waph-peddinry

WAPH-WEB APPLICATION PROGRAMMING AND HACKING

instructor : Dr.Phu Phung

Name: Narendra Reddy Peddireddy **Email**: peddirny@mail.uc.edu

Repository's URL: https://github.com/narendra2105/waph-peddinry.git

Narendra Reddy Peddireddy uses this private repository to house all of the course's code. The following is how this repository is organized.

Lab's

-lab 2:Front-end Web Development ### Overview of Lab2

The project's first section concentrated on building a simple HTML webpage with forms and using JavaScript to send requests to an Echo.php file. The second section, on the other hand, was more concerned with combining AJAX, CSS, JQuery, and Web API to improve styling and interactivity, which eventually produced a dynamic and interesting webpage.

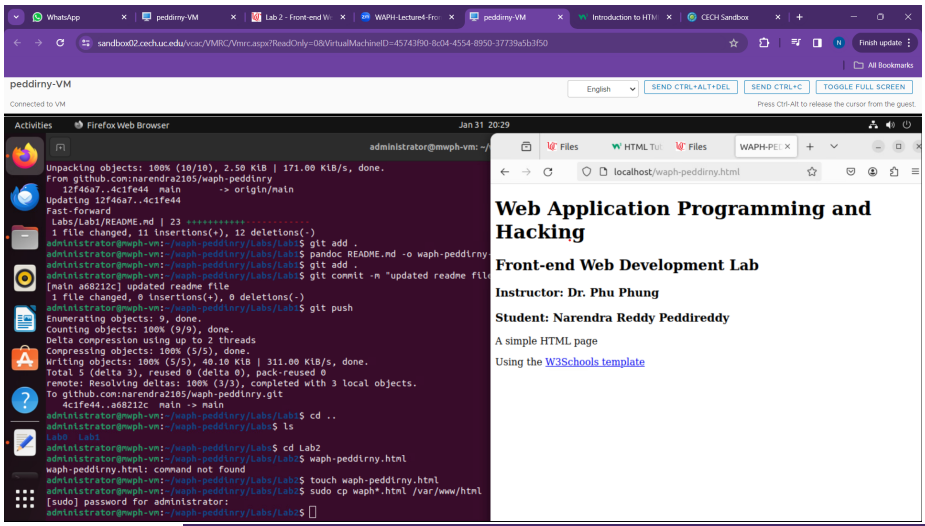
###Hands-on hacking exercises

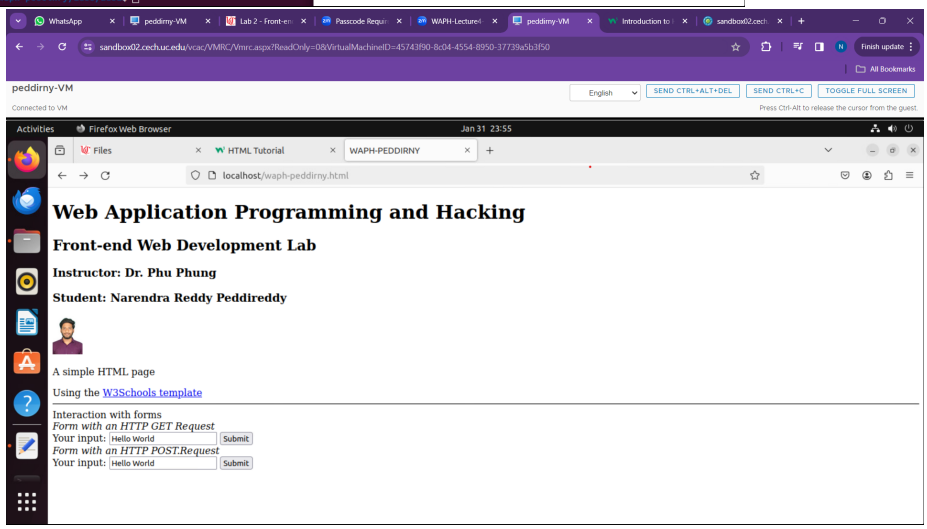
Task1: Simple HTML with JavaScript and forms

A :HTML

I created a simple HTML page called "waph-peddirny.html" with input forms and my headshot image "headshot.png" using the Sublime editor. I then copied the page to "/var/www/html" to deploy it on a server. In addition, I created another HTML file called "waph-peddirny.html" with links to W3Schools, my headshot image, and basic input form tags for responding.

Figure 1: Narendra Reddy Peddireddy Headshot





Http Get

HTTP Get respons

HTTP Post

HTTP post response

B. Simple JavaScript

The browser screenshot illustrates how I've used inline JavaScript inside HTML elements to display the current date and time. In addition, the time is dynamically updated with each key stroke, as shown in the screenshot that appears after a key press.

I employed a

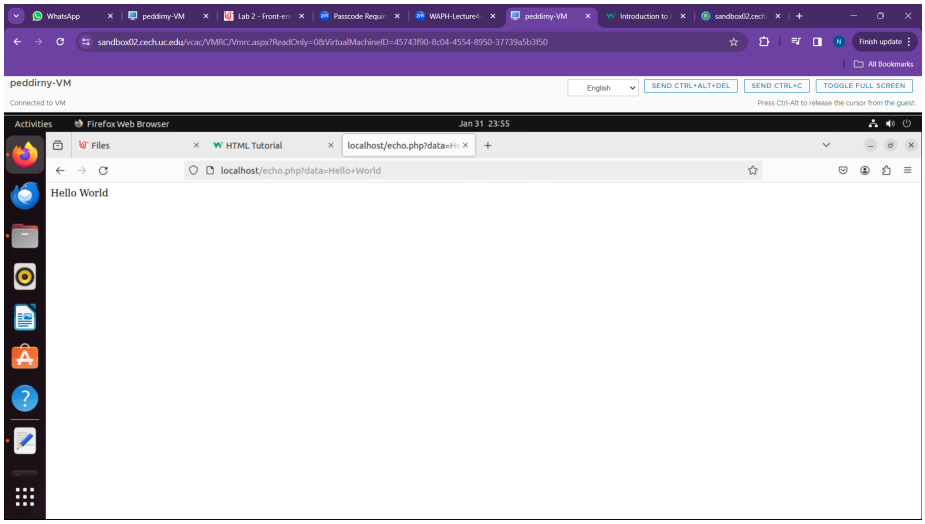


Figure 2: HTTP get responses

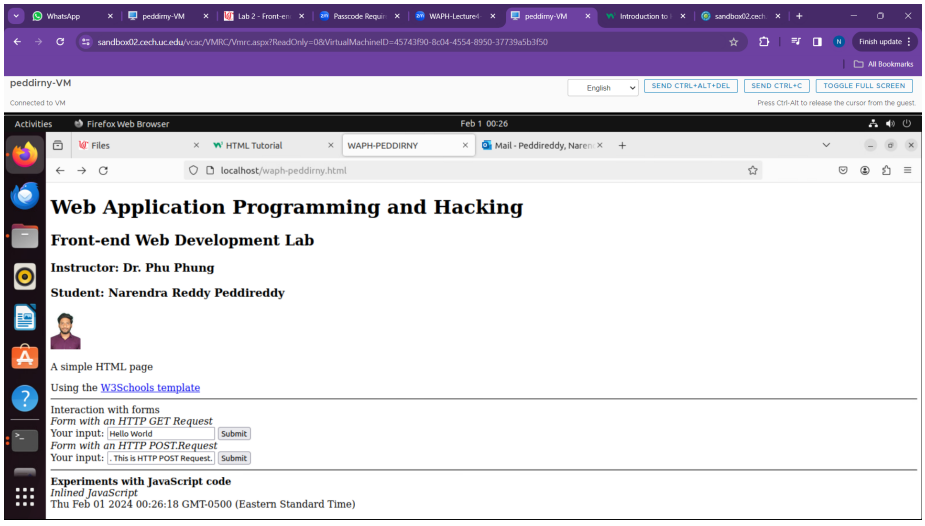


Figure 3: HTTP post

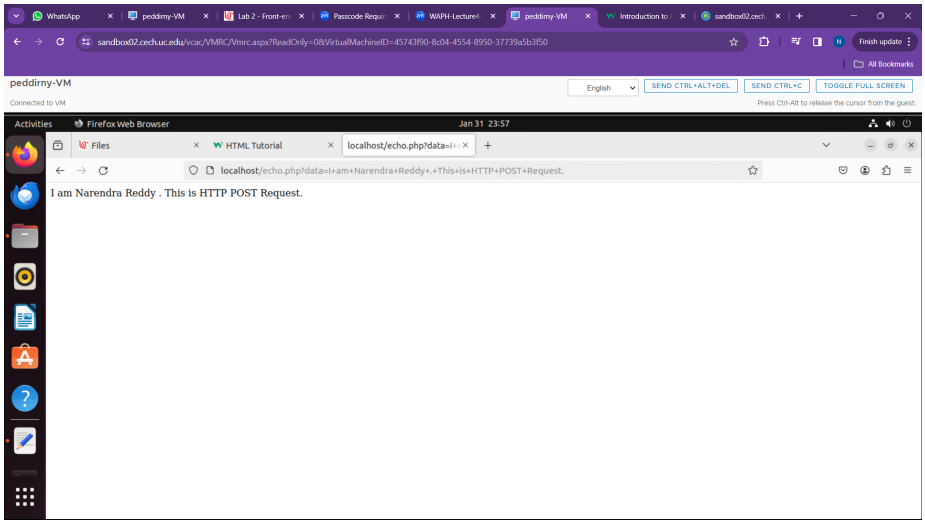


Figure 4: Screenshot4

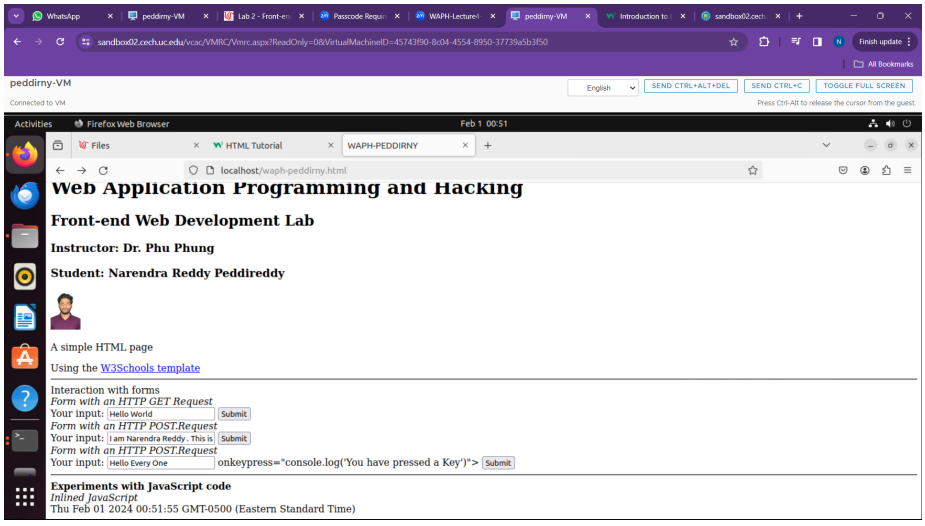


Figure 5: Screenshot5

tag to implement the digital clock display, as depicted in the provided snapshot.

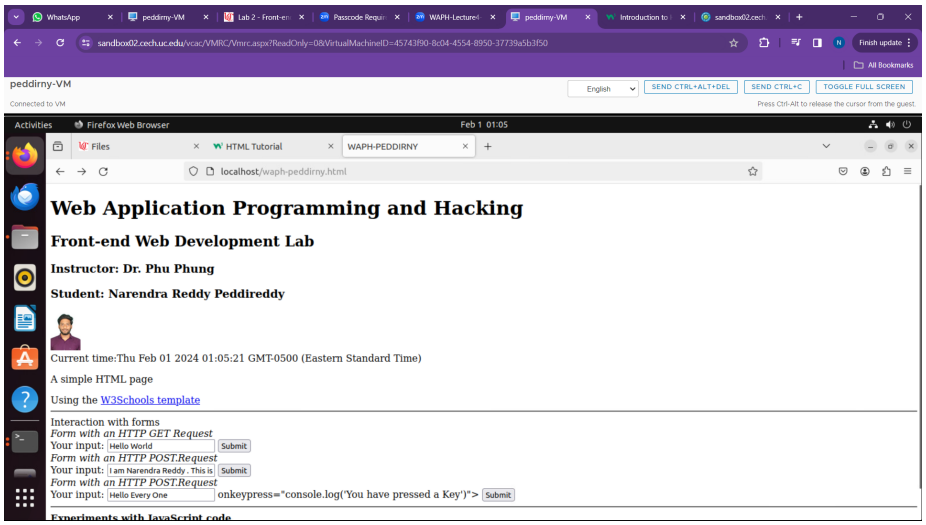
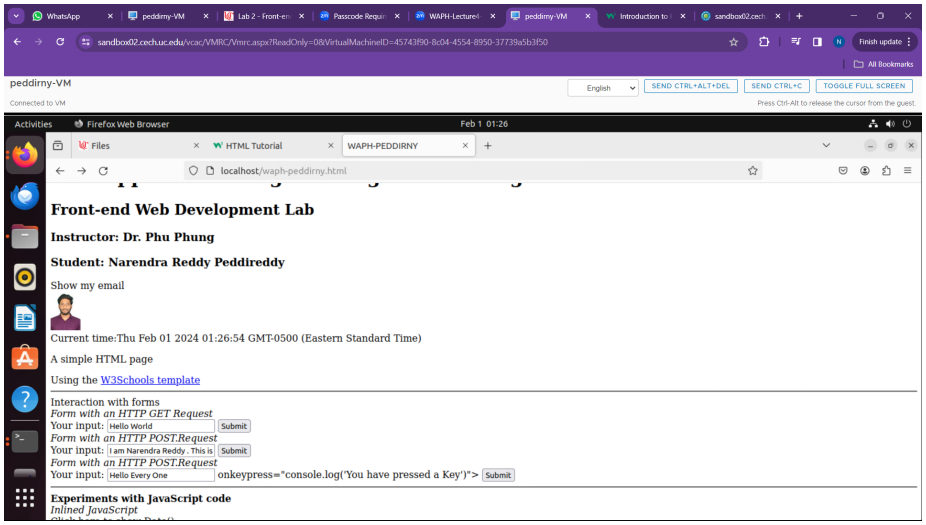


Figure 6: Screenshot6

To save the email address, a “email.js” file was created. It was then integrated into the HTML content using a

tag. Finally, a clickable

element with a onclick attribute was added, which called a method to show or conceal the email address inside the HTML body.



when i click on show my emails it displays my email.Below is the Screenshot

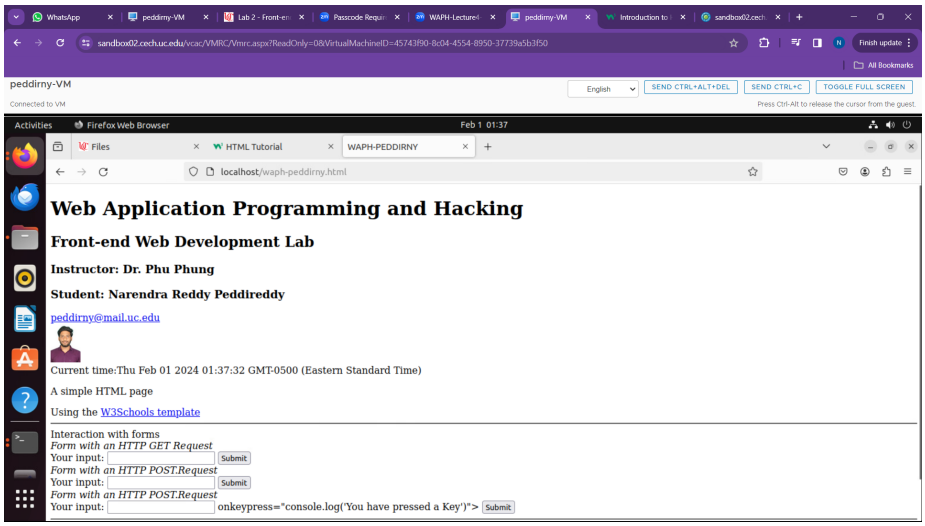


Figure 7: Email display

Analog Clock

I rendered the analog clock using the

element with the id “analog-clock” within the HTML body, as demonstrated in the snapshot that was provided. I included an external JavaScript library using the

tag with the source pointing to “https://waph-uc.github.io/clock.js” in order to incorporate an analog clock display into my HTML document.

Task 2: Integration of Ajax, CSS, jQuery, and Web API

A:Ajax

I created a new input button that, when clicked, launches a JavaScript code. This function uses Ajax to send an asynchronous GET call to “echo.php”; it then displays the content of the response in a specified

element inside the HTML document, allowing it to record user input. In order to show how the request and response processes operate, I also showed how the network connections work using the developer tools in the browser.

Ajax response in browser.

B:CSS

I used the

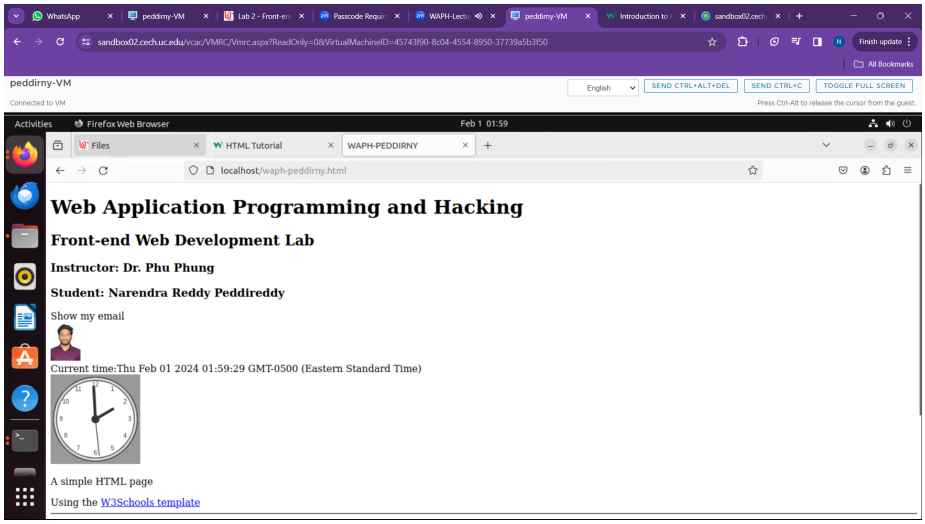


Figure 8: Analog Clock

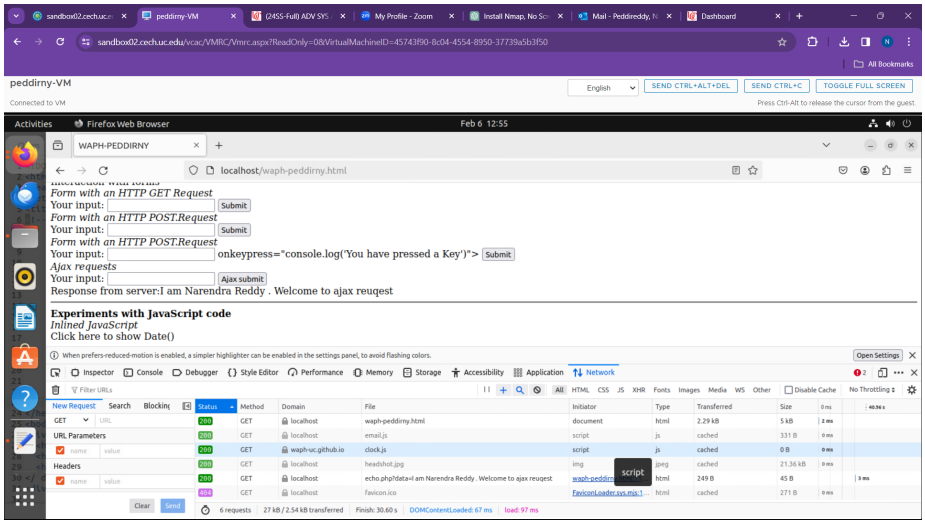


Figure 9: Ajax

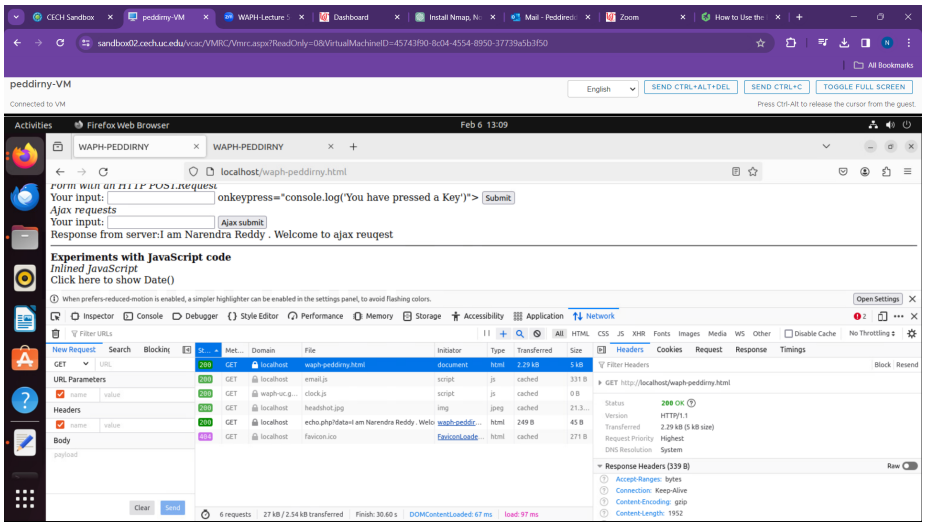


Figure 10: Ajax

tag to define custom styles, like button formatting and colors, and the tag to include an external stylesheet from "https://waph-uc.github.io/style1.css" to apply CSS styling to the HTML webpage. This changes the webpage's overall appearance and structure, as seen in the screenshot of the browser that was provided.

C:jQuery

i Using a

element, I integrated the jQuery library into my HTML document and used tags to construct two buttons. I used jQuery to construct a function called "jQueryAjax()" that uses jQuery to display the server's response after capturing user input and sending an Ajax GET request to "echo.php". The exchanges that occur after clicking the "Ajax Get echo" button are shown in the network properties snapshot.

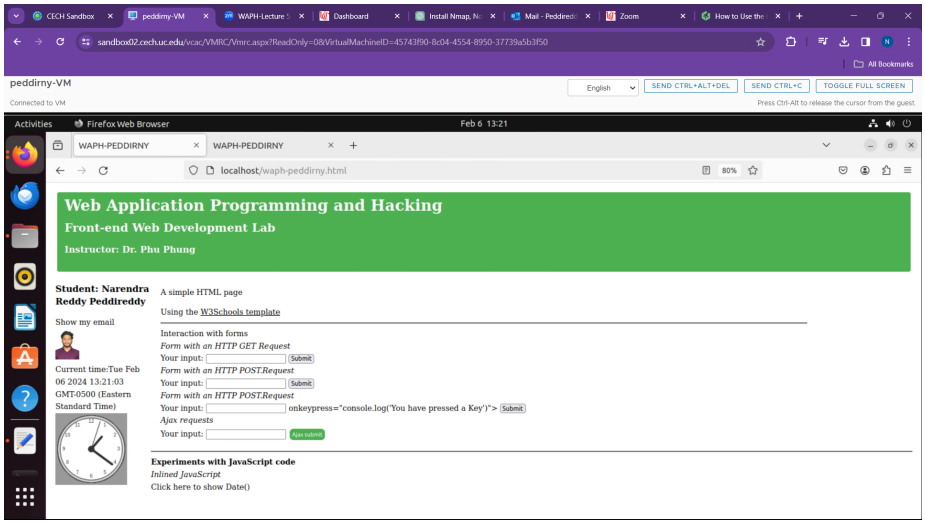


Figure 11: CSS

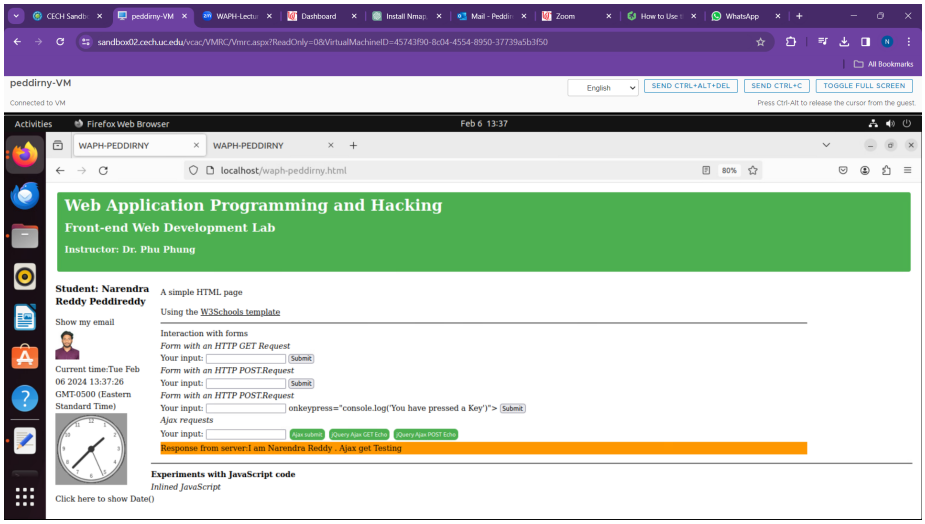


Figure 12: jQuery

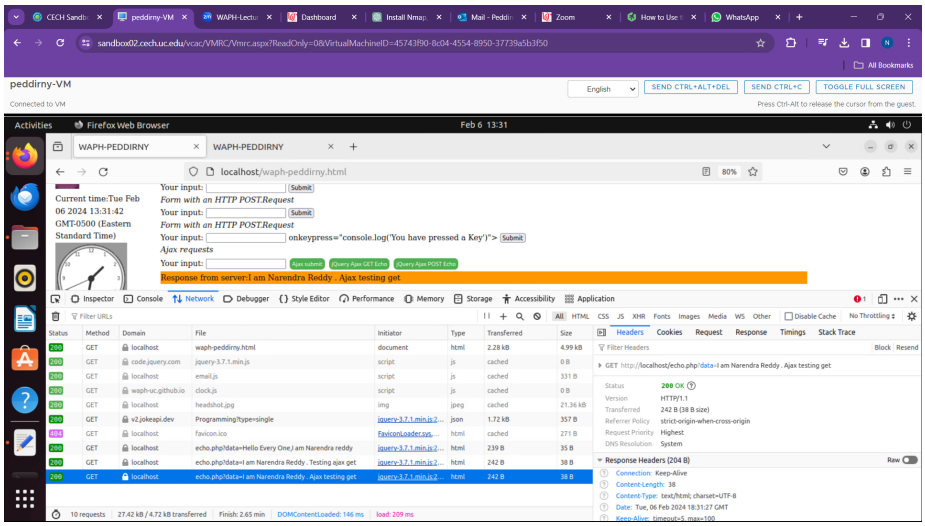
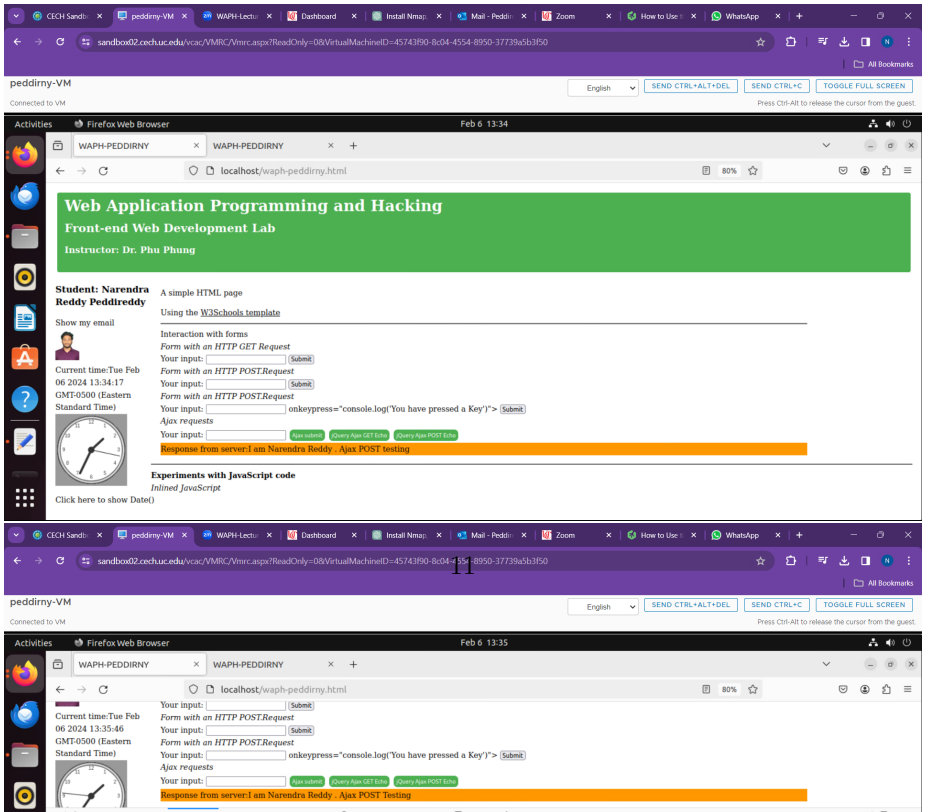


Figure 13: jQuery reponse

jquery Get reponse

ii Using the jQueryAjaxPost() function, I created an Ajax POST request that replicates the previously mentioned capabilities. This function gathers user input, uses jQuery to display the server's answer, then sends a POST request with the input data to "echo.php". The browser and console views during the jQuery POST request are shown in the screenshot that has been supplied. It draws attention to the distinction between GET and POST methods: POST requests are more secure than GET requests because the data they send is not visible in the URL.



D:Web API integration

i I handled the response to extract and display the joke portion by sending a request to the "v2jokeapi.dev" API endpoint using jQuery Ajax. This procedure is demonstrated by the provided JavaScript code, which updates the HTML element with the id "response" to display the day's programming joke and logs the API response in the console. The screenshot that goes with it shows the window that displays the joke that was acquired using the API.

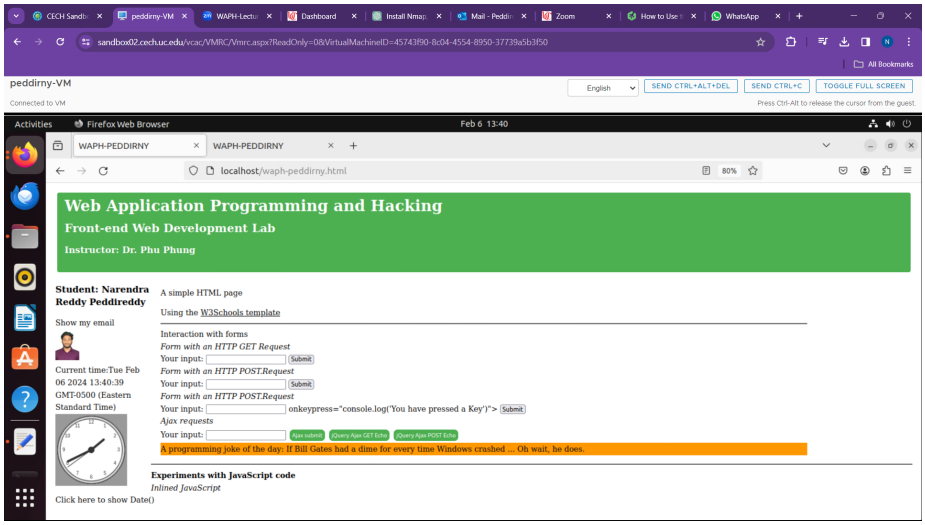


Figure 14: Web API joke

ii I developed a feature that uses user input to randomly retrieve age data from "api.agify.io" using the Fetch API. An asynchronous function is run upon clicking the "Guess Age" button, which calls the guessAge() method. The function obtains information from the API by using the name input, analyzes the result, and modifies the HTML element with the id "response" to exhibit the approximate age. The age information is shown in the snapshot; examining the browser will reveal more details.

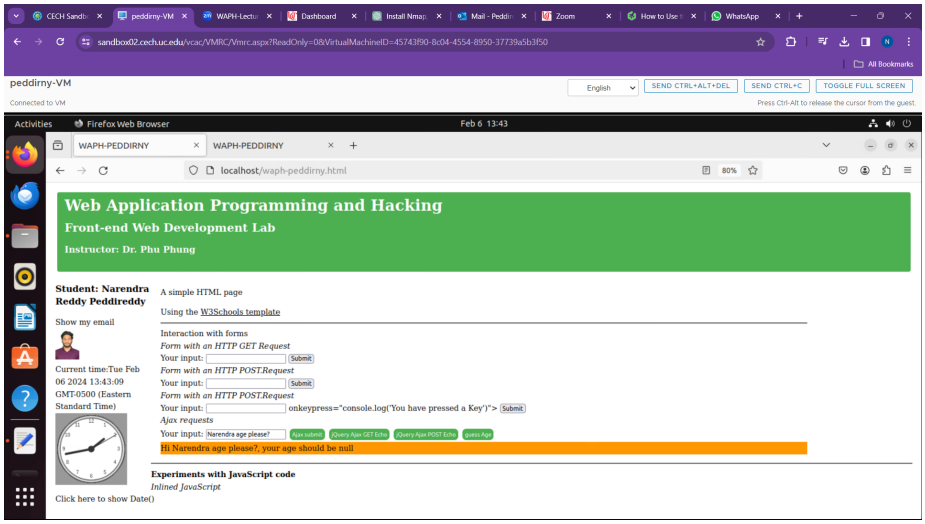


Figure 15: Guess Age

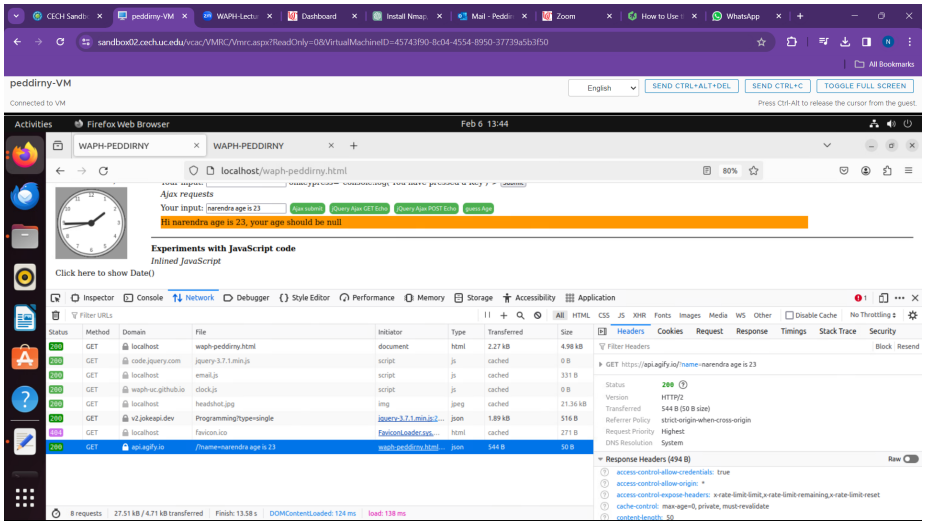


Figure 16: Guess Age response