

WAPH-Web Application Programming and Hacking

Instructor: Dr. Phu Phung

Student

Name: Aditya Pawar

Email: pawarat@mail.uc.edu

Short-bio: Aditya Pawar is a pre-junior majoring in Computer Science.



Figure 1: Aditya's headshot

Last Update: 06/09/2024

Repository Information

Respository's URL: <https://github.com/AdityaPawarr7/AdityaPawarr7.github.io>

This is a private repository for Aditya Pawar to store all code from the course. The organization of this repository is as follows.

Individual Project 1

Front-end Web Development with a Professional Profile Website on github.io cloud service

Overview

For Individual Project 1 in front-end web development, I created and deployed a personal website on GitHub.io as a professional profile. This website includes my name, headshot, contact information, resume, background, education, experiences, and skills. Additionally, I added a new HTML page to introduce the “Web Application Programming and Hacking” course and related hands-on projects. I used an open-source CSS template, specifically Bootstrap, to style my website, making it suitable for potential employers. To track page visits, I integrated Flag Counter. My website includes basic JavaScript features such as a digital clock, an analog clock, a show/hide email feature, and another functionality of

my choice. I also integrated two public web APIs: the JokeAPI, which displays a new joke every minute, and WeatherAPI to show the weather in Cincinnati. Furthermore, I implemented JavaScript cookies to remember returning visitors. If it's their first visit, the website displays "Welcome to my homepage for the first time!" Otherwise, it shows "Welcome back! Your last visit was ,," with this value updating on each visit.

Through this project, I learned important skills in creating a personal portfolio, using Bootstrap for design, and using cookies to keep track of users. I also improved my ability to use JavaScript and public APIs, and in a way applying all the knowledge that we've learned in the lectures until now.

In this project, I expanded on front-end web development skills by developing a Professional Profile Website and deploying it on `github.io` cloud service. This project has general, non-technical, and technical requirements which are met as shown in the content below with screenshots.

General requirements:

- Create and deploy a personal website on GitHub cloud (`github.io`) as a professional profile with your resume, including your name, headshot, contact information, background, e.g., education, your experiences and skills (25 pts).

For this task, we followed the instructions as given in Lecture 6 slides and followed along with Professor Phung to create a new public repository named `AdityaPawarr7.github.io` in order to host our professional portfolio website. On top of that, I added in the experiences on my resume, my name, headshot, contact information, background, education, and skills as well.

The link for the webpage is `AdityaPawarr7.github.io`

The screenshots below show all the information that the requirements talk about.

General Information Screenshot:

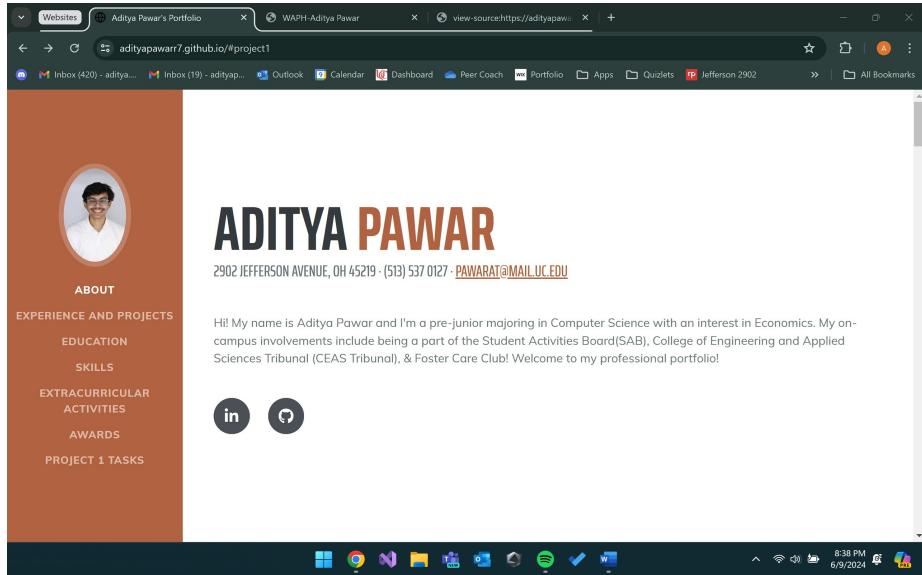


Figure 2: General Information

Experiences and Projects:

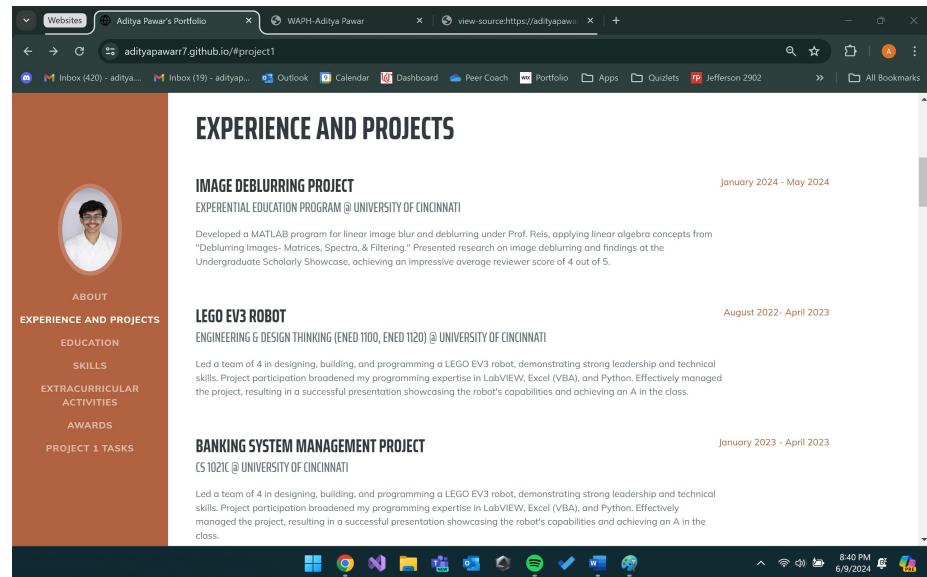


Figure 3: Experiences and Projects

Education:

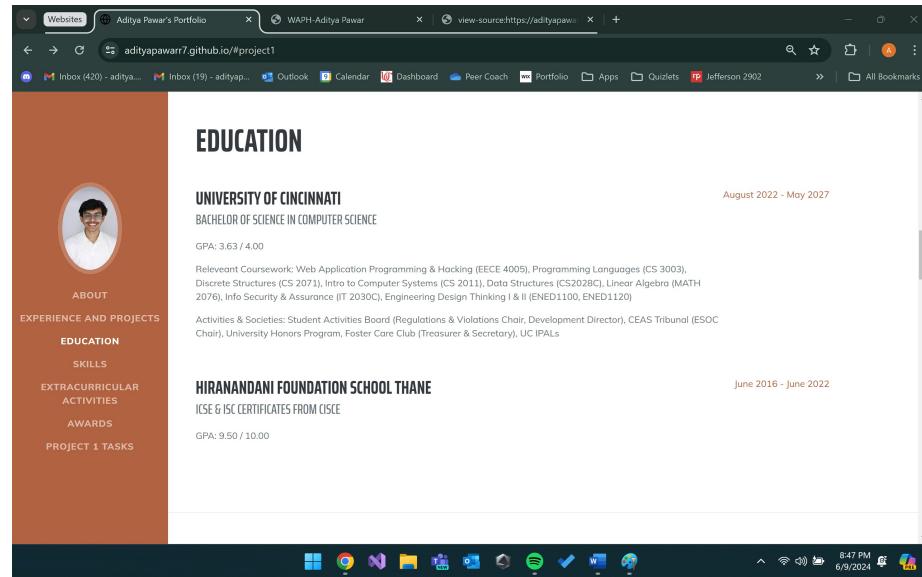


Figure 4: Education

Skills:

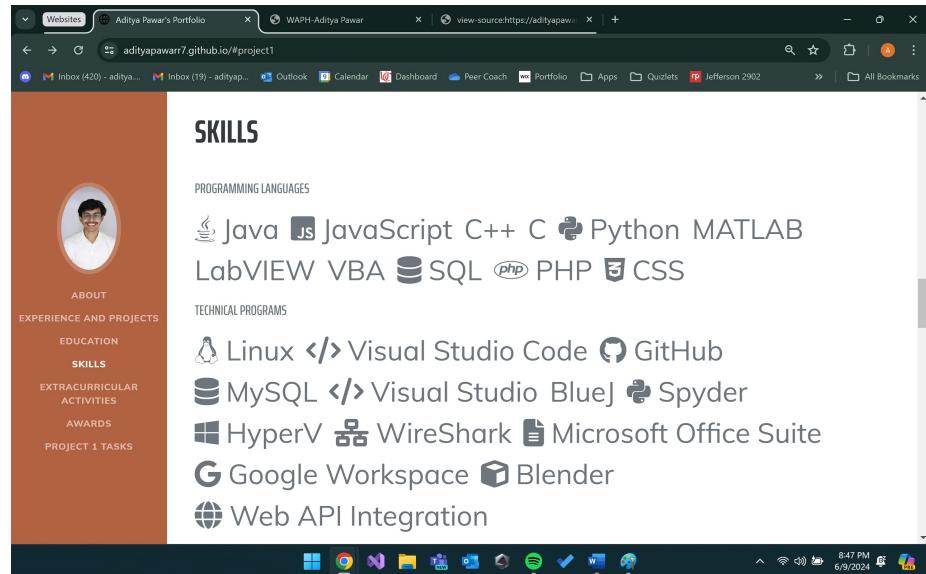


Figure 5: Skills

Extracurriculars:

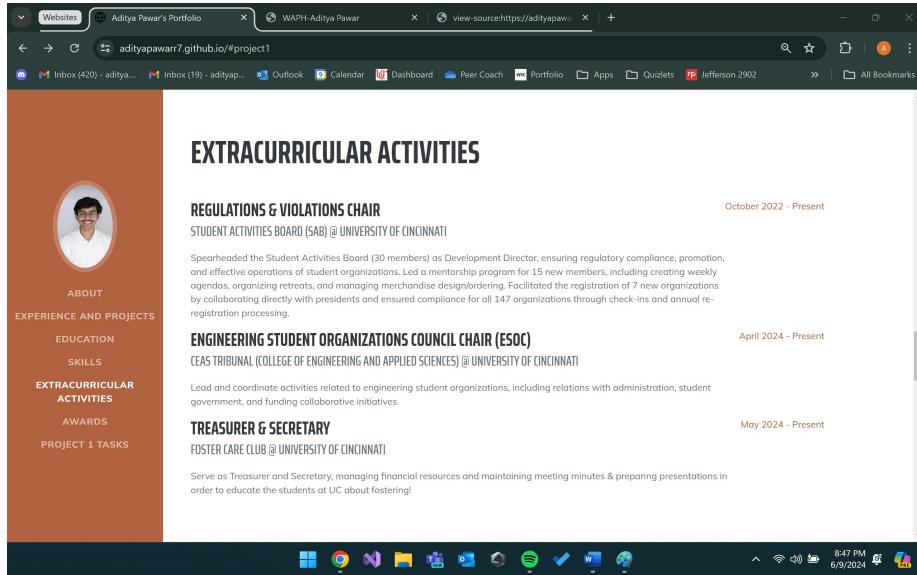


Figure 6: Extracurricular

Awards:

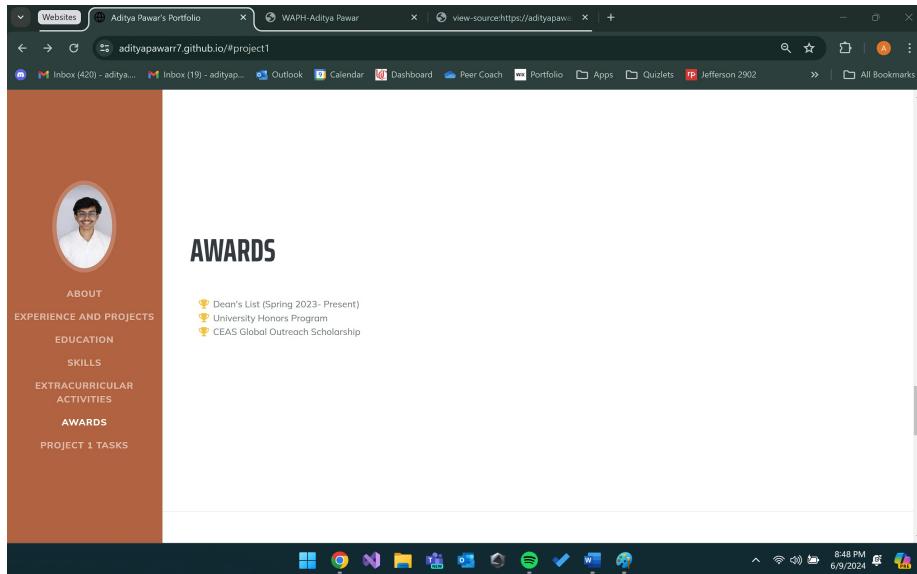
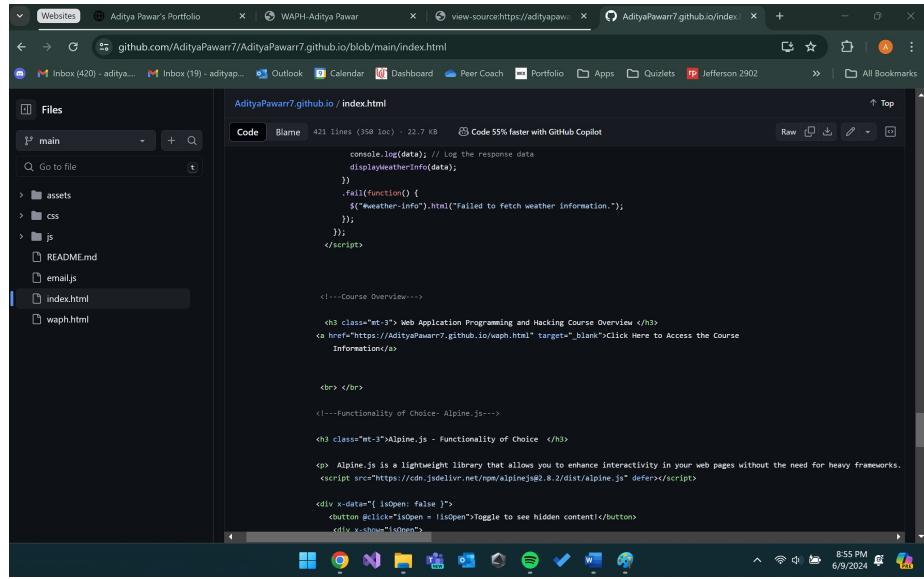


Figure 7: Awards

- + Create a link to a new HTML page to introduce this “Web Application Programming and Hacking” course and related hands-on projects

For this task, we have to link our html page `index.html` to `waph.html` which will have the course overview for our class of Web Application Programming and Hacking. The following screenshots contain the piece of code linking the two html files and what `waph.html` looks like

Link to waph.html:



The screenshot shows a GitHub repository interface for 'AdityaPawar7' with the file 'waph.html' selected. The code editor displays the following HTML and JavaScript content:

```
console.log(data); // Log the response data
displayWeatherInfo(data);
}
.fail(function() {
  $("#weather-info").html("Failed to fetch weather information.");
});

```

<!--Course Overview-->

```
<h3>Web Application Programming and Hacking Course Overview</h3>
<a href="https://AdityaPawar7.github.io/waph.html" target="_blank">Click Here to Access the Course Information</a>

<br> <br>

<!--Functionality of Choice- Alpine.js-->
```

```
<h3>Alpine.js - Functionality of Choice</h3>


Alpine.js is a lightweight library that allows you to enhance interactivity in your web pages without the need for heavy frameworks.


<script src="https://cdn.jsdelivr.net/npm/alpinejs@0.8.2/dist/alpine.js" defer></script>

```

```
<div x-data="[{ isOpen: false }]>
  <button @click="isOpen = !isOpen">Toggle to see hidden content!</button>
  <div x-show="isOpen">
```

Figure 8: Link to waph.html

waph.html 1:

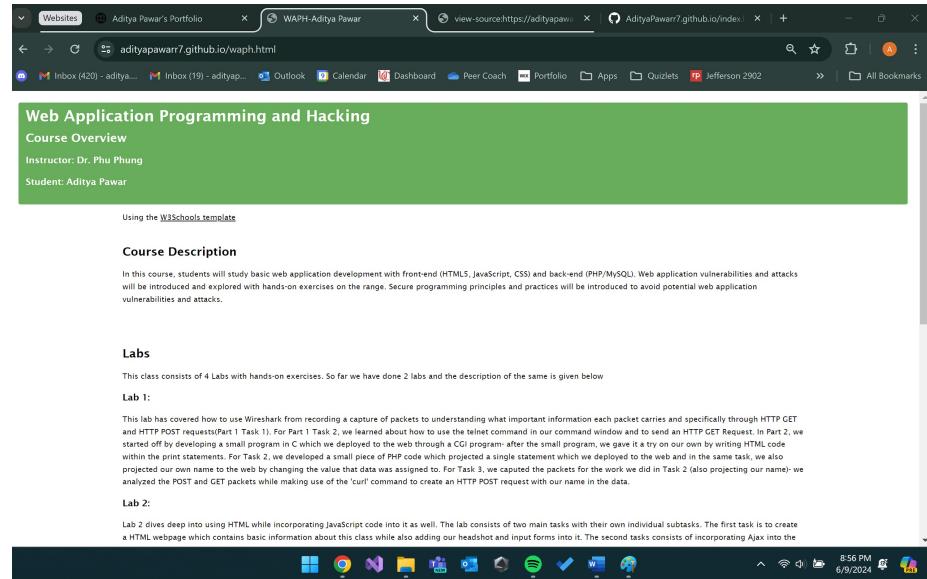


Figure 9: waph.html 1

waph.html 2:

This lab has covered how to use Wireshark from recording a capture of packets to understanding what important information each packet carries, and specifically through HTTP GET and HTTP POST requests (Part 1 Task 1). For Part 1 Task 2, we learned about how to use the telnet command in our command window and to send an HTTP GET request. In Part 2, we started off by developing a small program in C which we deployed to the web through a CGI program; after the small program, we gave it a try on our own by writing HTML code within the print statements. For Task 2, we developed a small piece of PHP code which projected a single statement which we deployed to the web and in the same task, we also projected our own name to the web by changing the value that data was assigned to. For Task 3, we captured the packets for the work we did in Task 2 (also projecting our name); we analyzed the POST and GET packets while making use of the 'curl' command to create an HTTP POST request with our name in the data.

Lab 2:
Lab 2 dives deep into using HTML while incorporating JavaScript code into it as well. The lab consists of two main tasks with their own individual subtasks. The first task is to create a HTML webpage which contains basic information about this class while also adding our headshot and input forms into it. The second tasks consists of incorporating Ajax into the input forms, using CSS to change up the style of the webpage, and Web API Integration using the JokeAPI and Agify API. This lab helps us prepare for individual project 1 and was extremely useful to me as I knew nothing about Ajax, CSS, & Web API Integration.

Hackathons
This class consists of 4 individual Hackathons for which we have approximately 72 hours in order to complete the attacks. So far we have done Hackathon 1 and the description of the same is below.

Hackathon 1:
This is the first of 4 hackathons and it deals with attacking 7 different levels of webpages from 0-6 in order to show "Level # - Hacked by Aditya Pawar". There are two tasks in this Hackathon. The first task involves the 7 different levels of webpages to attack to show the alert of being hacked by Aditya Pawar but along with this we also have to guess the source code of echo.php in order to gain better understanding about how to develop safer webpages. In the second task, we have to put our skills of hacking in order to fortify pieces of code that we've already written echo.php in the first subtask and waph-pawanar-task2d.html in the second subtask. We do this by adding in input validation and input encoding to prevent attacks. This hackathon has helped me immensely in order to gain an in-depth understanding on how exploiters target webpages by bypassing the XSS filters and will make me always account for the safety of the program that I'm developing.

Individual Projects
We have a number of Individual Projects to do as well in this class and so far we've done one and waph.html is a part of this project.

Figure 10: waph.html 2

waph.html 3:

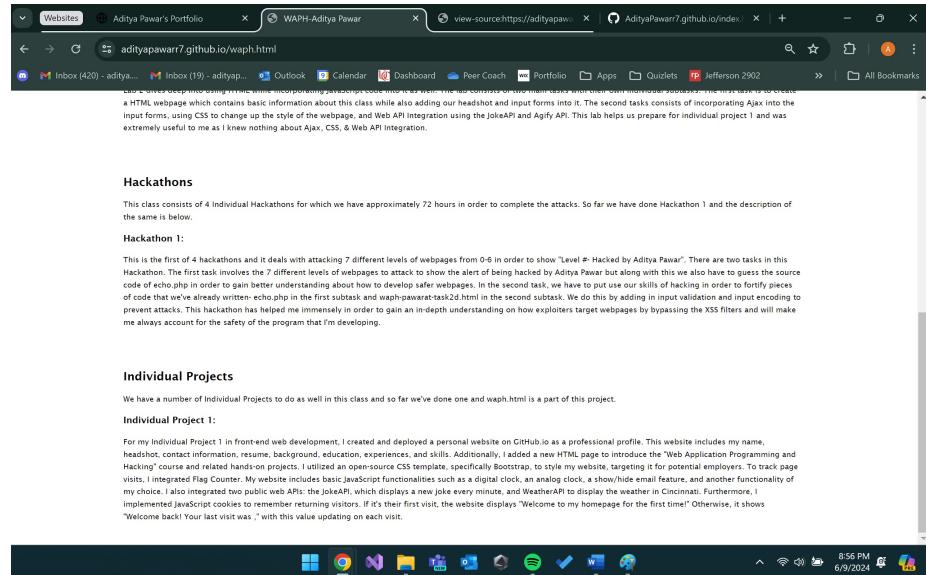


Figure 11: waph.html 3

Non-technical requirements

- Use an open-source CSS template or framework such as Bootstrap

Now, we have to style up Professional Portfolio, we make use of Bootstrap. In order to use Bootstrap, we look up free Bootstrap themes available, download one of them, unzip them and upload all of its contents to our GitHub Repository. For my case, I chose a theme from the following link Start Bootstrap Resume Theme, I downloaded the file, unzipped it, and uploaded all of its contents to our repository.

The repository contains folders for assets, css, js, and the index.html page which came along with the downloaded theme. We work on the index.html to suit our needs. The screenshots below contain a glimpse of the theme, you can also view the website itself. Along with this, I've also included a screenshot of the files in the repository to show what content came along with the zip file.

Theme Screenshot:

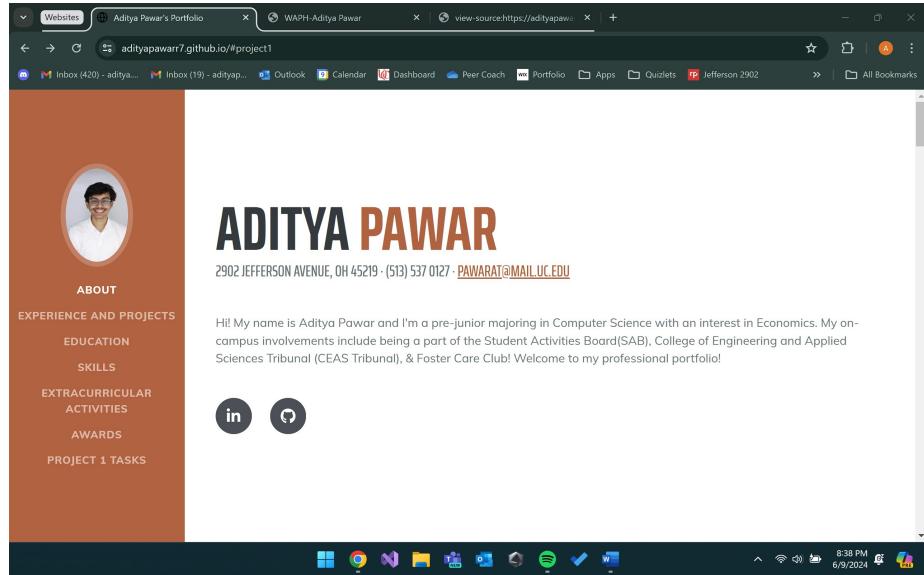


Figure 12: Theme

Files Screenshot: (Pay attention to the contents within the white outlined box)

The screenshot shows a GitHub repository interface for 'AdityaPawar7.github.io'. The left sidebar displays the file structure:

```

    └── main
        ├── assets
        │   ├── favicon.ico
        │   ├── marker
        │   └── profile.jpg
        ├── css
        │   ├── marker
        │   └── styles.css
        ├── js
        │   ├── marker
        │   ├── script.js
        │   └── README.md
        └── index.html

```

The right pane shows the code editor for 'index.html' with the following content:

```

1 <!DOCTYPE html>
2 <html lang="en">
3     <head>
4         <meta charset="utf-8" />
5         <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />
6         <meta name="author" content="" />
7         <title>Aditya Pawar's Portfolio</title>
8         <link rel="icon" type="image/x-icon" href="assets/favicon.ico" />
9         <!-- Font Awesome Icons (Free version) -->
10        <script src="https://use.fontawesome.com/releases/v6.3.0/s/all.js" crossorigin="anonymous"></script>
11        <!-- Google Fonts -->
12        <link href="https://fonts.googleapis.com/css?family=CaesarssonExtraCondensed:500,700" rel="stylesheet" type="text/css" />
13        <link href="https://fonts.googleapis.com/css?family=Multi400,400i,600,600i" rel="stylesheet" type="text/css" />
14        <!-- Core theme CSS (includes Bootstrap) -->
15        <link href="css/styles.css" rel="stylesheet" />
16    </head>
17    <body id="page-top">
18        <!-- Navigation -->
19        <nav class="navbar navbar-expand-lg navbar-dark bg-primary fixed-top" id="sideNav">
20            <a class="navbar-brand js-scroll-trigger" href="#page-top">
21                <open class="d-block d-lg-none">Aditya Pawar</open>
22                <open class="d-none d-lg-block">
23            </open>
24        </nav>
25        <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">

```

Figure 13: Files

- Include a page tracker, for example: [https://analytics\]\(https://analytics.withgoogle.com/\), <https://flagcounter.com/>.](https://analytics.withgoogle.com/)

Now we have to include a page tracker, we choose flagcounter.com for our purposes. For this, we generate link from it's website and integrate it as shown in the screenshots below. The first screenshot shows the code and the second one shows the flag counter in action.

Flag Counter Code:

```
<a href="http://s11.flagcounter.com/more/wPY">
350
351
352
353
354     $(document).ready(function() {
355       function displayTime() {
356         $('#digital-clock').html("Current time: " + new Date());
357       }
358       setInterval(displayTime, 500);
359     });
360
361
362     $(document).ready(function() {
363       var canvas = $('#analog-clock')[0];
364       var ctx = canvas.getContext('2d');
365       var radius = canvas.height / 2;
366       ctx.translate(radius, radius);
367       radius = radius * 0.98;
368       setInterval(drawClock, 1000);
369     });
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
```

Figure 16: Digital Clock Code

Digital Clock Code:

Figure 17: Digital Clock Code

Digital Clock UI:

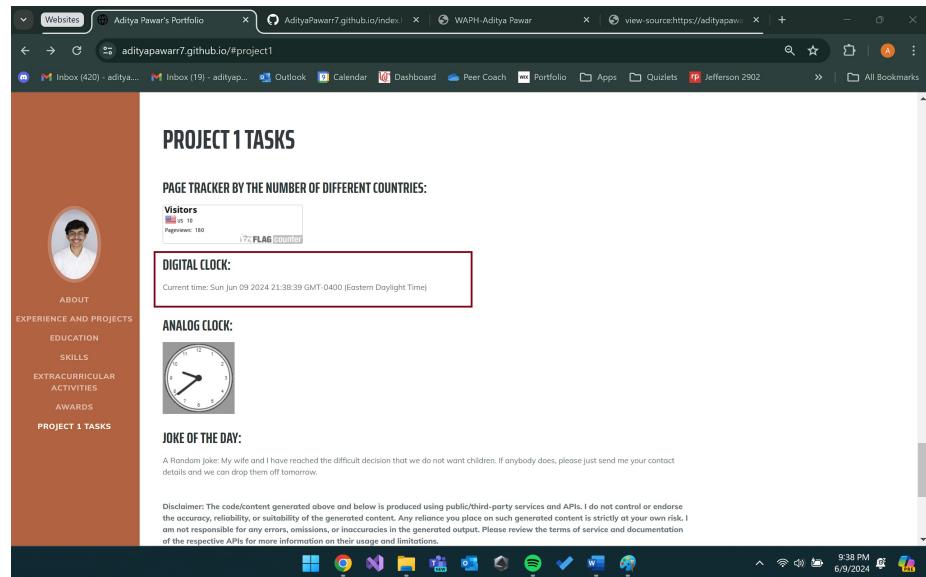
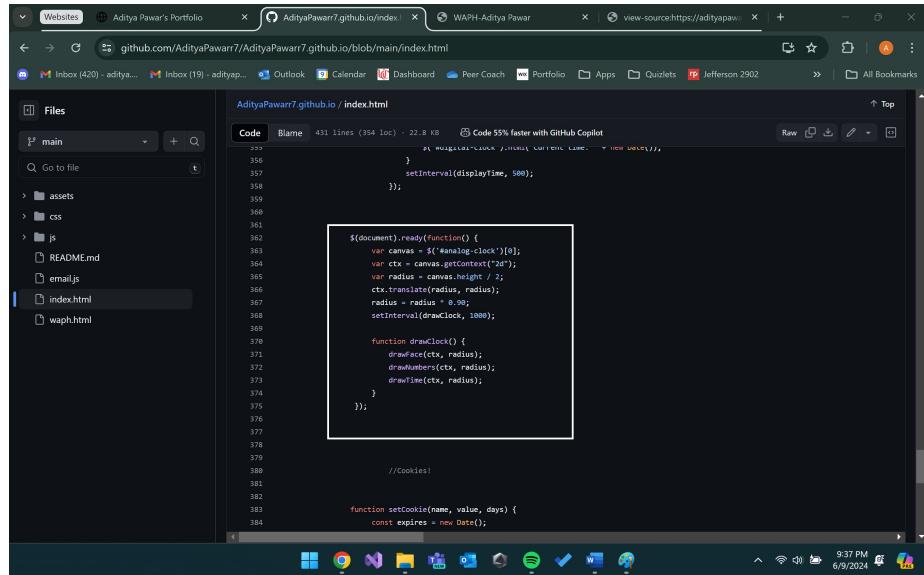


Figure 18: Digital Clock UI

In order to implement the analog clock, we refer to the script given to us by the professor and we also change the code given in Lab 2 to be in jQuery for our purposes. The screenshots below show the code and the UI Implementation.

Analog Clock Code:



The screenshot shows a GitHub repository interface with the file `index.html` selected. The code editor displays the following JavaScript code for an analog clock:

```
$(document).ready(function() {
    var canvas = $('#analog-clock')[0];
    var ctx = canvas.getContext("2d");
    var radius = canvas.height / 2;
    ctx.translate(radius, radius);
    radius = radius * 0.98;
    setInterval(drawClock, 1000);

    function drawClock() {
        drawFace(ctx, radius);
        drawNumbers(ctx, radius);
        drawTime(ctx, radius);
    }
});

function drawFace(ctx, radius) {
    var grad = ctx.createRadialGradient(radius, radius, 0, radius, radius, 10);
    grad.addColorStop(0, "#333");
    grad.addColorStop(0.3, "#333");
    grad.addColorStop(1, "#fff");
    ctx.fillStyle = grad;
    ctx.beginPath();
    ctx.arc(0, 0, radius, 0, 2 * Math.PI);
    ctx.fill();
}

function drawNumbers(ctx, radius) {
    var num = ["12", "1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11"];
    for (var i = 0; i < 12; i++) {
        var angle = Math.PI / 6 * i - Math.PI / 2;
        var x = radius * Math.cos(angle);
        var y = radius * Math.sin(angle);
        ctx.fillText(num[i], x, y);
    }
}

function drawTime(ctx, radius) {
    var now = new Date();
    var sec = now.getSeconds();
    var min = now.getMinutes();
    var hr = now.getHours();

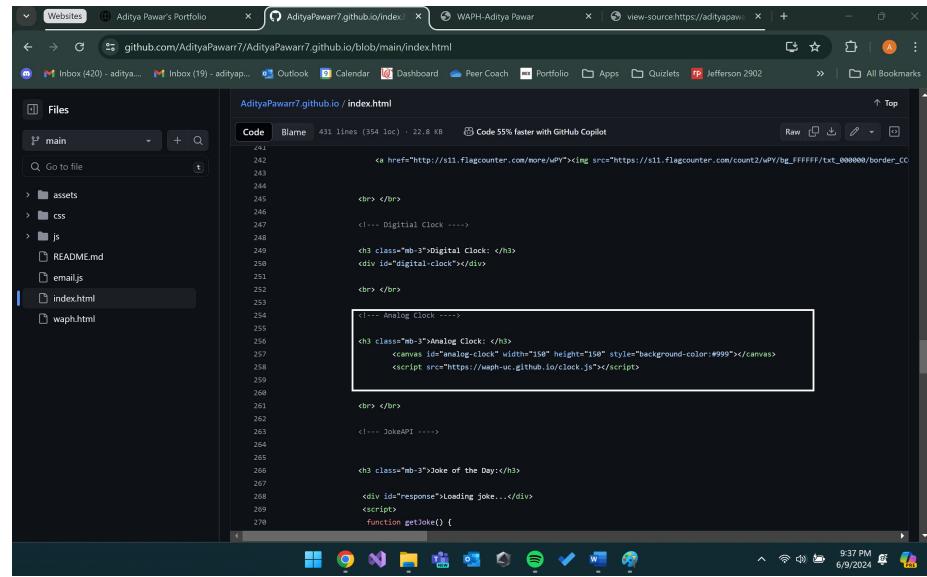
    var secAngle = sec / 60 * 2 * Math.PI;
    var minAngle = min / 60 * 2 * Math.PI;
    var hrAngle = hr / 12 * 2 * Math.PI;

    var secX = radius * Math.cos(secAngle);
    var secY = radius * Math.sin(secAngle);
    var minX = radius * Math.cos(minAngle);
    var minY = radius * Math.sin(minAngle);
    var hrX = radius * Math.cos(hrAngle);
    var hrY = radius * Math.sin(hrAngle);

    ctx.beginPath();
    ctx.moveTo(0, 0);
    ctx.lineTo(secX, secY);
    ctx.lineTo(minX, minY);
    ctx.lineTo(hrX, hrY);
    ctx.stroke();
}
```

Figure 19: Analog Clock Code

Analog Clock Code:



The screenshot shows a code editor window displaying the `index.html` file from the GitHub repository `AdityaPawar7.github.io`. The code is written in HTML and includes some CSS and JavaScript. A red rectangular box highlights the following section of the code, which defines an analog clock:

```
242     <a href="http://s11.flagcounter.com/more/wMw">
243
244
245     <br> <br>
246
247     <!-- Digital Clock ---->
248
249     <h3 class="mb-3">Digital Clock: </h3>
250     <div id="digital-clock"></div>
251
252     <br> <br>
253
254
255
256     <!-- Analog Clock ---->
257
258     <h3 class="mb-3">Analog Clock: </h3>
259     <canvas id="analog-clock" width="150" height="150" style="background-color:#000"></canvas>
260     <script src="https://waph-uc.github.io/clock.js"></script>
261
262     <br> <br>
263
264     <!-- JokeAPI ---->
265
266
267     <h3 class="mb-3">Joke of the Day:</h3>
268
269     <div id="response">loading joke...</div>
270     <script>
271         function getJoke() {
```

Figure 20: Analog Clock Code

Analog Clock UI:

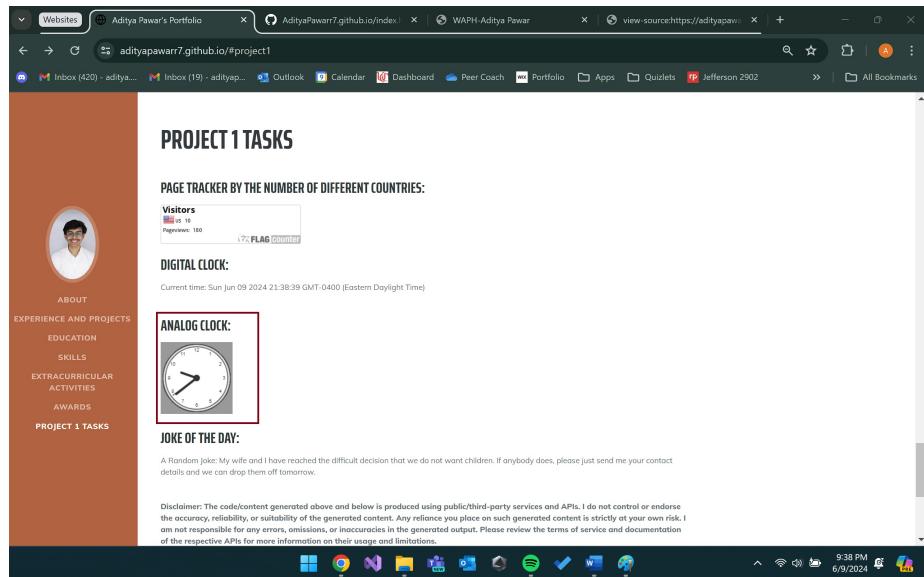
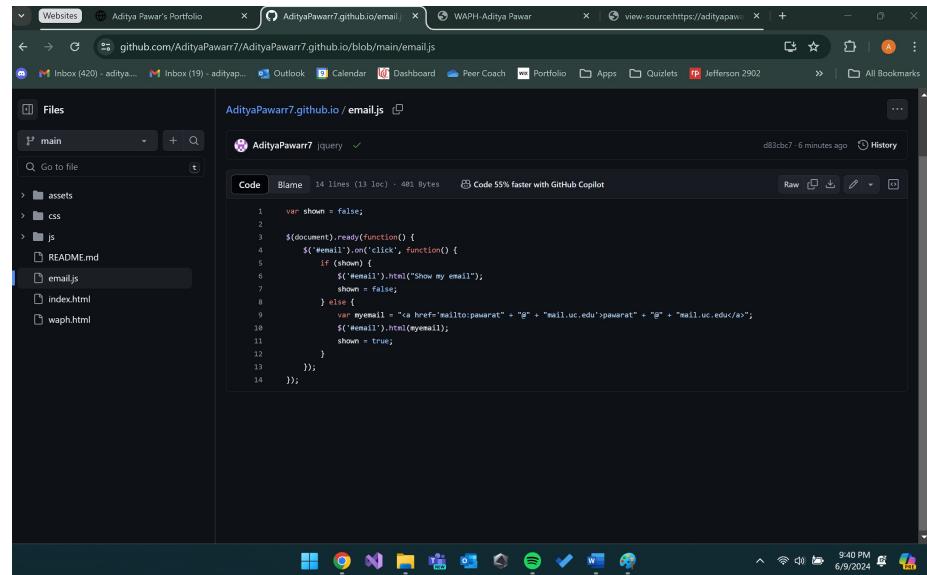


Figure 21: Analog Clock UI

Now, to implement email show/hide function, we refer to our `email.js` file- the code for which has also been change to jQueery form- we sill refer it from the `email.js` file though. The screenshots below show the code and the UI Implementation

email.js code:



The screenshot shows a GitHub repository interface. The left sidebar lists files: assets, css, js, README.md, email.js (which is selected), index.html, and waph.html. The main area displays the contents of the email.js file. The code is as follows:

```
1  var shown = false;
2
3  $(document).ready(function() {
4      $('#email').on('click', function() {
5          if (shown) {
6              $('#email').html("Show my email");
7              shown = false;
8          } else {
9              var myemail = "<a href='mailto:pawar@" + "g" + "mail.uc.edu'>pawar@" + "g" + "mail.uc.edu</a>";
10             $('#email').html(mymail);
11             shown = true;
12         }
13     });
14});
```

Figure 22: email.js code

email.js code implementation in index.html:

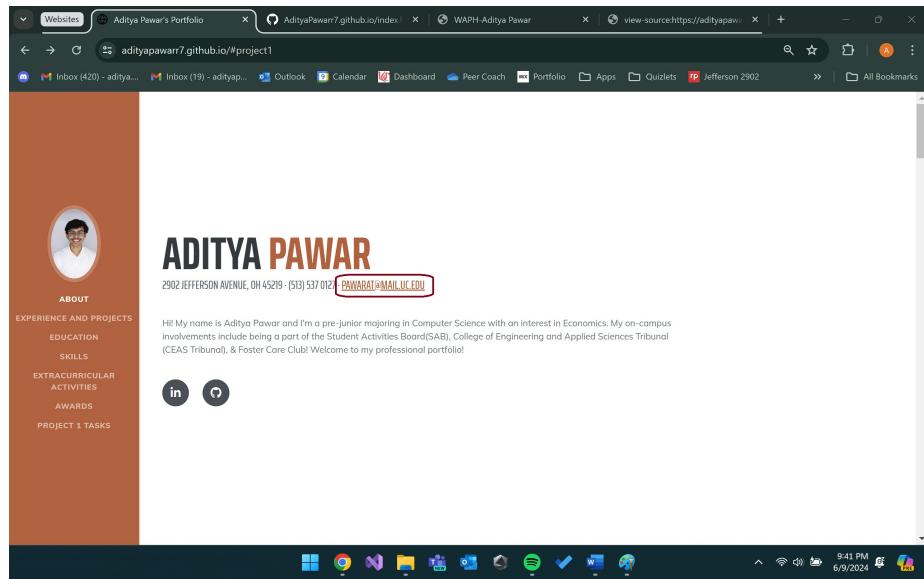


Figure 23: email.js code in index.html

Email showHide UI Implementation (Before being Clicked):

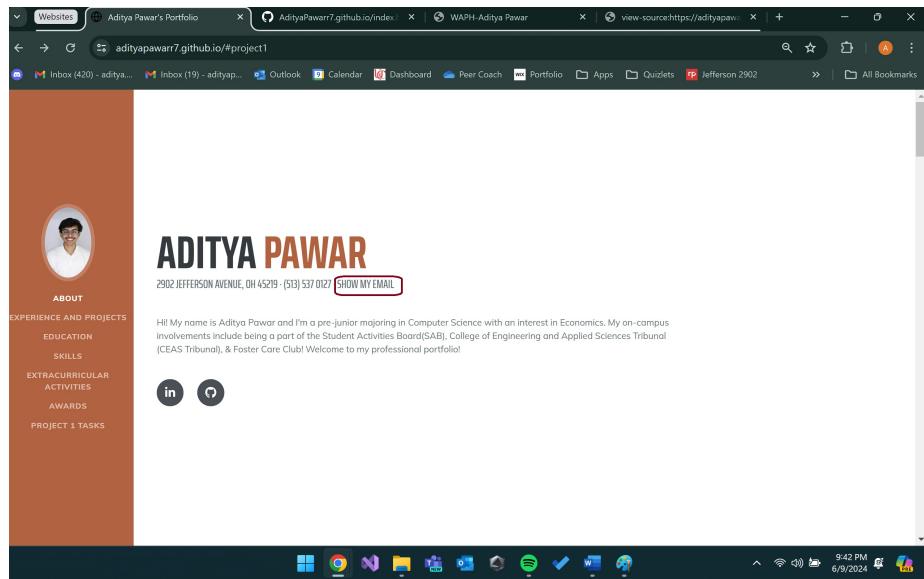


Figure 24: Before Being Clicked

Email showHide UI Implementation (After being Clicked):

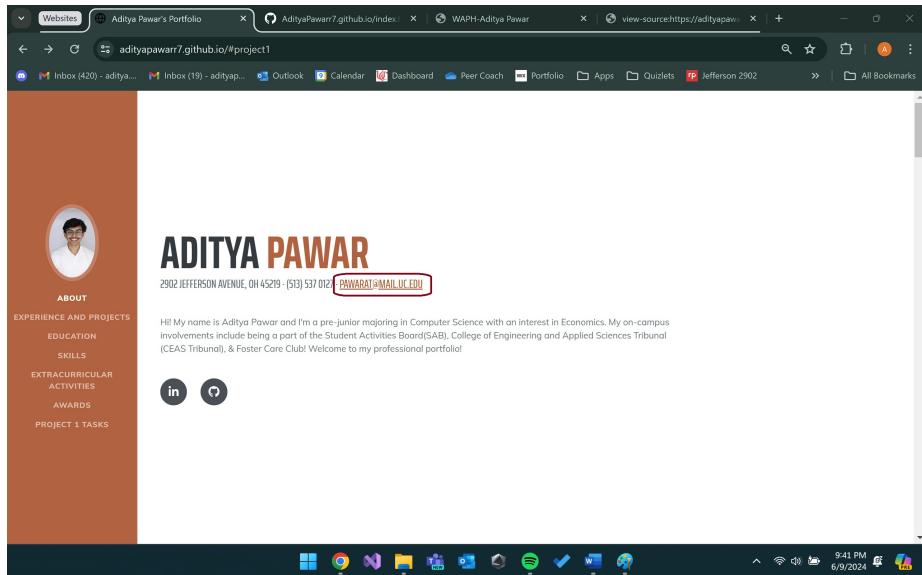


Figure 25: After Being Clicked

Finally, in order to implement a functionality of our choice- we implement Alpine.js- which is a lightweight library that allows you to enhance interactivity in your web pages without the need for heavy frameworks. We use it in our scenario as a toggle button to show hidden content to show its implementation. Screenshots below show its code in `index.html` and its UI Implementation.

Alpine.js code:

The screenshot shows a GitHub repository page for 'AdityaPawar77/AdityaPawar7.github.io'. The 'index.html' file is open in a code editor. The code uses Alpine.js to toggle hidden content based on a 'isopen' boolean. It also includes a digital clock component using jQuery.

```
<div class="mt-1"><Web Application Programming and Hacking Course Overview.</div>
<a href="https://adityapawar77.github.io/waph.html" target="_blank">Click Here to Access the course information</a>

<br>
```

```
<!--functionality of Choice- Alpine.js-->
<div class="mt-1" x-data="alpinejs - functionality of Choice </h3>
<p> Alpine.js is a lightweight library that allows you to enhance interactivity in your web pages without the need for heavy frameworks. </p>
<script src="https://cdn.jsdelivr.net/npm/alpinejs@3.2.1/dist/alpine.js" defer></script>
<div x-data="{ isopen: false }"
      @click="isopen = !isopen">toggle to see hidden content</button>
<div x-show="isopen">
    This content will be shown or hidden based on the value of isopen. Great job finding out the hidden content, this would not have been possible without using Alpine.js!
</div>
</div>
```

```
</section>

<script type="text/javascript">
//jquery digital clock
$(document).ready(function(){
    function displaytime(){
        $('#digital-clock').html("Current time: " + new Date());
    }
    setinterval(displaytime, 1000);
})
```

Figure 26: Alpine.js code

Alpine.js UI Before being Toggled:

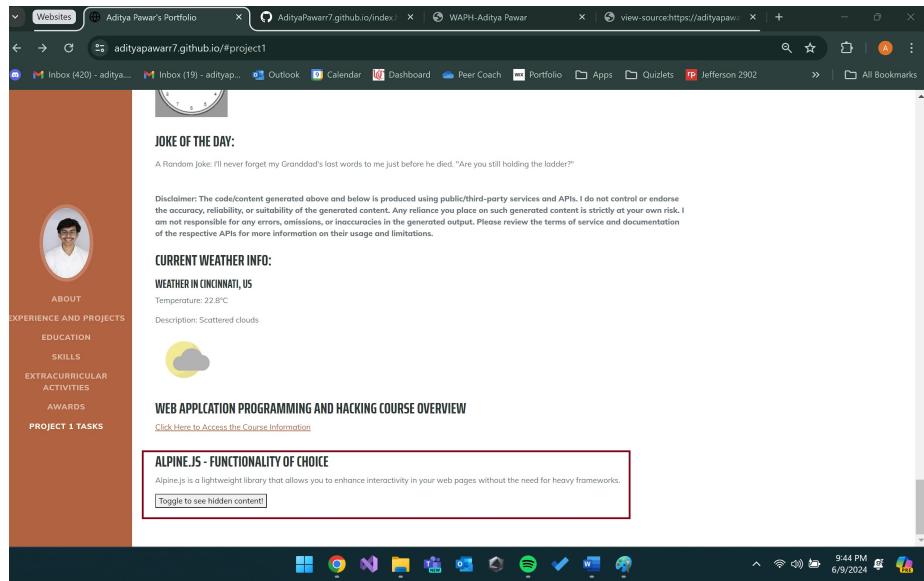


Figure 27: Alpine.js UI Before Toggle

Alpine.js UI After being Toggled:

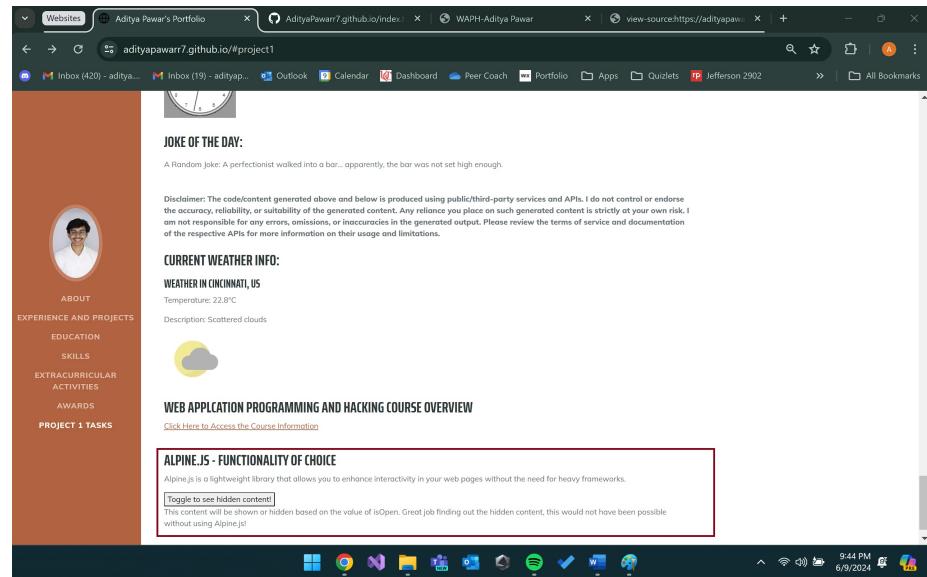


Figure 28: Alpine.js UI Before Toggle

- Two public Web APIs integration: *Ensure you include a disclaimer stating that the public/third-party services generate the generated contents below and that you are not responsible.*

Screenshot of the Disclaimer:

Disclaimer Screenshot:

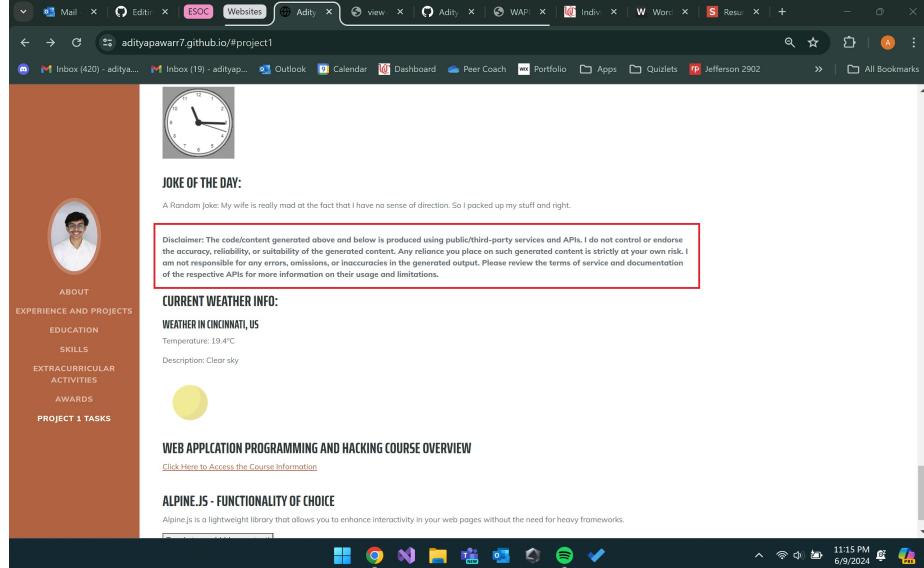
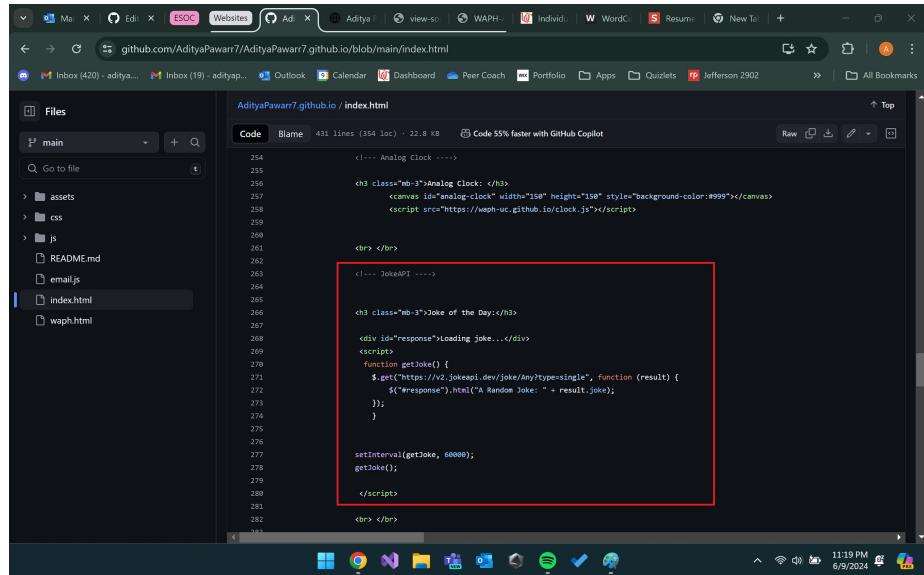


Figure 29: Disclaimer Screenshot

1. Integrate the jokeAPI (<https://v2.jokeapi.dev/joke/Any>, similar to Lab 2.2.d.i) with Any category of joke to display a new joke in your page every 1 minute.

Now, we have to integrate jokeAPI similar to Lab 2 but with the ‘Any’ category to display a joke and refresh it every 1 minute. We write the function `getJoke()` and set the interval to 6000 which is in milliseconds which is equal to 1 minute. The screenshots below show the code of jokeAPI and the implementation in UI. I can confirm that the 1 minute refresh works.

jokeAPI Code Screenshot:



The screenshot shows a GitHub code editor interface for the repository `AdityaPawar7/AdityaPawar7.github.io`. The current file is `index.html`. The code editor displays the following content:

```
254     <!-- Analog Clock -->
255
256     <h3 class="mb-3">Analog Clock:</h3>
257     <canvas id="analog-clock" width="150" height="150" style="background-color:#999"></canvas>
258     <script src="https://waph-uc.github.io/clock.js"></script>
259
260
261     <br> <br>
262
263     <!-- JokedPT -->
264
265
266     <h3 class="mb-3">Joke of the Day:</h3>
267
268     <div id="response">Loading joke...</div>
269     <script>
270         function getJoke() {
271             $.get("https://v2.jokeapi.dev/joke/any?type:single", function (result) {
272                 $("#response").html("A Random Joke: " + result.joke);
273             });
274         }
275
276
277         setInterval(getJoke, 60000);
278         getJoke();
279
280     </script>
281
282     <br> <br>
```

Figure 30: jokeAPI Code Screenshot

jokeAPI UI Screenshot:

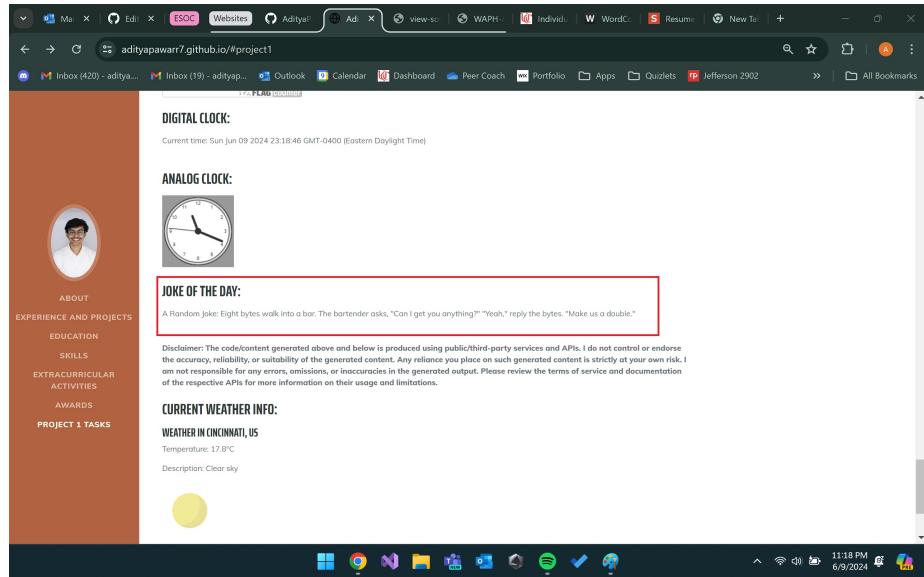
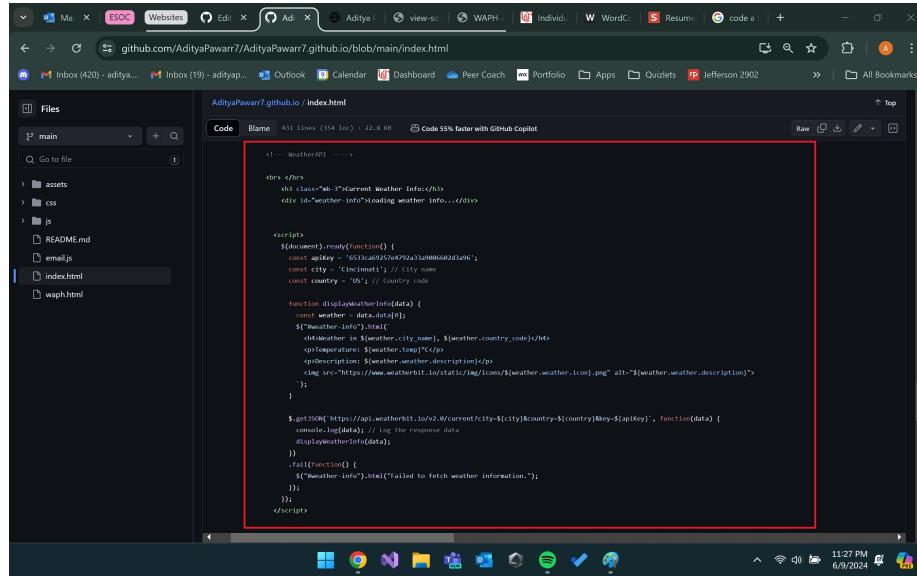


Figure 31: jokeAPI UI Screenshot

3. Integrate a public API with graphics and display that graphic/image in your page. Examples: <https://xkcd.com/info.0.json>, <https://www.weatherbit.io>.

Now, we integrate the WeatherAPI into our Personal Portfolio Website. We do this by firstly signing up on the API's website to get a master key and then code the function `displayWeatherInfo()` in jQuery. The screenshots below show the code of the function and it's implementation in the website to show weather for Cincinnati.

WeatherAPI Code Screenshot:



The screenshot shows a GitHub code editor interface. The left sidebar displays a file tree for a repository named 'AdityaPawar7.github.io'. The 'index.html' file is selected and open in the main editor area. The code is written in HTML and JavaScript. A red box highlights the entire code block in the editor.

```
<!-- WeatherAPI -->

<div>
  <div class="row">
    <div id="current-weather-info"></div>
    <div id="weather-info">Loading weather info...</div>
  </div>
</div>

<script>
  $(document).ready(function() {
    const apiKey = '6533c46922d479213a08660b21a0b';
    const city = 'Cincinnati'; // City name
    const country = 'US'; // Country code

    function displayWeatherInfo(data) {
      const weather = data.data[0];
      $('#weather-info').html(`

        ${weather.name}, ${weather.country_code}</h2>
        ${weather.temperature}</p>
        ${weather.description}</p>
        ![${weather.icon}](https://www.weatherbit.io/static/img/icons/${weather.icon}.png)
      `);
    }

    $.getJSON(`https://api.weatherbit.io/v2.0/current?city=${city}&country=${country}&key=${apiKey}`, function(data) {
      console.log(data); // Log the response data
      displayWeatherInfo(data);
    })
    .fail(function() {
      $('#weather-info').html("Failed to fetch weather information.");
    });
  });
</script>
```

Figure 32: WeatherAPI Code Screenshot

WeatherAPI UI Screenshot:

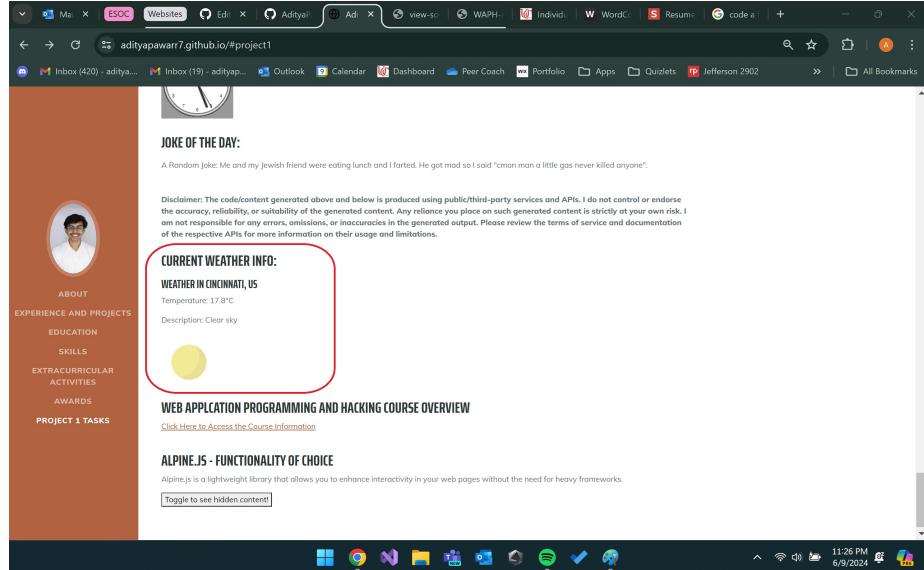


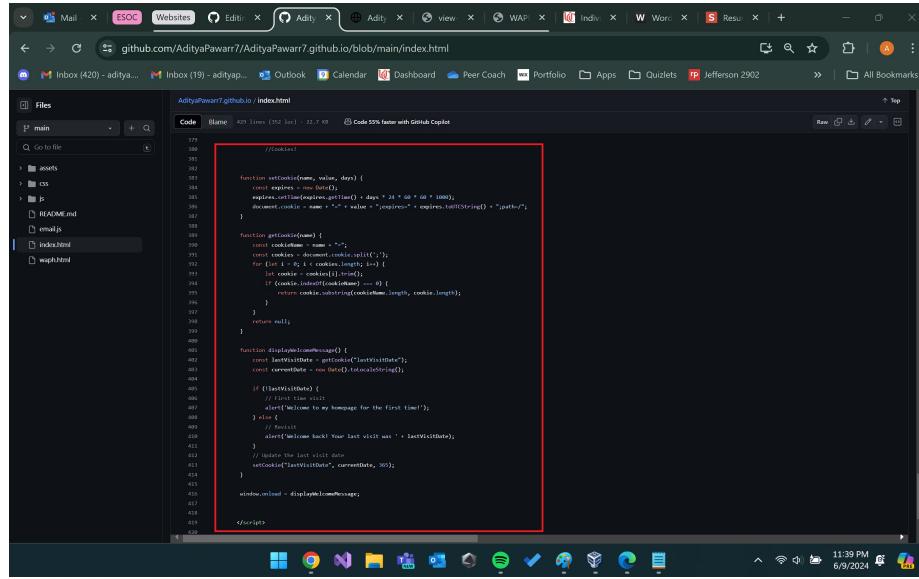
Figure 33: WeatherAPI UI Screenshot

- Use JavaScript cookies to remember the client (10 pts): If first-time visit, display the message “Welcome to my homepage for the first time!”; otherwise, display the message “Welcome back! Your last visit was ” (ensure that you update this value each time the same user visits -2pts if missing).

For this, we use the code given to us in the lecture slides and add more functions in order for it to work with our webpage. This code manages user visit information using cookies. When the webpage loads, the `displayWelcomeMessage` function is called, which retrieves the “`lastVisitDate`” cookie using the `getCookie` function. If the cookie does not exist, indicating a first-time visit, it shows an alert welcoming the user for the first time. If the cookie exists, it shows an alert with the date of the user’s last visit. It then updates the “`lastVisitDate`” cookie with the current date and time, set to expire in 365 days, using the `setCookie` function.

The following screenshots show the code of how the cookies are used, and its implementation on a new browser (Microsoft Edge) on which I have not opened my portfolio site yet and then showing what happens when I reload it.

Cookie Code Screenshot:



The screenshot shows a browser window displaying the source code of `index.html` from a GitHub repository. The code is a JavaScript file containing functions for setting and getting cookies, and a function for displaying a welcome message based on the user's previous visit. A red box highlights the code for setting a cookie.

```
//CookieSet
function setCookie(name, value, days) {
    var date = new Date();
    date.setTime(date.getTime() + (days * 24 * 60 * 60 * 1000));
    document.cookie = name + "=" + value + ";expires=" + date.toDateString() + ";path=/";
}

function getCookieValue() {
    const cookieName = "name";
    const cookies = document.cookie.split(';');
    for (let i = 0; i < cookies.length; i++) {
        let cookie = cookies[i].trim();
        if (cookie.indexOf(cookieName) === 0) {
            return cookie.substring(cookieName.length, cookie.length);
        }
    }
    return null;
}

function displayWelcomeMessage() {
    const lastVisitDate = getCookie("lastVisitDate");
    const currentDate = new Date().toDateString();
    if (!lastVisitDate) {
        // first time visit
        alert("Welcome to my homepage for the first time!");
    } else {
        // repeat visit
        alert(`Welcome back! Your last visit was ${lastVisitDate}`);
    }
    // Update the last visit date
    setCookie("lastVisitDate", currentDate, 365);
}
window.onload = displayWelcomeMessage;
```

Figure 34: Cookie Code Screenshot

First Time visiting the Webpage Screenshot:

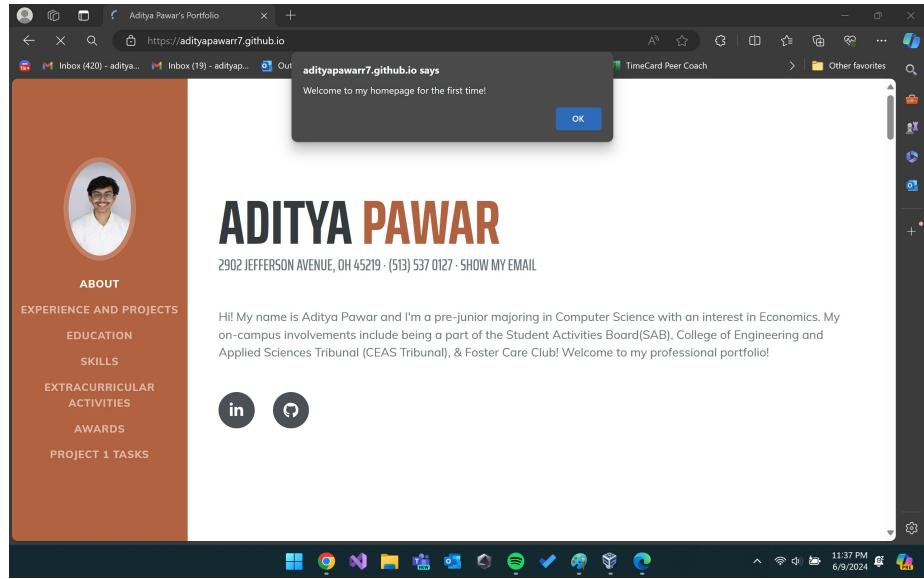


Figure 35: First Time visiting the Webpage

Second Time visiting the Webpage Screenshot (Refreshing):

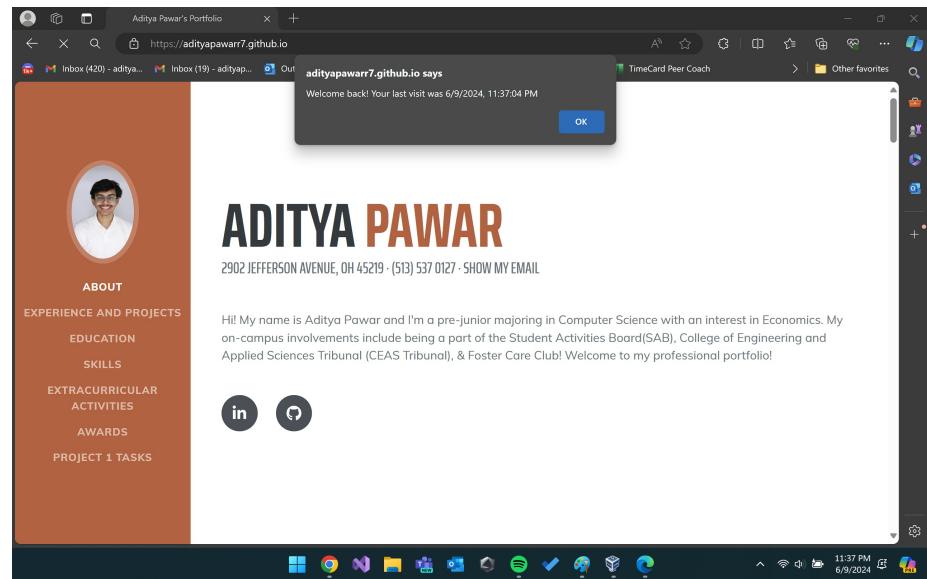


Figure 36: Second Time visiting the Webpage (Refreshing)