BUILDING LEAN, STANDALONE WEB SERVERS, AND ROUTING ENGINES

PROJECT REPORT

Presented to

Dr. Chris Pollett

Department of Computer Science

San Jose State University

In Partial Fulfillment

Of the Requirements for the Class

CS 297

By

Ajita Shrivastava

Fall 2023

Abstract

Scalability, optimization, and rapid development are among the primary concerns in web application development. This project introduces a collection of single-file, low-dependency, pure PHP servers and routing engines that have adopted an event-driven, asynchronous I/O approach. Currently comprising two server classes – WebSite and GopherSite – the project can function seamlessly within traditional servers like Apache, nginx, or lighttpd, or operate independently as standalone servers for applications utilizing its routing capabilities. The server is implemented to be request event-driven, supporting asynchronous I/O for efficient web traffic handling. In response to the increasing complexity of email setups and the reliance on external software, this research endeavors to develop single-file, low-dependency, pure PHP servers and routing engines implementing SMTP and IMAP protocols. The aim is to simplify email server configurations, minimize external software dependencies, and democratize the process of setting up email servers. Creating lightweight servers using PHP ensures ease of integration into diverse projects while maintaining robustness. Thus, redefining the landscape of server technology, providing a user-friendly and efficient solution to email server management.

TABLE OF CONTENTS

I. Introduction

In the current global landscape, where web application development stands as a cornerstone of digital interaction, the imperatives of scalability, optimization, and swift development have become paramount. Imagine the internet as a vast network of highways, and web applications as the cars navigating through. This project introduces another kind of road system - think of it as efficient pluggable highways. These highways, i.e. single file PHP servers, are designed to manage web traffic smoothly and quickly. They're like magical paths that help websites by providing a small yet efficient PHP server. But that's not all – this project also tackles another challenge: emails. Setting up email servers can be like solving a puzzle with too many pieces. The project simplifies this puzzle, making it easier for everyone to set up and manage their email servers. It's like giving you a user-friendly tool to handle your emails without needing a tech expert. Ultimately, this project serves as a versatile micro framework, ensuring a streamlined experience for developers to use under traditional servers like Apache, nginx, or lighttpd.

This report consists of four sections, each detailing one of the completed deliverables from this semester. Following these sections are the conclusion and future work. Each section elaborates on the deliverable's aim, the solution approach, and the results achieved. Like any research project, the first deliverable serves as an exercise to comprehend the existing technology and gain familiarity with the various aspects of this project. It entails creating routes using the web server to understand the structure and functioning of the server. The second deliverable provides a comprehensive understanding of the project's next phase: the transition from HTTP

1.1 to HTTP/2. Here, a simple program is developed to illustrate the conversion of a given HTTP 1.1 request into an HTTP/2 binary request format. The third deliverable introduces a new feature to the existing platform—middleware priority and ordering. Implemented as modular functions within the web server class, this addition enhances the project's capabilities to prioritize and order the execution of middleware functions. The fourth deliverable delves into extraction of the negotiated protocol. The implementation involves attempting to connect to a server to determine the protocol negotiated between the client and server for further communication.

In essence, this project and its deliverables establish a foundation of knowledge on the subject, provide an understanding of the features, and foster comfort of working on the project. This inherently charts the course for future project implementation. Finally, the section on future work outlines a direction for the future implementation of the project.

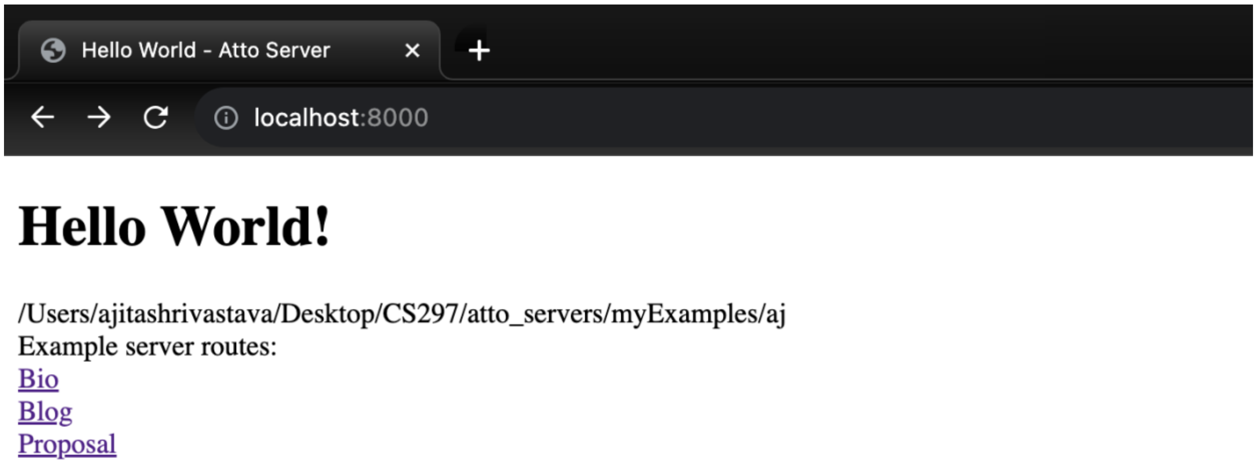
II. Deliverable 1: Implementation of Routing with Atto Server

The primary goal of this deliverable is to create a seamless system for directing web traffic within the Atto Server. By configuring routes in the index.php file, we aim to enable the server to display differsnt web pages, providing a practical understanding of its operational dynamics.

In setting up this routing system, we began by crafting the index.php file. This file acts as the central control for defining how incoming requests should be handled and directed to specific pages. The code written in this file establishes the groundwork for effective routing. Subsequently, running the server was a crucial step. By navigating to the folder containing the index.php file and executing the command php index.php, we ensured that the server was up and running, actively listening on port 8000.

To observe the routing in action, users simply needed to open their web browser and visit http://localhost:8000. This direct interaction with the main page provided a tangible demonstration of how the server manages different routes.

The successful implementation of these measures yielded a fully functional routing system within the Atto Server. The main page can be accessed through http://localhost:8000, and from there, navigate to various web pages, each associated with a distinct route as shown in the below screenshots.



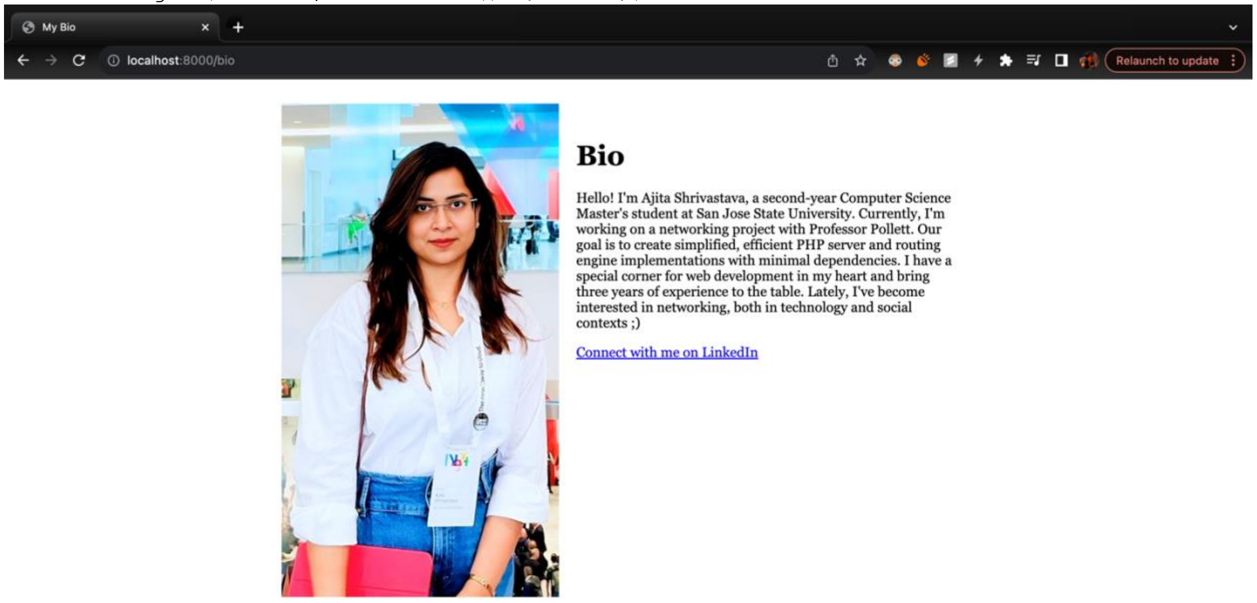
Hello World!

/Users/ajitashrivastava/Desktop/CS297/atto_servers/myExamples/aj
Example server routes:
Bio
Blog
Proposal

Figure 1: Results of the main page displaying the routes

Bio Page Route

```
$test->get('/bio', function() { ... });
```



Bio

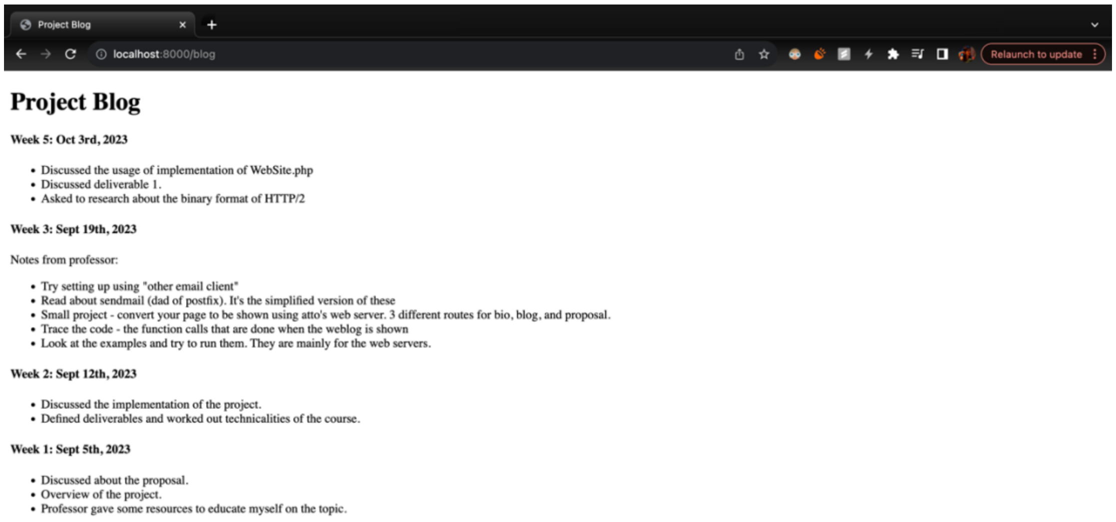
Hello! I'm Ajita Shrivastava, a second-year Computer Science Master's student at San Jose State University. Currently, I'm working on a networking project with Professor Pollett. Our goal is to create simplified, efficient PHP server and routing engine implementations with minimal dependencies. I have a special corner for web development in my heart and bring three years of experience to the table. Lately, I've become interested in networking, both in technology and social contexts ;)

Connect with me on LinkedIn

Figure 2: Results of the bio link route

Blog Page Route

```
$test->get('/blog', function() { ... });
```



Project Blog

Week 5: Oct 3rd, 2023

- Discussed the usage of implementation of WebSite.php
- Discussed deliverable 1.
- Asked to research about the binary format of HTTP/2

Week 3: Sept 19th, 2023

Notes from professor:

- Try setting up using "other email client"
- Read about sendmail (dad of postfix). It's the simplified version of these
- Small project - convert your page to be shown using atto's web server. 3 different routes for bio, blog, and proposal.
- Trace the code - the function calls that are done when the weblog is shown
- Look at the examples and try to run them. They are mainly for the web servers.

Week 2: Sept 12th, 2023

- Discussed the implementation of the project.
- Defined deliverables and worked out technicalities of the course.

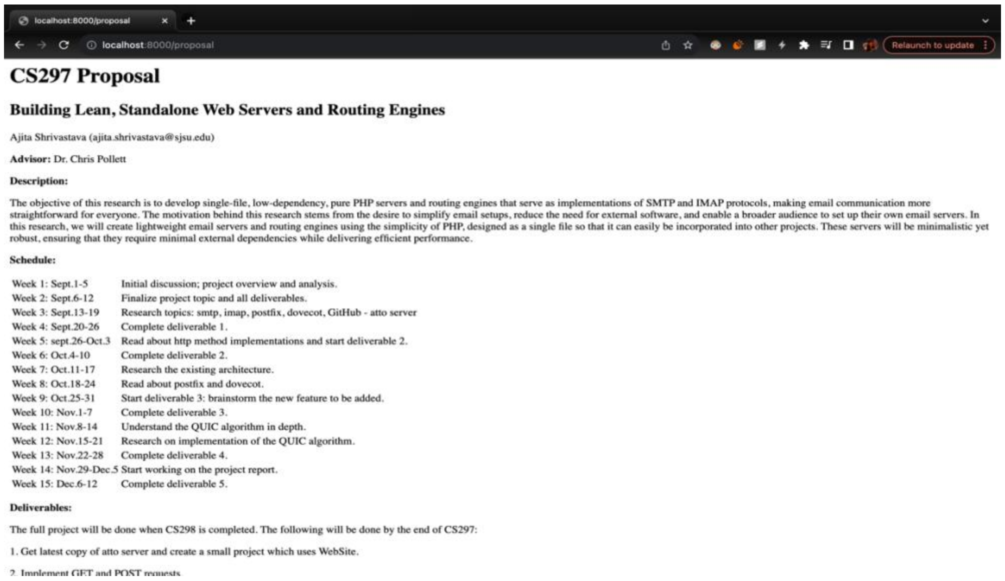
Week 1: Sept 5th, 2023

- Discussed about the proposal.
- Overview of the project.
- Professor gave some resources to educate myself on the topic.

Figure 3: Results of the project link route

Proposal Page Route

```
$test->get('/proposal', function() { ... });
```



CS297 Proposal

Building Lean, Standalone Web Servers and Routing Engines

Ajita Shrivastava (ajita.shrivastava@sjsu.edu)

Advisor: Dr. Chris Pollett

Description:

The objective of this research is to develop single-file, low-dependency, pure PHP servers and routing engines that serve as implementations of SMTP and IMAP protocols, making email communication more straightforward for everyone. The motivation behind this research stems from the desire to simplify email setups, reduce the need for external software, and enable a broader audience to set up their own email servers. In this research, we will create lightweight email servers and routing engines using the simplicity of PHP, designed as a single file so that it can easily be incorporated into other projects. These servers will be minimalistic yet robust, ensuring that they require minimal external dependencies while delivering efficient performance.

Schedule:

Week 1: Sept.1-5	Initial discussion; project overview and analysis.
Week 2: Sept.6-12	Finalize project topic and all deliverables.
Week 3: Sept.13-19	Research topics: smtp, imap, postfix, dovecot, GitHub - atto server
Week 4: Sept.20-26	Complete deliverable 1.
Week 5: sept.26-Oct.3	Read about http method implementations and start deliverable 2.
Week 6: Oct.4-10	Complete deliverable 2.
Week 7: Oct.11-17	Research the existing architecture.
Week 8: Oct.18-24	Read about postfix and dovecot.
Week 9: Oct.25-31	Start deliverable 3: brainstorm the new feature to be added.
Week 10: Nov.1-7	Complete deliverable 3.
Week 11: Nov.8-14	Understand the QUIC algorithm in depth.
Week 12: Nov.15-21	Research on implementation of the QUIC algorithm.
Week 13: Nov.22-28	Complete deliverable 4.
Week 14: Nov.29-Dec.5	Start working on the project report.
Week 15: Dec.6-12	Complete deliverable 5.

Deliverables:

The full project will be done when CS298 is completed. The following will be done by the end of CS297:

1. Get latest copy of atto server and create a small project which uses WebSite.

2. Implement GET and POST requests

Figure 4: Results of the proposal link route

III. Deliverable 2: Conversion of HTTP1.1 to HTTP/2

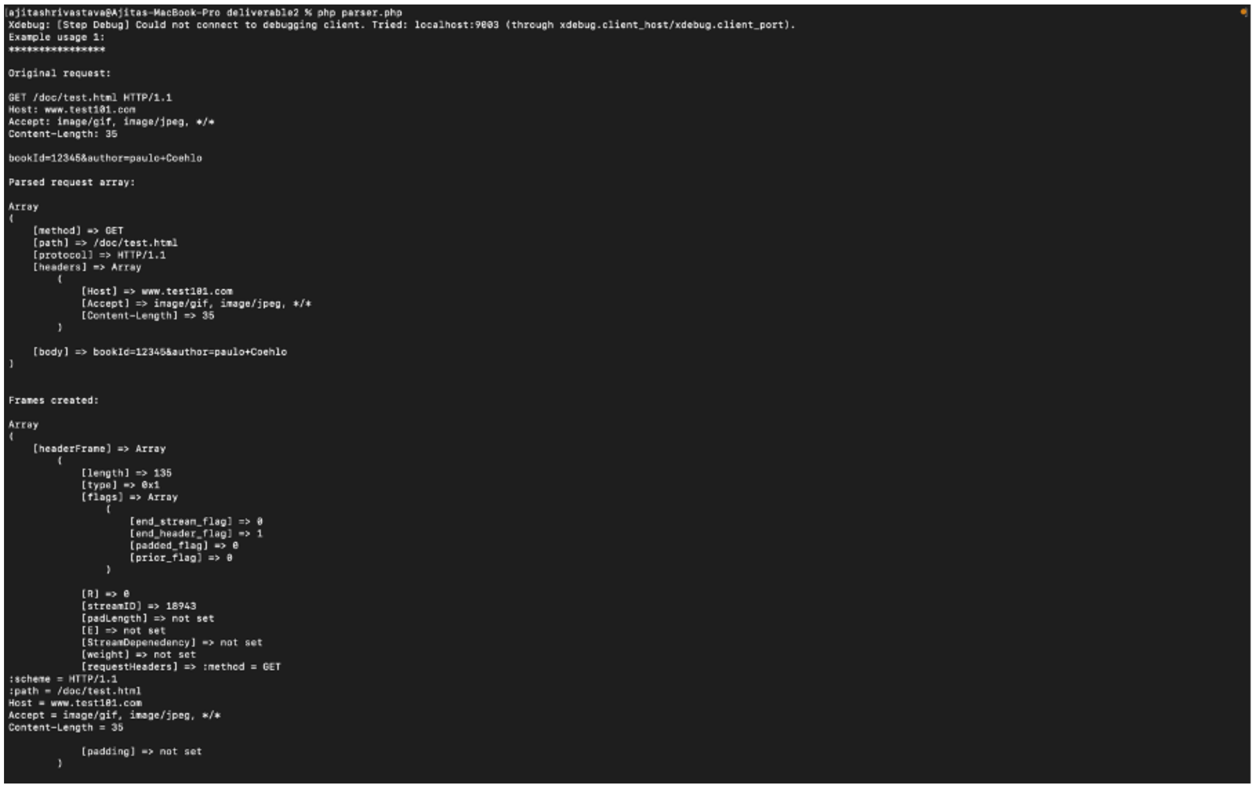
This deliverable involves the development of a program to convert an HTTP 1.1 request into the binary format of HTTP/2, offering insights into the conversion process. The program comprises three primary steps.

In the first step, the format is transformed by adding necessary headers and assigning their values. This involves extracting key components of the request, such as 'method', 'path', 'protocol', 'headers', and 'body'. Two functions contribute to this process: http1_request_parse($request_string) parses the request to create an array, and http1_to_http2_hex($request_array) replaces extracted parts with the new format.

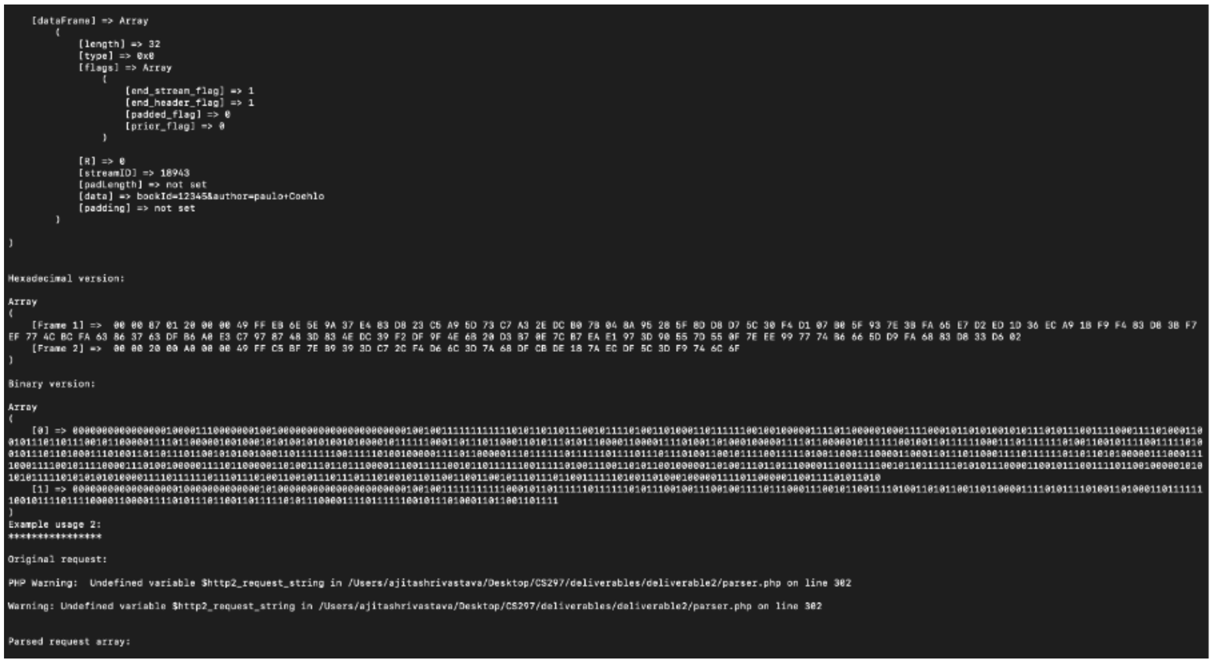
Following the new format, a request in HTTP/2 includes headers with binary values assigned to them. The createHeadersFrame($request_array, $lastFrame) function generates a frame for these headers. For example, the conversion of frame types is tabulated, with each type assigned a specific hexadecimal value.

The subsequent function, createDataFrame($request_array), creates a frame for the data stored in the original request by converting it into key-value pairs. This ensures the appropriate fitting into the HTTP/2 format.

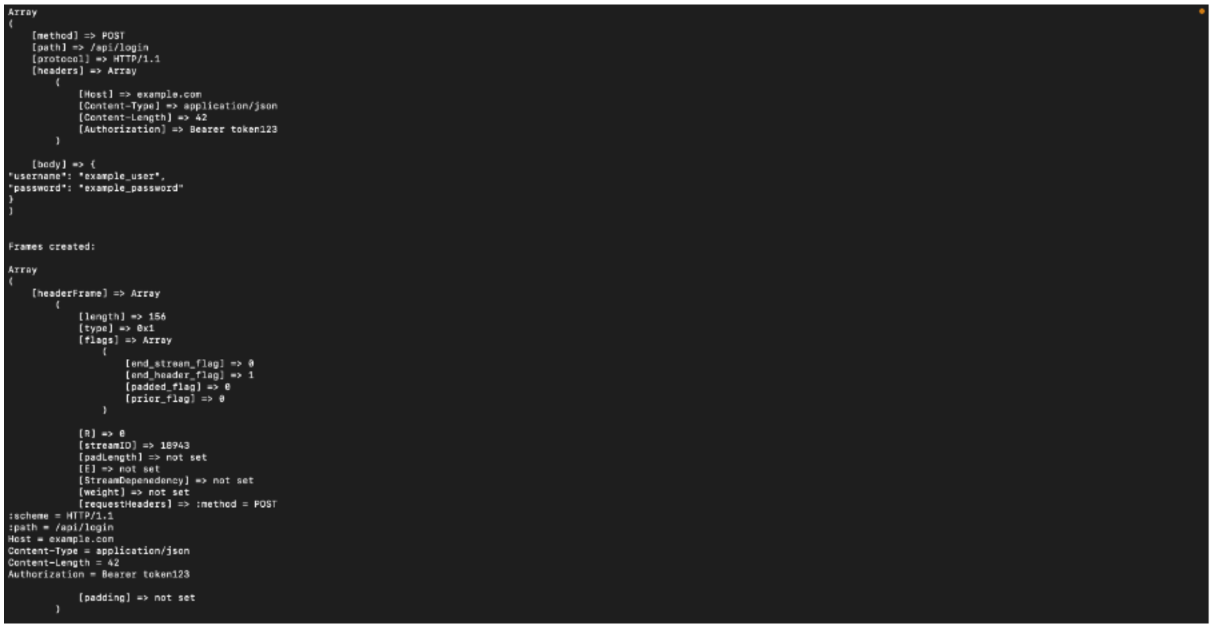
Finally, each frame is converted into binary, simplifying the process and facilitating further analysis. This comprehensive conversion process provides a clear understanding of how an HTTP 1.1 request evolves into the binary format of HTTP/2.



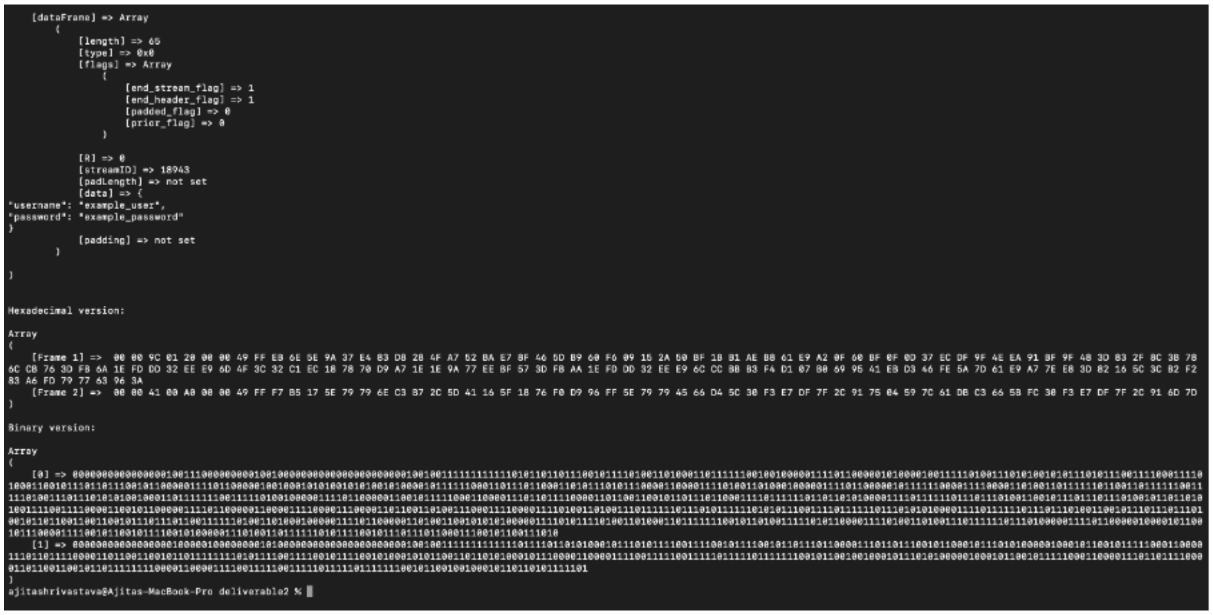
```
[ajitashrivastava@Ajitas-MacBook-Pro deliverable2 % php parser.php
Xdebug: [Step Debug] Could not connect to debugging client. Tried: localhost:9003 (through xdebug.client_host/xdebug.client_port).
Example usage 1:
****************

Original request:

GET /doc/test.html HTTP/1.1
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Content-Length: 35

bookId=12345&author=paulo+Coehlo

Parsed request array:

Array
(
    [method] => GET
    [path] => /doc/test.html
    [protocol] => HTTP/1.1
    [headers] => Array
        (
            [Host] => www.test101.com
            [Accept] => image/gif, image/jpeg, */*
            [Content-Length] => 35
        )

    [body] => bookId=12345&author=paulo+Coehlo
)


Frames created:

Array
(
    [headerFrame] => Array
        (
            [length] => 135
            [type] => 0x1
            [flags] => Array
                (
                    [end_stream_flag] => 0
                    [end_header_flag] => 1
                    [padded_flag] => 0
                    [prior_flag] => 0
                )

            [R] => 0
            [streamID] => 18943
            [padLength] => not set
            [E] => not set
            [StreamDepenedency] => not set
            [weight] => not set
            [requestHeaders] => :method = GET
:scheme = HTTP/1.1
:path = /doc/test.html
Host = www.test101.com
Accept = image/gif, image/jpeg, */*
Content-Length = 35

            [padding] => not set
        )
```



```
    [dataFrame] => Array
        (
            [length] => 32
            [type] => 0x0
            [flags] => Array
                (
                    [end_stream_flag] => 1
                    [end_header_flag] => 1
                    [padded_flag] => 0
                    [prior_flag] => 0
                )

            [R] => 0
            [streamID] => 18943
            [padLength] => not set
            [data] => bookId=12345&author=paulo+Coehlo
            [padding] => not set
        )

)


Hexadecimal version:

Array
(
    [Frame 1] =>  00 00 87 01 20 00 00 49 FF EB 6E 5E 9A 37 E4 83 D8 23 C5 A9 5D 73 C7 A3 2E DC B0 7B 04 8A 95 28 5F 8D D8 D7 5C 30 F4 D1 07 B0 5F 93 7E 3B FA 65 E7 D2 ED 1D 36 EC A9 1B F9 F4 83 D8 3B F7
EF 77 4C BC FA 63 86 37 63 DF B6 A0 E3 C7 97 87 48 3D 83 4E DC 39 F2 DF 9F 4E 6B 20 D3 B7 0E 7C B7 EA E1 97 3D 90 55 7D 55 0F 7E EE 99 77 74 B6 66 5D D9 FA 68 83 D8 33 D6 02
    [Frame 2] =>  00 00 20 00 A8 00 00 49 FF C5 BF 7E B9 39 3D C7 2C F4 D6 6C 3D 7A 68 DF CB DE 18 7A EC DF 5C 3D F9 74 6C 6F
)

Binary version:

Array
(
    [0] => 0000000000000000100001110000000100100000000000000000000001001001111111111110101101101110010111101001101000110111111001001000001111011000001000111100010110101001010111010111001111000111101000110
0101110110111001011000001111011000001001000101010010101001010000101111110001101110110001101011101011100001100001111010011010001000001111011000001011111100100110111111000111011111110100110010111100111111010
0101110110100011101001101101110110010101001000110111111100111110100100000111101100000111011111101111110111011101110111101001100101111001111101001100011100001100011011101100011110111111011011010100000111000111
1000111100101111000011101001000001111011000001101001110110111000011100111110010110111111001111101001110011010110010000011010011101101110000111001111100101101111110101011100001100101110011110110010000001010
10101111101010101010000111101111110111011101001100101110111011101001011011011001100101011101110110011111101001101000100000111101100000110011110101101 0
    [1] => 0000000000000000001000000000000010101000000000000000000001001001111111111100010110111111011111101011100100111001001111011100011100101100111101001101011001101100001111010111101001101000110111111
1001011110111100001100001111010111011001101111101011100001111011111100101110100011011001101111
)
Example usage 2:
****************

Original request:

PHP Warning:  Undefined variable $http2_request_string in /Users/ajitashrivastava/Desktop/CS297/deliverables/deliverable2/parser.php on line 302

Warning: Undefined variable $http2_request_string in /Users/ajitashrivastava/Desktop/CS297/deliverables/deliverable2/parser.php on line 302


Parsed request array:
```



```
Array
(
    [method] => POST
    [path] => /api/login
    [protocol] => HTTP/1.1
    [headers] => Array
        (
            [Host] => example.com
            [Content-Type] => application/json
            [Content-Length] => 42
            [Authorization] => Bearer token123
        )

    [body] => {
"username": "example_user",
"password": "example_password"
}
)


Frames created:

Array
(
    [headerFrame] => Array
        (
            [length] => 156
            [type] => 0x1
            [flags] => Array
                (
                    [end_stream_flag] => 0
                    [end_header_flag] => 1
                    [padded_flag] => 0
                    [prior_flag] => 0
                )

            [R] => 0
            [streamID] => 18943
            [padLength] => not set
            [E] => not set
            [StreamDependency] => not set
            [weight] => not set
            [requestHeaders] => :method = POST
:scheme = HTTP/1.1
:path = /api/login
Host = example.com
Content-Type = application/json
Content-Length = 42
Authorization = Bearer token123

            [padding] => not set
        )
```



```
    [dataFrame] => Array
        (
            [length] => 65
            [type] => 0x0
            [flags] => Array
                (
                    [end_stream_flag] => 1
                    [end_header_flag] => 1
                    [padded_flag] => 0
                    [prior_flag] => 0
                )

            [R] => 0
            [streamID] => 18943
            [padLength] => not set
            [data] => {
"username": "example_user",
"password": "example_password"
}
            [padding] => not set
        )

)


Hexadecimal version:

Array
(
    [Frame 1] =>  00 00 9C 01 20 00 00 49 FF EB 6E 5E 9A 37 E4 83 D8 28 4F A7 52 BA E7 8F 46 5D B9 60 F6 09 15 2A 50 BF 18 81 AE B8 61 E9 A2 0F 60 BF 0F 0D 37 EC DF 9F 4E EA 91 BF 9F 48 3D 83 2F 8C 3B 78
6C CB 76 3D FB 6A 1E FD DD 32 EE E9 6D 4F 3C 32 C1 EC 18 78 70 D9 A7 1E 1E 9A 77 EE BF 57 3D FB AA 1E FD DD 32 EE E9 6C CC 8B B3 F4 D1 07 B0 69 95 41 EB D3 46 FE 5A 7D 61 E9 A7 7E E8 3D 82 16 5C 3C B2 F2
83 A6 FD 79 77 63 96 3A
    [Frame 2] =>  00 00 41 00 A8 00 00 49 FF F7 B5 17 5E 79 79 6E C3 B7 2C 5D 41 16 5F 18 76 F0 D9 96 FF 5E 79 79 45 66 D4 5C 30 F3 E7 DF 7F 2C 91 75 04 59 7C 61 DB C3 66 5B FC 30 F3 E7 DF 7F 2C 91 6D 7D
)

Binary version:

Array
(
    [0] => 000000000000000010011100000000000100100000000000000000000000000010010011111111111101011011011100101111010011010001101111110010010000011110110000001010000100111110100111010100101011101011100111100011110
1000110010111011011100101100000111101100000010010001010100101010010100001011111100011011101100011010111010111000011000011110100110100010000011110110000001011111100001111000011010011011111101100110111111100 11
1110100111011101010100100011011111110011111010010000011110110000011001011111000110000111011011110000110110011001011011101100011110111111011011010100001111011111101110111010011001011101110111010010110110 10
1001111001111000001100101100000111101100000110000111100000111000001101100110100111000111100000111101001101001110111111101110101111110101011100111101111110111010101000011110111111011101110100110010111011101101
0010110110011001100101110111011001111110100110100010000011110110000011010011001010101000000111101111010011010001101111111001011010011111010110000111101001101001110111111011101000001111011000001000001011 00
1011100001111001011001011110010100000111010011011111101011110010111011101100011100101100111010
    [1] => 0000000000000000100000100000000010100000000000000000000000000001001001111111111110111101101010001011101011110011110010111100101101110110000111011011100101100010111010100000100010110010111110001100 00
1110110111100000110110011001011011111111010111001111001011110010100010101100110110101000101110000011000001111001111100111110111110111111110010110010010001011101010000010001011001011111000110000111011011110 00
01101100110010110111111111100000110000011110011111001111101111101111111001011001001000101101101011111101
)
ajitashrivastava@Ajitas-MacBook-Pro deliverable2 % 
```

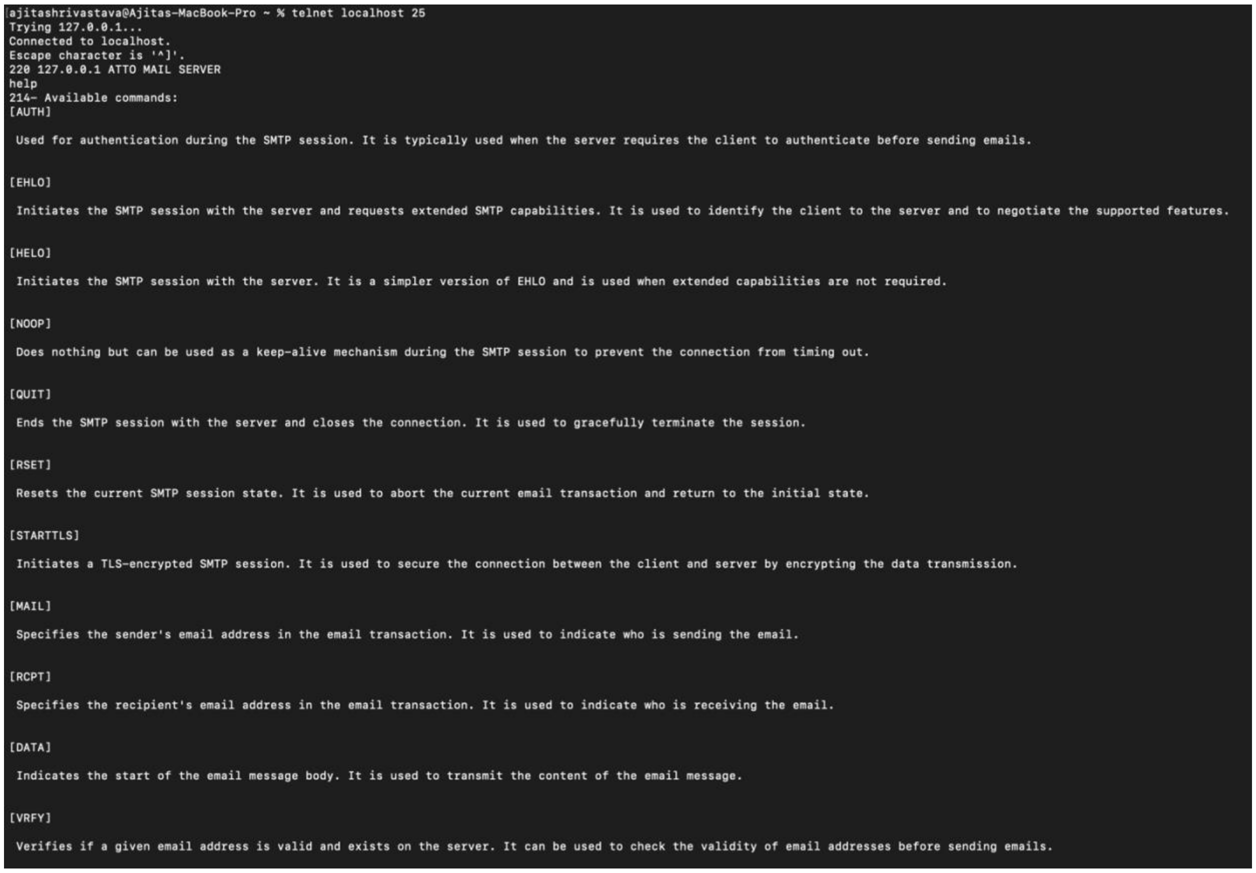
Figure 5: Results of the conversion program

IV. Deliverable 3: Additional Feature Implementation

In this deliverable, the implementation of the HELP command feature in MailSite.php is achieved. This feature enables the server to provide assistance to clients by offering detailed information about available commands. The HELP command does not affect the core functionalities of MailSite, such as the reverse-path buffer, forward-path buffer, or the mail data buffer. It can be invoked by clients at any point during communication. The HELP command's response follows a structured format, starting with a header indicating the beginning of available commands. It then lists each command along with its corresponding description, concluding with an end marker. The implementation involves several steps:

The first step involves creating a JSON file named 'cmds.json' that contains a list of all commands and their descriptions. This file format is chosen for its suitability in storing key-value pair information. Next, the 'parseHelp' function is implemented. This function reads the 'cmds.json' file, decodes its contents, and formats them into a printable string. It handles any file-related errors that may occur during the process. The 'parseHelp' function is then integrated into the 'parseRequest' function, which is responsible for parsing client requests. The response generated by 'parseHelp' is added to the 'out_stream' for transmission to the client. Lastly, the 'processRequestStreams' function is updated to accommodate the HELP command. Proper checks are implemented to handle the new command, and the server's response, including the HELP information, is included in the 'out_stream' and returned to the client.

Overall, the addition of the HELP command feature enhances the usability and user experience of MailSite, providing clients with valuable assistance and improving communication between the client and server. Below are some illustrations of the commands json file and the results.



```
ajitashrivastava@Ajitas-MacBook-Pro ~ % telnet localhost 25
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
220 127.0.0.1 ATTO MAIL SERVER
help
214- Available commands:
[AUTH]

 Used for authentication during the SMTP session. It is typically used when the server requires the client to authenticate before sending emails.

[EHLO]

 Initiates the SMTP session with the server and requests extended SMTP capabilities. It is used to identify the client to the server and to negotiate the supported features.

[HELO]

 Initiates the SMTP session with the server. It is a simpler version of EHLO and is used when extended capabilities are not required.

[NOOP]

 Does nothing but can be used as a keep-alive mechanism during the SMTP session to prevent the connection from timing out.

[QUIT]

 Ends the SMTP session with the server and closes the connection. It is used to gracefully terminate the session.

[RSET]

 Resets the current SMTP session state. It is used to abort the current email transaction and return to the initial state.

[STARTTLS]

 Initiates a TLS-encrypted SMTP session. It is used to secure the connection between the client and server by encrypting the data transmission.

[MAIL]

 Specifies the sender's email address in the email transaction. It is used to indicate who is sending the email.

[RCPT]

 Specifies the recipient's email address in the email transaction. It is used to indicate who is receiving the email.

[DATA]

 Indicates the start of the email message body. It is used to transmit the content of the email message.

[VRFY]

 Verifies if a given email address is valid and exists on the server. It can be used to check the validity of email addresses before sending emails.
```

Figure 6: Results of the HELP command invoked

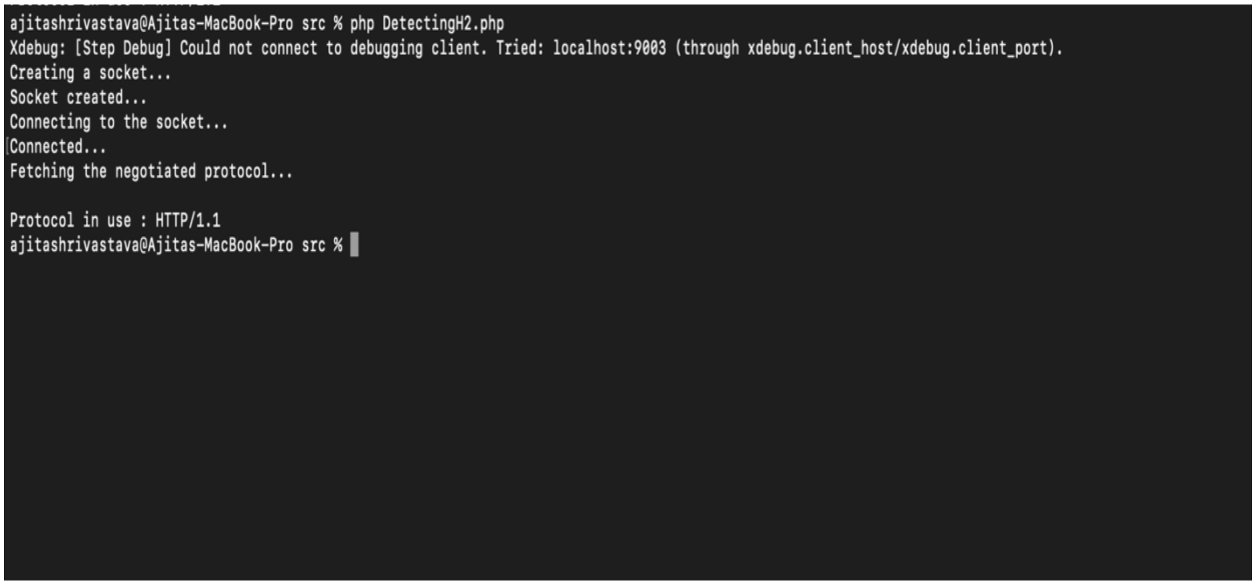
V. Deliverable 4: HTTP/2 IDetection

This deliverable entails the development of a PHP program aimed at establishing a connection with a server and retrieving the negotiated protocol between the client and server. The ‘negotiated protocol extraction’ plays a vital role in the project as it aims to eventually support all versions of HTTP. This part will help determine the appropriate responses accordingly.

The initial step involves the creation of a TCP/IP socket using PHP's socket functions, enabling a channel for communication. The program tries to establish a connection to the designated server through the utilization of the socket_connect() function. This connection attempt either successfully establishes communication channels or prompts an error message in the event of connection failure.

Upon successfully establishing a connection, the program proceeds to extract the negotiated protocol from the handshake packets exchanged during the initial communication between the client and server. This extraction process entails dispatching a request to the server and meticulously capturing the protocol information encapsulated within the handshake packets. Through methodical dissection and analysis of the handshake protocol, the program discerns the mutually agreed-upon protocol designated for subsequent interactions between the client and server.

By systematically executing these procedures, the PHP program achieves its fundamental objective of connecting to the server and retrieving the negotiated protocol. This accomplishment serves as a foundation for facilitating seamless and efficient communication channels between the client and server entities, thereby fostering enhanced interoperability and data exchange capabilities across network environments.



```
ajitashrivastava@Ajitas-MacBook-Pro src % php DetectingH2.php
Xdebug: [Step Debug] Could not connect to debugging client. Tried: localhost:9003 (through xdebug.client_host/xdebug.client_port).
Creating a socket...
Socket created...
Connecting to the socket...
Connected...
Fetching the negotiated protocol...

Protocol in use : HTTP/1.1
ajitashrivastava@Ajitas-MacBook-Pro src %
```

Figure 7: Results of the detection program

VI. Conclusion

In conclusion, the progression of this project has been instrumental in advancing the realm of web application development and email server management. The development of single-file, low-dependency, pure PHP servers and routing engines, exemplified by WebSite and GopherSite, has ushered in a new era of scalability, optimization, and rapid development in web environments. By embracing an event-driven, asynchronous I/O approach, these servers offer developers unparalleled flexibility in managing web traffic within both traditional server setups and standalone applications.

Additionally, the project's exploration into email server management represents a significant leap forward in simplifying the complexities associated with setting up and managing email servers. Through the implementation of SMTP and IMAP protocols within single-file PHP servers, the project aims to streamline email server configurations, reduce external software dependencies, and democratize the process of email server management.

Each deliverable achieved throughout the project's lifecycle has played a crucial role in propelling its objectives forward. From laying the groundwork with atto servers to navigating the intricacies of detecting HTTP/2 and negotiating protocols, each task has contributed to a deeper understanding of server technology and its practical implications for web development.

As the project looks towards the future, the insights gleaned from these milestones provide invaluable guidance for continued development and innovation. With a steadfast focus on meeting the evolving needs of developers and users alike, this project stands poised to shape the future of web application development and email server management.

References

[1] Riabov, Vladimir V. "SMTP (Simple Mail Transfer Protocol)." River College, 2005.

[2] Mullet, Dianna, and Kevin Mullet. Managing Imap. O'Reilly Media, Inc., 2000.

[3] Gourley, David, and Brian Totty. HTTP: The Definitive Guide. O'Reilly Media, Inc., 2002.

[4] Pollard, Barry. HTTP/2 in Action. Simon and Schuster, 2019.

[5] Belshe, Mike, Roberto Peon, and Martin Thomson. "Hypertext Transfer Protocol Version 2 (HTTP/2)." No. rfc7540. 2015. RFC Manual.

[6] Langley, Adam, et al. "The QUIC Transport Protocol: Design and Internet-Scale Deployment." Proceedings of the Conference of the ACM Special Interest Group on Data Communication, 2017.

[7] Carlucci, Gaetano, Luca De Cicco, and Saverio Mascolo. "HTTP over UDP: An Experimental Investigation of QUIC." Proceedings of the 30th Annual ACM Symposium on Applied Computing, 2015.

[8] Dent, Kyle D. Postfix: The Definitive Guide: A Secure and Easy-to-Use MTA for UNIX. O'Reilly Media, Inc., 2003. Postfix Overview.

[9] Hildebrandt, Ralf, and Patrick Koetter. The Book of Postfix: State-of-the-Art Message Transport. No Starch Press, 2005.