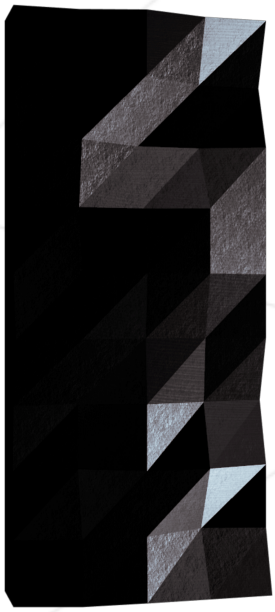

Web

Matcha project

Summary: Because, love too can be industrialized.

Version: 4.1

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 is about creating a dating website.

```
You will need to create an app allowing two potential lovers to meet,
from the registration to the final encounter.
```

A user will then be able to register, connect, fill his/her profile, search and look into the profile of other users, like them [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, nor server-side nor client-side.
- For this project you are free to use the language you want.
- Clientside, you must use HTML, CSS, and Javascript
- You can use micro-frameworks, and all the libraries in the world for this project.
- You are free to use UI librairies (React, Angular, Vue, Bootstrap, Semantic, all of them at once..)
- No security breaches allowed. You must at least handle what's in the mandatory part, but we invite you to go even further. Everything depends on it.
- We will consider that a "micro-framework" has a router, and eventually templating, but no ORM, validators or User Accounts Manager.[1]. As long as you respect these constraints you are free to use what you like.
- If you need some inspirations, we will suggest as main languages:
  - 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++.
- ou should use a relational or graph-oriented database. The database to use is free (MySQL, MariaDB, PostgreSQL, Cassandra, InfluxDB, Neo4j ...). You will also have to forge your requests by hand, like grown-ups. But if you are smart, you can make your own library to wrap your queries.

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

- You're free to use the web server you like most may it be `Apache`, `Nginx` or a `built-in web server`.
- Your whole app must be compatible at least with `Firefox` (>= 41) and `Chrome` (>= 46).
- 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 must have correct validations and the whole website must be secure. This part is mandatory and will be checked extensively in defense. To give you an idea, here are a few elements that are not considered secure:
  - To have a "plain text" password stored in your database.
  - To be able to inject HTML of "user" Javascript code in unprotected variables.
  - To be able to upload unwanted content.
  - To be able to alter a SQL request.

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 asking at least an email address, a username, a last name, a first name and a password that is somehow protected. After the registration, an e-mail with an unique link must be sent to the registered user to verify his account.

The user must then be able to connect with his/her username and password. He/She must be able to receive an email allowing him/her to re-initialize his/her password should the first one be forgotten and disconnect with 1 click from any pages on the site.

IV.2 User profile

- Once connected, a user must fill his or her profile, adding the following information:
  - The gender.
  - Sexual preferences.
  - A biography.
  - A list of interests with tags (ex: #vegan, #geek, #piercing etc...). These tags must be reusable.
  - Pictures, max 5, including 1 as profile picture.
- At any time, the user must be able to modify these information, as well as the last name, first name and email address.
- The user must be able to check who looked at his/her profile as well as who "liked" him/her.
- The user must have a public "fame rating" [1].
- The user must be located using GPS positionning, up to his/her neighborhood. If the user does not want to be positionned, you must find a way to locate him/her

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

even without his/her knowledge.[2] The user must be able to modify his/her GPS position in his/her profile.

IV.3 Browsing

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

- You will only propose "interesting" profiles for example, only men for a heterosexual girls. You must manage bisexuality. If the orientation isn't specified, the user will be considered bi-sexual.
- You must cleverly match[3] profiles:
  - Same geographic area as the user.
  - With a maximum of common tags.
  - With a maximum "fame rating".
- You must show in priority 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 run an advanced research selecting one or a few criterias such as:

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

As per the suggestion 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 consult the profile of other users. Profiles must contain all the information available about them, except for the email address and the password.

When a user consults a profile, it must appear in his/her visit history.

The user must also be able to:

- If he has at least one picture "like" another user. When two people "like" each other, we will say that they are "connected" and are now able to chat. If the current user does not have a picture, he/she cannot complete this action.

[2] Yes that's what dating website do...

[3] Weight at least on several criterias.

- Check the “fame rating”.
- See if the user is online, and if not see the date and time of the last connection.
- Report the user as a “fake account”.
- Block the user. A blocked user won’t appear anymore in the research results and won’t generate additional notifications.

A user can clearly see if the consulted profile is connected or “like” his/her profile and must be able to “unlike” or be disconnected from that profile.

IV.6 Chat

When two users are connected,[4] they must be able to “chat” in real time.[5] How you will implement the chat is totally 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:

- The user received a “like”.
- The user’s profile has been checked.
- The user received a message.
- A “liked” user “liked” back.
- A connected user “unliked” you.

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, env variables etc... must be saved locally in a .env file and ignored by git. Publicly stored credentials will lead you directly to a failure of the project.

[4] Meaning they “like” each other.
[5] We’ll tolerate a 10 secondes delay.
[6] We’ll also tolerate a 10 secondes delay.

Chapter V

Bonus part

Here are some ideas:

- Add Omniauth strategies for the connection.
- Import pictures from Facebook and/or Google+.
- Create an interactive map of the users (which implies a more precise GPS localization via JavaScript).

The bonus part will only be assessed if the mandatory part is PERFECT. Perfect means the mandatory part has been integrally done and works without malfunctioning. If you have not passed ALL the mandatory requirements, your bonus part will not be evaluated at all.

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.