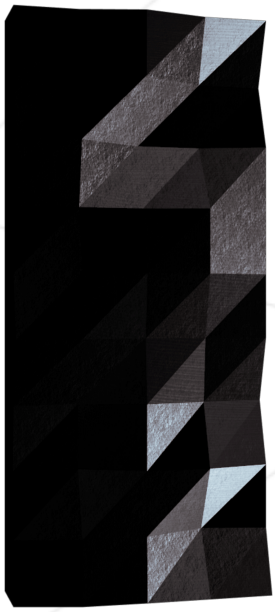

Web

Matcha project

Summary: Because, love too can be industrialized.

Version: 4.2

Contents

Chapter I

Foreword

This second millennium has forever changed and upheld the habits and manners of Internet. The choice is guided by technologies, and room for luck is increasingly smaller. Human relationships, seed of any modern society are increasingly created artificially by meeting sites algorithms and social networks, between people that match to some very precise criteria.

Yes, romanticism is dead, and Victor Hugo is probably spinning in his grave.

Chapter II

Introduction

This project aims to create a dating website.

```
You will need to create an application that allows two potential
lovers to meet, from the registration process to the final encounter.
```

Users will be able to register, log in, complete their profile, search and view the profiles of other users, and show interest in them with a "like"[1], chat with those that "liked" back.

[1] "Like" being a very bad word, you're invited to find a more explicite word for that action.

Chapter III

General Instructions

- Your application must not have any errors, warning or notice, either server-side or client-side.
- For this project, you are free to use any programming language of your choice.
- You are free to use micro-frameworks and any libraries you need for this project
- You are free to use UI libraries such as React, Angular, Vue, Bootstrap, Semantic, or any combination of them.
- No security breaches are allowed. You must handle at least what's in the mandatory part, but we encourage you to go even further. Everything depends on it.
- We will consider that a "micro-framework" includes a router and possibly templating, but does not include an ORM, validators, or a User Account Manager.[1]. As long as you respect these constraints you are free to use what you like.
- If you need some inspiration, we suggest using the following languages as the main ones:
  - Sinatra for Ruby.
  - Express for Node (yes, we consider this to be micro-framwork).
  - Flask for Python.
  - Scalatra for Scala.
  - Slim for PHP (Silex is not authorized because of doctrine integration).
  - Nickel for Rust.
  - Goji for Golang.
  - Spark for Java.
  - Crow for C++.
- You should use a relational or graph-oriented database. The database should be a free one such as MySQL, MariaDB, PostgreSQL, Cassandra, InfluxDB, Neo4j, etc. You will also need to create your queries manually, like mature developers do. However, if you're clever, you can create your own library to simplify your queries.

[1] This definition will be authoritative during defence even if you can find a different one on the Internet.

- You are free to choose the web server that best suits your needs, whether it is `Apache`, `Nginx` or a `built-in web server`.
- Your entire app must be compatible with at least the latest versions of `Firefox` and `Chrome`.
- Your website must have a decent layout: at least a header, a main section and a footer.
- Your website must be usable on a mobile phone and keep an acceptable layout on small resolutions.
- All your forms should have proper validation and the entire website must be secure. This is a mandatory part and will be extensively checked during the defense. To give you an idea, here are a few elements that are considered insecure:
  - Storing plain text passwords in your database.
  - Allowing injection of HTML or user Javascript code in unprotected variables.
  - Allowing upload of unwanted content.
  - Allowing alteration of SQL requests.

Chapter IV

Mandatory part

You will need to create a Web App with the following features:

IV.1 Registration and Signing-in

The app must allow a user to register by requesting at least their email address, username, last name, first name, and a password that is somehow protected. After registration, an email with a unique link must be sent to the registered user to verify their account.

The user must be able to login using their username and password, and also receive an email allowing them to reset their password if they forget it. Additionally, the user must be able to log out with just one click from any page on the site.

IV.2 User profile

- Once a user is connected, they must fill out their profile by providing the following information:
  - The gender.
  - Sexual preferences.
  - A biography.
  - A list of interests with tags (e.g. #vegan, #geek, #piercing, etc.), which must be reusable
  - Up to 5 pictures, including one to be used as a profile picture.
- At any time, the user must be able to modify this information, as well as their last name, first name, and email address.
- The user must be able to check who has viewed their profile,
- as well as who has "liked" them.
- The user must have a public "fame rating" [1].

[1] Up to you to define what "fame rating" means as long as your criteria are consistant.

- The user must be located using GPS positioning, up to their neighborhood. If the user does not want to be positioned, you must find a way to locate them even without their knowledge.[2] The user must be able to modify their GPS position in their profile.

IV.3 Browsing

The user must be able to easily get a list of suggestions that match their profile.

- You will only propose "interesting" profiles. For example, only men for a heterosexual girls. You must manage bisexuality.If the user's orientation isn't specified, they will be considered bisexual.
- You must cleverly match[3] based on:
  - Same geographic area as the user.
  - A maximum of common tags.
  - A maximum "fame rating".
- You must prioritize showing people from the same geographical area.
- The list must be sortable by age, location, "fame rating", and common tags.
- The list must be filterable by age, location, "fame rating", and common tags.

IV.4 Research

The user must be able to conduct an advanced search by selecting one or more criteria, such as:

- An age gap.
- A "fame rating" gap.
- A location.
- One or multiple interest tags.

For the suggested list, the resulting list must be sortable and filterable by age, location, "fame rating" and tags.

IV.5 Profile of other users

A user must be able to view the profiles of other users. Profiles must contain all available information about them, except for the email address and password.

When a user views a profile, it must be added to their visit history.

The user must also be able to:

[2]

- Yes that's what dating websites do...

[3] Weight at least several criterias.

- "Like" another user's profile picture. When two people "like" each other's profiles, they will be considered "connected" and can start chatting. If the current user does not have a profile picture, they cannot complete this action.
- You should also remove your "like" to an user whom you had previously "liked", The user will no longer generate notifications, and you will not be able to chat with them anymore.
- Check the "fame rating" of another user.
- See if a user is currently online, and if not, see the date and time of their last connection.
- Report a user as a "fake account".
- Block a user. A blocked user will no longer appear in the search results and will not generate additional notifications. And, of course, it will no longer be possible to chat with him.

A user can clearly see if the profile they are viewing is connected or has "like" their profile and must be able to "unlike" or disconnected from that profile.

IV.6 Chat

When two users are connected,[4] they must be able to "chat" in real-time.[5] The implementation of the chat is up to you. The user must be able to see from any page if a new message is received.

IV.7 Notifications

A user must be notified in real-time[6] of the following events:

- When the user receives a "like".
- When the user's profile has been viewed.
- When the user receives a message.
- When "liked" user also "likes" the user back.
- When a connected user "unlikes" the user.

A user must be able to see, from any page, that a notification hasn't been read.

`For obvious security reasons, any credentials, API keys, environment variables etc., must be stored locally in a .env file and excluded from git. Storing credentials publicly can result in project failure.`

[4] Meaning they have "liked" each other.
[5] with a maximum delay of 10 seconds.
[6] with a maximum delay of 10 seconds.

Chapter V

Bonus part

Possible bonuses that you can implement to get extra points.

- Add Omniauth strategies for user authentication.
- Allow importing pictures from social network(snapchat, facebook, Google+, etc.).
- Develop an interactive map of users, which requires more precise GPS localization via JavaScript.
- Integration of video or audio chat for connected users.
- Implementation of a feature to schedule and organize real-life dates or events for matched users.

The bonus part will only be assessed if the mandatory part is PERFECT. "Perfec" means that the mandatory part has been completely implemented and works without any malfunctions. If you have not met ALL of the mandatory requirements, your bonus part will not be evaluated.

Chapter VI

Submission and peer-evaluation

Turn in your assignment in your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Don't hesitate to double check the names of your folders and files to ensure they are correct.

VI.1 Peer-evaluation

- Your code cannot produce any errors, warnings or notices either from the server or the client side in the web console.
- Anything not specifically authorized is forbidden.
- Finally, the slightest security breach will give you 0. You must at least manage what is indicated in the general instructions, ie NOT have plain text passwords stored in your database, be protected against SQL injections, and have a validation of all the forms and upload.