Web Dev Project Report

Rajat Das

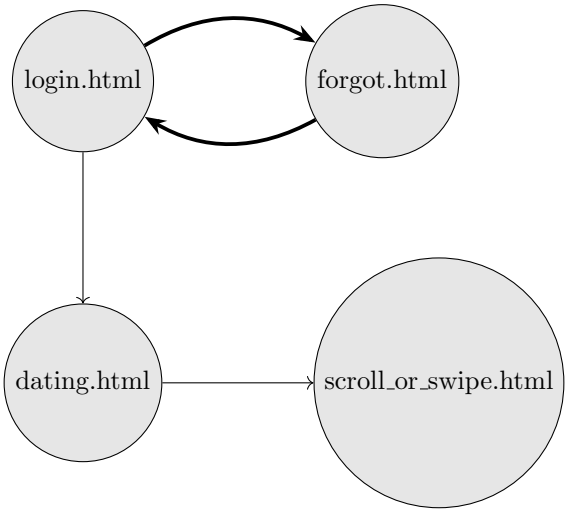
April 2024

Contents

1 Introduction

This project involves developing a dating website using HTML, CSS, and JavaScript. The website will allow users to create a profile by entering personal details such as name, age, interests, and hobbies. The user's profile will then be matched with other user profiles stored in a JSON file based on similarities in interests and hobbies. The website will have features like user authentication, forgot password functionality, swiping/scrolling through profiles, and displaying the best match. Additional customizations can be implemented to enhance the website's functionality and user experience.

2 Overview



- **Login.html :** To access the interface the user needs to login with an username and password. If the user forgets their password they can try to retrieve their password by clicking the **Forgot Password** button.
- **Forgot.html :** After clicking the Forgot Password button, the user is redirected to Forgot.html. The user can retrieve their password if they correctly answer the security question. The page also has a **Return to Login** button.
- **Dating.html :** The Dating.html page has a form that takes input from the user. After clicking **Submit** the user is shown their match on a new tab. The page also has a **See For Yourself** button that redirects to a page that lists all the profiles as cards.
- **Scroll_Or_Swipe.html :** Here, profiles of all the students are listed in the form of **cards**.

3 Groundwork

3.1 Fetch

In this project we are required to retrieve data from JSON files. I have used fetch to accomplish this.

```
fetch('url') // api for the get request
    .then(response => response.json())
    .then(data => console.log(data));
```

1. **fetch('yourfile.json') :** This initiates a GET request to retrieve the file $yourfile.json$. The fetch function takes one argument which is the URL or file path.
2. **.then(response => response.json()) :** The .then method is used to handle the file returned by fetch. In this case, it calls the .json method on it. This essentially converts the raw file into useable form.
3. **.then(data => { ... }) :** Now that we have the useable data, we will do something with the data in the curly braces.

I have used this code block extensively in this project.

Reference to fetch : https://www.geeksforgeeks.org/javascript-fetch-method/

3.2 querySelector

In JavaScript, querySelector is a method that allows to select and retrieve the first element that matches a specified CSS selector in a document. It is part of the Document Object Model (DOM) API and can be used to access and manipulate elements on a web page.

```
element = document.querySelector(selector);
```

Here, element is a reference to the first element that matches the selector, which can be any valid CSS selector (e.g., $\#id, .class, div, a[href = "..."]$). If no element matches the selector, it returns null.

Returning the First Match: If multiple elements match the selector, querySelector returns only the first match.

Use **querySelectorAll()** to get all matches.

I have used this to handle the **checkboxes** in **Dating.html**.

Reference to queryselector : https://www.geeksforgeeks.org/html-dom-queryselector-method/

3.3 Handling Javascript Arrays

Reference : https://www.w3schools.com/jsref/jsref_obj_array.asp

1. **Filter :** The filter() method creates a new array filled with elements that pass a test provided by a function.

```
const ages = [32, 33, 16, 40];
const result = ages.filter(checkAdult);

function checkAdult(age) {
  return age >= 18;
}
```

2. **Map :** The map() creates a new array from calling a function for every array element.

```
const numbers = [65, 44, 12, 4];
const newArr = numbers.map(myFunction)

function myFunction(num) {
  return num * 10;
}
```

3. **forEach :** The arr.forEach() method calls the provided function once for each element of the array. The provided function may perform any kind of operation on the elements of the given array.

```
let sum = 0;
const numbers = [65, 44, 12, 4];
numbers.forEach(myFunction);

function myFunction(item) {
  sum += item;
}
```

4. **Sort :** The sort() method in JavaScript arranges the elements of an array in place and returns the sorted array. By default, it sorts the elements alphabetically as strings. If numerical sorting is required, a custom comparison function can be provided as an argument.

Sort numbers in ascending order:

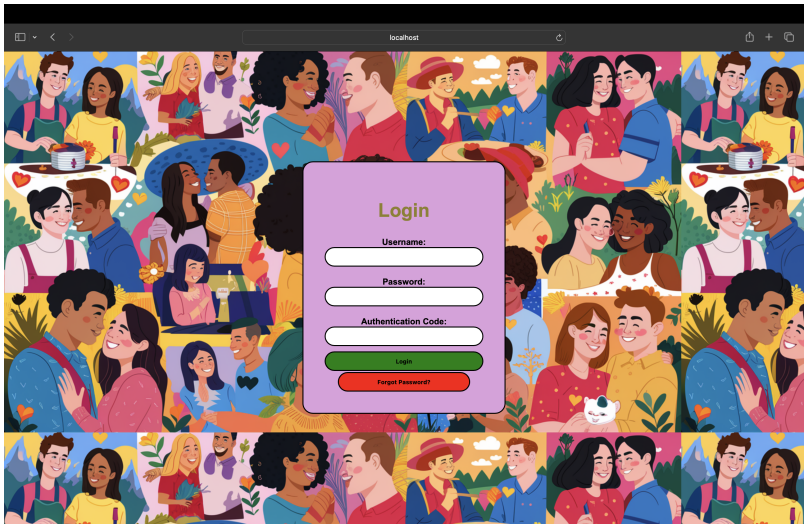
```
const points = [40, 100, 1, 5, 25, 10];
points.sort(function(a, b){return a-b});
```

5. **Includes :** The includes() method in JavaScript is used to determine whether a string contains another string within it. It returns true if the specified string is found, and false otherwise.

```
let str = "Welcome to GeeksforGeeks.";
let check = str.includes("Geeks");
console.log(check);
```

4 Basic

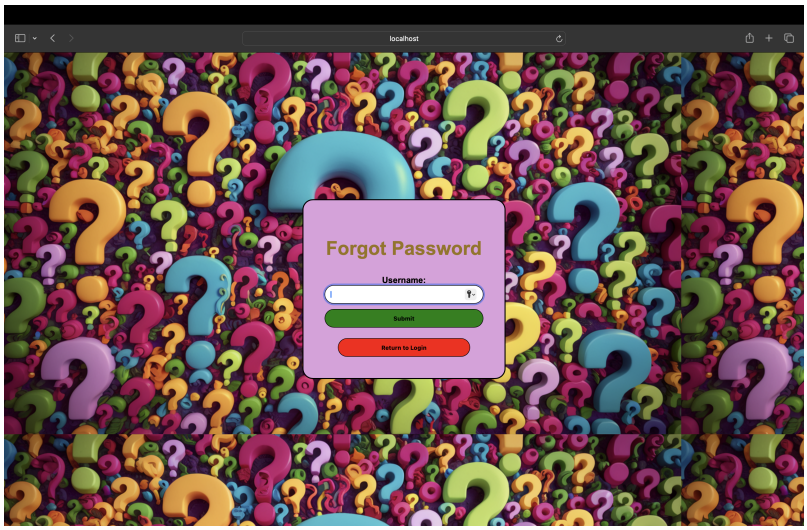
4.1 login.html



The login page has a form that validates username and password, a **Submit** button and **Forgot Password** button.

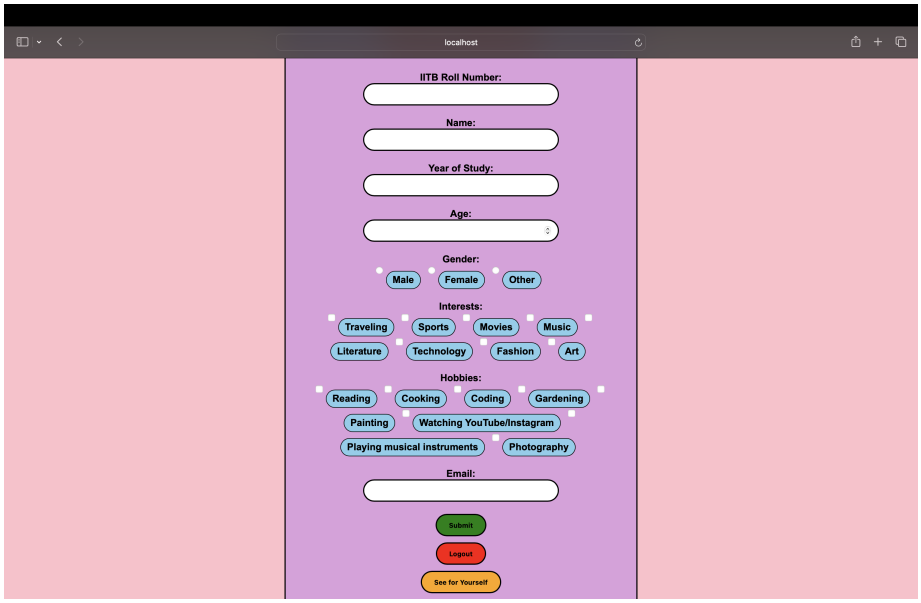
The CSS is written in **styleLogin.css**.

4.2 forgot.html



- Firstly, the page asks for an **username**.
- When the **Submit** is pressed, the code checks if the username exists and if it does, the security question is displayed.
  The security question is the **secret_question** field in **login.json**.
- When the user answers the question, the answer is compared to the **secret_answer** field in **login.json** and if it matches, the password is displayed.
- Then the user can go back to the login page with the **Return to Login** button.

4.3 dating.html



This page has has a form that the user can fill according to their liking and try to find a match among entries in **students.json**. The form features :

- Form for IITB Roll Number, Name, Year of Study, Age, Email.
- **Radioboxes** for **Gender**.
- **Checklists** for **Hobbies & Interests**.

After filling the form, the user can get their match by clicking the **Submit** button.

The page also features :

- A **button** to a page that displays all profiles as cards.
- **Logout** button that redirects to **login.html**.

4.4 script.js

This is the main script that handles the form inputs from the user and gives out a match. The working of the script is as follows :

- **Loading the students' profile data from students.json**
  - Firstly I have declared an empty array named **students**.
  - Then , I have used the **Fetch API** to load data from **students.json** and store it in **students** array.
  - After the data is loaded, an event listener is attached to the form with the ID **dating-form** to handle form submissions
- **Form Submission Event Handler**
  - I have defined the function **handleFormSubmit** to handle the form submission event.
  - The function starts by preventing the default form submission behavior using **event.preventDefault()**.
  - It then retrieves the user input from various form fields, such as roll number, name, year of study, age, gender, interests, hobbies, and email.
  - **Interests** and **Hobbies** are stored in arrays. Ref : https://stackoverflow.com/questions/590018/getting-all-selected-checkboxes-in-an-array
  - All the user input is compiled into an **userProfile** object.
- **Finding the Best Possible Match**
  - The **findingBestMatch** function is called with the **students array** (this is about the students in students.json) and the **userProfile** object (this is about the user).
  - Firstly the function filters out the profiles from **students** array whose **Gender** field is not equal to that of the user and stores them in **possibleMatchesOfOppositeGender** object.

Some Code Blocks :

```
// Calculate match benchmark based on interests and hobbies
// In the rudimentary algorithm consisting of calculation of a score(sum), I have just
// Added a weight of 1.5x to hobbies, since I feel that is more important
// Also calculate age difference, which will be used to sort later
const matchBenchmark = possibleMatchesOfOppositeGender.map(student => {
  const ageDifference = Math.abs(student.Age - userProfile.Age);
  const intersectionScore = getIntersectionOfArrays(student.Interests, userProfile.Interests) + 1.5 * getIntersectionOfArrays(student.Hobbies,
  return { student, ageDifference, intersectionScore };
});
```

This code snippet is responsible for creating an array called **matchBenchmark** that contains objects representing potential matches with additional information used for ranking and sorting the matches.

1. **possibleMatchesOfOppositeGender.map(student => ...)**: This uses the map method on the possibleMatchesOfOppositeGender array, contains a list of students of the opposite

gender from the user's gender. The map method iterates over each student in the array and executes the provided function.

2. **const ageDifference = Math.abs(student.Age - userProfile.Age) :** For each student, it calculates the absolute difference between the student's age and the user's age (userProfile.Age). This value is stored in the ageDifference variable.
3. **const intersectionScore = getIntersectionOfArrays(student.Interests, userProfile.Interests) + 1.5 * getIntersectionOfArrays(student.Hobbies, userProfile.Hobbies) :** This line calculates an "intersection score" based on the common interests and hobbies between the student and the userProfile.
    - **getIntersectionOfArrays(student.Interests, userProfile.Interests)** calculates the number of common interests between the student and the user.
    - **getIntersectionOfArrays(student.Hobbies, userProfile.Hobbies)** calculates the number of common hobbies between the student and the
    - I have assigned a weight of **1.5** to hobbies, as I feel that is more important.
    - The intersections of interests and weighted hobbies are added together and stored in the **intersectionScore** variable.
    - **return student, ageDifference, intersectionScore ;:** For each student, the function returns an object containing the original student object, the calculated ageDifference, and the calculated intersectionScore.

```
// Sorting the matches based on intersection score, age gap, and hobbies intersection
matchBenchmark.sort((a, b) => {
  if (a.intersectionScore === b.intersectionScore) {
    if (a.ageDifference === b.ageDifference) {
      return getIntersectionOfArrays(b.student.Hobbies, userProfile.Hobbies) - getIntersectionOfArrays(a.student.Hobbies, userProfile.Hobbies);
    }
    return a.ageDifference - b.ageDifference;
  }
  return b.intersectionScore - a.intersectionScore;
```

The sorting is done in the following order :

1. Sort by intersectionScore in descending order
2. If intersectionScore is the same, sort by ageDifference in ascending order
3. If intersectionScore and ageDifference are the same, sort by hobbies intersection

- **matchBenchmark.sort((a, b) => ...):** This is the sort method of the array, which accepts a comparison function as an argument. The comparison function takes two elements a and b from the array and returns a value that determines the sorting order.
- **if (a.intersectionScore === b.intersectionScore):** This checks if the intersectionScore property of a and b is the same. If true, it means that the intersection scores of the two elements are equal.
- **if (a.ageDifference === b.ageDifference):** If the intersectionScore is the same for a and b, it then checks if the ageDifference property is also the same.

If ageDifference is the same, it sorts by the intersection of hobbies between the student and the user profile. It calculates the intersection size for b and a using the getIntersectionOfArrays function, and then returns the difference: getIntersectionOfArrays(b.student.Hobbies, userProfile.Hobbies) - getIntersectionOfArrays(a.student.Hobbies, userProfile.Hobbies). This effectively sorts the elements based on the number of common hobbies, in descending order.

- **return a.ageDifference - b.ageDifference:** If the intersectionScore is the same for a and b, but their ageDifference is different, it sorts them based on the ageDifference in ascending order. This means that elements with smaller age differences will be placed before those with larger age differences.
- **return b.intersectionScore - a.intersectionScore:** If the intersectionScore of a and b is different, it sorts them based on the intersectionScore in descending order. This means that elements with higher intersection scores (more common interests and hobbies) will be placed before those with lower scores.

```
  // Return the best match
  return matchBenchmark.length > 0 ? matchBenchmark[0].student : null;
}
```

1. **matchBenchmark.length > 0:** This checks if the matchBenchmark array has at least one element. If the array is not empty, it means there are potential matches found.
2. **?:** This is the ternary operator, which is a shorthand for an if...else statement. It takes the form condition ? valueIfTrue : valueIfFalse.
   Reference : https://www.geeksforgeeks.org/what-are-ternary-operator-in-javascript/
3. **matchBenchmark[0].student:** If the matchBenchmark array is not empty (the condition matchBenchmark.length > 0 is true), this part of the expression is evaluated. matchBenchmark[0] refers to the first element in the matchBenchmark array, which should contain the best potential match based on the sorting logic applied earlier. .student accesses the original student object within that element.
4. **null:** If the matchBenchmark array is empty (the condition matchBenchmark.length ¿ 0 is false), this part of the expression is evaluated, and the function returns null.

```
// Function to calculate the intersection of two arrays
// Used to get intersection of the students' and users' hobbies and intersests
function getIntersectionOfArrays(arr1, arr2) {
  return arr1.filter(item => arr2.includes(item)).length;
}
```

Reference : stackoverflow.com/questions/1885557/simplest-code-for-array-intersection-in-javascript

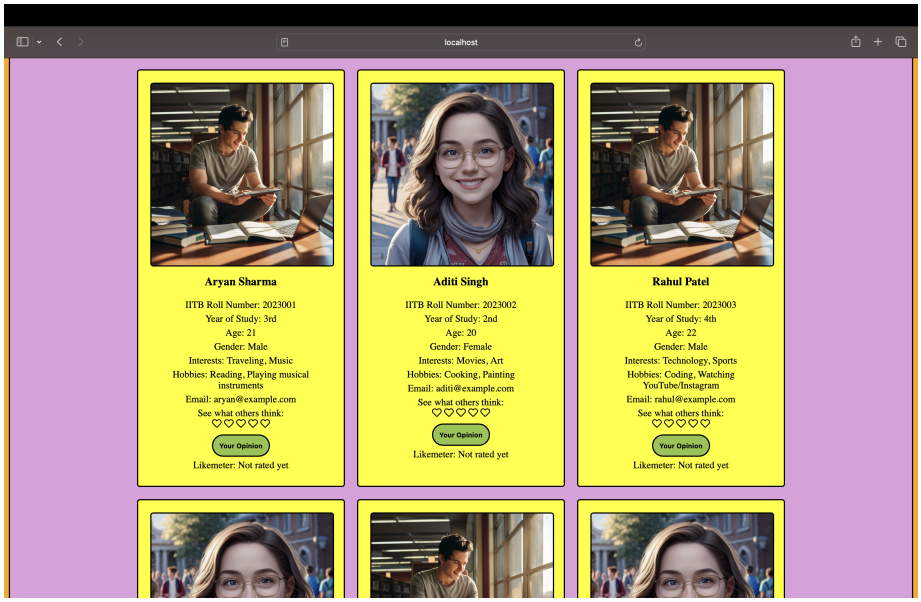
1. The function takes two arguments: arr1 and arr2, which are the two arrays for which we want to find the intersection.

2. Inside the function, the filter method is used on arr1. The filter method creates a new array with all elements that pass the test implemented by the provided function.
3. The provided function for filter is an arrow function: item => arr2.includes(item).
4. For each item in arr1, the arrow function checks if arr2 includes that item using the includes method.
5. If arr2 includes the item, the arrow function returns true, and that item is included in the new filtered array.
6. After filtering arr1, we're left with a new array containing only the elements that are present in both arr1 and arr2.
7. Finally, the length property is used on the filtered array to get the count of the common elements, which represents the intersection size.

- **Opening a New Window with Match Details**
  - If a best match is found, a new window is opened using **window.open**.
  - A div element is created and appended to the new window's document body.
  - The **generateMatchHTML** function is called with the best match data to generate an HTML string representing the match details.
  - The generated HTML is set as the **innerHTML** of the div element, displaying the match details in the new window.
  - If no match is found, an alert is shown with the message "Sorry, no suitable match found."

4.5 scroll_or_swipe.html

At first I was using javascript within the html itself, but then due to the customisation it got too large, so I decided to seaparate the javascript. The javscript is written in **scrollScript.js**.



HTML :

1. The HTML file sets up the basic structure of the web page.
2. It includes a $< div >$ element with the class **card-container** where the student profile cards will be rendered.
3. It links to an external CSS file styleScroll.css for styling the profile cards.
4. It also links to the **Font Awesome CSS library** to use icons for the rating system. (will elaborate ahead in customisations)
5. At the bottom, it includes the scriptScroll.js file, which contains the JavaScript code for fetching student data and rendering the profile cards.

Javascript :

1. I have used the Fetch API to retrieve student data from a JSON file (students.json).
2. For each student in the fetched data, it creates a new $< div >$ element with the class card to represent the student's profile card. This is facilitated with the **forEach()**
3. Inside each card, I have created elements to display the students' photo, name, roll number, year of study, age, gender, interests, hobbies, and email.
4. **More about the code to be discussed in Customisations**

NOTE : I have added **http://localhost:8000/** before the file path in the photo field of **students.json** as I was not getting the images in localhost before that.

5 Customisations

5.1 TOTP Verification for Login

I have added another factor at the login page that requires a **6 digit authentication code** to login.
I have used this library : https://www.npmjs.com/package/totp-generator
Installation : npm i totp-generator

How to use

```
import { TOTP } from "totp-generator"

// Keys provided must be base32 strings, ie. only containing characters ma
const { otp, expires } = TOTP.generate("JBSWY3DPEHPK3PXP")

console.log(otp) // prints a 6-digit time-based token based on provided ke
```

Default token settings

- SHA-1
- 30-second epoch interval
- 6-digit tokens

Even though I installed the library using npm install, I was encountering some problems. Therefore, I used the Skypack CDN link : https://cdn.skypack.dev/totp-generator
This requires an updated version of the **login.json** with the field **totpKey**.

This site can be used to generate the 6 digit codes from the keys : https://totp.danhersam.com/

5.2 Added Audio to make the site more responsive

This was accomplished by :

1. Adding <audio id=”audio-id”><source=”audio-file-path” type=”audio/mpeg”></audio> to the html.
2. Using getElementById to get the audio.
3. Adding audio.play() to the eventlistener.

5.3 Likemeter

The code for this is in **scriptScroll.js**. The code functions as follows :

1. It creates a "rating section" within each card, which includes:
   - A label indicating **See what others think**.
   - **Five heart icons (using Font Awesome icons)** to represent the rating.
   - A button labeled **Your Opinion** that, when clicked, prompts the user to rate the student on a scale of **1 to 5**.
   - A label displaying the student's current average rating or "Not rated yet" if no rating has been given.
2. When the **Your Opinion** button is clicked, the **rateStudent** function is called, which prompts the user to enter a rating and then calls the **updateStudentRating** function to update the student's rating and update the heart icons and **likemeter** accordingly.
3. The **updateStudentRating** function handles the following tasks
   - Adds the new rating to the student's ratings array.
   - Calculates the average rating for the student.
   - Updates the student's likemeter field with the average rating.
   - Calls the **updateHeartIcons** function to update the heart icons based on the new average rating.
   - Calls the **updateLikemeter** function to update the likemeter label with the new average rating.
4. The updateHeartIcons function updates the heart icons to reflect the student's average rating. It uses the Font Awesome CSS classes to change the icons from outlined hearts to filled hearts based on the rating value.
5. The updateLikemeter function updates the likemeter label with the new average rating, formatting it to **two decimal places**.

Reference : https://www.w3schools.com/icons/fontawesome_icons_intro.asp

5.4 Sending Email to Any Profile

```
<!DOCTYPE html>
<html>
<head>
    <title>Send Mail</title>
    <script src="https://smtpjs.com/v3/smtp.js">
    </script>

    <script type="text/javascript">
        function sendEmail() {
            Email.send({
                Host: "smtp.gmail.com",
                Username: "sender@email_address.com",
                Password: "Enter your password",
                To: 'receiver@email_address.com',
                From: "sender@email_address.com",
                Subject: "Sending Email using javascript",
                Body: "Well that was easy!!",
            })
                .then(function (message) {
                    alert("mail sent successfully")
                });
        }
    </script>
</head>

<body>
    <form method="post">
        <input type="button" value="Send Email"
               onclick="sendEmail()" />
    </form>
</body>
</html>
```

I have used the method as linked in the **problem statement pdf**.
Reference : https://www.geeksforgeeks.org/how-to-send-an-email-from-javascript/

Only changes I made were, adding a prompt to get email id and password from the user.

So, now on each card on **scroll_or_swipe.html** there is a **Go For It!** button, which when clicked prompts the user for credentials and then handles the sending of the email.