висока варіація TCP-потоку в діапазоні від 1 до 25

секунд.

Трасування WebSocket: WebSocket має два стани: відкритий або закритий. Крім того, він дотримується концепції маскування. Маскований WebSocket є безпечним, що означає, що жодні дані не можуть бути викрадені, тобто задана комбінація IP-адреси та порту маскується.

На рисунку 3.6 показано кількість даних WebSocket, надісланих та отриманих протягом 100 секунд. Обсяг даних, надісланих та отриманих протягом заданого періоду часу, вказує на затримку, яку витрачає сервер застосунків. Обсяг даних WebSocket обернено пропорційний нестабільності з'єднання. Для розглянутого застосунку обсяг даних WebSocket становить 2590.

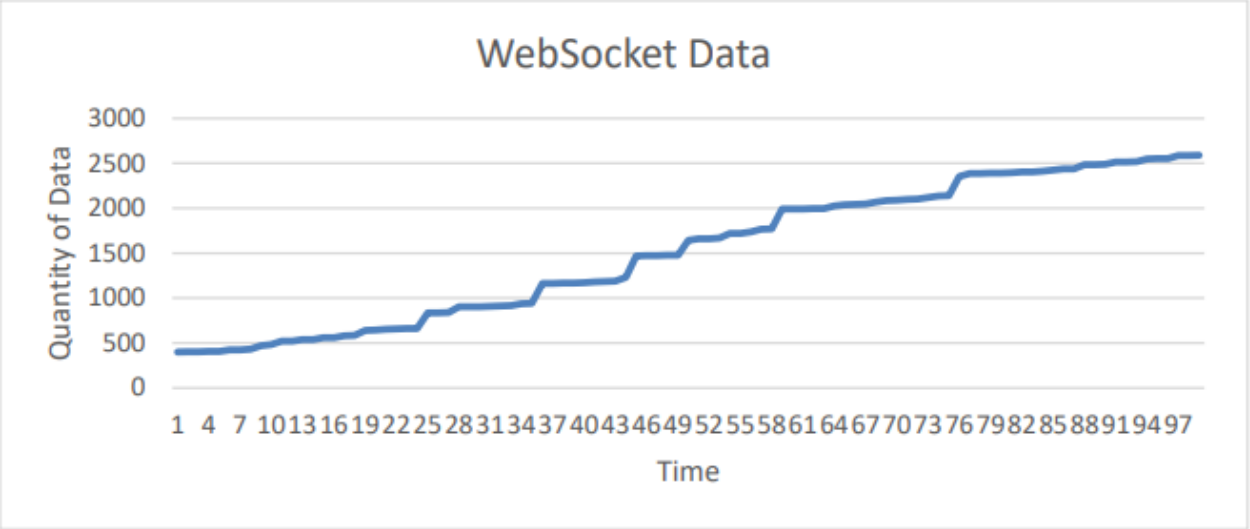


Рисунок 3.6 - Кількість даних Angular-Flask WebSocket

На рисунку 3.7 зображено довжину даних WebSocket, надісланих та отриманих до 100-ї секунди. Довжина даних WebSocket приблизно однакова для всіх програм. Повідомлення, надіслані в усіх програмах, однакові, тому довжина надісланих та отриманих даних WebSocket приблизно однакова. Для цієї програми середня довжина даних становить 111 байт.

На рисунку 3.8 показано чергу та час зупинки для початкового WebSocket-з'єднання [35]. Продуктивність фреймворку падає зі збільшенням WebSocket-з'єднання.

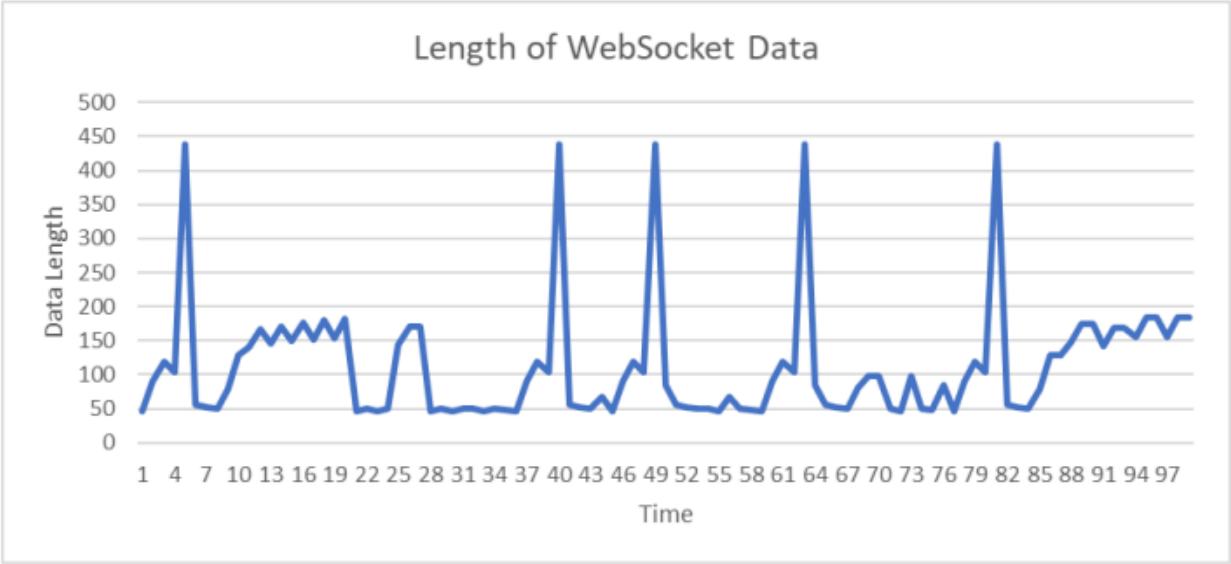


Рисунок 3.7 - Довжина даних Angular-Flask WebSocket

Час збільшується. Час очікування в черзі для з'єднання WebSocket у цій програмі становить 5,47 мілісекунд, тоді як час зупинки становить 0,20 мілісекунди.

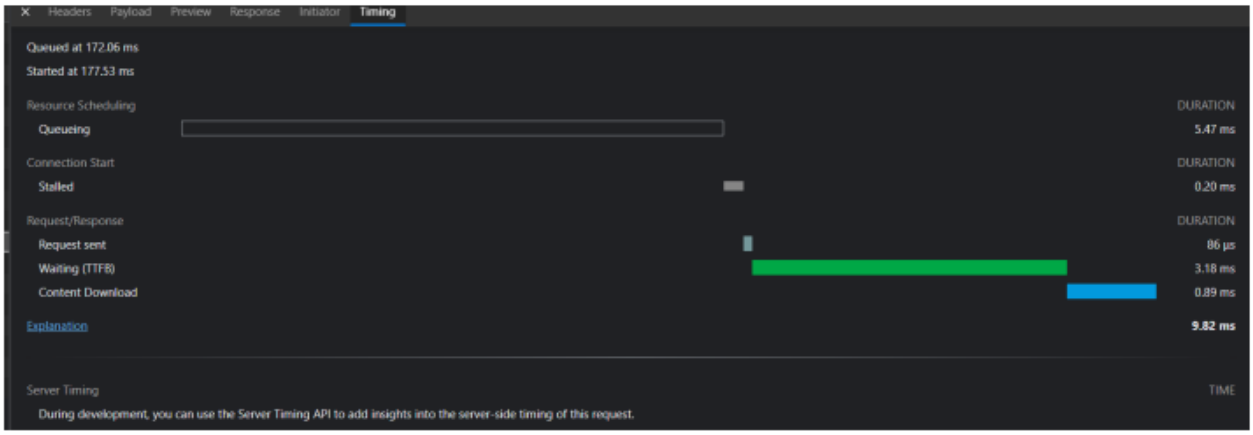


Рисунок 3.8 - Час зупинки Angular-Flask WebSocket.

На рисунку 3.9 показано час очікування запиту, доставленого через WebSocket-з'єднання. Час очікування описується як час, необхідний WebSocket-з'єднанню для запиту першого байта даних, або час до першого байта (TTFB). Подібно до часу зупинки, час очікування також обернено пропорційний

продуктивності фреймворку. Час очікування для цієї програми становить 562 мілісекунди.

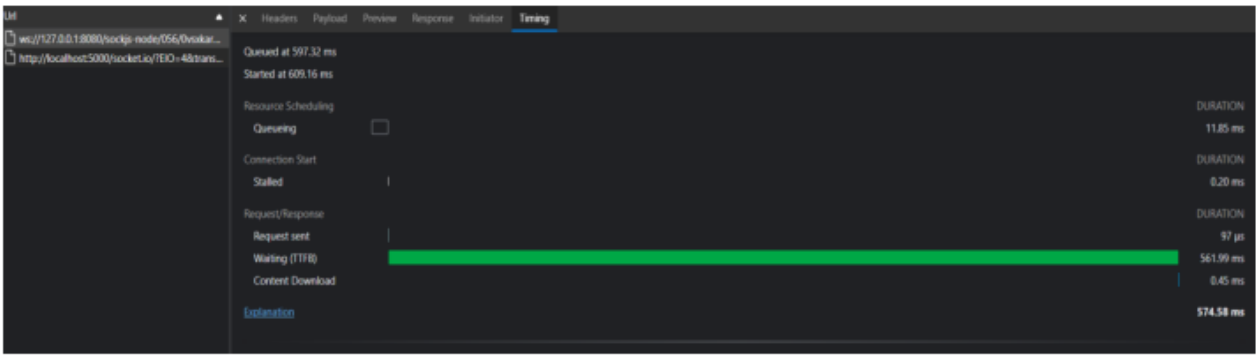


Рисунок 3.9 - TTFB для Angular-Flask WebSocket.

3.2 Аналіз продуктивності та швидкості розробки веб-додатків

Продуктивність програми тестується за допомогою «Елемента перевірки» браузера [36]. Браузер допомагає у створенні звіту Lighthouse, який аналізує застосунок. Аналіз продуктивності включає індекс швидкості, час взаємодії, час блокування та зміщення макета застосунку (Рисунок 3.10).

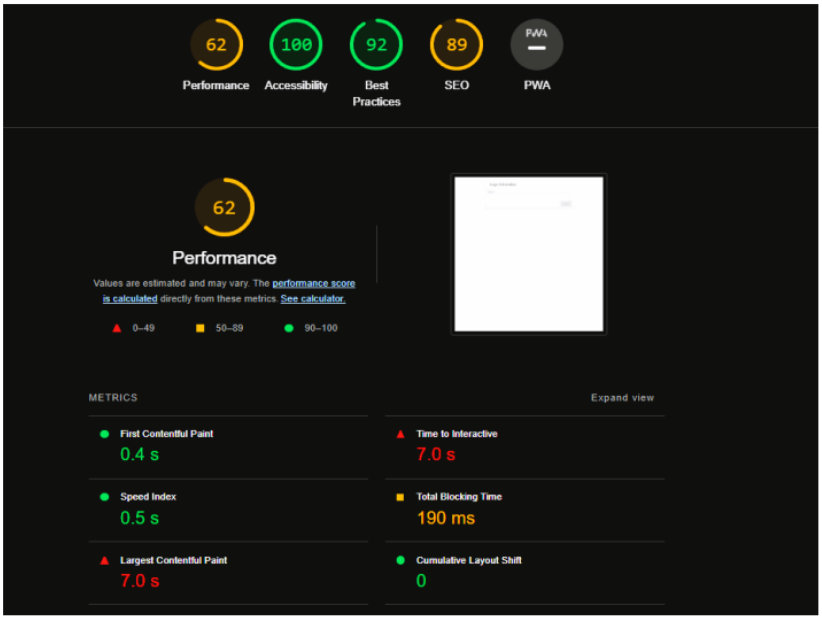


Рисунок 3.10 - Продуктивність застосунку Angular-Flask.

Аналіз мережі React JS та Rails. Як і в інших застосунках, мережевий аналіз цього фреймворку включає аналіз HTTP, TCP та WebSocket запитів і відповідей, які надсилаються та отримуються сервером.

Відстеження HTTP-пакетів: На рисунку 3.11 показано кількість HTTP-пакетів, які були надіслані та отримані протягом 100 секунд. Кількість HTTP-пакетів у цьому випадку становить 3109.

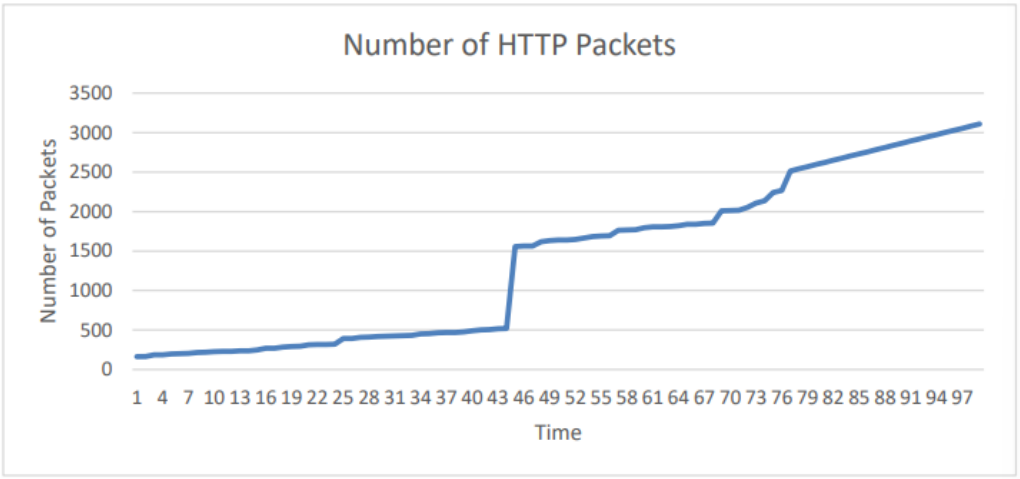


Рисунок 3.11 Кількість HTTP-пакетів у React-Rails

На рисунку 3.12 показано довжину HTTP-пакетів, надісланих та отриманих за 100 секунд. Довжина HTTP-пакетів вимірюється в байтах. У цьому випадку максимальна довжина HTTP-пакета становить близько 946 байт.

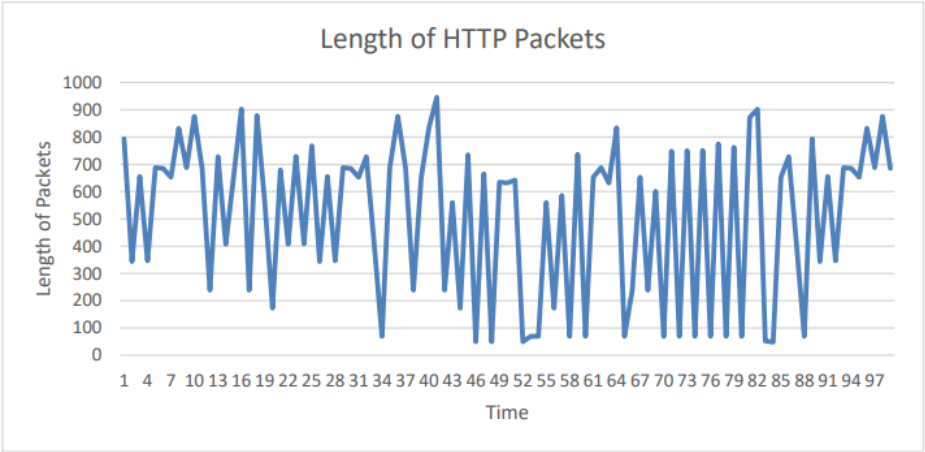


Рисунок 3.12 - Довжина HTTP-пакетів у React-Rails.

Відстеження пакетів TCP: На рисунку 3.13 показано кількість пакетів TCP, надісланих та отриманих протягом 100 секунд. Максимальна кількість пакетів TCP, зафіксована для даного додатка, становить 1780.

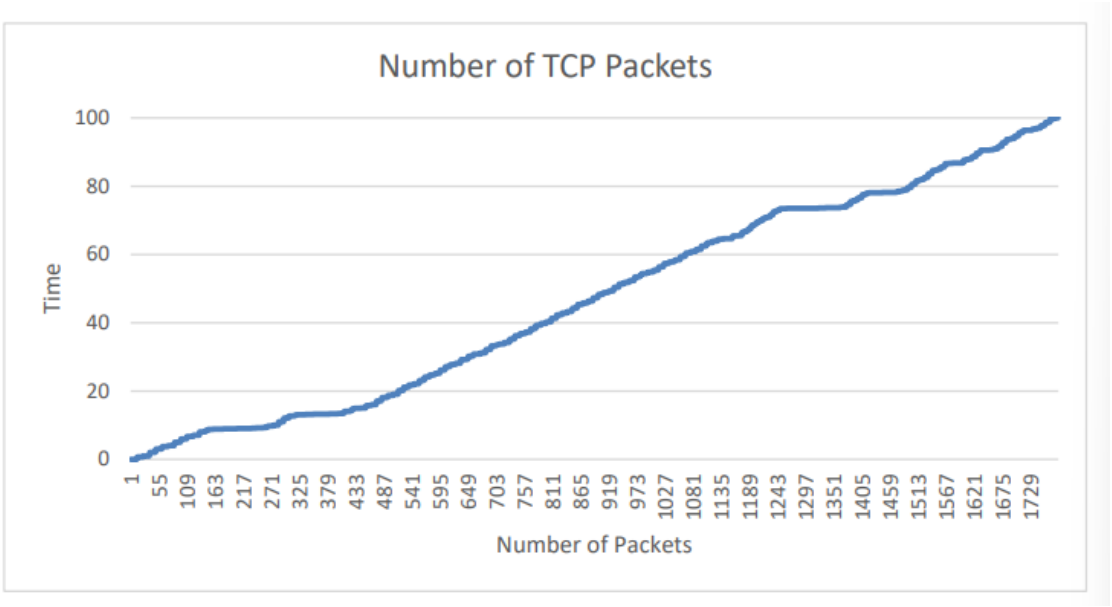


Рисунок 3.13 - Кількість TCP-пакетів у React-Rails.

На рисунку 3.14 описує довжину TCP-пакетів (виміряну в байтах), надісланих та отриманих до 100-ї секунди. Максимальна довжина TCP-пакету, досягнута в цьому застосуванні, становить 102 байти.

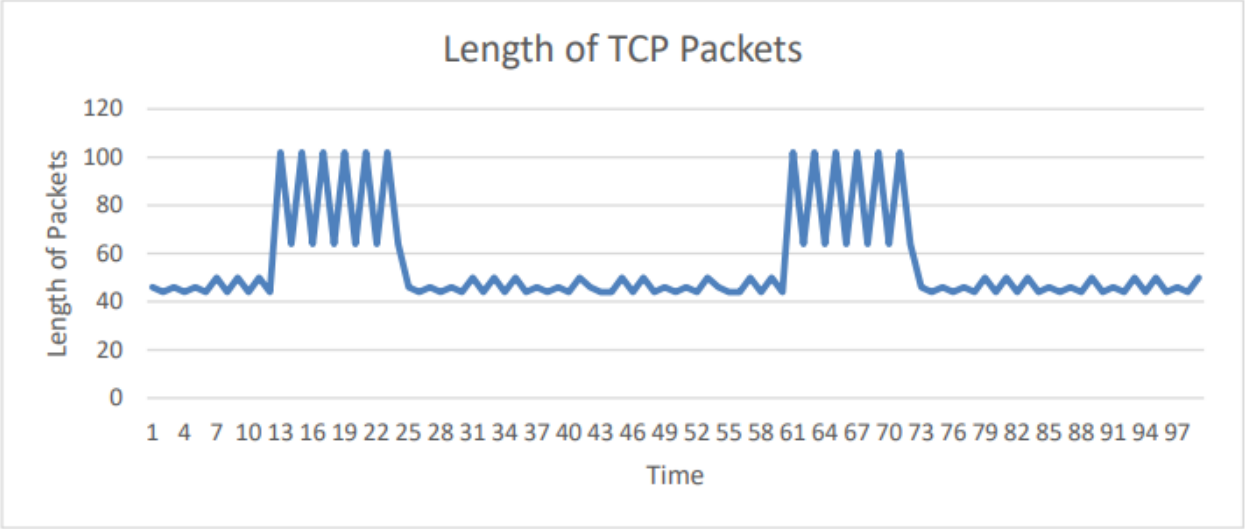


Рисунок 3.14 - Довжина TCP-пакетів у React-Rails.

Відстеження WebSocket: На рисунку 3.16 показано кількість даних WebSocket, надісланих та отриманих протягом 100 секунд. Для розглянутого додатка обсяг відстежених даних WebSocket становить 910.

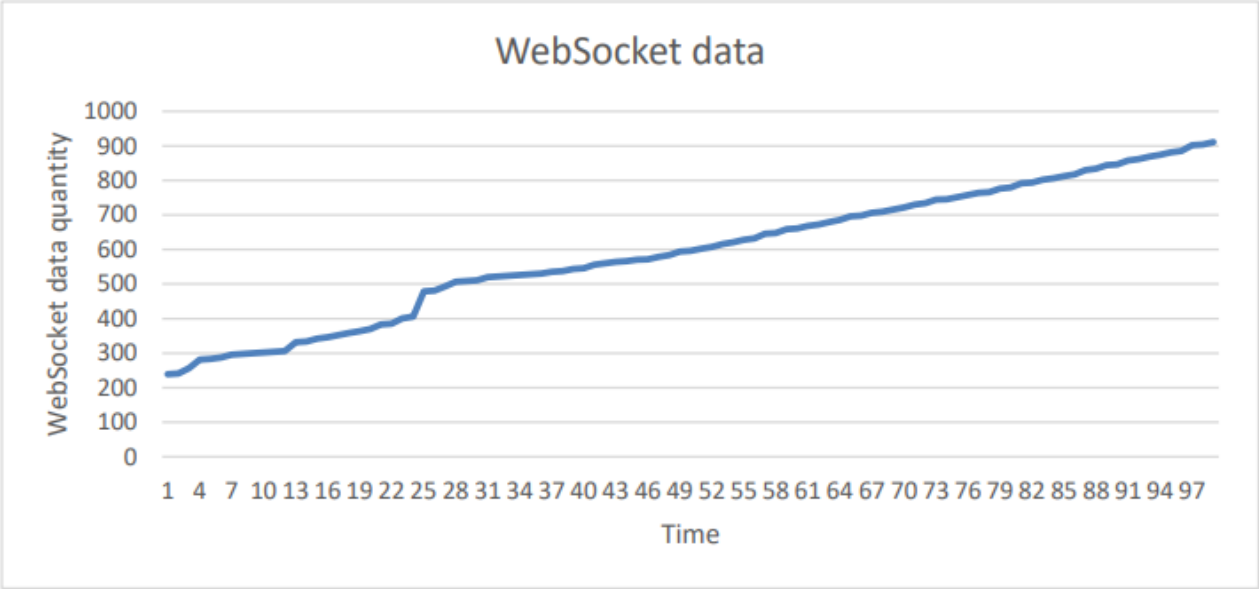


Рисунок 3.16 - Кількість даних WebSocket у React-Rails

У таблиці на рисунку описано довжину даних WebSocket, надісланих та отриманих до 100-ї секунди. Для цієї програми середня довжина даних становить 109,7 байт.

Аналіз продуктивності

Продуктивність цієї програми також вимірюється за допомогою «Inspect Element» браузера. Для аналізу програми створюється звіт Lighthouse. Аналіз включає індекс швидкості, час взаємодії, час блокування та зміщення макета програми (Рисунок 3.17).

Подібно до інших веб-фреймворків, цей аналіз продуктивності також виконується за допомогою «Елемента перевірки» браузера. Згенерований звіт Lighthouse включає індекс швидкості, час взаємодії, час блокування та зміщення макета застосунку. (Рисунок 3.18)

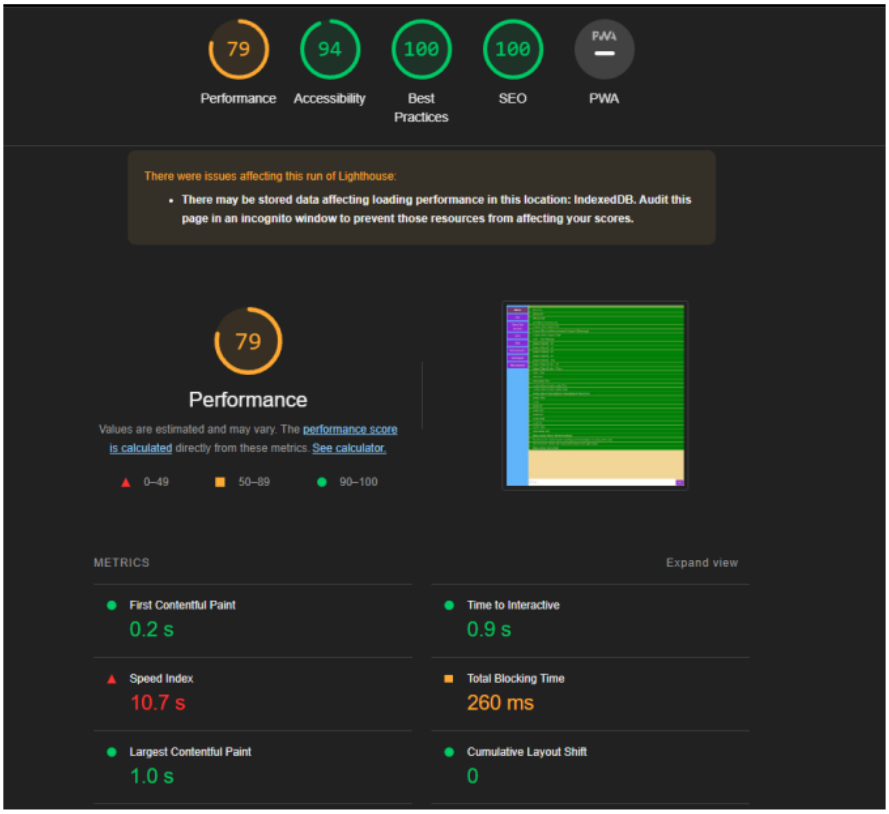


Рисунок 3.17 - Продуктивність застосунку React-Rails Chat.

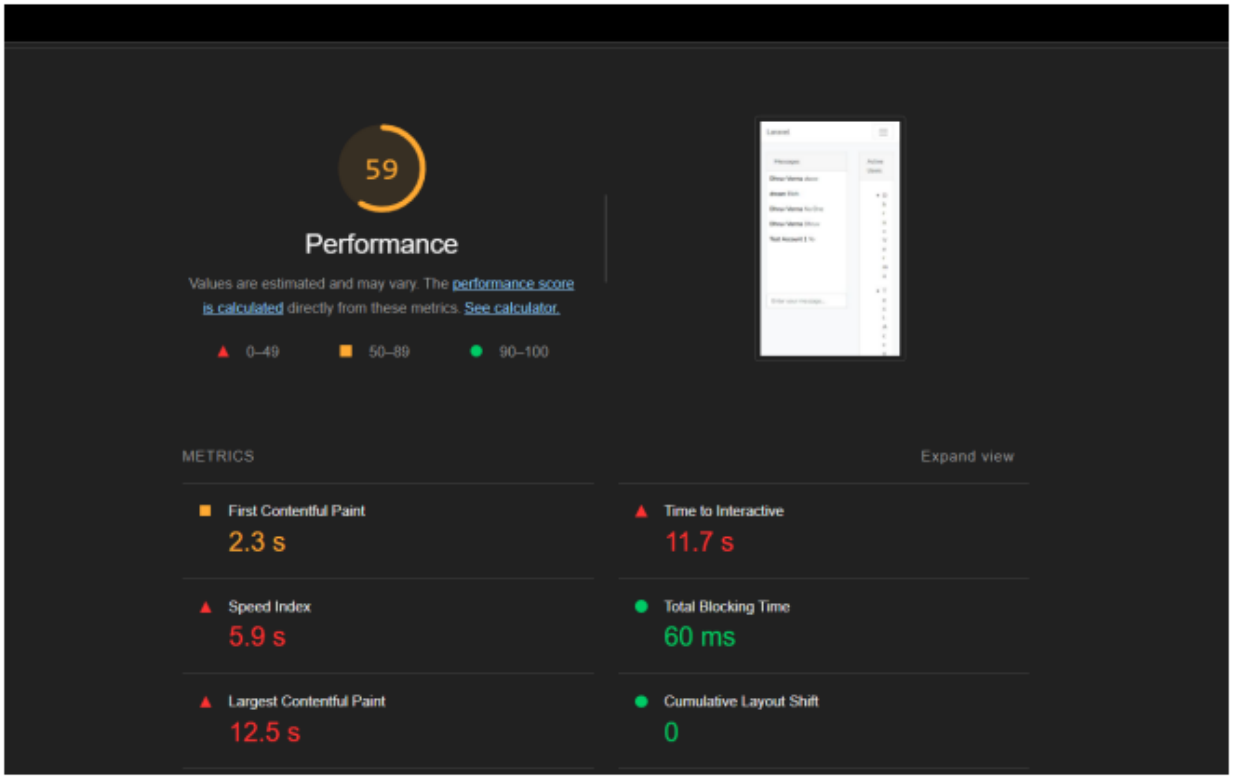


Рисунок 3.18 - Продуктивність застосунку Vue-Laravel Chat.

Аналіз застосунків, розроблених на React-Native та Express, набагато простіший порівняно з аналізом застосунків на Swift. Native Framework надає можливість запускати застосунок у браузері. Браузер потім може протестувати продуктивність за стосунку [37]. Згенерований звіт містить значення індексу швидкості, часу інтерактивності, часу блокування та зсуву макета застосунку. (Рисунок 3.19)

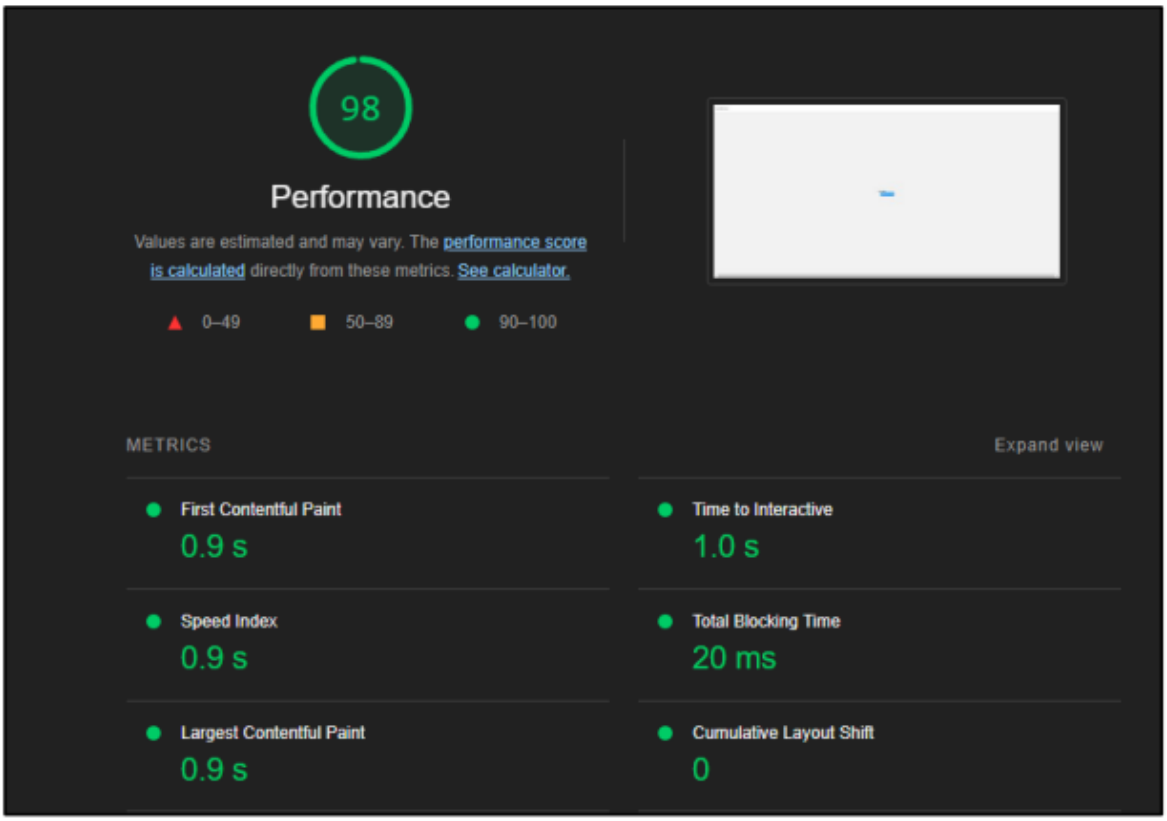


Рисунок 3.19 - React-Native-Express Продуктивність чат-застосунку.

3.3 Результати дослідження та їх інтерпретація

Результати порівняльного аналізу різних фреймворків веб-розробки представлені в наступних розділах.

Розглядаються фреймворки

Таблиця 3.2

Веб-фреймворки для чат-застосунку.

№	Front-End (Клієнтська частина)	Back-End (Серверна частина)
1	Angular JS	Flask
2	React JS	Rails
3	Vue JS	Laravel

У таблиці 3.2 наведено комбінацію фреймворків, які були використані під час реалізації веб-застосунку.

Таблиця 3.3

Нативні фреймворки для чат-застосунку.

№	Front-End (Мобільний інтерфейс)	Back-End (Серверна частина)
1	Swift	Swift
2	React Native	Express

У таблиці 3.3 наведено комбінацію фреймворків, які були використані під час реалізації Native-Application.

Для реалізації чат-додатку використовуються фронтенд-фреймворк Angular JS та бекенд-фреймворк Flask. На відміну від інших популярних фронтенд-фреймворків, Angular підтримує впровадження залежностей, що важливо під час роботи з фреймворком Flask. Більше того, Angular досить ефективно обробляє складні односторінкові додатки, що безпосередньо відповідає здатності Flask обробляти кілька маршрутів і водночас забезпечувати легкий та швидкий додаток. Flask також дозволяє кодувати додаток в об'єктно-орієнтованому стилі, що відповідає об'єктно-орієнтованому підходу Angular.

Поєднання React та Rails забезпечує доступ до кількох вбудованих бібліотек коду, що скорочує час та зусилля розробки. Додаток, створений за допомогою цього поєднання, вирізняється стабільністю та якістю, як показано в таблиці 3.3. На відміну від інших комбінацій фреймворків, легкий React та складний Rails працюють разом, утворюючи додаток, який використовує менше ресурсів пам'яті

та підвищує продуктивність.

Laravel та Vue йдуть рука об руку. Vue можна описати як мінімалістичний фронтенд-фреймворк, який чудово працює в односторінкових інтерфейсах користувача, тоді як Laravel покращує продуктивність Vue. Вбудовані бібліотеки для розробки на Vue у фреймворку Laravel роблять їх гарним поєднанням. Стеки Laravel Vue дозволяють розробнику ефективно створювати односторінкові програми з безшовним фронтендом.

Фреймворк Swift використовується виключно для розробки додатків для iOS, оскільки не існує комбінацій різних фреймворків, які можна використовувати для розробки додатків для iOS. iOS дотримується суворої політики та не дозволяє запускати сторонні додатки на платформі.

Як і React, React-Native також базується на односторінковому застосунку (SPA). Це означає, що до мобільного застосунку можна отримати доступ з однієї нативної сторінки. Ця функціональність дозволяє уникнути завантаження нової сторінки з кожною дією, тим самим забезпечуючи спрощений користувацький досвід. Express, з іншого боку, створює сервер, який відповідає функціональності React-Native, і це поєднання створює дуже швидкий і стабільний застосунок.

Результати фреймворків веб-застосунків.Аналіз бенчмарків

На рисунку 3.20 описано оцінки, отримані відповідними вебфреймворками, залежно від критеріїв. Кожна комбінація фреймворків оцінюється відповідно до їхньої продуктивності та вимірювань.

Розробка та легкість модифікації

З опитування видно, що комбінація фреймворків React та Rails отримала найвищий бал, тоді як комбінація Vue та Laravel – найнижчий. Бал базується на розробці, налагодженні, модифікації та тестуванні застосунку.

Продуктивність

Також очевидно, що комбінація фреймворків React та Rails перемагає в цій категорії, тоді як комбінація Vue та Laravel набрала найменше балів [38]. Продуктивність базується на згенерованому звіті маяка та часі зупинки, який витрачається на пакети даних програми. На рисунках показано згенерований звіт

маяка та час зупинки для кожного фреймворку відповідно.

Легкість розгортання

За цим критерієм комбінація фреймворків Angular та Flask збігається з критерієм React та Rails. Цей критерій є суб'єктивним, проте важливим аргументом і вимірюється часом і зусиллями, витраченими на розгортання застосунку на сервері. Для порівняння, фреймворки Flask та Rails набагато легше розгортати, тоді як Laravel вимагає величезних зусиль.

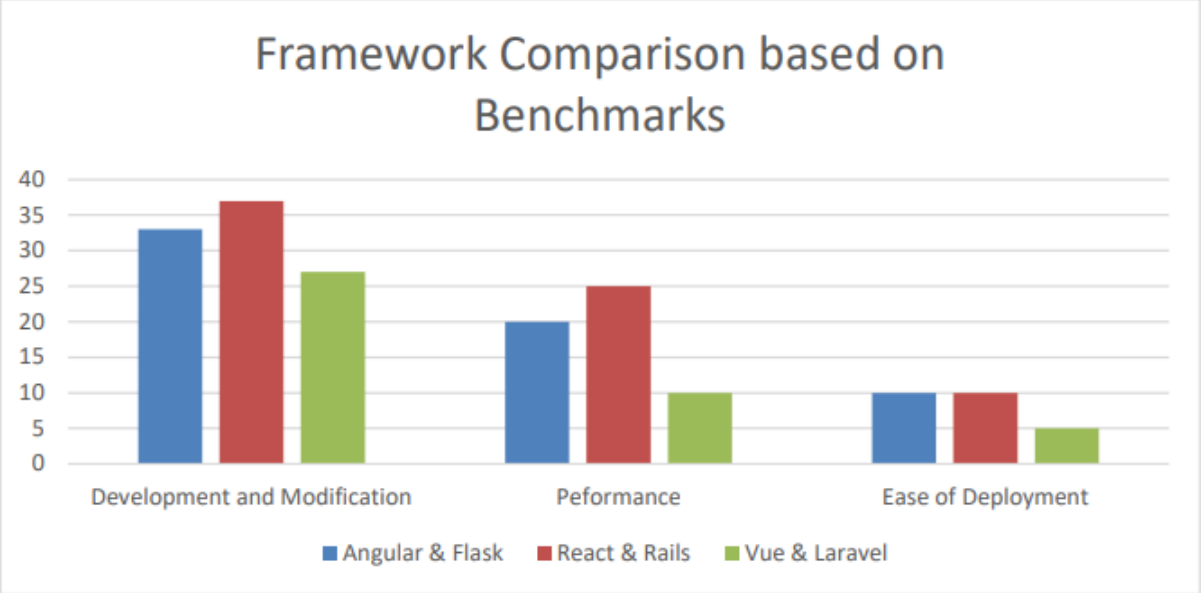


Рисунок 3.20 - Веб-фреймворк порівняння бенчмарків.

З наведеної тенденції також видно, що комбінація фреймворків Angular та Flask стабільно залишалася на другій позиції, за винятком критерію «Легкість розгортання», де вона зрівнялася з комбінацією фреймворків React та Rails.

Аналіз мережі

У цьому розділі описано комбінований аналіз відповідних веб-фреймворків залежно від протоколів. Протоколи, що використовуються для аналізу кожної програми, це HTTP, TCP та WebSocket. Кожен із фреймворків оцінюється відповідно до обсягу генерованого трафіку.

HTTP

На рисунку 3.21 показано кількість (кількість та довжину) пакетів, згенерованих HTTP-потоком кожної програми, реалізованої з використанням відповідних фреймворків. Видно, що комбінація фреймворків React та Rails генерує менше трафіку порівняно з двома іншими [39]. Це означає, що цей фреймворк не вимагає постійних запитів для генерації HTML-структур. Отже, він працює краще, ніж інші фреймворки.

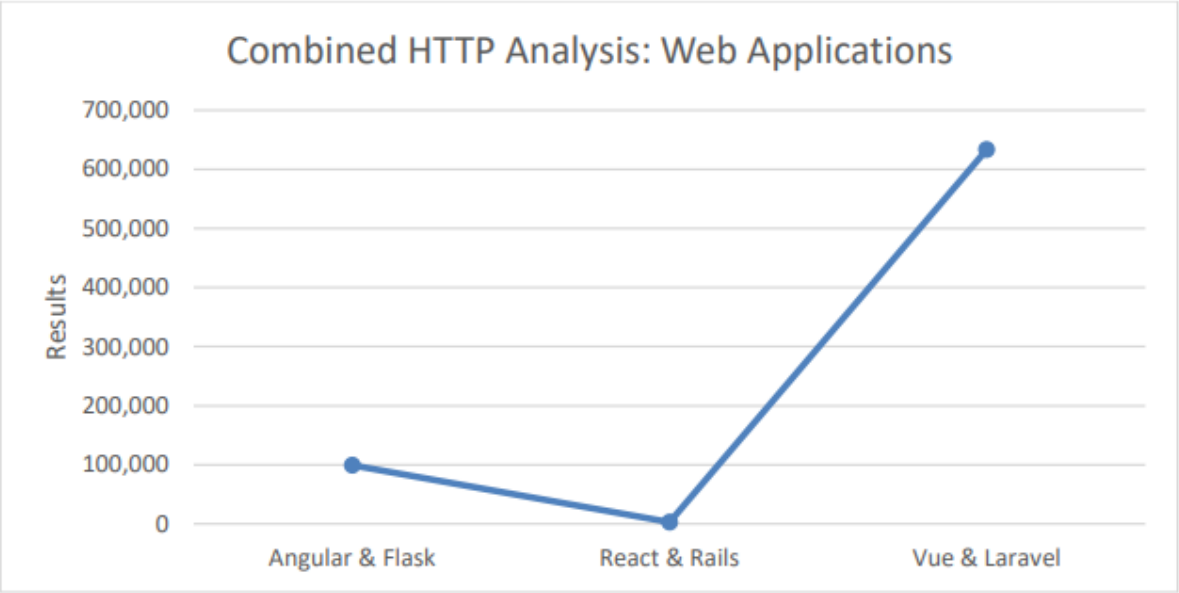


Рисунок 3.21 - Веб-застосунки аналіз HTTP.

На рисунку 3.22 показано кількість (кількість та довжину) пакетів, згенерованих TCP-потоком кожної програми, реалізованої з використанням відповідних фреймворків. Помічено, що комбінація фреймворків Angular та Flask генерує найменший обсяг трафіку з усіх. Це означає, що цей фреймворк не вимагає постійного обміну пакетами, і тому він краще виконує цей критерій. Однак, слід також зазначити, що комбінація фреймворків React та Rails генерує порівнянний обсяг TCP-трафіку з комбінацією фреймворків Angular та Flask.

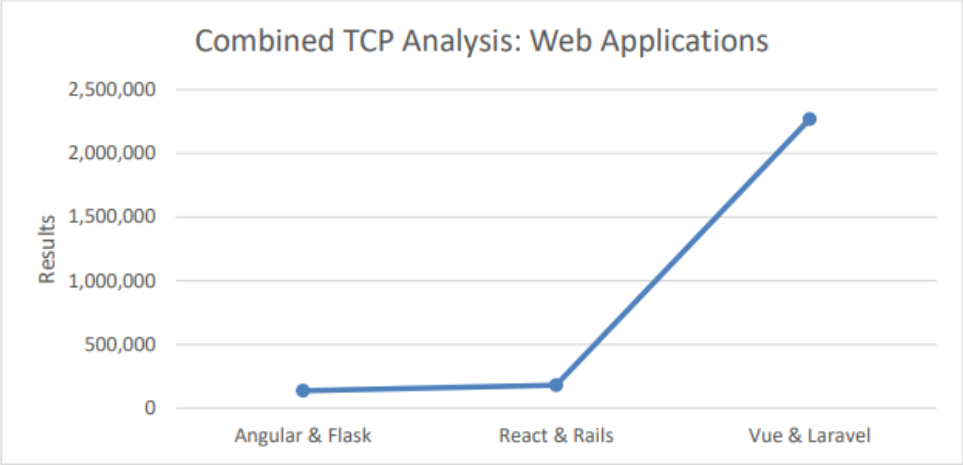


Рисунок 3.22 - Веб-застосунки аналіз TCP.

Вебсокет на рисунку 3.23 показано обсяг (кількість та довжину) пакетів, згенерованих потоком WebSocket кожного застосунку, реалізованого з використанням відповідних фреймворків. Не дивно, що комбінація фреймворків React та Rails генерує найменший обсяг трафіку з усіх. Слід також враховувати, що кожен застосунок отримав однаковий обсяг даних. Це означає, що фреймворк React та Rails не вимагає великої кількості пакетів, і тому він найкраще виконує цей критерій. Однак, слід також зазначити, що комбінація фреймворків Vue та Laravel генерує порівнянний обсяг трафіку WebSocket з комбінацією фреймворків Angular та Flask.

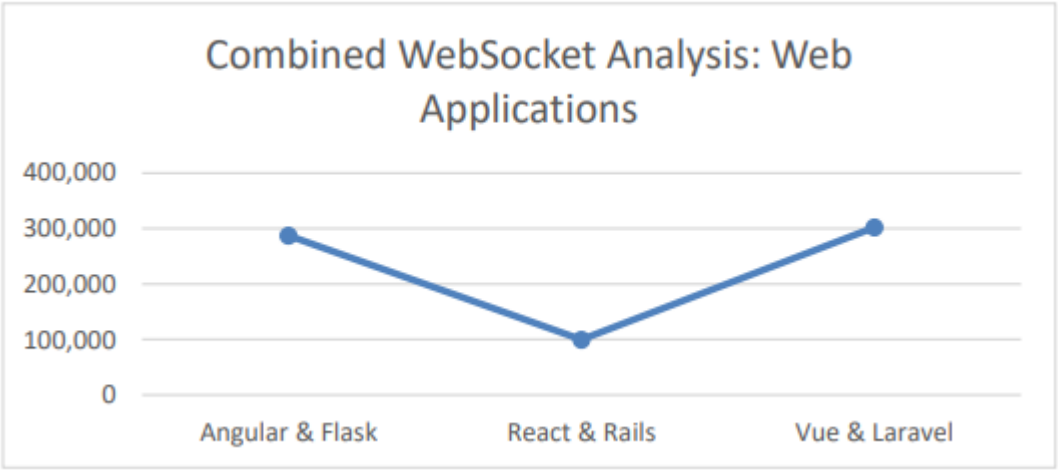


Рисунок 3.23 - Веб-застосунки аналіз WebSocket.

Результати цих мережевих аналізів показують, що комбінація фреймворків React та Rails вимагає найменшої кількості пакетів трафіку для запиту та відповіді на той самий обсяг даних порівняно з іншими комбінаціями фреймворків. Таким чином, комбінація фреймворків React та Rails перемагає інші. Результати фреймворків нативних за стосунків. Аналіз бенчмарків. На рисунку 3.24 описано контрольні бали, отримані відповідними нативними фреймворками залежно від критеріїв. Кожна з комбінацій фреймворків оцінюється відповідно до їхньої продуктивності та вимірювань.

Розробка та легкість модифікації. З опитування видно, що комбінація фреймворків React-Native та Express отримала вищий бал порівняно з фреймворком Swift.

Продуктивність .Також очевидно, що комбінація React-Native та Express фреймворку ледь перемагає в цій категорії. Продуктивність базується на часі, який потрібен застосунку для завантаження та функціонування. Обидва фреймворки витратили однаковий час під час аналізу, але комбінація React-Native та Express фреймворку отримала незначно вищий бал.

Легкість розгортання .За цим критерієм обидва розглянуті фреймворки набрали однакову кількість балів. Розгорнути обидва додатки досить просто. У обох випадках для вказання необхідного сервера та розгортання додатка достатньо одного рядка коду. Це не вимагає жодних зусиль чи зайвих витрат часу. Отже, ці фреймворки набрали однакову кількість балів.

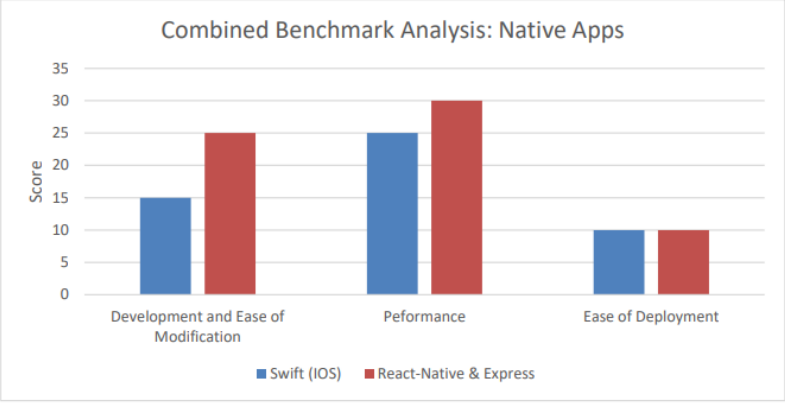


Рисунок 3.24 - Нативні програми аналіз показників.

Описано комбінований аналіз відповідних нативних фреймворків залежно від протоколів. Протоколи, що використовуються для аналізу кожної програми, це HTTP, TCP та WebSocket [40]. Кожен з фреймворків оцінюється відповідно до обсягу генерованого трафіку.

HTTP

На рисунку 3.25 показано кількість (кількість та довжину) пакетів, згенерованих HTTP-потоком кожного реалізованого застосунку. Зрозуміло, що застосунок на основі фреймворку Swift потребує найменшої кількості HTTP-трафіку для генерації інтерфейсу користувача та його компонентів.

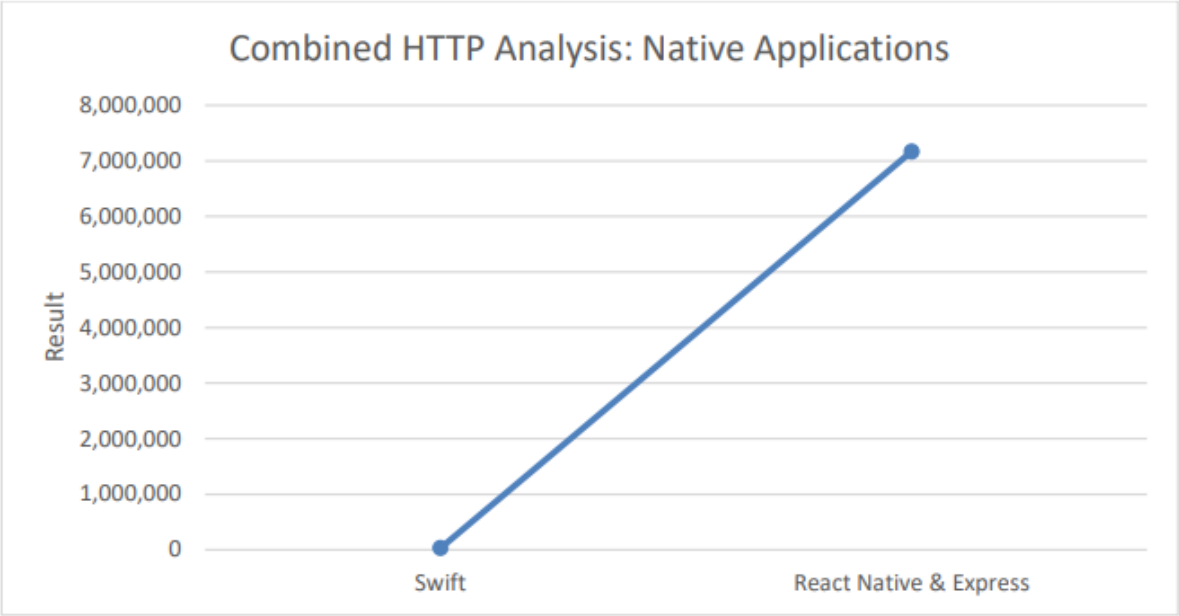


Рисунок 3.25 - Нативні програми аналіз HTTP.

TCP

На рисунку 3.26 показано обсяг (кількість та довжину) пакетів, згенерованих TCP-потоком кожної програми, реалізованої з використанням відповідних фреймворків. У цьому випадку фреймворк Swift також вимагає меншого обсягу трафіку порівняно з програмами React та Express. Отже, програма Swift отримує вищі бали за цим критерієм.

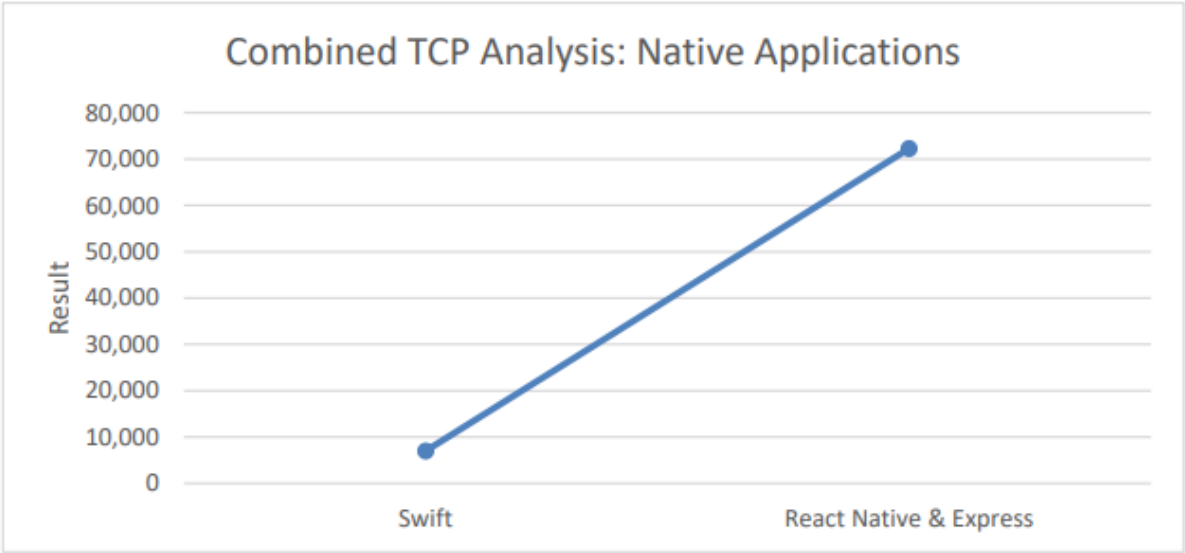


Рисунок 3.26 - Рідні програми аналіз TCP.

Вебсокет

На рисунку 3.27 показано обсяг (кількість та довжину) пакетів, згенерованих потоком WebSocket кожного застосунку, реалізованого з використанням відповідних фреймворків. Величезним сюрпризом є те, що комбінація фреймворків React Native та Express генерувала значно менше трафіку WebSocket порівняно з фреймворком Swift. Це означає, що застосунок React Native та Express має стабільніше з'єднання між своїм клієнтом та сервером.

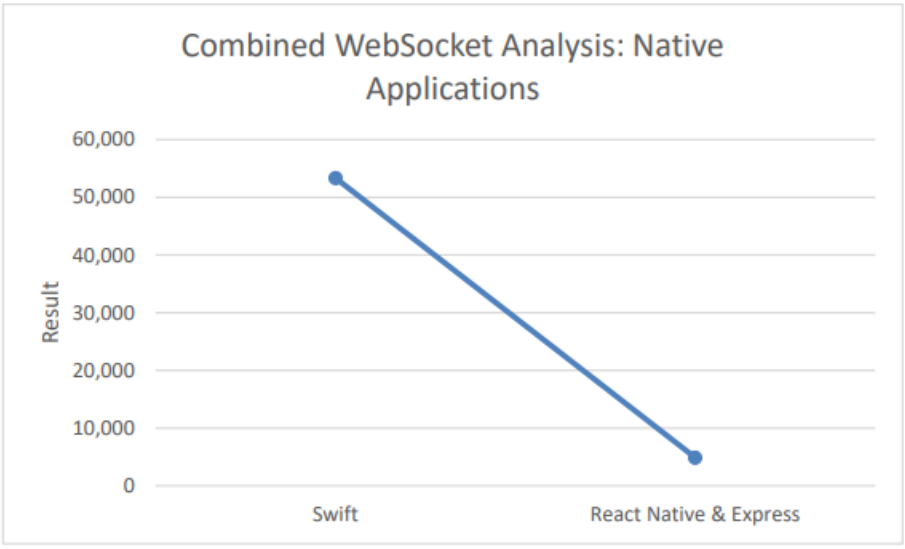


Рисунок 3.27 - Нативні застосунки аналіз WebSocket.

Результати цих мережевих аналізів показують, що комбінація фреймворку Swift вимагає найменшої кількості пакетів трафіку для створення інтерфейсу користувача та зв'язку із сервером. Однак, для роботи чат-додатку потрібен величезний обсяг трафіку.

З результатів можна зробити висновок, що застосунок, розроблений з використанням фреймворків React та Rails, працює краще порівняно з іншими вебфреймворками. Також для нативних застосунків фреймворк React-Native має кращі показники порівняно з фреймворком Swift. Основним недоліком Swift є те, що він доступний лише для платформ iOS, тоді як комбінація фреймворків React-Native та Express використовує кросплатформну функціональність.

Для розробленого чат-додатку було проведено бенчмарк-оцінювання. Не можна однозначно зробити висновок, що React та Rails працюватимуть краще порівняно з іншими фреймворками, використовуючи аналіз, оскільки використовувалися лише певні атрибути, пов'язані з критеріями бенчмарку. Інші критерії бенчмарку, такі як «підтримка плагінів», «міркування безпеки» та «підтримка AJAX», не використовувалися. Ці критерії не вписувалися в поточний чат-додаток, оскільки самі WebSocket забезпечують додатковий рівень безпеки сокетів під час розмови між клієнтом та сервером.

Аналогічно, для нативних та кросплатформних застосунків, не можна з упевненістю зробити висновок, що фреймворки React-Native та Express працюватимуть краще, ніж фреймворки Swift. Однак можна передбачити, що для певного проекту ці фреймворки можуть працювати краще, ніж інші.

Дослідження зіткнулося з різними труднощами та обмеженнями під час захоплення та аналізу мережі. Ці труднощі узагальнено в наступному розділі:

1. Першим завданням було ізолювати трафік на платформі Windows. Основною проблемою було подолання затримки SSDP під час передачі пакетів.
2. Політику CORS доводилося досить часто скидати під час аналізу програми. Це обмежувало роботу програми на різних серверах, що виявилося величезною перешкодою.
3. Тестування iOS-додатків було ще одним викликом, який виявився

досить складним. Оскільки Apple не дозволяє використовувати сторонні додатки, продуктивність iOS-додатку вимірювалася вручну за допомогою мережевого набору даних.

4. Деякі атрибути критеріїв порівняння не згадуються для відповідного чат-застосунку. Ці критерії залежать від середовища застосунку, а це означає, що деякі з цих критеріїв добре підійдуть для різних типів застосунків.

5. Трансляція WebSocket за допомогою Laravel пройшла не дуже гладко. Виправлення помилки зайняло щонайменше 7 годин через відсутність належної документації.

6. Розглянутий додаток для чату було реалізовано через обмежений обсяг набору даних. Тому критерії бенчмарку можуть показувати різні результати під час тестування у набагато більшому масштабі.

Програму захоплення мережевих пакетів на пристрої One Plus 8T довелося вручну зупиняти, а потім перезапускати кілька разів після того, як VPN-сервіси забезпечили стабільне з’єднання.

3.4 Висновки по розділу

У третьому розділі виконано оцінювання впливу різних фреймворків на швидкість розробки веб-додатків. Розроблено модель порівняння ефективності технологічних рішень та проведено аналіз продуктивності процесу створення програмного забезпечення з використанням різних підходів. Отримані результати підтвердили, що використання сучасних фреймворків дозволяє значно скоротити час розробки, підвищити якість коду та спростити підтримку програмних систем. Також встановлено, що ефективність використання фреймворків залежить не лише від технічних характеристик технології, а й від рівня підготовки команди, архітектурних рішень та специфіки поставлених задач.

ВИСНОВКИ

У ході виконання дипломної роботи було досліджено вплив вибору фреймворків на швидкість створення веб-додатків, що є важливим аспектом сучасної веб-розробки. Робота охоплює комплексний аналіз підходів до використання фреймворків, їх функціональних можливостей та ефективності в контексті скорочення часу розробки програмного забезпечення. У процесі дослідження було розглянуто сучасні технології веб-розробки, визначено їх переваги та недоліки, а також реалізовано тестовий веб-додаток, що дозволило на практиці оцінити їх вплив на швидкість створення програмних продуктів.

На початковому етапі було проведено аналіз теоретичних основ використання фреймворків, визначено їх роль у побудові архітектури веб-додатків та вплив на продуктивність розробки. Було встановлено, що фреймворки значно спрощують процес створення програмного забезпечення за рахунок повторного використання коду, наявності готових компонентів та стандартизованих підходів до розробки. Водночас, вибір конкретного фреймворку залежить від вимог проєкту, досвіду команди та специфіки завдань.

У практичній частині роботи було реалізовано веб-додаток (чат-систему) з використанням різних фреймворків, зокрема React, Vue, Laravel та Ruby on Rails. Це дозволило оцінити їх з точки зору швидкості розробки, зручності використання, гнучкості та можливостей масштабування. Було встановлено, що фронтенд-фреймворки, такі як React і Vue, забезпечують високу швидкість розробки інтерфейсів завдяки компонентному підходу, тоді як бекенд-фреймворки, такі як Laravel і Ruby on Rails, значно спрощують реалізацію серверної логіки за рахунок вбудованих інструментів і шаблонів.

Особливу увагу було приділено порівняльному аналізу фреймворків, у рамках якого розроблено модель оцінювання їх ефективності. Аналіз продуктивності показав, що швидкість створення веб-додатків залежить не лише від обраного фреймворку, але й від рівня абстракції, екосистеми, наявності готових бібліотек та інструментів, а також досвіду розробників. Використання сучасних

фреймворків дозволяє значно скоротити час розробки, зменшити кількість помилок та підвищити якість кінцевого продукту.

У процесі тестування було виявлено, що найбільший вплив на швидкість розробки мають такі фактори, як простота налаштування середовища, наявність документації, підтримка спільноти та можливості інтеграції з іншими технологіями. Водночас, складні або надмірно універсальні фреймворки можуть збільшувати час освоєння та ускладнювати процес розробки на початкових етапах.

Загалом, результати дослідження підтверджують, що правильний вибір фреймворку є критично важливим для забезпечення ефективності процесу розробки веб-додатків. Використання сучасних технологій дозволяє оптимізувати робочі процеси, скоротити час реалізації проєктів та підвищити конкурентоспроможність програмного забезпечення.

Разом із тим, варто враховувати, що універсального рішення не існує, і вибір фреймворку має базуватись на конкретних вимогах проєкту. У подальшому дослідження може бути розширене шляхом аналізу інших фреймворків, дослідження впливу мікросервісної архітектури або використання хмарних технологій, що дозволить отримати більш глибоке розуміння факторів, які впливають на швидкість розробки веб-додатків.

СПИСОК ПОСИЛАНЬ НА ДЖЕРЕЛА

1. Angular Documentation. (2025). angular.io. https://angular.io/docs
2. React Documentation. (2025). react.dev. https://react.dev
3. Vue.js Documentation. (2025). vuejs.org. https://vuejs.org/guide/introduction.html
4. Laravel Documentation. (2025). laravel.com. https://laravel.com/docs
5. Ruby on Rails Guides. (2025). rubyonrails.org. https://guides.rubyonrails.org
6. Node.js Documentation. (2025). nodejs.org. https://nodejs.org/en/docs
7. Express.js Guide. (2025). expressjs.com. https://expressjs.com
8. Next.js Documentation. (2025). nextjs.org. https://nextjs.org/docs
9. Nuxt.js Documentation. (2025). nuxt.com. https://nuxt.com/docs
10. Django Documentation. (2025). djangoproject.com. https://docs.djangoproject.com
11. Microsoft. (2025). ASP.NET Core Documentation. learn.microsoft.com. https://learn.microsoft.com/en-us/aspnet/core
12. Google. (2025). Web Performance Optimization Guide. web.dev. https://web.dev/performance
13. Mozilla. (2025). Web Performance. developer.mozilla.org. https://developer.mozilla.org/en-US/docs/Web/Performance
14. Fowler, M. (2002). Patterns of Enterprise Application Architecture. https://martinfowler.com/eaaCatalog/
15. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns. Pearson.
16. Martin, R. C. (2017). Clean Architecture. Pearson.
17. Martin, R. C. (2008). Clean Code. Prentice Hall.
18. Freeman, E., & Robson, E. (2021). Head First Design Patterns. O’Reilly.
19. Banks, A., & Porcello, E. (2020). Learning React. O’Reilly.
20. Flanagan, D. (2020). JavaScript: The Definitive Guide. O’Reilly.

21. Tilkov, S., & Vinoski, S. (2010). Node.js: Using JavaScript to Build Scalable Network Programs. IEEE Internet Computing.

22. Pautasso, C. (2016). RESTful Web Services vs. Big Web Services. IEEE.

23. Richards, M. (2020). Software Architecture Patterns. O'Reilly.

24. Newman, S. (2021). Building Microservices. O'Reilly.

25. Spinellis, D. (2012). Code Quality: The Open Source Perspective. Addison-Wesley.

26. Sadalage, P., & Fowler, M. (2012). NoSQL Distilled. Addison-Wesley.

27. Google. (2024). Core Web Vitals. web.dev. https://web.dev/vitals/

28. Lighthouse Performance Tool. (2025). developers.google.com. https://developers.google.com/web/tools/lighthouse

29. State of JS Survey. (2024). stateofjs.com. https://stateofjs.com

30. Stack Overflow Developer Survey. (2025). stackoverflow.co. https://survey.stackoverflow.co

31. Gartner. (2024). Web Development Trends. gartner.com. https://www.gartner.com/en/information-technology

32. McKinsey. (2024). Developer Productivity Report. mckinsey.com

33. IBM. (2024). Modern Web Application Development. ibm.com

34. AWS. (2025). Building Scalable Web Applications. aws.amazon.com

35. Red Hat. (2024). Modern Application Development. redhat.com

36. DigitalOcean. (2025). Choosing a Web Framework. digitalocean.com/community/tutorials

37. MDN. (2025). JavaScript Frameworks Overview. developer.mozilla.org

38. W3C. (2025). Web Standards. w3.org

39. Khan, A., & Lee, S. (2023). Performance Comparison of Modern Web Frameworks. Journal of Web Engineering, 22(4), 455–478.

40. Zhang, Y., & Kumar, S. (2024). Impact of Framework Selection on Web Application Development Speed. ACM Computing Surveys, 56(3), 1–28.

БІБЛІОГРАФІЧНА ДОВІДКА

Тема бакалаврської роботи: " Вплив вибору фреймворків на швидкість створення веб-додатків "

Обсяг пояснювальної записки: 75 аркушів

Дата закінчення бакалаврської роботи 10 червня 2026р.

Підпис студента _____