CS108 Project - Looking for a date?

Abhi Jain

April 2024

Contents

1 Introduction

The dating website project developed for CS108 represents a culmination of core principles in web development and creative problem-solving. By integrating HTML, CSS, JavaScript, Node.js, and npm, the project aims to provide users with an immersive platform for discovering potential matches based on personal preferences and interests.

In alignment with the course objectives, the project tackles fundamental challenges in web development, including login authentication, profile creation, and dynamic content generation.

2 Project Submission Details

2.1 Submission Files

1. `login.html`
   `forgot.html` not submitted as its functioning was included in the `login.html` only
2. `dating.html`
3. `scroll_or_swipe.html`
4. `match.html`
5. `change.html`
   For some customisations
6. `style.css`
7. `script.js`
8. `server.js`
   For implementation of `Node.js`
9. `package.json`
   Metadata file containing project details and dependencies
10. `package-lock.json`
    Records precise dependency versions to ensure consistent installations in Node.js
11. `Directory: Images`
    Contains all the visual assets used throughout the site
12. `report.pdf`
13. `students.json and login.json`
    JSON files given with problem statement - modified to add a few more Users.
14. `Directory: Photos`
    Profile photos for the corresponding users - given with the problem statement.

2.2 How to run the website?

In case of any discrepancies, here are the versions of Node.js and npm maintained by me in my system:

1. `node --version v20.5.1`
2. `npm --version 9.8.0`

Follow the following steps to run the website:

1. To set up the project, first, ensure all the submitted files are located in the same directory. If you've received a ZIP file, extract its contents to the desired directory.
2. Once the files are in place, navigate to the project directory using the command line interface. Then, run `npm install` to fetch all the dependencies specified in the `package.json` and `package-lock.json` file.
3. After installing the dependencies, execute `node ./server.js` to start the server.
4. Finally, open a web browser and go to the URL `http://localhost:3000` to access the running application.

3 Basic Tasks

3.1 Login Page

The login page serves as the initial point of entry for users seeking access to the dating website's input interface. As outlined in the problem statement, users are required to provide a valid username and password to proceed further. Upon submission of credentials, the system verifies the user's identity against pre-registered entries stored in a file named `login.json`.

The login process is facilitated by both front-end and back-end components. The `login.html` file contains the HTML structure and form elements necessary for user input, while the server-side logic is implemented using Node.js. When users submit their credentials via the login form, a request is sent to the server, where the details are validated against the entries in `login.json`.

In accordance with the project specifications, appropriate error messages are displayed to users in case of invalid credentials or other authentication failures. This ensures a seamless and secure login experience, allowing registered users to access the website's features while maintaining the integrity of user data.

On the right side there is a slide show of love images and quotes alongside giving an aesthetic vibe.

3.2 Forgot Password?

In the login page, a "Forgot Password" button is provided to facilitate password recovery for previously registered users who might have forgotten their credentials. This feature is implemented using Node.js and integrated into

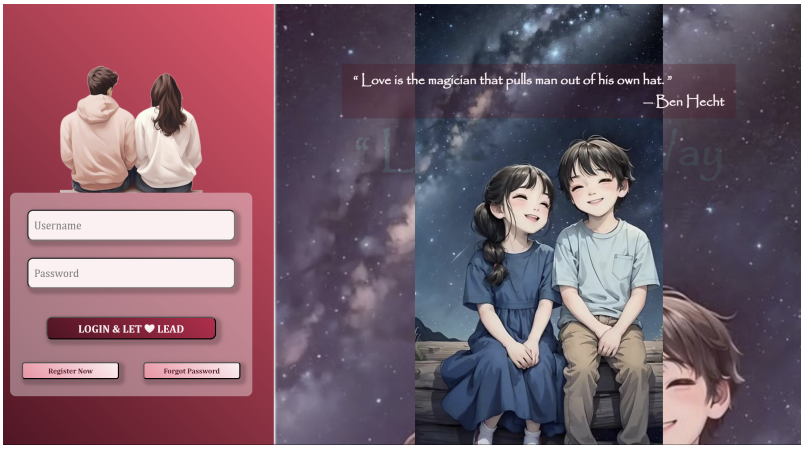

Figure 1: login image

the `login.html` page. However, it is also mentioned that integration into the `login.html` page itself is allowed.

When a user clicks the "Forgot Password" button, they are prompted to enter their username. Upon submission, the system retrieves the corresponding secret question from the `login.json` file and prompts the user to answer it. The system checks the provided answer, treating it as case-insensitive to enhance user convenience.

If the user enters the correct answer to the secret question, their password is displayed on the screen through alert. Otherwise, an appropriate error message is displayed to guide the user.

The integration of the password recovery system into the `login.html` page ensures a seamless user experience, allowing users to recover their passwords without navigating away from the login interface.

This approach enhances user accessibility and security by providing a straightforward mechanism for password recovery while maintaining robustness against potential credential misuse.

3.3 Input Interface

The input interface, implemented in the `dating.html` file and styled with `style.css`, offers a highly interactive user experience for inputting personal details. It adheres to the structure outlined in the `students.json` file, ensuring seamless integration with the existing data format.

The input interface is organized into several engaging and interactive components:

1. **Basic Information:** Users are prompted to provide essential details such as their name, IITB Roll Number, year of study, age, and email address. These fields serve as the foundational elements of user profiles.
2. **Hobbies Selection:** A dynamic checkbox-based interface allows users to select from a range of hobbies. The interactive nature of this component enables users to indicate multiple hobbies effortlessly.
3. **Interests Selection:** Similar to the hobbies selection process, users can choose their interests through an interactive checkbox interface. The intuitive design encourages users to express their diverse interests effectively.
4. **Photo Upload:** The interface features an intuitive file upload mechanism for users to upload their profile picture. Upon selection, the chosen image is visually displayed alongside the form, enhancing the overall user experience.

The "Submit" button serves as a pivotal element in the user interaction flow, facilitating a seamless transition to the scroll_or_swipe page. While its primary function is to initiate the process of finding the right match based on the provided details, its strategic placement and clear labeling contribute to an intuitive user experience.

Upon clicking the "Submit" button, users are seamlessly redirected to the scroll_or_swipe page, where a visually appealing card-based interface awaits. The central card prominently presents the option to find the right match, strategically positioned to catch the user's attention immediately. This deliberate design choice not only streamlines the user journey but also enhances the overall professionalism and aesthetic appeal of the website.

Through its interactive elements and user-centric design, the input interface enhances user engagement and simplifies the data entry process. The intuitive interface encourages users to provide comprehensive and accurate information, thereby facilitating effective matchmaking based on user preferences.

By organizing the input process into distinct steps, the interface enhances usability and ensures a smooth user experience. Each step focuses on specific aspects of data collection, allowing users to provide detailed and accurate information effortlessly.

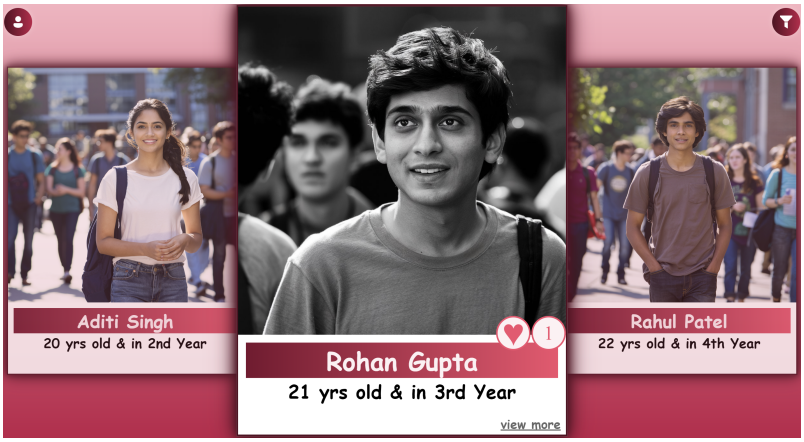

Figure 2: scroll_or_swipe page

3.4 Scrolling/Swiping

In the `scroll_or_swipe.html` page, each profile is displayed within a card format, featuring essential details such as the student's image, name, year of study, and age. Additionally, a "View More" option is provided for users to access comprehensive details of each profile.

The central card is the focal point of the interface, ensuring that users' attention is directed towards the currently displayed profile.

To navigate through the profiles, users have multiple options:

1. **Drag Gesture:** Users can simply drag the screen in the desired direction to scroll through the profiles smoothly.
2. **Click Interaction:** Clicking on a specific card allows users to directly view the corresponding profile, providing a straightforward navigation method.
3. **Mouse Scroll:** Users can utilize the mouse scroll wheel to navigate through the profiles vertically. However, it's important to note that this method may have limitations, particularly when using a laptop trackpad.
4. **Keyboard Navigation:** Alternatively, users can utilize the right and left arrow keys on the keyboard to navigate through the profiles horizontally, offering a convenient option for those who prefer keyboard-based interactions.

By providing multiple navigation methods, including intuitive gestures and keyboard shortcuts, the `scroll_or_swipe.html` page ensures a seamless and user-friendly browsing experience for exploring the various student profiles stored in the `students.json` file.

3.5 Finding the Right Match

The process of finding the right match involves comparing the interests, hobbies, age, and year of study of each student profile with the current user's preferences. This comparison is performed using a scoring system, where higher scores indicate better matches.

1. **Loading Student Profiles:**
   - Initially, the script fetches the student profiles from the `students.json` file using the fetch API.

2. **Scoring System:**
    - Each student profile is evaluated based on various criteria, including interests, hobbies, age, and year of study.
    - A scoring mechanism is employed to assign a score to each profile based on the similarity of interests and hobbies, as well as the proximity of age and year of study to the current user's preferences.
    - The scoring algorithm assigns higher scores for profiles with closer matches in interests, hobbies, age, and year of study. The scoring system is explained in the following subsection (ref:3.5.1).
3. **Iterative Comparison:**
    - The script iterates through each student profile in the fetched data.
    - For each profile, it calculates a score based on the mentioned criteria.
    - The maximum score among all profiles is tracked to determine the best match.
4. **Determining the Right Match:**
    - The profile with the highest score is considered the right match for the current user.
    - In case of ties, where multiple profiles have the same highest score, one profile is arbitrarily chosen as the right match.
5. **Exclusion Criteria:**
    - The current user's own profile is excluded from the matching process to ensure that the right match is not the user themselves.
    - Additionally, if a match type filter is applied (such as gender preference), profiles that do not meet the specified criteria are skipped. Actually, for making the website more inclusive, I gave a option to choose your right match type in the side menu (ref:5.4) whose default value is set to opposite gender but can be changed as required to male, female or all.

By employing this algorithm, the script efficiently identifies the most compatible match for the user based on their preferences and the available student profiles.

3.5.1 Scoring Algorithm

The scoring system employed in the algorithm calculates the compatibility score between two student profiles based on various factors, including year difference, age difference, interests, and hobbies. Each factor contributes to the total score, with different weightage assigned to each criterion.

- **Year Difference and Age Difference:**
    - A maximum of 10 marks is allotted for both year difference and age difference.
    - For each additional year of difference in year of study or age, 2 marks are deducted from the maximum score.
- **Interests and Hobbies:**
    - A maximum of 30 marks is allocated for both interests and hobbies.
    - The calculation of the score for interests and hobbies is based on the similarity between the profiles' interest and hobby selections.
    - The score is incremented based on the proportion of shared interests and hobbies, with additional considerations for the number of interests and hobbies selected by each profile.
    - The formula for calculating the score involves factors such as the length of the interests and hobbies lists, as well as a proportional increase in score for shared interests and hobbies.

This scoring system ensures that profiles with closer matches in terms of interests, hobbies, year of study, and age receive higher compatibility scores, thereby facilitating the identification of the most suitable match for each user.

```
//Interests_Score
let n1 = currentStudentData.Interests.length;
let n2 = student.Interests.length;
increase = (7.5 / n1) + (7.5 / n2) + (15 / (n1 + n2));
for (let interest of student.Interests) {
    if (currentStudentData.Interests.includes(interest)) {
        score += increase;
    }
}
```

Listing 1: Scoring System Algorithm for Interests

3.6 Output Interface

In the user interface, specifically on the `swipe_or_scroll.html` page or in the side menu accessible from all pages, users have the option to initiate the process of finding the "right match." This functionality is triggered by clicking the "Find Right Match" button or clicking the button from the side menu.

Upon activation, the system systematically identifies a suitable match for the user based on predefined criteria. The matching process is facilitated by JavaScript code implemented in the `script.js` file.

In the side menu, users can specify their preferred match type by selecting from options such as "Opposite Gender," "Male," "Female," or "All." By default, the match type is set to "Opposite Gender," but users have the flexibility to adjust this setting according to their preferences.

From the matching algorithm, once the system identifies a suitable match, it displays the match's profile along with relevant details in a new tab or within the same page interface, depending on user preference. The displayed information typically includes the match's name, age, year of study, interests and hobbies.

The system is designed to ensure that a match is always provided.

4 Server-Side Node.js Implementation

Node.js serves as the backbone for several critical functionalities within the dating website project. Its asynchronous nature and efficient handling of I/O operations make it ideal for tasks such as user authentication, registration, profile customization, and file manipulation.

4.1 Key Components

- **Login, Registration, and Forgot Password Functionality:** Node.js handles HTTP requests for user authentication by verifying credentials stored in JSON files. It ensures secure access to the website and facilitates password retrieval mechanisms. (References: 3.1, 3.2)
- **Customizations:** Node.js is instrumental in implementing customization features such as modifying user details, increasing/decreasing likes in JSON files, and managing user preferences. These functionalities leverage Node.js's asynchronous capabilities to process user requests concurrently. (References: 5.2, 5.1, 5.5)
- **Profile Picture Upload:** Node.js, coupled with the Multer middleware, simplifies the process of uploading profile pictures to the server. Multer enhances file handling capabilities, making it efficient for handling multipart/form-data requests. (References: 5.3)

4.2 Necessity of Node.js for Editing JSON Files

Node.js is indispensable for tasks involving file manipulation, particularly when dealing with JSON files like `login.json` and `students.json`. The built-in file system module (`fs`) enables CRUD(Create, Read, Update, Delete) operations on JSON files efficiently and securely.

4.3 Sample Code Snippets

4.3.1 Handling POST requests for user registration:

```
// Sample code demonstrating handling POST requests for user registration
else if (req.method === 'POST' && req.url === '/register') {
    // Parse request body to extract user data
    let body = '';
    req.on('data', chunk => {
        body += chunk.toString();
    });
    req.on('end', () => {
        const user = JSON.parse(body);
    // Read existing users from login.json
        fs.readFile('login.json', (err, data) => {
            if (err) {
                console.log(err);
            } else {
                const users = JSON.parse(data);
                // Add new user to the list of users
                users.push(user);
                // Write updated user list back to login.json
                fs.writeFile('login.json', JSON.stringify(users), (err) => {
                    if (err) {
                        res.end(JSON.stringify({ registration: false }));
                        console.log(err);
                    } else {
                        res.end(JSON.stringify({ registration: true }));
                    }
                });
            }
        });
    });
}
```

4.3.2 Handling Normal File Access

```
// Sample code demonstrating serving static files
else {
    // Determine file path based on request URL
    let filePath = '.' + req.url;
    if (filePath === './') {
        filePath = './login.html'; // Default file to serve
    }

    // Determine content type based on file extension
    const extname = String(path.extname(filePath)).toLowerCase();
    const contentType = {
        '.html': 'text/html',
        '.css': 'text/css',
        '.js': 'text/javascript',
```

```
        '.json': 'application/json',
        '.png': 'image/png',
        '.jpg': 'image/jpg',
        '.gif': 'image/gif',
        '.svg': 'image/svg+xml',
    }[extname] || 'application/octet-stream';

    // Read file and serve it to client
    fs.readFile(filePath, (err, data) => {
        if (err) {
            if (err.code === 'ENOENT') {
                res.writeHead(404);
                res.end('404 Not Found');
            } else {
                res.writeHead(500);
                res.end('500 Internal Server Error');
            }
        } else {
            res.writeHead(200, { 'Content-Type': contentType });
            res.end(data, 'utf-8');
        }
    });
}
```

This code snippet handles the serving of static files such as HTML, CSS, JavaScript, images, and other assets. When a request is received for a file, it determines the file path based on the request URL. Then, it determines the content type based on the file extension to set the appropriate HTTP response header.

The code uses the Node.js built-in modules `fs` for file system operations and `path` for working with file paths. These modules enable reading files from the server's file system and determining file extensions, respectively.

The `fs.readFile` function reads the requested file asynchronously, and the `res.end` function sends the file content back to the client with the appropriate content type header.

This mechanism allows the server to serve static files, such as HTML pages, CSS stylesheets, JavaScript scripts, and images, to the client browser, enabling the rendering of web pages and other resources.

4.4 Utilization of Fetch Command

Client-side interaction with the server-side Node.js implementation is facilitated using the `fetch` API. Below is a sample code snippet demonstrating its usage for user authentication:

```
function authenticateUser(username, password) {
    // Send a POST request to the server with the username and password
    fetch('/authenticate', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify({ username, password }),
    })
        .then(response => response.json())
        .then(data => {
            if (data.authenticated) {
                // Handle authenticated user
            } else {
                // Handle authentication failure
            }
        })
```

```
        .catch((error) => {
            console.error('Error:', error);
        });
}
```

5 Customisations

5.1 Registration and Adding Personal Details

In the registration process, a form is provided within the `login.html` page. This form prompts the user to input various personal details including:

- **Username:** The username must be at least 5 characters long and cannot contain spaces. Additionally, the system checks if the username is not already taken by querying the server using the `fetch` API.
- **Password:** The password must be at least 8 characters long.
- **Confirm Password:** This field requires the user to re-enter their chosen password to ensure accuracy.
- **Secret Question:** A security measure to help users recover their password in case it's forgotten.
- **Secret Answer:** The answer to the secret question, which should be memorable but not easily guessable.

Only registered users are permitted to submit their personal details, with each user being allowed to do so only once. Upon successful registration, the user's information is appended to the `login.json` file, which serves as a repository for student profiles. Subsequently, the user is redirected to the login page.

Upon login, if the user has not previously submitted their personal details, they are redirected to `dating.html`. However, if the user has already provided their personal details, they are redirected to `scroll_or_swipe.html`.

When a user enters their details in `dating.html`, the information is appended to the `students.json` file. To track whether a user has registered before or not, a registration key is added to each user object in `login.json`. Additionally, the username associated with each profile is stored within the student objects in `students.json`.

Initially, the original `students.json` file lacks a `username` field linked to each profile, and `login.json` does not contain the `registration` key. This discrepancy is rectified through the `script.js` file, which assigns a default username of "N/A" to each profile in `students.json` and sets the `registration` key to `false` for all users in `login.json` as a form of error handling. For new users, their information is appended to the respective files as usual.

```
document.addEventListener('DOMContentLoaded', function () {
    //check the local storage for if files are correct
    if (!localStorage.getItem('filesCorrect') || localStorage.getItem('filesCorrect') ==
     'false') {
        fetch('/recheckJson', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
            }
        })
            .then(response => response.json())
            .then(data => {
                if (data.addition) {
                    localStorage.setItem('filesCorrect', 'true');
                    console.log("JSON file is correct");
                } else {
```

```
                    console.log("JSON file is incorrect");
                }
            })
            .catch((error) => {
                console.error('Error:', error);
            });
    }
})
```

Listing 2: script.js code for rechecking files

To ensure that the desired keys are present in the JSON files, a local storage variable is created, and the script responsible for this functionality is executed from the `server.js` and `script.js` files without encountering any errors. The code ensures that the old files are modified correctly to incorporate the customisations. The fetch command calls this request in `server.js` to make the modifications.

To ensure a seamless login and logout process, a logout option is provided on each page within the side menu. This enhances the user experience, providing a professional touch akin to that of a sophisticated website.

5.2 Modifying Personal Details

Upon logging in, users have the option to modify their personal details by accessing the "Change Personal Details" section in the side menu. In this section, the user's existing details, retrieved from the `students.json` file, are displayed. However, for security reasons, the modification of the roll number and name is not allowed. The same interface used for registration and initial personal details submission is leveraged for modifying personal details, offering a seamless and consistent user experience.

The user can edit the required details directly in the interface. Before submitting the changes, the system ensures that the new username is not already taken and that the password and username meet the required conditions. This validation process helps maintain data integrity and security.

Upon submission, the changes are sent to the server through fetch APIs. These APIs handle the updating of the corresponding details in the JSON files stored on the server.

Additionally, the interface provides options to change the username and password. Clicking on these options redirects the user to `change.html`, where they can perform the desired changes.

This functionality ensures that users can conveniently update their personal information while maintaining security measures to protect sensitive data.

5.3 Uploading the Profile Pics

To enable users to upload their profile pictures, a dedicated endpoint was implemented in `server.js`. This functionality leverages the Multer middleware, which facilitates the handling of multipart/form-data, typically used for file uploads.

The code snippet defines a Multer storage configuration, specifying the destination directory for storing uploaded files and generating unique filenames to prevent conflicts. The uploaded files are stored in the `photos` directory within the server.

```
const storage = multer.diskStorage({
    destination: function (req, file, cb) {
        const destinationDir = path.join(__dirname, 'photos');
        cb(null, destinationDir);
    },
    filename: function (req, file, cb) {
```

```
        const originalFileName = file.originalname;
        const fileExtension = path.extname(originalFileName);
        const newFileName = req.headers['new-filename'] + fileExtension;
        req.newFileName = newFileName; // Add new file name to req object
        cb(null, newFileName);
    }
});

const upload = multer({ storage: storage }).single('file');
```

Listing 3: Multer setup for file upload

The `upload` middleware function processes a single file from the incoming request under the 'file' field. If an error occurs during the upload process or no file is provided, appropriate error messages are returned to the client.

This customization required extensive debugging and iterations to overcome challenges with file handling, such as converting image files to binary and ensuring consistent naming conventions. Despite these backend complexities, the user experience remains unaffected, while enhancing the efficiency of content transfer within the website.

The use of Multer simplifies file upload handling and provides robust features for managing file storage, ensuring seamless integration of profile picture upload functionality into the application.

5.4 Side Menu

Under the Side Menu section, I've developed a sophisticated side menu designed to enhance website navigation. This intuitive menu, accessible by hovering over the user icon on the top left corner, provides seamless access to various features:

- **Find a Match**: Directs users to the matching page (`matching.html`) where they can discover potential matches.
- **Swipe the Profile**: Navigates users to `scroll_or_swipe.html`, offering an interactive experience to browse profiles.
- **Enter/Change Personal Details**: Facilitates users in updating their personal information.
- **Choose Match Type**: Offers options to select match preferences, including "All", "Male", or "Female".
- **Change Username**: Allows users to modify their username as desired.
- **Change Password**: Enables users to update their password securely.
- **Logout**: Provides a convenient option to sign out, ensuring a smooth transition.

With its aesthetically pleasing UI and consistent visibility across all pages, this side menu serves as a pivotal element, seamlessly connecting users to various sections of the website. Its presence elevates the overall user experience, akin to that of a professional website.

5.5 Likes/Like Meter

In implementing the Like Meter feature, JavaScript was utilized to create a dynamic and interactive experience for users browsing profiles on the `scroll_or_swipe.html` page.

1. **User Interaction:** Users can like/rate profiles by either clicking the like button or pressing the space bar while viewing a profile.

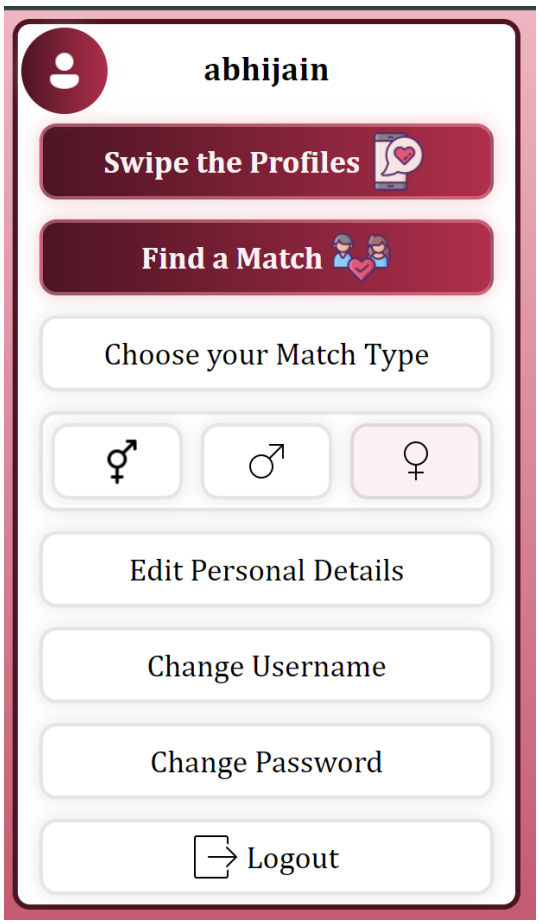

Figure 3: side menu

2. **Fetch API Call:** Upon liking a profile, a fetch API call is triggered to update the like count in the `students.json` file on the server. This call modifies the corresponding student's JSON object to reflect the updated like count.
3. **Limitation:** There is a limitation where the same user can like a profile multiple times upon page reloads. However, during the same browsing session, the user cannot like a profile more than once.
4. **Animation:** An appealing animation for the like button was implemented to enhance user engagement. This adds a visual element to the liking process, making it more interactive and enjoyable for users.
5. **Impact:** The Like Meter feature allows users to express their preferences and contribute to the overall popularity of profiles. It encourages user interaction with profile content and adds depth to the browsing experience.

Including this feature enhances the overall usability and engagement of the platform, making it more appealing to users and facilitating greater interaction with profile content.

5.6 Filters

The addition of filters enables users to refine their profile search based on specific criteria such as interests, hobbies, year of study, gender, and more. This feature enhances the browsing experience by providing users with personalized results that closely match their preferences.

1. **User Interface:** Filters are accessible through the filter icon located at the top-right corner of the `scroll_or_swipe.html` page. Clicking on this icon opens a menu where users can select their desired filter criteria.
2. **Filter Options:** Users can select various filter options such as interests, hobbies, year of study, gender, and any other custom filters that have been implemented. These options allow users to specify their preferences and narrow down their search results accordingly.
3. **Filtered Results:** Upon selecting filter criteria, the profiles displayed during scrolling/swiping are dynamically updated to show only those that meet the specified criteria. This ensures that users are presented with relevant profiles that align with their selected preferences.
4. **Improved Matching:** By applying filters, users can find their "right match" more efficiently, as the selection is narrowed down to profiles that closely match their preferences. This increases the likelihood of finding compatible matches and enhances user satisfaction.

The inclusion of filters adds flexibility and personalization to the profile browsing experience, empowering users to discover profiles that resonate with their interests and preferences.

5.7 UI Enhancements

The user interface (UI) of the website has been enhanced with several new features and improvements, aimed at elevating the overall browsing experience and improving user engagement. Some of the key enhancements include:

- **Love Quotes:** Inspirational or romantic love quotes are displayed at strategic points throughout the website, adding a touch of sentimentality and emotional appeal to the user experience.
- **Login Page Animation:** The login page features animated elements or transitions, creating a visually dynamic and engaging entrance for users as they access the website. These animations help capture user attention and enhance the overall aesthetic appeal of the login process.
- **Smooth Scrolling/Swiping:** The scrolling/swiping functionality within the website has been optimized for smooth performance, ensuring seamless navigation between profiles and enhancing the fluidity of the browsing experience.
- **Side Menu:** A side menu or navigation panel has been added to the interface, providing users with convenient access to various features, options, and settings. The side menu enhances usability by organizing key functionalities in an easily accessible manner.
- **Enhanced Checkboxes:** Checkbox elements used throughout the website have been customized or stylized to improve visual appeal (ref:[9]) and usability. Enhanced check boxes may feature unique designs, animations, or interactive elements that make them more engaging for users.
- **Button Click Effects:** Buttons and interactive elements within the UI may feature click effects or animations that provide visual feedback to users upon interaction. These effects enhance the responsiveness of the UI and create a more dynamic user experience.
- **Additional Features:** Various additional features, such as hover effects, transition animations, custom icons, and tooltips, may have been implemented to further enhance the UI's visual appeal and user interaction.

By incorporating these UI enhancements, the website aims to deliver a more immersive, visually appealing, and user-friendly experience for visitors, thereby enhancing overall engagement and satisfaction.

References

[1] Coolors. Palette. https://coolors.co/palette/590d22-800f2f-a4133c-c9184a-ff4d6d-ff758f-ff8fa3-ffb3c1-ffccd5 Accessed in 2024.

[2] GeeksforGeeks. Geeksforgeeks - computer science portal for geeks, 2022.

[3] GitHub. Github copilot: Your ai pair programmer, 2021.

[4] Icons8. Icons8. https://icons8.com/, Accessed in 2024.

[5] OpenAI. Chatgpt: Openai's conversational ai model, 2021.

[6] Pexels. Pexels. https://www.pexels.com/, Accessed in 2024.

[7] Pinterest. Pinterest. https://www.pinterest.com/, Accessed in 2024.

[8] W3Schools. W3schools online web tutorials, 2022.

[9] W3Schools. How To - Custom Checkbox. https://www.w3schools.com/howto/howto_css_custom_checkbox.asp, Accessed in 2024.