CS108 Project
Report

HTML CSS Javascript Webdev

Bachelor of Technology in Computer Science and Engineering

Submitted by
23B1068, Vaibhav Singh

Department of Computer Science and Engineering
INDIAN INSTITUTE OF TECHNOLOGY BOMBAY
Powai, Maharashtra, India – 400 076

Spring Semester 2024

Contents

1 Problem Definition

The goal here is to create a basic dating website. We're using HTML, CSS, and JavaScript to build it. Simply put, we want to make a place where you can enter some info about yourself and then find someone who might be a good match for you among a bunch of students. It's all about making a website that's easy to use and helps people connect with each other.

2 Basic Tasks

Following are all the basic tasks that needed to be done.

2.1 Login Page

The project includes a login page feature, which acts as a gateway to access the input interface. Users are prompted to enter a username and password combination. If the entered credentials match those of a registered user stored in the "login.json" file, access is granted. Otherwise, appropriate error messages are displayed. This ensures that only registered users can access the website, even at a later time. The login page is implemented in a ".html" file named "login.html".

2.2 "Forgot Password?" Button

Additionally, the project incorporates a "Forgot Password?" feature to cater to instances where registered users forget their passwords. Upon clicking the designated "Forgot Password?" button on the login page, users are directed to a new page named "forgot.html" or, if integrated, remain on the login page itself. Here, users input their username, triggering the display of the corresponding secret question associated with the username from the "login.json" file. If the user correctly answers the secret question, the password is revealed on the screen. Otherwise, appropriate error messages are displayed, ensuring a streamlined password recovery process for registered users.

2.3 Input Interface

The task involves crafting an input interface using HTML and CSS for users to fill in their personal details. These details correspond to the fields in the "students.json" file, including "IITB Roll Number", "Name", "Year of Study", "Age", "Gender", "Interests", "Hobbies", "Email", and "Photo".

As customisation, I have also included 'Interested in' question to know the orientation of the user and display results accordingly.

For the "Gender" field, radio buttons are implemented to allow users to choose a single option. For "Interests" and "Hobbies", checkboxes are provided for users to select multiple options for each category.

Additionally, a "Submit" button is included, which, upon clicking, initiates the search for the person's ideal match. A "Logout" button is also provided to return users to the login screen for added convenience.

All of this functionality is designed and implemented within the "dating.html" file for the HTML structure and "style.css" file for the visual styling.

2.4 Swiping

A feature is added to the input interface to enable users to browse through all the students' profiles and photos stored in the "students.json" file. This page can be accessed through the navbar.

Upon clicking the button, users are directed to a new page named "scroll_or_swipe.html". This page allows users to freely scroll or swipe through profiles, viewing details and photos of each student in the dataset.

This page filters out the profiles according to the 'interested in' input provided by the user on the dating.html.

2.5 Finding the "right" match

The task involves improving the "Submit" button on the input interface ("dating.html") to find a suitable match for the user among the profiles in "students.json" using JavaScript. The match is based on shared interests and custom criteria. If no suitable match is found, an appropriate message is displayed. If there are multiple matches, one is chosen arbitrarily. The matchmaking process is flexible, adapting to individual preferences.

I used Jaccard's matching algorithm to generate the 'right' or we can say best match. This algorithm basically gives a number which is basically number of common interests/hobbies divided by total number of hobbies/interests. In other words size of intersection of their hobbies/interests divided by size of union of their hobbies/interests. greater the number of things common between two user... greater the score. To scale up the score, I have returned $\sqrt[2]{originalscore}$ and then multiplied it by 100 to convert it in percentage.

2.6 Output Interface

After clicking "Submit," a new tab will display the "right match" for the person, including their details if a match is found. The interface for this is structured in a separate ".html" file named "match.html", utilizing the same CSS ("style.css") and JavaScript ("script.js") files. On clicking submit in dating.html, first foundMatch.html is displayed. On hovering over the card in foundMatch.html, user can see the match. In case there is no match... The script leads the user to another webpage where an appropriate message is displayed.

3 JavaScript Code

I've implemented 23 functions, all coded in script.js. Some JavaScript is also present in HTML pages, such as Left NavBar functionality.
Following is the brief description of every function used:

3.1 submitHandler()

- Gets called when the submit button is pressed on the dating.html page.
- Takes in the input filled by the user and calls `findMatch()`.
- Stores the input provided by the user in local storage in the form of JSON.

3.2 findMatch()

- This function is called when the `submitHandler()` function is executed.
- It fetches the `students.json` file.
- Converts the response into a JavaScript object.
- Iterates through the object, calculating compatibility scores between the user and each student.
- Redirects to `foundMatch.html` upon completion.
- Converts the matched student data into a JSON string and stores it in local storage.

3.3 renderMatchFound()

- This function renders the match on the `matchFound.html` page.
- It retrieves the matched student data from local storage.
- Checks if the compatibility score is below 0.4 and redirects to `sorry.html` if so.
- Displays the matched student's name, image, and compatibility score.

3.4 renderMatch()

- This function renders the details of the matched student on the `match.html` page.
- It retrieves the matched student data from local storage.
- Displays the matched student's name, interests, hobbies, age, image, IITB roll number, year of study, and gender.

3.5 scoreCalculator(user1, user2)

- This function calculates the compatibility score between two users.
- It implements Jaccard's Algorithm to calculate the score based on shared interests and hobbies.
- Returns the compatibility score scaled to a percentage.

3.6 onLogin()

- This function is triggered when the login button is clicked.
- Fetches the `login.json` file.
- Verifies the entered username and password.

3.7 checkLoginData(data)

- This function verifies the user's login credentials and displays appropriate error messages.
- Retrieves the input username and password.

- Checks if the username exists and if the password matches the username.

3.8 findSecret()

- This function retrieves a user's secret question based on their username.
- Fetches the `login.json` file.
- Displays the user's secret question if the username exists.

3.9 checkSecretAnswer()

- This function checks the user's secret answer for password recovery.
- Retrieves the entered answer and username.
- Compares the entered answer with the stored secret answer for the given username.

3.10 checkFilter(person, filter)

- This function gets called when someone applies a filter.
- It takes `filter` and `person` as input and returns if the person passes the filter.

3.11 scroll()

- This function renders everyone (who matches the filters) in the scroll-container.
- It is called when `scroll_or_swipe.html` is loaded.

3.12 removeFilters()

- This function removes filters applied to the scroll-container.
- It redirects back to `scroll_or_swipe.html`.

3.13 logoutFun()

- This function is called when logout is clicked on the navbar.
- It removes all data stored in the local storage to prevent any conflicts if a different user logs in.

3.14 openFilterBar()

- This function is called when the filter button in the scroll_or_swipe.html page is clicked.
- It opens the filter bar by adjusting its height and changes the appearance of the filter button.

3.15 closeFilterBar()

- This function is called when the filter button in the scroll_or_swipe.html page is clicked and the filter bar is open.
- It closes the filter bar by adjusting its height and changes the appearance of the filter button.

3.16 FilterFunction()

- This function is called when the filter button is clicked in the filter bar in the scroll_or_swipe.html page.
- It collects the interests and hobbies selected by the user and stores them in local storage as filter data.
- It then redirects to the scroll_or_swipe.html page.

3.17 renderOwnDetails()

- This function is triggered when the user clicks on home when on any page other than dating.html.
- It retrieves the user's details and image from local storage and displays them on the page.

3.18 scroll_the_scroll_container(e)

- This function is called when the user presses any key on the scroll_or_swipe.html page.
- If the right arrow key is pressed, it scrolls the scroll-container to the next box, displaying the next user.
- If the left arrow key is pressed, it scrolls the scroll-container to the previous box, displaying the previous user.

3.19 setCookie()

- This function sets a cookie named `username` with the value of the input field with the ID `username`.
- The cookie expires after 1 hour.

3.20 getCookie(key)

- This function retrieves the value of the cookie with the specified `key`.

3.21 checkCookie()

- This function checks if the `username` cookie exists.
- If the cookie exists, it redirects the user to the `dating.html` page.

3.22 checkCookiePages()

- This function checks if the `username` cookie exists.
- If the cookie doesn't exist, it redirects the user to the `login.html` page and displays a prompt to log in.

3.23 removeFilterButton()

- This function checks if any filters are applied.
- If no filters are applied, it hides the remove filters button.