

Bachelor's Thesis

Turku University of Applied Sciences

Bachelor of Engineering, Information and Communications Technology

2022

Dhruv Verma

A Comparison of Web Framework Efficiency–
Performance and network analysis of modern web
frameworks



Bachelor's Thesis | Abstract

Turku University of Applied Sciences

Bachelor of Engineering, Information and Communications Technology
Bachelor of Engineering, Information and Communications Technology

2022 | 115

Dhruv Verma

A Comparison of Web Framework Efficiency

- Performance and network analysis of modern web frameworks

The world of web frameworks has evolved from its early days of being a static website to now a dynamic and interactive web application. With the ever-growing web development technologies, many developers are struggling to pick the best framework.

Subjective and individual opinions are no longer relevant to decide a good framework. Therefore, the goal of this thesis was to implement a chat application using modern web frameworks and provide an efficiency comparison by performing an in-depth network and performance analysis based on surveys and reports, of frameworks such as React, Angular, Rails, Flask and Swift.

Though, it cannot be successfully deduced that any framework performance better than the other. The performance of a framework depends on the type of application in development. However, it can be predicted that for a simple application the results imply that the combination of React JS and Rails is the best.

Keywords:

WebSocket, Requests, Responses, Protocol, Development, Frameworks,
Native Applications

CONTENTS

List of Abbreviations	9
1 INTRODUCTION	11
1.1 Background Theory	11
1.2 Research Problem	13
1.3 Research Method	14
1.4 Review of Literature	14
1.5 Structure of thesis	15
2 CONCEPTUAL LAYOUT OF THE RESEARCH	17
2.1 Network Traffic	17
2.1.1 TCP	18
2.1.2 HTTP	19
2.1.3 WebSocket	19
2.2 Website Components	20
2.2.1 Front-End	21
2.2.2 Back-End	21
2.3 Brief History of Web Frameworks	22
2.4 Evolution of Web Frameworks	23
2.4.1 Early Web Development	23
2.4.2 Modern Web Development	28
3 DATA AND METHODS OF RESEARCH	32
3.1.1 Native and Cross-Platform Applications	33
3.1.2 Web Applications	36
4 IMPLEMENTATION OF THE APPLICATIONS	40
4.1 GUI Flow	41
4.2 Models	41
4.2.1 Chat Message Model	42
4.2.2 User Model	42
4.3 Web Applications	43

4.3.1 React JS and Rails	43
4.3.2 Angular JS and Flask	47
4.3.3 Vue JS and Laravel	50
4.4 Native and Cross-Platform Applications	53
4.4.1 Swift	53
4.4.2 React-Native and Express	57
5 COMPARISON MODEL FOR THE FRAMEWORKS	61
5.1 Benchmarks	61
5.1.1 Development and Ease of Modification	61
5.1.2 Ease of Deployment	61
5.1.3 Generated HTML Structure	62
5.1.4 Framework Performance	62
5.1.5 Corresponding to modern standards	63
5.2 Frameworks	64
5.2.1 Angular JS and Flask	64
5.2.2 React JS and Rails	69
5.2.3 Vue JS and Laravel	74
5.2.4 Swift iOS	79
5.2.5 React Native and Express	83
6 RESULTS AND CONCLUSION	88
6.1 Frameworks in consideration	88
6.2 Results of Web-Application Frameworks	90
6.2.1 Benchmark Analysis	90
6.2.2 Network Analysis	91
6.3 Results of Native-Application Frameworks	94
6.3.1 Benchmark Analysis	94
6.3.2 Network Analysis	95
7 Conclusion	98
8 Bibliography	100

Equations

Equation 1. Server Time Delay - Windows	37
Equation 2. Sever Time Delay - Windows	37
Equation 3. Resulting Server Time Difference	37

Figures

Figure 1. TCP Model.	18
Figure 2. HTTP Model	19
Figure 3. WebSocket Connection.	20
Figure 4. Front-End Example.	21
Figure 5. Evolution of Web Frameworks.	22
Figure 6. AJAX Requests.	23
Figure 7. Three Layer Architecture.	24
Figure 8. Added Layers - Three Layer Architecture.	25
Figure 9. MVC Structure.	26
Figure 10. MVVM Structure.	27
Figure 11. Modern API Request Structure.	28
Figure 12. Example - Web Application.	29
Figure 13. Example – Native Application.	31
Figure 14. Schema of Study.	33
Figure 15. Network Flow – Android.	35
Figure 16. Captured Packet Details – Android.	35
Figure 17. Network Flow - iOS	36
Figure 18. OSI Model (Imperva, 2021).	38
Figure 19. CSV - Generated Chart.	39
Figure 20. Chat Application Structure.	40
Figure 21. Chat Application - GUI flow.	41
Figure 22. Chat Application - Data Models.	42
Figure 23. React JS - User Login.	44
Figure 24. Rails - Broadcasting Channel.	45

Figure 25. React - Working Chat Application.	46
Figure 26. React - Chat Application server logs.	46
Figure 27. React - Chat Application Postman Test.	47
Figure 28. Angular – Login.	47
Figure 29. Flask - WebSocket Connection.	48
Figure 30. Angular – Working Chat Application.	49
Figure 31. Angular - Chat Application server logs.	49
Figure 32. Vue - Login Components.	51
Figure 33. Xampp - Apache server & SQL Database.	51
Figure 34. Laravel - Pusher API.	52
Figure 35. Vue – Chat Application Server Logs.	52
Figure 36. Vue - Working Chat Application.	53
Figure 37. Swift UI Components.	54
Figure 38. Swift - WebSocket Connection.	55
Figure 39. Swift - Data Hashing.	55
Figure 40. Swift - Chat Application Server Logs.	56
Figure 41. Swift - Working Chat Application.	56
Figure 42. Swift - Manual Connection Test.	57
Figure 43. React-Native Navigation Stack.	58
Figure 44. React-Native WebSocket Connection.	58
Figure 45. React-Native Chat Application Server logs.	59
Figure 46. React-Native Working Chat Application.	60
Figure 47. Application Deployment Criteria.	62
Figure 48. Angular-Flask HTTP Packet Numbers.	64
Figure 49. Angular-Flask HTTP Packet Length.	65
Figure 50. Angular-Flask - No. of TCP Packets.	65
Figure 51. Angular-Flask - Length of TCP Packets.	66
Figure 52. Angular-Flask WebSocket Data Quantity.	67
Figure 53. Angular-Flask WebSocket Data Length.	67
Figure 54. Angular-Flask WebSocket Stalling Time.	68
Figure 55. Angular-Flask WebSocket TTFB.	68
Figure 56. Angular-Flask Application Performance.	69

Figure 57. React-Rails Number of HTTP Packets.	70
Figure 58. React-Rails Length of HTTP Packets.	70
Figure 59. React-Rails Number of TCP Packets.	71
Figure 60. React-Rails Length of TCP Packets.	71
Figure 61. React-Rails Quantity of WebSocket Data.	72
Figure 62. React-Rails Length of WebSocket Data.	72
Figure 63. React-Rails WebSocket Stalling Time.	73
Figure 64. React-Rails Chat Application Performance.	74
Figure 65. Vue-Laravel Number of HTTP Packets.	75
Figure 66. Vue-Laravel Length of HTTP Packets.	75
Figure 67. Vue-Laravel Number of TCP Packets.	76
Figure 68. Vue-Laravel Length of TCP Packets.	76
Figure 69. Vue-Laravel Quantity of WebSocket Data.	77
Figure 70. Vue-Laravel Length of WebSocket Data.	77
Figure 71. Vue-Laravel WebSocket Stalling Time.	78
Figure 72. Vue-Laravel Chat Application Performance.	78
Figure 73. Swift - Number of HTTP Packets.	79
Figure 74. Swift - Length of HTTP Packets.	80
Figure 75. Swift - Number of TCP Packets.	80
Figure 76. Swift - Length of TCP Packets.	81
Figure 77. Swift - Quantity of WebSocket Data.	82
Figure 78. Swift - Length of WebSocket Data.	82
Figure 79. React-Native-Express - Number of HTTP Packages.	83
Figure 80. React-Native-Express - Length of HTTP Packages.	84
Figure 81. React-Native-Express - Number of TCP Packages.	84
Figure 82. React-Native-Express - Length of TCP Packages.	85
Figure 83. React-Native-Express - Quantity of WebSocket Data.	85
Figure 84. React-Native-Express - Length of WebSocket Data.	86
Figure 85. React-Native-Express - WebSocket Stalling Time.	86
Figure 86. React-Native-Express - Chat Application Performance.	87
Figure 87. Web Framework - Benchmark Comparison.	91
Figure 88. Web Applications - HTTP Analysis.	92

Figure 89. Web Applications - TCP Analysis.	93
Figure 90. Web Applications - WebSocket Analysis.	93
Figure 91. Native Applications - Benchmark Analysis.	95
Figure 92. Native Applications - HTTP Analysis.	96
Figure 93. Native Applications - TCP Analysis.	96
Figure 94. Native Applications - WebSocket Analysis.	97

Tables

Table 1. Framework based on programming language.	29
Table 2. Framework - Benchmark Criteria (Percentage).	63
Table 3. Web Frameworks for Chat Application.	88
Table 4. Native Frameworks for Chat Application.	88

List of Abbreviations

AJAX	Asynchronous JavaScript And XML (AJAX) is a method which allows the web pages to be updated asynchronously rather than loading one part of the webpage at one time.
API	Application Programming Interface (API) is an intermediary layer which allows two applications to communicate with each other.
CSV	Comma-Separated Values (CSV) is a delimited text file which separates values by using commas.
GUI	Graphical User Interface (GUI) is a graphic based interface that uses icons, menus and clicks to perform interactions with the application.
HTTP	Hypertext Transfer Protocol (HTTP) is an application layer protocol which used to transmit media documents such as generated HTML structures.
MVC	Model-View-Controller (MVC) is a common software design pattern that is used to implement user interface while separating the functions and database layers.
MVVM	Model-View-View-Model (MVVM) is a software design pattern that allows the separation program logic and user interface control layers in the web application.
REST	REpresentational State Transfer (REST) is a set of architectural rules which transfer a representation of the state of the data in the application.
SPA	Single Page Application (SPA) is a web application that interacts with the end user by dynamically changing the

data of the current user page rather than re loading the new page.

- SSDP Simple Service Discovery Protocol (SSDP) is a text-based protocol which uses User Datagram protocol (UDP) to transport packets.
- SSL Secure Socket Layer (SSL) is a standard security layer that establishes an encrypted link between a server and a client.
- TCP Transmission Control Protocol (TCP) is a communication standard which provides a way for the devices to communicate with each other and send packages.
- WS WebSocket (WS) is a TCP based communication connection which provides a constant communication between the client and the server over a single handshake.

1 INTRODUCTION

Since the introduction of the Internet and World Wide Web in the early 1990s, there have been many remarkable discoveries in the terms of technologies of sharing data and communication. In recent years, many organizations have witnessed unprecedented growth in the field of Web Development. Most of the working sectors and businesses have adopted the internet as their primary platform. As a result, the need for fast, dependable, and engaging internet applications has increased. (EDUCBA, 2019)

1.1 Background Theory

Early websites used to depend on HTML to exchange information and data. While HTML-based websites can easily transmit static content such as text documents, photographs and blogs, its application for more complicated models is quite restricted. HTML cannot meet the requirements for application integration, flexible models, and portable environments which led to the introduction of a three-tier architecture which allows the separation of logic layers in an application. Thus, this improved three-tier architecture including a presentation layer, a business layer and a dedicated database layer. The presentation layer supports the user and web application interface, and the business layer handles data retrieval and posting, validation and rules enforcement. Lastly, the data layer connects to the database using the given modules, classes, and models. In general, this architecture allows web applications to work with complex models and structures while at the same time decoupling the database and present layers of user interface from the business layer. In addition to the common benefits of modular software, this architecture was designed to allow any of the three tiers to be upgraded or changed independently if any technology or requirement changes. (IBM Cloud Education, 2020)

Numerous frameworks emerged due to this architecture, and subsequently, many companies developed technologies to complement each layer. For instance:

UI Layer: HTML, CSS, and JavaScript

Back End Layer: Perl, PHP, C#, and ASP

Database Layer: SQL, MySQL and Oracle

Browser-Server Communication Layer: XML and AJAX

Building a website or hosting a server with these technologies is a tedious process. A web framework eliminates the need for programming and setting up basic code from scratch. Therefore, the introduction of frameworks rendered these mentioned technologies obsolete.

In today's era of website technology, there are three main types of applications: native, web and progressive web. Native applications are platform-dependent and require specialized programming languages and development kits, whereas Web apps are platform-independent webpages that feel like native apps in many respects. Native applications access the device hardware while web applications have limited access. However, both applications have their own advantages and shortcomings based on the used framework and platform. On the other hand, Progressive Web Applications (PWA) are developed using a specific set of standard patterns that allows these applications to run on the native environment while still having the common web architecture.

Today, Web applications are developed using PWA templates and methodologies to enhance user experience and reduce server load. Additionally using a CMS makes developing the application much easier and it provides better integration and stability. Instead of writing the source code, CMS allows the developer to create, edit and manage data fields such as text, images, audio, and videos.

The study is used as a foundation for examining a real-life web development case and implementation of a chat application across different platforms and devices.

The document's structure is divided into three sections. The first part provides a deep insights into web and mobile frameworks. The second one presents the comparison of different application framework models to gain a better understanding of their attributes. The methods chosen to assess the effectiveness of each framework are also discussed. Following that, the implementation of one application using different frameworks is analyzed. Afterward, the frameworks' relative performance is evaluated, and all the findings are summarized.

1.2 Research Problem

Various web frameworks have been developed in the field of web development in recent times. Of these, frameworks such as Angular, React, Swift, Laravel, Express, and Flask have prospered. These web frameworks provide different capabilities which are suited for different types of applications. As a result, there are several comparative studies regarding the web frameworks such as *Comparison of Mobile Web Frameworks* (Heitkötter, et al., 2014), and *A comparison model for agile web frameworks* (Ignacio Fernández-Villamor, 2008). These comparative studies examine the frameworks and perform a subjective analysis of their capabilities.

There is not a comparison model based on the performance and network analysis of the applications. Therefore, the research on the comparison of web frameworks will be focused on analyzing the web frameworks using different benchmark criteria, network, and performance analysis.

The motivation of the thesis is to examine the performance of the concerned frameworks by developing the same chat application across popular web and mobile frameworks. The determined performance is used to conduct a cross-comparison among all the concerned frameworks.

1.3 Research Method

Web development framework combinations such as React and Rails, Angular and Flask, React Native and Express, Vue and Laravel, and Swift have been used in the analysis. A test chat application has been implemented using all these framework combinations. This application would allow for an in-depth comparison analysis of the frameworks.

For comparison analysis, benchmark criteria like development, ease of deployment, ease of modification, framework performance are considered. Similarly, the network analysis of the frameworks includes analyzing HTTP, TCP, and WebSocket Packets.

The implemented chat application was run for a span of 10 minutes and was refreshed frequently to generate unique traffic. This traffic was then captured for detailed analysis.

1.4 Review of Literature

A comparison model aims to ensure the stability and integrity of the framework while aiming for high performance. The main components of the comparison model include development, network, and performance analysis.

The problem with majority of the comparison models is that almost all of them compare at least two frameworks, but none of them compare data acquired from network measurements. The study *Comparing Web Frameworks* (Raible, 2006) is based on personal experience of the author. The lack of case studies only leaves an author's personal opinions on the frameworks. The study describes some frameworks using examples with source code. The comparison criteria, however, are only limited to internationalization, AJAX support, and validation of the framework. Some results are based on the author's perspective and therefore can vary from person to person while choosing a modern web framework.

The study *Evaluation and Implementation of Progressive Web Application* (Thakur, 2018) does refer to frameworks such as React and Angular but does not drive any comparative conclusion. The study refers to the implementation of Progressive Web applications using React JS. The author describes using this framework due to its popularity among the developers. However, there is no mention of a technical comparison. The study only gives an overview of the development process of a news application.

Another research *Comparative study on Python web frameworks: Flask and Django* (Ghimire, 2020) describes a comparison study between Flask and Django. Both of these frameworks are developed using Python. However, the comparison criteria only include design patterns, request routing, flexibility, and error handling. These results do not offer any technical comparisons for the frameworks.

1.5 Structure of thesis

The thesis is divided into seven chapters.

Chapter 1 presents the motivation for the research and the research problem and objectives. It introduces web development and web frameworks

Chapter 2 provides the conceptual layout of the research. It describes the origin and evolution of web frameworks. It deeply explains the network flow within the application. It explains HTTP, TCP and WebSocket and gives an in-depth knowledge of the working principle behind the network.

Chapter 3 presents the methods used to capture the required network data from the running application. The methods include the research carried out on the web applications and the native applications.

Chapter 4 discusses the implementation of the application using every concerned framework.

Chapter 5 describes the comparison model and the analysis carried out on the frameworks.

Chapter 6 discusses the results of the analysis

Chapter 7 discusses the conclusion that have been made from the observations.

2 CONCEPTUAL LAYOUT OF THE RESEARCH

With the emergence of the new web development technologies every year, the developers must analyze each application framework suitable for the project. The analysis are based on technical prospects such as network analysis of the application. The newly released framework may offer new benefits and integrity, but it might only be for specified platforms and operating systems. Therefore, it is becoming a necessity to choose frameworks that are well suited for the project. This suggests several important questions such as:

- 1) What is a framework?
- 2) How is network analysis of a framework performed?
- 3) Is there a comparison model for frameworks?

Accordingly, the following sections will determine the concept for network analysis, the definition and classification of modern frameworks, details about frameworks and a comparison model for the frameworks. With a prior knowledge of compared results of different frameworks, it is easier to select the framework which would work the best with the required project management methodology and operating platform.

2.1 Network Traffic

Network packets make up the network traffic. Network analysis, also known as network traffic analysis, can be described as a method for analysis network packets. It includes the monitoring of different network activities such as HTTP, TCP and WebSocket, collecting relevant information and analyzing it in real-time. It can be used for a variety of purposes, like detecting fault on a network, studying the behavior of the network, and identifying malicious activities. Moreover, it is also helpful in identifying the vulnerable protocols which might be the target of any malicious activity in the future. While it might not be of a big use in a personal level, but it is extremely important on an enterprise level, where thousands of

machines are connected to various chains of servers and network devices. These devices produce a vast amount of traffic. So, these servers are most suitable choice for most attackers, therefore network analysis provides a great deal of identification to overcome these threats. (Techopedia, 2019)

The main network requests in an application are followed through the transfer protocols which are: Transmission Control Protocol (TCP), Hypertext Transfer Protocol (HTTP) and WebSocket (WS).

2.1.1 TCP

TCP is connection-oriented protocol which defines how the computers send the packets of data to each other. TCP establishes a connection and maintains it until the exchange of the packets is finished. It is used to organize the packets of data in a secure way to ensure the proper transmission. TCP plays an important role in establishing the rules and standard procedures to carry out the communication over the internet.

TCP (Figure 1) is stacked in a conceptual model for data exchange which is the TCP/IP stack model. The TCP/IP stack model repackages data at each layer based on the functionality and transport protocols.

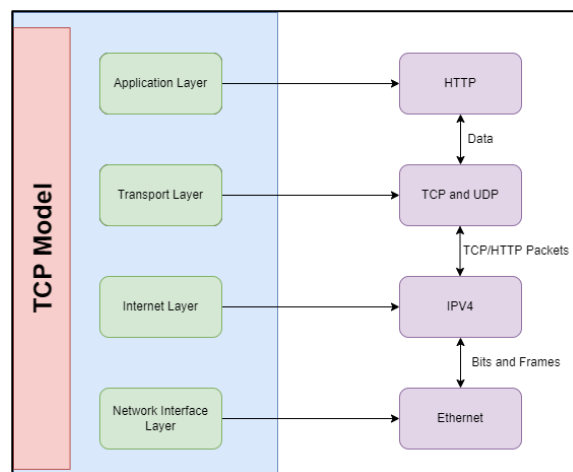


Figure 1. TCP Model.

2.1.2 HTTP

HTTP (Figure 2) is a protocol which utilizes the services of the TCP ports to facilitate the transfer of a document over the internet. Unlike the other connection the HTTP uses only one TCP connection which is 'Data Link' to proceed with the transfer. It is client-server protocol that fetches the HTML documents to exchange information on the Web. The client-side delivers the request message into the webserver which sends the requested content back to the client. During the request and the response, HTTP uses the Secure Socket Layer (SSL) to secure the entire communication.

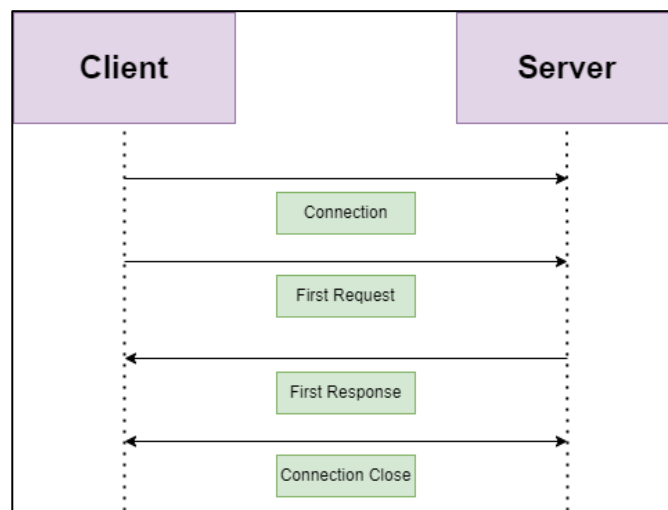


Figure 2. HTTP Model

2.1.3 WebSocket

WebSocket (Figure 3) is a stateful protocol which means that it sends a request to the server and expects a response. In case a response is not received, the stateful protocol resends the request. WebSocket maintains the connection between the client and the server until it is terminated by either side. The data is shared between the client and server continuously which eliminates the request polling time. The server does not have to wait for the client's state before responding to the client's request. Unlike HTTP requests, the WebSocket maintains a bi-directional connection with the client and the server. After the client

requests the connection with the server, a handshake between both parties happens. WebSocket does not require continuous handshaking, because it keeps the connection alive.

The WebSocket serves the main purpose of transferring real time information from the server to the client faster than HTTP. So, the trading applications, gaming applications, and chat applications use WebSocket.

The figure 3 shows the HTTP request handshake is performed between the client and the server. This handshake is soon followed by the bi-directional communication and then the closure of the connection.

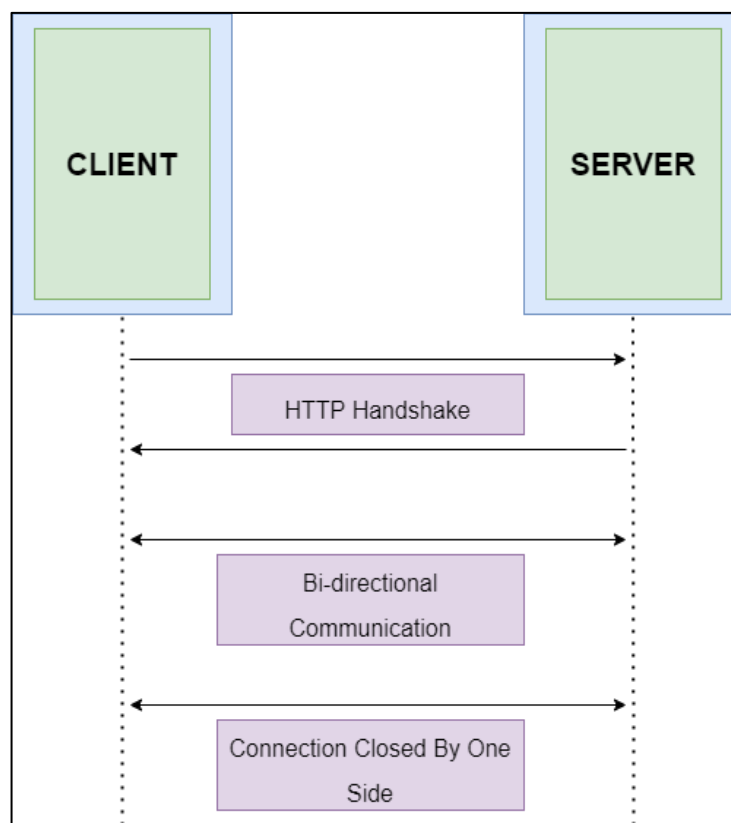


Figure 3. WebSocket Connection.

2.2 Website Components

A website is basically divided into two core parts: The Front-End and The Back-End. These core parts refer to the separation of the UI layer and the data layer.

2.2.1 Front-End

The front-end (Figure 4) of a website can be described as the interface of the website or web application which is accessible by the user. It is also known as the Client-Side of the website or web application. The front end is created using technologies such as HTML, CSS, and JavaScript. All the interactions to the server are made possible by the front end. The front end is responsible for the website's architecture to provide a simple, yet elegant user experience without affecting any functionality.

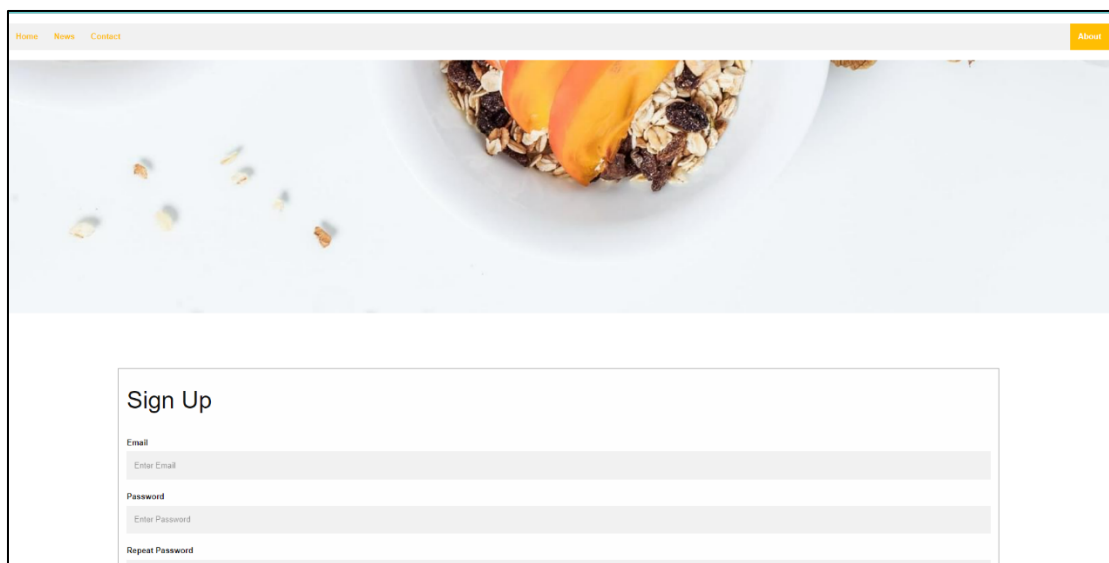


Figure 4. Front-End Example.

2.2.2 Back-End

The back end or the server-side of the website handles the server functionality. All the requests and responses are handled by the server of the website or web application. This may include retrieving data, posting information, updating data from and to the databases. It also processes all the logic that the website or web application requires. After completing the requests, the back end sends the response to the front end so that it could render the view for the user.

2.3 Brief History of Web Frameworks

Web Frameworks are considered to be the most important part of any web application. Web framework is a set of tools that helps build a website thus avoiding the bugs and conserving time. Both static and dynamic web pages can use frameworks. (Curie et al., 2019). Accordingly, a web application framework can be defined as the reusable set of programming language code libraries and tools designed to support the web application development. The upside of using web frameworks is the comprehensive efficiency and code organization.

There are countless types of web frameworks (Figure 5), but they all fall into the category of being progressive, native or hybrid. These frameworks are created for the general purpose of adding more web components to the original programming language.

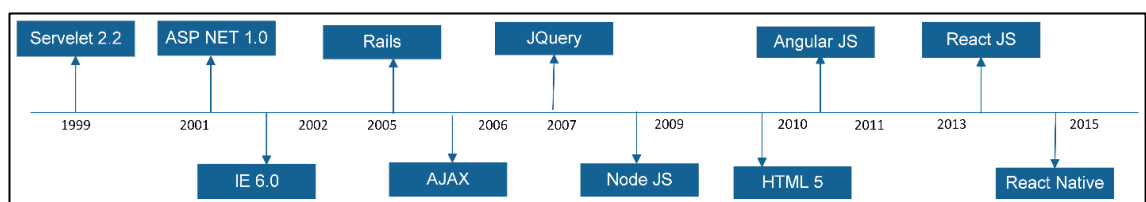


Figure 5. Evolution of Web Frameworks.

Web frameworks grew in popularity as JavaScript became more capable in the browsers. In 2004, the development of Asynchronous JavaScript and XML (AJAX) (Figure 6) technique went in demand for building dynamic websites without the need for full-page refresh. The demand for web development using JavaScript mandated the W3C to implement new set of rules. These rules required the browser vendors to implement the technologies which facilitate standardized web technologies such as HTML, JavaScript/ECMAScript, and CSS.

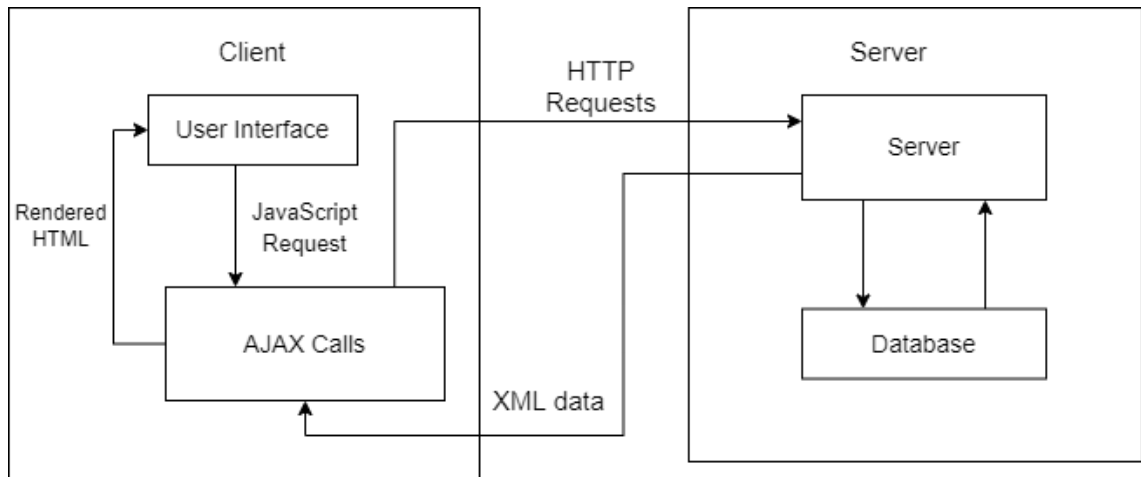


Figure 6. AJAX Requests.

To overcome the poor user experience and at the same time facilitate DOM manipulation, web frameworks started to evolve. The next generation of frameworks enforced good client-server architecture, followed modern web standards and were quite stable which offered advanced user experience. The emergence of AngularJS framework and Backbone framework proved that it was viable to create web applications which could run natively in the browser without the need for extremely fast computers.

2.4 Evolution of Web Frameworks

2.4.1 Early Web Development

Before the evolution of web frameworks, the websites followed a simple software architecture. It consists of: Presentation Layer, Business Layer and Persistence Layer. The concept of isolating each layer results in each layer working independently of the other. No layer is affected by the refraction of the other layer (Berninger, 2001). This isolates the changes and makes it easier to further develop that layer (Figure 7).

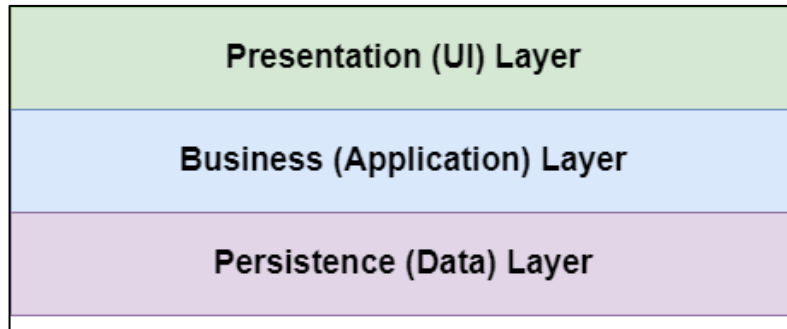


Figure 7. Three Layer Architecture.

Presentation Layer

Presentation layer is known as the user interface (UI) Layer is the topmost layer of the old software architecture. This layer provides presentation services which include the demonstration of the content to the end user using Graphical User Interface (GUI). The layer can be accessed through different client devices such as laptop, desktop, mobile, and tablets. The layer presents the content by interacting with other layers in the software architecture model.

Business Layer

Business layer is also known as the application layer which is the middle layer of the software architecture. The layer follows the set of rules which are required for the application to work. Hence, this layer comprises of Business Logic which typically runs on one or more application servers.

Persistence Layer

Persistence layer described as the data layer. It is the lowest layer of the software model. This layer is concerned with the storage and retrieval of the application data from the database, file server or any other storage media.

Large businesses followed the same suite of designing responsive websites using this software architecture. Though some websites require high cohesion where the pieces of codes are linked to each other. Therefore, to solve this problem, some additional layers are added to the software architecture (Figure 8).

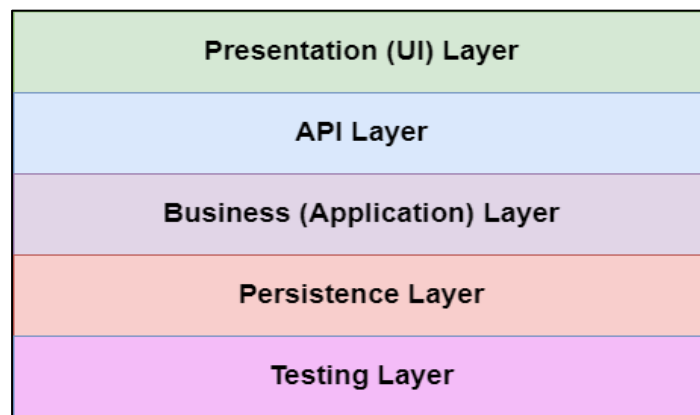


Figure 8. Added Layers - Three Layer Architecture.

During the rise of web development in 2000s, many internet sites relied on the servers to render out the views for the client side. The browser acted like a viewer which would display all the generated result. This result included dynamically generated HTML and CSS pages by the server. These websites followed a standard Model-View-Controller structure or MVC structure.

Model-View-Controller

Model View Controller or MVC (Figure 9) is a web architecture that follows the standard software architecture. The model layer comprises of the data layer which handles all the modification logic. Similarly, the controller layer handles the business logic and provides communication between the view and the model layer. Lastly the View layer comprises of the presentation layer and handles all the UI that is accessible to the user. (Majeed & Rauf, 2018)

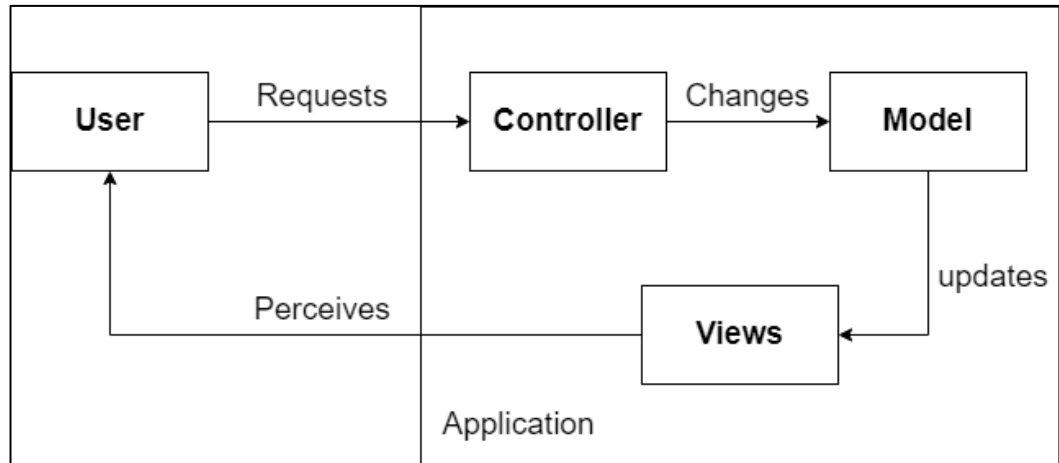


Figure 9. MVC Structure.

All the requests and responses can then be handled by jQuery or AJAX to make the web page interactive. Though this architecture faces a lot of complications such as coupling of components, front-end (views) and back-end (model and controller) integration which results in insufficient architecture.

In the 2000s many companies faced the same coupling problem with the MVC architecture which drove the developers to use the decoupled APIs from the server. Though this required the application to run with browser plugins which needed separate installation. These plugins were often outdated and had various security vulnerabilities making them obsolete to run web and native applications.

Model-View-ViewModel Architecture

The early 2000s provided a fix to these problems by introducing a Model-View-ViewModel (MVVM) architecture client with a Representational State Transfer (REST) API.

REST is an application programming interface or API which allows multiple clients to communicate with the server. It provides great flexibility to the developers as it eliminates the dependence of code libraries to access the web-services. It handles the request in a parameter form. These parameters

include – the endpoint of the API, the method of the API and the data transferred using the API.

The Endpoint: The Endpoint is a unique URL which represents the data object. HTTP requests are directed to the endpoint to interact with all the data resources. The data is also referred as the body of the request which represents the resource. The data contains the required information about the resource.

The MVVM (Figure 10) decouples the view, and the logic and model layers which makes development of the GUI a lot easier as compared to the MVC architecture. The ViewModel layer of this architect communicates with the Rest API to provide the requested resource to the client and present it using the View layer.

However, both the client and server side implement their own software architecture layers which makes the architecture complicated.

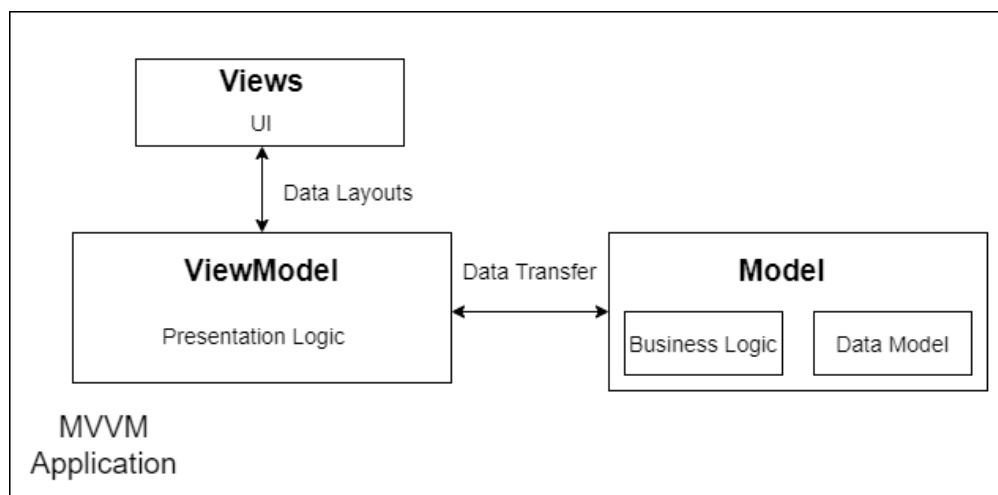


Figure 10. MVVM Structure.

The early web frameworks such as Backbone and AngularJS struggled to develop the websites with ill-designed architecture. In some cases, API(s) returned data models which exposed the relational data models to the web applications which introduced a new security vulnerability. Moreover, the badly mapped structure of data created uncontrolled coupling between layers of the

architecture which created unconditional complexities. Hence, the need for new web frameworks arose (Strawn, 2018).

2.4.2 Modern Web Development

The modern web development follows the modern frameworks. These frameworks enforce good architecture, follow modern web standards, provide scalability and, stability.

The modern applications (Figure 11) use API layers to flatten the data model before sending the response to the client which makes the data transfer optimal and the need for specialized API(s) is eliminated.

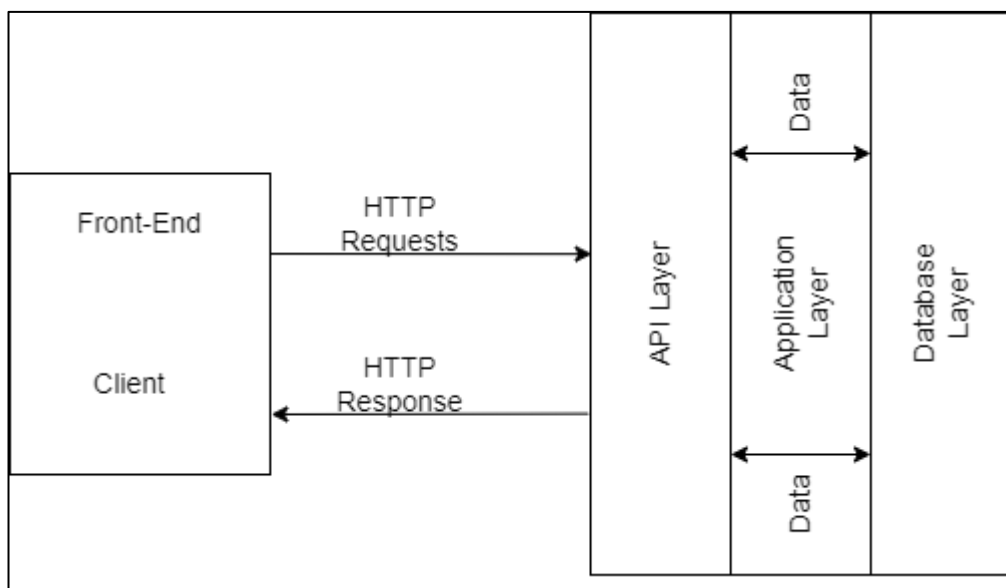


Figure 11. Modern API Request Structure.

Web Applications

In web development terms, a web application can be described as a client-side and server-side software application in which the client runs and requests in a web browser. The examples for these web applications include messaging services, retails websites, email clients and online forms.

The table 1 lists some modern web application frameworks, categorized by the back-end and front-end programming languages.

Table 1. Framework based on programming language.

Programming Language	Front-End Framework	Back-End Framework
JavaScript	React, Vue, Angular	Express (Node JS), Next.js
Python	Turbogears, Django	Django, Flask
Ruby	Ruby	Rails
PHP	Nil	Laravel, Symfony

A web application (Figure 12) requires a web server, an application server and a database to operate. All the requests from the client are managed by the web servers while the requested task is completed by the application server. All the required information is stored to the database.

These applications can be accessed through multiple browsers, can be accessed by multiple users at the same time and do not require any download. Hence these applications are a perfect candidate to deliver content to the users.

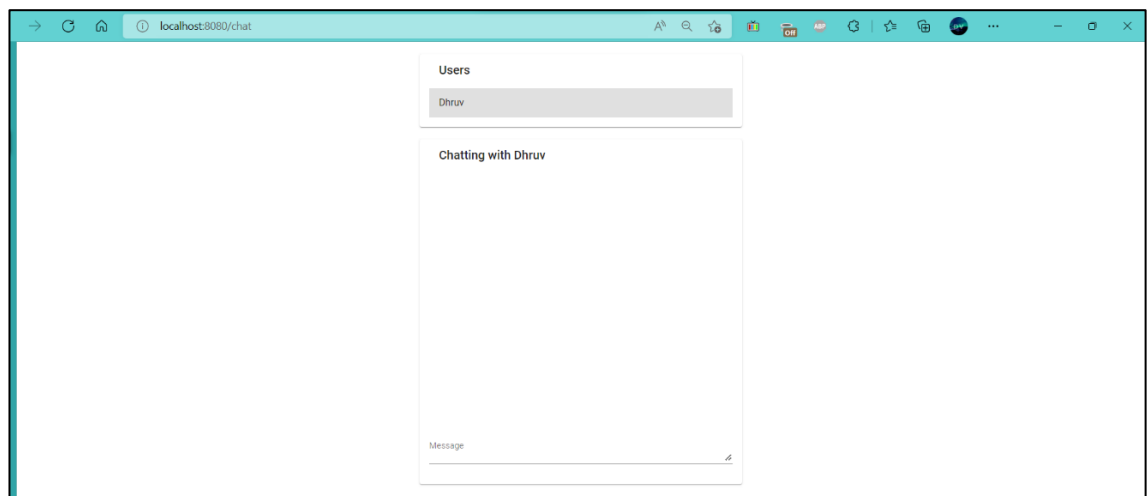


Figure 12. Example - Web Application.

Native and Cross-Platform Applications

The modern web development defines native applications as a program which is written to work on a specific platform. These native applications (Figure 13) are developed for the iOS and Android platforms. These applications work with the mobile device's OS to deliver the request content faster and provide more flexibility than the alternative applications.

The cross-platform frameworks operate on the idea of developing reusable code for building applications for different OS. For instance, the same code for an Android application can be used to develop an iOS application with a different architecture.

These applications can access the device's functionalities such as microphone, gyroscope or push notifications. Therefore, the practical uses for these applications range from simple music programs such as 'Black Player Ex' to social applications such as 'Twitter' or 'Facebook'.

Below are the native and cross-platform application frameworks, categorized by the device OS:

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

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

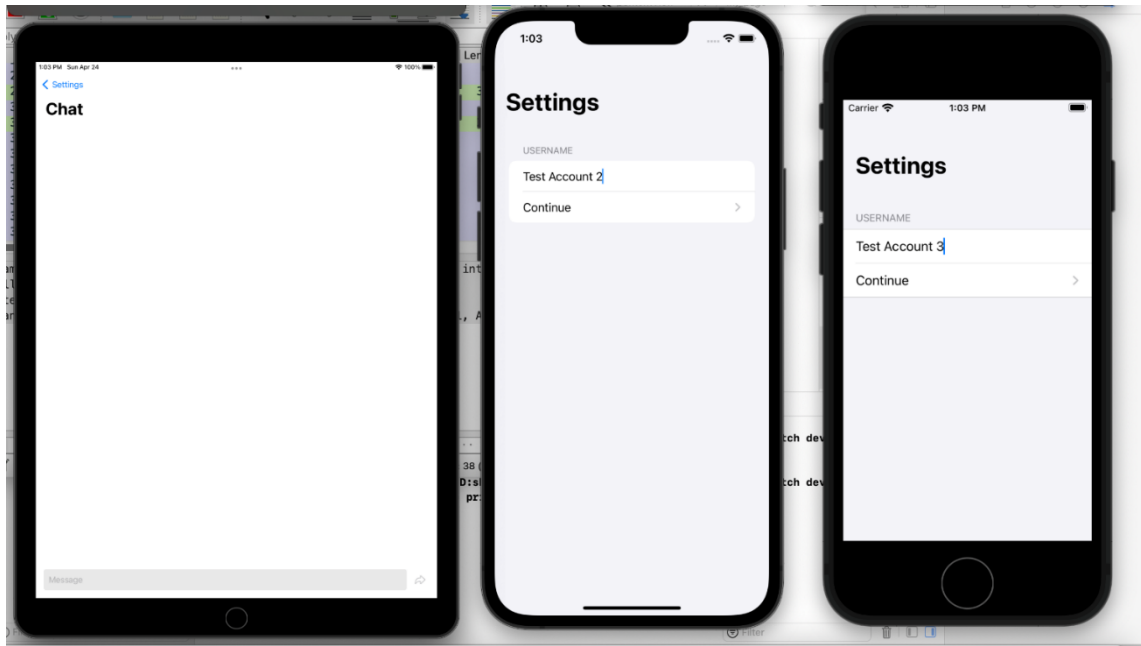


Figure 13. Example – Native Application.

3 DATA AND METHODS OF RESEARCH

The objective of this study is to analyze the performance of different frameworks, running the same application on the same server. Network capturing methodology has been used to get the required dataset for various frameworks. The most important part of network capture was to isolate the other network background running activities from the chat server network requests (Figure 14).

Obtaining data for the web applications is easier compared to the native applications because background communication makes it difficult to obtain the required dataset. Hence, it is important to isolate external networks from the main chat server.

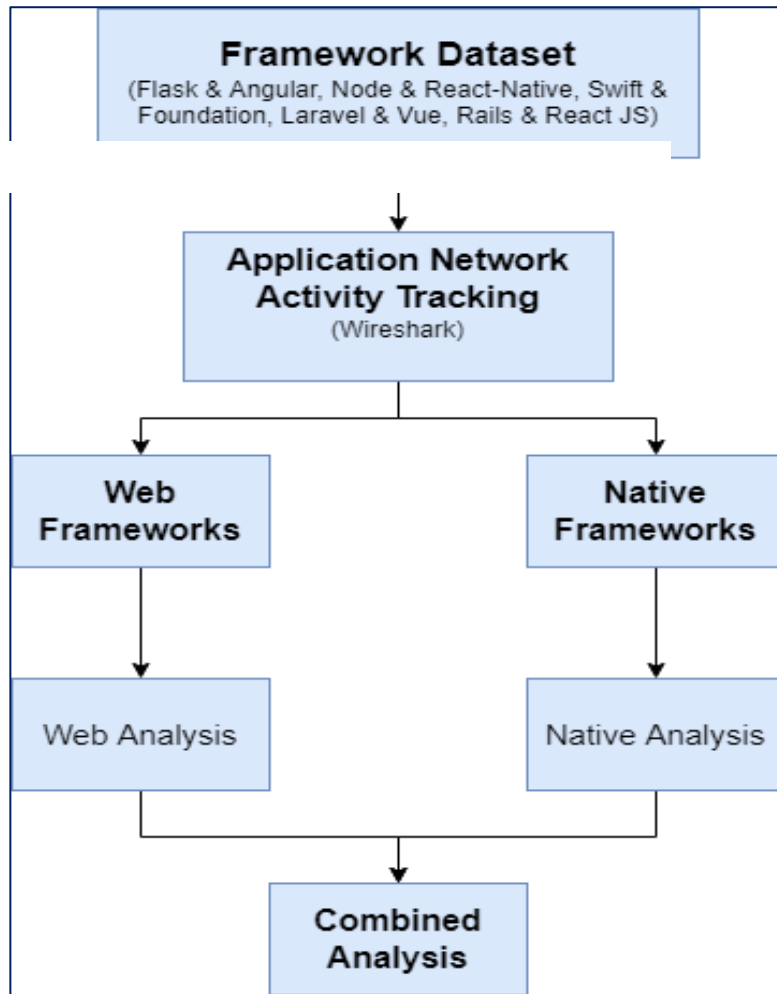


Figure 14. Schema of Study.

3.1 Data Generation

3.1.1 Native and Cross-Platform Applications

The native and cross-platform applications are divided into two categories: Android and iOS. There are different devices and tools to categorize the generated data which are fundamentally explained in the following sections:

Android

For the Android applications, Android emulator is used to generate the dataset. The emulation devices used are Pixel 4 (Android version 11), Pixel 2 (Android

version 10) and a physical handheld device One Plus 8T with Android 12. On these devices 'Packet Capture' application has been installed to block the external data and capture the required packets. The best way to isolate application's network activity is to block all the unnecessary connections and thus only accepting required requests from the dedicated chat server. Hence unwanted traffic generated can be avoided. The native framework uses the WebSocket and TCP packages to send the requested data and receive the response. The generated data was collected using the network packages in PCAP format. The server-side network protocol-stack was captured using the Wireshark application to isolate the TCP, HTTP & WebSocket packets.

The 'Packet Capture' uses the Android's built in VPN functionality to record all the network traffic which makes classification for different applications easier. To run the native chat application, 'Expo Go' was used. The 'Packet Capture' establishes a network interface which then configures the routing rules and provides the application with a descriptor. Each descriptor provides informative analysis on every incoming and outgoing packet on the network interface. The application establishes the local VPN connection with the main server via a tunnel (Figure 15). The captured packets are further sent to the external network and then further analysis can be made.

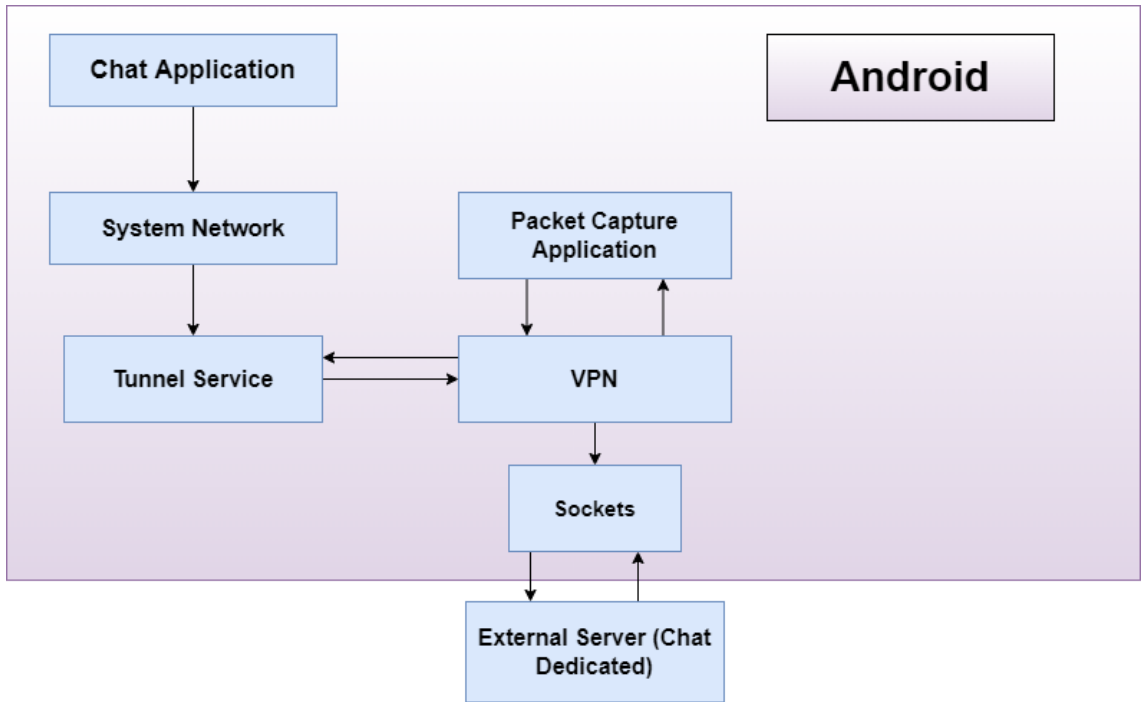


Figure 15. Network Flow – Android.

The packets are captured in the PCAP format which are then analyzed in the Network-Analysis application. (Carstens, n.d.) The packets are graphically represented to provide better analysis of the framework (Figure 16).

```

#1 <--- 05-06 14:25:32
GET /node_modules/expoAppEntry.bundle?
platform=android&web=1&webhook-false&strict=false&entry=false HTTP/1.1
Accept: multipart/mixed
Host: 192.168.0.114:19000
Connection: keep-alive
Accept-Encoding: gzip
User-Agent: okhttp/3.14.9

#2 ---> 05-06 14:25:32
HTTP/1.1 200 OK
X-Content-Type-Options: nosniff
Surrogate-Control: no-store
Cache-Control: no-store, no-cache, must-revalidate, proxy-revalidate
Pragma: no-cache
Expires: 0
Content-Type: multipart/mixed; boundary="bcqj74p9m495jpm4"
Date: Fri, 06 May 2022 11:20:30 GMT
Connection: keep-alive
Keep-Alive: timeout=5
Transfer-Encoding: chunked

47
If you are seeing this, your client does not support multipart response
te

--bcqj74p9m495jpm4
43
X-Web-View-Changed-Count: 0
X-Web-View-ID: c868b0ca42780
Content-Type: application/javascript
Last-Modified: Fri, 06 May 2022 11:21:13 GMT
Content-Length: 634280

607568
var __BUNDLE_START_TIME__=this.nativePerformanceNow?
nativePerformanceNow(Date.now(), _DEV_?true:process.hrtime())
|)_METRO_ID=__METRO_ID__?process.env.METRO_ID:'development';
(function (global) {
'use strict';

global._ = metroBundle;
global[_METRO_GLOBAL_PREFIX_ + "_s"] = define;
global._c = clear;
global._registerSegment = registerSegment;
var modules = clear();
var ENTRY = {};
  
```

Figure 16. Captured Packet Details – Android.

iOS:

For the iOS application, the devices used to capture the network are all emulators which are: iPad (9th Generation with iOS 15.4), iPod touch (7th Generation with iOS 15.4) and iPhone 13 pro (with iOS 15.4). Since all the runs are executed in the emulators, so external network needs no filtration. All the unnecessary requests are filtered by assigning a simple server proxy which is “Localhost” in this case (Figure 17).

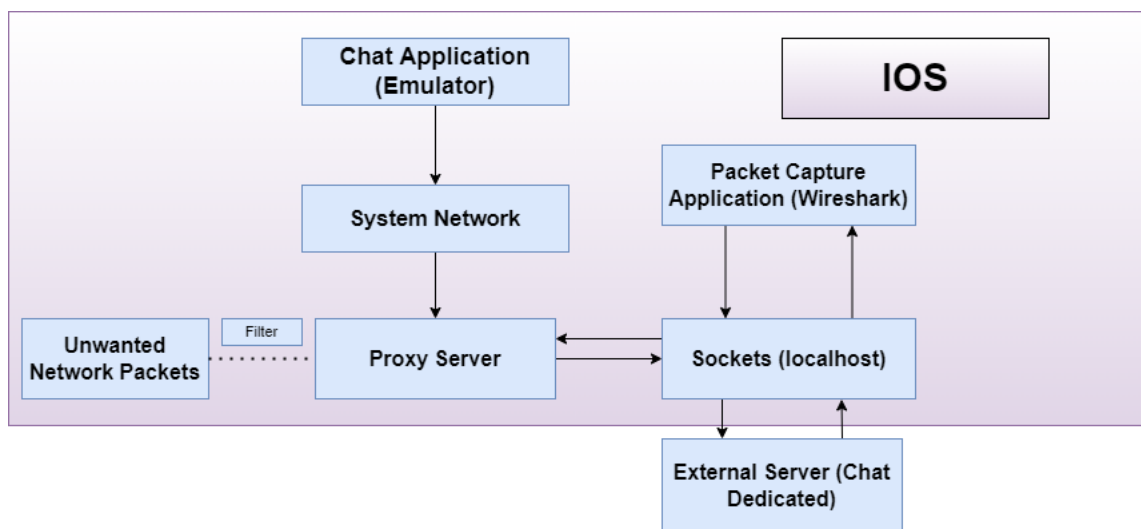


Figure 17. Network Flow - iOS

All the required network packets (HTTP, WebSocket and TCP) were captured and monitored through Wireshark and the network graphs were generated.

3.1.2 Web Applications

The network traffic analysis of web applications is less challenging compared to the native applications. To capture the network data of the web applications on Windows and Mac, a loopback network adapter is used to log all the relevant packets. In this was all the unnecessary background network traffic is restricted.

The testing of Applications on Mac environment is different than those on Windows. To solve the problem a special proxy was defined for Mac environment.

The proxy runs at the “Localhost” server which filters out the unnecessary network packets.

For the Windows application testing, the delayed time taken by constant communication between the Simple Service Discovery Protocol (SSDP) and the localhost has been neglected.

The following equation defines the relationship between the total time taken by the application and the time taken by SSDP during communication.

Equation 1. Server Time Delay – Windows.

$$totalTime_{app_1} = time_{app_1} + time_{communication_{SSDP_1}}$$

Equation 2. Sever Time Delay – Windows.

$$totalTime_{app_2} = time_{app_2} + time_{communication_{SSDP_2}}$$

In all the scenarios, the time taken by SSDP remains a constant, therefore it can be neglected from the final time difference. This gives out a precise measurement of the time difference between the applications which results is better analysis.

$$time_{communication_{SSDP_1}} = time_{communication_{SSDP_2}}$$

Equation 3. Resulting Server Time Difference.

$$\Rightarrow time_{difference} = time_{app_1} - time_{app_2}$$

Wireshark application has been used to extract the required network packets. The application is configured to run over the chat-server network to capture the WebSocket and HTTP traffic. It captures the network packets in the form of “Bits” from the local ethernet adaptor of the system and presents them according to the standard OSI reference model.

The Layer 7 or the Application layer of the OSI model handles the initiation of the requests while the Layer 4 (Transport) and Layer 3 (Network) handle all the network requests over the TCP, HTTP and WebSocket packets (Figure 18).

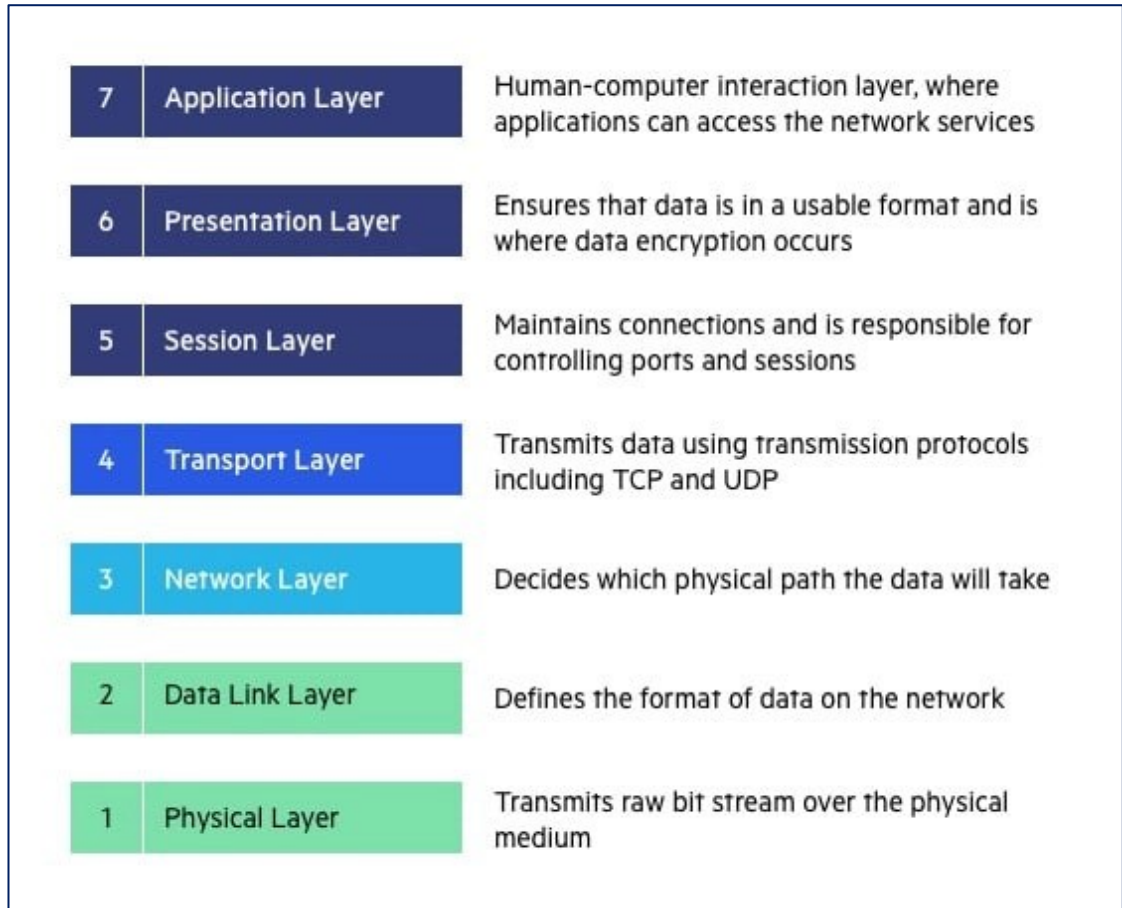


Figure 18. OSI Model (Imperva, 2021).

Methods

Wireshark is used to open the PCAP file which contains the generated packets, and the selected displayed fields are extracted to a Comma-Separated Values (CSV) file. The CSV files are classified into TCP, WebSocket and HTTP requests. All these CSV files contain the 'time taken by the source requests', 'time taken by the destination requests', 'the length of the packets' and 'number of packets per each request'.

The CSV files have the information regarding the number and length of packets generated during a specific time period. The detailed CSV files are plotted as a

graph to have a clear understanding of the comparison between different packets for different applications including native and web applications (Figure 19).

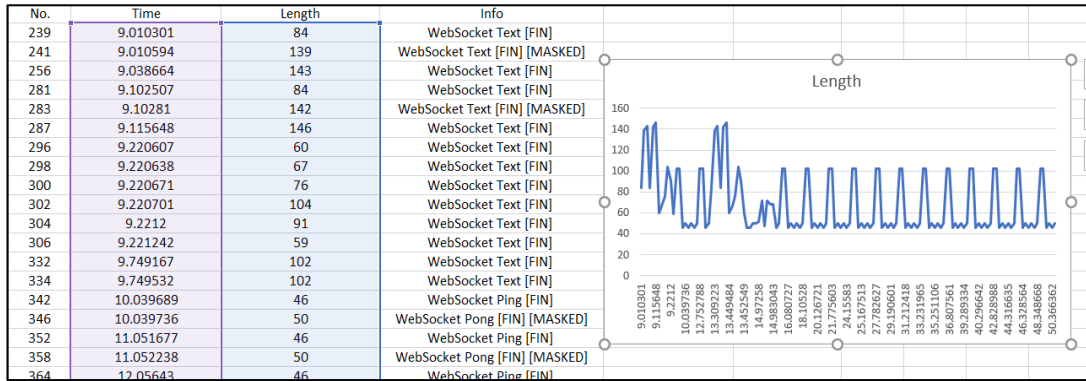


Figure 19. CSV - Generated Chart.

4 IMPLEMENTATION OF THE APPLICATIONS

The objective of this chapter is to provide insight on the implementation of the chat application using the considered frameworks – React JS and Rails, Angular and Flask, Vue and Laravel, Swift with Objective-C, and React-Native with Express. Every chat application is based on only one pattern which includes similarly designed front-end, back-end and the database model. Therefore, the application is similar on all the frameworks.

The chat application deals with text-based communication over the network using Web Sockets. In the application, a text-based chat communication can be set-up between two or more users. A chat room also allows multiple users to message and interact at the same time.

The application is divided into two modules which include a basic login screen and the chat room. This is further explained in the GUI flows of the said application. A GUI flow is a visual representation of the path taken by a user while using that application.

All the data is managed by models running in the back-end while the front-end has controllers which gets the information to display to the user (Figure 20).

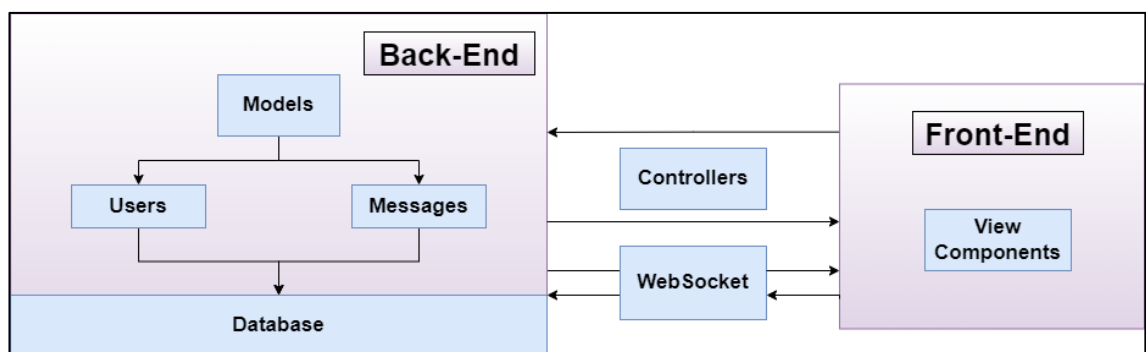


Figure 20. Chat Application Structure.

The data sent by the user to the server passes through the message and user channel which facilitate saving of the information in the database. Similarly, the

data requested by the user is displayed on the front-end via the controllers and the database.

The front-end maintains a continuous connection with the server using Web Sockets. The connection persists until it is closed by any party that is participating in the communication. The communication is said to be over when all the WebSocket packages are transferred through the TCP port and there is no request or response left.

4.1 GUI Flow

The application is divided into two modules. The first screen is a simple login page, where the user is only required to enter the username. The second screen which is a chat screen is only accessible after the login screen.

The chat screen allows communication either between the two users or among all the users in the chat room (Figure 21).

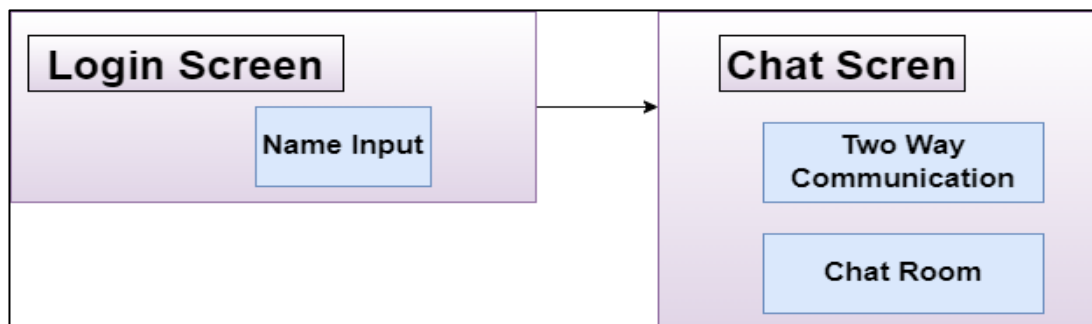


Figure 21. Chat Application - GUI flow.

4.2 Models

Like the GUI flows, the Domain Model is also divided into two classes: User and Message. The user model holds the user schema which defines how the user is added and modified in the database. Similarly, the chat message schema defines the addition and presentation of the chat message (Figure 22).

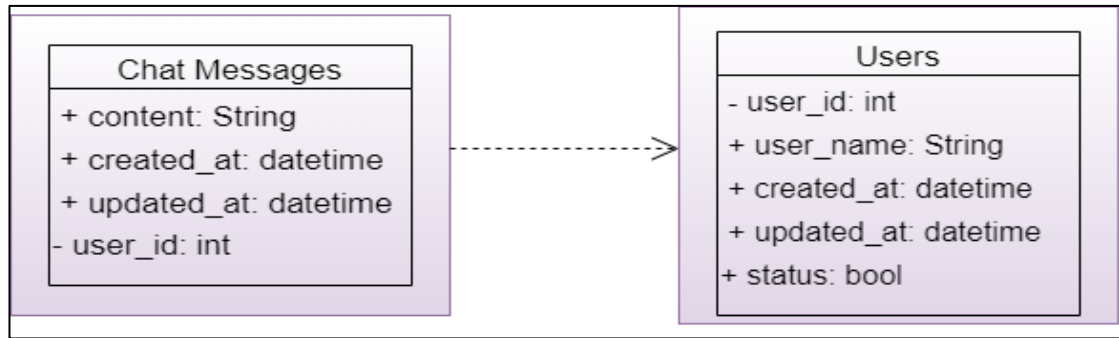


Figure 22. Chat Application - Data Models.

4.2.1 Chat Message Model

In the figure 22, it is clear that the chat message model contains certain parameters like:

Content: This is the body of the chat message. This is the communication message which is sent by the user either to other user or to the chat room for other users to read.

Created/Updated At: These parameters define the date and time of the message. These parameters include the time when the message was created by the user and the time when the message was updated.

User ID: This parameter defines the user of the message. It is directly linked to the user model to present the username with the sent message in the chatroom.

4.2.2 User Model

Like the chat model, the user model contains the parameters which are explained as:

User ID: This is the “ID” parameter for the users. It is linked to the Chat Model acting as a foreign key. It is used to display the messages linked to the user.

Username: This parameter defines the username. The username is presented in the chat room whenever the user sends a message.

Status: This parameter defines the status of the user. It tells the other users if the said user is active or not.

Created/Updated At: These parameters define the date and time of the message. These parameters describe the timestamp of the message i.e., the time of the creation of message and when the message was updated by the user.

4.3 Web Applications

All the web chat applications follow the similar idea of implementation. Apart from the framework dependencies, every application has been installed with a WebSocket dependency and a CORS dependency (Cross-Origin Resource Sharing) as well. The CORS is a HTTP protocol-based header. This allows a server to load resources to any foreign domain to load resources other than the permitted local domain.

4.3.1 React JS and Rails

The first chat application is developed using the front-end framework – React JS and the back-end framework Rails. React is based on the programming language JavaScript while Rails emerged from the language Ruby. Similar to the other implemented applications, this one is too based on figure 21 GUI flow and figure 22 database model.

Front-End

For the said technologies in consideration, the main application is divided into multiple modules. Before the application is build, the navigation between multiple modules is implemented.

After testing the navigation, the application is divided into multiple components. These components include - Chat Module, Message Module, User List Module and User Module. All these components work together to form a navigation stack to build the UI of the application which is perceived by the user.

As explained by the figure 23, it clearly depicts that a user needs to login in first before it can join a chat. The 'currentUserLoggedIn' component determines the accessibility of the chat screen.

```
{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 className="">
        <Messages />
      </div>
      <div>
        <Entry />
      </div>
    </div>
  </div>
```

Figure 23. React JS - User Login.

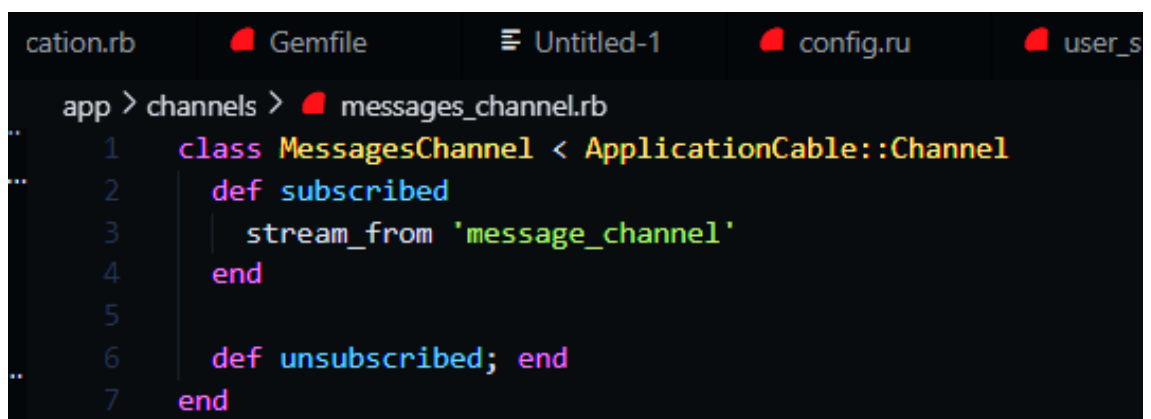
Only the authorized users are given access to navigate through the chat and message screen where they can also communicate and interact with each other. After the login screen, the user is navigated to the chat screen. The chat screen allows the communication between the users.

Back-End

The back-end of this application is built using Rails. In the initial stage, a development environment is set up where the application can run. After the initial stage, the installation of dependencies required for the application takes place.

The front-end and back-end are connected using WebSocket. All the client's requests and responses are send/received through WebSockets. These requests are then passed to the server which are stored in the database. PostgreSQL is used as the database for this case.

The 'ApplicationCable' module in figure 24 helps in integration of WebSocket in the application. The connection is further broadcasted and streamed on the application which allows multiple users to interact without facing any port problems.

A screenshot of a code editor with a dark background. The editor shows a file named 'messages_channel.rb' located in the 'app > channels' directory. The code defines a class 'MessagesChannel' that inherits from 'ApplicationCable::Channel'. It includes two methods: 'subscribed' which calls 'stream_from' with the argument 'message_channel', and 'unsubscribed' which is an empty method definition. The code is as follows:

```
..
1  class MessagesChannel < ApplicationCable::Channel
..
2    def subscribed
3      stream_from 'message_channel'
4    end
5
6    def unsubscribed; end
..
7  end
```

Figure 24. Rails - Broadcasting Channel.

Working Example

The chat screen displays the active users and the communication channel (Figure 25).

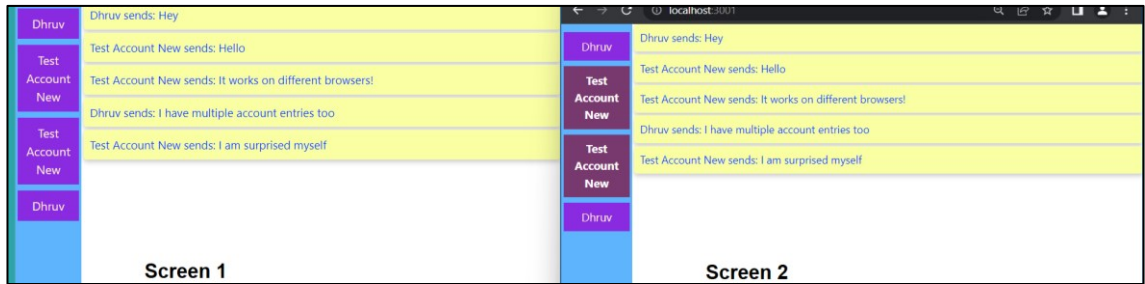


Figure 25. React - Working Chat Application.

The server also logs into the console which helps in debugging the connection between the client and the server. The console displays every message ranging from user registration to messages (Figure 26).

```

TRANSACTION (0.2ms) BEGIN
app/controllers/users_controller.rb:16:in `add_message'
Message Create (0.8ms) INSERT INTO "messages" ("content", "created_at", "updated_at", "user_id") VALUES ($1, $2, $3, $4) RETURNING "id" [
ted_at", "2022-05-17 06:48:22.635119"], ["user_id", 1]]
app/controllers/users_controller.rb:16:in `add_message'
TRANSACTION (11.5ms) COMMIT
app/controllers/users_controller.rb:16:in `add_message'
[onCable] 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)

```

Figure 26. React - Chat Application server logs.

Testing

The testing includes two tests. The 'Conn Report' test-case checks for the status which should be 200. This status means that the response by the server is 'OK'.

The 'Response Add' test checks if the sent data is received by the server without any data errors.

From the figure 27, it is clear that the application has passed both the tests in the server test application (postman in this case).

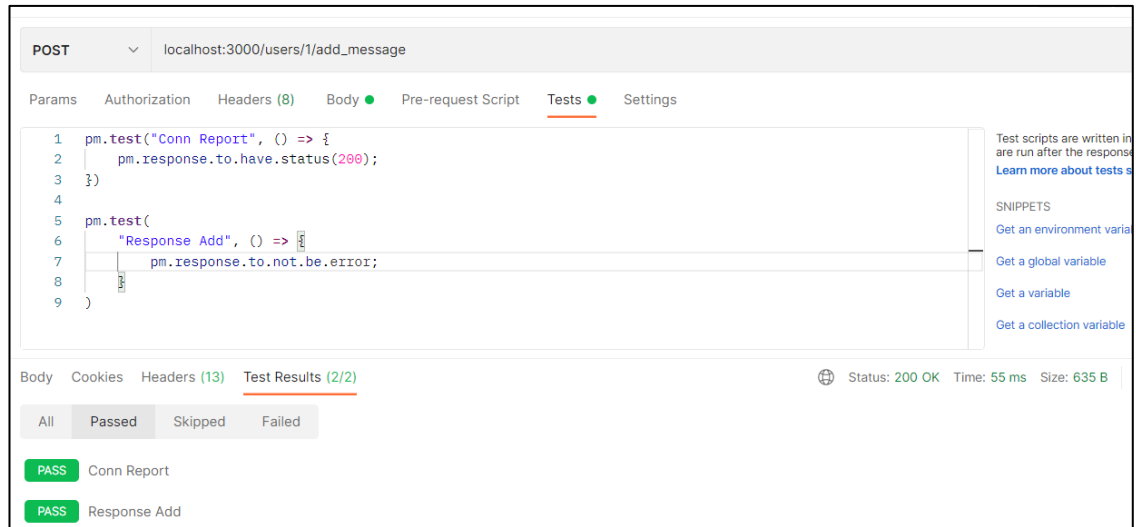


Figure 27. React - Chat Application Postman Test.

4.3.2 Angular JS and Flask

The second chat application was developed with front-end framework Angular and the back-end framework Flask. The application follows the figure 21 GUI flow and figure 22 database model.

Front-End

It follows the same initialization steps as described in previous case.

As seen in the figure 28, the path 'chat' in the Routes section has a parameter 'canActivate'. This parameter has the value AuthGuard which forces the user to log-in the application before it can use the chat functionality.

```

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 },
];

```

Figure 28. Angular – Login.

Back-End

The back-end of the application in this case is coded using Flask. In the initial step a virtual environment is prepared so that application can run and tested. The preparation of the environment is soon followed by the installation of the dependencies of the application.

In order to link with the front-end, a stable connection using the WebSocket. The requests and responses correspond to the state of the WebSocket. If the state of the WebSocket is connected, then the connection between the client and the server is stable (Figure 29).

The application also keeps tracks of the requests and the responses received and sent by the server. The console messages help in debugging of the application while at the same time providing an insight to the application's requests.

```
10
11     users = {}
12     @websocket.on('connect')
13     def on_connect():
14         print('Client connected')
15         websocket.emit('my response', {'data': 'Connected'})
16
17     @websocket.on('disconnect')
18     def on_disconnect():
19         users.pop(request.sid, 'No user found')
20         websocket.emit('current_users', users)
21         print("User disconnected!\nThe users are: ", users)
22
```

Figure 29. Flask - WebSocket Connection.

Working Example

The chat screen displays the active users and the possibility to communicate with them (Figure 30).

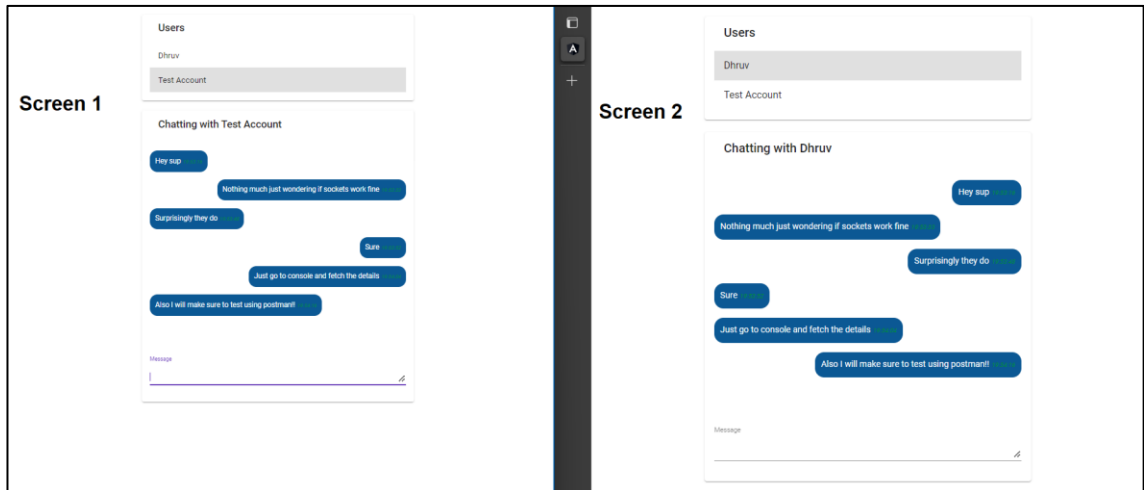


Figure 30. Angular – Working Chat Application.

The console messages provide the visual concept of the back-end functionality. The first event describes the stabilized connection between the client and the server. Following the connection, a new user is registered to the server. The console displays the ID and the name of the user.

Before getting disconnected from the server, the user's messages are also echoed to the console. This provides an insight to the user's message content and other attributes related to the message like date, time, sender and receiver (Figure 31).

```

Windows PowerShell
* Restarting with stat
* Debugger is active!
* Debugger PIN: 103-680-792
Client connected
New user sign in!
The users are: {'q9rkiDYLTaLucwELAAAB': 'Dhruv Verma'}
received message: {'message': 'df', 'to': 'q9rkiDYLTaLucwELAAAB', 'date': '2022-04-25T14:32:01.723Z'}
Client connected
New user sign in!
The users are: {'q9rkiDYLTaLucwELAAAB': 'Dhruv Verma', 'DgpyCX4R5-SdZkIxAAAD': 'Public'}
received message: {'message': '\nYo', 'to': 'DgpyCX4R5-SdZkIxAAAD', 'date': '2022-04-25T14:32:24.632Z'}
received message: {'message': 'Hi', 'to': 'q9rkiDYLTaLucwELAAAB', 'date': '2022-04-25T14:32:29.526Z'}
received message: {'message': '\ner', 'to': 'DgpyCX4R5-SdZkIxAAAD', 'date': '2022-04-25T14:32:59.460Z'}
received message: {'message': '\nef', 'to': 'DgpyCX4R5-SdZkIxAAAD', 'date': '2022-04-25T14:33:21.341Z'}
received message: {'message': '\ner', 'to': 'q9rkiDYLTaLucwELAAAB', 'date': '2022-04-25T14:34:06.883Z'}
received message: {'message': '\n3ew\\', 'to': 'q9rkiDYLTaLucwELAAAB', 'date': '2022-04-25T14:34:08.910Z'}
received message: {'message': 'rt\\n', 'to': 'q9rkiDYLTaLucwELAAAB', 'date': '2022-04-25T14:34:12.973Z'}
User disconnected!

```

Figure 31. Angular - Chat Application server logs.

Testing

The testing includes two tests 'Response Add' and 'Conn Report'. The former checks for any errors in the application requests and responses while the later checks for the application status. The application passed both tests with an 'OK' response.

4.3.3 Vue JS and Laravel

The third chat application is implemented using the front-end framework – Vue JS and the back-end framework Laravel. Like other front-end frameworks, Vue is also based on the programming language JavaScript while Laravel is developed from the language PHP. Like the other implemented applications, this one is also based on figure 21 GUI model and figure 22 database model.

Front-End

After the application is initialized as the project, it is also divided into different couple components. These components include - Chat component and Main component. The first step is to implement the main login screen for the users and then the development of the chat screen begins.

Before initializing any other functionality, navigation components are developed. These components will help the user navigate through the login screen to the chat screen.

As seen in the figure 32, the components chat includes two components: 'mainLoginComponent' and 'ChatsComponent'. These components describe the routes in way which determines that the chat screen is not accessible without logging in the application first.

```

// 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'));

```

Figure 32. Vue - Login Components.

Back-End

The back-end of this application is coded using Laravel. The initial step is to prepare a local server for the application to run. The local server is prepared using Xampp application (Figure 33).

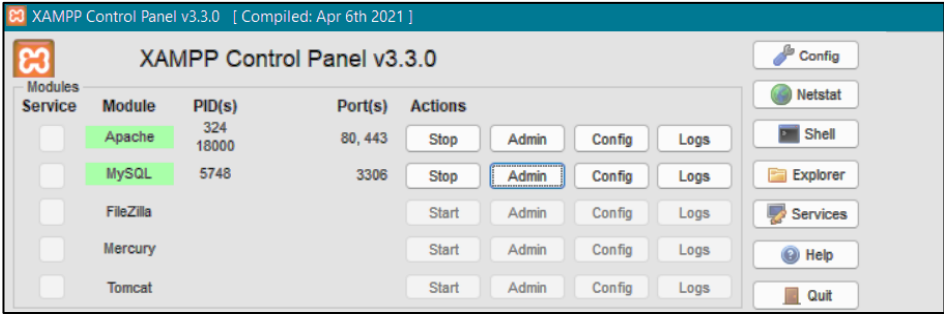


Figure 33. Xampp - Apache server & SQL Database.

In the figure 33, the Apache and MySQL services are turned on. These services provide the opportunity to run the server and the database locally.

The pusher API (Figure 34) module in Laravel helps in initializing and broadcasting the messages using WebSocket. The controllers are configured to broadcast the chat channel on any origin, thus solving the CORS issue. The messages and the users are then saved to MySQL database which follows the same data model as discussed previously.

```
12     'apps' => [  
13         [  
14             'id' => env('PUSHER_APP_ID'),  
15             'name' => env('APP_NAME'),  
16             'key' => env('PUSHER_APP_KEY'),  
17             'secret' => env('PUSHER_APP_SECRET'),  
18             'enable_client_messages' => true,  
19             'enable_statistics' => true,  
20             'encrypted' => true,  
21         ],  
22     ],
```

Figure 34. Laravel - Pusher API.

Working Example

All the activities including requests and responses are logged into the console which helps in keep tracking of the WebSocket connection. The console data includes user creation, user events (receiving and sending messages) and user details (Figure 35).

Data	Length	Time
u2807 {"event":"pusher:connection_established","data":{"socket_id":"254156557.793394874","activity_timeout":"30"}}	114	13:46:26.130
u2806 {"event":"pusher:subscribe","data":{"auth":"anyKey-2db56a147bb6562d7d727f2efff1425d94396eb4b3e24c86d9dd7e90aee2672","channel_data":{"user_id":"3","user_info":{"id":"3","name":"Test ...	363	13:46:26.541
u2807 {"event":"pusher:internalsubscription_succeeded","channel":"presence-chat","data":{"presence":{"ids":["1","1","1","1","1","3"],"hash":{"1":"1":"1","name":"Dhruv Verma"},"email":"dhruv...	523	13:46:26.542
u2806 {"event":"pusher:ping","data":{}}	33	13:46:56.561
u2807 {"event":"pusher:pong"}	23	13:46:56.562
u2806 {"event":"pusher:ping","data":{}}	33	13:47:26.567
u2807 {"event":"pusher:pong"}	23	13:47:26.568
u2806 {"event":"pusher:ping","data":{}}	33	13:47:56.579
u2807 {"event":"pusher:pong"}	23	13:47:56.581
u2806 {"event":"pusher:ping","data":{}}	33	13:48:26.595
u2807 {"event":"pusher:pong"}	23	13:48:26.596
u2806 {"event":"pusher:ping","data":{}}	33	13:48:56.606

Figure 35. Vue – Chat Application Server Logs.

The chat screen displays active users and the possibility to communicate with them (Figure 36).

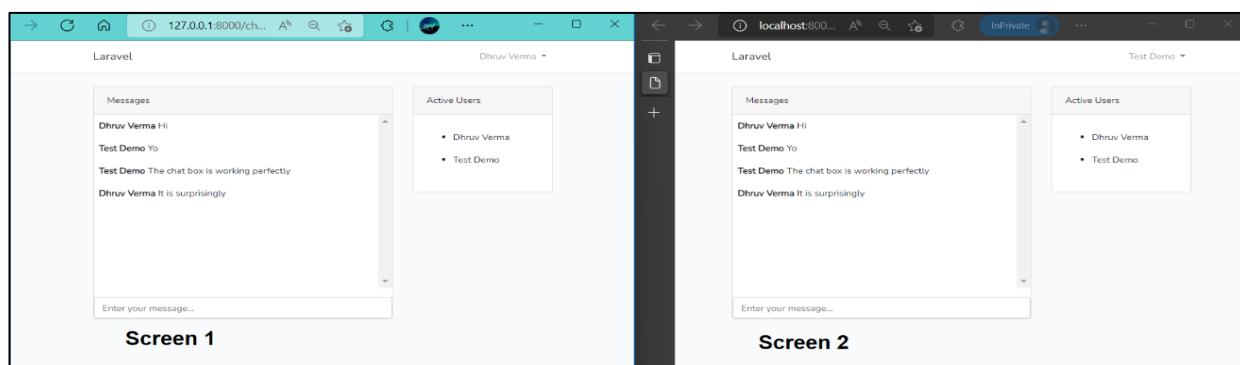


Figure 36. Vue - Working Chat Application.

Testing

The testing includes two tests. A post request is sent to the server with message in a JSON format. The 'Conn Report' test checks for the response status to be 200 which translates to an 'OK' response. The application received the 'OK' status response.

4.4 Native and Cross-Platform Applications

Native applications are solely implemented for the mobile devices depending on the OS. However, cross-platform applications are developed accordingly to run on multiple-platforms such as both iOS and Android. These chat applications follow the same ideology of communicating using WebSocket. Apart from the framework dependencies, these implemented applications have been installed with a WebSocket dependency with an UI dependency for the client-side.

4.4.1 Swift

The fourth chat application is developed using the libraries of the framework Swift. Based on the programming language Objective-C, this framework designs the applications for the iOS platform. The front-end is developed using the SwiftUI library to provide a lively experience to the user. The back-end is based on the Foundation library of the Swift framework. Similar to the web applications, this

native app is also implemented using figure 21 model and figure 22 database model.

Front-End

Similar to the previous cases, the project developed in this case is also divided into two components which include: Login component and Chat component. Firstly, the main screen is developed where the user can log in. After this the navigation link is implemented.

The 'navigationLink' (Figure 37) component navigates the user from the login screen to the chat screen. Before implementing the chat component, the view is initialized to test out the navigation.

```
4 //
5 // Created by Dhruv Verma
6 //
7
8 import SwiftUI
9
10 struct SettingsScreen: View {
11     @EnvironmentObject private var userInfo: UserInfo
12
13     private var isValidUsername: Bool {
14         userInfo.username.trimmingCharacters(in: .whitespaces).isEmpty
15     }
16
17     var body: some View {
18         Form {
19             Section(header: Text("Username")) {
20                 TextField("E.g. Dhruv Verma", text: $userInfo.username)
21
22                 NavigationLink("Continue", destination: ChatScreen())
23                     .disabled(!isValidUsername)
24             }
25         }
26     }
27 }
```

Figure 37. Swift UI Components.

Back-End

The first step is to prepare a socket connection for the application to run. The socket connection is regularly pinged every 10 seconds in order to check if the connection between the client and the server is stable. (Figure 38)

```

// 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)

```

Figure 38. Swift - WebSocket Connection.

The data is saved into the local database after defining the models for the Chat and user components. To secure the application, all the data is hashed (Figure 39) before passing through the WebSocket connection and to the database.

```

8 import Foundation
9 import WebSocketKit
0
1 extension WebSocket: Hashable {
2     public static func == (lhs: WebSocket, rhs: WebSocket) -> Bool {
3         ObjectIdentifier(lhs) == ObjectIdentifier(rhs)
4     }
5
6     public func hash(into hasher: inout Hasher) {
7         hasher.combine(ObjectIdentifier(self))
8     }
9 }
0

```

Figure 39. Swift - Data Hashing.

Working Example

The user requests and the server responses are logged to test and debug correctly. These logs help in keep track of the data in the WebSocket connection. The terminal outputs the data which includes user login and user chat messages (Figure 40).

```
0.0, m02: 0.0, m03: 1.0)
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)
```

Figure 40. Swift - Chat Application Server Logs.

The chat screen displays active users and the possibility to communicate in a chat room. The working demo is implemented on three different iOS simulators to verify the multi-device compatibility of the native application (Figure 41).

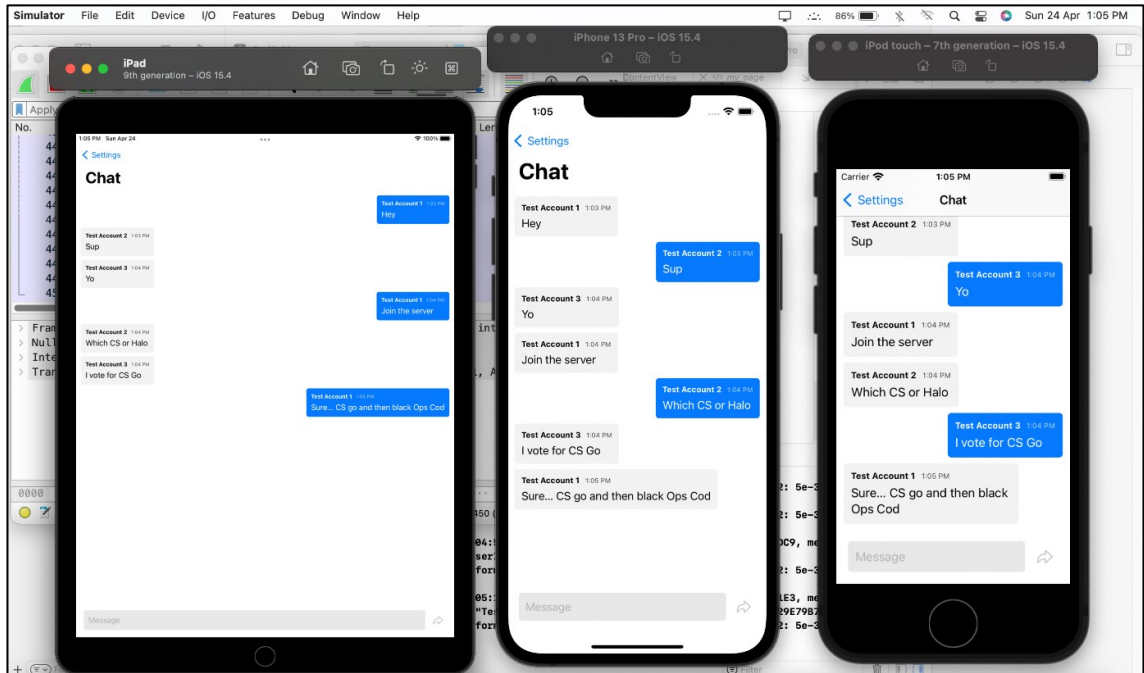


Figure 41. Swift - Working Chat Application.

Testing

The testing of the native application is different than the other web applications. Swift framework does not allow the access of the local server using third-party tools such as postman. Therefore, the connection was tested manually using the console of the safari browser (Figure 42).

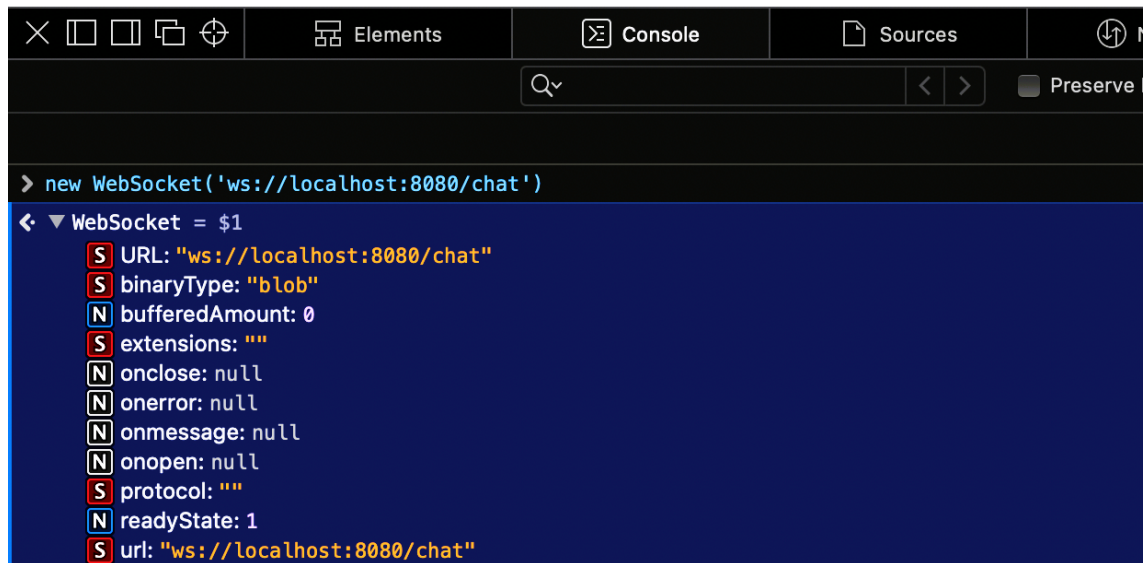


Figure 42. Swift - Manual Connection Test.

4.4.2 React-Native and Express

Lastly, this application is developed using the front-framework React-Native and the back-end framework Express. Both the front-end and back-end frameworks are based on JavaScript. Like the other implemented applications, this one also follows figure 21 UI model and figure 22 database model.

Front-End

After the project is initialized, it is also bifurcated into two components: User component and Chat component. The user component handles the login functionality, and the chat component handles the message functionality. In this

development case, the main screen is developed along with navigation stack. The main screen then initializes the login component and after that the chat component (Figure 43).

```
const Stack = createStackNavigator()

const MyStack = () => {
  return(
    <Stack.Navigator>
      <Stack.Screen
        name="LoginPage"
        component={LoginPage}
        options={{ headerShown: true, title: "Join the Chat" }}
      />
      <Stack.Screen
        name="Chat"
        component={Chat}
        options={{ headerShown: false }}
      />
    </Stack.Navigator>
  );
}

export default function App() {
  return(
    <NavigationContainer>
      <MyStack />
    </NavigationContainer>
  );
}
```

Figure 43. React-Native Navigation Stack.

Back-End

The first step is to create the server for the application is to implement the WebSocket connection. The socket connection is checked for stability. The figure 44 shows the code for the connection between the client-side and the server-side.

```
const WebSocket = require('ws');
const server = http.createServer(app);
const websocketServer = new WebSocket.Server({server});

websocketServer.on("connection", function connection(websocket) {
  websocket.on("message", function incoming(message, isBinary) {
    console.log(message.toString(), isBinary);
    websocketServer.clients.forEach(function each(client) {
      if (client.readyState === WebSocket.OPEN) {
        client.send(message.toString());
      }
    });
  });
});
```

Figure 44. React-Native WebSocket Connection.

Working Example

The user messages are logged into the console. These logs help in keep track of the data in the WebSocket connection. The terminal outputs the data which includes username and user chat messages. (Figure 45)

```
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
█
```

Figure 45. React-Native Chat Application Server logs.

As seen in the figure 46, it is clear that two users join the chat room by logging in. All the shared messages and the respective usernames are displayed in the console. The text messages are shared between the connected users.

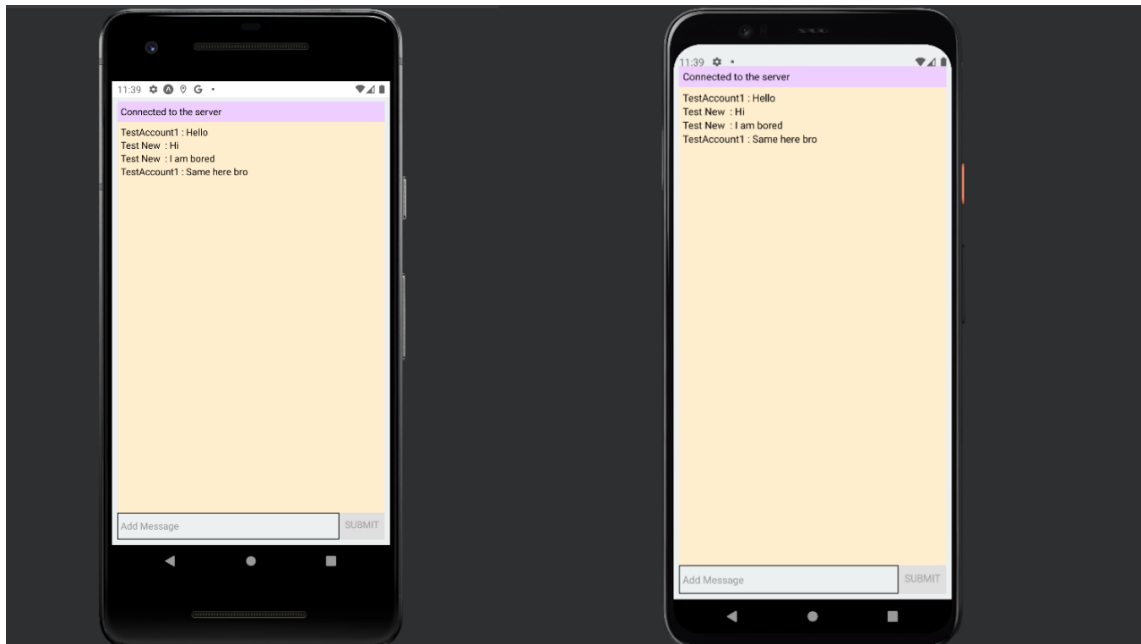


Figure 46. React-Native Working Chat Application.

Testing

The testing of the Android application is relatively easier compared to the iOS testing. The windows platform allows Android emulators to run postman tests on the server-side. Similar to other frameworks, the react-native application also received a status 'OK' response.

5 COMPARISON MODEL FOR THE FRAMEWORKS

Since every web development project has different requirements compared to the other, it is not easy to determine the proper comparison while choosing a web framework. Therefore, an attempt has been made to construct a proper model for this research which tries to include some important aspects of the frameworks.

This chapter concerns with the comparison between all the frameworks that were used to implement the chat applications. The results will be discussed in the later chapter.

5.1 Benchmarks

All the web and native applications have been compared according to the following benchmarks. These parameters were considered from the well-known benchmarking study by TechEmpower (Techempower, 2021)

5.1.1 Development and Ease of Modification

It is very necessary to consider the total-time spent on developing and the ease of modifying the applications. A 'web development' survey (Overflow, Stack, 2021) provides clear benchmarks for all the frameworks. The benchmarks include creating, debugging, and testing of the project.

5.1.2 Ease of Deployment

This creation describes the ease with which the application can be deployed to the server (including the front-end and the back-end). This benchmark depends on the components, the logic, and the database environment of the application. Some frameworks require little work for deployment while others may take a lot of time, therefore it is important to consider while comparing the frameworks (Figure 47).

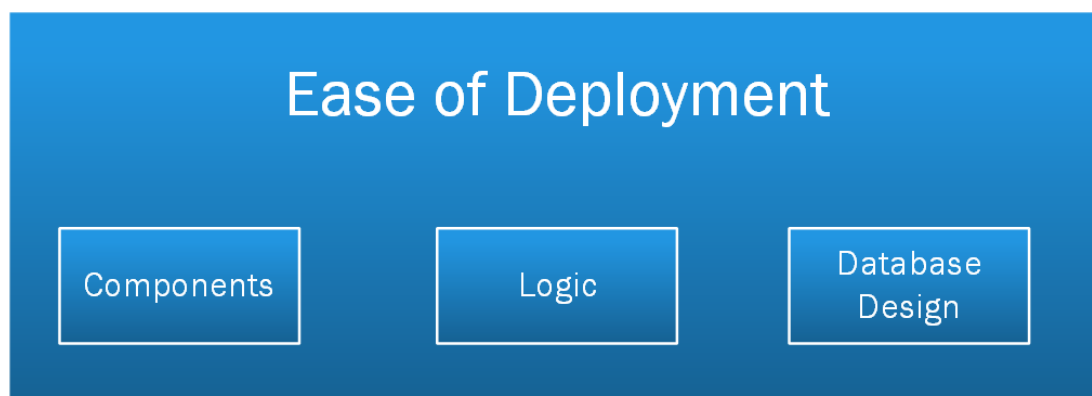


Figure 47. Application Deployment Criteria.

5.1.3 Generated HTML Structure

The amount of generated html code represents the flexibility and speed of the frameworks. A large amount of generated HTML structure means that the application must render a large amount of data and structure which reduces the performance. Hence, the amount of generated HTML code by the web application is inversely proportional to the time spent waiting for the webpage to load.

An application's generated HTML structure usually includes various JavaScript and metadata. These values are unique to each framework; therefore, the generated HTML size varies. The size of the structure is measure in Kilobytes (KB) per page.

5.1.4 Framework Performance

This criterion describes the performance of a framework and its Command Line Interface (CLI). The performance includes the starting time, request and response time, and WebSocket initialization time of the application. The higher time a framework takes, the lower is its performance score. If the framework performance is low, then the applications build on these frameworks performs slower. Since the applications do not perform well with a stable connection, hence browser's resources are wasted, and it takes a long time to load the webpages.

5.1.5 Corresponding to modern standards

It is very important for websites to follow modern website development standards. These standards include HTML 5, CSS 3, and JavaScript coding standards (ES6 coding standard). Modern browsers support these standards. Therefore, these frameworks are required to follow the standards as well for better compatibility.

If the development standards are poorly followed, it will result in weak generated HTML structure which in result will slow down the application and thus reduce performance.

All the considered benchmark criteria are sorted in priority order. The priorities are set according to the notion of predefine concepts. All the benchmarks add up to a total sum of 100 points. With respect to the web development technologies, two of major criterion like 'Development and Modification' and 'Performance of the Framework' are prioritized above every else, each having 45 and 30 points respectively. The complete benchmark criteria are defined in the table 2.

Table 2. Framework - Benchmark Criteria (Percentage).

Measurement Criteria	Measured Importance (in Percentage)
Development and Ease of Modification	45
Framework Performance	30
Ease of Deployment and Implementation	15
Corresponding to today's Standards	10

5.2 Frameworks

5.2.1 Angular JS and Flask

Network Analysis

Network analysis of this framework includes tracking of the HTTP, TCP and WebSocket request/responses.

Tracking HTTP Packets: The figure 48 shows the number of HTTP Packets sent and received for the interval of 100 seconds. The high amount of HTTP traffic flow leads to a constant communication between the client(s) and the server. It means that the client must render HTML and JavaScript constantly without breaking the connection. Total HTTP Packets received at the 100th second are around 1630.

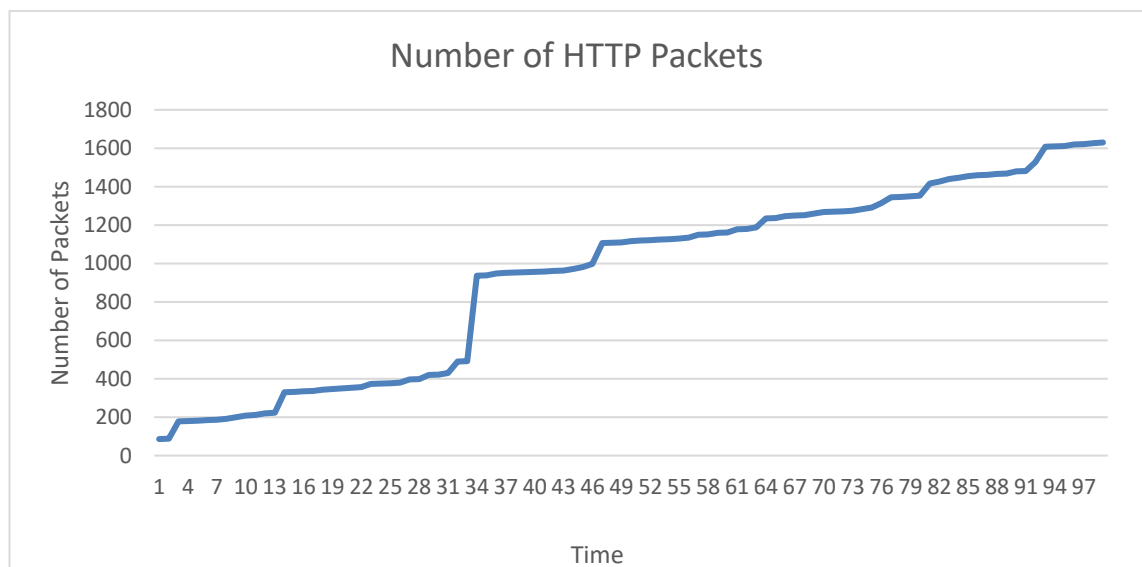


Figure 48. Angular-Flask HTTP Packet Numbers.

The figure 49 describes the length of HTTP Packets sent and received for the given timeframe (100 seconds). The length of the HTTP Packets is measured in bytes. It can be inferred that the size of rendered HTML pages is quite big as the maximum length of the HTTP Packet recorded is around 61006.

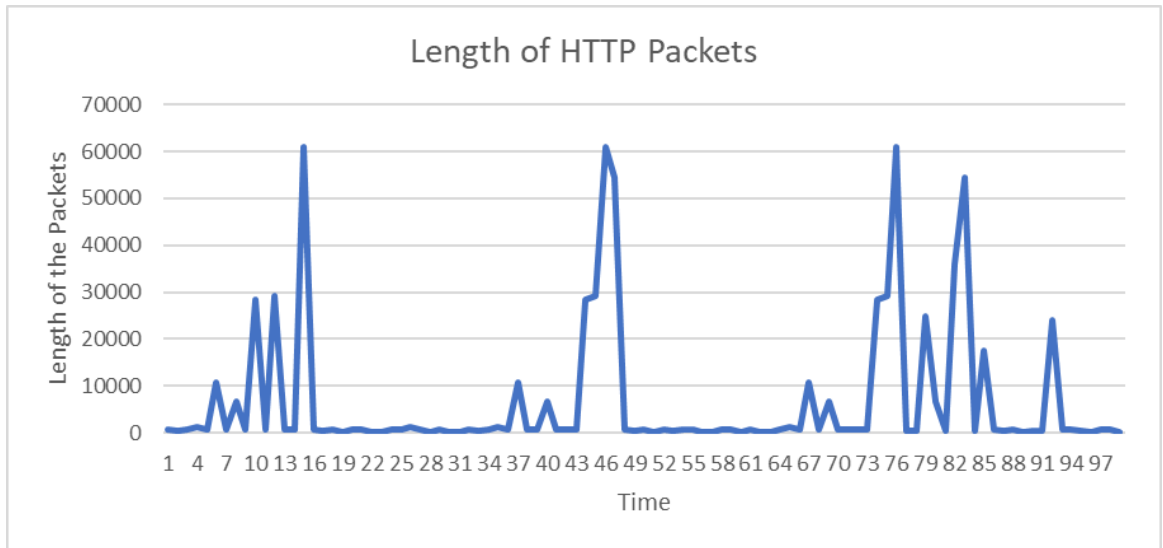


Figure 49. Angular-Flask HTTP Packet Length.

Tracking TCP Packets: The figure 50 shows the number of TCP packets sent and received for 100 seconds. The high number of TCP packets received informs that the framework's network design is weak. The maximum amount of the TCP packet recorded is 540.

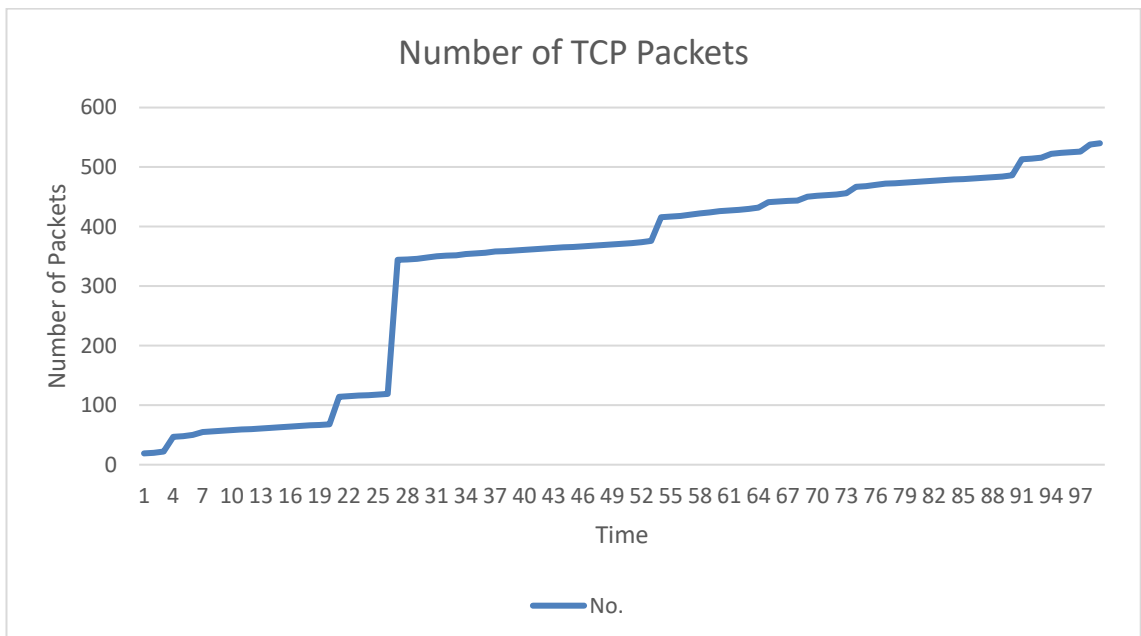


Figure 50. Angular-Flask - No. of TCP Packets.

The figure 51 describes the length of TCP packets (measured in bytes) sent and received till 100th second. The length of the TCP packets signifies the flow of data within the application. The constant lower scaling of the TCP packets results in stable traffic. The maximum length of the TCP packet achieved in this application is 253 bytes.

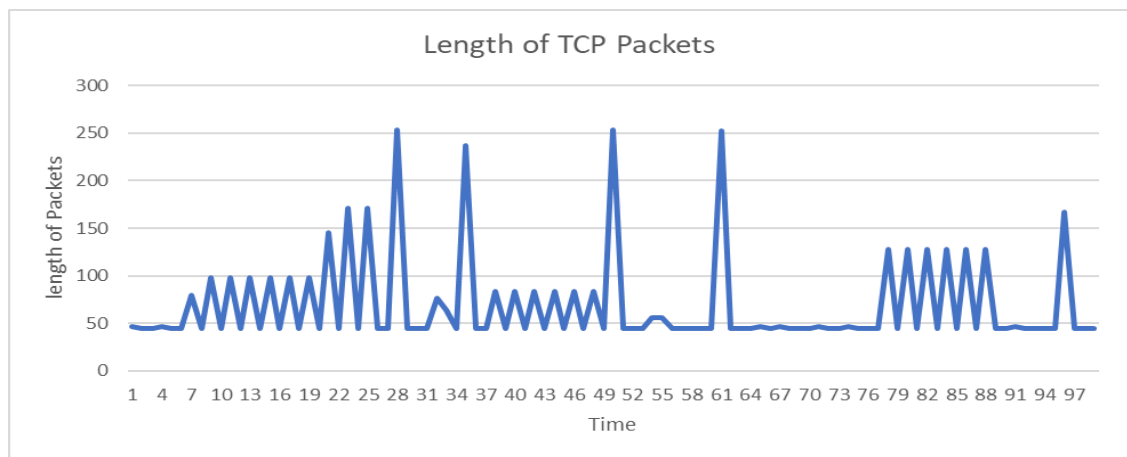


Figure 51. Angular-Flask - Length of TCP Packets.

A high variation in TCP flow is witnessed between 1-25 seconds.

Tracing WebSocket: The WebSocket has two states either open or closed. Moreover, it follows the concept of masking. A masked WebSocket is secure which means no data can be breached, it means that the given combination of IP address and port are masked.

The figure 52 shows the number of WebSocket data sent and received for 100 seconds. The amount of data sent and received over a set period of time signify the delay in time taken by the application server. The amount of WebSocket data is inversely proportional to the instability in connection. For the application in consideration, the amount of WebSocket data is 2590.

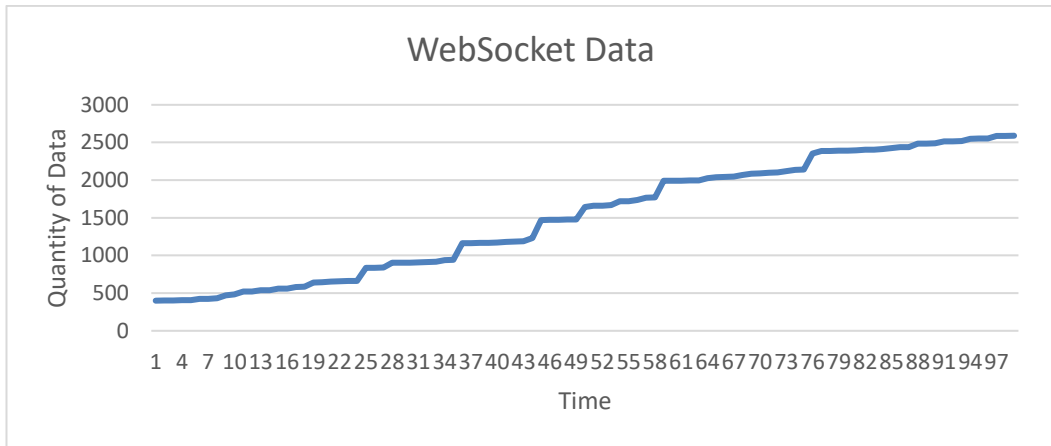


Figure 52. Angular-Flask WebSocket Data Quantity.

The figure 53 describes the length of WebSocket data sent and received till 100th second. The length of the WebSocket data is approximately the same for all the applications. The messages sent in all the applications are the same, thus the length of the WebSocket data sent and received is approximately equal. For this application, the average length of the data is 111 bytes.

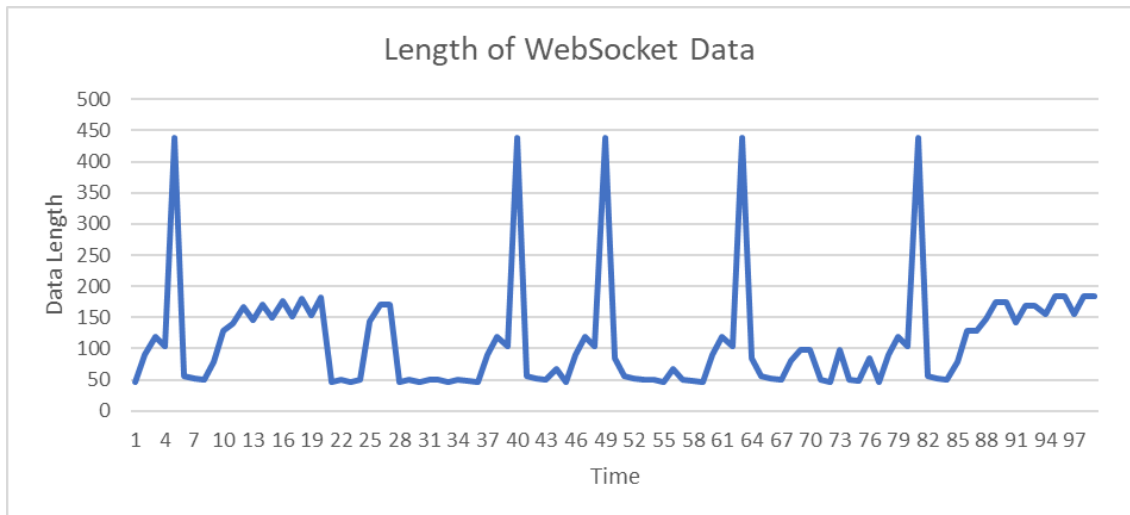


Figure 53. Angular-Flask WebSocket Data Length.

The figure 54 displays the queue and stalling time for the initial WebSocket connection. The framework's performance falls as the WebSocket connection

time increases. The queue time for the WebSocket connection in this application is 5.47 milliseconds while the stalled time is 0.20 milliseconds.

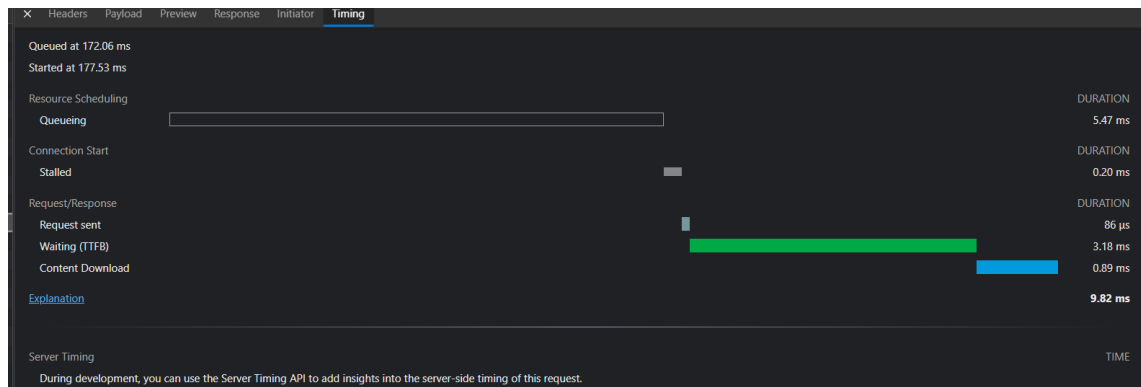


Figure 54. Angular-Flask WebSocket Stalling Time.

The figure 55 shows the waiting time for the request delivered by the WebSocket connection. Waiting time is described as the time taken by WebSocket connection to request the first byte of the data or Time to First Byte (TTFB). Similar to the stalling time, the waiting time is also inversely related to the performance of the framework. The waiting time for this application is 562 milliseconds.

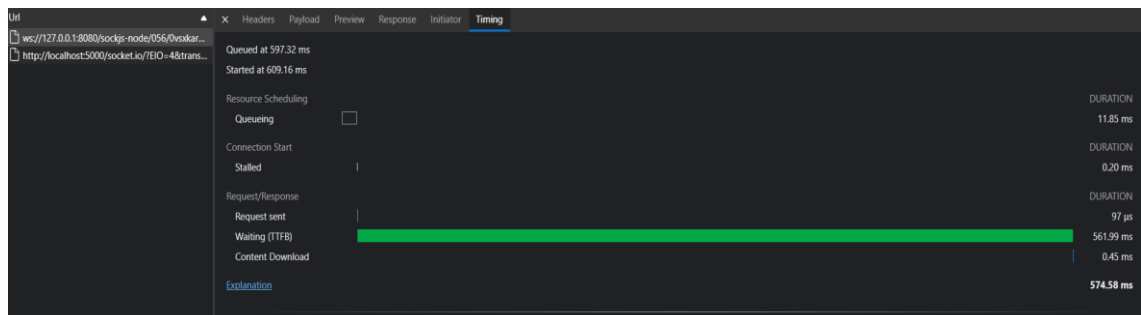


Figure 55. Angular-Flask WebSocket TTFB.

Performance Analysis

The performance of the application is tested using the 'Inspect Element' of the browser. The browser helps in generating a Lighthouse report which analyzes the

application. The performance analysis includes the speed index, interactive time, blocking time and layout shift of the application (Figure 56).

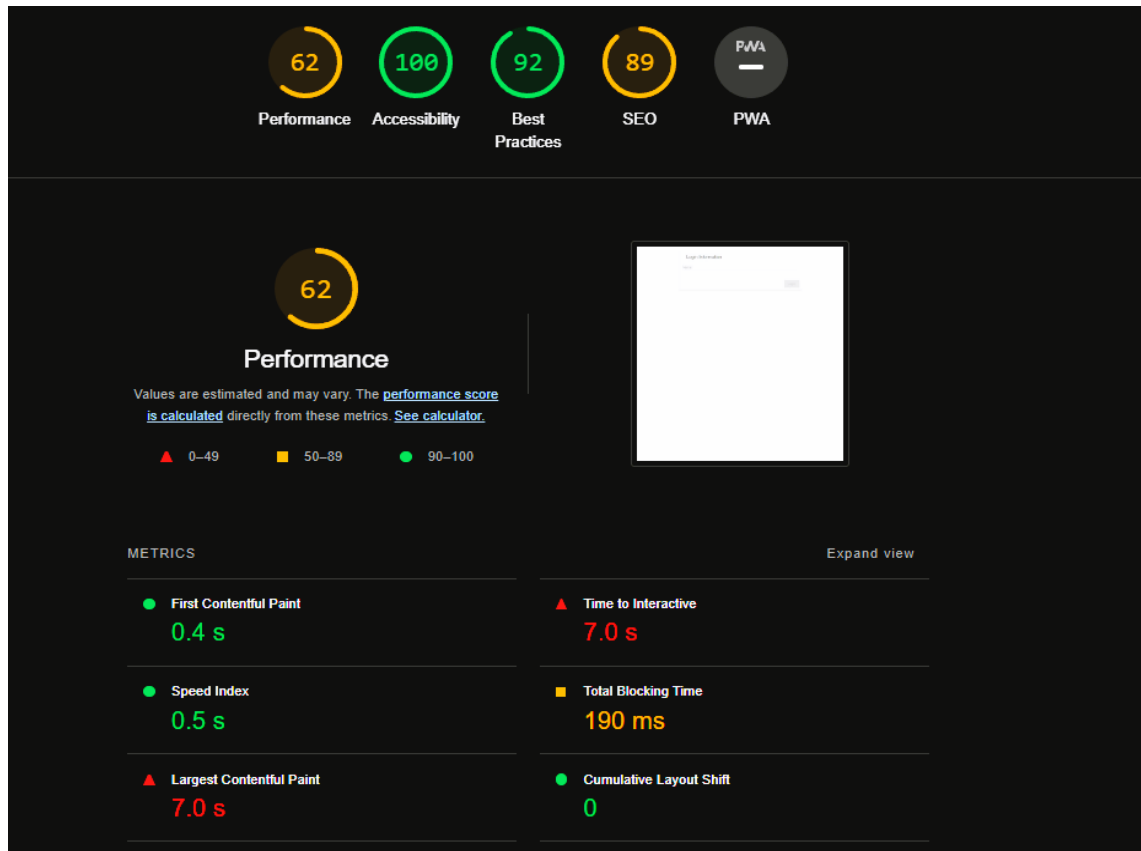


Figure 56. Angular-Flask Application Performance.

5.2.2 React JS and Rails

Network Analysis

Like other applications, network analysis of this framework includes analyzing HTTP, TCP and WebSocket requests and responses which are sent and received by the server.

Tracking HTTP Packets: The figure 57 displays the number of HTTP packets that are delivered and received within 100 seconds. The amount of HTTP packets for this case reaches 3109.

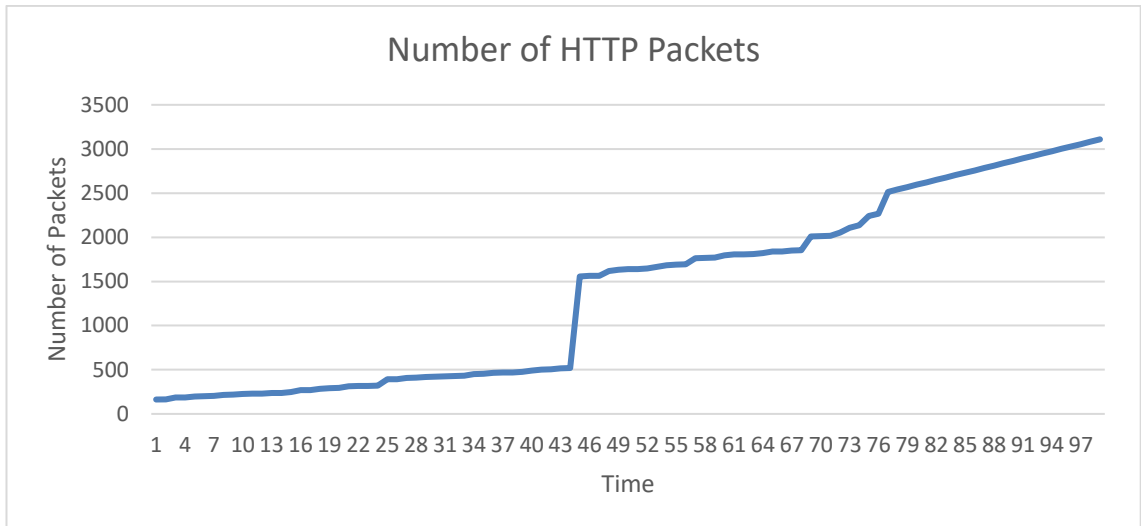


Figure 57. React-Rails Number of HTTP Packets.

The figure 58 describes the length of HTTP packets sent and received in 100 seconds. The length of the HTTP packets is measured in bytes. For this case, the maximum length of a HTTP packet is about 946 bytes.

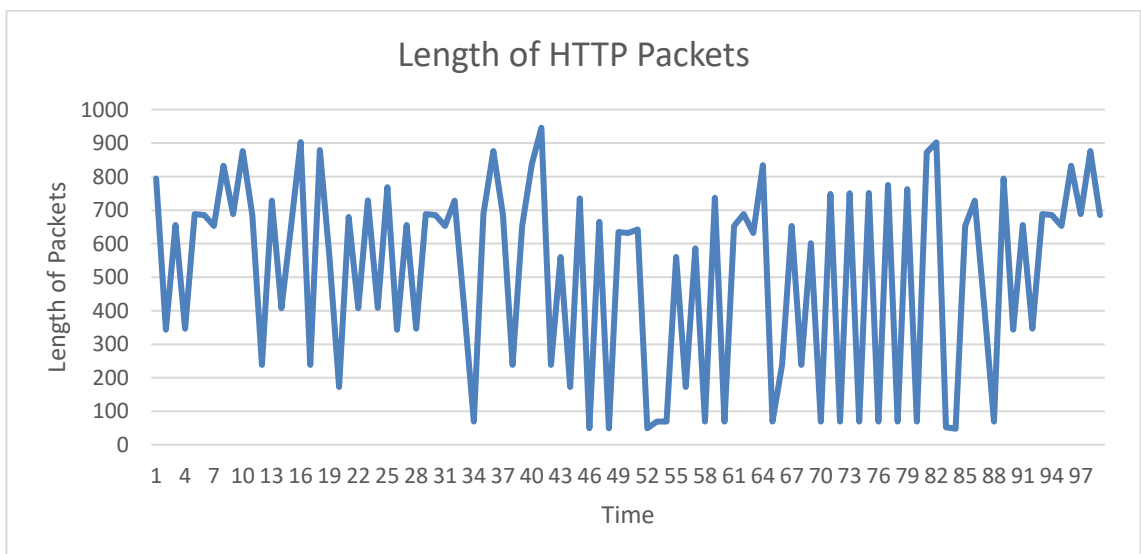


Figure 58. React-Rails Length of HTTP Packets.

Tracking TCP Packets: The figure 59 shows the number of TCP packets sent and received for 100 seconds. The maximum number of the TCP packets recorded for the application in consideration is 1780.

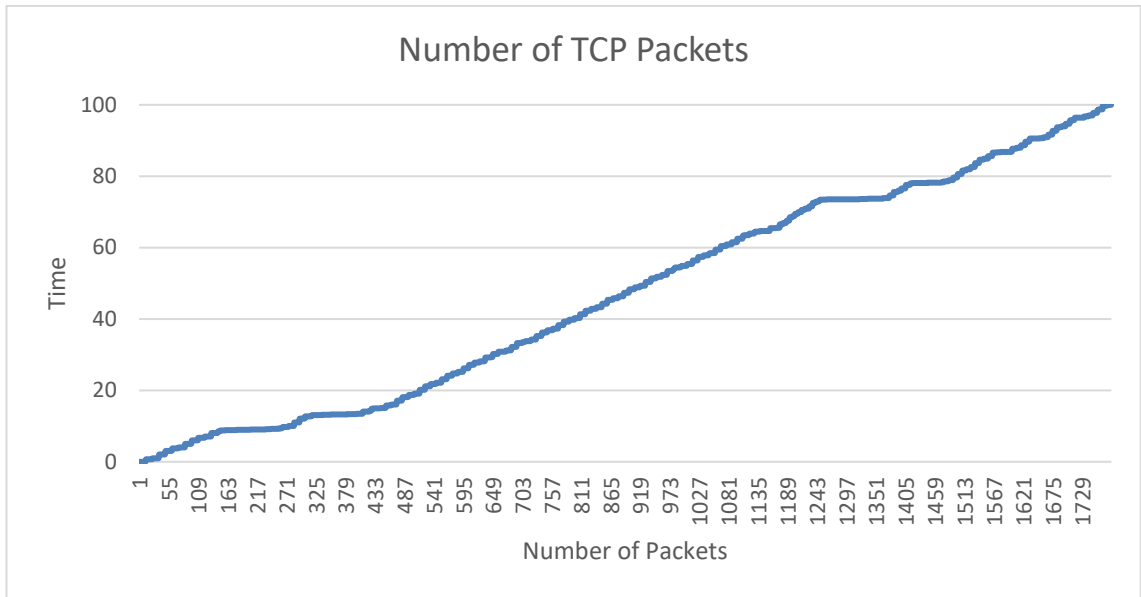


Figure 59. React-Rails Number of TCP Packets.

The figure 60 describes the length of TCP packets (measured in bytes) sent and received till 100th second. The maximum length of the TCP packet achieved in this application is 102 bytes.

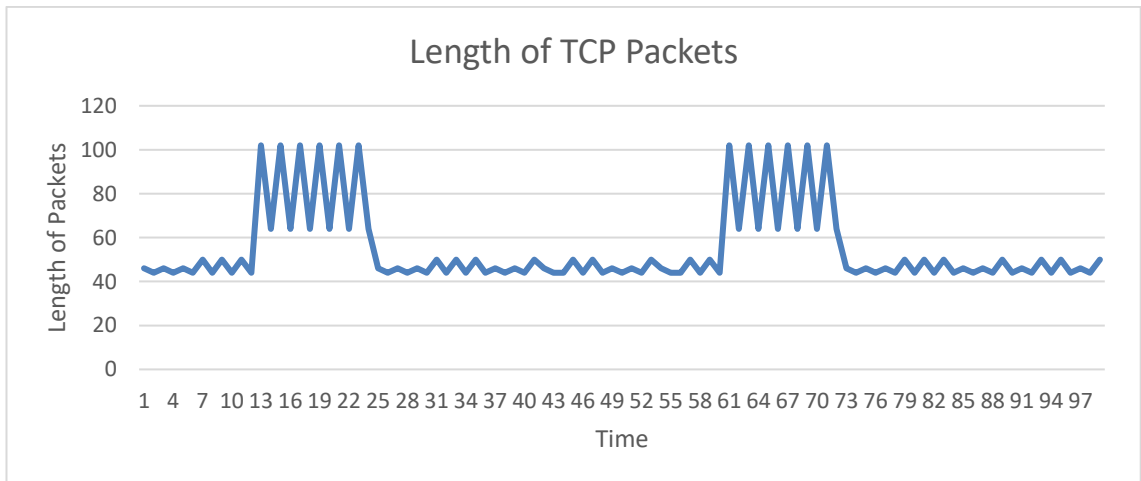


Figure 60. React-Rails Length of TCP Packets.

Tracing WebSocket: The figure 61 shows the number of WebSocket data sent and received for 100 seconds. For the application in consideration, the amount of WebSocket data tracked is 910.

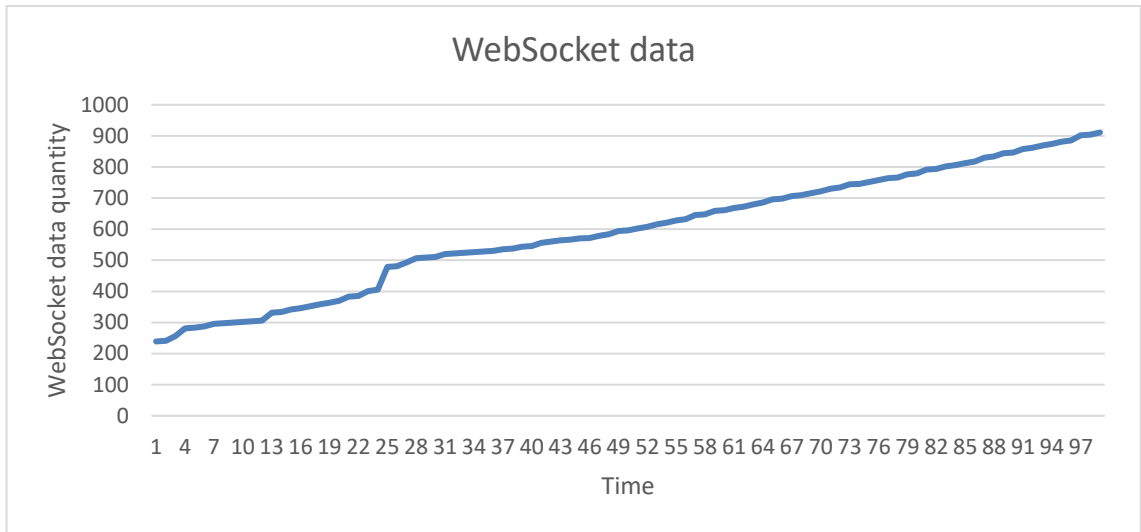


Figure 61. React-Rails Quantity of WebSocket Data.

The figure 62 table describes the length of WebSocket data sent and received till 100th second. For this application, the average length of the data is 109.7 bytes.

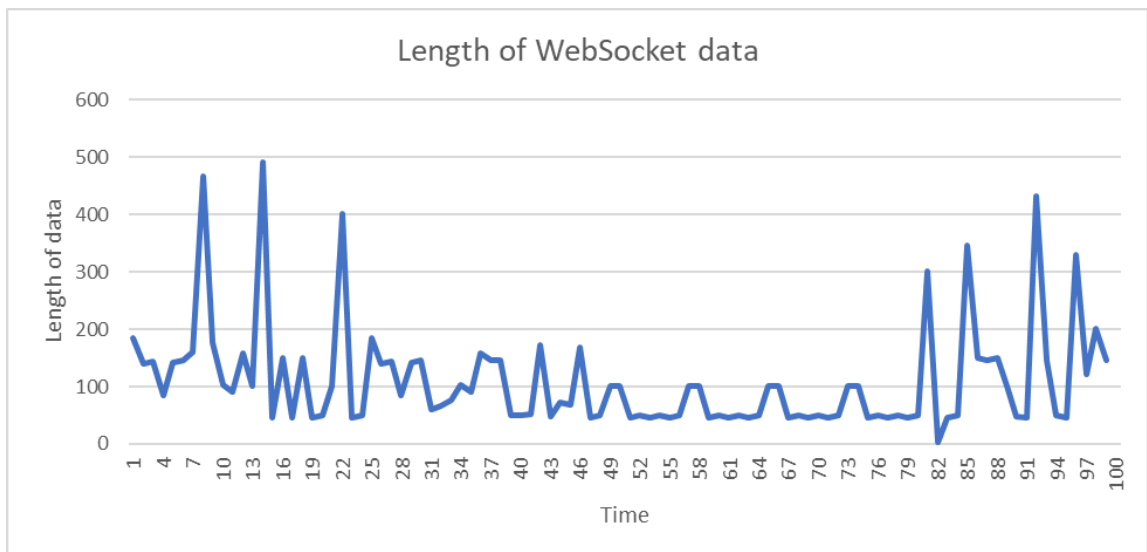


Figure 62. React-Rails Length of WebSocket Data.

The figure 63 displays the queue time and stalled time for the application. In this case the queue time for the WebSocket connection is 16.30 milliseconds while the stalled time is 60.41 milliseconds.

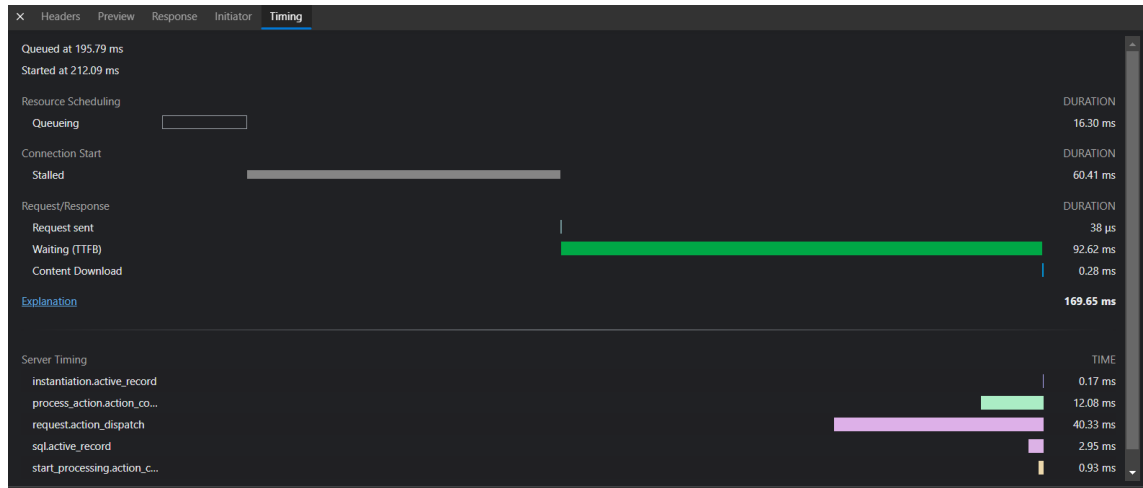


Figure 63. React-Rails WebSocket Stalling Time.

Moreover, it also shows the waiting time for the request delivered by the WebSocket connection for this application which is 92.6 milliseconds.

Performance Analysis

The performance of this application is also measured using the 'Inspect Element' of the browser. The Lighthouse report is generated for analyzing the application. The analysis includes the speed index, interactive time, blocking time and layout shift of the application (Figure 64).



Figure 64. React-Rails Chat Application Performance.

5.2.3 Vue JS and Laravel

Network Analysis

This test case also includes the tracking of the HTTP, TCP and WebSocket requests and responses in a given period of time.

Tracking HTTP Packets: The figure 65 displays the number of HTTP packets that are delivered and received within 100 seconds. The amount of HTTP packets for this case reaches 11018.

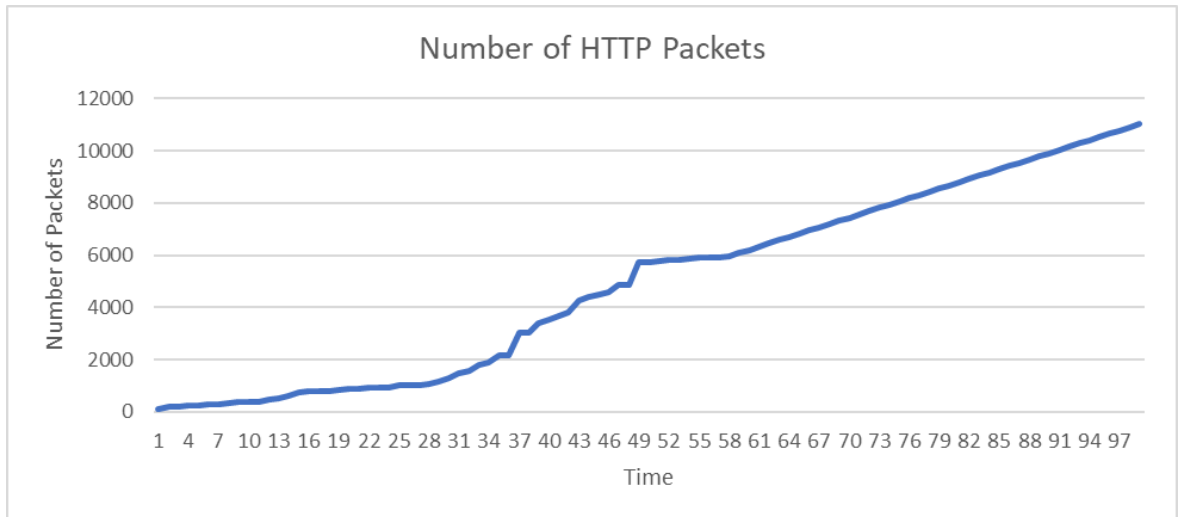


Figure 65. Vue-Laravel Number of HTTP Packets.

The figure 66 describes the length of HTTP packets sent and received in 100 seconds. For this case, the maximum length of a HTTP packet is about 57477 bytes.

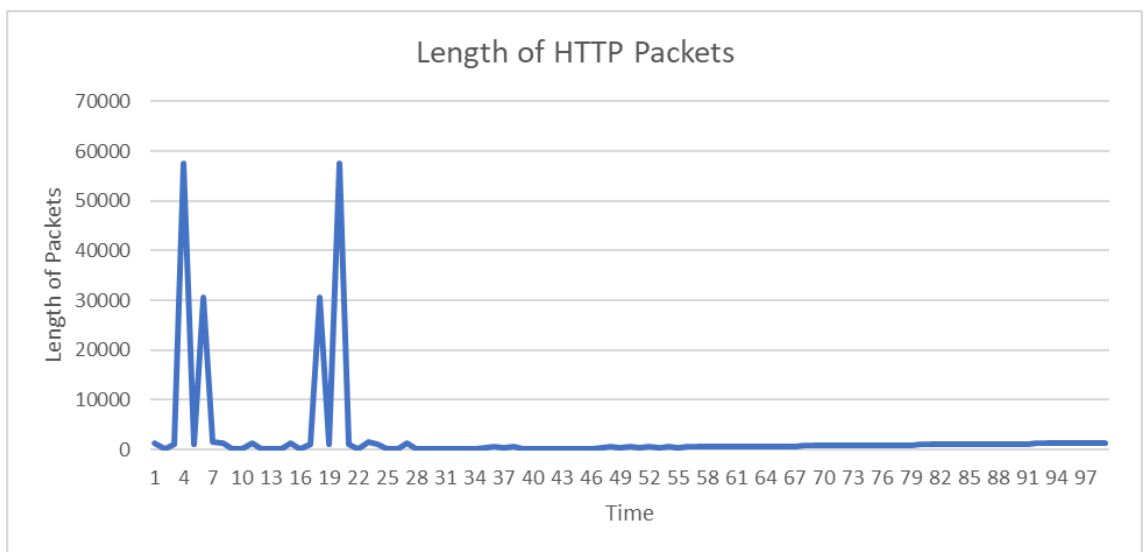


Figure 66. Vue-Laravel Length of HTTP Packets.

A big downfall in the HTTP flow traffic was witnessed after 21 seconds.

Tracking TCP Packets: In Laravel based application, the tracing of TCP packets is done on X11 port. All the TCP requests are forwarded to this port; therefore,

the network is analyzed on X11 port. The figure 67 shows the number of TCP packets sent and received for 100 seconds. The maximum number of the TCP packets recorded for the application in consideration is 3613.

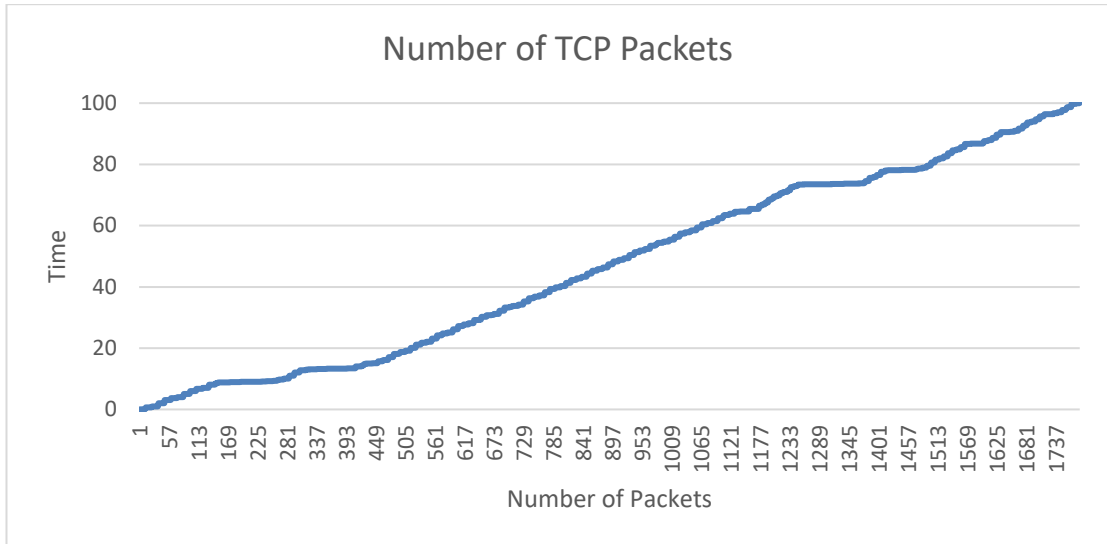


Figure 67. Vue-Laravel Number of TCP Packets.

The figure 68 describes the length of TCP packets sent and received in 100 seconds. The maximum length of the TCP packet achieved in this application is 628 bytes.

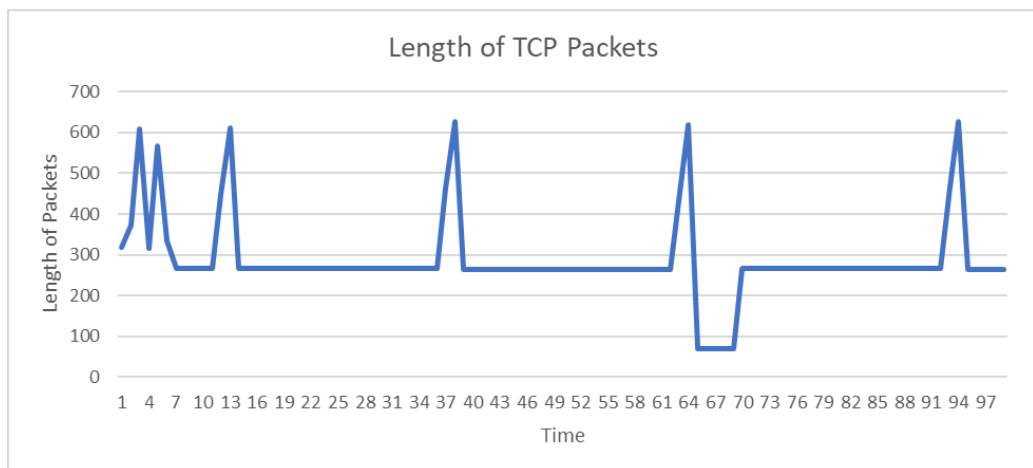


Figure 68. Vue-Laravel Length of TCP Packets.

Tracing WebSocket: The figure 69 shows the number of WebSocket data sent and received for 100 seconds. For the application in consideration, the amount of WebSocket data tracked is 2710.

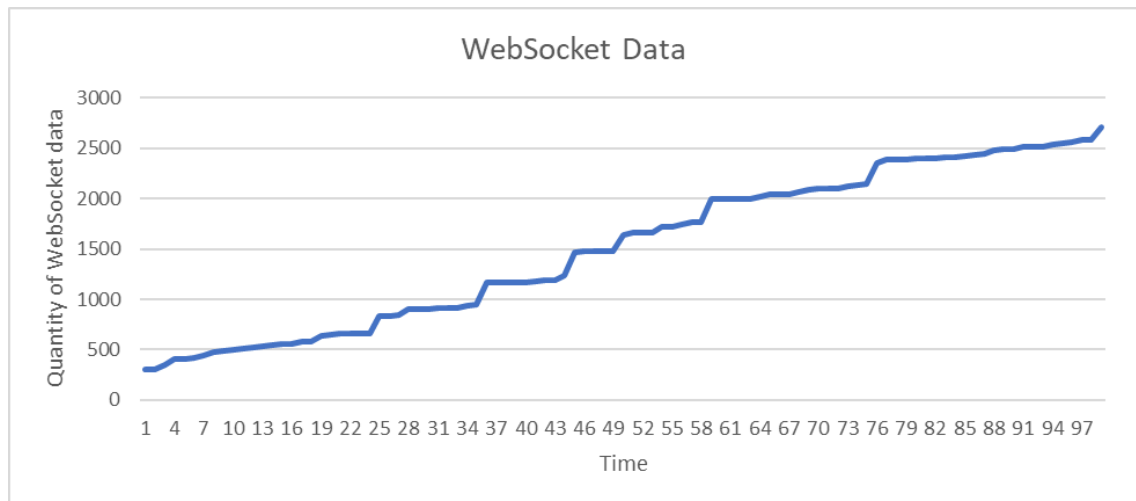


Figure 69. Vue-Laravel Quantity of WebSocket Data.

The figure 70 describes the length of WebSocket data sent and received till 100th second. For this application, the average length of the data is 111.5 bytes.

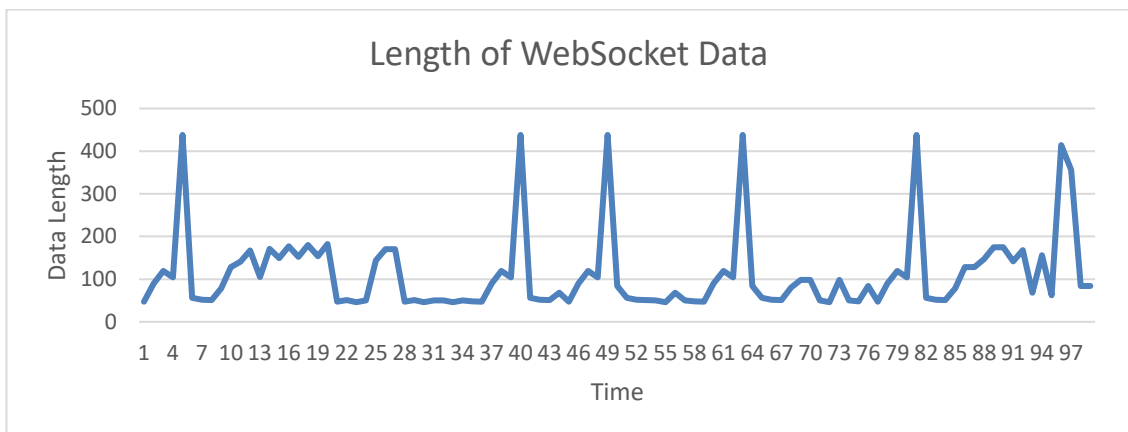


Figure 70. Vue-Laravel Length of WebSocket Data.

The figure 71 displays the queue time and stalled time for the application. In this case the queue time for the WebSocket connection is 2.74 milliseconds while the stalled time is 0.28 milliseconds.

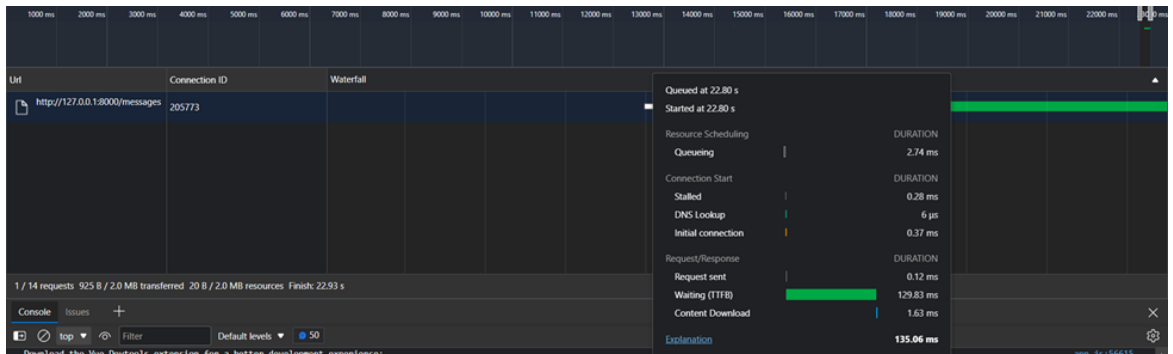


Figure 71. Vue-Laravel WebSocket Stalling Time.

Additionally, it also shows the waiting time which is 129.8 milliseconds in this case.

Performance Analysis

Similar to other web frameworks, this performance analysis is also done with the 'Inspect Element' of the browser. The generated Lighthouse report includes the speed index, interactive time, blocking time and layout shift of the application. (Figure 72)

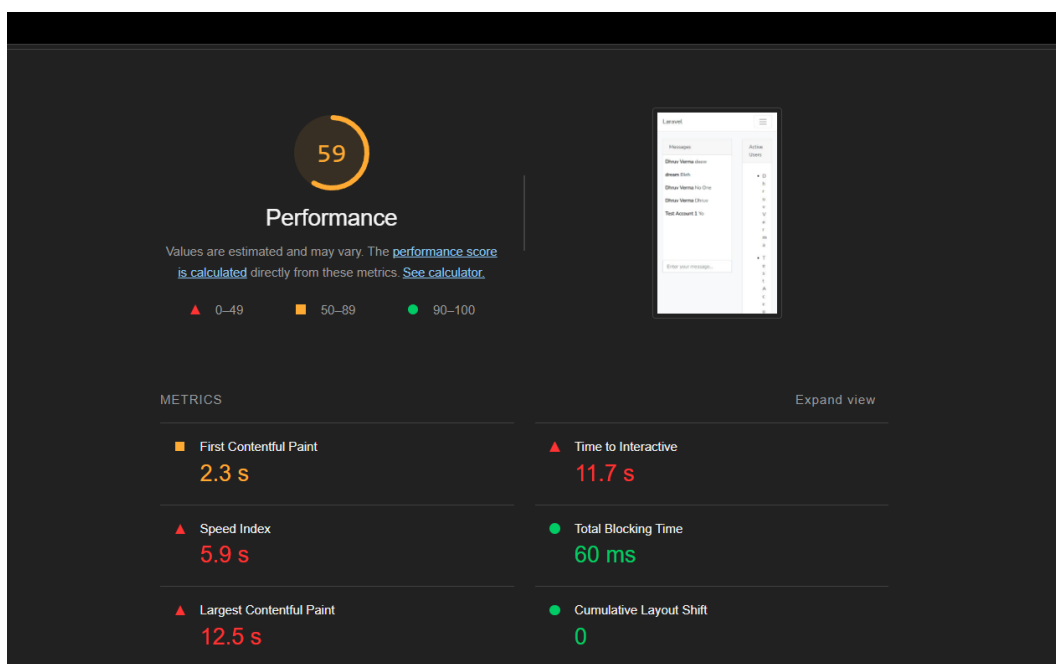


Figure 72. Vue-Laravel Chat Application Performance.

5.2.4 Swift iOS

Network Analysis

Like the web applications, this analysis also follows the tracking of the HTTP, TCP and WebSocket requests and responses in a given period of time.

Tracking HTTP Packets: Compared to web applications, native applications do not require a large amount of HTML structure. Also, the iOS platform makes it easier to render the application structure once. Since there is only a one-time request to the server, therefore the captured HTTP packets are also less.

The figure 73 displays the number of HTTP packets that are delivered and received within 6 seconds. The amount of HTTP packets for this case reaches 77.

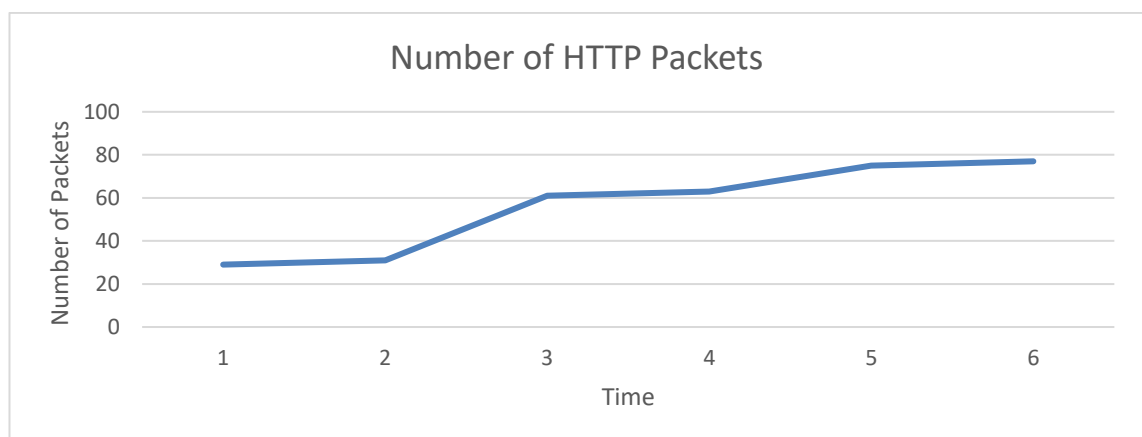


Figure 73. Swift - Number of HTTP Packets.

The figure 74 describes the length of HTTP packets sent and received in 6 seconds. For this case, the maximum length of a HTTP packet is about 395 bytes.

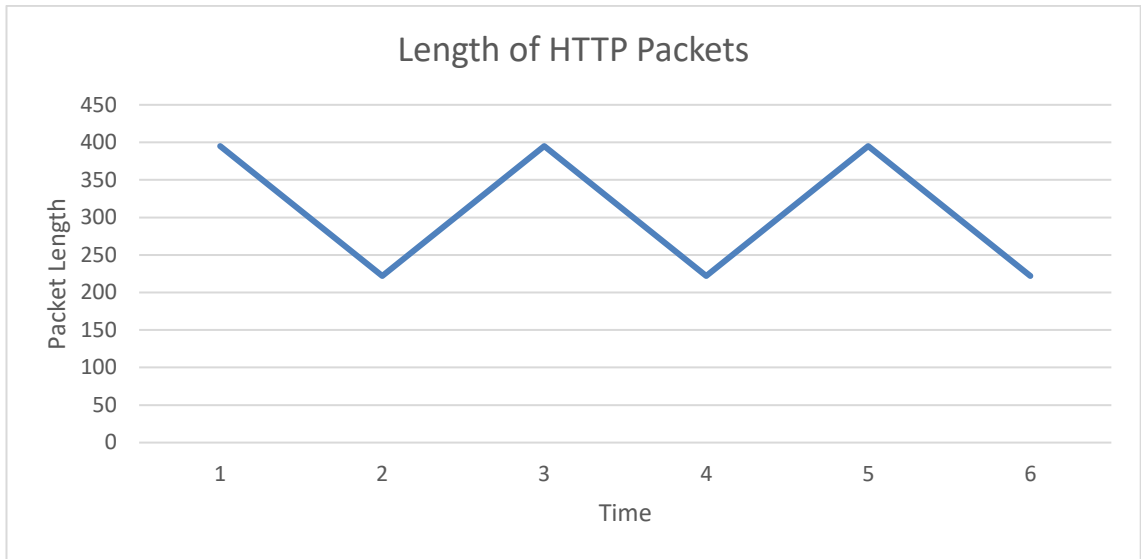


Figure 74. Swift - Length of HTTP Packets.

Tracking TCP Packets: The amount of TCP packets is also analogous to the HTTP packets. The figure 75 shows the number of TCP packets sent and received for 100 seconds. The maximum number of the TCP packets recorded for the application in consideration is 32.

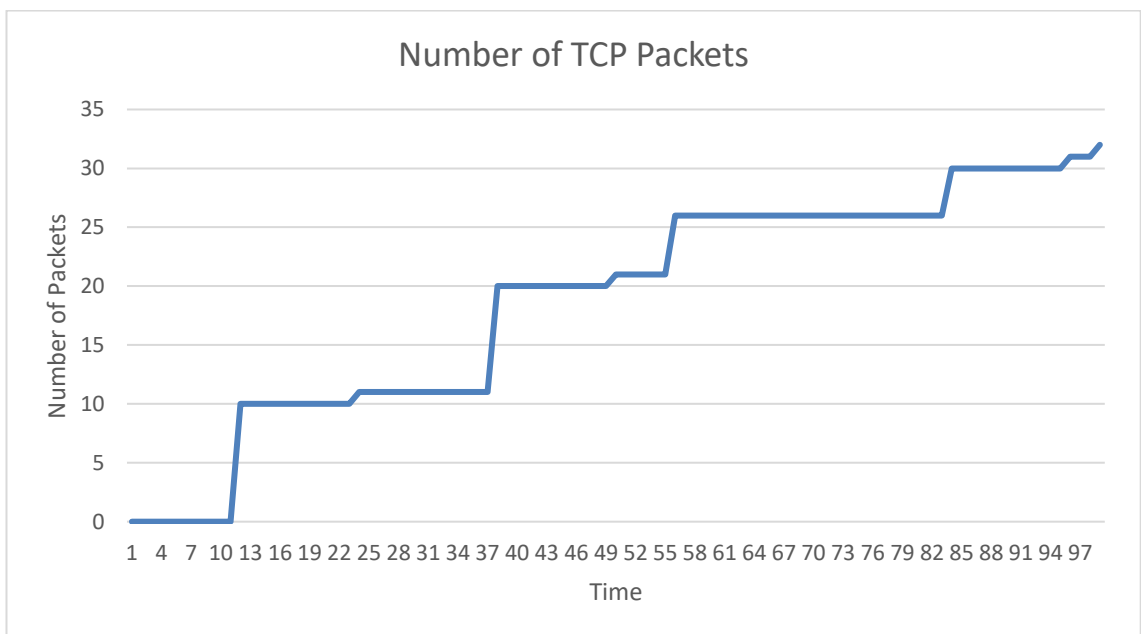


Figure 75. Swift - Number of TCP Packets.

The figure 76 describes the length of TCP packets sent and received in 100 seconds. The maximum length of the TCP packet achieved in this application is 219 bytes.

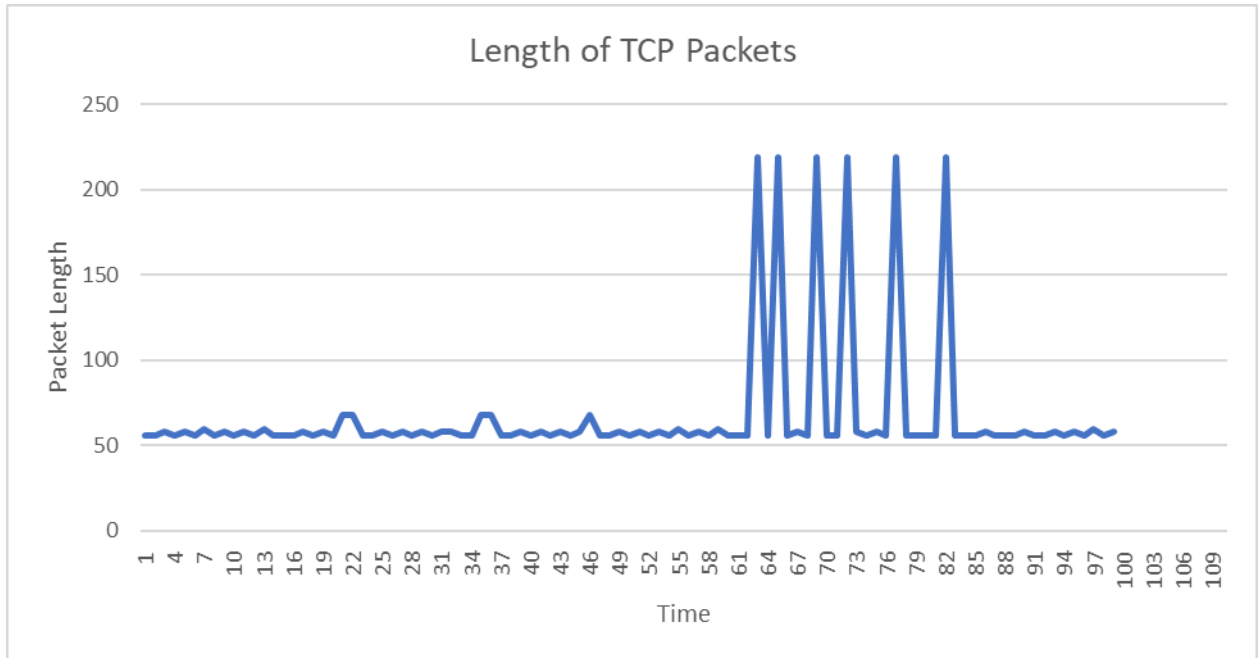


Figure 76. Swift - Length of TCP Packets.

Tracing WebSocket: The figure 77 shows the number of WebSocket data sent and received for 100 seconds. For the application in consideration, the amount of WebSocket data tracked is 487.

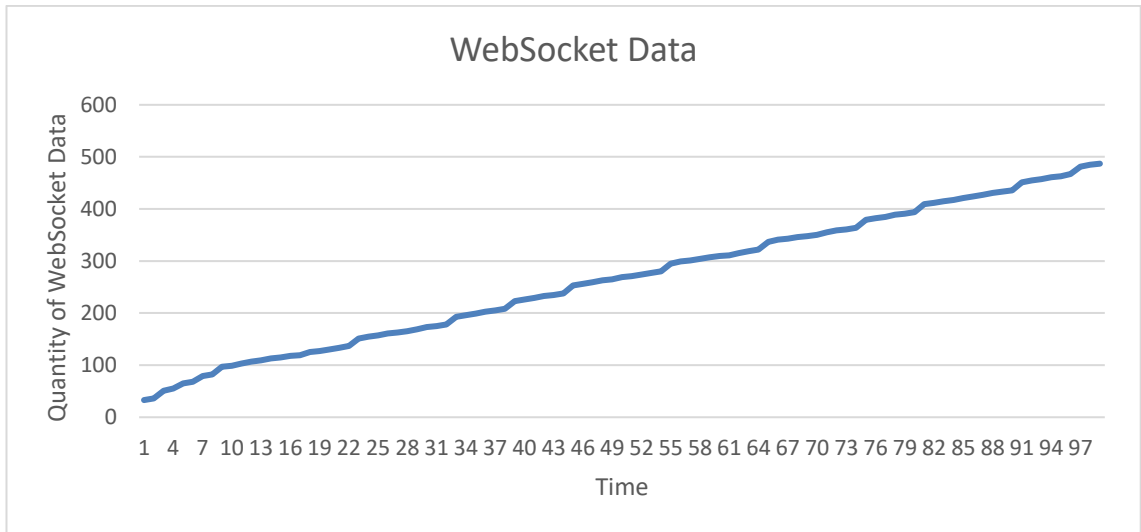


Figure 77. Swift - Quantity of WebSocket Data.

The figure 78 describes the length of WebSocket data sent and received till 100th second. For this application, the average length of the data is 109.5 bytes.

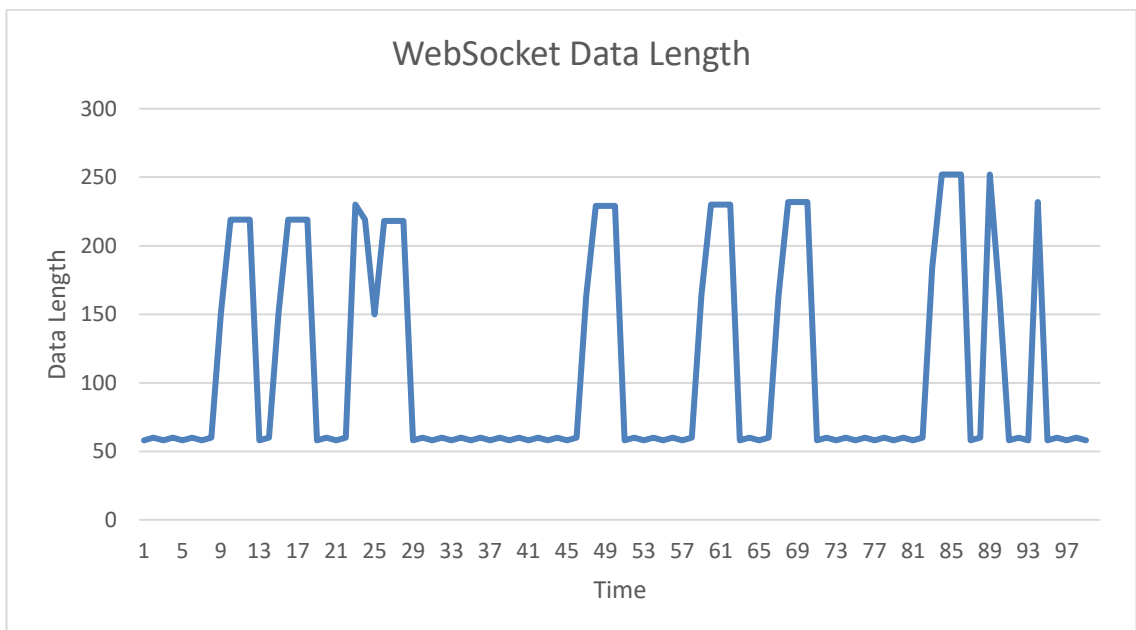


Figure 78. Swift - Length of WebSocket Data.

Through the Wireshark analysis, it is also determined that the queue time for the considered application is 1.29 milliseconds, and the stalled time is 0.46 milliseconds.

Performance Analysis

The analyses of iOS applications varies significantly compared to the analyses of the web applications. The iOS applications do not run on the browser, so the performance measurements are based on response time of the application in the emulator devices.

5.2.5 React Native and Express

Network Analysis

Like the web applications, this analysis also follows the tracking of the HTTP, TCP and WebSocket requests and responses in a given period of time.

Tracking HTTP Packets: Compared to the iOS applications, native applications produce a huge amount of HTTP traffic. Since there is switch between 80 and 443 protocols of HTTP requests hence, the captured HTTP packets are large in quantity.

The figure 79 displays the number of HTTP packets that are delivered and received within 100 seconds. The amount of HTTP packets for this case reaches 5247.

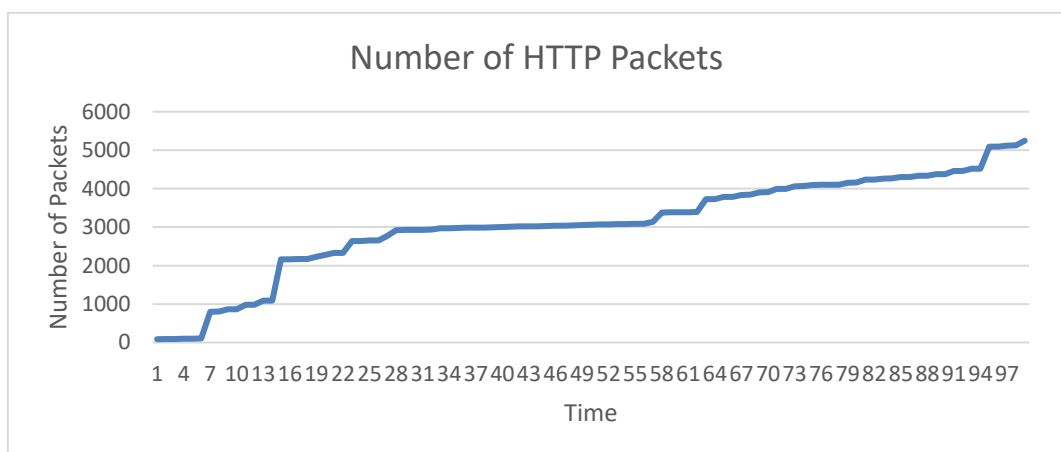


Figure 79. React-Native-Express - Number of HTTP Packages.

The figure 80 describes the length of HTTP packets sent and received in 6 seconds. For this case, the maximum length of a HTTP packet is about 1366 bytes.

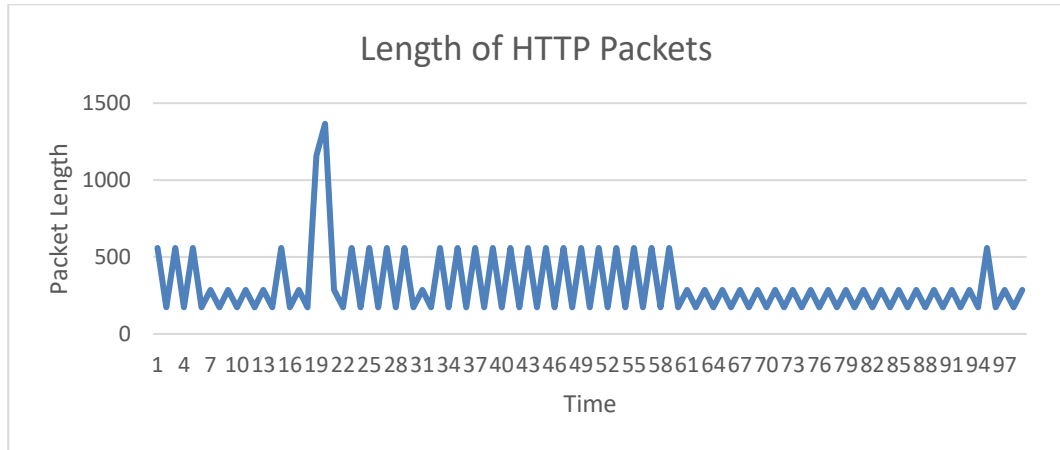


Figure 80. React-Native-Express - Length of HTTP Packages.

Tracking TCP Packets: The figure 81 shows the number of TCP packets sent and received for 100 seconds. The maximum number of the TCP packets recorded for the application in consideration is 871.

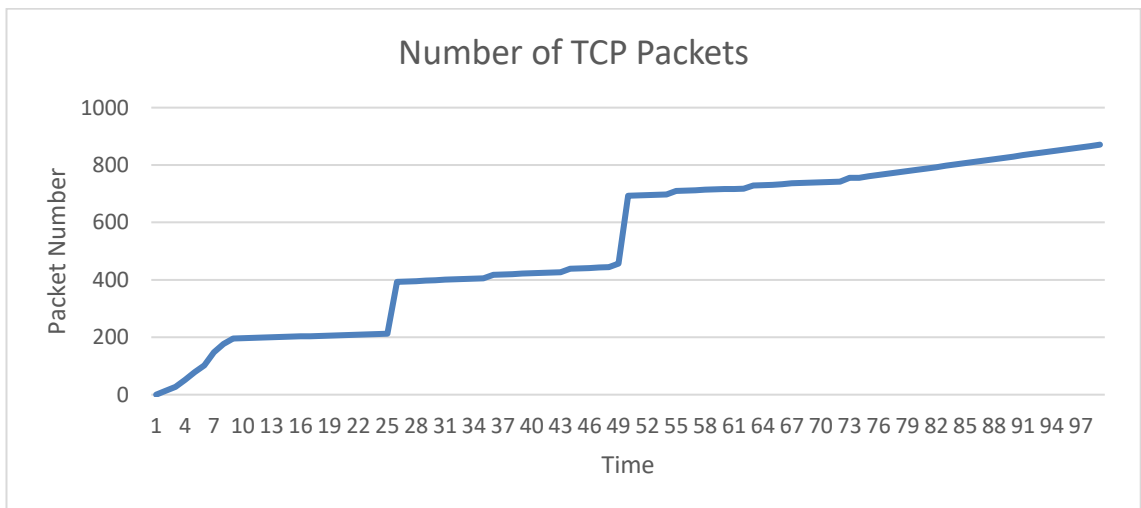


Figure 81. React-Native-Express - Number of TCP Packages.

The figure 82 describes the length of TCP packets sent and received in 100 seconds. The maximum length of the TCP packet achieved in this application is 83 bytes.

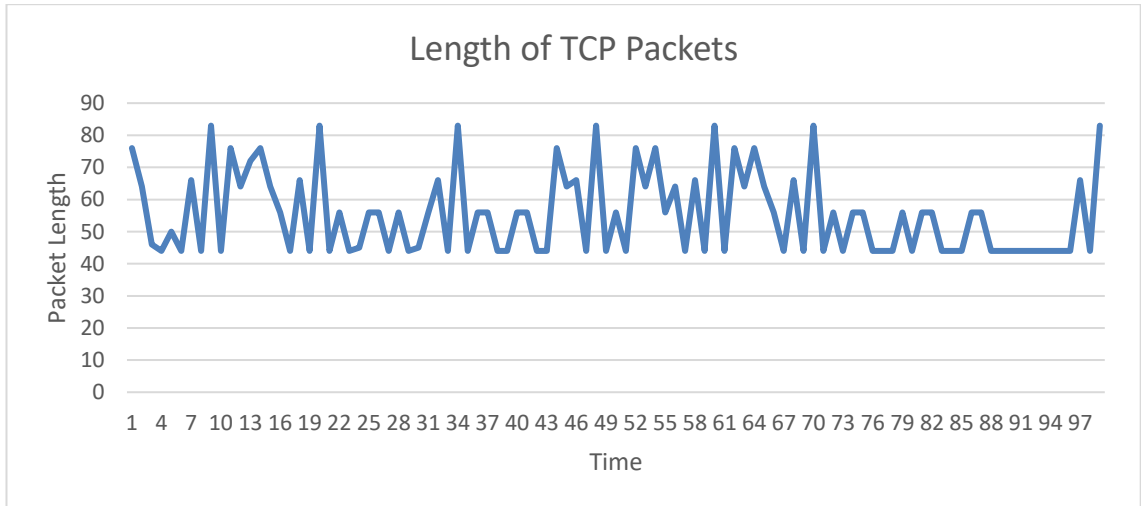


Figure 82. React-Native-Express - Length of TCP Packages.

Tracing WebSocket: The figure 83 shows the number of WebSocket data sent and received for 100 seconds. For the application in consideration, the amount of WebSocket data tracked is 45.

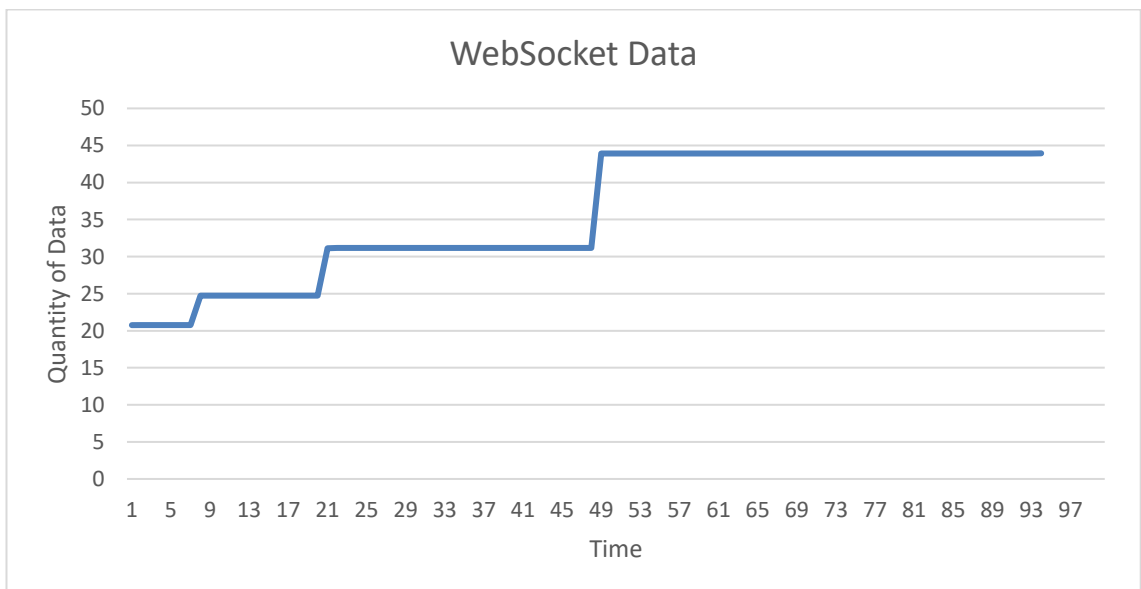


Figure 83. React-Native-Express - Quantity of WebSocket Data.

The figure 84 describes the length of WebSocket data sent and received till 100th second. For this application, the average length of the data is 108.5 bytes.

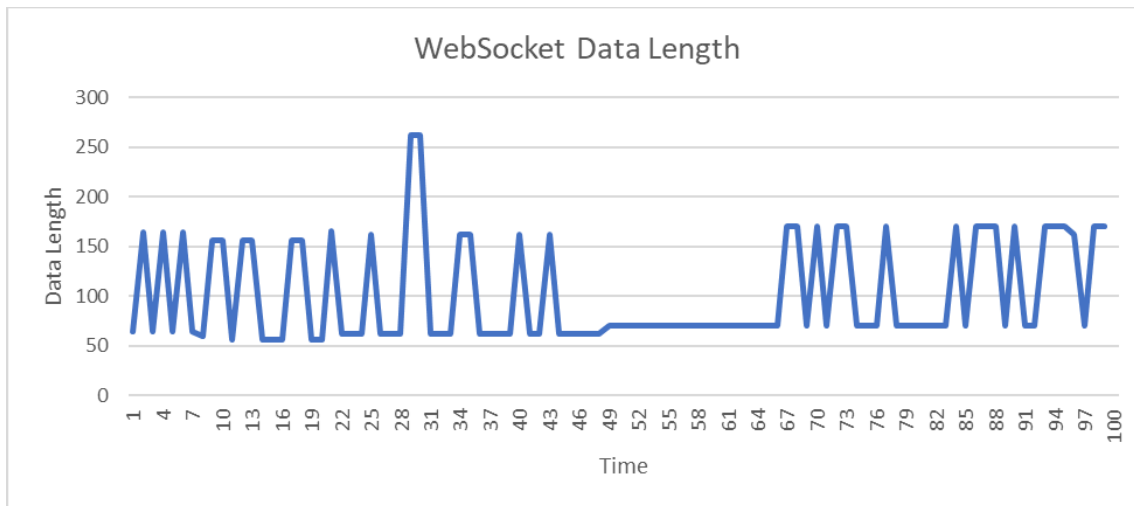


Figure 84. React-Native-Express - Length of WebSocket Data.

The figure 85 displays the queue time and stalled time for the application. In this case the queue time for the WebSocket connection is 2.39 milliseconds while the stalled time is 1.16 milliseconds. Moreover, the Waiting Time is 10.64 milliseconds in this case.

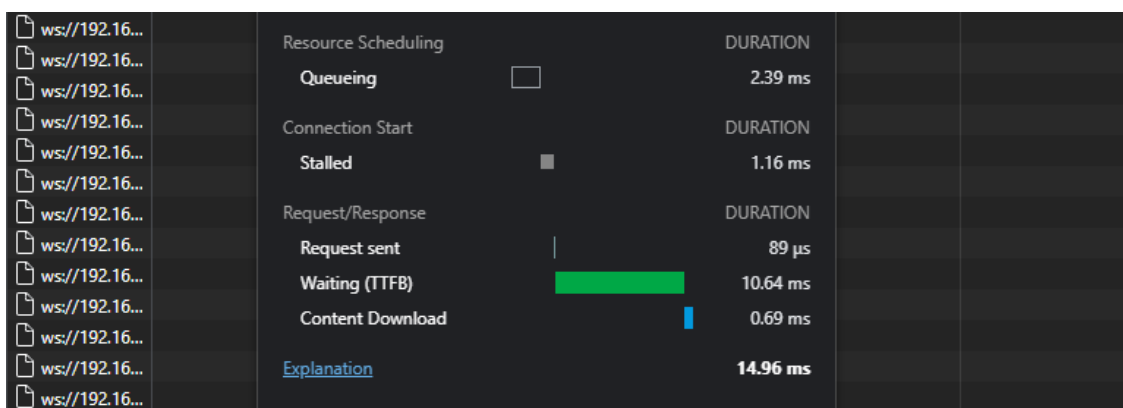


Figure 85. React-Native-Express - WebSocket Stalling Time.

Performance Analysis

The analyzing of react-native and express application is far easier compared to the analyses of Swift application. The native framework provides the opportunity to run the application in the browser. The browser can then test the performance of the application. The generated report provides values for the speed index, interactive time, blocking time and layout shift of the application. (Figure 86)

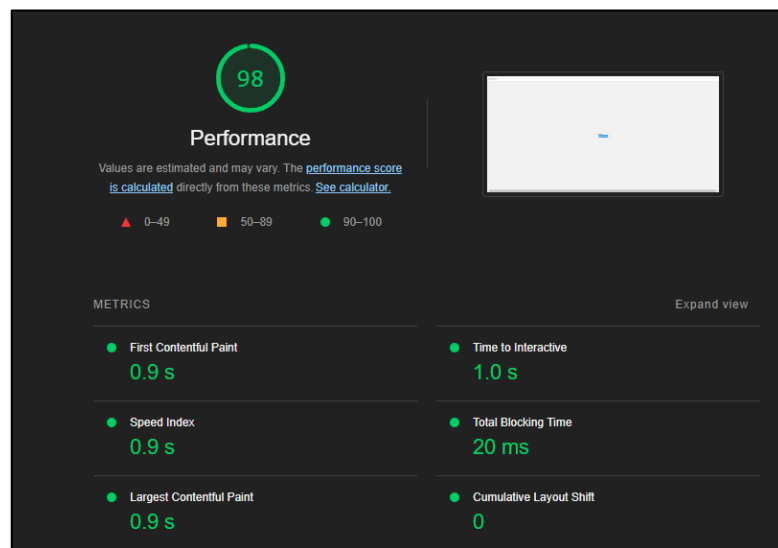


Figure 86. React-Native-Express - Chat Application Performance.

6 RESULTS AND CONCLUSION

The result obtained from the comparative analysis of different web-development frameworks is presented in the following sections.

6.1 Frameworks in consideration

Table 3. Web Frameworks for Chat Application.

S No.	Front-End	Back-End
1	Angular JS	Flask
2	React JS	Rails
3	Vue JS	Laravel

The table 3. contains the combination of frameworks that were used in implementation of the Web-Application.

Table 4. Native Frameworks for Chat Application.

S No.	Front-End	Back-End
1	Swift	Swift
2	React Native	Express

The table 4. contains the combination of frameworks that were used in implementation of the Native-Application.

The front-end framework Angular JS and back-end framework Flask are used together to implement the chat application. Unlike other popular front-end frameworks, Angular supports dependency injection which is important while working with the Flask framework. Moreover, Angular is quite efficient at handling complex single-page application which directly corresponds to Flask's ability to handle multiple routes and still provide a lightweight and fast application. Flask

also allows the application to be coded in an object-oriented style which correlates with Angular's object-oriented approach.

The combination of React and Rails provide access to several built-in code libraries which reduces developing time and effort. The application created with this combination excels in stability and quality of the application as shown in table 4. Unlike other framework combinations, the lightweight React, and the complex Rails work together to form an application which uses less memory resources and increases performance.

Laravel and Vue go hand in hand. Vue can be described as a minimalistic front-end framework that excels in single page user interfaces while Laravel enhances Vue's performance. The built-in libraries for the Vue development in Laravel framework make them a good combination. The Laravel Vue stacks allows a developer to efficiently build a single page application with a seamless front-end.

The swift framework is solely used for the iOS application development as there are no combinations of different frameworks which can be used to develop iOS applications. iOS follows a strict policy and does not allow third-party applications to run on the platform.

Just like React, React-Native is also based on the single-page application (SPA). It means that the mobile application can be accessed from a single native page. This functionality avoids loading a new page with every action, thus providing a streamlined user experience. Express on the other hand, creates a server which corresponds to the React-Native functionalities and this combination creates a very fast and stable application.

6.2 Results of Web-Application Frameworks

6.2.1 Benchmark Analysis

The figure 87 describes the benchmark scores received by the concerned web frameworks depending on the criteria. Each of the framework combination is graded according to their performance and measurements.

Development and Ease of Modification

From the survey, it is witnessed that React and Rails framework combination achieved the highest score while Vue and Laravel combination scored the lowest. The score is based on the development, debugging, modifying and testing of an application.

Performance

It is also clear that React and Rails framework combination wins in this category while Vue and Laravel combination scored the least points. The performance is based on the generated lighthouse report and stalling time taken by the application's data packets. The figures display the generated lighthouse report and the stalling time for each framework respectively.

Ease of Deployment

In this criterion, Angular and Flask framework combination ties to React and Rails one. This criterion is a subjective, yet a significant argument and is measured by the time and effort spent on deploying the application on a server. By comparison, it is far easier to deploy Flask and Rails frameworks while Laravel takes a huge amount of effort.

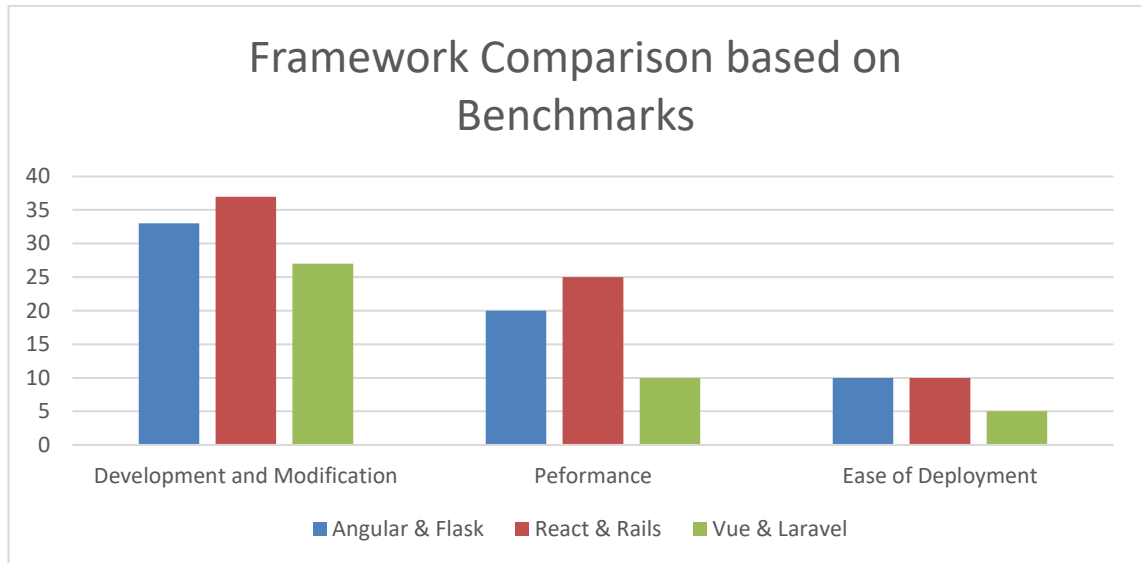


Figure 87. Web Framework - Benchmark Comparison.

From the given trend, it is also witnessed that Angular and Flask framework combination consistently remained in the second position except for the ‘Ease of Deployment’ criterion where it tied React and Rails framework combination.

6.2.2 Network Analysis

This section describes the combined analysis of the concerned web frameworks depending on the protocols. The protocols used to analyze each application are HTTP, TCP and WebSocket. Each of the framework is graded according to their amount of generated traffic.

HTTP

The figure 88 shows the amount (number and length) of packets generated by the HTTP flow of each application implemented using the concerned frameworks. It is witnessed that React and Rails framework combination generates less traffic compared to the other two. It implies that this framework does not require constant requests to generate the HTML structures. Hence, it performs better than the other frameworks.

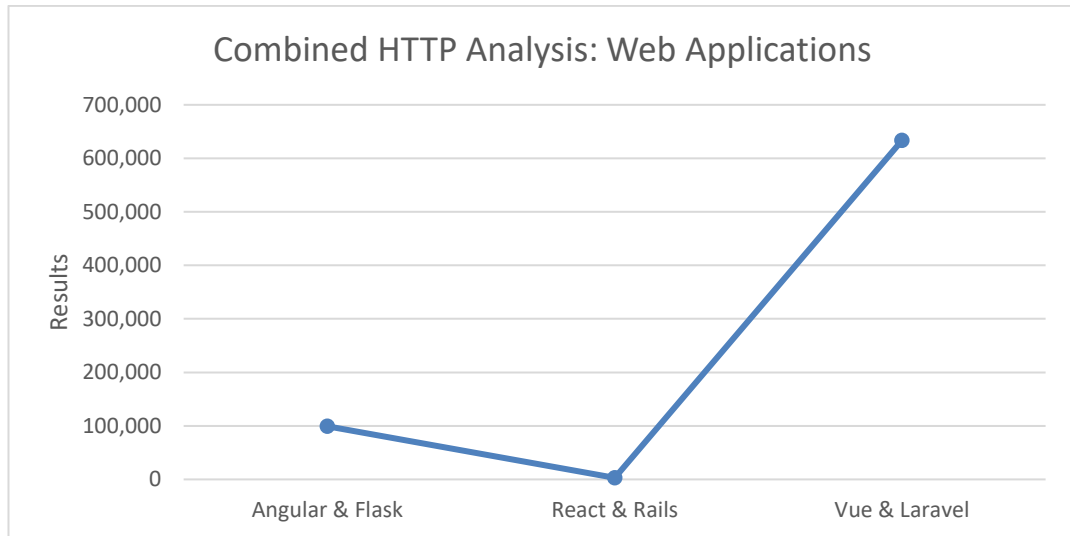


Figure 88. Web Applications - HTTP Analysis.

TCP

The figure 89 shows the amount (number and length) of packets generated by the TCP flow of each application implemented using the concerned frameworks. It is noticed that Angular and Flask framework combination generates the least amount traffic of them all. It implies that this framework does not require constant packet communication and therefore it performs better in this criterion. However, it should also be observed that React and Rails framework combination generates comparable amount of TCP traffic to the Angular and Flask framework combination.

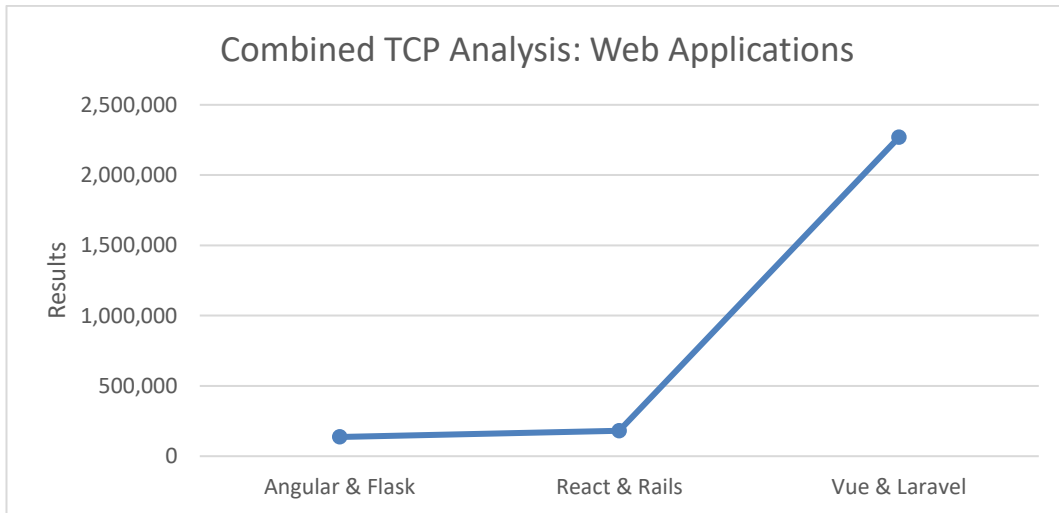


Figure 89. Web Applications - TCP Analysis.

WebSocket

The figure 90 shows the amount (number and length) of packets generated by the WebSocket flow of each application implemented using the concerned frameworks. It is no surprise that the React and Rails framework combination generates the least amount traffic of them all. It should also be taken into consideration that each application received the same amount of data. It implies that React and Rails framework does not require high number of packets and therefore it performs the best in this criterion. However, it should also be observed that Vue and Laravel framework combination generates comparable amount of WebSocket traffic to the Angular and Flask framework combination.

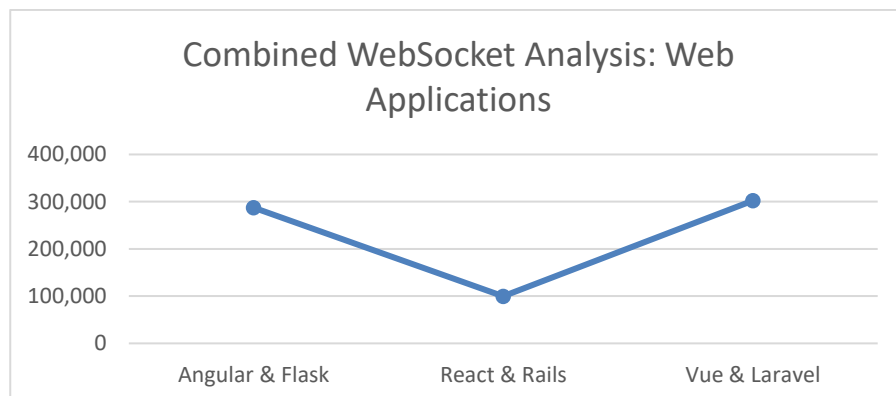


Figure 90. Web Applications - WebSocket Analysis.

The results of these network analyses indicate that React and Rails framework combination require the least amount of traffic packets to request and response the same amount of data compared to the other framework combinations. Therefore, React and Rails framework combination triumphs over the others.

6.3 Results of Native-Application Frameworks

6.3.1 Benchmark Analysis

The figure 91 describes the benchmark scores received by the concerned native frameworks depending on the criteria. Each of the framework combination is graded according to their performance and measurements.

Development and Ease of Modification

From the survey, it is witnessed that React-Native and Express framework combination achieved the higher score compared to the Swift framework.

Performance

It is also clear that React-Native and Express framework combination wins barely in this category. The performance is based on the time taken by the application to load and function. Both the frameworks took similar amount of time during the analyses, but React-Native and Express framework combination secured the higher score marginally.

Ease of Deployment

In this criterion, both the concerned frameworks secure equal points. It is quite simple to deploy both the applications. It takes one line of code to mention the required server and deploy the application in both the cases. They do not require

any effort or excess amount of time. Hence, these frameworks scored the same points.

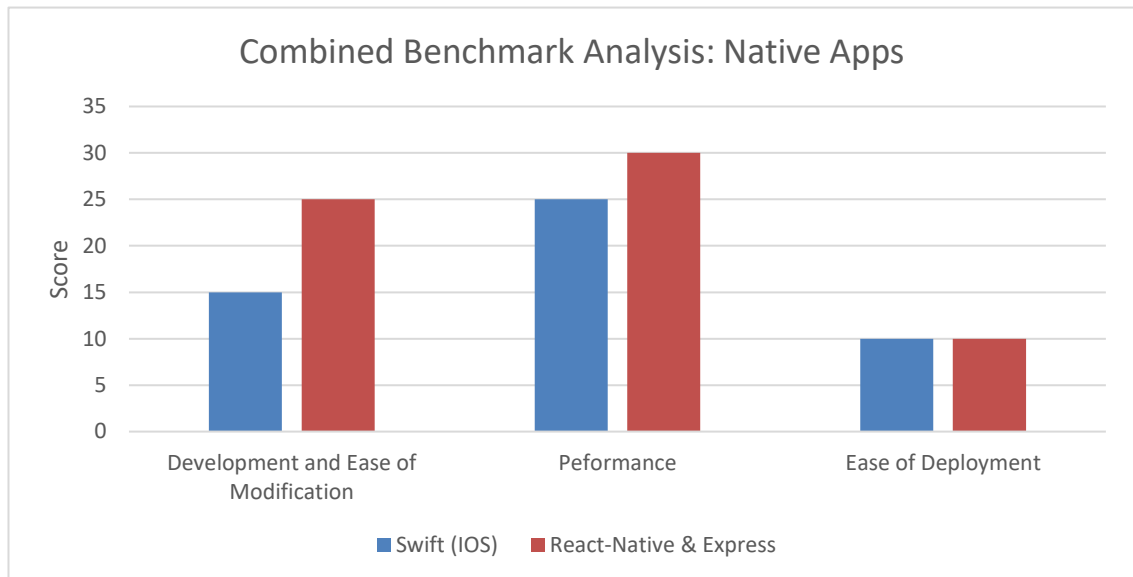


Figure 91. Native Applications - Benchmark Analysis.

6.3.2 Network Analysis

This section describes the combined analysis of the concerned native frameworks depending on the protocols. The protocols used to analyze each application are HTTP, TCP and WebSocket. Each of the framework is graded according to their amount of generated traffic.

HTTP

The figure 92 shows the amount (number and length) of packets generated by the HTTP flow of each application implemented. It is clear that the Swift framework-based application requires the least amount of HTTP traffic to generate the UI and its components.

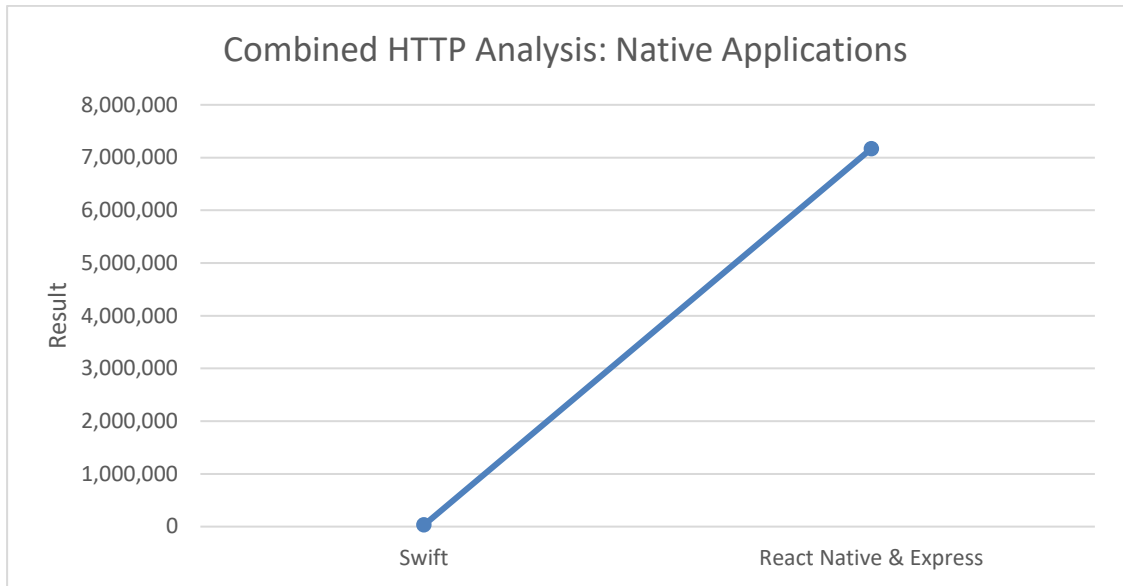


Figure 92. Native Applications - HTTP Analysis.

TCP

The figure 93 shows the amount (number and length) of packets generated by the TCP flow of each application implemented using the concerned frameworks. In this case too, the Swift framework requires less amount traffic compared to the React and Express application. Hence the Swift application scores more in this criterion.

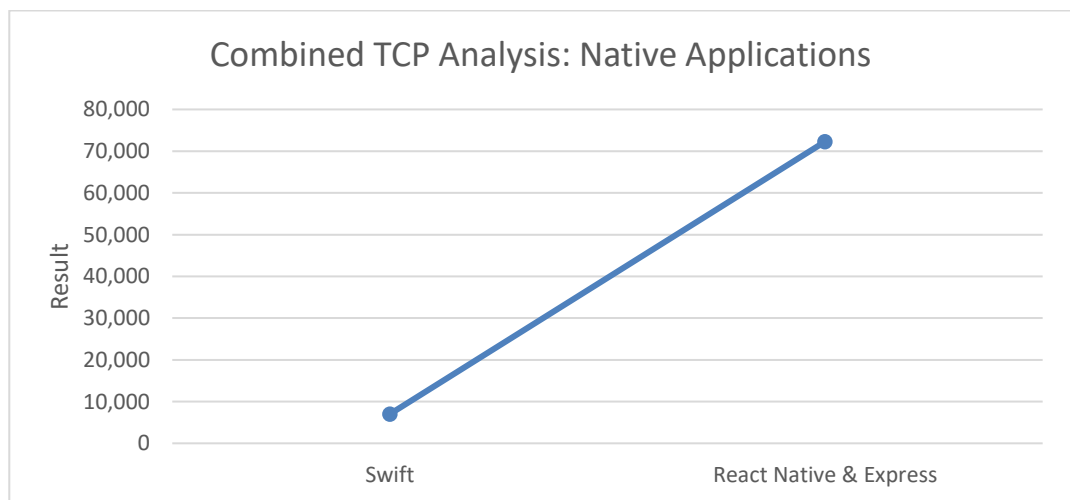


Figure 93. Native Applications - TCP Analysis.

WebSocket

The figure 94 shows the amount (number and length) of packets generated by the WebSocket flow of each application implemented using the concerned frameworks. It is a huge surprise that React Native and Express framework combination generated very less WebSocket traffic compared to the Swift framework. This signifies that the React Native and Express application has a more stable connection between its client and server.

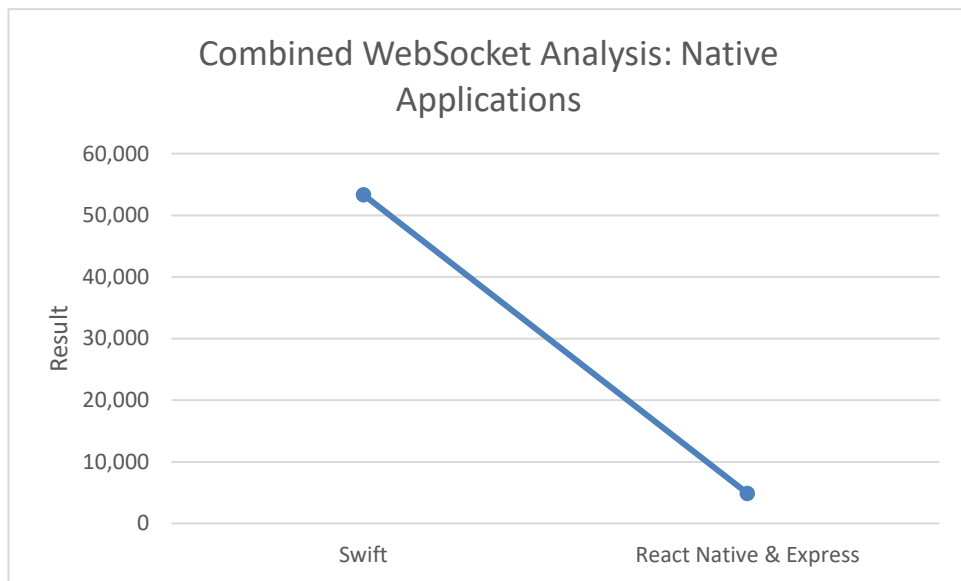


Figure 94. Native Applications - WebSocket Analysis.

The results of these network analyses indicate that Swift framework combination require the least amount of traffic packets to generate the UI and communicate with the server. However, it required a huge amount of traffic for the chat-application to work.

7 Conclusion

From the results, it can be inferred that the application developed using React and Rails framework performs better as compared to the other web frameworks. Also, for the native applications, React-Native framework has better benchmark compared to the Swift framework. The major drawback of Swift being that it is only accessible for the iOS platforms whereas React-Native and Express framework combination utilize the cross-platform functionality.

The benchmark scoring was done for to the chat application that was developed. It cannot be perfectly concluded that React and Rails will perform better as compared to other frameworks using analysis because only certain attributes related to the benchmark criteria were used. Other benchmark criteria like 'plugin support', 'security considerations' and 'AJAX support' were not used. These criteria were did not fit into the current chat application as WebSocket themselves provide an extra layer of socket security during the conversation between the client and the server.

Similarly for the native and cross-platform applications, it cannot be successfully deduced that React-Native and Express framework will perform better than the Swift frameworks. However, it can be predicted that for a given project, these frameworks can perform better than the others.

Challenges and Limitations

The research study encountered various challenges and limitations during network capturing and analysis. These challenges are summarised in the following section:

1. The first challenge was to isolate the traffic on the Windows platform. The main problem was to overcome the SSDP lag during the packet transfer.

2. The CORS policy had to be reset quite frequently during the analysis of the application. It restricted the application from running on different server which proved to be a huge obstacle.
3. iOS application testing was another challenge that proved to be quite difficult. Since Apple does not allow any third-party applications, therefore the performance of the iOS application was measured manually using the network dataset.
4. Some of the attributes on the benchmark criteria are not mentioned for the concerned chat application. These criteria depend on the application's environment which means some of these criteria will be well suited for different types of applications.
5. Broadcasting of WebSocket using Laravel did not go smoothly. It took at least 7 hours to fix the error due to lack of proper documentation.
6. The considered chat application was implemented due to the limited amount of dataset. So, the benchmark criteria might show different results while testing on much larger scale.
7. The network packet capturing application on the One Plus 8T Device had to manually stopped and then relaunched multiple times after the VPN services provided a stable connection.

8 Bibliography

Berninger, V. S. S. D. a. H. D., 2001. *CHAPTER 7 - Assessment for Reading and Writing Intervention: A Three-Tier Model for Prevention and Remediation..* [Online]

Available at:

<https://www.sciencedirect.com/science/article/pii/B9780120585700500094>

[Accessed May 2022].

Blog by Railsware, 2020. *Everything You Need to Know about Ruby on Rails Web Application Framework..* [Online]

Available at: <https://railsware.com/blog/ruby-on-rails-guide/>.

[Accessed 2022].

Carstens, T., n.d. *PROGRAMMING WITH PCAP.* [Online]

Available at: <https://www.tcpdump.org/pcap.html>

[Accessed may 2022].

Dasari Hermitha Curie, Joyce Jaison. Jyoti Yadav, J Rex Fiona, 2019. Analysis on Web Frameworks. *Journal of Physics: Conference Series*, 11, Volume 1362, p. 012114.

Docs, MDN Web, 2019. *The WebSocket API (WebSockets).* [Online]

Available at: [https://developer.mozilla.org/en-](https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API)

[US/docs/Web/API/WebSockets_API](https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API)

[Accessed 2022].

EDUCBA, 2019. *What is Internet Application | Top 8 Application of Internet with Advantages.* [Online]

Available at: <https://www.educba.com/what-is-internet-application/>

[Accessed May 2022].

Ghimire, D., 2020. *Comparative study on Python web.* [Online]

Available at:

https://www.theseus.fi/bitstream/handle/10024/339796/Ghimire_Devndra.pdf

Heitkötter, H., Tim A., R., Majchrzak, B. & Weber, T., 2014. Comparison of Mobile Web Frameworks. *Lecture Notes in Business Information Processing*, p. 119–137.

IBM Cloud Education, 2020. *What is Three-Tier Architecture*. [Online]
Available at: <https://www.ibm.com/cloud/learn/three-tier-architecture>
[Accessed May 2022].

Ignacio Fernández-Villamor, J. D.-C. L. a. I. C., 2008. A comparison model for agile web frameworks. *Proceedings of the 2008 Euro American Conference on Telematics and Information Systems*.

Imperva, 2021. [Online]
Available at: <https://www.imperva.com/learn/application-security/osi-model/>

Infosec Resources, n.d. *Hypertext Transfer Protocol (HTTP) with Wireshark*. [Online]
Available at: <https://resources.infosecinstitute.com/topic/hypertext-transfer-protocol-http-with-wireshark/>
[Accessed 2022].

InterviewBit, 2021. *Angular Vs React: Difference Between Angular and React*. [Online]
Available at: <https://www.interviewbit.com/blog/angular-vs-react/>
[Accessed May 2022].

InterviewBit, 2022. *Top 10 Web Development Frameworks [2022]*. [Online]
Available at: <https://www.interviewbit.com/blog/web-development-frameworks>

Majeed, A. & Rauf, I., 2018. *MVC Architecture: A Detailed Insight to the Modern Web Applications Development*. [Online]
Available at: <https://crimsonpublishers.com/prsp/pdf/PRSP.000505.pdf>
[Accessed May 2022].

mraible, 2010. *history-of-web-frameworks-timeline*. [Online]
Available at: <https://github.com/mraible/history-of-web-frameworks-timeline>
[Accessed May 2022].

Overflow, Stack, 2021. *2021 Developer Survey*. [Online]
Available at: insights.stackoverflow.com
[Accessed May 2022].

portswigger, n.d. *Testing for WebSockets security vulnerabilities | Web Security Academy*. [Online]
Available at: <https://portswigger.net/web-security/websockets>
[Accessed 2022].

Raible, M., 2006. *Comparing Web Frameworks*. [Online]
Available at: <https://equinox.dev.java.net/framework-comparison/WebFrameworks.pdf>

SearchNetworking, n.d. *What is the OSI model? The 7 layers of OSI explained*. [Online]
Available at: <https://www.techtarget.com/searchnetworking/definition/OSI>
[Accessed 2022].

StackHawk, n.d. *Laravel CORS Guide: What It Is and How to Enable It*. [Online]
Available at: <https://www.stackhawk.com/blog/laravel-cors>
[Accessed 2022].

Strawn, J., 2018. *Design Patterns by Tutorials: MVVM*. [Online]
Available at: <https://www.raywenderlich.com/34-design-patterns-by-tutorials-mvvm>
[Accessed May 2022].

Techempower, 2021. *TechEmpower Framework Benchmarks*. [Online]
Available at: <https://www.techempower.com/benchmarks/>
[Accessed May 2022].

Techopedia, 2019. *What is Network Traffic? - Definition from Techopedia*. [Online]

Available at: <https://www.techopedia.com/definition/29917/network-traffic>

[Accessed May 2022].

Thakur, P., 2018. *Evaluation and Implementation of Progressive Web Application*. [Online]

Available at:

<https://www.theseus.fi/bitstream/handle/10024/142997/PWA%20thesis.pdf>

Tiganov, D., Cho, J., Ali, K. and Dolby, J., 2020. *SWAN: a static analysis framework for swift*. [Online]

Available at: <https://dl.acm.org/doi/10.1145/3368089.3417924>

[Accessed 2022].

Wireshark, n.d. *TCP Analysis*. [Online]

Available at:

https://www.wireshark.org/docs/wsug_html_chunked/ChAdvTCPAnalysis.html

[Accessed 2022].