Looking for a date

Aditi Singh

27 April 2024

Contents

1 Introduction

The objective of this project was to design and implement a basic dating website using fundamental web technologies such as HTML, CSS, and JavaScript. This dating platform facilitates personalized matchmaking, where individuals can sign up, complete a detailed profile, and utilize a tailored matching algorithm to discover their ideal partner.

2 Basic Requirements

Before proceeding, ensure that your system possesses all the necessary components to extract and execute the code effectively. This includes having node js installed in your pc and all it's necessary modules which we will include over the course of website. These mainly includes `multer` and `bodyParser`. Alongwith that you will need a functioning web browser, a text editor or integrated development environment (IDE) for editing code, and a reliable internet connection for accessing online resources and libraries if required.

3 Usage

There is a very simple protocol how to operate the website

1. Extract the submission.zip.
2. Open the directory containing in terminal and run the command `node server.js` (make sure that no other server is running on port 4000)
3. You will see the following message `Server is running on port 4000`
4. "Next, open a web browser and enter the following address in the URL bar:
   `http://localhost:4000/welcome.html`
   This will direct you to the welcome page of the local server hosted on port 4000.

4 Basic Working

The project comprises multiple HTML and CSS files, each serving distinct purposes in the website's functionality and design. I will now explain the functionality and structure of these files to elucidate their roles in the project's implementation.

- **welcome.html**
  The welcome page is where new users get greeted on the website. It has two big buttons: "Sign In" for people who already have accounts and "Sign Up" for those who need to register. The page is redirected to `login.html` and `register.html` respectively on clicking those buttons. There's also a button at the bottom right to easily send in reviews. You can also view previously sent reviews through this. For more reference regarding this refer 2. This html page has both internal css and javascript.
- **login.html**
  The login page has a form where registered users can enter their login information to access the main website. The data for logging in is retrieved from a file called `login.json`. If the password is correct then the user is directed to `user.html` and if the password is incorrect then the error messages are displayed. Alongwith these we have options for user if they forgot their password they are redirected to `forgot.html` and if the user wants to register there is a option for it to be redirected to `register.html`. There is a button titled "Back to Home Screen" through which the user can go back to `welcome.html`. The CSS for this is in `styling.css` while the javascript is internal.

- **forgot.html**
  This page is made for users who have forgot their password. The form asks for the user's username and check for it's validity and if the username is valid it asks a personalized secret question. If the user answers the question correctly then the website provides the password for that username and redirects to `login.html`. The CSS is present in `styling.css` while the javascript is internal.
- **register.html**
  This page is for registering users on the website who are not already registered. It asks for four parameters usename,password,secret question and it's answer. The username must be unique and should not be already present in the `login.json`. This uses server side scripting that is nodejs to alter json file from the server side. The CSS for this is present in `styling.css` and the javascript is present in `server.js`.
- **user.html**
  After a successful login, the user is redirected to this page. Here, the user is provided with options to choose what they want to do. The options include add their profile, update their profile, Find their match, Scroll through profiles or want to logout. Alongwith there is a small arrow up button at the right side of the screen if user wants to fill any reviews regarding the website. It has internal css and javascript. At the bottom right corner there is a review button on toggling which a review box appears for the user to give reviews.
- **add_profile.html**
  This page is obtained when you click on `Make your own profile` in `user.html`. This page serves the purpose of enabling a user to add their profile to the server, which contains multiple profiles. It becomes particularly useful when a user doesn't find a suitable match and wishes to keep their profile on the website. This allows other users to access their profile when they search for a date. This uses server side scripting and node js for appending the data in `students.json`. The CSS for this is done in `formsheet.css`
- **update.html**
  This page is obtained when you click on `Update your profile` in `user.html`.This feature allows users to update their profile if they wish to do so. Users can enter their roll number to access the form they previously filled out, enabling them to make changes and updates as needed.This also uses server side scripting and node js for appending the data in `students.json`. The CSS for this is done in `formsheet.css`
- **dating.html**
  This page is obtained when you click on `Find your match` in `user.html`.This main page is where users input their profile details, such as age, gender, hobbies, interests, and other parameters. After submitting this information, my algorithm determines the best match for the user. The result is then displayed on the `match.html` page, showing the user their ideal match. The javascript for this page is present in `script.js` while the CSS is present in `formsheet.css`
- **match.html**
  This page, `match.html`, showcases the matched data from the matching process. It includes a button labeled "Wanna try again" that redirects users back to `dating.html` for another attempt. Additionally, there's another button labeled "Talk to your right match now" which sends a personalized email to the user's best match. This uses internal javascript and it's CSS is present in `newstylesheet.css`
- **scroll_or_swipe.html**
  This page is designed for users to scroll through all the profiles available in the `students.json` at the current moment. Users have the option to like any profile they find interesting. Additionally, there's a hamburger menu at the top left corner that, when clicked, pops up the

filter section for the page. Through this filter section, users can refine their view to display only the profiles that meet their specific criteria. This uses both client side as well as server side programming language. Server side language i.e. Nodejs is used for incrementing the likeCount in the profiles while the rest all functionalities are provided by internal js. The CSS for this is present in `newstylesheet.css`.

- **myprofile.html**
  This page is accessed after users submit their profiles in either add_profile.html or update.html. Upon submission, the code sets their username as a local storage variable, and they are directed to this page. Here, the page retrieves the value from local storage and displays the user's profile for review. If any changes are needed, users can navigate to update.html to make corrections to their profile.This uses internal javascript and it's CSS is present in `newstylesheet.css`

5 The Matching Algorithm

First of all I have created an array of student roll numbers who are of opposite gender from the given details through the form. Following that I've utilized a variable named "score" to determine the best match. This variable is dependent on another variable called "nscore.".The variable "nscore" increments by 1 for each common hobby/interest found between profiles. Here student contains students from different gender only.

```
student.Interests.forEach(function(interest) {
            if (selectedInterests.includes(interest)) {
                nscore += 1;
            }
        });
        student.Hobbies.forEach(function(hobby) {
            if (selectedHobbies.includes(hobby)) {
                nscore += 1;
            }
        });
```

Now the variable score comes into the picture. The variable is a weighted value of the number of interests and hobbies of both student and the user.

$$score = \frac{nscore}{Interests.length + Hobbies.length + student.Interests.length + student.Hobbies.length - nscore}$$

After this the score is decremented by some factor for age difference

```
               score=score-(Math.abs(student.Age - age)/10);
```

Now, I've introduced a variable named "highestScore" along with auxiliary variables to store the values related to the individual who currently holds the highest score. Additionally, I've implemented tie-breaking criteria, which is:

```
if (score > highestScore) {
    highestScore = score;
    rollNumberOfHighestScore = student['IITB Roll Number'];
    ageDifference = Math.abs(student.Age - age);
    yearofmatch = student.Year;
}
else if(score == highestScore){
    if (Math.abs(student.Age - age) < ageDifference){
        highestScore = score;
        yearofmatch = student.Year;
        rollNumberOfHighestScore = student['IITB Roll Number'];
        ageDifference = Math.abs(student.Age - age);
    }
    else if(year == student.Year){
        highestScore = score;
        rollNumberOfHighestScore = student['IITB Roll Number'];
        yearofmatch = student.Year;
        ageDifference = Math.abs(student.Age - age);
    }
}
```

After this I conclude that

```
if (highestScore < 0.2) {
         return 'nomatch';
        }
else {
        return rollNumberOfHighestScore;
}
```

This eventually lead me to the roll number of the student with highest score and hence the best match.

6 Customizations

Here is the list of customization I have done in this project over the course of project. Many of them were explained in the section 4. Still I will explain them thorughly.

1. **Register options:-** In addition to enabling users to simply log in, as part of the basic task, I have implemented a feature that allows new users to register on the dating website. This functionality was achieved through a basic use of `nodejs`. Through this any new user can register and their data could be stored in `login.json`. Their is a seperate page oriented for this that is `register.html`. For the basic understanding and usage of nodejs i have reffered [3] and [2]

2. **Review System:-** I've integrated a review system on the website where any user can share their thoughts or experiences from their visit. Users are free to add text detailing their experiences. Additionally, users have the option to view reviews submitted by others. All the reviws are stored in using nodejs, which is used for later reference when clicked on the button `View Previous Reviews`. Please note that this button wouldn't work if the `review.json` is empty or doesn't exist. The reviews are viewed as a list and on hovering over a particular review you can view the name of the author who gave the review. These review boxes are present in `welcome.html` and `user.html` at the bottom right corners.

3. **Adding your profile:-** I've included an option for users to add their profiles to the website, making them visible to all visitors. This feature allows users who haven't found a suitable match to display their profile on the website, enabling future visitors to review it. This has few more features which i will list underneath.

   (a) **Browse Photos:-** You can browse and select photos from your computer. After browsing, you can view a preview of your selected photo directly on the page. This was done by module called `multer`.I prompt the user to upload their photo to the website, and upon submission, I rename the photo accordingly and place it inside the "photos/" directory.

   (b) **Adding comments:-** You have the option to write comments, a bio, or preferences that you'd like others to know about you. This feature enables users to draft and share information about themselves. This is completely optional and is viewed in the profile if some-one sees this.

   (c) **User friendly UI:-**The forms I've created are designed to be user-friendly and highly interactive. The CSS styling enhances the user experience, encouraging users to engage with the forms more actively.

   This all is done in `add_profile.html`

4. **Update your Profile:-** In addition to adding profiles, users also have the option to update their profiles. They can access the form by entering their IITB Roll number. Once they access the form, it's **prefilled** with the information they provided earlier, allowing them to make changes more conveniently. Onsubmit the new details are displayed and updated in the `students.json`. This uses basic Nodejs as the above customizations as well. If any field while refilling the form is left empty then it's prefilled version is taken while if changed the new one is taken.

5. **Like buttons:-**While scrolling through profiles, users can like profiles they find appealing. The like button starts as a colorless star, but upon liking a profile, the button changes to yellow. Additionally, the user's like count increases, and the overall like count is displayed while scrolling through profiles. I have implemented a variable called likeCount which increases by 1 if a already liked profile is liked or set to 1 if it is liked for the very first time. Further this no. is written in `students.json` using `fs.writefile`. This can be displayed in `scroll_or_swipe.html`

6. **Filter buttons:-** In the scroll_or_swipe.html page, I've implemented filter buttons to assist users in refining their profile search according to their interests. The filter button is present at the top-left corner of the page essentially depicted by 3 horizontal lines. These buttons offer several features, including:

   (a) **Filters for Gender,Year of Study:-** I have created filters for gender and Year of Study which filters on their respective fields.

   (b) **Age Filters:-** I've enhanced the filter buttons by providing a range of ages, allowing users to specify the desired age range for potential matches. This approach is more convenient than having separate buttons for each age, making the filtering process smoother and more user-friendly.

   (c) **Interests and Hobbies filter:-** I've implemented interests and hobbies filters that function based on an intersection rather than a union set. This means that if a user selects two hobbies, only profiles that have both of those hobbies will be visible, rather than profiles that have either one. This ensures a more precise filtering process, matching users with profiles that closely align with their interests.

   (d) **Hamburger Menu:-**I've designed the filter buttons in a Hamburger menu style, enhancing user-friendliness and aesthetics. This stylish approach not only makes the filter buttons more visually appealing but also ensures ease of use for the users.

   In general these filter buttons work when the user clicks on the button "Apply Filters" in the Menu for the Filters. Onclick of the button I use a variable called `filteredStudents` which is the data container of only those feilds whoose profiles we want to see. This is later displayed on the page and then we can view them.

7. **Mail Right Match:-** I've implemented a feature where, after finding their perfect match, users can send a personalized email directly to them. The email is **pre-typed** for the user's convenience. Users have the option to either send the email directly or make any desired changes before sending it to their match. This was done by this code snippet.

```
<button type="button"
   onclick="window.open('mailto:${student.Email}?subject=${username}
   your perfect match&body=Hello there !! Myself ${username}
   ${conteent} ');">Talk to your right match now!</button>
```

In this *conteent* is a predefined variable which conntained a part of the body which was written. For this I took reference from [1].

8. **Match Status:-** In `match.html`, I've incorporated a custom Match Status option, which indicates the match's status based on the algorithm outlined in section 5. The page retrieves the score from local storage and categorizes the match into four broad classes: ideal, optimal, decent, and average. This is how I have classified these classes:

```
if (score>=0.7){
    status="You are an ideal match for each other";
}
else if (score<0.7 && score>=0.5){
    status="You are a optimal match for each other";
}
else if (score<0.5 && score>=0.3){
    status="You are a decent match for each other";
}
else{
    status="You are an average match";
}
```

9. **Customized Welcome messages:-** At the beginning of most HTML pages, I've integrated customized greeting messages for users, welcoming them to the site. These messages are personalized by retrieving the respective name from the last page set in local storage and then displaying it on subsequent pages. This addition enhances the user-friendliness of the website by creating a more welcoming experience for users. Alongwith these I have created welcome pages that are `welcome.html` and `user.html` which increases user Interface with the website and makes the website easy to use

References

[1] *How to send an email from JavaScript*. URL: https://stackoverflow.com/questions/7381150/how-to-send-an-email-from-javascript.

[2] *Node js documentation*. URL: https://nodejs.org/docs/latest/api/.

[3] OpenAI. *ChatGPT:* 2021. URL: https://openai.com/research/chatgpt.