**Team Members:**

Angela Steinmetz, Anahita Ahmadi, Aayushi Agrawal, Siddharth Bansal, Sanjana Ilango, Anisha Reddy, Logan Norman, Ramneek Singh

Software Requirements

We will be utilizing HTML, CSS, JavaScript (React JS), Java, Spring Framework

Hardware Requirements

There are no unique hardware requirements, other than having access to a laptop in order to get to the webpage. For the backend, a server will be needed with at least 4 GiB of Ram, a CPU, and storage to power a DBMS.

Programming Language:

We will mostly be using HTML and CSS in order to style and create our pages. For functionality, some javascript will be used in order to ensure button presses actually work. In addition, in order to make component creation easier, we will be using React JS.

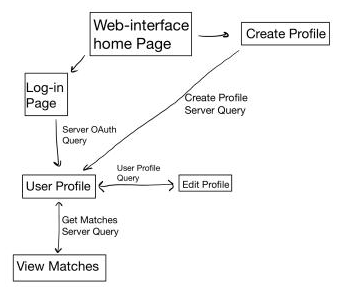
The server side of the web interface will make use of the SQL and a DBMS system to effectively manage user data. Additionally, the server will use a RESTFUL API powered by the Spring framework and Maven to manage the build. Alongside the major frameworks, the server will make use of the Java Gson library to manage converting and manipulating Json data within the server. The server will use Java exclusively, and SQL script for managing database queries.

Inheritance will be used on the server side to properly return Json formatted data. This will allow easy facilitation of handling data and efficient return of data to the web interface.

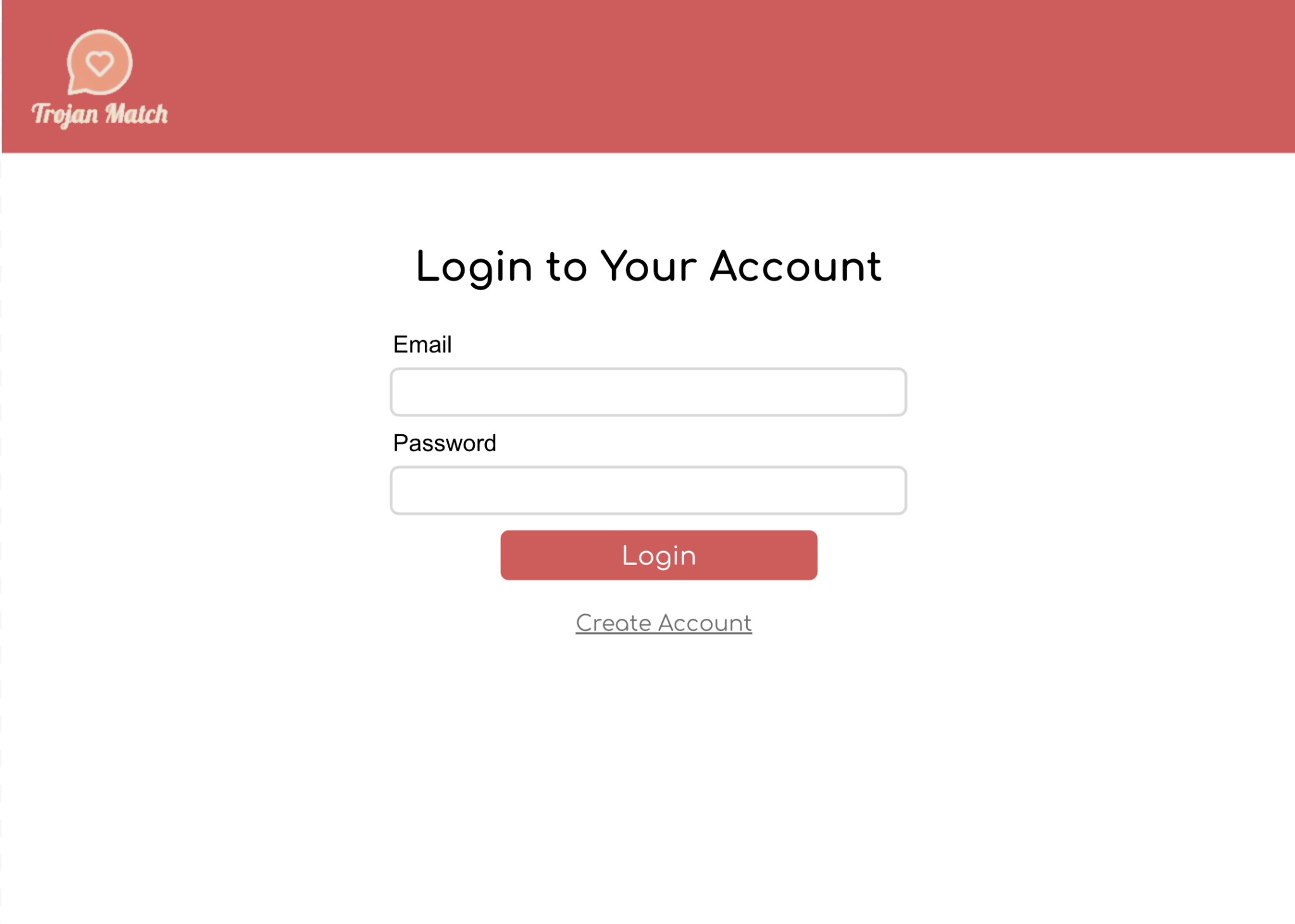
Ultimately, the server will be deployed to Google Cloud Computing services and have a URL for easy access to the website.

Web-Interface

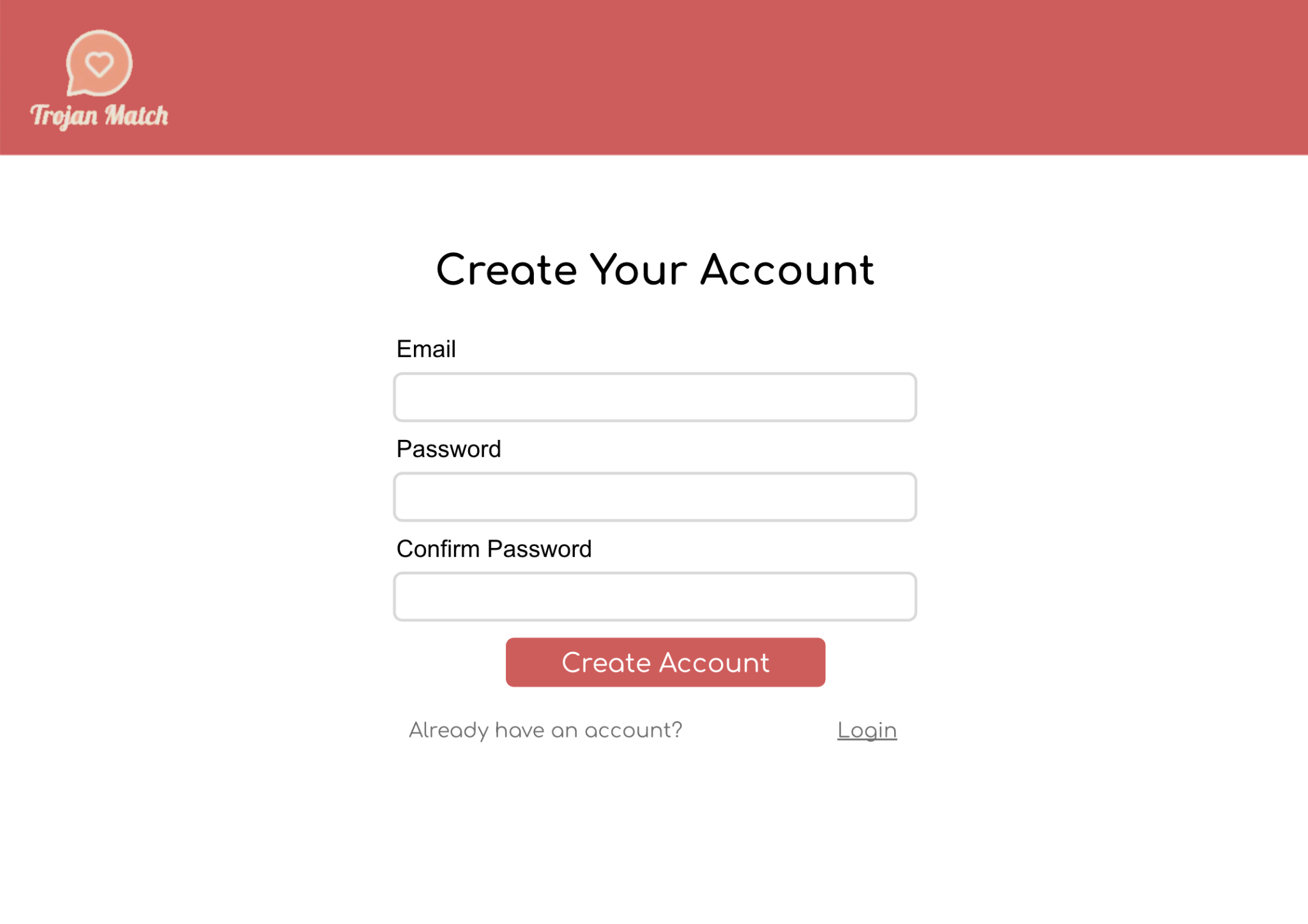
Running-States Diagram



Designs:

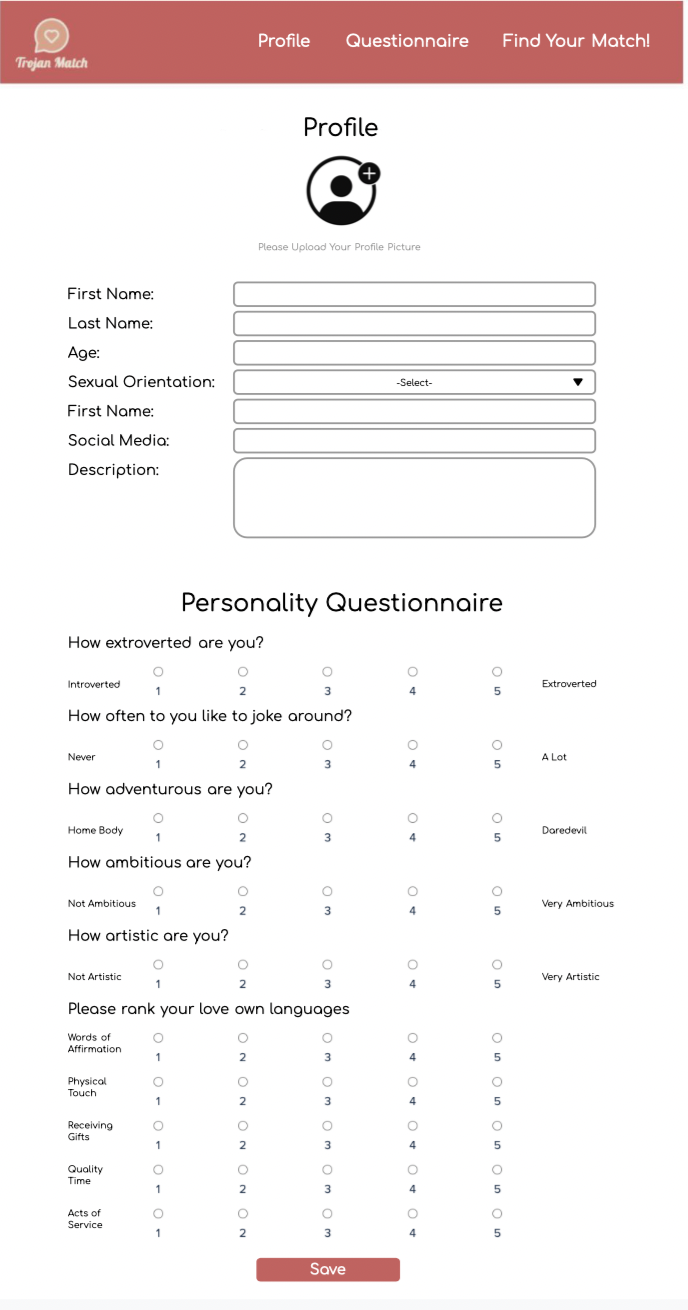


The screen above is the log-in page, where users can put in a correct email and password combination, click the log-in button, and log-in. If the email or password do not match, a warning will pop up noting that. There is also a link to another web page, which is the create account page that is shown below:



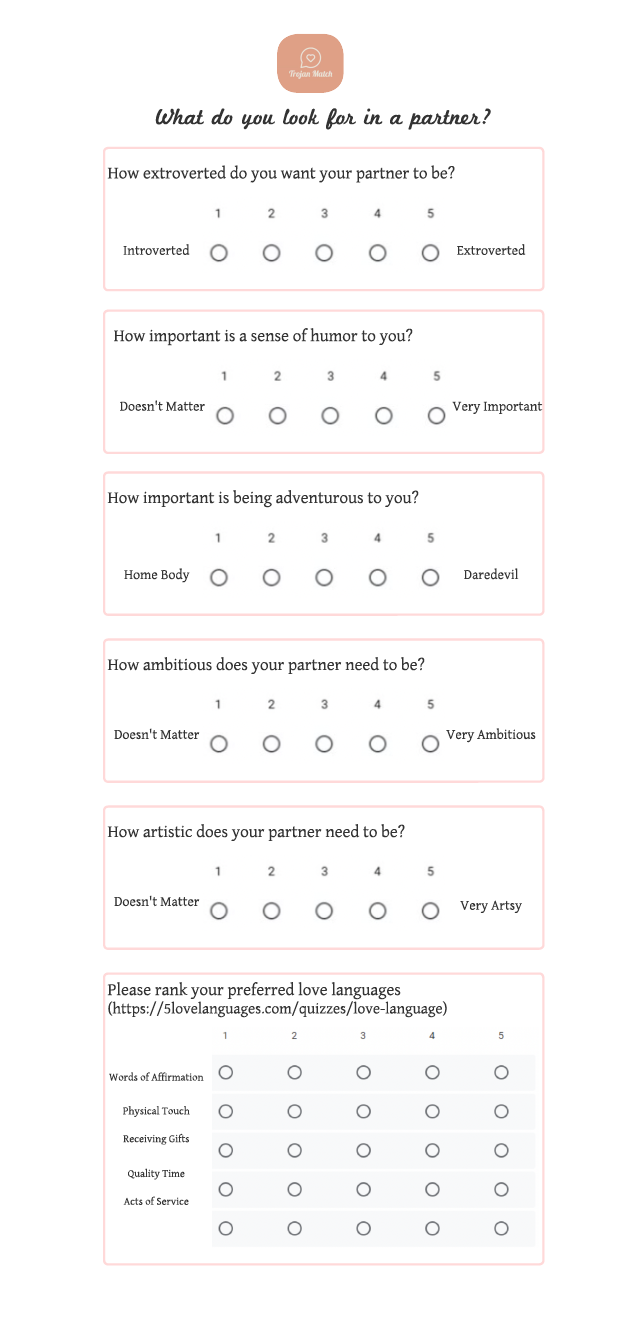
This page will be used for new users to create an account. This page has new users put in their email, and there are two password input fields in order to ensure that users put in the password they actually want. Clicking the create account button will log them in and store this information for them. For users that already have an account, they can click the “login” button on the bottom to bring them back to the login page.

Once users have logged in, they will be brought to their profile page where they can create their profile:



There are two main sections to the profile page. The first one is a place for users to input their name, age, sexual orientation, social media, and any other descfiption they want into text fields. The second part correlates to the form that will be used to match users. Each user will first fill out their own form to detail their personality traits through a series of questions. Pressing the save button will save any changes or edits they make. A menu will be at the top which will allow users to switch back and forth to other pages. This will include the ability to go to the match page, questionnaire page, and the profile page (in that order).

The next page is the questionnaire page, which asks users a series of questions (that are the same as their own personal questions) to see what type of person they want their significant other to be. A design of that is below:



Similar to the questions in the profile page, these questions will ask about certain types of personality traits, and will have users rank them from 1-5 depending on importance.

The last page is the match screen, where users will get updated on their match every day. This match will change everyday while users don’t mutually match with each other. If the matched couple mutually match with each other, they will be “removed” from the matching algorithm until they opt back in. A mockup of these pages are below:

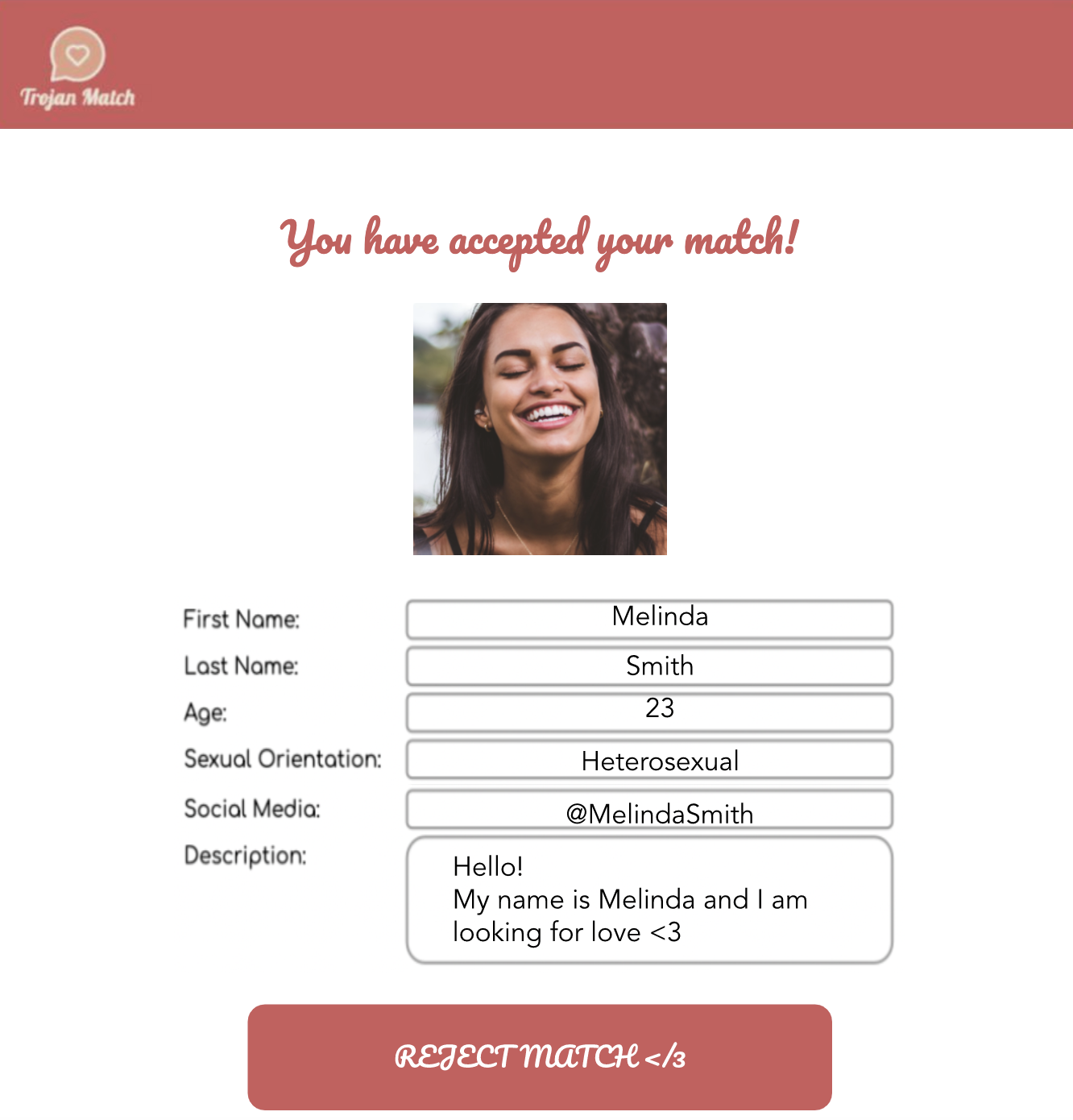


The daily match screen, which will show a new partner everyday and parts of their profile. A button on the bottom will allow the user to match with their potential partner.



If there are no matches found, or the user created a profile after the matches have been outputted for the day, this screen will be shown to the user so they know they don’t currently have a match

Finally, if both people in the match mutually match with each other, then the following screen will be shown:



Here, if users end up deciding to reject the match after talking to them, they will be able to do so by clicking the reject match button on the bottom of the screen.

ServerSide:

General Json Format:

For the purpose of communication, the RestAPI server will respond to various HTTP requests with Json formatted strings. The Json returned will have various fields which the web-interface can decode to provide the correct display of data. The Json data returned will vary based on which API endpoint the request was made. However, *each* json response will contain a “status” field which will contain a status code. Status code 200 will indicate that the query was made successfully. Status code 404 will indicate that the query was not able to find specific data. Status code 417 indicates the failure of the HTTP request.

EndPoints:

The web-interface of our application will communicate with our RestAPI server in order to get/add data of our users. The Web-Interface will communicate via endpoints. The following endpoints will be used, as well as provide their respective functionality:

Authenticate a userLogin:

/OAuth

Functionality

This endpoint will be queried to verify whether or not a user exists in our system. Given that the user exists in our system, this request will return a Json formatted string containing the user’s profile data as well as status code 200. In the situation that the user does not exist or the user was not found, the endpoint will respond with a json containing status code 404. The Json will additionally have a user token.

Query

To make the query, the API requires the user name and password of the user in the header of the API request. The header will be stripped from the request and verified using our OAuth() function in our Authorization class.

Get Profile

/user/profile

Functionality

This endpoint will be queried to obtain the profile information of a certain user. Given that the user exists in our system, this request will return a Json formatted string containing the profile information of a certain user. Returns status code 200 for success, code 404 for user not found, or code 417 if the request was made without being logged in to an account.

Query

To make the query, the API requires the user token to be in the header of the API request, as well as the user name of the profile the request aim’s to retrieve. Serverside, the query will be passed off to the getProfile() function in the Profile class.

Create a new profile

/user/createProfile

Functionality

This endpoint will be queried to create a user profile. Within the header of this function call, the query will be made with all the necessary information regarding a new profile: First name, last name, age, sexual orientation, social media information, and a quick description of themeselves. Returns status code 200 for success or code 417 if the request was not able to make a profile. The json returned will contain the information of the profile of the person

Query

The query will need to have the headers of the first name, last name, age, sexual orientation, social media information, and a quick description. On the server side, the query will be passed off to createProfile() in the Profile class.

Matches

/user/matches

Functionality

This endpoint will serve as a way for people to view their current matches. The response from the server will include the profile data of all of the people the current user was matched with. The user needs to be logged in to be able to make this request.

Query

The query will need to have the user’s token in the header. This query will be passed off to our getMatches() function in our Matches class. Returns status code 200 for success or code 417 if the request failed.

Database:

The database will function independently from the rest of the code. We will create a seperate class, DataStore, that will take care of storing/retrieving/updating all of the necessary data from the DBMS using SQL. The DataStore class will have 2 functions: getQuery() and postQuery(). In the postQuery function, the input will be a user profile. The postQuery() will take this data and update the database. The getQuery() will take a string as a parameter which will be the name of the person, and that will cause the getQuery() to make a query to the database. The response of the getQuery() will be in the form of a custom object. The object will have a Map data structure which will map each of the names to their profiles. This will ensure fast access to each of the returned profiles.

Algorithms

Our match making algorithm will utilize various metrics to ensure a proper match between people. First, the algorithm will prioritize people that have listed each other in the list of their preferences. Considering each person’s preference list will give a greater chance of being picked to be together at the end of the algorithm. Using the scores that people gave themselves and their preferences, we will determine a heuristic score for each possible match that they could have. This heuristic will be based on how different their preferences are from each other, and will result in a score on a scale of 1-10, where 10 denotes most compatible. Then, these scores, along with which partner they were made up with, will be thrown into a max-heap. Once all of the potential matches are placed in the heap, the algorithm will continue to pop from the top and check whether or not either of the partners have been matched. Given that neither have been matched, we will match that couple and mark both partners as taken.

Given that we will be considering every possible match in this scenario, we are guaranteed that every person will be matched with someone in the event that there are an equal number of men and women. Realistically, there will likely not be an equal number of both, which will result in a few people from a party without a match.

Finally, for the purpose of including people of all sexual orientations, this algorithm will run once for each group. This will allow appropriate matches for all sexual orientation groups.

Proposed Pseudocode for Algorithm

Max-Heap matches;

For each sexual orientatoin:

For each person, j,

For each in person, i

X = heuristic (j, i);

Push x into matches

List official matches

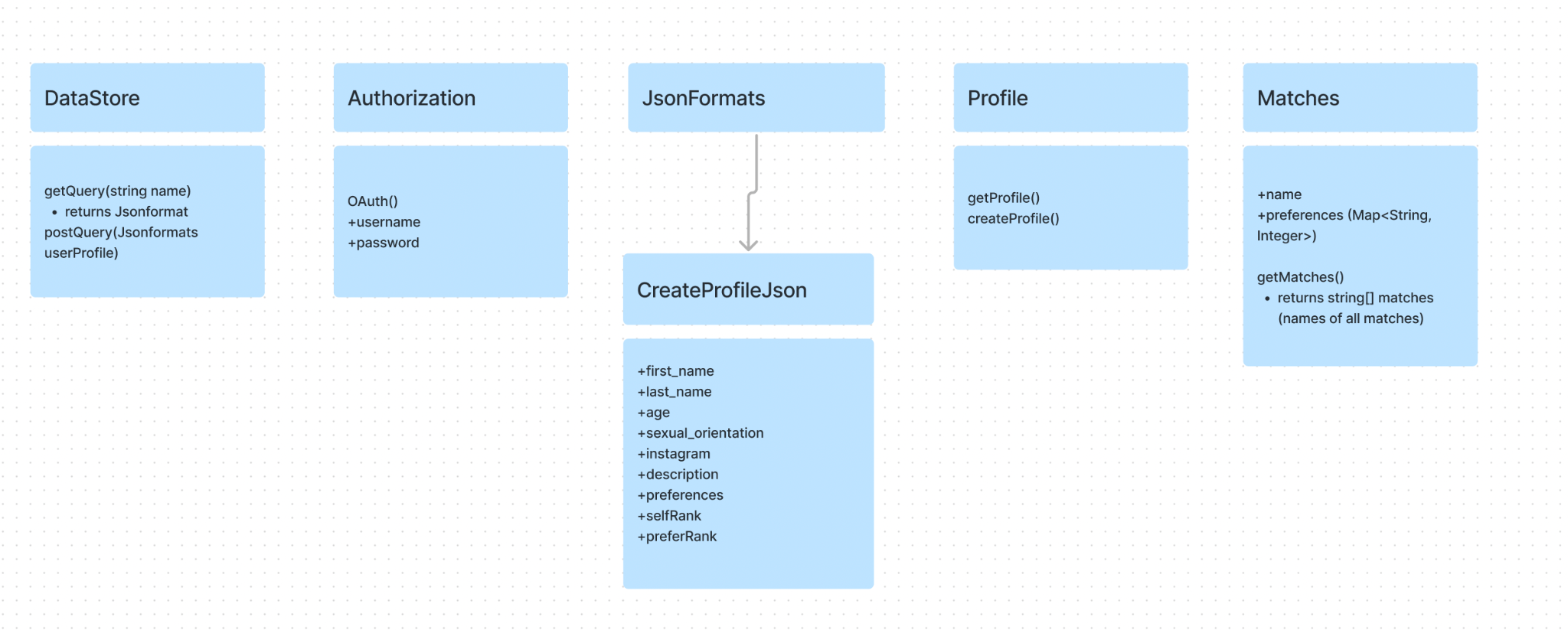
While matches is not empty:

X = matches.pop();

If neither partner from x has been matched

Make x an official couple and add it to oficial matches

Inheritance Diagram



Method and Variable Descriptions

getQuery

This method will be responsible for retrieving the userdata after their login has been authenticated. The arguments that will be passed in will be the name of the user to get a json format file of all their data.

postQuery

This method will be responsible for displaying the user profile information. The method will take in the json format file created by the previous method and then parses it for it to then be displayed for the user

OAuth

This method will take in a user’s username and password to authenticate the login. The username and password will be strings and the method will then verify if the login information is correct for the user or not. If it is inaccurate, the method will then display an error message saying that the user information is incorrect.

CreateProfile

This method will be invoked to create the user’s profile. The method will take in their first and last name, their sexual orientation, a description about themselves, their instagram handle and their preferences. Their age, self ranking and prefered ranking will all be ints. Their preferences will be a map of strings to numbers. The highest number will be their highest preferred person. create profile will output a json file which will be created by the CreateProfileJson method.

GetProfile

This method will be responsible for retrieving the profile information for the user after their login information has been authenticated.

GetMatches

This method is responsible for retrieving the matches of the user. This method will output all the names of the matches that the user is matched with.