WAPH-Web Application Programming and Hacking

Instructor: Dr. Phu Phung

Student

Name: Sai Sandeep Pasham

Email: pashamsp@mail.uc.edu

Figure 1: SaiSandeep Pasham

Lab 2 - Front End Web Development

Overview: This front-end development lab focuses on numerous elements of web development, including core topics like HTML, JavaScript, Ajax, CSS, and the JQuery library.

The first portion of the lab involves creating an HTML web page with basic tags and forms. JavaScript is then incorporated using four methods: inline JS, JS within script tags, JS in an external file, and JS code from a remote repository.

To enhance the webpage's appearance, the HTML page is integrated with CSS. Various CSS methods, including inline CSS, internal CSS, and external CSS, are employed to create an elegant look. Additionally, JQuery is utilized to facilitate AJAX get and post calls to echo.php.

In the final stages of the lab, two web services are incorporated into the HTML code using JQuery Ajax and the fetch method. One service generates a random joke, while the other is designed for age guessing. The process of generating a PDF file from the README.md is accomplished using Pandoc.

https://github.com/pashamsp/waph-pashamsp/blob/main/labs/lab2/README.md

Part 1 : Basic HTML with forms, and JavaScript

Task 1. HTML

A task required creating a basic HTML webpage with tags such as `<h1>`, `<h2>`, `<h3>`, `<a>`, `<img>`, `<form>`. The generated file was called waph-pashamsp.html.

Included file `waph-pashamsp.html`:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>WAPH-SaiSandeep Pasham </title>
</head>
<body>
<div >
    <div id="top">
        <h1>Web Application Programming and Hacking</h1>
        <h2>Front End Development Lab </h2>
        <h3>Instructor : Dr Phu Phung</h3>
    </div>
    <div >
        <div id="menubar">
        <h3>Student : Sai Sandeep Pasham</h3>
        <img src="images/headshot.jpg" alt="SaiSandeep headshot" width="50">
        </div>
        <div id="main">
            <p>A Simple HTML Page</p>
            Using the <a href="https://www.w3schools.com/html">W3 Schools Template</a>
            <hr>
            <b>Interaction with HTTP Forms</b>
            <div>
                <i>Form with HTTP GET Request</i>
                <form action="/echo.php" method="GET">
                <lable for="data">Enter the input text</lable>
                <input type="text" name="data" onkeyup="console.log('clicked on a Key')">
                <input type="submit" value="submit">
                </form>
            </div>
            <div>
                <i>Form with HTTP POST Request</i>
                <form action="/echo.php" method="POST">
                <lable for="data">Enter the input text</lable>
                <input type="text" name="data" onkeyup="console.log('clicked on a Key')">
                <input type="submit" value="submit">
                </form>
```

```
            </div>
        </div>
    </div>
</div>
</body>
</html>
```

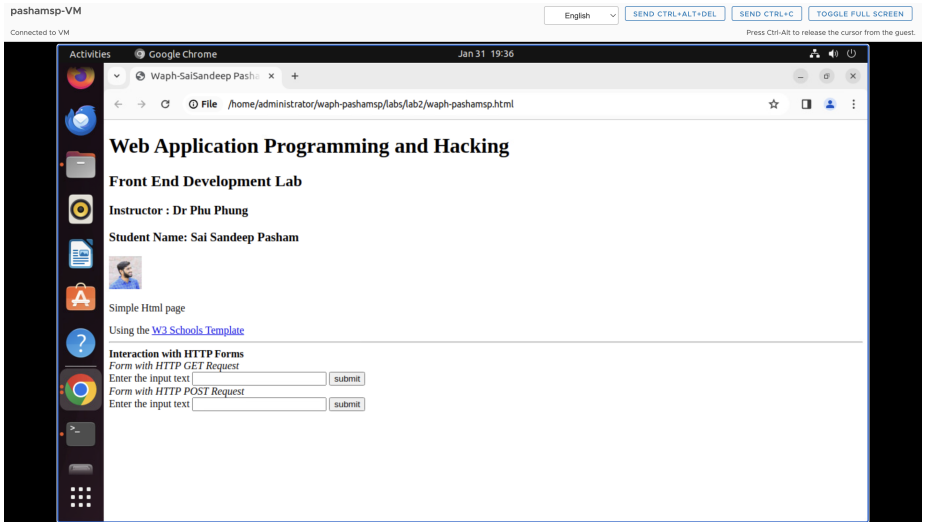

Figure 2: A simple HTML Page

Task 2. Simple JavaScript

This task provided a fundamental introduction to JavaScript syntax and various methods of incorporating JavaScript code into HTML files.

-Inline JavaScript code was utilized to showcase the current date and time upon clicking, and it was also employed to record the onClick event in the console.

```
    <div>
        <b>Experiments with JavaScript code</b><br>
        <i>Inlined JavaScript</i>
        <div id="il_Date"
            onClick="document.getElementById('il_Date').innerHTML=Date();
                console.log('clicked on a key')">Click here to show date()</div>
    </div>
```

-JavaScript code in a `<script>` tag to display a digital clock.

```
<script>
        function displayTime() {
            document.getElementById('digit-clock').innerHTML=" Current_Time is : "+ Date();
```

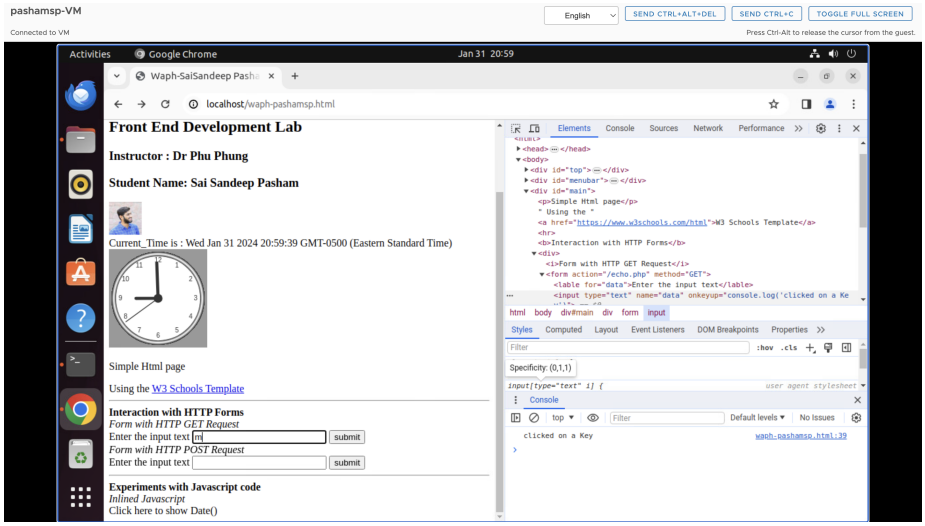

Figure 3: Console screen when clicked

```
        }
        setInterval(displayTime,500);
</script>
```

-When an email is clicked, JS code in the JS file and HTML code in the page are used to show or conceal it.

```
var visible = false;
function showOrHideEmail(){
 if (visible){
    document.getElementById('email').innerHTML=" Show my Email";
        visible=false;
     }
else{
    var myEmail="<a href='mailto:pashamsp" +"@"+
            "mail.uc.edu'>pashamsp"+"@"+"mail.uc.edu</a>";
    document.getElementById('email').innerHTML=myEmail;
    visible= true;
    }
}
```

```
<div id="email" onclick="showOrHideEmail()">my email</div>
<script type="text/javascript" src="email.js"></script>
```

-Creating an analog clock from external JavaScript code and embedding it in an HTML page..

```
<canvas id="analog-clock" width="150" height="150" style="background-color:#999"></canvas>
```

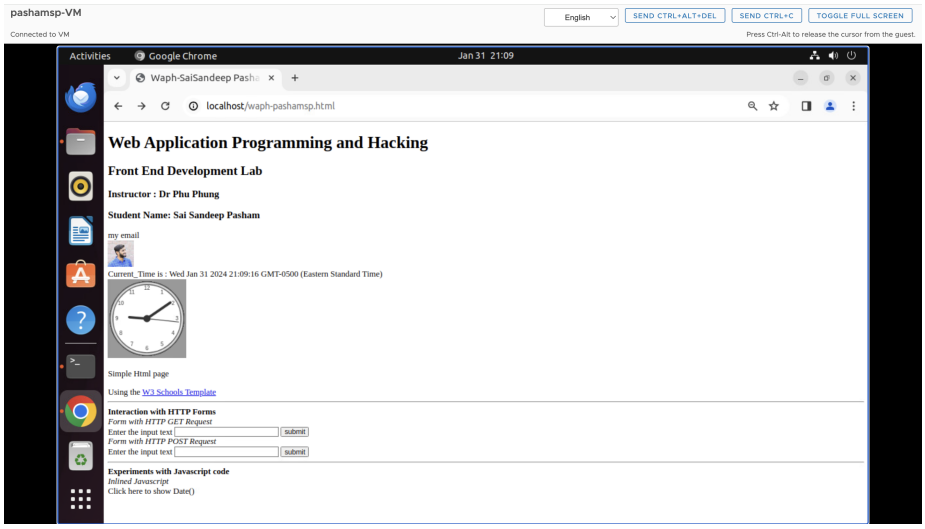

Figure 4: webpage after adding javascript code

```
<script src="https://waph-uc.github.io/clock.js"></script>
<script type=text/javascript>
            var canvas=document.getElementById("analog-clock");
            var ctx=canvas.getContext("2d");
            var radius = canvas.height/2;
            ctx.translate(radius,radius);
            radius=radius*0.90;
            setInterval(drawClock,1000);
            function drawClock(){
                drawFace(ctx,radius);
                drawNumbers(ctx,radius);
                drawTime(ctx,radius);
                }
</script>
```

Part II - Ajax, CSS, jQuery, and Web API integration

Task 1: Ajax

The HTML code has been designed to capture user input and send a GET call to echo.php via AJAX. The resulting response is then shown in a specific div. Because it is a GET request, the input is sent as a path variable in the URL.

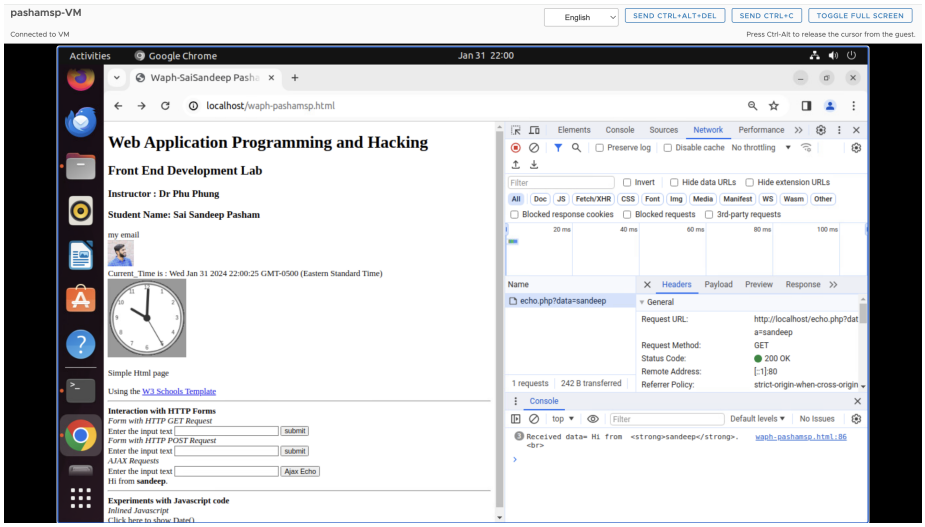

Figure 5: Making an Ajax get call with sandeep as input

```
<div>
    <i>AJAX Requests</i><br>
    <lable for="data">Enter the input text</lable>
    <input type="text" name="data" id="data">
    <input type="submit" value="Ajax Echo" onclick="getEcho()">
    <div id="response"></div>
</div>
<script>
    function getEcho(){
        var input = document.getElementById("data").value;
        if(input.length==0){
        return ;
        }
        var xhttp = new XMLHttpRequest();
        xhttp.onreadystatechange = function(){
    //alert("readyState "+ this.readyState +", status "+this.status+", statusText= "+th
        if(this.readyState==4 && this.status==200){
            console.log("Received data= "+xhttp.responseText);
            document.getElementById("response").innerHTML= xhttp.responseText;
```

```
            }
            }
            xhttp.open("GET", "echo.php?data="+input, true);
            xhttp.send();
            document.getElementById("data").value="";
            }
    </script>
```

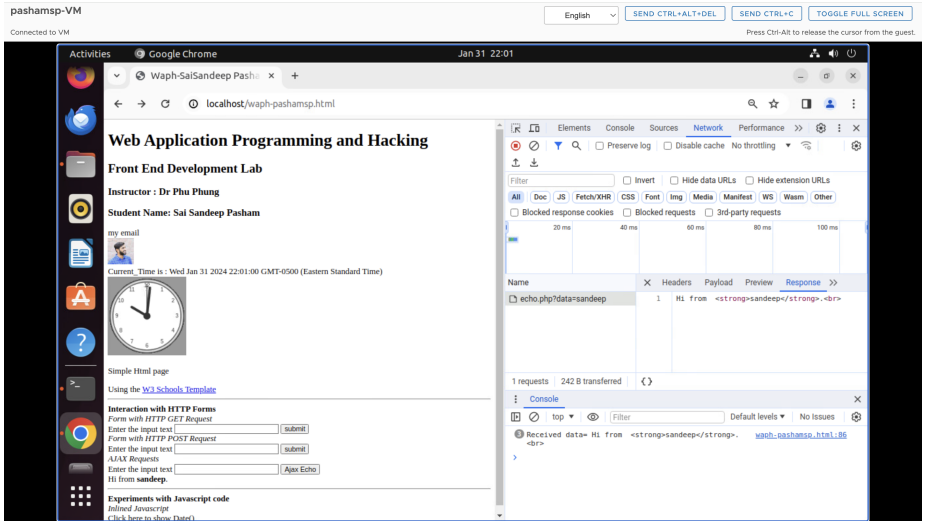

Figure 6: Inspecting the response of Ajax call

The inspect view was used to investigate the Ajax call response. The request method was GET, and the response code was 200 OK. The input information was included in the URL.

Task 2: CSS

a) Inline CSS

```
<body style="background-color: lavender;">
<h1 style="color: coral;">Web Application Programming and Hacking</h1>
```

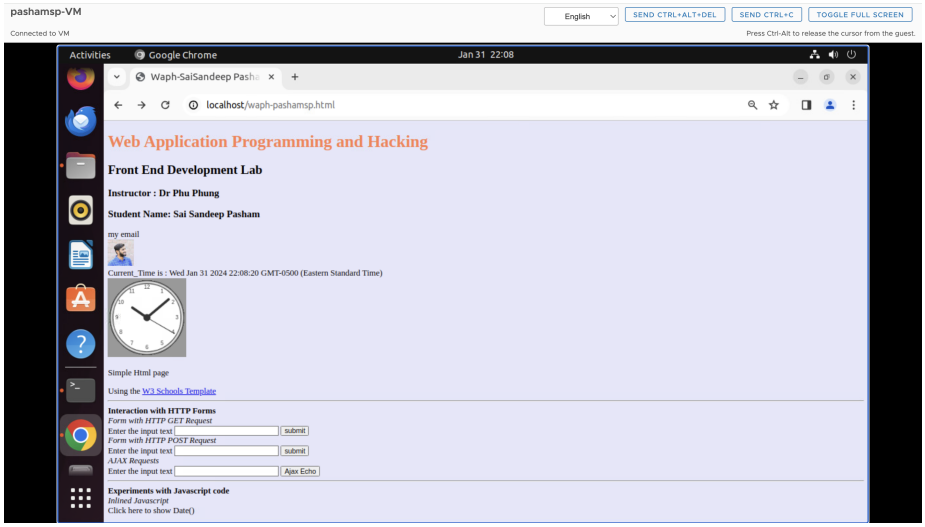

Figure 7: modifed webpage after adding inline CSS

b) Internal CSS.

```
<style>
    .button{
        background-color:#4CAF50;
        border:none;
        color:white;
        padding:5px;
        text-align:center;
        text-decoration:none;
        display:inline-block;
        font-size:12px;
        margin:4px2px;
        cursor:pointer;
    }
    .round{
        border-radius:8px;
    }
    #response{
        background-color:#ff9800;
    }
```

```
<!-- HTML code -->
</style>
<input class="button round" type="submit" value="Ajax Echo" onclick="getEcho()">
<input class="button round" type="submit" value="JQuery Ajax Echo" onclick="getJqueryAja
<input class="button round" type="submit" value="JQuery Ajax Echo Post" onclick="getJque
<div id="response"></div>
```

c) External CSS from the remote repository provided in the lecture.https://waph-uc.github.io/style1.css.

```
<link rel="stylesheet" type="text/css" href="https://waph-uc.github.io/style1.css">
<!-- HTML code -->
<div class="container wrapper">
<!-- HTML code -->
    <div class="wrapper">
<!-- HTML code -->
    </div>
</div>
```

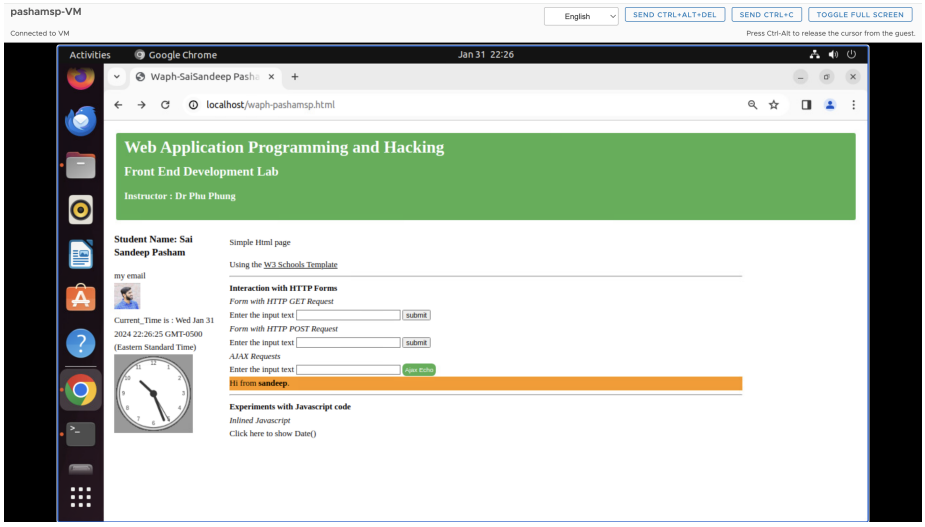

Figure 8: web page after adding internal CSS and external CSS

Task 3: JQuery

The JQuery library is now included in the HTML code, along with two buttons, Jquery Ajax Get and Jquery Ajax Post. These buttons use JQuery to facilitate GET and POST calls to the echo.php file.

i. The inpect view analyses the response to an Ajax GET request to echo.php. The call was GET, and the status code was 200 OK.

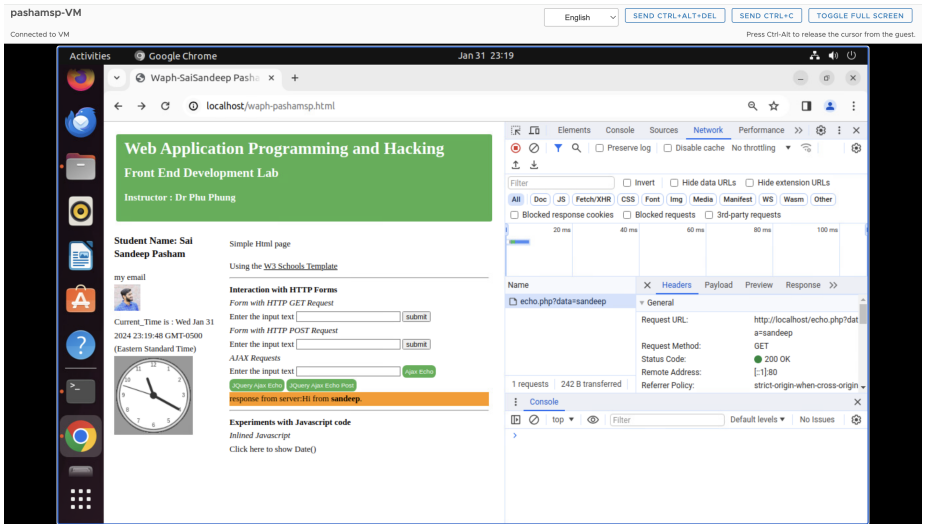

Figure 9: JQuery Ajax GET request to echo.php

```
<!-- HTML code -->
<input class="button round" type="submit" value="JQuery Ajax Echo" onclick="getJqueryAja
<!-- HTML code -->
<script>
    function getJqueryAjax(){
        var input=$("#data").val();
            if(input.length==0)
                return;
        $.get("echo.php?data="+input,
                  function(result){
                      printResult(result);
                  });
        $("#data").val("");
        }
    function printResult(result){
        $("#response").html(result);
        }
</script>
```

i. The inpect view analyses the response to an Ajax POST request to echo.php. The call was POST, and the status code was 200 OK.

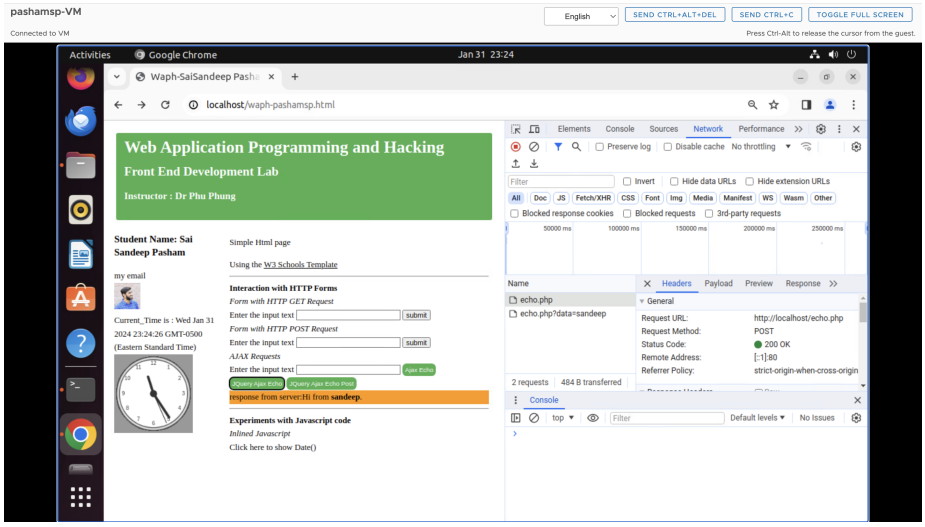

Figure 10: JQuery Ajax POST request to echo.php

```
<!-- HTML code -->
<input class="button round" type="submit"
    value="JQuery Ajax Echo Post" onclick="getJqueryAjaxPost()">
<!-- HTML code -->
<script>
    function getJqueryAjaxPost(){
        var input=$("#data").val();
        if(input.length==0)
            return;
        $.post("echo.php",{data:input},function(result){
                printResult(result);
                });
        $("#data").val("");
        }
    function printResult(result){
        $("#response").html(result);
        }
</script>
```

Task 4: WEB API Integration.

i. Using Ajax on https://v2.jokeapi.dev/joke/Programming?type=single

Using JavaScript code. jQuery Ajax has been developed to send a GET request to the given web service. The response, which was initially in JSON format, is transformed to a string using the JSON.stringify() method and displayed in the console.

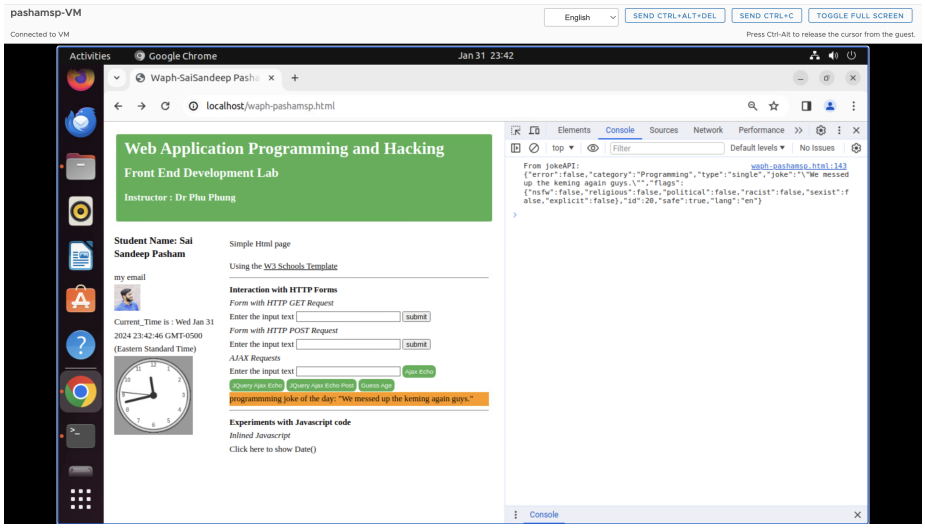

Figure 11: Random Joke displayed when the page is loaded

result.joke extracts the joke from this response. The service generates a new random joke every time the homepage is loaded, resulting in a dynamic and engaging experience with each page refresh.

```
<!-- HTML code -->
<script>
$.get("https://v2.jokeapi.dev/joke/Programming?type=single",function(result){
            console.log("from joke API: "+ JSON.stringify(result));
            $("#response").html("Programming joke of the day: " +result.joke);
            });
</script>
<!-- HTML code -->
```

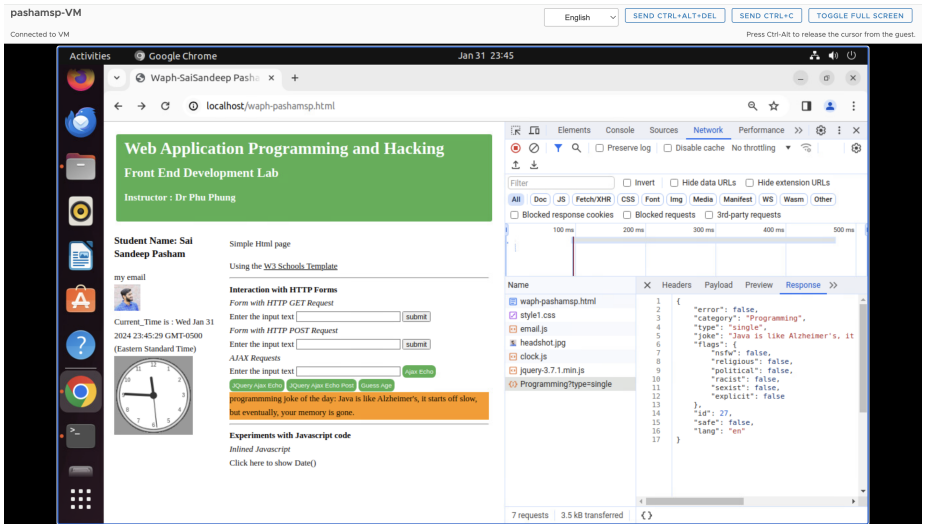

Figure 12: Response of the webservice in inspect view

ii. Using the `fetch` API on https://api.agify.io/?name=input The JavaScript fetch method is used to send an asynchronous HTTP request to the given web service. To accommodate the call's asynchronous nature, the function is marked with the async keyword, and await is used to coordinate the answer. The HTTP request is of type GET, and the expected status code is 200 OK.

```
<script>
async function guessAge(name){
            const response= await fetch("https://api.agify.io/?name="+name);
            const result= await response.json();
            $("#response").html("Hello "+name+" ,your age should be "+result.age);
        }
</script>
```

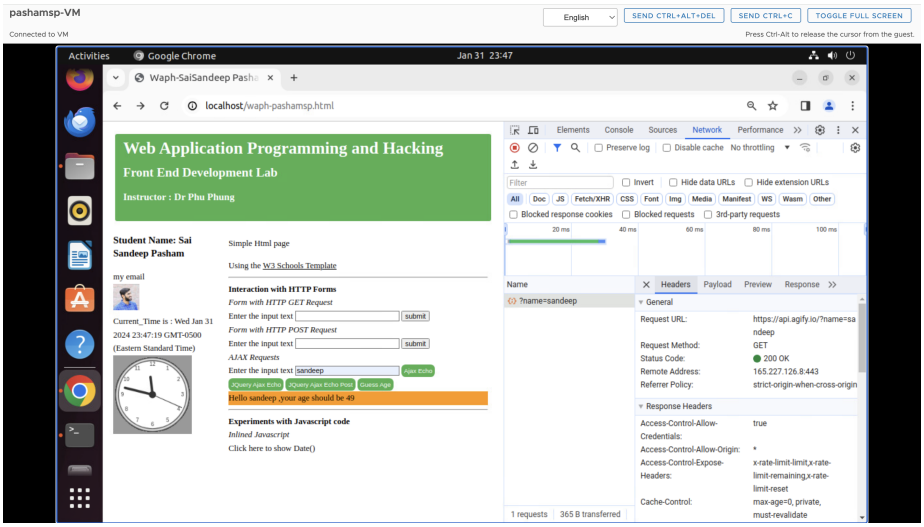

Figure 13: HTTP request to api.agify.io

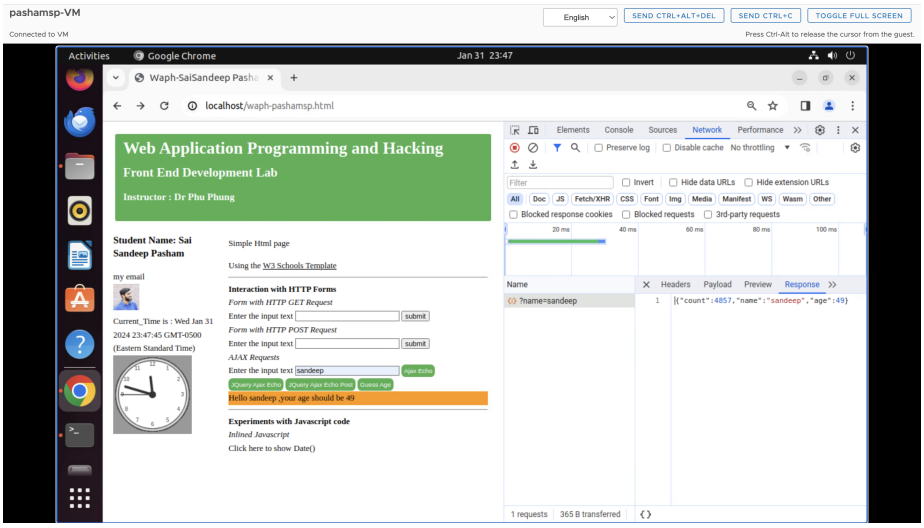

Figure 14: Response from api.agify.io

Below is the final webPage after completing all the tasks.

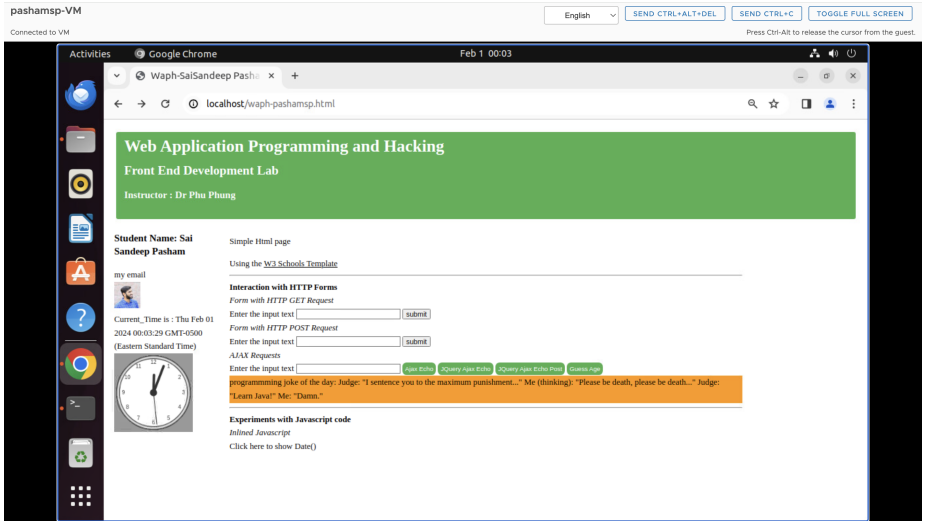

Figure 15: Lab 2 waph-pashamsp.html

Following this, a Labs/Lab2 folder was created to hold the project report, and the changes were pushed. The Pandoc programme generated the project report from the README.md file.