Web Development Dating Project

Vivek Vardhan K

April 28, 2024

1 Introduction of the Project

This document provides an overview of a dating website created using HTML , CSS and JavaScript that takes input from the user and finds the right match for the user among a set of students using Javascript . The website has a login and signup system built using NodeJs. This website's name is `LoveLink`.

2 Basic working of LoveLink

2.1 Login

The `login.html , login.js` files handles the login logic . It fetches the data of users from `login.json` file. It has two variables initiated in JavaScript `$isAuthenticated $usernameExists` which go through each student in the `login.json` file and if the username and password matches , it allows the user to login and redirects the user to the page `dating.html`. If the username exists but the password is incorrect , website correspondingly shows the error message that `'Password is incorrect. Please try again.'` .
The corresponding javascript code is

```
document.getElementById('loginForm').addEventListener('submit',
function(event) {

// Get the username and password from the form
  const enteredUsername = document.getElementById('loginUsername').value;
const enteredPassword = document.getElementById('loginPassword').value;

// Fetch the login.json file
  fetch('./login.json')
.then(response => response.json())
.then(usersData => {
let isAuthenticated = false;
let usernameExists = false;
```

```
 // Loop through the users data
  for (const user of usersData) {
 // Check if the entered username matches the user's username in the
data
  if (user.username === enteredUsername) {
usernameExists = true;
// If the username matches, check the password
if (user.password === enteredPassword) {
isAuthenticated = true;
break; // Stop the loop as we found a match
} }
```

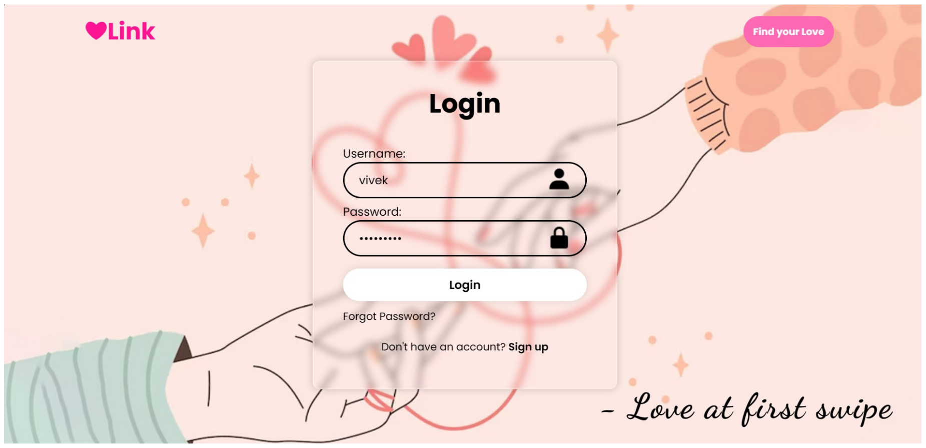


Figure 1: Login Page interface

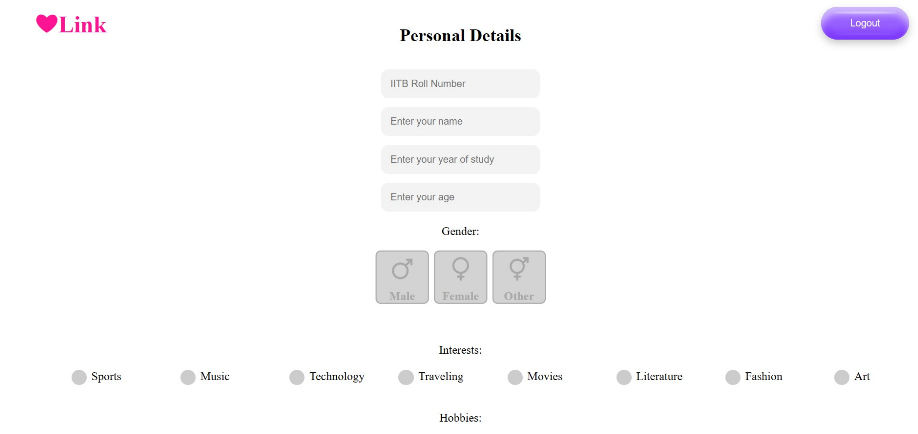
2.2 Input Interface

The `dating.html` file handles this input interface . It has input fields for

- **IITB Roll Number**
- **Name**
- **Year of Study**
- **Age**
- **Gender** [`Male` , `Female` , `Other`]
- **Interests** [`Travelling` , `Sports` , `Movies` , `Music` , `Literature` , `Technology` , `Fashion` , `Art`]

- **Hobbies** `[ Reading, Cooking, Coding, Gardening, Painting, Watching YouTube/Instagram, Playing musical instruments , Photography ]`
- **Email**
- **Photo**

When clicked on Submit it redirects it to `match.html` where the best match is shown and if there is no match it will show an error message that `"No good match is found"` .



Personal Details

IITB Roll Number

Enter your name

Enter your year of study

Enter your age

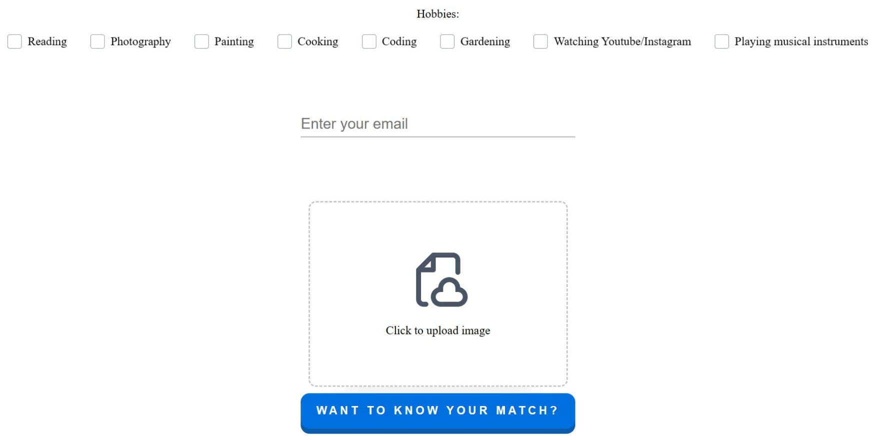
Gender:

Interests:

Sports Music Technology Traveling Movies Literature Fashion Art

Hobbies:

Figure 2: Input interface for user to enter details


Hobbies:

Reading Photography Painting Cooking Coding Gardening Watching Youtube/Instagram Playing musical instruments

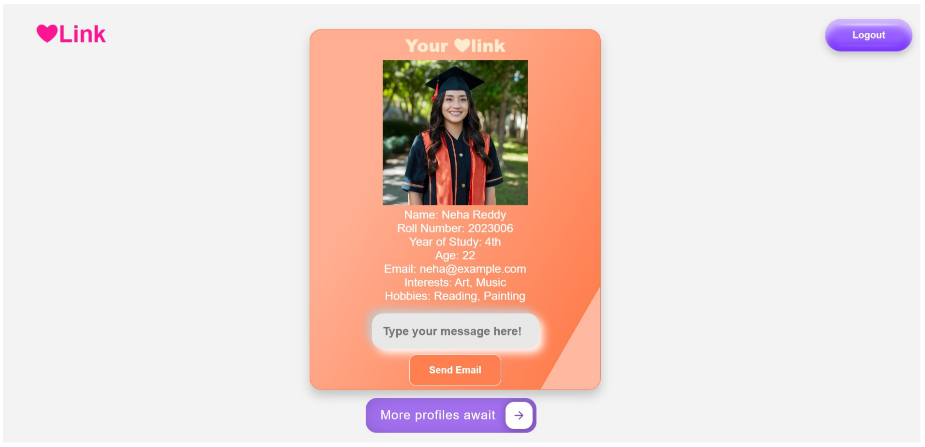
Enter your email

Click to upload image

Figure 3: Input interface to upload photo and enter email

2.3 Finding the right match

The `script.js` file handles the work of finding the right match . It first checks (through the Roll Number) if the logged in user is one of the `students.json` file . If it is so , then it removes that profile while checking for the right match . If the logged in user is a male , then the algorithm searches for females with the highest number of sum of common interests and common hobbies . If there is a tie , then it prioritises the profiles over age. For males , it gives priority to the girls of the same age first and then the girls of younger age and then the girls of older age . For Females , it gives priority to the boys of same age and then the boys of older age and then the boys of smaller age . For others , there is no specific restriction of gender . Even then ,if the tie exists then the algorithm prioritises Hobbies over interests . After all these , if the tie still exists then it shows the error message `"No good match is found"` . And the new match is displayed in a new tab loaded in `match.html` .



Name: Neha Reddy
Roll Number: 2023006
Year of Study: 4th
Age: 22
Email: neha@example.com
Interests: Art, Music
Hobbies: Reading, Painting

The essential part of code for displaying the right match is shown below

```
document.addEventListener('DOMContentLoaded', function () {
 // Get the form and output elements
  const form = document.getElementById('detailsForm');
const outputElement = document.getElementById('output');

form.addEventListener('submit', function (event) {
event.preventDefault(); // Prevent form submission

    const formData = new FormData(form);
 // Fetch the students.json file
  fetch('./students.json')
```

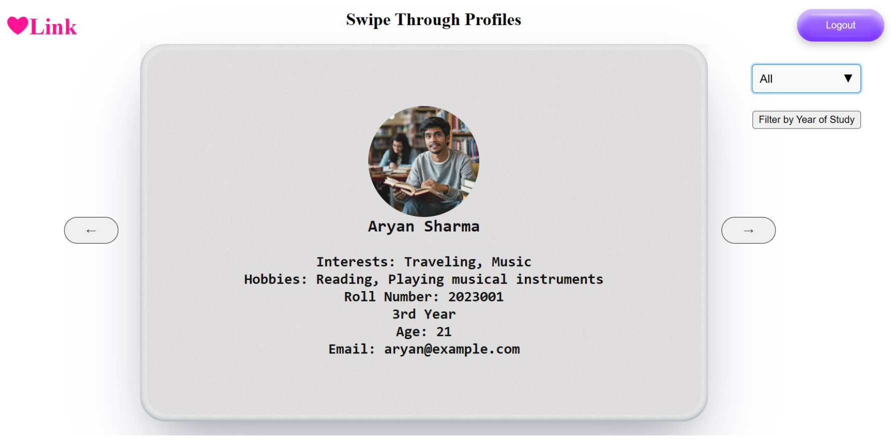
```
.then(response => {
if (!response.ok) {
throw new Error('Error fetching data.');
}
return response.json();
})
.then(data => {
 // Get the roll number of the logged-in user (from the form)
  const loggedInRollNumber = rollNumber;
const potentialMatches = filterPotentialMatches(data, gender,loggedInRollNumber);
calculateCommonInterestsAndHobbies(potentialMatches, interests, hobbies);

 // Sort matches based on the sum of common interests and hobbies
  // Then consider age relative to the logged-in user's age
  potentialMatches.sort((a, b) => {
const totalA = a.commonInterestsCount + a.commonHobbiesCount;
const totalB = b.commonInterestsCount + b.commonHobbiesCount;

 // Compare the sum of common interests and hobbies
  if (totalA !== totalB) {
return totalB - totalA;
} else {
let priorityA, priorityB;
if (gender === 'Male') {
 // Male user dating:  prioritize same age females, then younger females,
then older females
  priorityA = agePriority(a.Age, age, 'female');
priorityB = agePriority(b.Age, age, 'female');
} else if (gender === 'Female') {
 // Female user dating:  prioritize same age males, then older males,
then younger males
  priorityA = agePriority(a.Age, age, 'male');
priorityB = agePriority(b.Age, age, 'male');
}
else if(gender === 'Other') {
 // Other gender user dating:  prioritize same age others, then other
ages, and then males/females
  priorityA = agePriority(a.Age, age, a.Gender);
priorityB = agePriority(b.Age, age, b.Gender);
}
if (priorityA !== priorityB) {
return priorityA - priorityB;
} else {
 // Prioritize hobbies over interests in case of a tie
  if (b.commonHobbiesCount !== a.commonHobbiesCount) {
return b.commonHobbiesCount - a.commonHobbiesCount;
} else {
return b.commonInterestsCount - a.commonInterestsCount;
} } } }

// Display the best match
displayBestMatch(potentialMatches[0]);
})

 // Exclude the logged-in user from potential matches based on roll
number
 potentialMatches = potentialMatches.filter(person => person['IITB
Roll Number'] !== loggedInRollNumber);

return potentialMatches;
}

    // Save the best match's details to localStorage
 localStorage.setItem('bestMatch', JSON.stringify({
Name:  bestMatch.Name,
rollNumber:  bestMatch['IITB Roll Number'],
yearOfStudy:  bestMatch['Year of Study'],
age:  bestMatch.Age,
email:  bestMatch.Email,
Photo:  bestMatch['Photo'],
interests:  bestMatch.Interests.join(', '),
hobbies:  bestMatch.Hobbies.join(', ')
}));
```

2.4 Scroll or Swipe through Profiles

It shows the profiles that are fetched from **students.json** file . We can move through the profiles by clicking on next and previous buttons. This traverses the profiles in a circular manner having no start or end making the next and previous buttons more flexible .

The corresponding code for it is this

```
 // Add an event listener to the previous button
 document.getElementById('prevButton').addEventListener('click', function
() {
 // Decrement the current profile index
 currentProfileIndex--;
 // Check if the index is out of bounds and wrap around to the end
 if (currentProfileIndex < 0) {
currentProfileIndex = filteredProfiles.length - 1;
}
 // Display the previous profile
```



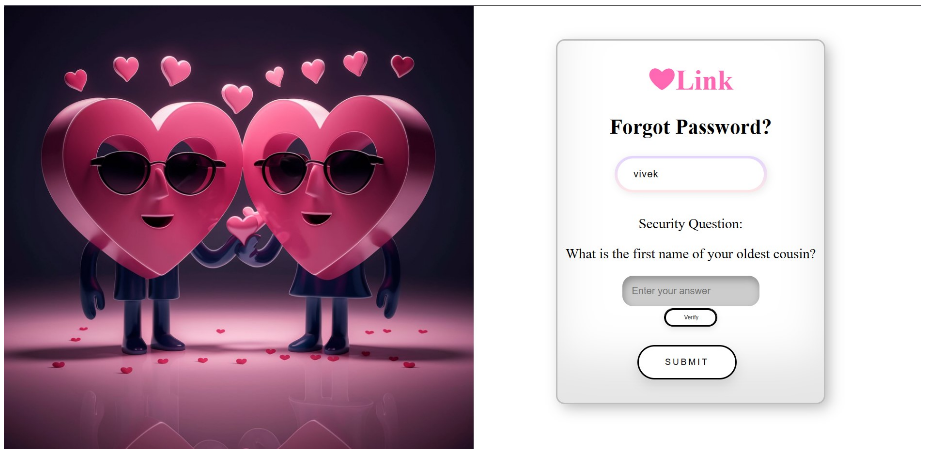
Aryan Sharma

Interests: Traveling, Music
Hobbies: Reading, Playing musical instruments
Roll Number: 2023001
3rd Year
Age: 21
Email: aryan@example.com

```
 displayProfile(currentProfileIndex);
});
 // Add an event listener to the next button
 document.getElementById('nextButton').addEventListener('click', function
() {
 // Increment the current profile index
 currentProfileIndex++;
 // Check if the index is out of bounds and wrap around to the start
 if (currentProfileIndex >= filteredProfiles.length) {
currentProfileIndex = 0;
}
 // Display the next profile
 displayProfile(currentProfileIndex);
});
});
```

2.5 Forgot Password?

When you click on `Forgot Password?` in `login.html` , it moves to `forgot.html` . On entering the username, if the username exists in `login.json` file then it asks the security question which is present in `login.json` file and if the correct security answer is entered then it shows the password of the corresponding user else corresponding error messages are shown. All the logic for checking username existence and display of security question is written in `forgot.js`.



Forgot Password?

vivek

Security Question:

What is the first name of your oldest cousin?

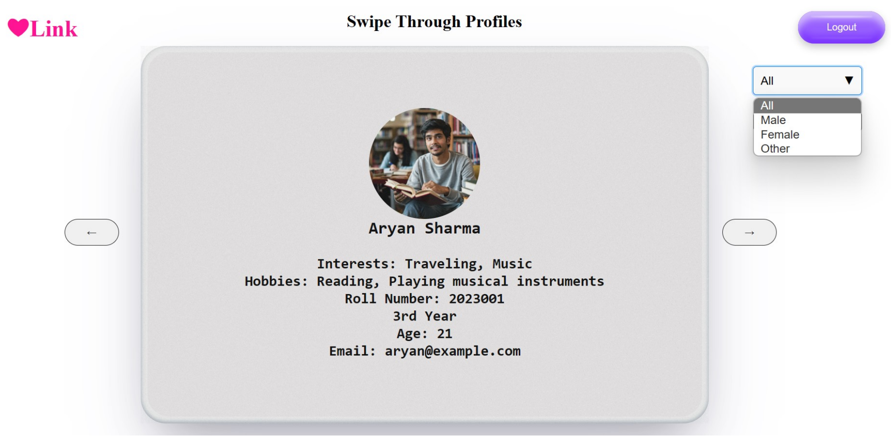
Figure 4: Forgot Password Interface

3 Customisations

3.1 Filters

In scroll_or_swipe.html , when clicked on the "All" button then it shows the dropdown box with options containing **"All"** , **"Male"** , **"Female" &** **"Others"** . When clicked on a specific gender then it shows only profiles of the selected gender profiles. This has been done through filtering profiles in students.json based on gender and creating a new data array to display . When clicked on the button **"Filter by year of study"**. It sorts the profiles based on their year of study in the ascending order. Initially it sorts on the ascending order and when clicked again it shows the profiles based on their year of study in the descending order . This can sort the profiles of specific gender based on their year of study too.

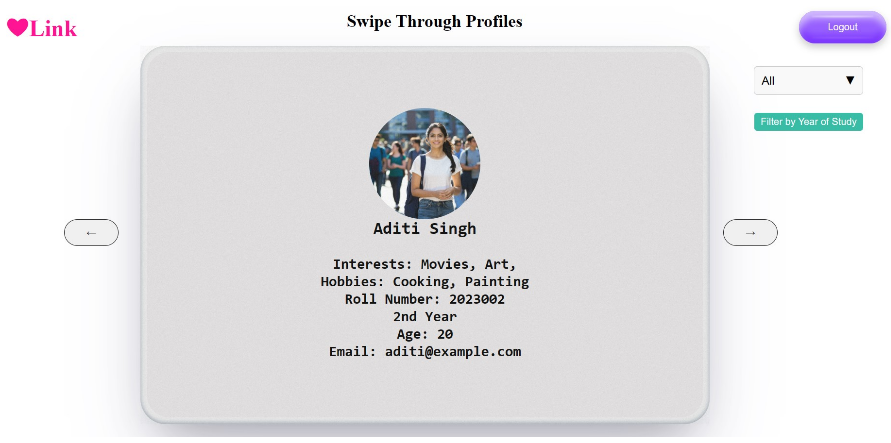
```
The code related to it is below
const genderFilter = document.getElementById("genderFilter");
genderFilter.addEventListener("change", function () {
 // Get the selected gender from the dropdown
  const selectedGender = genderFilter.value.toLowerCase();
 // Filter the profiles based on the selected gender
  if (selectedGender!== "all") {
filteredProfiles = profilesData.filter(profile => profile.Gender.toLowerCase()
=== selectedGender);
} else {
filteredProfiles = profilesData;
}
});
```



Aryan Sharma

Interests: Traveling, Music
Hobbies: Reading, Playing musical instruments
Roll Number: 2023001
3rd Year
Age: 21
Email: aryan@example.com

Figure 5: Scroll or swipe - show specific gender profiles



Aditi Singh

Interests: Movies, Art,
Hobbies: Cooking, Painting
Roll Number: 2023002
2nd Year
Age: 20
Email: aditi@example.com

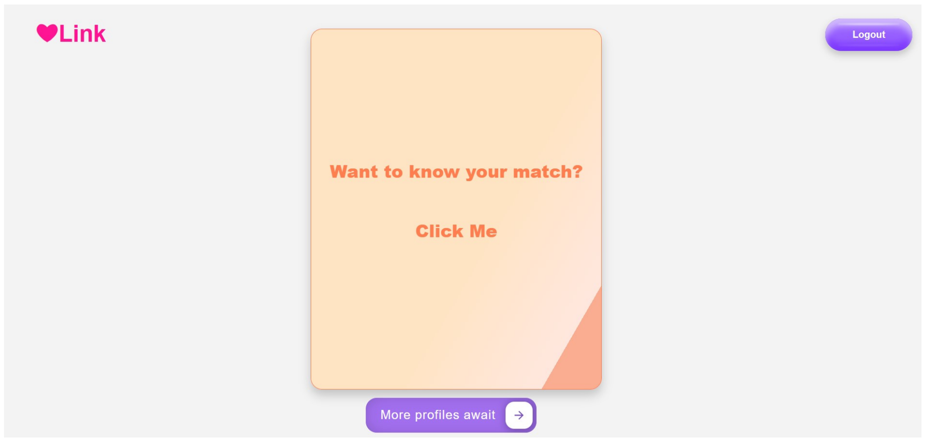
Figure 6: Scroll or swipe - filter based on year of study

```
// Working of filter by year of study button
 let ascendingOrder = true;
document.getElementById('filterByYearButton').addEventListener('click',
function () {
// Toggle the sorting order
if (ascendingOrder) {
 // Sort the profiles in ascending order by year of study
  filteredProfiles.sort((a, b) => parseInt(a['Year of Study']) - parseInt(b['Year
of Study']));
} else {
 // Sort the profiles in descending order by year of study
  filteredProfiles.sort((a, b) => parseInt(b['Year of Study']) - parseInt(a['Year
of Study']));
}
 // Change the sorting order when clicked again
  ascendingOrder =!ascendingOrder;
// Reset the current profile index
currentProfileIndex = 0;
});});
```

3.2 Making UI better and more interactive

The login box which appears in the `login.html` file is blurred on a background symbolising love between a couple. In the login page, icons representing username and password are added to add more beauty to the page. A lot of animated buttons and inputs are put in the website pages . A lot of CSS properties and CSS animations are used to make these buttons. To create certain animations , SVGs have been used , as creating them with naive HTML,CSS AND JS can be very time-taking and not producing very great results. For example , Whenever the match is shown , it shows first on a card to click on it , when clicked on it , it shows the match with a twinkling sound . Additionally for page navigation there are buttons to navigate through pages . In the forgot password page on clicking the **LoveLink** logo it redirects to `login.html`. On `dating.html` , On clicking the logout button it returns to the login page and on clicking the "More profiles await" button it leads to the page of `scroll_or_swipe.html` . On clicking logo of LoveLink then it redirects to `dating.html` and onclicking Logout button it redirects to `login.html`. "**Logout**" button is provided in most pages which redirects `login.html`.

```
<button class="click-me" id="clickButton">Click me
</button>

<audio id="clickSound">
  <source src="audios/click-me-twinkling-sound.mp3"
  type="audio/mpeg">
  Your browser does not support the audio tag.
</audio>

<script>
document.getElementById('clickButton').addEventListener(
'click', function()
{
  var clickSound = document.getElementById('clickSound');
  clickSound.play();
});
</script>
```

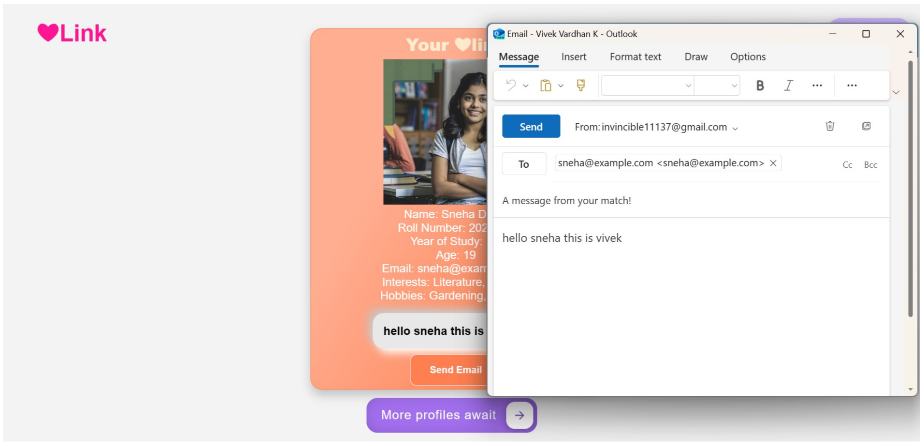


3.3 Mailing the right match

When you click on **Send Email** in the match.html , it opens a mail window to mail our best match to their Mail ID . This first retrieves the Email of the best match from '**bestMatch**' in local storage . This '**bestMatch**' is saved into local storage by dating.js through the dating algorithm. Then it encodes the message and body to the mail Link through **encodeURIComponent** and when the script is run , the mail Link opens the default mail app to send the best match an Email.

Code related to mailing the best match is shown below

```
document.getElementById('sendEmailBtn').addEventListener('click', function()
{
 // Retrieve the email of the best match from localStorage
  const bestMatchEmail = JSON.parse(localStorage.getItem('bestMatch')).email;
 // Retrieve the message input
  const message = document.getElementById('emailMessage').value;
```



Name: Sneha D
Roll Number: 202
Year of Study:
Age: 19
Email: sneha@exam
Interests: Literature,
Hobbies: Gardening,

hello sneha this is

From: invincible11137@gmail.com

To: sneha@example.com <sneha@example.com>

A message from your match!

hello sneha this is vivek

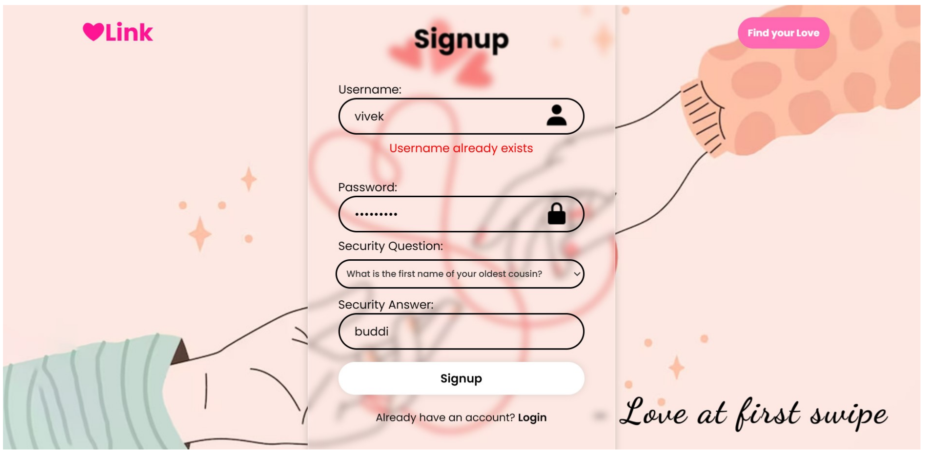
Figure 7: Mailing the right match

```
 // Check if email and message are not empty
  if (bestMatchEmail && message) {
 // Construct the mailto link
  const subject = encodeURIComponent("A message from your match!");
const body = encodeURIComponent(message);
const mailtoLink = ‘mailto:$bestMatchEmail?subject=$subject&body=$body‘;

 // Open the mail client
  window.location.href = mailtoLink;
} else {
alert('Please enter a message before sending.');
} }
```

3.4 Signup Integration with NodeJS

NodeJS has better capabilities over Javascript to deal with JSON files . When you click on Signup in `login.html` , it receives the details of the user and checks if the username already exists or not . If it does , it does not append it to `login.json` file but if a new username is being signed up , it appends to `login.json` details through the method **UsersData.push(newUser)** and then after successful signup it redirects to `dating.html` file. The signup logic is specifically handled by NodeJs and the login logic is alone handled by `login.js` file.
The file `app.js` handles the signup logic . Some part of `app.js` is provided below

```
// Handle signup form submission
 app.post('/signup', (req, res) => {
```


Signup

Username:

vivek

Username already exists

Password:

•••••••••

Security Question:

What is the first name of your oldest cousin?

Security Answer:

buddi

Figure 8: Signup Page

```
const { username, password, securityQuestion, securityAnswer } = req.body;
console.log('Received form data: ', { username, password, securityQuestion,
securityAnswer });
 // Read the existing users data from login.json
 fs.readFile(path.join(__dirname, 'public', 'login.json'), 'utf8',
(err, data) => {
if (err) {
console.error('Error reading login.json:', err);
return res.status(500).json({ error: 'Internal server error' }); }
let usersData = [];
if (data) {
try {
usersData = JSON.parse(data);
} catch (parseErr) {
console.error('Error parsing login.json:', parseErr);
return res.status(500).json({ error: 'Internal server error' }); }
 // Check if the username already exists
 const existingUser = usersData.find(user => user.username === username);
if (existingUser) {
 // Username already exists, return an error message
 return res.status(400).json({ error: 'Username already exists' });
}
 // Create a new user object
 const newUser = {
username,
password,
secret_question: securityQuestion,
secret_answer: securityAnswer,
};
 // Append the new user to the existing users data
  usersData.push(newUser);

 // Save the updated users data to login.json
  fs.writeFile(path.join(__dirname, 'public', 'login.json'), JSON.stringify(usersData,
null, 2), (err) => {
if (err) {
console.error('Error writing login.json:', err);
return res.status(500).json({ error:  'Internal server error' });
}

 // Redirect to dating.html after successful signup
  res.status(200).json({ message:  'Signup successful' }); }); });
});
// Check for if username already exists
 app.post('/checkUsername', (req, res) => {
const { username } = req.body;
fs.readFile(path.join(__dirname, 'public', 'login.json'), 'utf8', (err,
data) => {
if (err) {
console.error('Error reading login.json:', err);
return res.status(500).json({ error:  'Internal server error' });
}
let usersData = [];
if (data) {
try {
usersData = JSON.parse(data);
} catch (parseErr) {
console.error('Error parsing login.json:', parseErr);
return res.status(500).json({ error:  'Internal server error' }); }
 // Check if the username already exists
  const existingUser = usersData.find(user => user.username === username);
if (existingUser) {
 // Username already exists, return an error message
  return res.status(400).json({ error:  'Username already exists' });
}

 // Username does not exist, proceed with signup
  return res.status(200).json({ message:  'Username available' });
});});});
```

4 How to launch my website?

- Make sure that you have node installed in your machine .
- Open a terminal & navigate to the directory where `app.js` has been unzipped .
- Type "`node app.js`" in the terminal and the website starts to run . Now open a browser and load this link `http://localhost:3000` .
- And that's it ! The website starts to run .
- Make sure that no file paths and files are changed to ensure complete loading of each webpage.

5 Conclusion

In conclusion, the development of LoveLink, an innovative dating website project, has been a wonderful journey with some thrilling experiences. Through the integration of HTML, CSS, and JavaScript, LoveLink creates meaningful connections among users based on shared interests. It integrates matching algorithms to facilitate compatibility based on various criteria such as interests, hobbies, and age. With its intuitive interface , users can effortlessly navigate through profiles and ultimately find companionship that aligns with their unique preferences and values.This project personally taught me how to solve the bugs related to JS and developed my imaginative and creative skills. By combining these elements, the website streamlines the process of finding love and enhances the overall experience for users, making it an invaluable tool in the quest for genuine connections in today's digital landscape.