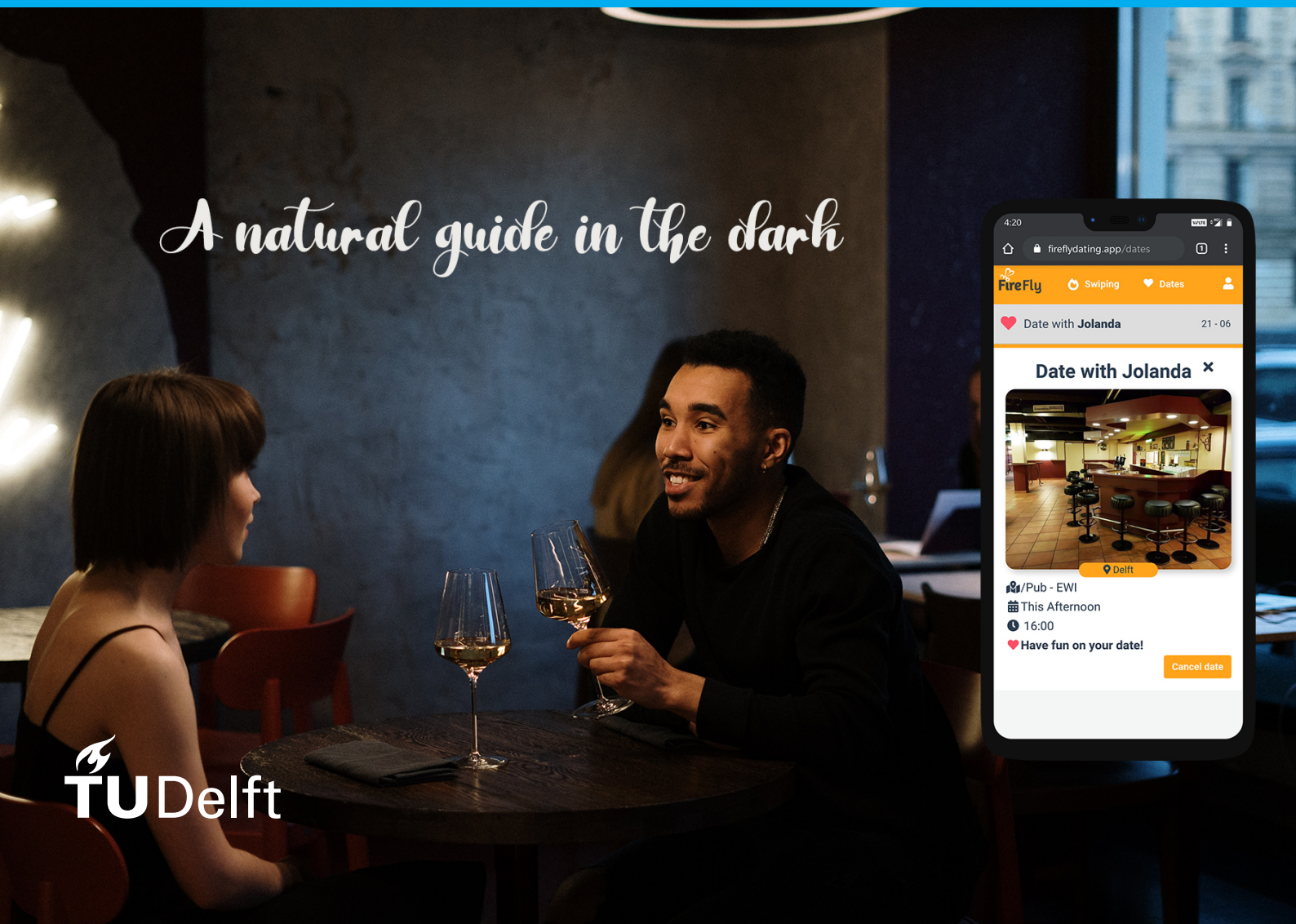
Bachelor End Project: FireFly Dating

Bachelor Thesis

Colin Geukes
Caspar Krijgsman
Steven Lambregts
Vincent Wijdeveld
Matthijs Wisboom



Bachelor End Project: FireFly Dating

Bachelor Thesis

by

Colin Geukes
Caspar Krijgsman
Steven Lambregts
Vincent Wijdeveld
Matthijs Wisboom

Project duration:	April 20, 2020 – July 3, 2020	
Thesis committee:	Dr. H. Wang,	TU Delft, Bachelor Project Coordinator
	Ir. T.V. Aerts,	TU Delft, Coach
	M.S. Salarbux,	Firefly, Client

An electronic version of this thesis is available at http://repository.tudelft.nl/.

Some content of this thesis is confidential and is therefore marked in black.

TU Delft

Preface

This report is for the Bachelor End Project (BEP). It is created by Colin Geukes, Caspar Krijgsman, Steven Lambregts, Vincent Wijdeveld, Matthijs Wisboom as a part of the Bachelor Computer Science program at the Delft University of Technology.

Over the course of ten weeks, we have built a new dating application where users can swipe and go on a blind date via the application. This project was commissioned by FireFly.

Our sincerest thanks to everyone at FireFly for their support during the project. A special thanks to Siraadj Salarbux, owner and contact person of FireFly, for taking the time to coordinate us.

Finally, we would like to thank our coach, Ir. Taico Aerts software developer and teacher at the Delft University of Technology, for always finding time in his busy schedule to guide us and provide criticism where it was needed.

Colin Geukes
Caspar Krijgsman
Steven Lambregts
Vincent Wijdeveld
Matthijs Wisboom
Delft, June 2020

Executive Summary

The FireFly company was created on the idea of blind dating. Most dating apps need the user to have an extensive conversation through a chat box before settling for a date. The product of FireFly tries to remedy that by having the first conversation be face to face. Over ten weeks, we have built the FireFly Dating application.

We started researching other dating applications and matching algorithms to decide the requirements for FireFly Dating. After this, we implemented these requirements from scratch.

In the final product, users can register an account, like or dislike other users profiles to indicate preferences for the matching algorithm, enroll for a blind date, and get matched. The application also includes an automated emailing system and a full reporting and feedback system. Furthermore, the system allows for an Administrator to get an overview of the application's data. The Administrator can insert, edit or archive Dating Establishments and Time Slots. As well as view user reports and ban users if necessary. The final application is a product that portrays the Product Owner's vision in that dating should be easily accessible and offline. At the end of the project, together with the Product Owner, it can be concluded that the FireFly Dating application was successful.

Contents

1 Introduction

In modern society, many people use dating applications [1]. Within the application, the users swipe dozens of people, and once two users swipe each other to the right, they become a match and could start a conversation. Chatting over the Internet is more convenient and efficient than trying to chat with someone face to face [1]. Besides, by using dating applications, users can reach more people over the Internet than they can in their local city [1, 2]. Dating applications also make it easier for people to find what they are looking for, even if that is very concrete [3]. However, users will most likely not act further than the apps chat functionality. According to Siraadj Salarbux, the client of this Bachelor End Project (BEP), people waste their time in the chatroom and often have the same conversation again but with a different person.

Online chatting has proven to result in social compensation. People are often too shy to meet new people, but they do not have to show their faces with chatting. They need an ego boost to help them to go on a date and meet new people. According to research, only a third of cyber friendships make some form of offline contact, and only 6% form an intimate or romantic relationship [4].

Therefore, FireFly came up with the idea to move the conversation, that is usually held in the chat window of the dating application, to the bar or the café. According to FireFly, the problem with modern dating applications is that there is too much time spent in the chat window.

The team has created a new blind dating application from scratch. There was a design delivered by the client, but the team was free to change it. The team delivered a Minimal Viable Product (MVP) to the client that met the must-have requirements of this project.

In this report, the development process and the final design of the Bachelor End Project (BEP) of FireFly Dating are documented. First of all, the problems that this BEP project addresses are described in Chapter 2. The research conducted before implementation is summarized in Chapter 3 and can be found in its entirety in Appendix B. Next, the design of the system is explained in Chapter 4. Then, the implementation of the product is separated into the design and the back-end. The implementation is fully described in Chapter 5. The testing systems used in this project are explained in Chapter 6. Furthermore, the final product is shown in Chapter 7. Also the ethical implications will be discussed in Chapter 8. The process and the application's development are examined in Chapter 9, upon which conclusions are drawn in Chapter 10. Finally, to finish it up, the discussion and recommendations can be found in Chapter 11.

2

Problem definition and Analysis

This thesis attempts to solve a problem found in modern dating applications: people spend too much time chatting and not meeting other people. This problem is further elaborated in this chapter by explaining the needs and motivation of the client and the challenges which each of the stakeholders faces. First, the context of the problem will be addressed in Section 2.1. Secondly, the challenges faced are described in Section 2.2. Finally, a conclusion about the problem and the analysis is given in Section 2.3. Chapter 3 will elaborate on the research.

2.1. Context

A dating application is an online dating service that helps its users meet romantic partners. A user can, among other things, set data such as their name, height, biography, and upload photos. Users can view each other and eventually go on a date if both users show interest. Researching a variety of dating applications, the following characteristics of the software could be found in all of them: Users were able to swipe other users. By doing this, the algorithm learns about preferences of the appearance of people according to the direction of their swipes [5]. These preferences tell the system if they like the other user or if they do not. When both users like each other, they are a match, and they can talk to each other. In some dating applications, users can specify their preferences and ask the system to go on a date. However, the client has identified the following problem: Most dating apps have a low match to date ratio. It is not uncommon that nearly all the users' time ends up in the chatroom provided by the dating app [6]. According to the client, users often have the same conversation repeatedly with people. The client considers the chatroom to be an ineffective method of determining whether a person is a suitable match. Thus, the client has set out to create a dating application that focuses on dating rather than chatting. The goal of FireFly Dating is to make it easier for users to meet each other in person through blind dates, rather than wasting time chatting online.

2.2. Challenges for Stakeholders

When considering how to solve people spending to much time in online chatrooms and not spending enough time meeting each other, there are multiple stakeholders to consider. The three stakeholders of this project are the user, the Dating Establishment Owners, and the client/product owner. In this section, the unique challenges of the different stakeholders will be described. These challenges must be taken into account when creating the final product to solve the problem.

2.2.1. Users

Users often have the experience of not gaining that many matches [7]. Finally, getting a match and turning this match into a (successful) date can be a time-consuming process [8]. People often trust third-party apps to find a good match with whom they can go on a date. It is shown that a user's swiping quantity does not guarantee more matches. Women usually have more matches than men do, while men have to start the conversation, which does often not lead to a date [7]. Therefore, the application must make it effortless to find a match and plan a date with another user. Only through this

method is it possible to minimize time spent online and maximize the time meeting new people.

2.2.2. Dating Establishment Owners

Dating Establishments want to receive more customers so they can increase their profits. However, they must be able to track the number of people they expect to host at their establishment. Therefore, they require a method of being notified when a new date will be visiting their establishment. Dating Establishments must not experience any downsides for participating in the ecosystem of the application.

2.2.3. Product Owner & Administrator

The Product Owner wants to run a business based on hosting a platform that allows users to match and date with each other. On the other hand, the platform must keep the users satisfied and secure. The product owner must be able to perform administrative actions to ensure satisfaction and security. These tasks include banning users, adding new Dating Establishments, and ensuring the app's correctness of the information.

2.3. Conclusion

In conclusion, the system should allow users to date without chatting (too long) in the application itself. The dates that are made for them should meet their preferences. The dates should take place in Dating Establishments, and these Dating Establishments should not experience any disadvantage from reserving a table specifically for users of the application. Finally, the Product Owner should have full control over the system the ensure user satisfaction.

3

Research

Before implementation could begin, decisions had to be made to accomplish the requirements from the problem definition in Chapter 2. Before this research, a list of requirements has been set with the client and can be viewed in Appendix A. During the research phase, extensive research was done to determine the best systems for the application. A revised requirements list has also been established during the research phase. After the coach reviewed the project plan, some requirements were changed in priority. The changes were made because these requirements were not contributing to the vital components of the MVP. The final list of requirements, combined with the research report, can be viewed in Appendix B. For the research phase, the following research questions were formulated:

- What requirements does the client desire for the correct functioning of the product?
- How will the implementation challenges related to payment, refund, and reporting be solved?
- How will a matching algorithm that accurately matches people be created?
- What frameworks would best suit this project for different areas of implementation?

In this section, the research report's findings are summarized, and conclusions from the research are repeated.

3.1. Requirements

The requirements of the final application of FireFly Dating are split up into three parts. First, a list of stakeholders lay the base of the participants that interact with the system. Secondly, the functional requirements will be discussed. These are the features envisioned by the client and describe the interaction with the various stakeholders. Lastly, the non-functional requirements are listed. This list elaborates more on the requirements that are specific for the system and cannot directly be translated into a single feature.

3.1.1. Stakeholders

This section explains the set of stakeholders that interact with the application. It also elaborates on the roles these stakeholders have in the system.

User

The most important stakeholder is the user of the application. The user should be able to easily navigate and interact with the application. Also, the user should be able to participate in the various features like setting up a profile, swiping, dating, and provide feedback on their dates.

Administrator

The Administrator is the stakeholder that is assigned to keep the application up and running. They will deal with adding new Dating Establishments, locations, and Time Slots. As well as manage reporting, and inspecting data.

Dating Establishment Owner
The Dating Establishment Owner (DEO) provides the application with locations at which dates can take place. These DEOs will have contact with the Administrator, who will set up dates at their establishments.

Product Owner
The Product Owner is the rightful owner of the FireFly Dating product and has a unique set of objectives envisioned for the project to make it a successful and profitable business.

3.1.2. Functional Requirements
The functional requirements for the MVP have been established with the vision of the Product Owner and the prediction of time required to finish the desired features of the project. For the MVP users must be able to:

- Create and delete an account.
- Log in to the application.
- Set their dating preferences and user information.
- Be able to swipe other profiles.
- Apply for a blind date with locations and Time Slots.
- Receive notifications of an updated date status.
- Provide feedback on a date that has taken place.
- Cancel a date.

A more detailed list of these requirements can be viewed in Section B.2.2. Some of these requirements include reporting, rescheduling, administrative tasks, and DEO tasks within the application. This list also includes non-essential requirements. Some non-essential features that have not been implemented in the final application are redacted on the request of the Product Owner.

3.1.3. Non-Functional Requirements
Some of the requirements do not translate directly into features. In consultation with the client, these requirements are created to give the user a seamless experience when using the applications. It also allows the application to grow, granting a user experience without noticeable errors and constant uptime, as well as securing potentially profitable business. Users have to be able to access the application securely from a mobile device or personal computer, and the Client App should be easy to use. The user's connection should have a secured layer that protects the user's data from unauthorized use, to achieve online safety.

As the team will only work on the project for ten weeks, the codebase should be well documented so that future developers will have an easy time understanding and can continue further development of the system. The codebase should also be migratable as it will start as a web application but could also be transferred to a smartphone application.

3.2. Implementation
Some of the features in the functional requirements need to be researched in order to allow for a proper implementation. One of these features is the payment system. For a user to enroll for a date, they will need to pay a small fee. There are a few options for Payment Service Providers (PSPs), and the main criteria for a suitable PSP are the ease of use, availability, and has to support IDEAL. In the end, the decision was set to be between Stripe[1] and Mollie[2] as they both fit the criteria and are made for startup applications such as this one. The final decision was left to the client.

[1] https://stripe.com/en-nl
[2] https://www.mollie.com/en

If a date would be canceled, the participants require a refund of their payment. This refund could be done in different ways, such as refunding money to the user's bank account or turning the money into tokens for the FireFly Dating application. Tokens would be more beneficial as it would require less implementation, especially with third-party software. Tokens are also more cost beneficial as a PSP requires a fee for every transaction. The legalities surrounding such a credit system could be an obstacle if this feature were to be implemented, and therefore, a legal team's consultancy would be recommended.

Another point of research is the feature of moderation. To make the users feel safe, a user that breaks the rules should be able to be reported and banned. The application will rely on the feedback of users to bring attention to users that misbehave. An Administrator can then view this feedback and decide on the validity of the feedback and take appropriate actions. The DEO could also provide their input to get an unbiased view on the matter. FireFly Dating could reward DEOs for their supervision by allocating more dates to their establishment, see Section B.3.3.

3.3. Matching

The matching algorithm is a significant part of a blind dating application. The application will need to create dates based on the information it has, which is initially very little in the case of FireFly Dating.

First, the algorithm will need to use absolute filters like age range or gender preference to accompany the user's distinct preferences. After applying this filter, the only data available to make any recommendation on is the feedback received from swiping and the feedback from a date at a Dating Establishment.

Both swiping and date feedback are boolean values and are used to indicate when two users have liked or disliked each other during swiping or when two users experienced a date as positive or negative. Furthermore,Collaborative Filtering (CF) could be used for this application as it works on sparse and absolute data to calculate the likelihood of users liking each other, which should then be used to recommend a date based on other users' data.

The algorithm also has the option to take the popularity of people into account. When users like/dislike ratio are closer together, they tend to be a better match than when the ratios are further apart. So users that are liked often by people are more likely to be selected for a date. This ratio is called the Elo score, see Section B.4.4.

Combining these implementations will result in a suitable matching algorithm for the application. A complete report of the research of the complete algorithm can be found in Section B.4.

3.4. Framework Analysis

For the FireFly Dating application, a set of frameworks will be used to build the application. Therefore research had to be conducted on front-end, back-end, and testing frameworks to choose the most suited frameworks for the application. A detailed analysis of the frameworks can be found in Section B.5.

3.4.1. Front-end

The front-end frameworks that were considered were Angular[3], React[4], and VueJS[5]. A full report of the comparison can be found in Section B.5.1. Angular was soon dismissed as the framework was not adequately built for mobile and required more work to set up for smaller applications. React, and VueJS both fit the application well. They both have high developer satisfaction, support the usage of the majority of libraries, and proper mobile support. The final decision was to use the VueJS framework as it allows for greater efficiency as well as prior experience of some team members.

3.4.2. Back-end

The back-end frameworks are split into three parts: the database, the server-side, and the matching algorithm.

[3] www.angular.js
[4] www.reactjs.org
[5] www.vuejs.org

Database
Several database structures were considered, and a full report can be found in Section B.5.2. The application will rely on a relational database as the application will be static in its structure. Insertions will only have values that match strong definitions such that queries are more straightforward and faster to execute. MySQL[6] and PostgreSQL[7] comparison studies show that MySQL is more straightforward to use, and better supported [9]. The size, scalability, speed, security, and plugins of MySQL suit the application's needs and all team members have had prior experience with the framework. Therefore, MySQL was chosen.

Server-side
The server-side scripts will be written in JavaScript, with NodeJS as the runtime environment. NodeJS provides excellent speed and reliability for executing queries. A detailed comparison with other runtime environments can be viewed in Section B.5.2. The JavaScript language was also chosen to keep a uniform language throughout the application and speed up the development.

Matching Algorithm
The matching algorithm will be developed in JavaScript to keep a uniform language across the application. NodeJS was chosen for the best execution of heavy tasks. A comparison with other frameworks can be viewed in Section B.5.2.

3.4.3. Testing
Since the previous decisions led to the entire application to be programmed in JavaScript, only one testing framework is required for front- and back-end. Two different popular testing frameworks that were compared were Jest[8] and Mocha[9]. A detailed comparison can be found in Section B.5.3. However, to summarize, the significance of the application relies on a correctly behaving back-end. Mocha operates better with NodeJS, and thus Mocha was chosen as the testing framework.

[6] www.mysql.com
[7] www.postgresql.org
[8] www.jestjs.io
[9] www.mochajs.org/

4

Design

FireFly Dating is a consumer-facing product. Therefore, in consultation with the Product Owner, the team decided that it was essential to take special care to ensure that the app is designed with a good User Experience (UX). Therefore, the team has made trade-offs between adding features and polishing the design. This process will be discussed in this section.

4.1. Design Goals

When starting this project, it was essential to set crucial design goals to ensure that the product would function in such a manner that it provides benefit to its users.

Clean User Interface

The User Interface (UI) was designed to be simplistic and easy to use by users. The goal was to ensure that all user-facing screens of the Client App only show the minimal required amount of information, to avoid screen clutter.

A second aspect concerning the UI is conscious of the location of each component. The goal was to place the elements so that they do not feel crowded or claustrophobic to the user. Instead, they should feel open and clear.

Mobile First Development

An essential part of the design process was to ensure that all screens would function well in a desktop and mobile browser environment. Therefore, it was paramount to guarantee a pleasant UX on mobile devices. When implementing the design, the team must always use a mobile-first style of development.

Intuitive Interface

It is vital to make sure that the flow of the Client App is logical and intuitive. This flow helps the user to enjoy the application's experience. To make the app more logical, it should guide the user to the next step.

Playful Design

As expressed by the Product Owner, he wanted the Client App to feel playful and informal rather than severe and purely functional. The goal of this is to make the app appealing to the current college-age target audience of the app. It also serves to make the usage of the app feel more enjoyable rather than a chore.

4.2. Design Process

Throughout the development, there were different design stages. First, the team received an initial design created by the Product Owner. After this, the team built their design in Adobe XD, and eventually, after several revisions with the Product Owner, the team's final design was implemented. These stages and the resulting models are explained in this section. The process of creating a screen was similar for all of them. Therefore we have chosen to only focus on the Swiping Page as an example.

4.2.1. Design Goals and Initial Design

During the first two weeks, the team, in consultation with the Product Owner, set the project's design goals. These goals are described in Section 4.1. These goals created the foundation of the design aspects of this project. During this time, the team also discussed which color pallet[1] to use. The Product Owner provided the team with an initial design that he created in preparation for this project, as seen in Fig. 4.1. This design gave the team a general idea of how the design should look and feel, and functioned as the basis for the development. During this stage, a flow diagram was created in which the general flow of the application is presented; this flow diagram can be found in Fig. C.1. With this diagram, the team had a clear overview of what screens needed to be designed. Creating such a flow diagram also provided an excellent tool to validate if the team and the client had the same design vision in mind.

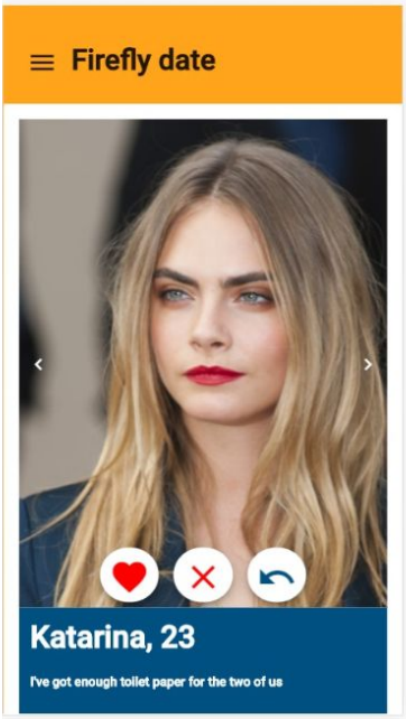


Figure 4.1: Swiping Page Design Provided by the Product Owner

4.2.2. Design by BEP Team

During the first week of actual development, one of the primary goals was to create a design that would show the app's general functionality. This design was created in Adobe XD as this is a tool that is capable of creating a UI design in a quick manner, which is not possible when implementing in code right away. The team was able to share these designs with the Product Owner and receive feedback on them before implementation. It was a conscious choice to focus primarily on the main UI components in this initial design: the swiping page and the dating page. The designs of the mobile version of the swiping page can be seen in Fig. 4.2 & Fig. 4.3.

4.2.3. Implementation Phase

After creating and reviewing the designs in a non-implemented form, the team was ready to implement the actual design. Each page's design was initially done using sample data rather than having it already be connected to the back-end. After the design had been implemented in VueJS/SCSS, the reviewing process could begin. The pull request system used in GitHub was a crucial part of optimizing the design. It allowed all team members to view and request changes to the design before becoming part

[1] https://colorhunt.co/palette/177867

of the main branch. During the implementation phase, the team also discussed and demonstrated the design to the Product Owner. Based on the Product Owner's feedback, the team was able to further improve and iterate on the design.

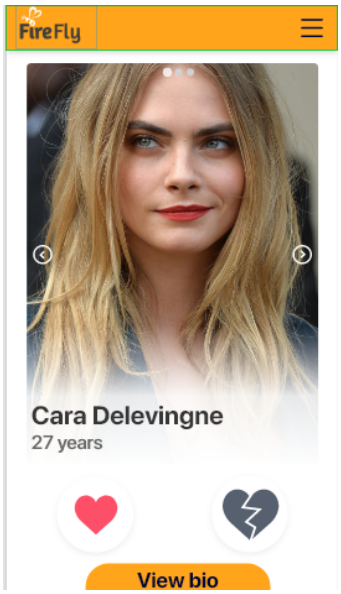


Figure 4.2: Mobile Device Swiping Page made with Adobe XD

Figure 4.3: Mobile Device Swiping Page showing Biography made with Adobe XD

Figure 4.4: Final Mobile Design of Swiping Page

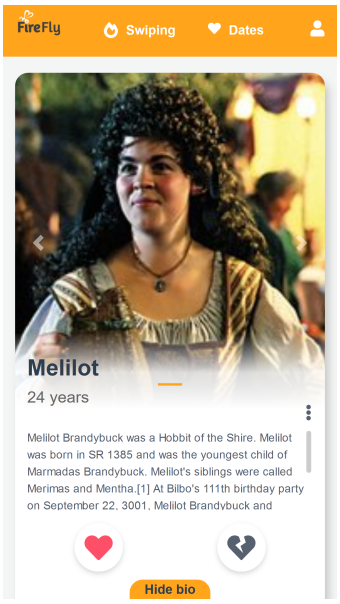


Figure 4.5: Final design of swiping page, showing the bio, for mobile devices.

4.2.4. Final Design

After eight weeks of development, the team was able to create a design that met the majority of the design goals; these goals will be reflected in the next section. The final results are quite similar to the design, as can be seen in Fig. 4.4 & Fig. 4.5. The differences between the original design, shown in Fig. 4.2 & Fig. 4.3, and the final product are shown in Fig. 4.4 & Fig. 4.5. These differences were implemented because, during the implementation phase, it was discovered that these changes worked

better when using the application. To the eyes of the client, the team succeeded in creating a clean and aesthetically pleasing design in the time that was set for this Bachelor End Project (BEP).

4.3. Reflection on Design Goals

At the start of this project, initial design goals were defined to ensure that the project design satisfied the requirements of the Product Owner and, thus, the user's needs. This section will reflect on how well these design goals were accomplished.

Clean User Interface
The team succeeded in avoiding too much clutter on the different screens of the Client App. The screens only have the necessary amount of information to allow the user to navigate the app effectively. The team and the Product Owner consider this goal as achieved. In the future, when more user testing has been done, this goal might have to be revised.

Mobile First Development
All screens function correctly in a mobile environment as far as we were able to test. Together with the Product Owner, the team has concluded that this goal was achieved effectively. However, mobile-first was only tested on browsers and not entirely on mobile devices. Thus some flaws may be undiscovered; this is because, with the authentication service, it was unable to view the application on the phone before deploying it to a server.

Intuitive Interface
The screens implemented do follow a logical progression in the app. However, the team feels that if there was more development time available, there could have been some additional improvements in this regard. Such as automatically guiding the user to the next screen and allowing for more natural gestures. The team would have liked to improve the intuitiveness of the final design, but the team had to make a trade-off between implementing more crucial features of the MVP and designing the application to be more intuitive. After discussing with the Product Owner, the team concluded that it was more valuable to improve the application's basic feature set.

Playful Design
The goal of achieving a playful design was achieved quite successfully in this app. the playfulness is expressed through a few different ways: the color scheme[2], the choice of icons[3], and the informal explanation texts. The Product Owner positively evaluated the app's general look and found it to be an enjoyable and fun experience. Therefore the team considers this design goal as successfully achieved.

4.4. System

Another aspect of design that was not ignored is the design choice made concerning the system design. In this section, a few critical design decisions that were made during the project will be highlighted. These design choices aided the development of the project and ensure successful future development.

4.4.1. Front-end Back-end Communication

In the early stages of the development, there was a conscious choice to make a sharp separation between the front- and back-end. The only communication between these two sides of the project would be done through Application Programming Interface (API) calls. The reason for this is that the Product Owner expressed the desire to switch to a standalone mobile application in the future. For this reason, front- and back-end must be separated so that a mobile app can make use of functionality without altering the back-end.

4.4.2. Choice of Programming Languages

As described in Appendix B, the choice was made to primarily code in JavaScript. This decision was made to minimize development and learning time because all team members had prior experience with

[2] https://colorhunt.co/palette/177867
[3] https://fontawesome.com/

JavaScript. Using mainly one language, resulted in a minimal learning curve throughout this project for all team members. In the end, an additional programming language was used for the matching algorithm, namely Python. Using Python for this task optimized the IO infrastructure of NodeJS. Chapter 5 will elaborate on the use of Python.

4.5. Conclusion

In conclusion, at the start, the team cooperated with the Product Owner to set different design goals for the project. These goals were taken into account when designing the final product. The process of implementing the design happened in different phases. First, the design goals were set. Then an initial design was created. Finally, the team implemented the design into the codebase. Overall, almost all design goals were achieved. Only the intuitive design goal was not fully met due to time constraints.

Concerning the system design, the team made a conscious choice to keep the front- and back-end separated as much as possible. This will allow for the creation of a standalone mobile application in the future. Additionally, about system design, the team chose to use JavaScript as the primary programming language to improve the efficiency of development due to familiarity. Overall the design process was successful in this BEP project.

5

Implementation

This chapter focuses on the implementation of the application and its components. It starts by giving an overview of the system as a whole and then focuses on the individual components. It opens with the front-end systems, first the Client App, and after that, the Admin App. Next, it explains the workings and the integration of both the back-end system and the matching. Lastly, it provides an overview and explanation of the database structure and uses.

5.1. Overview

The overview describes the workings of the front- and back-end. The front-end section first covers the Client App. After this, it moves on to the Admin App. The back-end section explores all the server-sided systems that are in place, excluding the database and matching. For an overview of the directory structure of the project see Appendix D.

5.1.1. Front-end

Both front-end systems have the VueJS framework at their core. VueJS is a reactive component-based framework making it ideal for a large but maintainable codebase.

Client App

The Client App is a mobile-first web application using HTML, SCSS, and ECMAScript 6 (ES6) in the VueJS framework. This web application serves as the primary interface between users and the FireFly Dating app.

The application is designed as a single-page application. Single-page applications update the current webpage with new content instead of moving to a new page [10]. By only partially changing the content, it saves unchanged content from being reloaded[1]. So when a user moves from $/settings$ to $/dating$ the page does not change to a new web page, it just loads the appropriate content. In order to load the appropriate content, VueJS contains a router[2]. This router distinguishes between two types of pages, namely, public and private pages. Private pages are only accessible after authentication. An overview of the routes can be found in Fig. 5.1

Firebase, a development platform for mobile and web applications, provides the authentication flow of the application. This platform allows us to link multiple OAuth services like those of Google and Facebook to the application with relatively low effort[3].

The application uses API calls to receive and send data. With the swiping page, multiple candidates are prepared at the server-side while the client is swiping. By preloading theses candidates, the Client App provides a smooth transition without the need for long waiting times.

[1] https://ozitag.com/blog/spa-advantages
[2] https://router.vuejs.org/guide/
[3] https://firebase.google.com/docs/auth/web/start

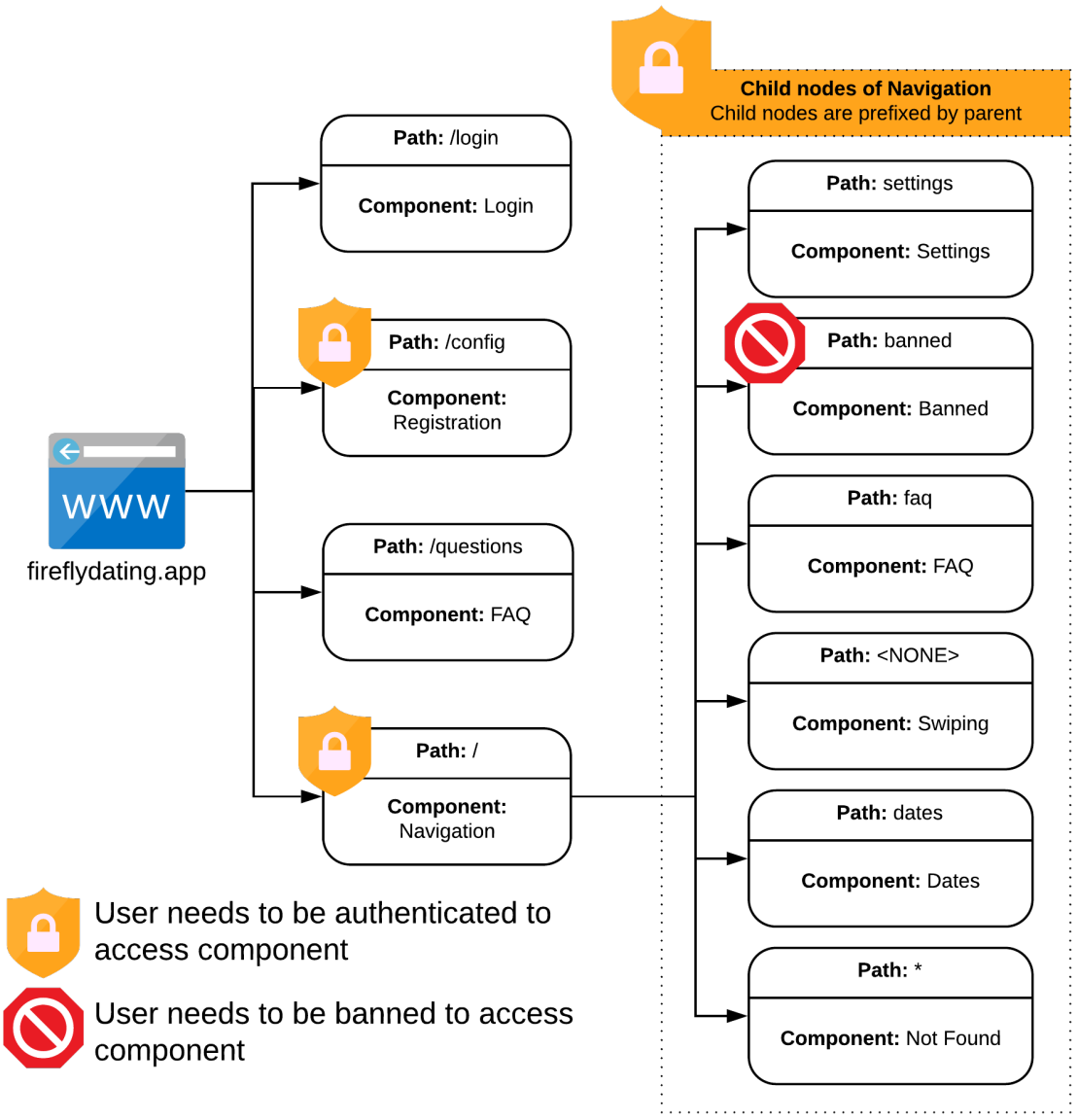


Figure 5.1: An Overview of Routes in the Client App

Admin App

The Admin App uses most of the same technologies as the Client App. The main difference is that instead of the BootstrapVue[4] library, the Admin App uses Vuetify[5] library as this provides the needed Create, Read, Update and Delete (CRUD) data-tables[6,7].

Like in the Client App, the Admin App pages are managed by the router. For the Admin App, all pages are private except for the login page. An overview of the Admin App routes can be found in Fig. 5.2

The Admin App does not make use of the Firebase authentication service. Instead, it uses a token to verify the session. On the login page, an Administrator can provide its token to create a verified session.

[4] https://bootstrap-vue.org/
[5] https://vuetifyjs.com/en/
[6] https://vuetifyjs.com/en/components/data-tables/#crud-actions
[7] https://www.codecademy.com/articles/what-is-crud

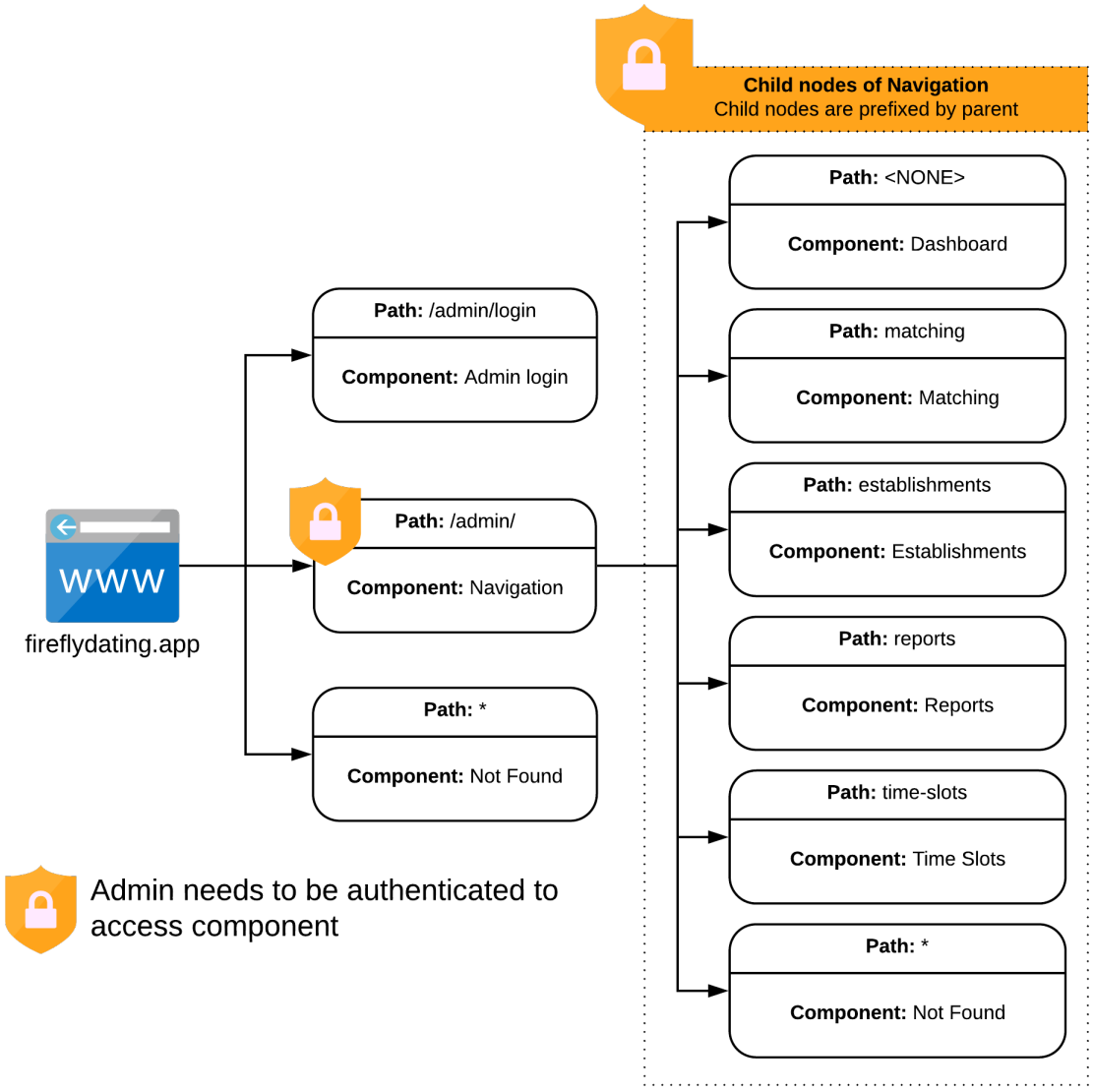


Figure 5.2: An Overview of Routes in the Admin App

This token is hardcoded in the back-end, see Section 11.1.7 Admin Login for more information.

In the Admin App, the Administrator can manage all the data needed for the FireFly Dating application to function. They can perform CRUD operations on all kinds of data, such as time slots. The Admin App is also the place where Administrators view report cases. These are provided with the information related to each report. Based on this, they can judge a presented case and take appropriate actions.

5.1.2. Back-end

The back-end primarily consists of a NodeJS server. This server is responsible for communicating the data to and from the clients. The communication is based on API calls using GET and POST requests[8].

[8] https://www.w3schools.com/tags/ref_httpmethods.asp

Routing and Call Handling
These API calls or routes can be structured using the Express[9] library for NodeJS. Express allows for grouping these routes in routers giving the server a clear and easy to understand structure. For this project, routes are grouped based on their interaction with objects. For example, a route that retrieves the user data should, therefore, be located in the $userRouter$ file. These routers then allow us to assign prefixes to routes. A request to retrieve the user data would look like this: $/user/get$ where $/user$ refers to the user router and $/get$ points to the get method inside the user router. A full overview of the call structure can be found in Appendix K.

Authentication
Not all routes should be accessible by everyone. Therefore, an authentication system was implemented in order to guard routes from access. This system would assign users a session once they logged in. This session object controlled access to routes and allowed us to identify users server-sided. For the implementation of this authentication system, Firebase Authentication[10] is used. Firebase Authentication will henceforth be referred to as just Firebase. Firebase is a service that allows the connection of a multitude of OAuth 2.0 services without having to create specific implementations for each OAuth service. OAuth is an authentication service provided by an external party, for example, Google. Firebase allows users to log in to the application using their Facebook or Google account. Additional OAuth providers can be added to the system without an implementation change. A provider can easily be added by changing the Firebase configuration.

Object-Relational Mapping
Object-Relational Mapping (ORM) is a method of converting data from one system to another. In this project, this was from MySQL data to a JavaScript object. The back-end uses BookshelfJS[11] and KnexJS[12] to help with these conversions. KnexJS is a query builder for a multitude of DataBase Management Systems (DBMSs), including MySQL and MariaDB. KnexJS is the bridge between NodeJS and the database. BookshelfJS, ORM library that uses KnexJS, allows database entries to be treated as JavaScript objects and can fetch and save data to and from the database without the need for writing queries. Using BookshelfJS allows for implementing additional fields to a schema without having to rewrite all the queries.

Mailing
The application relies on email for notifications. Therefore the emails must not end up in the spam filter but land in the user's inbox. The project uses SendGrid[13] to provide legitimacy for analytics on the emails. The SendGrid-NodeJS library was used to implement sending emails from the server, that are created using Embedded JavaScript templating (EJS). EJS is a framework that allows building HTML pages dynamically using JavaScript. The main template can be outfitted with one of the modules depending on the type of email. Data is then passed on to this module to fill the email with the correct information, like names, dates, and places. Once all the data and modules are in place, the NodeJS server renders this page to create a plain HTML. It also generates a plain text version of the email to provide support for mail clients that do not support HTML emails. The HTML and plain text are passed on to the SendGrid library and are sent to the users.

5.2. Database Structure
This section describes the structure and implementation of the database technologies used in this project.

5.2.1. MySQL
The database engine used for this project is MySQL 8.0.0. All the queries and libraries used are compatible with MariaDB 10.2.2, making the MySQL database interchangeable with MariaDB. Connection

[9] https://expressjs.com/
[10] https://firebase.google.com/docs/auth
[11] https://bookshelfjs.org/
[12] http://knexjs.org/
[13] https://sendgrid.com/

to the database is made through KnexJS and a configuration file. The default configuration uses the root user without a password and connects to a database called $firefly$. If a secret configuration file is locally present, then that configuration will be used instead.

5.2.2. Building the Database

The migration and seeding features of KnexJS are used to deploy and maintain the database structure.

Migration

Database migration (or schema migration) is a version control system for the structure of a database [11]. The database tables can be constructed or updated to the latest version by executing a change-list in chronological order. These change-list files also contain the instructions for downgrading the database to the previous version. This construction of up- and downwards changes allows for jumping to specific database versions during development. One has to keep in mind that migrating downwards usually leads to loss of data.

Seeding

Seeding (or database seeding) is the act of filling the database with initial data. The seeding framework consists of a list of chronologically executed commands that alter the content of the database. FireFly Dating makes use of seeding to populate the default values for the gender, language, report, and location tables. This project also uses seeding for manual testing. These testing seeds contain fake users with their preferences, dating establishments, time slots, enrollments, and feedback. These testing seeds allows for rapidly testing new features against a multitude of different users with their settings.

5.2.3. Schemas and Relations

The database consists of 24 schemas. Of these 24, two are managed and created by KnexJS to keep track of migrations. One is used to store the session data and is operated by Express and KnexJS. The remaining 21 schemas form the core of all data used by FireFly Dating and are all connected in some way. These connections are in the form of a foreign key[14]. Foreign keys are fields that point to a field in another table. A description of all tables can be found in Appendix G. Not all the extra constraints are displayed here, as some are derived from KnexJS. A full schematic overview, including the relations, can be found in Fig. F.1 & Fig. F.2. These figures are spread out over two pages to improve readability. Fig. F.1 is centered around the user table. Fig. F.2 is centered around the date table. The line connecting the two pages represents links to the id field in the $user$ table and the $user\_id$ fields in other tables.

5.3. Matching and Dates

For an application focused on dating, it is essential to match people and create actual dates. In order to improve the matching, users can specify their preferences and constraints. This section will cover the list of constraints, the implementation of the swiping, creation of matches, and the eventual construction of a date.

5.3.1. Constraints

In order to create suitable matches, a list of constraints is put in place. This list consists of the following constraints.

Gender interest - makes sure that two users match each other's sexual orientation

Languages capabilities - ensures that two users can have a conversation on their date.

Age range - makes sure that two users fall in each others age range

Available locations - make sure that two users can meet up at the same location

Available times - ensures that two users have a joined time when they can meet.

[14] `https://www.w3schools.com/sql/sql_foreignkey.asp`

Date planned - prevent users that already have a date linked to the enrollment to go on another date for that enrollment.

Dated before - prevents users that already dated each other from going on another date.

Banned status - prevents banned users from going on a date.

These constraints ensure that users do not get matches that are outside of their general interests. If one of these constraints is not met between two users, then they cannot be matched.

5.3.2. Swiping

The system uses swiping for preference indication. During this phase, users get a set of candidates that match most of the constraints. The time and location constraints are ignored as these are not present during this phase. The user can indicate for each candidate whether this is a good match or not. Once a user has indicated their interest in a candidate, this candidate will then not show up in swiping for four weeks, allowing users to re-swipe after an extended period. This gives them a chance to reconsider and give the illusion of a larger candidate pool.

5.3.3. Matching

Before implementing the matching algorithm, more research into running substantial computational functions in NodeJS was required. In an article by Gimeno [12], he explains how to use worker-threads to prevent the blocking of the main thread while running computationally intensive calculations. These claims are backed up by a guide on the official NodeJS website [13]. As programming language run in such a worker-thread, Python was chosen. This choice was made due to the familiarity with Python and its ease running scripts from the command line.

Once users choose to go on a date, they fill in their available time, and locations. With this data, it creates an enrollment. Every night at a quarter to twelve, the server starts the matching algorithm. This time was chosen as it is expected to be a time of low usage of the system. It starts by making a list of each Time Slot, location and Dating Establishment that still has spots available. Then for every one of these entries, it checks if there are users enrolled that are available at that time and location. Every available user gets added to a list. It then constructs all the pairs from this list that would meet the constraints specified in Section 5.3.1.

The algorithm uses the data from the swiping phase to rank this list. If user A liked user B during the swiping phase, then that pair gets a plus one on their rank. The other way around works the same, meaning that a pair could maximally have a rank of two and a minimal rank of zero. This ranked list is sorted on rank. With i as the number of available spaces, the first i pairs of users get a date entry in the database. The algorithm concludes by printing the date ids inserted in the date table, as shown in Table G.2 as a result of matches. These ids then get processed by the server and turned into proper dates.

5.3.4. Dates

Once the server receives the list of date ids, it starts to turn those into dates. Turning ids into dates mean that both users need to be notified that they are matched. The user receives an email notification of the match, and the icon on the dating page changes, as shown in Fig. 7.9. A match does not mean a date. In order for a match to become a date, both users will need to accept the match. If both users accepted, then the server sends an email to the Dating Establishment confirming the date. Both users will then get an email with their reservation at said Dating Establishment. At this point, a date could still be canceled by both users. Cancellation will notify all parties about the cancellation via email. It also frees up the spot at the dating establishment. In the current state of writing, canceling can be done until the start of the date. After the date took place, users can provide feedback and possibly share contact information.

6 Testing

In order to create reliable software, it is of utmost importance that is has been thoroughly tested. Doing so ensures that the created codebase has a high maintainability and a proven success rate. This chapter will describe the testing flow of the project.

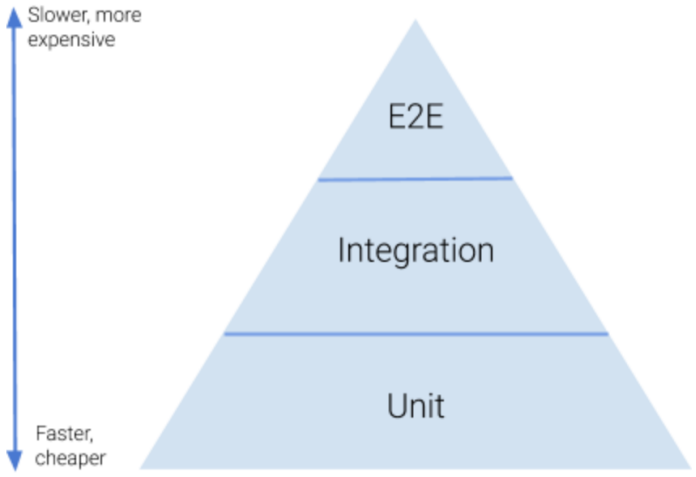


Figure 6.1: Testing Pyramid [14]

6.1. Unit Testing

Unit Testing focuses on testing the smallest pieces of code possible. Testing small pieces of code has many different advantages. Firstly, according to Novoseltseva [15], it is easy to test the code, as each method is split up into testable pieces, instead of a complex integration. Secondly, finding and locating errors is done relatively quickly, as they are located in a small code section. Thirdly, it allows testing in an early stage of the project. Lastly, it is a cost-effective method of testing, since failures can be found early on and be addressed. If these failures happen later on in the process, it is difficult to locate these errors. Thus embracing unit testing as the basis of the testing on this project is an excellent choice. Unit Testing is the foundation of other testing methods, as seen in Fig. 6.1. Unit testing is the most significant method of testing used within the project.

In the initial setup, the framework Mocha[1] was selected as the testing framework of this project, see Section B.5.3. Mocha was chosen because in theory it is supported by both NodeJS and VueJS.

[1] `https://mochajs.org/`

However, it turned out that Mocha was poorly integrated with VueJS; this made it impossible to run test coverage checks with Mocha on the VueJS components. Such a bad interaction made it impossible to use Mocha as the framework to test VueJS in this project. We stated earlier, see Section B.5.3, that using multiple frameworks in a single project is not a practical choice and should be avoided. However, since the front- and back-end are both separate sub-projects, it is possible to use multiple testing frameworks and gain the benefits of both. After a single week of trying to get Mocha working on the front-end, we switched to the testing framework Jest[2], while Mocha maintains to be the testing framework of the back-end. The combination of having both Mocha and Jest is the best suited for this project as a whole.

All the percentages shown in this section are based on the coverage of unit tests. Other testing methods are not described in this section.

6.1.1. Back-end

As of now, only JavaScript files are tested, the back-end also houses two Python[3] files which are not tested. Only one of those files has actual functionality. This file is responsible for finding matches between people based on their availability, preferences, and swiping, as described in Section 5.3.3. Matching is almost entirely done within a Structured Query Language (SQL) query. This query returns a set of candidates. The only functionality that the Python script has is to wrap these retrieved candidates in a date object. Wrapping candidates in an object is not a complex function, but it should be tested appropriately in future versions, see Chapter 11. Due to time constraints adding a third testing framework for a single file was not feasible.

In the back-end, Mocha is used as the testing framework. This testing framework is accompanied by Istanbul[4] as the coverage reporter. The testing coverage passed for the back-end if the back-end code was at least 80% covered by unit testing. This coverage combined with Continuous Integration (CI) allowed the code to be automatically tested in each pull request. Doing so makes it more likely that the main branch is always a correct runnable instance of the software. Throughout the project, high testing coverage was achieved and maintained. The testing coverage of the back-end, excluding the Python files, was greater than 99% throughout the project. Such a high coverage allows for good maintainability and proven correctness of the created software.

6.1.2. Front-end

In the front-end Jest is used as the testing framework, VueJS has its own plugin[5] to allow Jest to function, including a built-in coverage reporter. In the initial phase of the project, the team opted that no testing was required for the front-end as the result of testing front-end components is limited and time-consuming. However, throughout this project, the team noticed that this was not true. Testing on the front-end found and terminated several errors that were made. It does not cost more time to test the front-end in comparison to the back-end. However, they both require a different way of testing, thus switching between different testing frameworks can sometimes be time-consuming, as they feel quite similar but act differently. The required testing coverage of the front-end is equal to the required coverage of the back-end. Both need to achieve 80% individually to pass the minimum test coverage requirements. Running these tests with CI ensures that the software on the main branch is always stable and does what it is supposed to do. Jest also allows for End to End (E2E) testing. However, this type of testing is slower and more expensive, as seen in Fig. 6.1. The option to test for E2E is still a good feature that could be implemented in the future stages of the project. After switching from Mocha to Jest, the front-end could be properly tested. Since implementing Mocha left the front-end at a 0% testing coverage, switching to Jest significantly increased the testing coverage. The testing coverage soared up to 99% and stayed there throughout the project. Such a high coverage of unit testing of the front-end ensures that each component will function as expected.

[2] https://jestjs.io/
[3] https://www.python.org/
[4] https://istanbul.js.org/
[5] https://www.npmjs.com/package/@vue/cli-plugin-unit-jest

6.2. Manual Testing

The entire project is tested by unit tests. While these tests are excellent, they do not test the correct interaction between components. System testing is doing through the means of manual testing. This testing proved to be quite sufficient as the back-end calls are routes that return a data object. The majority of browsers support some form of development tools [16]. With these tools, the network packets can be viewed and checked if the correct packets were created and retrieved. If the wrong packets were retrieved, then it was an issue on the back-end side. If the correct packets were retrieved, but not correctly integrated on the front-end, it was caused by the front-end. This made it extremely easy to isolate the source of the problem and resolve it. Doing manual tests instead of integration tests was feasible as the logic on the front-end is quite limited and relied primarily on visual components.

Solving the errors on the front-end did not consume much time as VueJS allows for Hot Reloading[6]. This hot reload allowed the team to alter the served page without reloading and rebuilding that page. Altering a piece of code updates the rendered page immediately, thus allowing the developer to see if the correct behavior was achieved by altering the code.

These tests were mostly done by the developers of the software; however, the Product Owner also performed manual inspections to see if the software was up to his standard. The Product Owner rarely requested a change. If a change was requested, then it was a relatively minor visual change. Manual testing was a great asset to ensure that the Product Owner's expectations were adequately addressed and fulfilled.

6.3. User Testing

Even though the project reached the state where user testing was viable, no beta testing took place [7]. This was due to the situation surrounding COVID-19, making it hard to arrange meeting locations for people and finding people willing to participate during this crisis [17].

To perform something close to user testing on the system, the team acted out user behaviors. For these tests, each team member created one or more fake user-profiles and enrolled for a date. We would then discuss the expected outcomes and sketched scenarios of these users. Once the scenarios were created, the system would run the matching algorithm, and the results would be compared to the expected results. Afterward, the scenarios would be carried out to see if the system would work for all of them. By using this testing structure, a couple of small bugs were found and fixed. One of these bugs included was being matched with banned users.

In order to gain a better understanding of the look and feel of the application, a small user study was conducted. In this user study, a handful of users were asked to use our application. At first, this was done with a minimal explanation of the app. These test results pointed out that the settings page is not intuitively structured as some users were unclear on the meaning of the gender preference field. After users had the chance to explore the app on their own, they got an explanation and were asked to perform specific steps from the process. A few examples of these steps included updating their settings and uploading photos, changing their photos and ordering them, and creating a date enrollment. The full list of feedback is stated below.

- Some users did not like that there is only the option of connecting with a Google or Facebook account.
- The logo icon does not redirect to the 'homepage'; this would be the expected behavior.
- The gender setting in the top part of the settings page had an unclear link to preference. Some users thought that they had to set their gender twice.
- The file size of 500kb per picture is way too small.
- The settings page is very long. Maybe think about creating tabs for each section.
- The picture navigation arrows in the swiping screen are always visible, even if there is no picture to move to.

[6] https://vue-loader.vuejs.org/guide/hot-reload.html
[7] https://www.softwaretestinghelp.com/beta-testing/

- The message that shows when there are no matches is not clear. It should specify that it could be due to strict preference settings.
- It is not indicated that the algorithm is searching after creating a date enrollment.
- It is unclear that swiping is optional.
- Some users are confused with a swiping option in a blind dating app.
- Deletion of photos does not work on all mobile phones.
- The registration menu is not entirely visible on all phones.
- The biography field is hard to read on smaller mobile devices.
- There is no scroll bar for some of the menus when the content does not fit.

6.4. Stress Testing

When FireFly Dating has a significant number of users, it still needs to function correctly without the users experiencing delays. For this reason, stress-testing was implemented in the testing of the system.

Currently, the only part that is appropriately stress-tested is the fetching of candidates. This query retrieves a bulk of candidates for the user to swipe through. Once this bulk is depleted, the query will be called again to fill up the candidates' backlog for the user to swipe through. This query was stress-tested in a database with eight million users. The execution time of the query only took around two seconds. This run-time happens in the background so that the user will not experience any delay. Such a low execution time is excellent for such an extensive database.

It should be checked if this query does not take too long when multiple users are simultaneously fetching new candidates. Currently, it is not possible to automatically log in with different stress-tests accounts to call the database and time the interactions. Due to the required authentication with Firebase to create a session on the back-end. In order to allow stress-testing to occur with multiple users, the authentication service needs to be bypassed; this could cause a breach in security if done improperly. In the future, it is possible to bypass this authentication service to allow stress-testing to occur.

Stress-testing with multiple users will provide insights into how the system handles high traffic situations; this is essential in the future for functions that are more complex and time-consuming.

6.5. Continuous Integration

The code that is located on the main branch of the GitHub of this project must always be correct and free of known errors. This reliability is done via Continuous Integration (CI). If someone wants to merge a branch with the main branch, all the tests must be run alongside the static analysis. These checks are done before the developer is allowed to merge their branch with the main branch through the means of automated testing. CI makes sure that these tests ran and were accepted or rejected. If these were rejected, then the developer is denied the option to merge their branch with the main branch. If these checks pass, then at least two other developers need to approve these changes. Not all errors are found and addressed with these automated testing. Thus human supervision is required to ensure that the added functionality does what it is supposed to do.

In the earlier stages, CircleCI[8] was used for the CI. Working with CircleCI went okay until the project had many pull requests simultaneously. CircleCI is free software in which each project has a certain amount of free minutes to spend on their CI per week. These were quite a limited amount of minutes and were quickly drained during a week, making it unable to merge any more code with the main branch as all the checks failed since they could not be run. The second time we ran out of credits, the team discontinued CircleCI and went with GitHub Actions[9] for the remainder of the project. GitHub Actions allows for CI the same way as CircleCI. However, GitHub Actions provide the project with more credits and a more cost-friendly payment system to gain more credits for the scale of the project. The system is a one-time buy option instead of a subscription. This change allowed the merges to occur once again and assured that each pull request contained proper code.

[8] https://circleci.com/
[9] https://github.com/features/actions

7

Final Product

This chapter will elaborate on the final product as a deliverable for the BEP. First, a detailed walkthrough will be given, which will be done in two parts. The first part will be a walkthrough of the Client App, and the second part will be the Admin App. After this walkthrough, we reflect on the MoSCoW requirements from Section B.2.2 and view whether or not the application meets the defined requirements.

The final user-flow of the application is visible in the flow diagram shown in Appendix C.

7.1. Walkthrough

In this section, a detailed explanation of all screens within the application will be given, elaborating on each interactive element the user can interact with. This section is split up into two parts, the first of which will show the application from the user's perspective. The second part will focus on the interactions an Administrator has in the Admin App.

7.1.1. User

This subsection explains the screens and interactions of the Client App. The user's interactions will be described, and the entire process from account creation to dating feedback is explained.

Login

Once the user connects to $firefly dating.app$, user is presented with, is the Login screen, see Fig. 7.1. If the user has already registered and has a stored session, the user will automatically be forwarded to the Swiping page. Otherwise, the user can create an account or log in by choosing either Facebook or Google to log in or create an account using Firebase, see Section 5.1.2. Furthermore, on this page, the user will have the possibility to access the Frequently Asked Questions (FAQ) page in the top right corner. When a user presses either option to log in, one of two things will happen. If the user has not registered yet or has not completed the registration, the user will be forwarded to the Registration page. If the user already has an account with valid information, they will be forwarded to the Swiping page. However, if the user is banned, they will be shown a message indicating they have been banned.

Navigation

The navigation bar can be one of two variants. The first variant appears when the user is not logged in to the system, allowing the user only to cycle between a few pages. However, when logged in, the user can access multiple screens. These include access to the Swiping, Dates, Settings, FAQ, and Logout page. The not logged-in navigation bar is visible in Fig. 7.1 and the logged-in navigation bar is visible in Fig. 7.4.

Frequently Asked Questions

The FAQ page is displayed in Fig. 7.2. When a user has questions about the application, they can access this page via the navigation bar to find an answer to their possibly frequently asked question.

Figure 7.1: Login Page of FireFly Dating

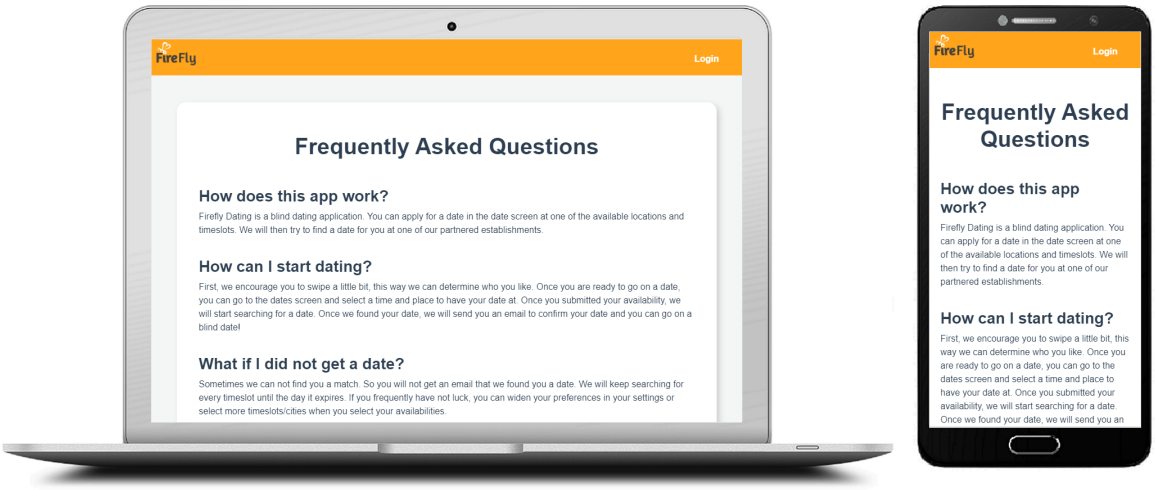


Figure 7.2: FAQ Page of FireFly Dating

Registration

After the user logs in for the first time, the Registration page is presented. In this screen, the user needs to enter their information. These fields include their first name, last name, gender, and date of birth. This information cannot be changed later by the user, therefore this screen was created. Users will always return to this page after login if their provided information is incomplete. The user is not allowed to enter numbers or emojis as first or last name or enter a date of birth that does not make them at least eighteen or at most eighty years old. A message is displayed if specific fields are incorrectly filled in, as can be seen in Fig. 7.3. After registration is complete, the user will be forwarded to the Settings page.

Settings

The Settings page can be accessed through the drop-down menu on the right side of the navigation bar, or automatically after registration is completed. From here, the user could set up their profile, set preferences, or delete their account.

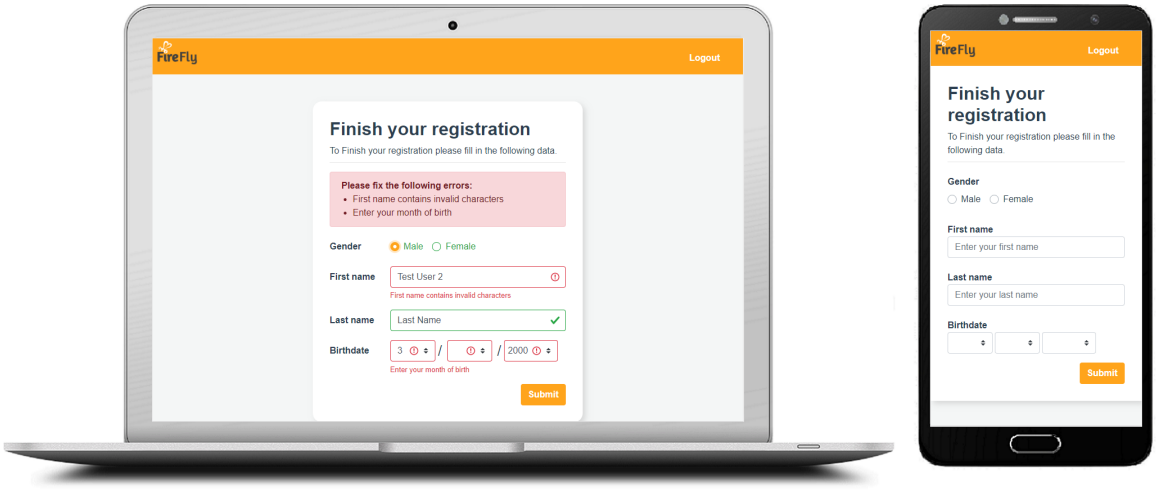


Figure 7.3: Registration Page of FireFly Dating

■ *Preferences*

The preferences contain the user's preferences used for swiping and dating. From this panel, Fig. 7.4, the user could alter their gender interests, desired age range, and languages the user can date in. When these fields are not filled, the user cannot take part in swiping or be considered for matching.

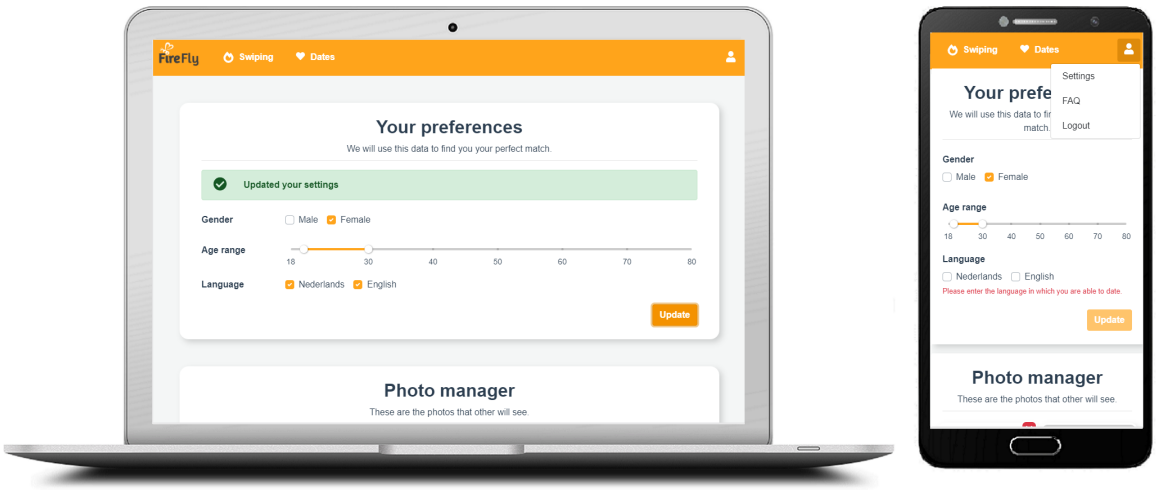


Figure 7.4: Preferences within the Settings Page of FireFly Dating

■ *Photo Manager*

For a user to be visible to other users while swiping, the user must have at least one photo uploaded. The user can upload photos on the settings page, see Fig. 7.5. A user can add a photo by clicking on one of the empty upload icons. A user could also upload multiple photos at once. Errors will be displayed on the screen if, for any reason, the photos cannot be uploaded, for instance, if the file size is too large. Once at least two photos are uploaded, the user could change the order of their pictures by dragging one to the desired frame. Images can be removed by pressing the cross on the top right of the image.

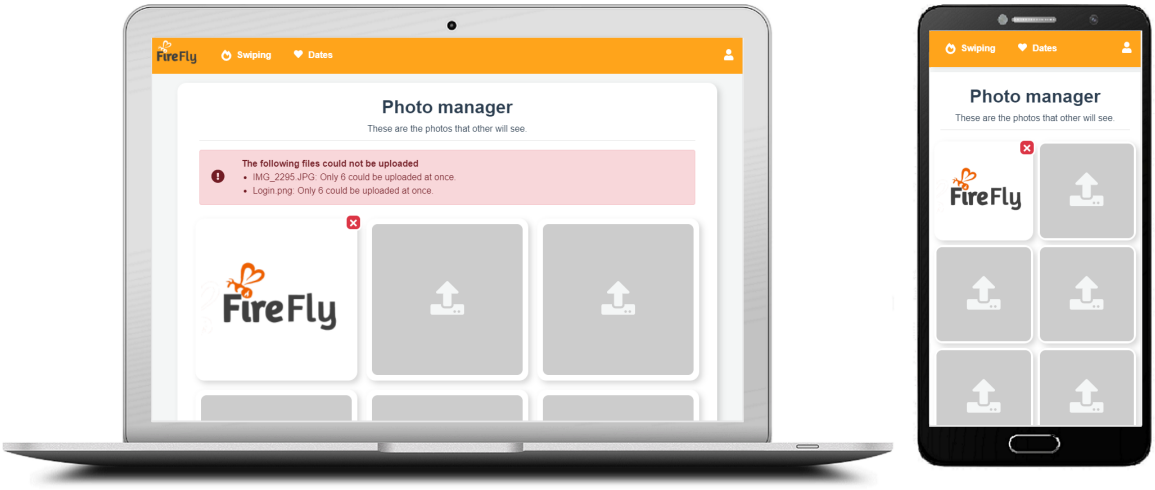


Figure 7.5: Photo Manager within the Settings Page of FireFly Dating

■ ***User Information***

The user information shows the personal information of the user, which is visible in Fig. 7.6. Some fields in this menu cannot be changed. These unchangeable fields were filled in at the Registration page. A user could set a nickname, which is the name other users will see when they come across the user in any way. The height of a user could also be set. However, as of now, this information is not being used elsewhere. To complete the profile, a user could also set their biography. This biography is the message other users will see in the biography section when they see the user during swiping. Lastly, at the bottom of the Settings page, a user has the ability to delete their account. The deletion of an account will remove all the information the user has ever given, adhering to the privacy policy. By pressing the delete button, the user needs to re-authenticate to confirm the delete account action, to prevent undesired account deletions.

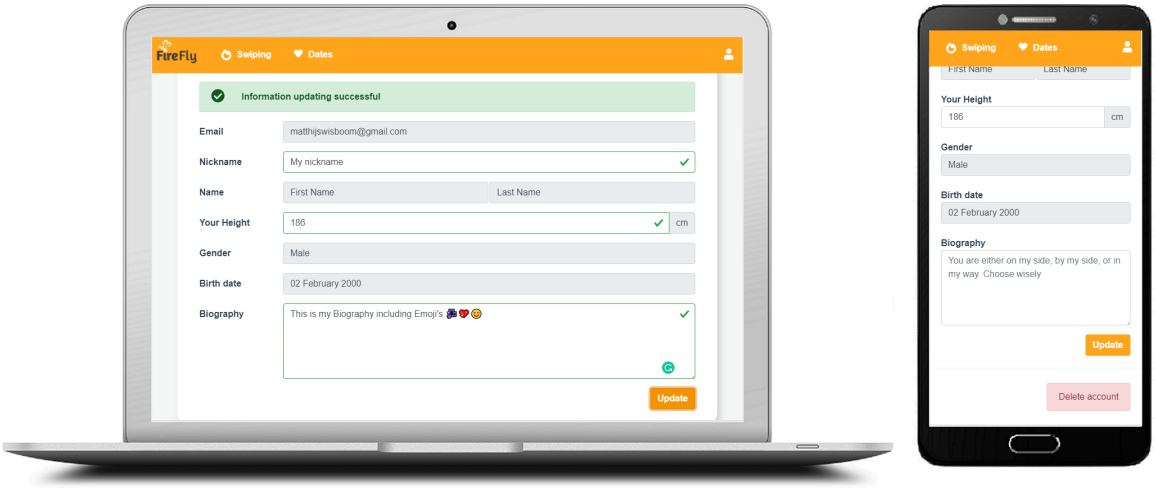


Figure 7.6: User Information within the Settings page of FireFly Dating

Swiping

After a user successfully logging in and has already correctly registered, the user is forwarded to the Swiping page, as seen in Fig. 7.7. A user can only start to like or dislike other profiles when the preferences have been set correctly in Settings. If these settings are incomplete or there is no profile left that satisfies the preferences, a message will be shown to them.

The Swiping page has been set up to gather useful matching data. Whether or not a user likes or dislikes a profile indicates the dating algorithm in which other profiles the user is interested. Therefore it is helpful for the user to swipe. Every swiping window displays another user's profile that fits the user's preferences set on the settings page. From this window, the user can scroll through the pictures, read the biography, report the profile, or like/dislike the profile. After a profile has been swiped, a new profile is loaded.

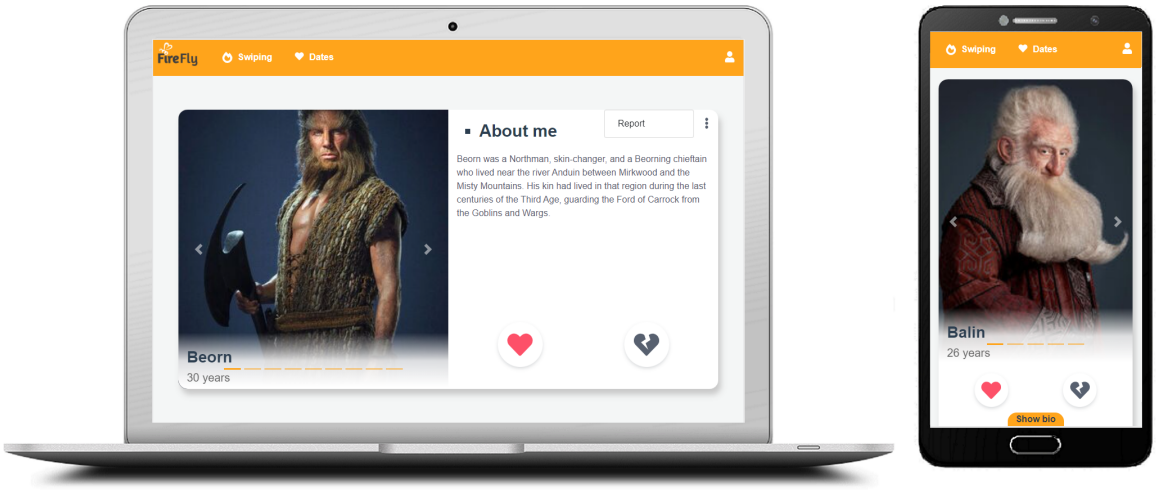


Figure 7.7: Swiping Page of FireFly Dating

Dates
Once the user enters the Dates screen, they will be presented with the Availability screen. Availability, dates, and enrollments are shown on the laptop screen's left-hand side, or the top of the phone screen like is visible in Fig. 7.10. The dating process is one of eight states. These states are indicated by icons, and the meaning of said icon is explained in Fig. 7.9.

■ ***Availability***
The 'plan a date' dialog will be shown to the user once entering the Dates screen via the navigation bar, visible in Fig. 7.8. This dialog allows the user to enroll for a date. The Time Slots shown are fetched from the database that occur in the upcoming two weeks, starting two days from the day the page was entered. This way, once enrolled, a user does not have to wait a long period to finally go on a date. The user has to set a minimum of Time Slots that they are available for a date and at least one location where the date could be located. The data is then be used by the matching algorithm so that the participants could be considered for the next dating rounds.

■ ***Information***
Once a user has a pending date with another user, they have the ability to accept the date by selecting the date in the dates panel on the left of the laptop or top of the phone and either accept or cancel a date in the future as shown in Fig. 7.10. This date will only show as long as it still has to take place. A date can only be canceled once the date has been accepted. When either party of the date has canceled, both parties will see the broken heart icon. They will also both receive an email stating the date has been canceled. Parties will also receive emails when the state of a date changes from searching to a heart, meaning a date has been found and could be accepted. The last confirmation mail the parties will receive is when the heart changes from gray or yellow to red, meaning both parties have confirmed the date. An overview of every icon is shown in Fig. 7.9.

■ ***Feedback***
After a date has concluded, meaning the time the date took place is in the past, the user is confronted with the empty dialog icon on the dates screen. Once clicked, a small questionnaire about the date

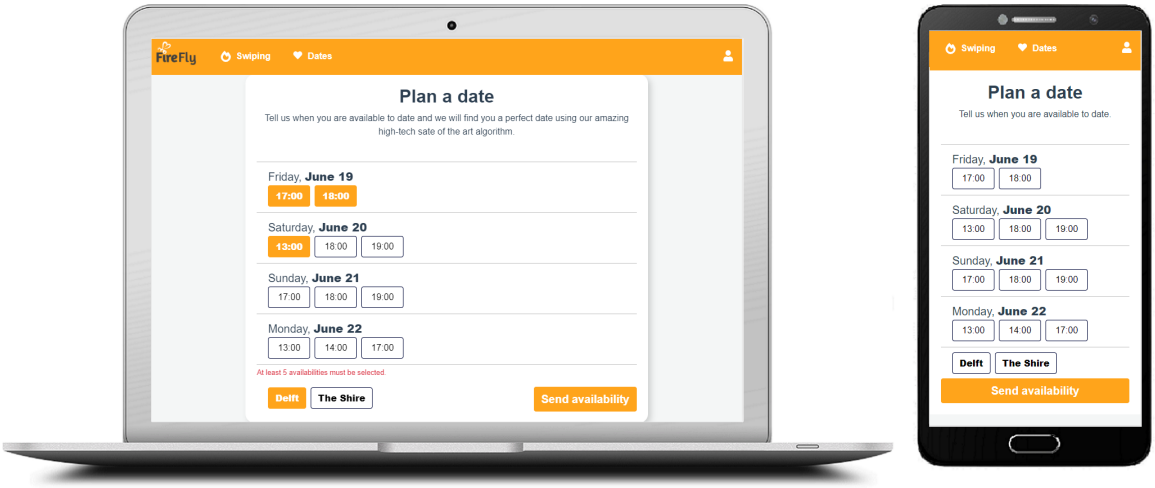


Figure 7.8: The dates page where you could set your availability of FireFly Dating

gray heart
This icon means that both users of a date have not accepted the date.

Red heart
This icon means that both users of a date have accepted the date and are thus ready to go on said date.

Yellow heart
This icon means that only one user of the date has accepted. When the heart is pulsing, it means the other user has not accepted yet.

Broken heart
A broken heart means the date has been canceled and will not take place.

Empty dialog
An empty dialog means the user can leave feedback on a date that took place.

Filled dialog
A filled dialog is shown when the user has provided feedback and is waiting for the other party to also give feedback.

Searching
This icon means the user has enrolled for a date and will be included in the next matching of dates.

Figure 7.9: The Various Date States

will be presented, as shown in Fig. 7.11. The user feedback has three outcomes: The first outcome is when the user indicates the date was either good or bad but did not want to report the other user or send them a message. The second outcome is the option to report the other user. Reporting should be done when there was a problem with the date or if the date never showed up. After such a report has been filled, an Administrator could review the report and take further action, more of this function will be elaborated in Section 7.1.2. The last outcome is when the date went well. If this is the case, the user can send the other user a message of their liking, such as sharing contact information. These messages will only be emailed to each other when both parties have included a message.

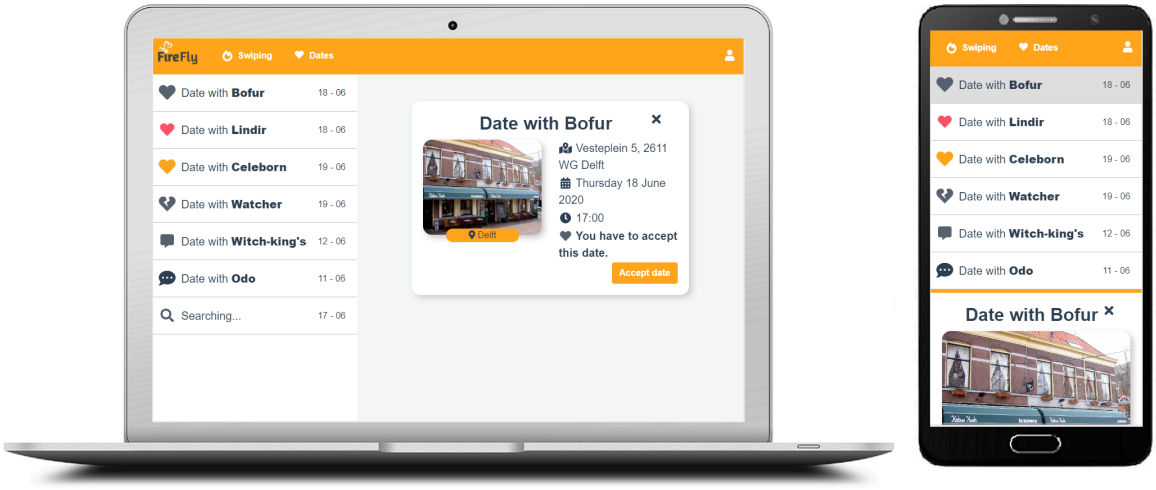


Figure 7.10: The dates page where a user can view their pending dates in FireFly Dating

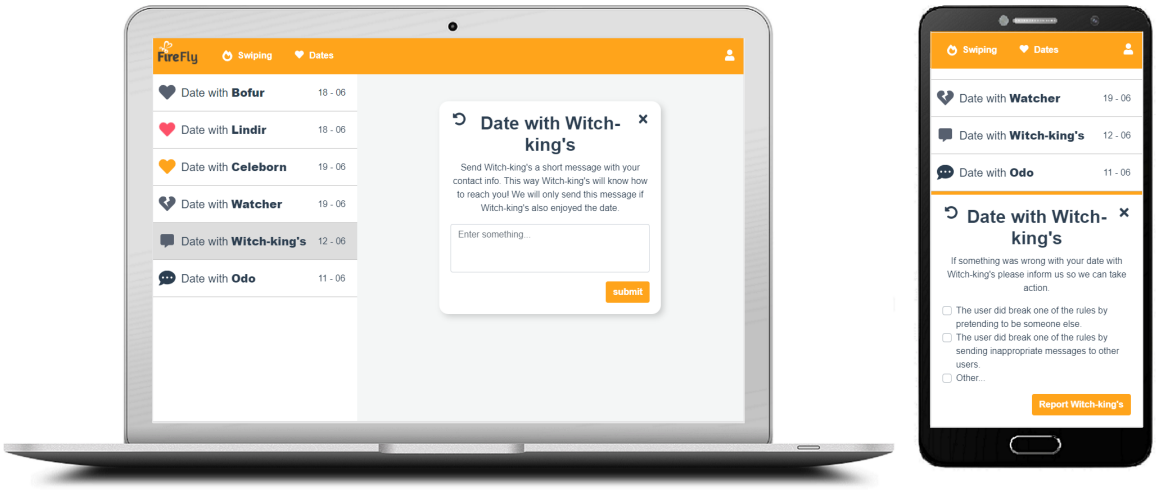


Figure 7.11: The dates page where a user can leave feedback about a date in FireFly Dating

7.1.2. Administrator

The Admin App has been created to allow the Administrator to easily alter data used in the application, take action on misbehaving users, and have a general overview of the statistics. An Administrator can access the Admin App through $fireflydating.app/admin$ using an Administrator password. The screenshots for all the screens are located in Appendix H.

Dashboard

The dashboard is the main page of the Admin App, which can be viewed in Fig. H.5. The page gives statistics on the data from the application. These statistics include:

Total users - The total amount of users registered to the Client App.

Total Pictures - The total amount of user pictures uploaded to the Client App.

Total Swipes - The total amount of likes and dislikes that have been given on the Client App.

Age of users - A bar graph representing the age of the users.

Gender Ratio - A pie chart displaying the ratio between male and female users.

Gender Interest Ratio - A pie chart displaying the ratio between gender interests of users.

Establishment Location Partition - A pie chart displaying number of Dating Establishments per location.

Establishment availabilities - A bar graph of each Dating Establishment containing the number of time slots assigned, the total number of dates hosted/assigned, and the total number of tables assigned to the time slots combined.

Matching
The matching panel can be used to control and view the matching algorithm's results, as shown in Fig. H.4. From this panel, an Administrator can force the matching algorithm to run, outside of the regular interval. This force running is mainly meant for testing purposes. After matching has concluded, the list below will update with the latest dates that have been created. This table has references to user ids and date id to find the entry in the database if needed. The refresh button is used to load the dates list from the server.

Establishments
The establishment panel allows an Administrator to add, edit, search, archive and restore Dating Establishments for the application, see Fig. H.3. All changes will be directly updated to the database. Also, the same pie chart as described in Section 7.1.2 is shown here.

Time Slots
The Time Slots panel allows the Administrator to add, edit, search, archive and restore Time Slots. As well as linking Time Slots with Dating Establishments and allocate the number of available tables to these entries. The changes made here will directly be updated to the database. Furthermore, the same Dating Establishment availabilities graphs as described in Section 7.1.2 are shown here. A screenshot of this page can be viewed in Fig. H.2.

Reports
The last panel in the Admin App allows the Administrator to view users that have been reported, called report cases, shown in Fig. H.1. An overview of the user that has been reported is shown on this page, including all user information, preferences and photos. The Administrator will see the report cases one by one, receiving the user with the most pending reports first. From this page, the Administrator can view the reports that have been submitted, compare them to the users information and ban the user if the Administrator desires. If the Administrator does not ban the user then all their reports are discarded. From this screen, the report types users can choose from can also be added, edited, archived or restored.

7.2. Completed requirements
At the start of this project, a list of requirements was setup. These requirements include all the features that could eventually be implemented in the application. A full list of these requirements can be found in Section B.2. For this section, the achieved functional and non-functional requirements are presented, and a conclusion will be reached whether this BEP has provided a sufficient implementation.

7.2.1. Functional Requirements
The following subsection will include a list of every feature that has been implemented from the requirements list from Section B.2.2. All *must haves* have been implemented, as well as $14/27$ *should haves* and $3/21$ *could haves*. There have been some features implemented that are not tracked by the initial requirements but were essential in making the application more conform. In Section B.3, the usage as a payment system of PSPs is researched. However, the payment requirement was later discarded by the Product Owner and was therefore not implemented in the final application.

Must Haves
The "must haves" section contains the most critical parts. Without these parts, the application would not function, and the project will fail.

- As a user, I must be able to register my account so that I can use the app and store my data.
- As a user, I must be able to login to the application so that I can use the application.
- As a user, I must be able to set my general information such as name and birth date so that I can be matched with other users of about my age.
- As a user, I must be able to upload photos so that I can use these on my profile.
- As a user, I must be able to edit my biography so that I can tell other users about myself.
- As a user, I must be able to set my gender so that the matching algorithm can use this data.
- As a user, I must be able to remove my account so that all my data is gone.
- As a user, I must be able to swipe a candidate so that I can indicate my preferences.
- As a user, I must be able to view details like biography and images of the candidate so that I can make a more informed choice on my preferences.
- As a user, I must get candidates that follow the preferences I set so that it is a better candidate.
- As a user, I must be able to set the location(s) at which I am able to go on date so that I can make sure that I am able go there.
- As a user, I must be able to set the languages I am able to date in so that I get better candidates.
- As a user, I must be able to set my age range preferences so that I get better candidates.
- As a user, I must be able to specify what genders I am interested in so that I get correct candidates.
- As a user, I must be able to provide feedback after the date so that I can indicate if it was a good match.
- As a user, I must be able to the option to share my contact information so that we can continue the conversation after the date.
- As a user, I must be able to specify in what fixed time slots I can date so that I can be on the date.
- As a user, I must get candidates that fall in my specified time frame so that I have the time to go on a date.
- As a user, I must receive a reservation when I am matched so that I know that I am going on a date.
- As a user, I must be able to cancel a date so that the date can be cancelled.
- As a Dating Establishment Owner, I must receive an e-mail with the date information for dates planned at my establishment so that I know that users are coming.

Should Haves
”Should haves” are high-priority features. These features are not essential to launch but are improvements on the app.

- As an administrator, I should be able to review the report cases so that I can deal with them.
- As an administrator, I should be able to ban any account for breaking the rules so that the platform stays safe.
- As an administrator, I should be able to add new establishments to the system so that the platform can expand.
- As an administrator, I should be able to set time slots available for establishments so that dates can be scheduled.

- As a user, I should be able to verify my e-mail address so that my account is verified.
- As a user, I should be able to report the other users for untruthfulness if this is the case so that this user can be punished.
- As a user, I should have a way to contact customer support via e-mail.
- As a user, I should not be able to receive inactive candidates during swiping so that I do not waste time on them.
- As a user, I should be able to set my height so that the matching algorithm can use this data.
- As a user, I should not see the same candidate multiple times so that I do not have to swipe the same person multiple times.
- As a user, I should be able to set how many dates I want when choosing the time frames so that I can go on multiple dates in one matching session.
- As a user, I should have a FAQ page about the app so that I can look for answers about the app.
- As a Dating Establishment Owner, I should be able to provide availability for a date so that the system knows if the location is available.
- As Dating Establishment Owner, I should receive a notification if a date is canceled so that I can take this into account.

Could Haves
”Could haves” are features that are nice to have but not needed. This category is of lower importance than the ”should haves” group.

- As a user, I could chose to register using a third party authentication so that I do not have to make a new account.
- As a user, I could have questions to help me write my biography.
- As a user, I could be matched with someone even if the matching score is minimal, i.e., when both users disliked each other.

7.2.2. Non-tracked Features
These are features that have not been established in the initial requirements but have been implemented to make the application more conform.

- As a user, I could receive an email when a planned date has been canceled so that I know my date is not going to happen.
- As a user, I could accept my date so that my date knows I have received the notification about the planned date.
- As a user, I can add emojis to my biography and feedback messages so that I can express myself better.
- As a user, I could set a nickname so that people do not see my real name in the application.
- As an administrator, I could view various statistics about the application’s data so that I have an overview of the application’s data.
- As an administrator, I could archive certain entries in the database so that I do not need to permanently delete said entries.
- As an administrator, I could easily read, update, and archive Dating Establishments so that I do not have to use an external program to do so.
- As an administrator, I could easily read, update, and archive Time Slots so that I do not have to use an external program to do so..

7.2.3. Non-functional Requirements

In this section, we reflect on the non-functional requirements set in the research report, Section B.2.3.

Accessibility

The Client App is easily accessible on a mobile device or personal computer. The Client App has been developed in a mobile-first and scales well across various screen sizes. After user testing, see Section 6.3, it has been concluded that the application can be used with ease without a technical background. The system is responsive, and only small data packets are sent by each request. The only limitation is the inability to use the application on Internet Explorer as VueJS does not support it. However, as this is an older browser, we believe it will not significantly affect the users' accessibility.

Agreements

The BEP client has set the requirement that users need to have at least one drink at the Dating Establishment. As of now, this is communicated to the user via the FAQ. Once an official Terms of Service (TOS) has been created by the client, this can be easily added as confirmation to the acceptance of a date. Also, the system allows participants of a date to share contact information through the application when they both experience the date as positive instead of forcing them to share contact information on the date.

Scalability

The non-functional requirement of the application is met. New locations and Dating Establishments can easily be added to the Administrators application to allow the application to grow. If the application becomes a success, and the website gets stressed, multiple websites could easily be set up to mitigate the traffic between different servers. The same goes for the Admin Apps that could be hosted on a separate website to reduce traffic to a single website host. As far as it goes for the database, it was tested that this handles large amounts of users efficiently. Thus all the different parts of the applications are made with scalability in mind, and scaling will be no issue for the Administrators and Product Owner.

Stability

The stability requirement is met since errors that occur are not shown to the user. The requested data is merely unavailable at that moment, and by refreshing the data could be requested to solve it. An API is implemented, and said API could efficiently utilize multiple versions[1]. Thus the older software does not have a compatibility problem with the newer released software.

Security

The security aspects of the system are met. There are a different set of routes that guard unauthorized access to the system, see Section 5.1. Each library implemented is the latest stable version of said library, thus having limited security vulnerabilities. Routes that provide any private information can only be seen by the adequately authorized user or authorized Administrator. Private information is handled with care, and no memories are being kept in the memory, they are all safely stored inside the database, if the information does not need to be stored, then it is discarded. Before the final release of the software, all the rules created by the General Data Protection Regulation (GDPR) should be checked appropriately. The project tries to be as much as possible compliant to these set of rules, but it is likely that some rules were left unnoticed and are not adequately implemented in the product.

Maintainability

Version control is used to record the files' changes, such that older versions of that file could still be recalled. When an error occurred, the team could go back to an earlier stage and start rebuilding the requested feature.

The feedback from the Software Improvement Group (SIG) was also used to increase maintainability. The codebase fulfills a strict lint verification that increases the readability, helps prevent bugs, and prevents structural errors, see Section 11.2.

[1] https://www.xmatters.com/blog/devops/blog-four-rest-api-versioning-strategies/

Migratable
To make the migration to the smartphone possible, the team has taken the mobile-friendly front-end designs into account. In this way, the designs can easily be transformed into mobile applications. By separating the front-end from the back-end, the team created the opportunity to reuse the back-end calls when the front-end screens have to be migrated.

Uptime
The back-end is deployed on PM2 to ensure that in case of an error, the server will be restarted[2]. Almost all of the back-end functions have, where needed, extensive catch cases to ensure that the process keeps running. The computational intensive matching algorithm is only run at night to reduce strain on the system and prevent it from going down due to being overloaded.

Robustness
Substantial effort has gone into securing the robustness of the applications. Therefore, unless the back-end is shut down completely, the application will work without crashing. The user is also informed of errors from the server so that the user knows a problem occurred. The application can also handle erroneous input for any requests to the server by verifying the data in the back-end.

Documentation
All methods used in the system code have been commented, and a guide to get the application running is described on the GitHub page of the product. Variables have been appropriately named not to cause any confusion.

[2] https://pm2.keymetrics.io/

8 Ethics

This chapter will discuss the ethical implications of the project and the final product of this BEP.

FireFly Dating has been developed for FireFly. The product has been created for deployment as a blind dating application. The complete implementation is wholly owned by FireFly, and the developers are not eligible for any profit the application will generate in the future. All developers are not entitled to reuse or sell any of the software in projects outside those of FireFly.

FireFly Dating is used to match people and create blind dates at Dating Establishments. The users being matched need to log in through Firebase using either a Google or Facebook account, see Section 5.1. Therefore, FireFly Dating does not store any users' passwords.

The application does store personal information such as email, name, nickname, date of birth, gender, length, biography, sexuality, age preference, user photos, and both swiping and date feedback. The application uses an SSL connection to encrypt incoming and outgoing data, ensuring the safety of user information. Furthermore, only a user can access their data. The exceptions are swiping and date information like biography, first name, nickname, profile photos, age, and sexuality. This information is only used in swiping profiles and displaying a name for a matched user. Other data stored in FireFly Dating is information regarding the Dating Establishment. This data is provided by the Dating Establishment and includes, for example, name, email, phone, and location. However, this information is not private and could easily be retrieved online.

The use of this personal information might impose on privacy regulations. Therefore, a legal team should set up a Terms of Service agreement for the user to accept before completing the account registration.

9 Process

In this chapter, we will explain the methodologies and the processes used to remain productive throughout the project. First, there will be an explanation and review of the communication methods used by the team and the BEP stakeholders. Secondly, we will explain the effect of working with the Scrum method during the BEP. Finally, the initial planning made in the first week of the project will be reviewed and show that the team strictly followed the initially set goals.

9.1. Communication

Due to the global pandemic that is happening during the time of this BEP project [17], team communication and collaboration were different than normal circumstances. The members have not seen each other in person for the entire course of this project. During this time, all team members needed to adjust to a remote working style with which they previously had minimal experience.

9.1.1. Meetings with Product Owner

To ensure that the product was developed according to the Product Owner's wishes, the team had regular meetings with the Product Owner to discuss the progress. Meetings were usually planned a few days ahead of time and were held via Google Hangouts. Due to these meetings, the final result fits the Product Owner's needs and can be considered a successful BEP project.

During these meetings, there was a person who was assigned to record the minutes of that meeting, as described in Section A.3. The recording of minutes was done effectively by the team. All meeting minutes were stored on Google Drive.

9.1.2. Communication with Coach

The TU Delft coach guided us through the expectations that were set by the education program. Meetings between the team and the coach were also held on Google Hangouts. However, it was not necessary to hold these meetings on a weekly bases because the BEP was progressing smoothly without any problems that needed consolation of the coach.

9.1.3. Team Communication

Team communication happened through two communication channels, namely: Discord and WhatsApp. Discord was the primary means of communication during the workday, whereas WhatsApp functioned as the primary means of communication outside of work hours.

Office Hours

As described in Section A.3, the team had decided to hold work sessions from 10:00 till 18:00. During this time, all members were inside a Discord voice channel with each other. By doing so, they were able to communicate with each other easily. This method of being in constant communication with each other actively promoted collaboration within the team. All team members believe that had we not worked in such a manner, productivity would have been much lower.

At the start of the project, it was decided that if a team member were late without reason, there would be a punishment of having to put a fine into a virtual jar. In the end, this system was no longer required, as all team members arrived on time. There were only a few minor occurrences. Therefore it was decided to disregard this rule.

9.2. Team Division

In this section, the division of labor between the team will be stated. Each team member had a specific end role that was his main task during the project. Each team member contributed equally throughout the project.

Colin Geukes
Colin had the role of lead testing. In his role, Colin was end responsible for the state and the quality of the tests in the project, as well as the static analysis.

Colin was a front- and back-end developer for the project. His primary focus was on the implementation of the swiping, the admin panels, and the candidate selection algorithm.

Caspar Krijgsman
Caspar had the role of lead programmer. Being lead programmer means that he was responsible for the quality of the code implemented in this project. He made sure that all code is thoroughly inspected to meet the project standards.

Caspar was a back-end developer for the project. His primary focus was implementing back-end code, including the server, matching algorithm, and database structures.

Steven Lambregts
As a team leader, Steven ensured that all team members were present during the meetings. He motivated the team to keep up the excellent work. Steven inspired, where needed, others to share their opinion about the corresponding subject. Furthermore, he communicated the company goals and the deadlines to the team.

Steven worked on setting up email interactions for the application and allowing users to upload photos to the application.

Vincent Wijdeveld
Vincent had the role of lead UI/design. He transformed the initial ideas of the Product Owner into a design that would eventually become the final look of the application.

Vincent, therefore mainly worked on front-end development. Vincent also had the role of Scrum master. He ensured daily meetings were kept so that every member was up to date with each other.

Matthijs Wisboom
Matthijs had the role of lead communications. He was the primary contact point of the Coach, and if needed, he will reach out to the coordinator.

During development, Matthijs worked in front- and back-end development. He mainly focused on the integration between front- and back-end. Matthijs worked on the Settings page, as well as much of the integration for the Dates screen.

9.3. Scrum

Throughout the development of this project, the team made use of the Scrum methodology for working in an agile process. The Scrum methodology has provided us with a structure to continually focus on what tasks are essential and how to improve our ways of working. This section describes what Scrum activities were performed and how they affected the progress of the development.

9.3.1. Backlog

At the start of the project, a backlog document was created as an overview of the requirements. The list of requirements can be found in Section B.2.2. Some non-essential features on that list that have not been implemented in the final application are redacted on the request of the Product Owner. The requirements were organized according to the MoSCoW method. There was a conscientious effort to

focus on first completing the must-have requirements rather than working on the tasks that were the most convenient at the time.

The backlog that was created was translated to a list of GitHub epics [1]. Based on these epics, the team created the sub-tasks that were required to complete each epic.

9.3.2. Sprint Planning

The Product Owner gave the team the freedom to decide what order tasks should be implemented. The Product Owner's only condition was that the product must have the required set of must-haves at the end of week eight of the development period.

During the weekly sprint planning meeting, the team set what task should be completed by the end of the week. The enrollment for tasks often included selecting relevant epics and creating the required tasks to complete these epics. Doing this was essential to ensure high productivity throughout the week

9.3.3. Daily Scrum

As part of the Scrum activities, we performed the daily Scrum each morning at around 10:10 when all team members had arrived and settled in. Each member shortly described what they were working on and what issues they wanted to do next or needed help on. Allowing all team members to have a good overview of the general status of the project. There were some days when the team forgot to perform the daily Scrum. During these days, there was an evident lack of focus and purposeful development. After evaluating this in the weekly retrospectives, the team was more determined to perform the daily Scrum activities for optimal team functioning.

9.3.4. Review and Retrospective

The review and retrospective part of the Scrum methodology was observed in a slightly adapted manner to suit the team's working style. Both were often combined into one session. During the weekly sprint review, the team discussed that week's progress and reviewed what tasks were still pending to achieve the goals set during that week. If necessary, those tasks were then moved to the next week. Traditionally a sprint review includes a demo. Providing a demo was, however, usually not performed during the weekly sprint review. The reason being that only the development team was present during these meetings. The demo part was moved to whenever the next meeting with the Product Owner was scheduled.

During the weekly retrospective, the team discussed how the team performed during the sprint. The parts that went well and the parts that could be improved were evaluated. Then based on these discussions, the team could implement changes and improve the following weeks.

9.3.5. Tracking

Throughout this BEP, GitHub was the version control host of choice. GitHub provides excellent tools for issues tracking and management.

The initial backlog was created in Google Sheets. Working with Google Sheets worked for a first version but could not be maintained for the entire project. As previously described, the product backlog was stored in GitHub using GitHub issues [2]. The tasks from the backlog functioned as epics.

The team made use of GitHub Boards[3] to track the progress of the weekly sprint. The board was used with Agile Software Development [18]. Instead of creating significant complex issues, each significant issue was split into smaller workable issues [18]. These issues could be worked on in parallel, meaning multiple developers could be working on the same epic issue, but have distinctive sub-issues.

During the sprint planning, the issues that should be completed were moved to the "Current Sprint" column in the sprint board. After a team member has been assigned to an issue and has started working on it, the relevant issues card can be moved to the "In Progress" column. When the team member has completed this task and has created a pull request, the issues should be moved into the "Reviewing" column. Finally, when the relevant pull request has been approved and merged into the main branch, the issue can be closed and moved into the "Done" column.

[1] https://www.zenhub.com/blog/how-to-use-epics-and-milestones/
[2] https://help.github.com/en/github/managing-your-work-on-github/about-issues
[3] https://help.github.com/en/github/managing-your-work-on-github/about-project-boards

Using this system of columns allows for a clear overview of all tasks currently being worked on, providing the team with a sense of direction and progress.

9.4. Review of Original Plan

During the first week of this BEP, the team created a project plan which described an overview of what tasks should be completed. Overall the team managed to stay reasonably close to the original planning and did not have to deviate much. This initial plan can be found in Table 9.1.

Week		Tasks	Deliverables
1	April 20 - April 24	Project plan Start Research report	April 23 - Project plan
2	April 27 - May 1	Finish Research report Setup work environment	May 1 - Research report
3	May 4 - May 8	Discover frameworks Start Database design Setup testing environment	
4	May 11 - May 15	First design implementations Back-end systems	
5	May 18 - May 22	First visual working demo	
6	May 25 - May 29	Wrapping up integrating front- and back-end	First SIG upload
7	June 1 - June 5	Minimal Viable Product (MVP)	
8	June 8 - June 12	Testing on audience	
9	June 15 - June 19	Process feedback from testing	Second SIG upload
10	June 22 - June 26	Finalize report	June 24 - Final Report June 24 - Info Sheet June 29 - Final deliverable
11	June 29 - July 3		Presentation

Table 9.1: Original planning made in week 1 of the project

Week 1 & 2

The first two weeks went according to plan. During this time, we completed the Project Plan, Research Report, and finished setting up the work environment. The team also managed to start the setup of the testing environment.

Week 3 to 6

These weeks spent were the first phases of actual development. During this phase, the application transitioned from nothing to almost a working MVP using the Scrum methodology. During the initial planning, it was expected that the front- and back-end implementation would be separated. However, it was possible to implement the front- and back-end alongside each other and connect them when the necessary API calls were implemented. In week three, there were already visual components shown to the Product Owner rather than the first visual demo given in week five. Throughout the remainder of this period, the team was able to implement all features required for a functioning MVP similarly.

Week 7
This week the MVP was completed, and a demo of it was provided to the client. In this MVP, it was possible to do a complete all be it crude walkthrough of the application. It was possible to show the crucial features of the application.

Week 8 & 9
During weeks eight and nine, the project was finalized, and the first steps for the final report were taken. In these last weeks of development, the team focused on completing the final fixes and improvements required to deliver a product that would function as expected. The team was able to do user testing, as described in Section 6.3. Finally, when all open issues and must-haves were complete, the primary focus was shifted to complete the final report.

9.5. Maintainability
This section will discuss how the team kept the code maintainable by making use of the following practices.

9.5.1. Testing
The code was tested to have at least an 80% unit test coverage. Such a test coverage ensures that the individual units of the project would still be working as expected after code alterations. A more detailed explanation of the testing process can be found in Chapter 6.

9.5.2. Static Analysis
Not only actual testing of the software increase the maintainability, but also static analysis [19]. Reducing the size and complexity of the project result in increasing the maintainability, allowing developers to focus on writing new software instead of trying to maintain older pieces of code [19]. For this reason the project embraced two types of static analysis tools, namely ESLint[4] to conform the project to a correct standard, and JavaScript Copy/Paste Detection (JSCPD)[5] to reduce the amount of duplicated code.

ESLint
In other to increase maintainability, functions should not be too long and too complex. They should be small and simple. Such functions are easier to test and apply [15]. To ensure that this was the case throughout the entire project the team applied a strict set of rules to the linter, similar to the rules of SIG[6]. If a single error was found, then the linter did not accept the code. If warnings and no errors are raised then the code was accepted by the linter. These rules were also applied to the CI, so that each time a developer tried to merge into the main branch it was checked by lint. If it did raise errors then the merge was rejected and the code needed to be changed in order to be accepted.

In the initial stages of the project the linter was not strict. The default applied lint rules were weak. Thus the project did not have a strict set of rules to allow for good maintainability. As a result, each request to merge with the main branch was tied with a large quantity of errors that should have been addressed by the linter. In the middle of the project the lint rules were addressed and fixed. At the end of the project, after the first results were retrieved from SIG it was made stricter once again. In the final stage of the project, the code had a proper structure that allowed for correct behaviour, easy understanding and good maintainability.

The addition of ESLint to the project resulted in code that is: not unreasonably long, this means that the lines are not too long and each function does not have too many lines; not unreasonably complex, meaning that each function has a max amount of branches, input, and scopes; properly documented, as each function must be documented with JSDoc[7]. All these rules make sure that the code is maintainable and developer-friendly. The addition of JSDoc removes the downsides of JavaScript being a loosely typed language [20]. This means that functions can be called with different types of objects. JSDoc made sure that each parameter was properly described, so that using the function was not difficult at all, as it was properly stated what the specific function requires, does, and returns.

[4] `https://eslint.org/`
[5] `https://www.npmjs.com/package/jscpd`
[6] `https://www.softwareimprovementgroup.com/`
[7] `https://jsdoc.app/about-getting-started.html`

Code Duplication
Having the same functionality at different locations in the project is generally a harmful design flaw [21]. If the code is only changed at one location, and the other location is being neglected, then these two parts perform differently, while they should perform equally. This, of course, does not always raise issues, as sometimes these codes should perform differently [21]. However for this early stage project, code duplication does raise issues as the code changes rather frequently. If these changes are not applied everywhere then the projects tends to have flaws in the integration of different components [22].

To address the code duplication and the errors accompanied by it, JSCPD was added to the project. If a piece of code of five lines or more is duplicated, then this raises an error that should be addressed. This code duplication reporter was implemented shortly after the first SIG results. This practice kept the project maintainable.

9.5.3. Code Review
Manual inspection of code was an essential practice during the development, see Section 6.2. It made sure that unnoticed errors or incorrectly implemented functionality were not merged with the main branch. Consequently, the main branch was locked, disallowing the change of files in that branch directly.

In order to update the main branch, team members created a merge request. These merge requests have an automatic template that had to be filled in, so that other developers would know what the merge request was about and whether all the checks had been appropriately conducted. The developer that reviewed the merge request pulls the branch locally and checks the branch's added functionality. If everything worked, then the reviewer would accept the merge request. If something were not right, then the reviewer would request changes and reject. At least two developers, excluding the creator of the merge request, had to approve the merge request before merging into the main branch. If the branch was updated, then all the approvals are discarded. The discarding ensures that all the reviewers that had already approved it must view the merge request again. The issues that the merge requests solve are linked to issues of the project [8].

This system of code review through the use of merge requests had multiple significant advantages to improving the project's maintainability. Firstly, by multiple developers reviewing and inspecting code changes, errors could be caught before entering the main codebase. Secondly, more team members were aware of how certain parts of the code functioned, allowing all members to work more effectively on all parts of the code.

9.6. Conclusion
In conclusion, the team believes that this BEP project was completed successfully. This conclusion is drawn due to the performance of the team and processes that were put in place. Communication functioned effectively, even when working remotely, through Discord and Google Hangouts. By using the Scrum methodology, the team worked in a structured and focused manner. Working with these methods of communication ensured high productivity throughout the weeks. Regarding the original plan made in the first week of the project, the team was able to stick to this plan fairly well and complete the critical tasks on time. To guarantee high code maintainability throughout the project, the team made use of multiple code improvement techniques, namely: static analysis, code testing, and code review.

[8] https://help.github.com/en/github/managing-your-work-on-github/about-issues

10

Conclusion

This project started with FireFly describing their vision for a blind dating application. This application should enable its users to go on dates without wasting time chatting in an online environment. In order to accurately match users with one another, swiping should be implemented. Swiping collects data on the users' preferences to allow for accurate matching.

The product was built from scratch in ten weeks. During this period, the provided designs were reworked, improved, and finally, the system was implemented, tested, and deployed. Every state was discussed and evaluated extensively with the client. During the implementation, a few of the requirements were altered to fit the final product better.

Near the end of the project, the application was ready for user testing. Sadly, this could not take place due to the circumstances surrounding COVID-19. As a supplement for user testing, we simulated user behavior through fake users and their interactions. With these self-performed testing, many scenarios were covered, giving insight into how the application functions. Later, besides the development team, users performed a similar style of testing to provide the team with an outside perspective.

In the end, all must-haves, a couple of should-haves, and even a few could-haves were implemented. All the users that tested the application gave positive feedback. The client indicated to be extremely satisfied with the progress made and the final product.

From all of the points described above, it can be concluded that the FireFly Dating project was successful.

11

Discussion and Recommendation

This final chapter will discuss some parts of the system that could be improved. These improvements are not crucial for the MVP. However, they could improve the software.

An important recommendation is to continue similarly to what is described in Chapter 9. The process proved to be very efficient. Continuing this process will most likely result in a profitable future for this project.

11.1. Future Improvements

In this section, some improvements are provided that can be implemented right away. Most of these improvements are based on testing to ensure that the system has the correct behavior.

11.1.1. Snapshot Testing

In the front-end, Jest[1] is used as the testing framework, see Chapter 6. Jest is a robust framework that works well with automatic UI testing. Jest has a feature called Snapshot Testing[2], which makes it possible to create a snapshot of a particular component and check if it did not visually change.

This feature is currently not implemented in the project as Snapshot Testing is a time-consuming practice. Each component was rapidly evolving into a newer state, making snapshot testing not useful at this time of the project, as each component did not change.

It could have served the project during the time that BootstrapVue[3] was implemented in combination with Vuetify. These are both UI frameworks allowing the creation of remarkable and useful pages. However, sadly, they do not work well together as they both override each other's grid system. This incompatibility would have been discovered early on if Snapshot Testing was fully implemented.

11.1.2. Administrator UI Framework

BootstrapVue looks excellent, and is easier to work with judging from our experience. Vuetify is excellent for working with data. Therefore, BootstrapVue is a better-suited framework for the Client App, while Vuetify is better suited for the Admin App. As stated earlier, BootstrapVue and Vuetify do not work well together, so they should not be implemented in the same part of the application.

Currently, Vuetify is not used in the Client App. Only the Admin App still implements BootstrapVue. It is recommended that BootstrapVue is removed from the Admin App.

11.1.3. Database Integration Testing

The majority of testing of the project was done with Unit Testing, see Section 6.1. Unit tests are good practice for many components. However, these tests are not for the database components. The result retrieved from the database is mocked with the Unit Tests [4]. Meaning that it is currently unknown if the

[1] https://jestjs.io/
[2] https://jestjs.io/docs/en/snapshot-testing
[3] https://bootstrap-vue.org/
[4] https://sinonjs.org/releases/latest/mocks/

database correctly returns information. It could be the case that the database structure was updated, and the test still passes because of fake data to ensure that the database is fully functional. There should be database integration testing implemented.

11.1.4. Python Testing

Only JavaScript files are tested. However, the project has two Python files, one of which is a functional file for creating dates between users. This file should be tested in the future. The file currently houses a basic implementation for the matching algorithm that matches the user based on availability and swiping behavior. However, if more complex connections are implemented between users, it should be properly tested to ensure the correct behavior. As in the end, the actual users of the system do want a proper date. Implementing testing for Python is not a difficult task. As there are many testing platforms for Python available [23].

11.1.5. Stress Testing

The structure of stress testing is already present in the project. However, it is not possible to perform stress testing with multiple active users, see Section 6.4. It is useful to know how the system will handle high traffic situations. Currently, it is not possible to test this because the authentication service of Firebase prevents stress testing. Another login system must be created to allow test bots to access the server and make stress test calls. This access of bots must be carefully implemented, since bypassing the authentication service could cause a severe security issue if done improperly.

11.1.6. Email Client

The email client that is currently used is SendGrid[5], which allows the app to send emails securely. However, this email client is not entirely free. It can only send a fixed amount of emails per day. As a result, it could occur that the application cannot send any more emails without purchasing a subscription. There are free alternatives to SendGrid. However, these alternatives have a less complete feature set. One example of such an alternative is Nodemailer[6].

11.1.7. Admin Login

In the current stage of the project, an admin panel is created with embedded credentials. Thus gaining unauthorized access to this Admin App is not too difficult. A proper login system should be put in place to grant administrators access to all the administrator tools. Allowing multiple personal administrator accounts would be an excellent addition, especially for keeping track of what each administrator does, or changes to the system.

11.1.8. Terms of Service

FireFly Dating contains personal data of its users, and should, therefore, adhere to all applicable privacy regulations. Before the official deployment of the application, a thorough analysis of privacy regulation should be conducted for FireFly Dating to ensure no law has been broken.

11.2. Maintainability

Throughout the development of this BEP, the team has been striving to maintain a high level of code quality and maintainability. This maintainability has been achieved through rigorous unit testing, see Chapter 6, and carefully maintained code improvement practices described in Section 9.5. The team would strongly recommend that these practices continue in future development to ensure the same, high level, code quality.

11.3. Future additions

Some features were not implemented in the final product due to time constraints and external factors such as COVID-19. However, there is already a structure present to accompany these features. All these future additions are recommendations, and it is not required that they are implemented.

[5] https://sendgrid.com/
[6] https://nodemailer.com/about/

11.3.1. User Images
The images the user is allowed to upload are quite small, and the maximum allowed file size is 500 kilobytes. Most images that are made today are more massive than this size limit. It is essential not to overflow the server with large pictures, as it can quickly soak up all the storage. For this reason, the user should be allowed to upload larger images. However, these images need to be compressed on the server, or even on the client-side to reduce the load on the server.

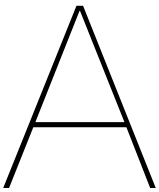
11.3.2. Phone Verification
In the early stages of the project, the Product Owner wanted to include phone verification. Phone verification is possible, as the authentication of the users is handled with Firebase Authentication Service, see Section 5.1.2. This service also include a mobile phone authentication service[7]. This feature is mostly free, as the application is allowed to have 10k users per month authenticated via a mobile device[8]. Since the project already uses Firebase, implementing the phone authentication service should not provide many difficulties.

11.3.3. User Studies
The application design and layout have only been tested by the developers, the client, and a handful of users. Thus it is unclear if the majority of the users that will use the application will be satisfied. In order to test if FireFly Dating meets the user's expectations, it is recommended that a more exhaustive user study should be performed. In such research, a user is given some time, around ten minutes, to explore the application. During this time, the user needs to use the application as they would without any help from the conductor. The conductor will write a report on all their actions. By combining multiple reports, it becomes visible which features are not frequently used and where the users experience difficulties. These user studies provide a view of the user interaction with the system, which is essential to test if the application meets the user's expectations.

11.3.4. Establishment Application
Currently, there are two different front-end applications, namely the Client App and the Admin App. Effectively a third could be created, namely the Establishment App. If created, the Establishment can fill their data, instead of the Administrator. This change releases a massive load from the Administrator, as each Dating Establishment could fill in their data instead of contacting the Administrator. The effectiveness of such a third application is partially described in Section B.3.3.

11.3.5. Time Slot Templates
As described above, the Administrator currently needs to fill in all the data of the Dating Establishments. This task also includes assigning Time Slots to each Dating Establishment. This assigning is a very laborious task, as each Time Slot needs to be manually created and assigned to a Dating Establishment with the number of available tables. It will be ideal if each Dating Establishment has a template that they could fill in per week. If the week is finished, then the same template is used to fill in the Time Slots for the next week. Doing so only requires the Dating Establishment to fill in the Time Slots a single time. This idea, combined with the Establishment App, is an efficient method to generate the new data needed for the entire application to continue running.

[7] https://firebase.google.com/docs/auth/web/phone-auth
[8] https://firebase.google.com/pricing



Project Plan

Bachelor End Project: FireFly Dating

Project Plan

by

Colin Geukes
Caspar Krijgsman
Steven Lambregts
Vincent Wijdeveld
Matthijs Wisboom

Project duration:	April 20, 2020 – July 3, 2020	
Thesis committee:	Dr. H. Wang,	TU Delft, Bachelor Project Coordinator
	Ir. T.V. Aerts,	TU Delft, Coach
	M.S. Salarbux,	Firefly, Client

Some content of this thesis is confidential and is therefore marked in black.

TU Delft

Contents

A.1. Introduction

FireFly Dating is a new dating application that solves a particular issue, this said issue is to minimize the time spent chatting and maximize the time spent on actually dating. This maximization process is the result of the application being centered around blind dates. Instead of continuously communicating online there should have been a date between the persons. Not only does it solve a particular issue in the mobile dating scene, [redacted].

This written project plan is the centralized guideline during the project and will serve as an agreement between all parties, namely the Bachelor Project coordinators, the TU Delft coach, the client and the project team.

In the project plan, information about several different parts is explained. The first part is the Project Scope in which the company and project are described. Secondly, in Section A.3, Project management, different aforementioned parties and the way to create a solid final product are explained. Finally, in Project Timeline the timeline of the project is displayed. In that section, the tasks per week accompanied with the weekly deliverables are shown.

A.2. Project Scope

To understand the reach of this project, we first need to understand the company and its values. We examine the vision of the project owner and elaborate on the features and systems he desires for FireFly Dating.

A.2.1. Company description

FireFly was created for the people that don't have the time to swipe and chat all day, but still want to experience the everyday dating scene. The founders of FireFly believe that dating has lost some of its human element. Most dates start out through an online chat. The founders want to reduce this online factor as much as possible. Therefore they resorted to short blind dates, in which participants are only obligated to have a single drink of choice.

At the start of this project, FireFly was still at the idea, solution and validation stage, with some outlined features. Our goal is to develop a Minimal Valuable Product such that the software can go online in the summer of 2020.

A.2.2. Project description

In this section we explain what the company is about and what the ideas are for the web application we are going to create.

Requirements overview

Here we will list the main top-level requirements for the FireFly project. During the creation of the backlog by the MoSCoW method, these requirements get divided into smaller features.

Account System

Like almost every application nowadays, FireFly Dating will need a user account system. The focus of this system will be on ways to verify users, like e-mail and phone number verification. User accounts must be able to hold personal data and optionally a [redacted].
During the research phase, we will look into the possibility of third-party authentication and its pros and cons.

Matching

An essential part of a dating app is matching. The user interaction linked to matching is swiping, as this is a wildly used method. In order to improve both the matching algorithm and the people visible during swiping, the user must be able to state its preferences like location and gender.

As a match will lead to a blind date, the user will have to pay a small fee before starting the matching action. If the user doesn't get a match, the payment gets refunded.

Matching algorithm

A matching algorithm is incorporated to supply the user with matches. The original matching restrains, proposed by the client, are persons' length and the available timeslots.

During the research phase, we will look into more approaches to achieve suitable matches.

Reservation
Once the matching algorithm has matched two people for a date and time, a reservation to a cafe/bar should be made. The reservation system will have a number of slots at various bars/restaurants where the matching algorithm can choose to reserve a spot. The establishments will receive a e-mail about the reservation and could later be extended via a reservation systems. All participants of the date will be informed of their meeting location, day, and time. If a user is not able to show up due to circumstances, he/she will be able to cancel the date at least 24 hours before the date occurs. Possibly, if the user cancels too many reservations, some form of punishment will be given to discourage cancellation of dates.

Contact and review
A yet to be determined amount of time after the date, a notification will pop up. This notification leads to a contact interface where the user and their date are connected. The user can also review their date to tell others how the date went, and tell the system if they did match or not. The user is also able to report the other user if the one they went on a date with does not match with the one represented in the application. The manner of specific implementation of this feature will further be researched in the research report, as we will need to handle with misuse of the system.

The contact functionality will be open for about 48 hours, such that users can share their contact details and move on to set their relation forward. In the scope of this project, an interface will be displayed where the users can share various contact details. Think about phone numbers, Snapchat, Instagram, etc. Whatever platform the users feel comfortable sharing. In the future, this functionality can be extended to include a chat functionality.

General requirements
The privacy of the users concerns the data that is stored in the application. This data should, under no circumstances, be released to the public. The application must be GDPR compliant. Meaning that, for example, people need to be informed and asked for consent before we can store their data, the data needs to be secure and must be deleted on request.

The application will have a clean user interface such that users can easily access all tools and do not get scared of complicated screens.

The product will be reliable and stable, undesired crashes will be avoided at all cost. If the product tends to crash it will restore from the point where it left and will not block the user from the system somehow. The system runs as expected and will let the user know if something went wrong, all errors will be caught and not be exposed to the user.

MoSCoW method
Below is the list of all the currently researched features divided into the MoSCoW division. As we are still in the research phase of the project, this list might be incomplete and incorrect. A revised and improved version of this list will be placed in the research report.

Glossary

- An *administrator* is a person that has administration rights over the application and is able to check the reports of users.
- A *user* is a person using the application that is not the administrator.
- A *dating establishment owner* is the owner of a restaurant or bar at which a date between users can take place through the usage of the application.

Must haves
The ”Must have” section contains the most critical parts. Without these parts, the application would not function, and the project will fail.

- As an administrator, I must be able to review the report cases so that I can deal with them.
- As an administrator, I must be able to ban any account for breaking the rules so that the platform stays safe.

- As an administrator, I must be able to add new establishments to the system so that the platform can expand.
- As an administrator, I must be able to set timeslots available for establishments so that dates can be scheduled.
- As a user, I must be able to register my account so that I can use the app and store my data.
- As a user, I must be able to verify my e-mail address so that my account is verified.
- As a user, I must be able to login to the application so that I can use the application.
- As a user, I must be able to set my general information such as name and birthdate so that I can be matched with other users of about my age.
- As a user, I must be able to upload photos so that I can use these on my profile.
- As a user, I must be able to edit my biography so that I can tell other users about myself.
- As a user, I must be able to set my gender so that the matching algorithm can use this data.
- As a user, I must be able to remove my account so that all my data is gone.
- As a user, I must be able to provide feedback the morning after the date so that I can indicate if it was indeed a good match.
- As a user, I must be able to report the other users for untruthfulness if this is the case so that this user can be punished.
- As a user, I must be able to provide feedback the morning after the date so that I can indicate the general mood and my sense of safety of the date.
- As a user, I must be able to the option to share my contact information so that we can continue the conversation after the date.
- As a user, I must have a way to contact customer support via e-mail.
- As a user, I must be able to swipe a candidate so that I can indicate my preferences.
- As a user, I must be able to view details like biography and images of the candidate so that I can make a more informed choice on my preferences.
- As a user, I must not be able to receive inactive candidates during swiping so that I do not waste time on them.
- As a user, I must be able to indicate that I only want verified users as candidates so that I feel more confident going on a date.
- As a user, I must get candidates that follow the preferences I set so that it is a better candidate.
- As a user, I must be able to set the location(s) at which I am able to go on date so that I can make sure that I am able go there.
- As a user, I must be able to set the languages I am able to date in so that I get better candidates.
- As a user, I must be able to set my age range preferences so that I get better candidates.
- As a user, I must be able to specify what genders I am interested in so that I get correct candidates.
- As a user, I must be able to specify in what fixed timeslots I can date so that I can be on the date.
- As a user, I must get candidates that fall in my specified time frame so that I have the time to go on a date.
- As a user, I must receive a reservation when I am matched so that I know that I am going on a date.

- As a user, I must be able to cancel 24 hours in advance so that the date can be canceled.
- As a user, I must be able to specify the social media that I will share if I choose to share my contact information so that users only contact me at the desired platform.
- As a dating establishment owner, I must receive an e-mail with the date information for dates planned at my establishment so that I know that users are coming.

Should have
”Should haves” are high-priority features. These features are not essential to launch but are improvements on the app.

- As a user, I should be able to verify my phone number so that my account is verified.
- As a user, I should be able to set my height so that the matching algorithm can use this data.
- As a user, I should be able to chat with the person I went on a date with if we both liked the date so that we can develop our relationship.
- As a user, I should not see the same candidate multiple times so that I do not have to swipe the same person multiple times.
- As a user, I should be notified if and when I matched with someone so that I know I have a match.
- As a user, I should be able to use a digital payment option so that I can participate in a dating round.
- As a user, I should be able to get a refund if the system does not match me so that I can get my money back.
- As a user, I should get candidates that fall in my height category so that it is a more suitable candidate for me candidate.
- As a user, I should have a FAQ page about the app so that I can look for answers about the app.
- As a user, I should receive a bit of information on my date (interests of the other) shortly before the date so that I know who I am meeting.
- As a dating establishment owner, I should receive requests for dates to partake in my establishment so that I can check the availability.
- As a dating establishment owner, I should be able to provide availability for a date so that the system knows if the location is available.
- As a dating establishment owner, I should receive a notification if a date is canceled so that I can take this into account.

Could haves
”Could haves” are features that are nice to have but not needed. This category is of lower importance than the ”should have” group.

- As an administrator, I could review users’ photos to verify them so that they can have the verified status.
- As a user, I could chose to register using a third party authentication so that I do not have to make a new account.
- As a user, I could set my hair color so that the matching algorithm can use this data.
- As a user, I could set my eye color so that the matching algorithm can use this data.
- As a user, I could have questions to help me write my biography.

- As a user, I could [redacted].
- As a user, I could [redacted].
- As a user, I could set the application language so that I can use the application in my preferred language.
- As a user, I could have a chat window that closes after 48 hours so that there is a time window to make decisions in.
- As a user, I could [redacted].
- As a user, I could [redacted].
- As a user, I could have it so that other users' popularity is taken into account when matching so that I find partners that better fit me.
- As a user, I could be matched with users that do not frequently cancel their dates.
- As a user, I could request a reschedule for a canceled date so that I have another chance to go on that date.
- As a user, I could be reminded of the time and location shortly before the date so that I do not forget it.
- As a dating establishment owner, I could have it so that reservations are integrated into my current reservation system.

Won't have
These are requirements that are out the scope of our project, but might be included later.

- As a user, I will not be able to download the app in the app store (ios and android).
- As a user, I will not be viewing any advertisements in between swipes.
- As a user, I will not be able to join multiple date rounds at once.
- As a user, I will not be notified if a date has been found in a timeslot that I did not choose. So I can reschedule other things.
- As a user, I will not be able to set up a second date through the app.
- As a dating establishment owner, I will not be able to control timeslots in the application.

A.3. Project management

To ensure an effective workflow during this project, we define the key responsibilities of each stakeholder. The client is the product owner. His role is to guide the design vision and communicate the requirements to the team. Our coach, Taico Aerts, is the representative of the TU Delft, his task is to uphold the interests of the university. Finally, the team, consisting of five TU Delft students in their final stage of their bachelor, is responsible for the development of the product.

A.3.1. Client

The client of this bachelor project is FireFly, a trademark of DevBux. As DevBux is a proprietorship, this means that Siraadj Salarbux, the owner of DevBux, is our client. Siraadj has already laid some groundwork on the FireFly project in the form of designs and a feature list. As a visionary, he will steer the development to make FireFly Dating live up to his visions.

A.3.2. Coach

Taico Aerts will be our coach during this project. As a coach, he will help us to uphold the interests of the TU Delft. His focus is not on the contents of the project, but on the team dynamics, software development methods, and effectiveness of the team. He is responsible for the approval of deliverables like the project planning and will provide feedback on the final report and presentation.

A.3.3. Project team

The project team consists of five bachelor students of the TU Delft. All team members contribute equal parts to the project, this will be checked by peer reviewing and analytical data from code commits such as the ones GitHub offers. During the beginning phases of the project, all team members will work on research and setup of frameworks together to ensure that all members have a thorough understanding of the underlying research and technology. After the initial setup phase, members will specialize in certain areas of the project. We refrain from assigning too specific focus areas for team members as we wish to remain flexible during the development process and assign tasks as necessary. Team member focus areas:

Team Member	**Focus area**
Colin Geukes	Front-end
Caspar Krijgsman	Back-end
Steven Lambregts	Back-end
Vincent Wijdeveld	Front-end and Matching algorithm
Matthijs Wisboom	Front-end

Scrum

We will work according to the Scrum methodology. The Scrum methodology provides a good working framework for the team members to ensure that progress is being made while remaining flexible enough to implement changes as necessary in the product. The client, Siraadj Salarbux, is the product owner, in communication with him the scrum master will ensure that the product meets all requirements. Vincent Wijdeveld, will act as scrum master during the weekly sprints. The scrum master will communicate with the product owner about the requirements and plan the weekly sprints together with the rest of the team. We will work in weekly sprints, each sprint will have the classic elements of a scrum sprint: sprint planning, daily scrum meeting, sprint review and sprint retrospective.

A.3.4. Meetings

In ten weeks, we will develop a web application commissioned by the client. We planned several meetings with the client and the coach to ensure that we keep on track. Within the team, we have daily meetings to clarify the daily schedule.

Team

Within the team, we have planned a daily meeting at 10:00 am. During this meeting, we will discuss what we are going to do that day and if there are any problems someone encountered. Due to COVID-19 all sessions will use Discord as a tool for communication, in this way, we can to talk and share data without having to meet in person. When someone has a valid reason not to attend the meeting, he is allowed to miss it. However, when someone is late for some reason, they must put a fine in a (virtual) jar, such that we can buy a cake or have a nice excursion. This fine will start at 3 Euro when the subject in question is 10 minutes late. Every 10 minutes after this, 1 Euro will be added to this fine if he continues to be late. The team will work at least 8 hours a day to fulfill the 42 hours a week work stated for this project.

Before the meetings, a team member is also urged to prepare the meeting to ensure it proceeds smoothly. Questions for the coach and/or the client will be discussed and, if needed, the concerning party is contacted by e-mail.

Every week on Monday (or the first business day of the week), the team will meet in a structured form, including an agenda. Mainly, the subjects will include retrospecting the last week and planning what will come the coming week. These weekly meetings will be documented, of which a schedule is created for taking minutes. The schedule for the coming weeks, starting from week 2, is defined as

follows:

Week	Taking minutes
2	Colin Geukes
3	Caspar Krijgsman
4	Steven Lambregts
5	Vincent Wijdeveld
6	Matthijs Wisboom
7	Colin Geukes
8	Caspar Krijgsman
9	Steven Lambregts
10	Vincent Wijdeveld
11	Matthijs Wisboom

Client
We planned to have a weekly meeting such that we can present our progress and, if possible a demo, such that he can share his opinions and give us feedback. These meetings will align with the weekly sprint retrospectives.

Coach
The team and the coach do not have a fixed meeting time. The team will communicate with the coach about their progress and when they would like to have a meeting. During this meeting, the deliverables, and any issues that might have been encountered are discussed. These meetings are online as well and done in the presence of all the team members.

A.4. Project Timeline

The project timeline consists of tasks and deliverables. Every week, the tasks will outline what will be worked on for that week. Deliverables are deadlines that need to be met as required by the Bachelor End Project. The team will be working in sprints using a Scrum methodology. The length of one sprint will be one week.

Week		**Tasks**	**Deliverables**
1	April 20 - April 24	Project plan Start Research report	April 23 - Project plan
2	April 27 - May 1	Finish Research report Setup work environment	May 1 - Research report
3	May 4 - May 8	Discover frameworks Start Database design Setup testing environment	
4	May 11 - May 15	First design implementations Back-end systems	
5	May 18 - May 22	First visual working demo	
6	May 25 - May 29	Wrapping up integrating front- and back-end	First SIG upload
7	June 1 - June 5	Minimal Viable Product (MVP)	
8	June 8 - June 12	Testing on audience	
9	June 15 - June 19	Process feedback from testing	Second SIG upload
10	June 22 - June 26	Finalize report	June 24 - Final Report June 24 - Info Sheet June 29 - Final deliverable
11	June 29 - July 3		Presentation

B

Research Report

Bachelor End Project: FireFly Dating

Research Report

by

Colin Geukes
Caspar Krijgsman
Steven Lambregts
Vincent Wijdeveld
Matthijs Wisboom

Project duration:	April 20, 2020 – July 3, 2020	
Thesis committee:	Dr. H. Wang,	TU Delft, Bachelor Project Coordinator
	Ir. T.V. Aerts,	TU Delft, Coach
	M.S. Salarbux,	Firefly, Client

An electronic version of this thesis is available at http://repository.tudelft.nl/.

Some content of this thesis is confidential and is therefore marked in black.

TU Delft

Contents

B.1. Introduction

To ensure the success of this BEP project, we have researched all aspects of the scope of the FireFly Dating app. All elements of the project have been carefully considered before the start of the implementation. This research report details how we have researched the different aspects of the project. We completed the research in chronological succession. This guarantees that all parts build on each other and use the knowledge obtained in the previous section to make better-informed decisions about the implementation.

We start by defining the problem that FireFly Dating is meant to solve in section Problem Analysis. After defining the problem, we establish a list of questions that need to be answered. After the questions we state the list of requirements, which have been created in collaboration with the project owner. Based on the established list of requirements, we can start to make decisions about the implementation. One of the most crucial elements of FireFly Dating is the matching algorithm. In this research report, we analyze what previous research has been done about matching algorithms of a dating application. We then use this knowledge to define how we are going to implement the matching algorithm in this project. The last part of this research report describes the analysis of different possible frameworks to determine what framework would best suit the needs of this application.

B.1.1. Problem Analysis

There are already many dating apps on the market, many of which work on a swiping basis. Apps like Tinder and Bumble allow matches to chat with one another. Chatting is versatile and can go in various directions. Users of these simple dating apps have different intentions in the use of such services. For some, it is an ego-boost, some are looking for love, a hookup, or just a simple conversation. In practice very few dates come from matches, and if it happens, from the moment the match occurs to meeting in the real world, it often takes multiple weeks. From a Canadian study, 81% of people had less than six dates while using dating apps, of which 40% planned the date after more than two weeks of chatting [24]. While possible, a connection is hard to establish through a chatroom, and most of the conversations end up dead. This manner of dating can be slow and often repetitive.

Other dating sites that allow you to view extensive profiles, where people spend time online searching traits they like in a match also shows its flaws. Users of dating sites spend seven times more time researching his/her date than on the actual date. While it has been shown that a significant portion of people's preference in a significant other is based on experiential aspects rather than search-able aspects like job, income, hobbies, etc. [25].

The FireFly Dating concept is one that tries to encompass these problems of extensive searching or having first to build up a relationship based on text messages for weeks before giving a shot on a real-life date. The FireFly Dating app will try to create a single expectation towards the user. To be able to meet new people and, in the ideal situation, help people find love. Based on an interactive and straightforward swiping method, a user will be able to give the matching algorithm a simple preference, and the application will set up a date between two matches. No repetitive chats, no extensive research before a date. A simple blind date.

B.1.2. Research Questions

Throughout this report, we attempt to answer several research questions. Answering these research questions will guide us to make the correct design and implementation decisions for the FireFly Dating app. Each section answers a main-research question; each of these sections has its own set of sub-questions to support the main-question. As a group, we have decided that we need to answer the following research questions:

Section B.2 Requirements Analysis
What requirements does the client desire for the correct functioning of the product?

- What are the functional system requirements, organized in the MoSCoW method, needed to create an MVP?
- What are the non-functional system requirements needed to create an MVP?

Section B.3 Implementation Research
How will we solve the implementation challenges related to payment, refund and reporting?

- What are the best options to handle payment in an online application?
- How will the application handle refunds?
- How will the application prevent misuse of the reporting system?

Section B.4 Matching Algorithm
How can we create a matching algorithm that accurately matches potential partners?

- How have other researchers attempted to solve this problem?
- How can we implement the matching algorithm?

Section B.5 Framework Analysis
What frameworks would best suit this project for the different areas of implementation?

- What framework is best suited for the implementation of the front-end?
- What framework is best suited for implementation of the back-end?
- What database system is best suited for the scope of this project?
- What testing framework is best suited for testing both the back-end and the front-end implementation?

B.2. Requirements Analysis

In this section, the requirements of the designed system will be analyzed. First, all the stakeholders are stated, accompanied by their interest in the system. Afterward, the functionality of these stakeholders is explained to its fullest in Functional Requirements. Finally, in the last subsection, Non-Functional Requirements, the requirements that are important for the correct development of the system are stated.

B.2.1. Stakeholders

In the final product, there will be entirely different independent parties, the stakeholders. They have an interest in the product, its activities, or the business. The stakeholders for this phase of the product are described below.

User
The first and most important group of the project will be the users of the application. The main objective of these users is to find a date, and therefore they are called the daters. When they start the app for their first time, they do not possess any knowledge of how to use the application. For this reason, the application should be as straightforward and simplistic that every user could use it properly. These daters can run the application on completely independent devices such as their personal computer, laptop, tablet, or phone.

In Fig. C.1, a flowchart is shown in which all steps a user can run through are visualized. The start of the application is on the top right, where a user has to register for an account. After this, the user can set up his account and start swiping if they want. If the user decides to go on a date, they enters a paywall, and the algorithm begins running. If it does not find a matching person or a suitable location for the date, the program will refund the entry fee, and the user can restart the process. If the matching algorithm finds a date and a location, the user receives a reservation and goes on a date. After this date the user accesses a feedback screen in which they can share their contact details or provide other feedback about their intended match.

Dating Establishment Owners
When a match has is created between daters they need a location to go on the blind date. The Dating Establishment Owners are the owners of restaurants or bars at which a date can take place. Before the date, these establishment owners are informed that they should have a spot reserved for the date, to ensure that it can properly take place at the location. These establishment owners will not be in contact with the daters before the date itself. They will only be contacted through the administrator stakeholder.

Administrator
The administrator is the stakeholder that is required to keep the application up and running. They will find establishments at which blind dates could take place. The administrator is the essential stakeholder to increase the scalability of the project. Without new locations for blind dates, the entire project could be at risk of the bottleneck effect because of limited capacity. The administrator could also serve as customer support. The support department could eventually be a separate stakeholder. However, that is outside the scope of the project.

Product Owner
The last stakeholder is the product owner, this entire project is his product, and the product owner has a unique set of objectives envisioned for the project. Completing these objectives will result in a reliable product and a successful, profitable business.

B.2.2. Functional Requirements
This section will cover the aspects that are required to make the program function. For the requirements, we used the user story notation and ranked them according to the MoSCoW method [26].

Must Haves
The ”must haves” section contains the most critical parts. Without these parts, the application would not function, and the project will fail.

- As a user, I must be able to register my account so that I can use the app and store my data.
- As a user, I must be able to login to the application so that I can use the application.
- As a user, I must be able to set my general information such as name and birthdate so that I can be matched with other users of about my age.
- As a user, I must be able to upload photos so that I can use these on my profile.
- As a user, I must be able to edit my biography so that I can tell other users about myself.
- As a user, I must be able to set my gender so that the matching algorithm can use this data.
- As a user, I must be able to remove my account so that all my data is gone.
- As a user, I must be able to swipe a candidate so that I can indicate my preferences.
- As a user, I must be able to view details like biography and images of the candidate so that I can make a more informed choice on my preferences.
- As a user, I must get candidates that follow the preferences I set so that it is a better candidate.
- As a user, I must be able to set the location(s) at which I am able to go on date so that I can make sure that I am able go there.
- As a user, I must be able to set the languages I am able to date in so that I get better candidates.
- As a user, I must be able to set my age range preferences so that I get better candidates.
- As a user, I must be able to specify what genders I am interested in so that I get correct candidates.
- As a user, I must be able to provide feedback after the date so that I can indicate if it was a good match.
- As a user, I must be able to the option to share my contact information so that we can continue the conversation after the date.
- As a user, I must be able to specify in what fixed time slots I can date so that I can be on the date.
- As a user, I must get candidates that fall in my specified time frame so that I have the time to go on a date.

- As a user, I must receive a reservation when I am matched so that I know that I am going on a date.
- As a user, I must be able to cancel a date so that the date can be cancelled.
- As a Dating Establishment Owner, I must receive an e-mail with the date information for dates planned at my establishment so that I know that users are coming.

Should Haves
”Should haves” are high-priority features. These features are not essential to launch but are improvements on the app.

- As an administrator, I should be able to review the report cases so that I can deal with them.
- As an administrator, I should be able to ban any account for breaking the rules so that the platform stays safe.
- As an administrator, I should be able to add new establishments to the system so that the platform can expand.
- As an administrator, I should be able to set time slots available for establishments so that dates can be scheduled.
- As a user, I should be able to verify my e-mail address so that my account is verified.
- As a user, I should be able to report the other users for untruthfulness if this is the case so that this user can be punished.
- As a user, I should be able to verify my phone number so that my account is verified.
- As a user, I should have a way to contact customer support via e-mail.
- As a user, I should not be able to receive inactive candidates during swiping so that I do not waste time on them.
- As a user, I should be able to indicate that I only want verified users as candidates so that I feel more confident going on a date.
- As a user, I should be able to specify the social media that I will share if I choose to share my contact information so that users only contact me at the desired platform.
- As a user, I should be able to set my height so that the matching algorithm can use this data.
- As a user, I should be able to chat with the person I went on a date with if we both liked the date so that we can develop our relationship.
- As a user, I should not see the same candidate multiple times so that I do not have to swipe the same person multiple times.
- As a user, I should be notified if and when I matched with someone so that I know I have a match.
- As a user, I should be able to set how many dates I want when choosing the time frames so that I can go on multiple dates in one matching session.
- As a user, I should not be scheduled for two or more dates on the same day in different cities so that I can partake in all of the dates.
- As a user, I should be prioritized over users that already have a date planned.
- As a user, I should be able to indicate to only date verified users, so that I feel safer.
- As a user, I should be able to use a digital payment option so that I can participate in a dating round.

- As a user, I should be able to get a refund if the system does not match me so that I can get my money back.
- As a user, I should get candidates that fall in my height category so that it is a more suitable candidate for me.
- As a user, I should have a FAQ page about the app so that I can look for answers about the app.
- As a user, I should receive a bit of information on my date (interests of the other) shortly before the date so that I know who I am meeting.
- As a Dating Establishment Owner, I should receive requests for dates to partake in my establishment so that I can check the availability.
- As a Dating Establishment Owner, I should be able to provide availability for a date so that the system knows if the location is available.
- As Dating Establishment Owner, I should receive a notification if a date is canceled so that I can take this into account.

Could Haves
”Could haves” are features that are nice to have but not needed. This category is of lower importance than the ”should haves” group.

- As an administrator, I could review users’ photos to verify them so that they can have the verified status.
- As a user, I could [redacted].
- As a user, I could [redacted].
- As a user, I could [redacted].
- As a user, I could [redacted].
- As a user, I could chose to register using a third party authentication so that I do not have to make a new account.
- As a user, I could set my hair color so that the matching algorithm can use this data.
- As a user, I could set my eye color so that the matching algorithm can use this data.
- As a user, I could have questions to help me write my biography.
- As a user, I could set the application language so that I can use the application in my preferred language.
- As a user, I could have a chat window that closes after 48 hours so that there is a time window to make decisions in.
- As a user, I could have it so that other users’ popularity is taken into account when matching so that I find partners that better fit me.
- As a user, I could be matched with users that do not frequently cancel their dates.
- As a user, I could request a reschedule for a canceled date so that I have another chance to go on that date.
- As a user, I could be reminded of the time and location shortly before the date so that I do not forget it.
- As a user, I could be matched with people with a similar desirability score.

- As a user, I could be matched with someone even if the matching score is minimal, i.e., when both users disliked each other.
- As a user, I could be matched on appearances I like when swiping, i.e., using hair/eye color so that I get better matches.
- As a user, I could be matched with people that share the same interests/hobbies so that I get better matches.
- As a user, I could be able to set my relationship preference so that the matching algorithm can take this into account.
- As a Dating Establishment Owner, I could have it so that reservations are integrated into my current reservation system.

Won't Haves
These are requirements that are out the scope of our project, but might be included later.

- As a user, I will not be able to download the app in the app store (ios and android).
- As a user, I will not be viewing any advertisements in between swipes.
- As a user, I will not be able to join multiple date rounds at once.
- As a user, I will not be notified if a date has been found in a timeslot that I did not choose. So I can reschedule other things.
- As a user, I will not be able to set up a second date through the app.
- As a Dating Establishment Owner, I will not be able to control timeslots in the application.

B.2.3. Non-Functional Requirements

In this section, we will elaborate more on the requirements that are specific for the system and cannot be directly translated into a feature.

Accessibility
The application should be accessible for all types of users; everyone without a technical background can create an account without any trouble. Users with a mobile device will be able to access the application. Even users with a slow network connection will also be able to reach the server since the system will be using Vue, which is a lightweight framework. The data packages that will be sent over the network will be as small as possible since the system will only submit the bare minimum.

Agreements
The users agree to have at least one drink at the planned date with their matched partner. This way, the Dating Establishment Owner will earn money from the date. The users also agree not to share their contact details such that they do not feel obligated somehow. They can share their details in the morning after the date.

Scalability
The system should be able to grow such that it can handle more people and attract a broader audience. In the beginning, the main focus will be on three cities, namely Delft, The Hague, and Leiden. We do this to make sure to have a working version. The owner of the project will be able to expand to more cities after the project is finished.

Stability
The system should be stable and should not crash when a user is using it. Older software is still compatible with newer releases.

Security
The system should have a working security layer that protects the data of the users from unauthorized use of data. The security of the system should be up-to-date, such that no perpetrator hacks the system and retrieves confidential data. All privacy sensitive information should be handled with care on the application. It is not allowed to store these data in an insecure manner temporarily. All confidential privacy information that is not needed any longer should be entirely discarded. The application should be designed in such a manner that it is GDPR compliant.

Maintainability
When the system fails, it should allow itself to be restored to its normal operating state as soon as possible. The defect that causes the system to fail should be corrected after the system restored to its working state, but with the highest priority such that it will not happen again any time soon after. The code evaluation from the SIG can be used in these metrics. The project will be using version control, recording all changes and improvements in the codebase on a single location. The project could always be restored to a stable earlier version.

Migratable
The system will start as a web application. Still, when it starts getting brand awareness, it will be redeployed to a smartphone application such that users can access the application easier. For this matter, the system should be easily transferred to another platform. The migration will, however, not be performed as part of this project.

Uptime
The system should always be up and running even when it needs to be updated; the users should not notice it, or the system should disable the functionality for some time.

Robustness
The system should contain the ability to deal with errors while it is running and stay online when an error occurs. It should be able to handle erroneous input, on which it does not crash because it is safely handled.

Documentation
The system should be documented such that other developers can upgrade or change the system without having them read through all the code. Since the project lasts only ten weeks, other developers will add or change functionality that have been created by the project team.

B.3. Implementation Research
This section will take a few of the features described in the requirements analysis and explore some potential challenges. Here we will answer the questions about the payment system, refund policy, and the report system.

B.3.1. Payment
When building a web application, it is essential to make sure that everything works, but it also needs to create revenue. Our client indicated not to want any advertisements but instead prefers a credit system where features are behind a paywall. This paywall means that the app will need a payment option. Implementing all these payment options yourself is a full-time job. That is why almost all companies use a PSP. These PSPs help you manage the payment environment by providing prebuild payment solutions. Most PSPs support a wide variety of payment options. PSPs often allow developers to customize the style of these solutions while still ensuring security.

Payment Service Providers
For the scope of this project, the focus of finding a payment solution is on ease of use and availability. Another essential feature is the support of IDEAL, as this is the most popular online payment option in the Netherlands [27]. Our research shows that Stripe is more focused on startups and has a lot of help and documentation on development with Stripe [28]. Our client has experience with PSP Mollie. Mollie is very comparable to Stripe in its size and focus [29]. Both Mollie and Stripe provide solutions that fit

within the frameworks that we are considering. In the end, we left the decision of the PSP to our client after we recommended Stripe or Mollie.

B.3.2. Refund

Most of the PSPs that we described in the previous section allow for refunds on almost all their payment options, excluding the gift cards. However, during one of our meetings, the question arose, should we refund the money or gift credits instead?

From a marketing perspective, using the credit system is beneficial. This implementation will ensure that even if the system does not find a user a match this round, the users would still have a chance during the next round. It would also mean that it is not required to implement a refund system for all the transactions. A single withdrawal interface will suffice to retrieve money back from the application.

Although it is unclear whether we can take this approach as we do not know for sure whether or not the legal reflection period applies to this application. When you pay for a date that takes place at a specific time and place, this law probably does not apply. However, if we use the credit system, it is not entirely clear whether or not this is the case. To create an MVP product under European law, we will assume that the legal reflection period applies to this product, as it is better to be safe than sorry.

B.3.3. Reporting

Every application that allows users to express themselves in any form requires moderation. The most common way of moderation is to enable users to report each other in cases of inappropriate behavior. But can we trust users to report fairly, and how do we deal with reported cases? This section will explore some of the best practices in moderation.

Supervisor Moderation

One of the most used forms of moderating is supervisor moderation. In this system, there are multiple users appointed to be moderators. These moderators have special privileges, like deleting or editing content of other users. Most social media platforms use a system called Commercial Content Moderation (CCM) [30]. This term, as described by Dr. Sarah T. Roberts, describes the activities of monitoring User-Generated Content (UGC) on social media platforms and ensuring they are conforming to the rules and laws that apply to that platform [30]. Either volunteers do this monitoring, or as in most cases, it is outsourced to a third party.

This supervisor moderation model is most likely not suited for this project. For a moderator to asses the situation, they need to observe the problematic behavior. In the case of the user profile as presented during the swiping action, moderators can check for inappropriate content. Validation of truthfulness is harder to moderate as it takes place on a date location, a place where the moderator is not present.

Distributed Moderation

Distributed moderation has two forms, user moderation, and spontaneous moderation. User moderation, as the name suggests, allows users to moderate each other [31]. These systems rely on users voting content that does not abide by the rules creating a low effort moderation system. The main downside is that for this system to function correctly, you need a lot of users and multiple votes before you can act on it [31].

As the final product is a dating application, users will get a limited amount of feedback, meaning that we need to act on a small amount of feedback. The danger of using small amounts of feedback is that users with malicious intends can disturb innocent users. Therefore we will need to find the balance between filtering out untruthful users without endangering innocent users.

Spontaneous moderation is an effect of human behavior, where people will ignore and comment on inappropriate content creating a natural sense of moderation [31]. We expect spontaneous moderation to have little to no effect on this application as users have little input on one another.

Dating Establishment

It is challenging to trust the users by providing accurate reports blindly. Since it is easy to submit false reports when a date went horribly, a user could claim that the other user never showed up and apply for a refund, even though the date happened, and both parties showed up. This behavior is not what should be permitted. There should be an external party that moderates the date while it is happening. The Dating Establishment could fulfill this exact role.

The Dating Establishment already functions as a safeguard for the date. They keep an eye out on their customers, thus also on the users during the date. This responsibility could be made broader by keeping track of which people showed up and especially did not show up. This could be done by simply writing down who does not show up, this is not too tricky and bothersome for a Dating Establishment to perform. Applying these tasks is beneficial for the Dating Establishment as well since they generate revenue when both users show up and they could be selected as a trusted Dating Establishment. It will also increase the sense of safety, resulting in more users feeling comfortable to use the application and go on a blind date.

If a user reports another user for some reason, then this should be forwarded to the Dating Establishment, where the date took place. The Dating Establishment could reply and verify the report. However, the Dating Establishment is not required to do so. If the Dating Establishment did not respond to a report within a certain amount of time, then the report could not be reviewed any longer by the Dating Establishment.

When enough Dating Establishments are connected to FireFly Dating, it is possible to have a personal trust ranking among these Dating Establishments. If a Dating Establishment provides proper moderation through feedback, then their trust ranking rises. Dating Establishments with a high trust ranking are safer for blind dates, and they should receive more dates at their Dating Establishment. This provides benefits for correctly moderating and safeguarding a date. The inner rankings should be visible for the Dating Establishment, so they know how to improve and what a low ranking does.

It is not feasible to enforce the Dating Establishment to cooperate with their moderating responsibilities. With the benefits system described above, in place, then it is more likely that the Dating Establishment uses their moderation responsibilities properly.

Conclusion

In an ideal situation, every user will be truthful and follow the rules. Sadly this is not the case, so we will need to moderate. We decided to go for a combination of systems as no single method provides a full solution.

We will rely on the feedback of users to bring attention to users that misbehave. A moderator will look at this feedback and decide on the validity of this feedback and take the appropriate action. In an optimal version, establishment owners would provide their input to get an unbiased view on the matter, although we realize that this would not always be feasible. For that reason, a reward system for Dating Establishments could be put in place. The Dating Establishments that provide their unbiased supervision will be rewarded with more dates assigned to their Dating Establishment. However, implementing such a reward system for Dating Establishments is outside of the scope of this project.

B.4. Matching Algorithm

The matching algorithm is a crucial part of a blind dating application. After the user has specified their preference and availability, the algorithm attempts to find the user a date. The algorithm takes different data points into account in order to determine a match for the user. In this section, we will discuss the different possibilities for the algorithm. We start by giving a formal definition of the problem. Afterwards this, we provide a brief overview of the current literature on matching algorithms for dating sites. Finally, we show how we plan to implement the algorithm, detailing what the minimum requirements are and how we can expand on this.

B.4.1. Problem Definition

The FireFly Dating application will make use of a matching algorithm to match users for blind dates. Unlike other dating applications, FireFly Dating will only use a small amount of user data. The main ideas to set preferences are: specifying age range, swiping candidates, and feedback after a date. The application will not ask the user for a massive amount of information for a matching algorithm, yet the algorithm of FireFly Dating should schedule the best match with the information given, and take into account other users that also need to be matched, which reduces to the Stable Marriage Problem [32]. CF in a recommender system like ones used on Amazon [33], uses user to item filtering. User X rates an item I with a score. This can then be used to predict the similar score of user Y on item I. The problem with this method is that matchmaking is different from item recommendation in that an item cannot choose its user. However, dating services will need to score users in both ways [34].

B.4.2. Research

Much research has been done into how matching algorithms work best in dating applications. These matching algorithms claim that they match users based on different aspects to find partners that suits best together. While most dating applications make use of some form of matching algorithm, researchers have expressed doubts about whether matching algorithms provide better long-term relationship results [35].

However, in practice, many dating sites still make use of some form of matching algorithm because they provide a reasonable prediction of a user's preference for partners [34]. The actual implementation of the algorithms used by different dating sites is often not made public. A Fast Company journalist revealed that Tinder makes use of a "desirability score" that functions similarly to Elo scores [36]. While Tinder claims to no longer use this algorithm [37], it can still be used as an initial way to achieve a desirability score for the user.

For the actual matching of users, a recommender system should be put in place. These systems take different parameters of the user's profile into account in order to match them with similar profiles. The two types of recommender systems that are primarily used are content-based systems and CF based systems [38].

Since the project will run with only a few inputs per profile (e.g. age, gender, location, etc.) which will mainly be used as absolute filters. For now, a content-based recommendation system will be ruled out of this project scope, as all attributes on profiles are absolute filters. It is also important to note that multiple researchers have found that CF based approaches outperform content-based approaches [38, 39]. Therefore we have chosen to create a CF based implementation for this project.

B.4.3. Collaborative Filtering

User-user CF can be implemented using different metrics to determine how similar two users' tastes are. A straightforward distance metric is Jaccard similarity [40], which takes two user ratings and compares them. However, Jaccard similarity only looks at which profiles the user has rated, not if this rating was positive or negative. An option here is not to include a negative swipe and only include positive swipes. However, this does limit the algorithm not to match frequent users and users that only sometimes swipe even though they might have the same taste. A solution to unknown ratings and a sparsely filled data collection is a Pearson Correlation [41]. This correlation uses Cosine similarity with a centering of the rankings around 0. This way, an unknown rating is considered as the mean of all the scores for a profile. The Pearson Correlation is displayed by the following formula:

$$sim(x,y) = \frac{\sum_{s \in S_{xy}} (r_{xs} - \overline{r_x})(r_{ys} - \overline{r_y})}{\sqrt{\sum_{s \in S_{xy}} (r_{xs} - \overline{r_x})^2} \sqrt{\sum_{s \in S_{xy}} (r_{ys} - \overline{r_y})^2}} \qquad \text{(Pearson Correlation [40])}$$

Where S_{xy} are the profiles rated by both users x and y, r_{xs} and r_{ys} are the ratings r for profile x and y respectively, $\overline{r_x}$ and $\overline{r_y}$ are the average rating of user x and y.

this formula also takes the pickiness of users into account. For users that tend to give many bad ratings, the score for an unknown profile will be relatively low. While a user that tends to like many user profiles will have a higher score on unknowns, taking into account user swiping manners. When there is an unknown profile rating for a user x, the algorithm will be able to take users that are most similar to user x and have rated the profile, and take the weighted average to predict a score for the user using the following formula:

$$r_{ix} = \frac{\sum_{j \in N(i;x)} s_{ij} \cdot r_{xj}}{\sum_{j \in N(i;x)} s_{ij}} \qquad \text{(Weighted Average [40])}$$

Where r_{ix} is the predicted rating on profile i of user x, s_{ij} is the similarity of profile i and j, r_{xj} is the rating on user u on profile j, $N(i;x)$ are the set profiles rated by x similar to i.

Ideally, when two users have liked each other, they are marked as a positive match. However, if either of the users have not rated the other, a positive match could still occur. Using swiping, this data can be populated relatively fast. This is not the case for data that comes from the feedback of dates, as there will be relatively much fewer dates than swipes. This will likely cause the data to take a long time to populate and requires the user to enter many dates before the algorithm can provide a decent match. Applying CF to match on appearance preference will likely work as intended.

As the profile will not contain experiential traits, matching personality will have to be done with different data. An option is to use the feedback from previous dates. However, matching dates with other users that have the same interests will suffer from a cold start as the profiles will have to be build up on both sides of the sexual preference. To remedy this effect, the data of swiping can be used for the matchmaking instead when dating feedback is sparse. CF is too expensive to run on every event. Therefore, it would be better if for example the application only computes this data every twelve hours.

B.4.4. Elo

To assist in the CF process we assign score based on their desirability derived from swiping behavior. Similarly to the scores used in chess ranking, we assign all users what is called an Elo score [42]. A similar method of scoring has been used by Tinder for a long time [36]. The Elo score indicates the desirability of users. While it might not be the perfect method of ranking people's desirability, it provides a basis to start the matching process with relatively little data.

The Elo score of a user is updated when users swipe each other on the swiping section of the app. When user u_1 swipes on user u_2, the Elo Score for both of them is updated. When a user is liked by another user, the score of the person being liked is increased, and the score of the liker is decreased. The amount with which the score is changed is dependent on the scores of the users. If a user is liked by another user with a higher score then more points are transferred than when the user was liked by a user with a lower score. This system of transferring points from one user to the other causes similarly rated users to trend towards each other. The Elo scores of the users is used to calculate the probability of whether user u_1 will like or dislike user u_2. Each user u_x has score s_x. The probability is calculated using the following formula:

$$P_1 = \frac{1.0}{1.0 + 10^{\frac{s_1 - s_2}{400}}} \qquad \text{(Probability of } u_1 \text{ liking } u_2 \text{ [42])}$$

$$P_2 = \frac{1.0}{1.0 + 10^{\frac{s_2 - s_1}{400}}} \qquad \text{(Probability of } u_2 \text{ disliking } u_2 \text{ [42])}$$

$$P_1 + P_2 = 1$$

Then after this probability is calculated the result $r \in \{0, 1\}$ of swiping is used to update the score of each user according to the following formula:

$$s_x = s_x + k(r - P_x))$$

In which the k-factor is a constant. The value of k can best be determined during testing of the algorithm to see how it influences the way that matches are made.

B.4.5. Conclusion

As this application uses very little data, recommendation systems such as Netflix will be used to recommend matches with only the use of swiping data and date feedback using CF. This system can then be improved by using an Elo score per user. This Elo score can then be applied as the weights for the CF part of the Algorithm. This will result in the best match that can be made from the data the application provides. The swiping functionality will not be based on recommender systems, as leaving this random will prevent first time users from being excluded in the number of appearances, which is desirable for the FireFly Dating application.

B.5. Framework Analysis

Nowadays, a lot of applications use frameworks to help do the heavy lifting, especially in web development. For front-end alone, there are over 20 different frameworks, so picking one suited for your project can be challenging. It is important to do sufficient research about the different possibilities to be able to make an informed decision about these frameworks. In this chapter, we will explore a set of options for both back- and front-end frameworks.

B.5.1. Front-End

To select the best framework for this specific project, three entirely independent frameworks are considered. These discussed frameworks are Angular, React, and VueJS. Each framework is discussed on the same criteria questions, namely: Is the framework convenient to use by the developer? How will the framework perform? Is the framework reliable for all the targeted devices?

Angular

Angular [43], developed by Google, was the first framework on the market. Angular is known for being a difficult platform to use. Therefore, it will take a while to learn Angular and use it to its full potential. These difficulties arise from Angular being enterprise-focused. Especially the applications of large enterprises. However, for smaller applications, this will provide more disadvantages than advantages. The documentation of each of the Angular components is adequately described. One feature of Angular is that it uses TypeScript [44]. TypeScript is an enhancement of JavaScript, providing a type-based and object-oriented implementation of JavaScript.

The performance of Angular is excellent; it is a reliable platform and has a consistent performance no matter the size of the project. However, as stated earlier for the smaller applications, the usage of Angular will most likely result in more overhead added to the project. The impact of this overhead is minimal but would result in unneeded work for our project. There already exist many tremendous and easy installable libraries. However, this is not the case with Angular. One of the most significant drawbacks of Angular is that most used libraries can not be added. They need to be ported to work with Angular specifically. Making a broad set of libraries inaccessible for the Angular platform.

The project will eventually be accessible from a wide range of different devices, including phones. Thus the platform must allow for correct phone behavior. Angular is most of the time used for desktop applications, and forwarding to a phone environment is more complicated than should be the case. Inconvenient workarounds must be created to get it properly working on all devices, which is not desirable for this project.

Some members of the project team have prior experience with Angular. These experiences are mostly negative. For this reason and the disadvantages outweighing the advantages, the project will not be using Angular as its framework.

React

React is a JavaScript library for designing user components and interfaces [45]. The library is designed by Facebook and allows for a rich mobile user interface [46].

React is known for its outstanding developer friendliness. This friendliness is the result of a platform that is easy to learn and use. React consists of standalone components with interactability. Preventing elements from becoming too broad and too complicated, but still being able to embrace the great interactability between different independent parts. Unlike Angular, which relies on complex code underneath the elements. The documentation of React is proper and does not form any difficulties while developing with the platform. One of the features of React is that it uses JSX[47] instead of JavaScript. JSX is a template language and a super-set of JavaScript. It enables excellent interactability between the renderer and the data residing in the code.

The Framework allows using all known libraries, which is a massive advantage in comparison to Angular. The React library, however, is rather extensive, which is not the best performance-wise. React is also known to be backward compatible, which is a useful feature for development after the project is finished.

The selling point of React is that it allows for seamless mobile device implementation, which is another significant improvement in comparison to Angular since the project will be exported to mobile devices.

VueJS

VueJS is an open-source community supported and maintained framework [48]. Both VueJS and React are more popular frameworks in comparison to Angular. Using VueJS or React results in almost the same advantages in contrast with Angular. However, they still have their differences.

VueJS has even better ease of use compared to React. It is beloved by many developers, and most developers that worked with VueJS want to continue using it for other projects, unlike Angular [49]. VueJS is easier to use than React and thus allows us to maximize time spent on the product rather than on learning a framework, which is an excellent advantage for a project with a limited time period. The framework is open-source. This allows for the documentation to be perfect, improved versions come with enhanced documentation, and the community can always provide more in-depth updates of said documentation. One could use the JSX language like React. However, it is entirely optional for the developer.

The performance of VueJS is excellent. It is a relatively small library and thus is more efficient to use across all devices. The developer is allowed to use all libraries, just as React. VueJS is backward compatible as long as the project follows the proper guidelines. Since VueJS is a lightweight framework, it is excellent to use as a mobile application [50], which is a great advantage for this specific process.

Some developers of the team have prior knowledge with Vue and recommend it as it is an excellent platform.

Conclusion

Angular is a framework suited for larger enterprises, making it more difficult for our project. It does not provide proper mobile support, and the majority of the already existing libraries cannot be used. For the reasons stated above, Angular does not fit as a framework for this project, so the choice remains between React and VueJS. Our comparison shows that both React and VueJS, fit the project well. They both have high developer satisfaction, support the usage of the majority of libraries, and have proper mobile support. In that sense, they do not differ remarkably from one another. However, using VueJS has a few more benefits than React, namely that it is a smaller, lightweight library in comparison to React. VueJS as a framework allows for greater efficiency, and for that reason, we choose VueJS as the front-end framework for this project. In conclusion, VueJS is the most advantageous for this specific application. Not only does it support the implementation of the project, but it also enhances it to its fullest potential.

B.5.2. Back-End

We decided to divide the back-end of the project consists of the database, the server-side, and the matching algorithm. We will start by discussing which database to choose from. Next, we will elaborate more on the server-side scripts. Finally, we will talk about the ways of implementing the matching algorithm.

Database

One of the first questions that need to be answered when selecting a database is if we need to choose a relational or a non-relational database. Next, we have to compare the database schemes that remain to find out which one suits the best for this project in terms of size, scalability, speed, plugins, and security.

(Non)-Relational Database

Relational databases are the most known and operate on SQL. Some examples of these databases are MySQL, PostgreSQL, and Oracle. The essential characteristic of relational databases is that they rely on strict definitions and relations. Most systems store their data using tables in which columns represent the data type. Every row is a record that has, in its columns, the values for that record. These tables have strong definitions meaning that a table meant for storing booleans can only store booleans. Most SQL databases will uphold the ACID principles making them very reliable.

Non-relational databases allow for a lot more flexibility. Instead of having defined table structures, these databases contain documents. The structure of these documents can be described but not enforced, which means that when a different version of data, which is in the same style but with different values i.e., JSON, is uploaded, it will still be accepted. The other characteristic of this open structure is

the scalability it provides. These properties make non-relational databases ideal for applications that are rapidly changing and fast-growing.

As our application will be static in its structure, it benefits more from the relational databases. We plan to insert only values that match the strong definitions, such that we can query the data more straightforward and access the data as fast as we can. Therefore this project will use a relational database.

Size

As we decided to use a relational database, we did research about which one to use. MySQL and PostgreSQL are compared in this study [9], and they show several differences that we can use. They concluded that MySQL is more popular and easier to use and thus makes up for the shortage of functions. Because of the higher reputation of MySQL, people tend to believe that by using it, they can improve their projects and work in a higher advanced level of development. The most significant difference between the two database systems that could matter for this project is the maximum database size and the maximum table size, as shown in Table B.1. But since these sizes are this large that we will not reach the maximum in this project, we decided that in this case, the choice between these two does not matter to us.

DBMS	**MySQL**	**PostgreSQL**
Maximum Database Size	Theoretical Unlimited(256TB)	Unlimited
Maximum Table Size	MyISAM: 256 TB; InnoDB: 64 TB	32 TB

Table B.1: *Differences between MySQL and PostgreSQL [9]*

Scalability and Speed

MySQL as a database server provides ultimate scalability. It can handle large capacities of data for example run an entire warehouse, which holds terabytes of information [51]. That study shows that the execution time and the memory utilization of MySQL in the compared DBMS is one of the lowest. Which is supported by running some select queries as shown in Fig. B.1 and Fig. B.2.

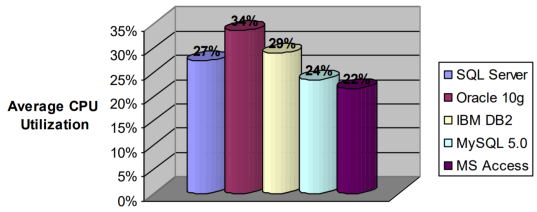


Figure B.1: Average CPU utilization [51]

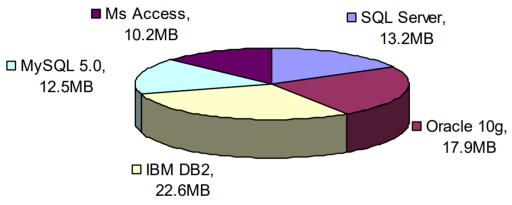


Figure B.2: Average memory usage [51]

■ ***Plugins***

MySQL provides support for a lot of application development needs. Since several programming languages already contain plugin libraries to support MySQL database support into nearly any application [52]. These plugins include stored procedures, functions, cursors, triggers, views, and more. These possibilities offer developers everything they need to be successful in building their database system.

■ ***Security***

Security is also one of the reasons we will use MySQL in this project. Since MySQL provides functions to encrypt and decrypt the value of the data, it cannot be read by perpetrators if they hack into the database. The connection between the database and the server is unencrypted by default, but it uses an authentication procedure to verify if the server is accessed correctly. The unencrypted connection is not a problem, since they run on the same system, causing the connection to be internally. The MySQL server checks three scopes: client name accompanied by the corresponding IP address, username and password [53].

Server Side

NodeJS [54] will be used to run scripts on the server-side to produce dynamic web page content. NodeJS is a JavaScript framework that makes it easier to have a clear overview of languages, since web pages use JavaScript to get their content loaded and visible to the users. We prefer to have the same style for client-side and server-side scripts.

NodeJS is mainly a framework that is used for developing high-performance that does not have the best interest in using the mainstream multithreading approach but prefer to use asynchronous I/O models, which are event-driven [55]. NodeJS performs significantly better in high concurrency situations than PHP and python [56]. NodeJS is also a relatively new technique in comparison to PHP, for example, which is used for small and middle scale websites. FireFly Dating application will contain a large number of users to match them at best. Therefore it seems to be the best programming language for the server-side scripts. The number of request per second for a select operation in a database is shown in Fig. B.3 and the time it takes with multiple users to run a script with a select query is visible in Fig. B.4. We followed the preferences of the paper in which they compared PHP, Python, and NodeJS to make our final decision and choosing for NodeJS as our server-side script language.

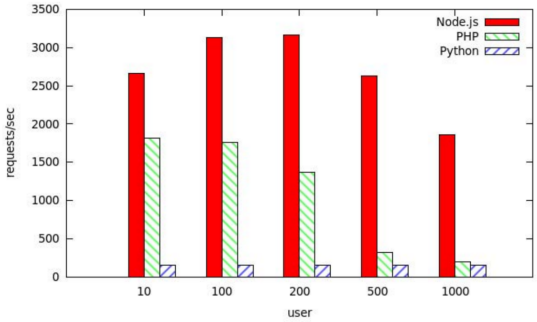


Figure B.3: Results for “Select Operation of DB” mean requests per second [56]

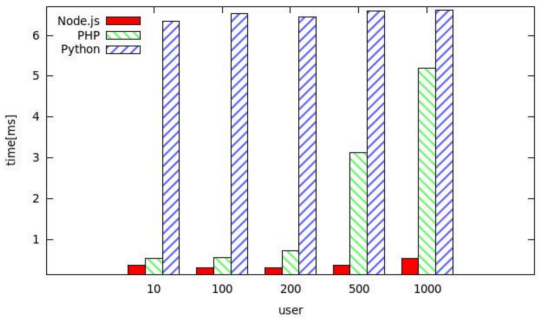


Figure B.4: Results for “Select Operation of DB” mean time per request [56]

The Matching Algorithm

As said in the previous section, NodeJS performs best in executing a task with a large number of users. In Fig. B.5 and Fig. B.6 it is shown that NodeJS performs best in executing heavy tasks, but they were not extraordinary. In Table B.2, it can be seen that NodeJS performs best in calculating the values of Fibonacci in requests per second and the meantime. From this, we can determine that NodeJS can perform best under heavy load, which is suitable for our matching algorithm.

On top of that, the team wants to use the same language in the matching algorithm as with server-side scripts. It will be easier to test the single parts and to connect them. Using a single language for several components is preferred by the team in terms of confusing languages.

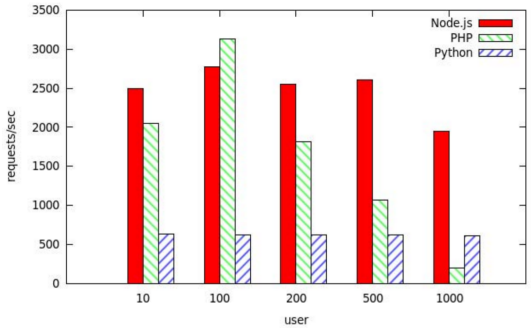


Figure B.5: Results for “Calculate Fibonacci(10)” mean requests per second [56]

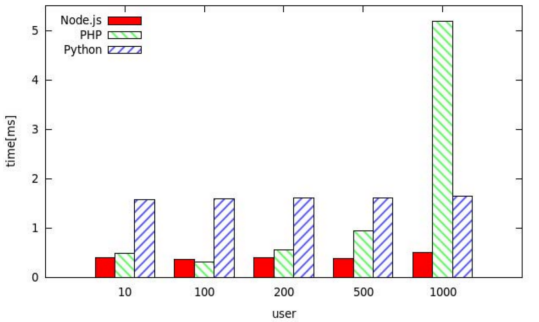


Figure B.6: Results for “Calculate Fibonacci(10)” mean time per request [56]

Web development technology	Calculate value of Fibonacci	Mean requests per second [#/sec]	Mean time per request [ms]
NodeJS	Fib (10/20/30)	2491.77/ 1529.4/ 58.85	0.401/ 0.654/ 16.993
Python-Web	Fib (10/20/30)	633.68/ 209.89/ 2.9	1.578/ 4.764/ 345.307
PHP	Fib (10/20/30)	2051.22/ 168.8/ 1.78	0.488/ 5.942/ 560.553

Table B.2: *Results for "Calculate Fibonacci (10/20/30))" [56]*

Conclusion

The back-end is divided into three different elements: the database, server-side scripts, and the matching algorithm. For each of these parts, we have examined what the best framework is.

MySQL is the DBMS that will be used in this project since it is a relational database with the best use of sizes, speed, scalability, and security. Another reason for us to use MySQL is that all the members of the project group already have previous experience with MySQL.

The server-side scripts will be written in JavaScript, with NodeJS as the runtime environment. The reason for this is that NodeJS provides the speed and reliability desired for executing queries.

The matching algorithm will be written in JavaScript as well. The speed and reliability of NodeJS, together with the fact that NodeJS is already used in other parts of the system, made us decide to use NodeJS for the matching algorithm.

B.5.3. Testing

This section discusses different testing platforms for the project. Only one testing framework is required as it will run entirely on JavaScript for the front- and back-end. Two different testing frameworks that are currently highly popular are Jest [57] and Mocha [58]. There are many more testing frameworks for JavaScript. However, Jest and Mocha seem to fit our project the best [59]. Both testing platforms embrace mocking as a testing implementation, have reliable documentation, and have meaningful code coverage reports [57, 58, 59]. Using multiple testing platforms simultaneously is not common enough to do, the benefits will be easily outweighed by the drawbacks. Thus using both Jest and Mocha is not what we want. A choice must be made, after all. Below, we will go more in-depth on the features of the different testing platforms.

Jest

Jest is a testing framework that focuses on simplicity and works well with Vue [57]. The front-end will be written in VueJS, making it efficient to test the front-end properly. Jest uses snapshot testing as regression testing to ensure that the front-end did not change unexpectedly [60]. This is done by creating a snapshot of the component and checking if it did not change, which is useful for keeping the front-end visually correct. Jest has a good testing experience, even inexperienced testers could implement proper testing with Jest [59].

Mocha

Mocha is a testing framework that operates on NodeJS [58], which is excellent as it is our back-end language. A more substantial part of the project is the back-end, thus Mocha fits this project well. Mocha allows support for all browsers, including headless browsers [59]. Headless browsers allow for automation testing [61], which is undoubtedly a feature that is significant for choosing the correct testing framework. It is a bit more complicated than Jest to set up and work with. However, since some members of the project team already has prior experience with Mocha, this does not apply to us.

Conclusion

The choice for the testing framework was between Jest and Mocha, as they are the more popular testing frameworks of this moment. Generally speaking, Jest has its focus on the front-end and Mocha on the back-end. Using both frameworks simultaneously is out of the question. The project will be more focused on the back-end than on the front-end, and the front-end features of Jest are quite limited. Thus Mocha fits the project better than Jest, and for that reason, Mocha will be used as the testing framework for this project.

B.6. Conclusion

We have explored and answered all questions mentioned in the Introduction. To conclude this report we give an overview of the answers to all the main research questions.

Functionality

Our client desires to have an MVP. A few of the main components of this MVP are user accounts, user preferences, matching users, supplying dates, and have feedback options after the date. Our client values a stable application that has a focus on providing a comfortable environment for the user. This comfortable environment should reassure the user to trust the app and go on a date.

Payment and moderation

The exact PSP is yet to be determined by our client. All the PSPs we consider do contain a solution for the project.

For the refund system, we will provide a way to refund purchased credits. We will, however, not refund the payment of an unavailable match, canceled date or no-show with money but rather with credits. This method will ensure that users stick with the application.

For moderation, we will use a hybrid implementation between user moderation and supervisor moderation. Users can report other users at two moments, during swiping and after a date. Moderators will review these reports and have the final judgment.

Matching algorithm

Due to the data sparsity of the application, a recommender system using CF creates matches based on the swiping data and the feedback. To improve on this system, every user gets an Elo score. These scores function as weights for CF.

Swiping will not be based on recommender systems. By making swiping random, we prevent first time users from being excluded due to lack of data.

Frameworks

We decided to go with JavaScript as our primary language, for both front- and back-end.

For our back-end framework, we choose NodeJS as it allows for high performance and aligns with the JavaScript preferences we stated before.

The front-end framework of choice is Vue. Vue supports all our needs and seems to be the best for our project in comparison to the other discussed front-end frameworks.

For our database system, we decide to go with MySQL as it is the system we have the most experience with and it had good results in performance tests.

C

Flowchart

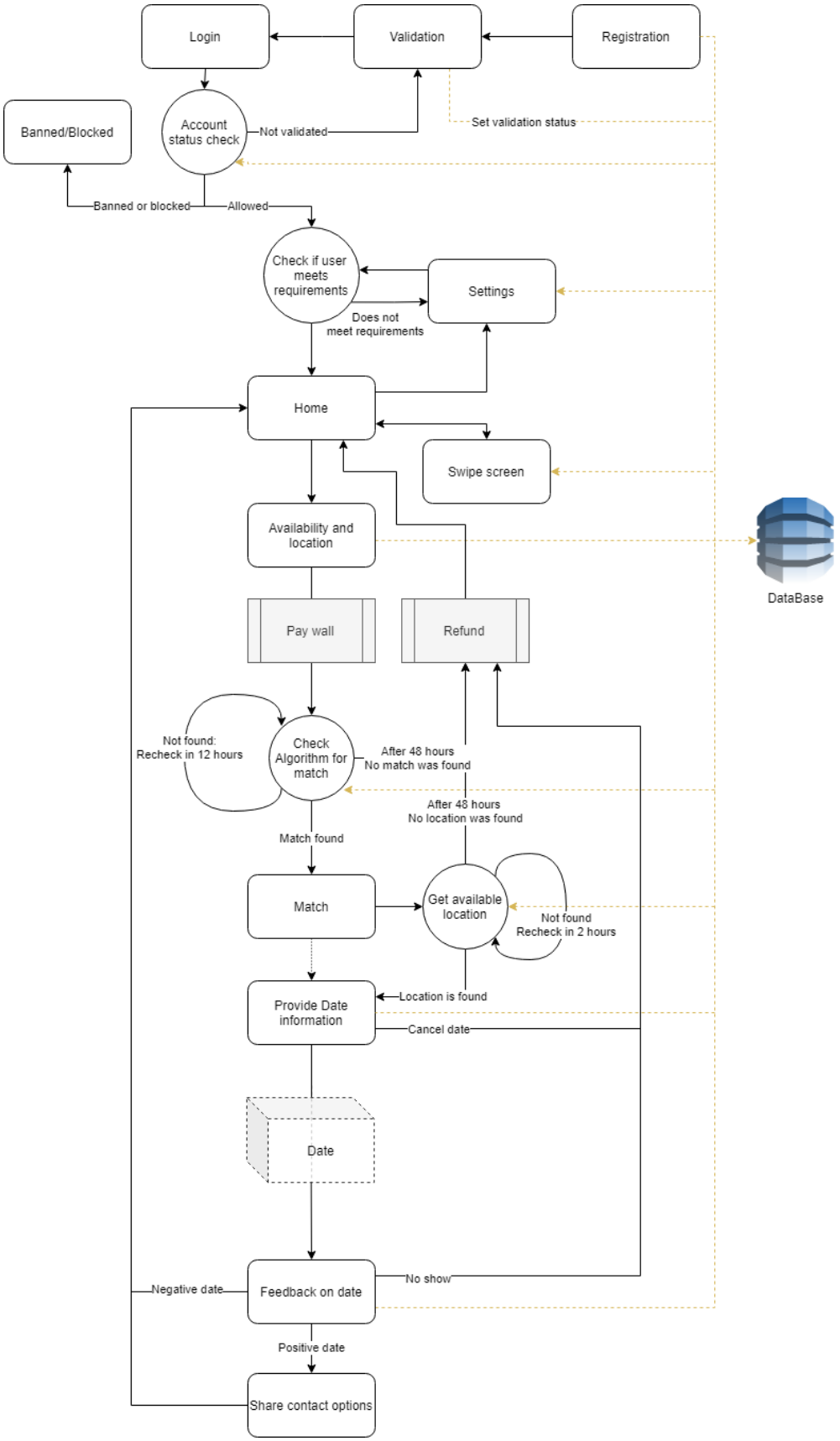


Figure C.1: Initial Functional Flow Diagram for the Minimal Viable Product

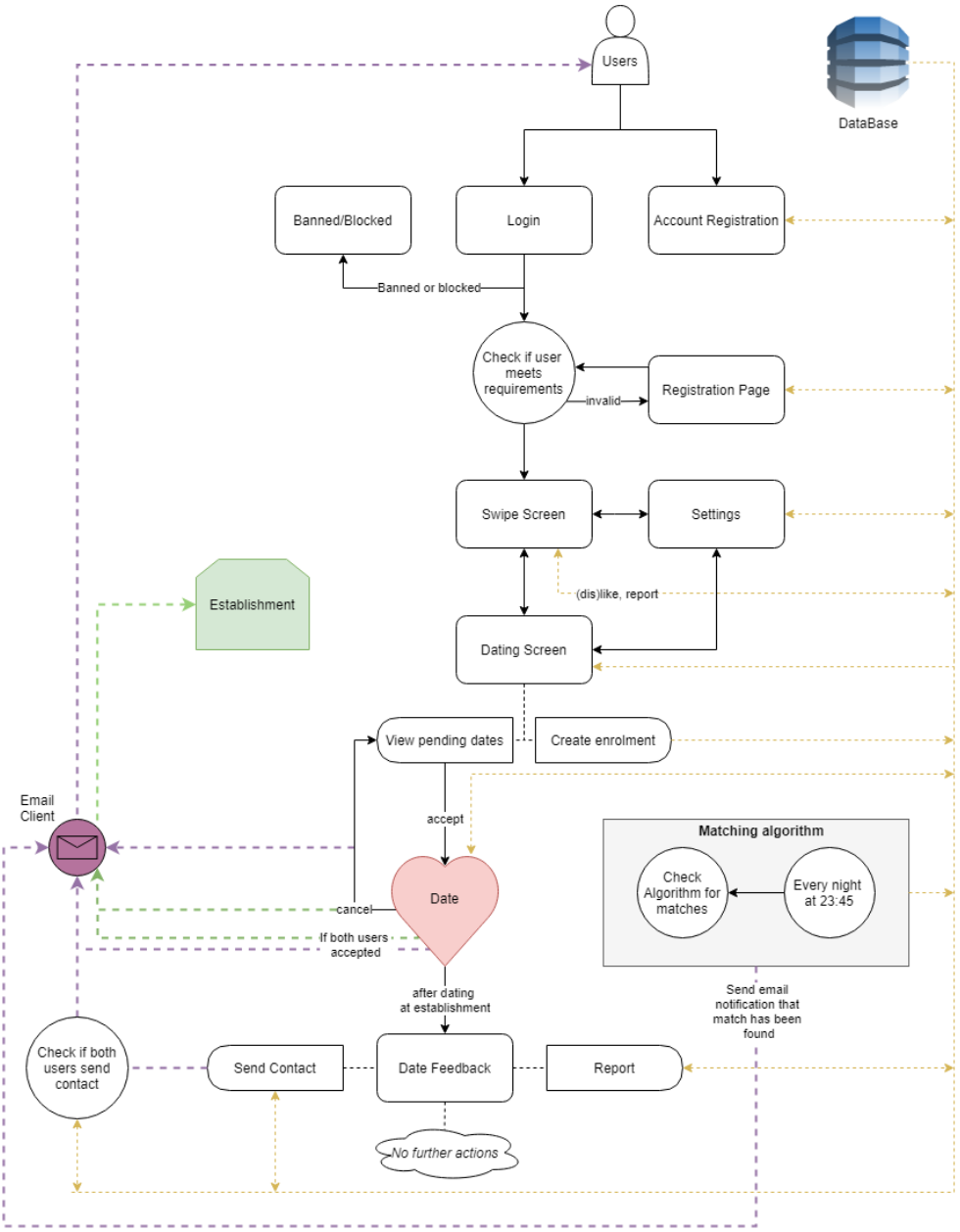


Figure C.2: Final Functional Flow Diagram for the Minimal Viable Product

D

Directory Tree

```
root/
├── backend/
├── frontend/
└── node_modules/
```

Figure D.1: The Directory Structure of the Root Directory

```
frontend/
├── node_modules/
├── public/
├── src/
│   ├── administrator/
│   │   └── components/
│   │       ├── dashboard/
│   │       ├── establishments/
│   │       ├── genders/
│   │       ├── reports/
│   │       ├── timeslots/
│   │       └── utils/
│   ├── client/
│   │   └── components/
│   │       ├── dating/
│   │       ├── profile/
│   │       ├── questions/
│   │       ├── setup/
│   │       └── swiping/
│   ├── common/
│   │   ├── components/
│   │   │   ├── navigation/
│   │   │   └── utils/
│   │   │       └── alerts/
│   │   └── scss/
└── tests/
```

Figure D.2: The Directory Structure of the Front-end Directory

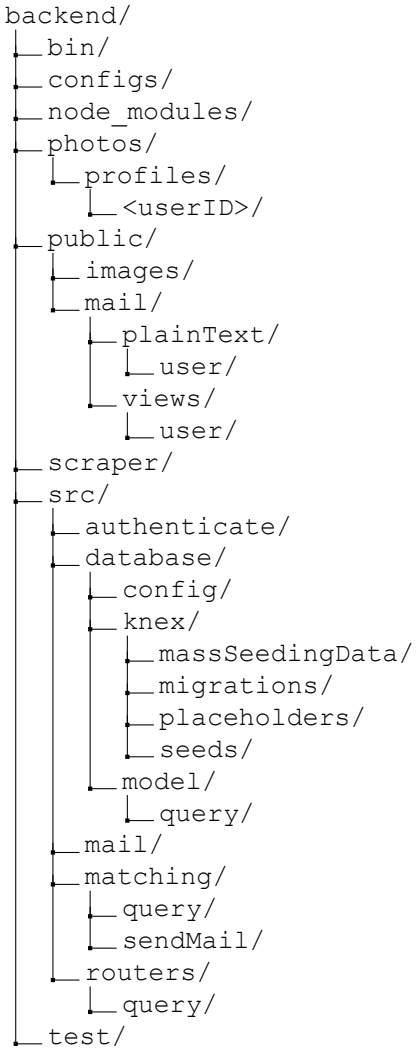


Figure D.3: The Directory Structure of the Back-end Directory

E

SIG Feedback

In weeks 6 and 9 of the course, the project team was required to submit their codebase to the Software Improvement Group (SIG). These times are respectively with week 4 and 7 of the development process. The team received feedback after the first submission with advice on how to improve the maintainability of the code. This feedback is as much as possible applied and can be seen in the second round of SIG feedback.

E.1. First Upload

The first time the code was uploaded on Sunday, May 31, 2020. The feedback was received on Friday, June 5, 2020. This feedback consists of advice in duplication, unit size, unit complexity, unit interfacing, and module coupling.

E.1.1. SIG Feedback

The codebase scores a 4 out of 5.5 stars on the maintainability model. As can be seen in Fig. E.1, the Unit size scores 0.5 stars while the majority of the other rating score around 5.0. The team can conclude that the unit size, because of the lower subscores, is a possible point of improvement.

SIG says that when they look at the unit size, they look at the percentage of code that has a length above average. These above-average lengths can have several causes, but the most common is that a method contains too much functionality. Often the method used to be small, but over time the method was expanded further. The presence of comments separating the code could indicate that this method is responsible for multiple things that can easily be separated. An example of a file is the *massSeedingData/users.js* that contained almost 8000 lines of code and only a single function.

When code is copied, the maintenance effort for fixing bugs or making changes increases. Bugs can be introduced if changes in duplicated parts are not consistent. Although the duplication rates 5.1, some files could be fixed to improve the maintenance.

Complex units have more execution paths, making them harder to understand and require more test cases. Especially *userRouter.js* had a McCabe Complexity of 19, which is way too much and has to be reduced to improve maintenance.

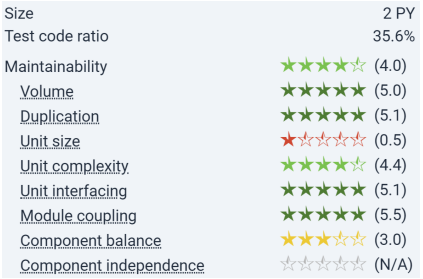


Figure E.1: First SIG Feedback

E.1.2. Actions taken

The unit size of the code was found to be too large and should be improved. The code found to be too large mostly included functions handling input for both the back- and front-end. The team sharpened the static code analysis tool such that these problems had to be fixed. These changes lead to improved maintainability.

In the back-end, the router functions were taken out of the constructor, such that the functionality remained. The number of parameters was reduced to a maximum of two; by doing this, it clarifies the working of the function. For a couple of methods, the complexity was too high. Therefore the function was split into multiple functions. Such that the complexity is separated over the functions and became more maintainable.

In the front-end, the problems mostly occurred when parsing the data from the server; preforming similar checks on the data caused duplicates with other functions. Creating a parse function that can be used in multiple functions is more maintainable. On top of that, it is also easier to change something instead of changing it multiple times, see Section 9.5.2.

With the addition of the stricter static code analysis tool, the team strives to improve the maintainability of the codebase and their SIG score.

E.2. Second Upload

The second time the code was uploaded to SIG was on Tuesday, June 16, 2020. The feedback was received on Monday, June 22, 2020. Due to a mistake on our side, the wrong file was uploaded, which resulted in an incorrect review. SIG was contacted, and answered with the option to reupload our submission. They also clarified the reason why the unit size of VueJS files was incorrect. This incorrect size had to do with an attribute in the script tag of the files that would cause the correction script to misbehave. We removed these attributes and took the opportunity to reupload our submission. The feedback that is described below will be that of the reuploaded submission.

E.2.1. SIG Feedback

The second submission scores a 5.0 out of 5.5 stars on the maintainability model. As can be seen in Fig. E.2, the unit size scores 3.7 stars, whereas other scores are above 5.0. The team can conclude that the unit size has improved, but not as much as it should have been.

Code duplication is one of the things that the team worked on to improve. In the front-end, several methods were moved such that they could be accessed globally. According to SIG, these files were improved, but since the number of files grew, the total code duplication also increased. This resulted in the number of stars received for duplication going from 5.1 to 4.3.

For unit complexity, 4.4 stars were given for the first upload. The McCabe Complexity of most of the files are fixed, since there are only three files left. *Matcher.py* is the file with a McCabe Complexity of 7, the others have 6. This results in 5.4 stars for unit complexity.

Name	Casparkrijgsman	
Size	1 PY (-1.30)	
Test code ratio	166.6% (+131.0)	
Code touched	8,636 LOC	
Code removed	30,968 LOC	
Maintainability	★★★★★ (5.0)	▲ 0.96
Volume	★★★★★ (5.3)	▲ 0.34
Duplication	★★★★½ (4.3)	▼ 0.82
Unit size	★★★★½ (3.7)	▲ 3.14
Unit complexity	★★★★★ (5.4)	▲ 0.97
Unit interfacing	★★★★★ (5.2)	▲ 0.12
Module coupling	★★★★★ (5.5)	= 0.00
Component balance	★★★★★ (5.5)	▲ 2.43
Component independence	☆☆☆☆☆ (N/A)	= 0.00

Figure E.2: Second SIG Feedback

E.2.2. Recommendations

Having a strict coding guideline to follow can be a nuisance, but in the end, it helps the creation of elegant and readable code. The team agrees that on future projects, these strict guidelines will be deployed from the start as having them at the beginning will improve a project. A point of improvement for the current project is the code duplication due to its decrease in value, as stated in the previous paragraph. This duplication is mostly present in the front-end Admin App, as this contains a lot of similar screens that were not created with the reduction of code duplication in mind.

F

Database UML

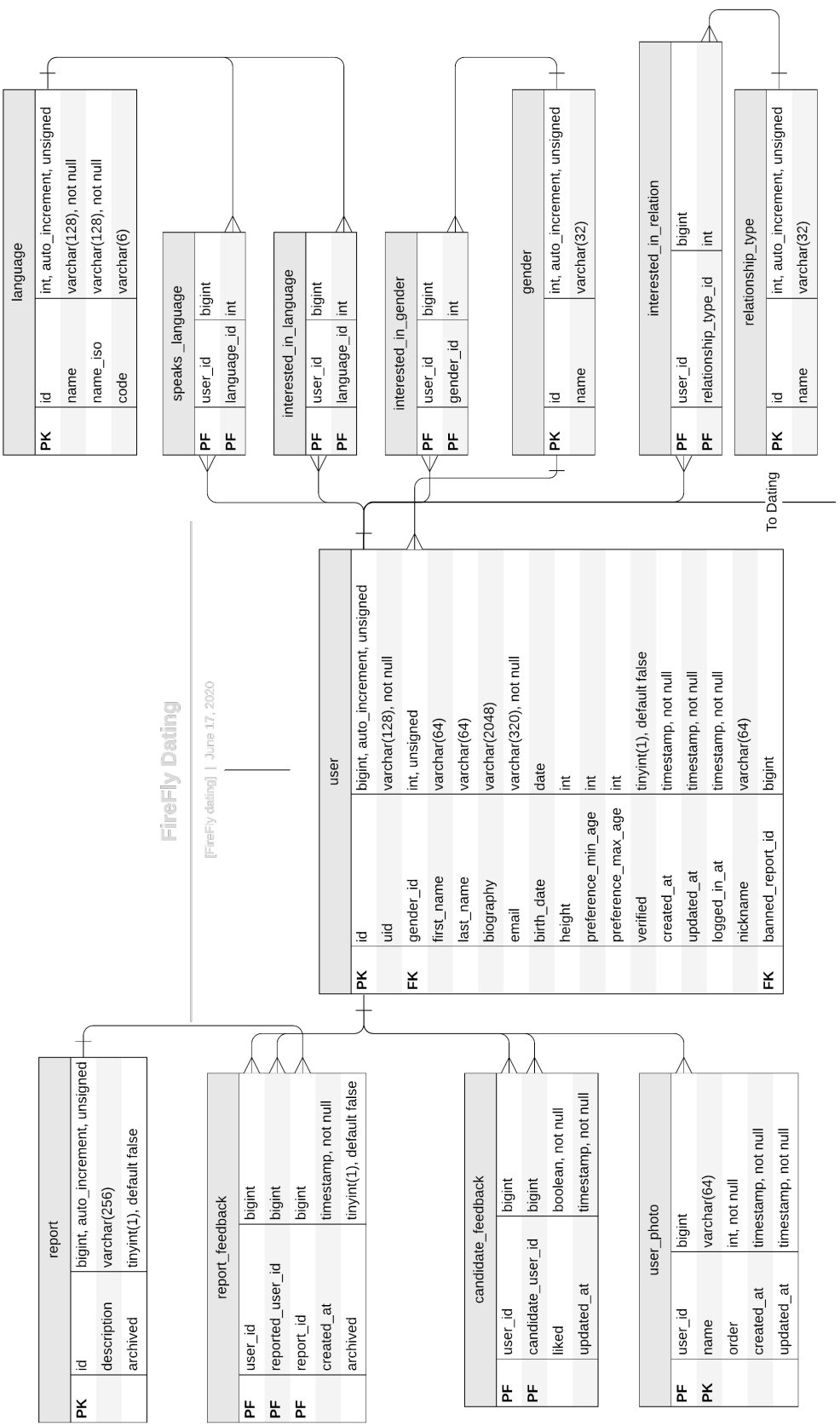


Figure F.1: Database UML for the user

Figure F.2: Database UML for dating

G

Database Tables

The $candidate\_feedback$ table is used to store all the feedback provided on a swiping candidate. $user\_id$ and $candidate\_user\_id$ refer to the $user$ table.

candidate_feedback					
Field	Type	Null	Key	Default	Extra
user_id	bigint unsigned	NO	PK	NULL	
candidate_user_id	bigint unsigned	NO	PK	NULL	
liked	tinyint(1)	NO		NULL	
updated_at	datetime	NO		CURRENT_TIMESTAMP	DEFAULT_GENERATED on update CURRENT_TIMESTAMP

Table G.1: Table Representing Database Table ‘candidate_feedback‘

The $date$ table holds all the date data of the application. Every date has an unique id. $user\_id\_a$ and $user\_id\_b$ refer to the $user$ table. $dating\_establishment\_id$ refers to the $dating\_establishment$ table and $time\_slot\_id$ to the $time\_slot$ table.

date					
Field	Type	Null	Key	Default	Extra
id	bigint unsigned	NO	PK	NULL	auto_increment
user_id_a	bigint unsigned	NO	FK	NULL	
user_id_b	bigint unsigned	NO	FK	NULL	
dating_establishment_id	bigint unsigned	NO	FK	NULL	
time_slot_id	bigint unsigned	NO	FK	NULL	
created_at	timestamp	NO		CURRENT_TIMESTAMP	DEFAULT_GENERATED
updated_at	timestamp	NO		CURRENT_TIMESTAMP	DEFAULT_GENERATED

Table G.2: Table Representing Database Table ‘date‘

The $date\_enrolment$ table is used to keep track of users that are searching for a date. Each enrollment has an unique id. $user\_id$ refers to the $user$ table. Once a date is found for an enrollment the $date\_id$ will be set. The $date\_id$ refers to the $date$ table.

date_enrolment					
Field	Type	Null	Key	Default	Extra
id	bigint unsigned	NO	PK	NULL	auto_increment
user_id	bigint unsigned	NO	FK	NULL	
date_id	bigint unsigned	YES	FK	NULL	
created_at	timestamp	NO		CURRENT_TIMESTAMP	DEFAULT_GENERATED
updated_at	timestamp	NO		CURRENT_TIMESTAMP	DEFAULT_GENERATED

Table G.3: Table Representing Database Table 'date_enrolment'

The $dating\_establishment$ table keeps track of all Dating Establishments that are registered to FireFly Dating. It is linked to the $location$ table via the $location\_id$ field.

dating_establishment					
Field	Type	Null	Key	Default	Extra
id	bigint unsigned	NO	PK	NULL	auto_increment
name	varchar(128)	NO		NULL	
description	varchar(1024)	YES		NULL	
location_id	bigint unsigned	YES	FK	NULL	
address	varchar(256)	NO		NULL	
rating	int	YES		NULL	
email	varchar(320)	NO		NULL	
phone	varchar(15)	NO		NULL	
lat	decimal(10,8)	NO		NULL	
long	decimal(11,8)	NO		NULL	
archived	tinyint(1)	YES		0	
url	varchar(255)	YES		NULL	

Table G.4: Table Representing Database Table 'dating_establishment'

The $dating\_feedback$ table is used to store all the feedback provided on a date. $user\_id$ and $feedback\_user\_id$ refer to the $user$ table. The $date\_id$ refers to the $date$ table

dating_feedback					
Field	Type	Null	Key	Default	Extra
user_id	bigint unsigned	NO	PK	NULL	
feedback_user_id	bigint unsigned	NO	FK	NULL	
date_id	bigint unsigned	NO	PK	NULL	
accepted	tinyint(1)	YES		0	
showed_up	tinyint(1)	YES		NULL	
successful	tinyint(1)	YES		NULL	
message	varchar(1024)	YES		NULL	
cancelled	tinyint(1)	YES		0	

Table G.5: Table Representing Database Table 'dating_feedback'

The $establishment\_availability$ table keeps track of all the Time Slots of Dating Establishments. It does this by referencing a Time Slot via $time\_slot\_id$ in the $time\_slot$ table. Each availability also specifies the amount of dates that could happen at that time.

establishment_availability					
Field	Type	Null	Key	Default	Extra
dating_establishment_id	bigint unsigned	NO	PK	NULL	
time_slot_id	bigint unsigned	NO	PK	NULL	
amount_of_dates_possible	int unsigned	NO		1	
archived	tinyint(1)	YES		0	

Table G.6: Table Representing Database Table ‘establishment_availability‘

The $gender$ table lists all the genders that are supported by the system. All entries have a unique id.

gender					
Field	Type	Null	Key	Default	Extra
id	int unsigned	NO	PK	NULL	auto_increment
name	varchar(32)	YES		NULL	

Table G.7: Table Representing Database Table ‘gender‘

The $interested\_in\_gender$ table lists the sexual preferences of a user. By linking the $user\_id$ with the $gender\_id$ it means that user a is interested in gender x. A user can have multiple interests. $user\_id$ references the $user$ table and $gender\_id$ references the $gender$ table.

interested_in_gender					
Field	Type	Null	Key	Default	Extra
user_id	bigint unsigned	NO	PK	NULL	
gender_id	int unsigned	NO	PK	NULL	

Table G.8: Table Representing Database Table ‘interested_in_gender‘

The $interested\_in\_language$ table lists the languages a user can date in. By linking the $user\_id$ with the $language\_id$ it means that user a is interested in language x. A user can have multiple interests. $user\_id$ references the $user$ table and $language\_id$ references the $language$ table.

interested_in_language					
Field	Type	Null	Key	Default	Extra
user_id	bigint unsigned	NO	PK	NULL	
language_id	int unsigned	NO	PK	NULL	

Table G.9: Table Representing Database Table ‘interested_in_language‘

The $interested\_in\_relation$ table lists the relationship types a user is looking for. By linking the $user\_id$ with the $relationship\_type\_id$ it means that user a is looking for relationship type x. A user can have multiple interests. $user\_id$ references the $user$ table and $relationship\_type\_id$ references the $relationship\_type$ table.

interested_in_relation					
Field	Type	Null	Key	Default	Extra
user_id	bigint unsigned	NO	PK	NULL	
relationship_type_id	int unsigned	NO	PK	NULL	

Table G.10: Table Representing Database Table ‘interested_in_relation‘

The $knex\_migrations$ table is used by the ORM to keep track of the migration state. These states are used to upgrade or downgrade the database to a specific version.

knex_migrations					
Field	Type	Null	Key	Default	Extra
id	int unsigned	NO	PK	NULL	auto_increment
name	varchar(255)	YES		NULL	
batch	int	YES		NULL	
migration_time	timestamp	YES		NULL	

Table G.11: Table Representing Database Table 'knex_migrations'

The $knex\_migrations\_lock$ table is used by the ORM to lock migrations to prevent running multiple migrations at once [1].

knex_migrations_lock					
Field	Type	Null	Key	Default	Extra
index	int unsigned	NO	PK	NULL	auto_increment
is_locked	int	YES		NULL	

Table G.12: Table Representing Database Table 'knex_migrations_lock'

The $language$ table lists all the languages available in the system. These languages are used to indicate what a user can speak and what languages they want to date in.

language					
Field	Type	Null	Key	Default	Extra
id	int unsigned	NO	PK	NULL	auto_increment
name	varchar(128)	NO		NULL	
name_iso	varchar(128)	NO		NULL	
code	varchar(6)	YES		NULL	

Table G.13: Table Representing Database Table 'language'

The $location$ table lists all the locations available in the system. These locations are used to facilitate date locations and indicate where Dating Establishments are located.

location					
Field	Type	Null	Key	Default	Extra
id	bigint unsigned	NO	PK	NULL	auto_increment
place	varchar(128)	NO		NULL	
archived	tinyint(1)	YES		0	

Table G.14: Table Representing Database Table 'location'

The $relationship\_type$ table list all the relationships types available in the system. These relationships types are used by users to indicate their interest.

relationship_type					
Field	Type	Null	Key	Default	Extra
id	int unsigned	NO	PK	NULL	auto_increment
name	varchar(32)	YES		NULL	

Table G.15: Table Representing Database Table 'relationship_type'

[1] http://knexjs.org/#Notes-about-locks

The $report$ table lists all the types of cases a user can be reported for. Each entry has an unique id.

report					
Field	Type	Null	Key	Default	Extra
id	bigint unsigned	NO	PK	NULL	auto_increment
description	varchar(256)	YES		NULL	
archived	tinyint(1)	YES		0	

Table G.16: Table Representing Database Table ‘report‘

The $report\_feedback$ table links a report instance to a user. If a user is reported than this user is the $reported\_user\_id$, the user that created the report is the $user\_id$. $report\_id$ refers to the $report$ table.

report_feedback					
Field	Type	Null	Key	Default	Extra
user_id	bigint unsigned	NO	PK	NULL	
reported_user_id	bigint unsigned	NO	PK	NULL	
report_id	bigint unsigned	NO	PK	NULL	
created_at	timestamp	YES		CURRENT_TIMESTAMP	DEFAULT_GENERATED
archived	tinyint(1)	YES		0	

Table G.17: Table Representing Database Table ‘report_feedback‘

The $session$ is used by the $express\text{-}session$[2] library and managed by the $connect\text{-}session\text{-}knex$[3] library. This table is used to store the user’s session data.

session					
Field	Type	Null	Key	Default	Extra
id	varchar(255)	NO	PK	NULL	
sess	json	NO		NULL	
expired	datetime	NO	FK	NULL	

Table G.18: Table Representing Database Table ‘session‘

The $speaks\_language$ table indicates that a user can speak a language. By linking the $user\_id$ with the $language\_id$ it means that user a can speak language x. A user can speak multiple languages. $user\_id$ references the $user$ table and $language\_id$ references the $language$ table.

speaks_language					
Field	Type	Null	Key	Default	Extra
user_id	bigint unsigned	NO	PK	NULL	
language_id	int unsigned	NO	PK	NULL	

Table G.19: Table Representing Database Table ‘speaks_language‘

[2] https://www.npmjs.com/package/express-session
[3] https://www.npmjs.com/package/connect-session-knex

The $time\_slot$ table lists all the Time Slots available in the system. Time Slots are used to indicate when a user is available and when Dating Establishments have availability.

time_slot					
Field	Type	Null	Key	Default	Extra
id	bigint unsigned	NO	PK	NULL	auto_increment
tag	varchar(32)	NO		NULL	
time	timestamp	NO		NULL	
archived	tinyint(1)	YES		0	

Table G.20: Table Representing Database Table 'time_slot'

The $user$ table lists all the users registered to FireFly Dating. This table contains most of the users' information. It also contains an entry to indicate if a user is banned. $gender\_id$ refers to the $gender$ table and $banned\_report\_id$ refers to the $report$ table.

user					
Field	Type	Null	Key	Default	Extra
id	bigint unsigned	NO	PK	NULL	auto_increment
uid	varchar(128)	NO	UK	NULL	
gender_id	int unsigned	YES	FK	NULL	
first_name	varchar(64)	YES		NULL	
last_name	varchar(64)	YES		NULL	
biography	varchar(2048)	YES		NULL	
email	varchar(320)	NO		NULL	
birth_date	date	YES		NULL	
height	int	YES		NULL	
preference_min_age	int	YES		NULL	
preference_max_age	int	YES		NULL	
verified	tinyint(1)	YES		0	
created_at	timestamp	NO		CURRENT_TIMESTAMP	DEFAULT_GENERATED
updated_at	timestamp	NO		CURRENT_TIMESTAMP	DEFAULT_GENERATED
logged_in_at	timestamp	NO		CURRENT_TIMESTAMP	DEFAULT_GENERATED
nickname	varchar(64)	YES		NULL	
banned_report_id	bigint unsigned	YES	FK	NULL	

Table G.21: Table Representing Database Table 'user'

The $user\_availability$ table keeps track of all the available times a user has for a date enrollment. If a user a has an enrollment b and is available on time x then $date\_enrolment\_id$ b will link to $time\_slot\_id$ x. $date\_enrolment\_id$ refers to table $date\_enrolment$ and $time\_slot\_id$ refers to the $time\_slot$ table.

user_availability					
Field	Type	Null	Key	Default	Extra
date_enrolment_id	bigint unsigned	NO	PK	NULL	
time_slot_id	bigint unsigned	NO	PK	NULL	

Table G.22: Table Representing Database Table 'user_availability'

The $user\_location$ table keeps track of all the locations a user can date at. If a user a has an enrollment b and can date at location x then $date\_enrolment\_id$ b will link to $location\_id$ x. $date\_enrolment\_id$ refers to table $date\_enrolment$ and $location\_id$ refers to the $location$ table.

user_location					
Field	Type	Null	Key	Default	Extra
date_enrolment_id	bigint unsigned	NO	PK	NULL	
location_id	bigint unsigned	NO	PK	NULL	

Table G.23: Table Representing Database Table ‘user_location‘

The $user\_photo$ table links the photos uploaded by users to their profile. The $order$ field is used to order the photos in the application. $user\_id$ refers to table $user$.

user_photo					
Field	Type	Null	Key	Default	Extra
user_id	bigint unsigned	NO	PK	NULL	
name	varchar(64)	NO	PK	NULL	
order	int	NO		NULL	
created_at	timestamp	NO		CURRENT_TIMESTAMP	DEFAULT_GENERATED
updated_at	timestamp	NO		CURRENT_TIMESTAMP	DEFAULT_GENERATED

Table G.24: Table Representing Database Table ‘user_photo‘

H

Administrator Panels

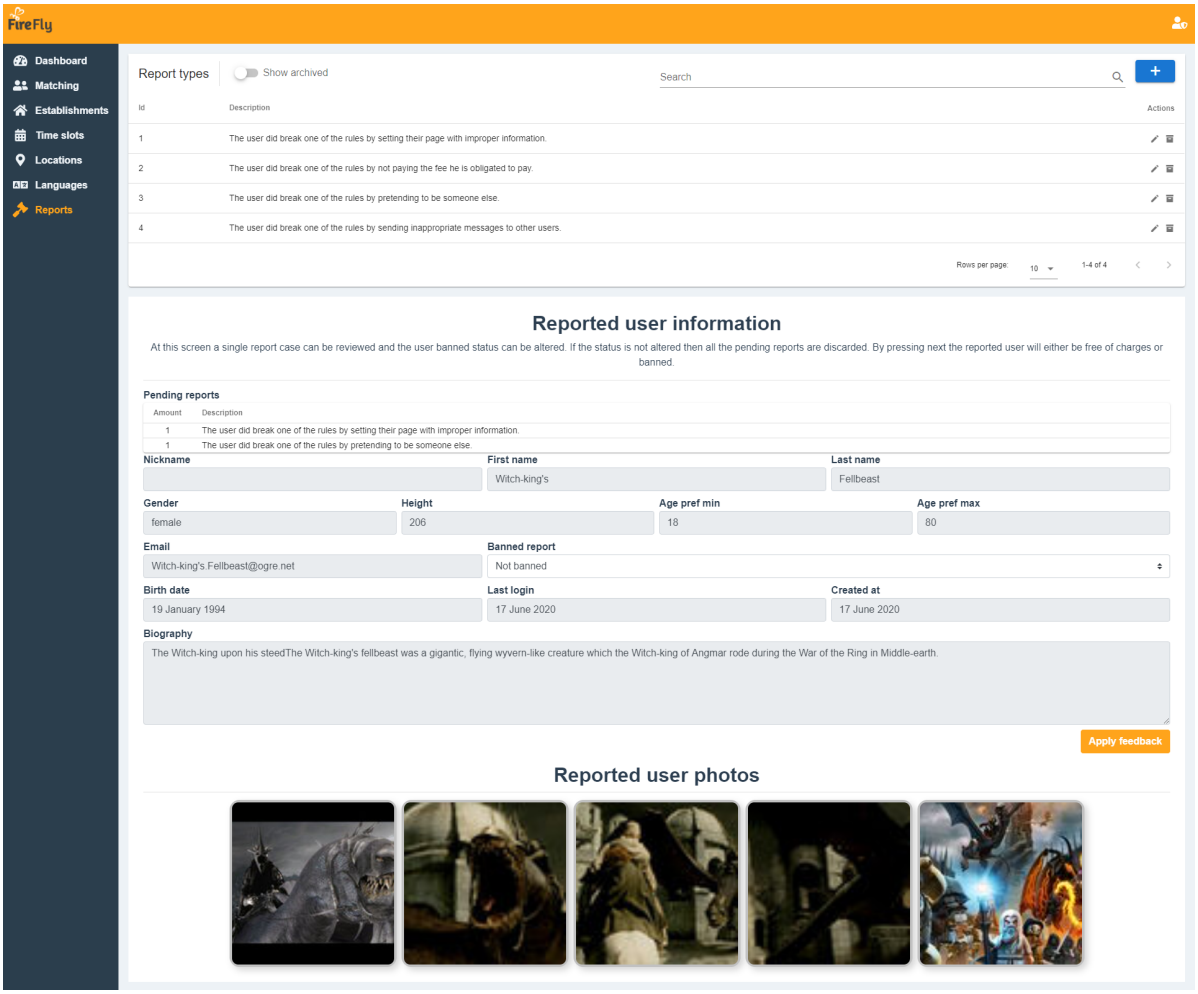


Figure H.1: Reports Panel of the Admin App

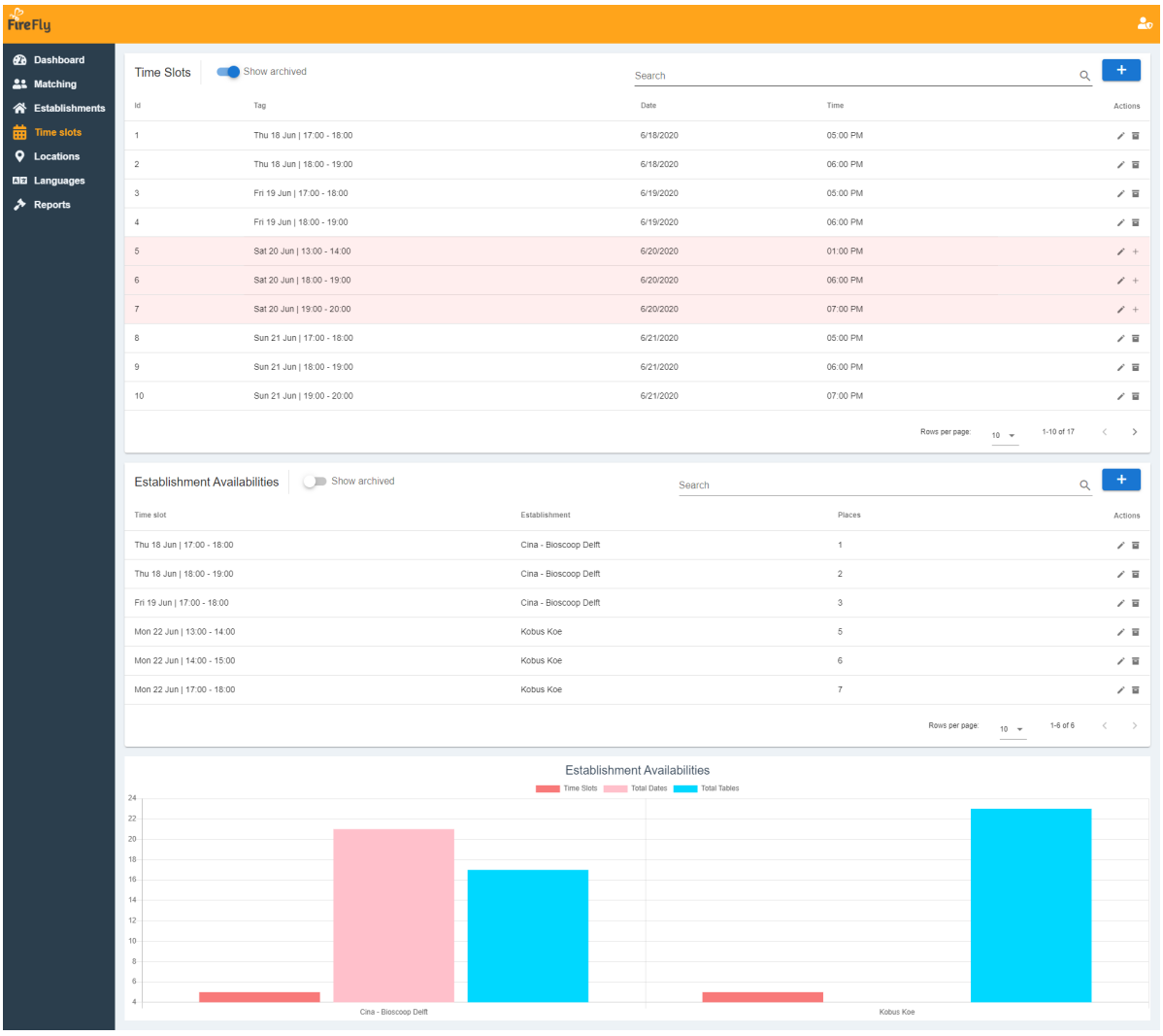


Figure H.2: Time Slots Panel of the Admin App

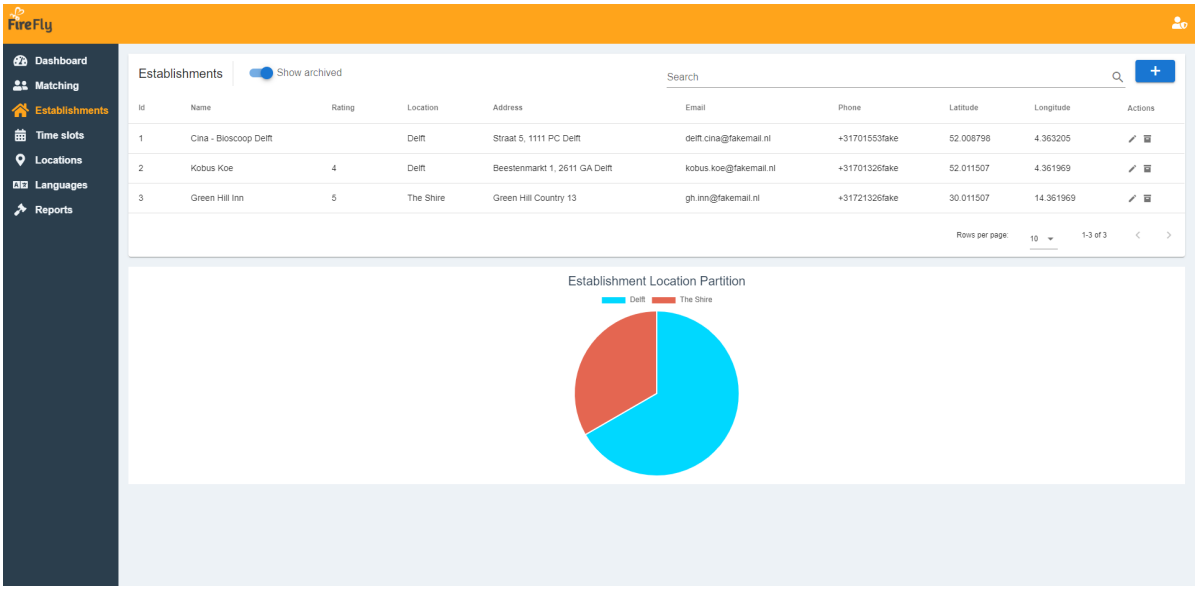


Figure H.3: Establishment Panel of the Admin App

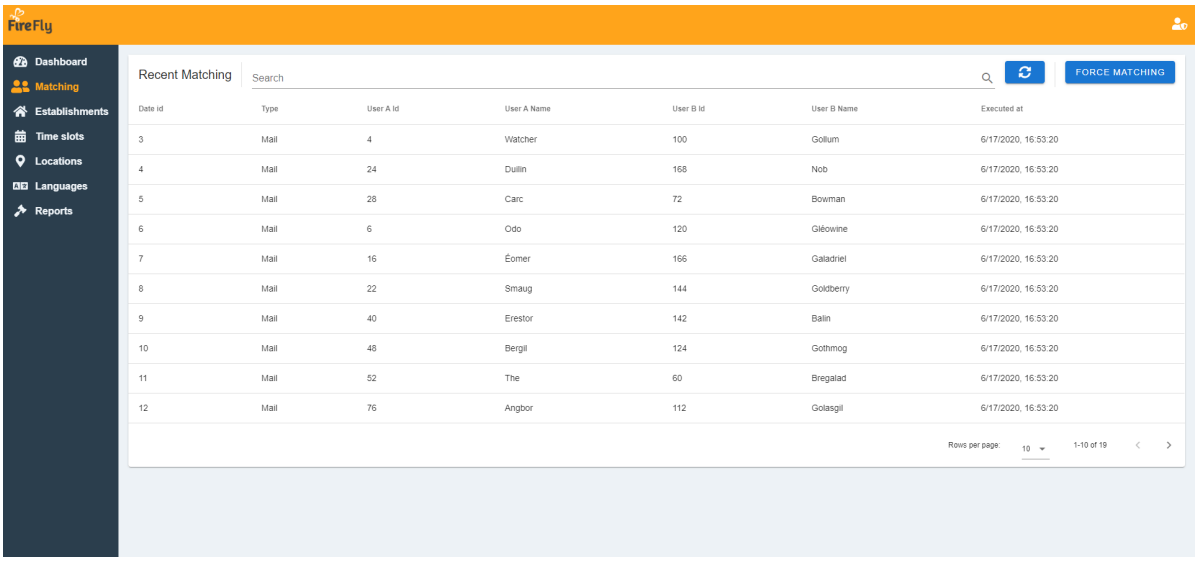


Figure H.4: Matching Panel of the Admin App

Figure H.5: Dashboard Panel of the Admin App

Project Description

This appendix contains the original project description as viewed on the Project Forum[1].

Create a blind dating web-application and automated date reservations at bars

What are we working on

Most dating apps have a low matching to dating ratio. Most of the time you spend is on getting the date itself, in the chatroom. And let's face it, that's not the best place to lit a spark or build a connection. We want to eliminate this wasted time and have a nice dating experience for everyone involved. This will be done with 1 on 1 blind dating. The web-application will reserve a spot for you and your date at a cooperating cafe/bar by looking at the availability of everyone involved.

Instead of letting an algorithm do all the matches, we let our users build the ground layer of their matches by swiping.

What the students will be developing

The students will be developing the matching and reserving system. Users should be able to change their profile info. The system should use an algorithm that smartly uses the swiping information of the user(s) and their availabilities to match people and reserve a spot at a cooperating place for their date. Also, a payment system should be incorporated.

The technologies that will be needed

- Node.js (back-end)
- React.js or Angular2 (front-end)
- Database system of choice

Additional practical information

The project owners and lead developers will be available for questions all throughout the project timeline. The developing work will happen in Delft.

[1] https://projectforum.tudelft.nl

J

Info Sheet

Create a Blind Dating Web-Application and Automated Date Reservations at Bars

FireFly Dating: Your Natural Guide in the Dark

The problem in modern dating applications is that users spend too much time in the chat window. At FireFly, they believe that the endless chatting moves away from the purpose of a dating application. Dating should be done in the real world at a café, by sharing a drink and a laugh. FireFly Dating is a blind dating application that focuses on an offline experience rather than online.

During the research phase of the project, the team established the list of the requirements that had to be met in order for the product to be successful. During this phase, the team also researched the best methods to match users in dating applications. It was concluded that a recommender system would be an optimal method of matching users.

Throughout the project's implementation phase, the team made use of the Scrum development methodology to implement all requirements. Due to the COVID-19 outbreak, the team had to deal with the difficulties of online collaboration. However, through the use of Discord, the team was able to overcome these difficulties mostly. The client preferred to have a working proof-of-concept. Thus, it was decided to focus on implementing the must-have requirements rather than implementing an advanced matching algorithm.

The final product is a web application that is accessible on many different platforms and devices. Users can enter their dating preferences, swipe other users, and get matched to go on a blind date. The application is thoroughly tested with unit testing, and the beta testing will start after the project is finished.

The project team recommended that the product owner tests the product in live beta tests and implement the insights gained from this. If this is done, the team believes that the FireFly dating app has great potential to be used in the real world.

Project team members

Colin Geukes
Lead testing, front-end & back-end developer

Caspar Krijgsman
Lead programmer & back-end developer

Steven Lambregts
Team leader, front-end & back-end developer

Vincent Wijdeveld
Lead UI, Scrum master & front-end developer

Matthijs Wisboom
Lead communications, front-end & back-end developer

All team members contributed to the report and to preparing the final project presentation.

Client

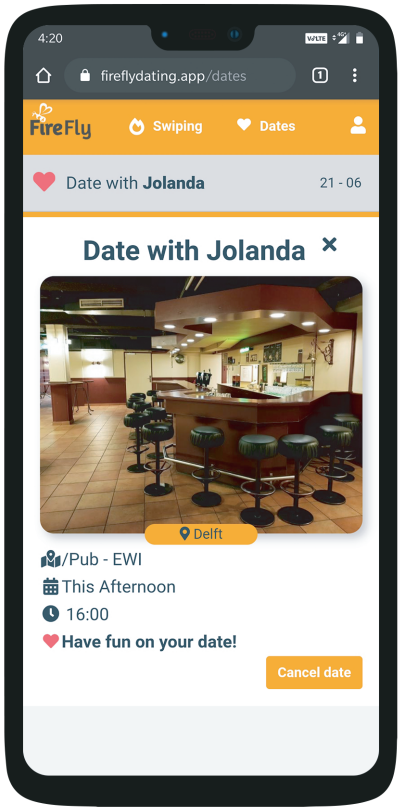
Siraadj Salarbux
CEO of FireFly Dating

Coach

Ir. Taico Aerts
Lecturer and Developer, TU Delft

Contact

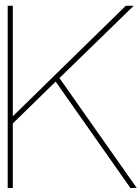
Matthijs Wisboom
matthijswisboom@gmail.com



FireFly TU Delft

Final presentation date: July 3rd, 2020

The final report for this project can be found at repository.tudelft.nl



Routes

Routing

In this wiki page all the routes of the application are described. These routes are split into separate front- and backend routing.

Backend Routing

In this section all the routing of the backend will be described.

There are several starting routes that house all the data requests of the concering part.

Route	Description	section
`/user`	The requests concerning the users	User
`/gender`	The requests concerning the genders	Gender
`/candidate`	The requests concerning the candidates	Candidate
`/location`	The requests concerning the locations	Location
`/language`	The requests concerning the languages	Language
`/time-slot`	The requests concerning the time-slots	Time Slot
`/image`	The requests concerning the images	Image
`/report`	The requests concerning the reports	Report
`/report-type`	The requests concerning the report-types	Report Type
`/admin`	The requests concerning the admin	Admin
`/establishment`	The requests concerning the establishments	Establishment
`/establishment/availability`	The requests concerning the establishment availabilities	Establishment Availability
`/dating-feedback`	The requests concerning the dating-feedback	Dating Feedback
`/matching`	The requests concerning the matching	Matching

Establishment

The route `/establishment` houses all the data requests concerning the establishments.

Auth	Type	Route	Description
admin	get	`/establishment/get`	Retrieves all the establishment entries.
admin	post	`/establishment/create`	Create a single establishment entry.
admin	post	`/establishment/update`	Updates a single establishment entry.
admin	post	`/establishment/archive`	(Un)archives a single establishment entry

`/establishment/get` `get` `admin`

Basic get request to retrieve all the establishments. Only admin authentication is required, no further params are needed.

`/establishment/create` `post` `admin`

The route to creating a new establishment entry. This can only take place with admin privileges.

Param	Type	Description

Param	Type	Description
name	String	The name of the establishment.
locationId	Number	The location id of the establishment.
address	String	The address of the establishment.
rating	Number	The rating of the establishment.
email	String	The email of the establishment.
phone	String	The phone number of the establishment.
lat	Number	Latitude of the establishment, used for maps.
long	Number	Longitude of the establishment, used for maps.

`/establishment/update` `post` `admin`

The route to update an existing establishment entry. This can only take place with admin privileges.

Param	Type	Description
id	Number	The id of the establishment you want to update.
name	String	The name of the establishment.
locationId	Number	The location id of the establishment.
address	String	The address of the establishment.
rating	Number	The rating of the establishment
email	String	The email of the establishment.
phone	String	The phone number of the establishment.
lat	Number	Latitude of the establishment, used for maps.
long	Number	Longitude of the establishment, used for maps.

`/establishment/archive` `post` `admin`

The route to (un)archive an existing establishment entry. This can only take place with admin privileges.

Param	Type	Description
id	Number	The id of the establishment you want to update.
archived	Boolean	The new archive status of the establishment entry.

User

The route `/user` houses all the data requests concerning the user.

Auth	Type	Route	Description
user	get	`/user/get`	Retrieves all data of the user.
admin	get	`/user/amount`	Get the amount of users

Auth	Type	Route	Description
admin	get	`/user/date-amount`	Get the amount of dates
admin	get	`/user/ages`	Get the amount of ages accompanied by their number of occurrences.
user	post	`/user/configure`	Configures the user with the data supplied.
user	post	`/user/set`	Updates the user with the data supplied.
user	post	`/user/date-enrol`	Enroll the user for a date.
user	post	`/user/delete`	Delete the user and all of its content from the server.
user	post	`/user/set-gender-interest`	Set the gender interest for the user.
user	post	`/user/set-language`	Set the language interest for the user.

`/user/get` `get` `user`

Basic get request to retrieve the data of the user. Only user authentication is required, no further params are needed.

`/user/amount` `get` `admin`

Basic get request to retrieve the amount of users. Only admin authentication is required, no further params are needed.

`/user/date-amount` `get` `admin`

Basic get request to retrieve the amount of dates. Only admin authentication is required, no further params are needed.

`/user/ages` `get` `admin`

Basic get request to retrieve the amount of ages. Only admin authentication is required, no further params are needed.

`/user/configure` `post` `user`

The route to configure the users basic data. This can only take place when the user is authenticated.

Param	Type	Description
first_name	String	The first name of the user.
last_name	String	The last name of the user.
birthDate	String	The birth date of the user.
gender_id	Number	The gender id of the user.

`/user/set` `post` `user`

The route to set the users settings. This can only take place when the user is authenticated.

Param	Type	Description
height	Number	The height of the user.
preferenceMinAge	Number	The prefered minimal age of candidates of the user.
preferenceMaxAge	Number	The prefered maximal age of candidates of the user.
nickname	String	The nickname of the user.
biography	String	The biography set by the user.

`/user/date-enrol` `post` `user`

The route to enrol the user for potential dates. This can only take place when the user is authenticated.

Param	Type	Description
-	-	-

`/user/delete` `post` `user`

The route to delete the users account. This can only take place when the user is authenticated. The user has to login again beforehand to renew the token.

Param	Type	Description
-	-	-

`/user/set-gender-interest` `post` `user`

The route to set the users gender interest. This can only take place when the user is authenticated.

Param	Type	Description
interests	array<Number>	The gender ids the user is interested in.

`/user/set-language` `post` `user`

The route to set the users language he speaks and is interested in. This can only take place when the user is authenticated.

Param	Type	Description
languages	array<Number>	The language ids the user speaks and is interested in.

Admin

The route `/admin` houses all the data requests concerning the administration panel.

Auth	Type	Route	Description
admin	post	`/admin/login`	Logs the admin in in the panel.
admin	post	`/admin/logout`	Logs the admin out from the panel.

`/admin/login` `post` `admin`

Login request to login in the admin panel. This will grand the admin privileges.

Param	Type	Description
token	String	The login token of the administrator.

`/admin/logout` `post` `admin`

Logout request to logout from the admin panel. This will take the admin privileges.

Param	Type	Description
-	-	-

Candidate

The route `/candidate` houses all the data requests concerning the candidates of the user.

Auth	Type	Route	Description
user	post	`/candidate/feedback`	Sets the users feedback to the candidate and creates a new queue with candidates.
user	post	`/candidate/populate`	(Re)populates the user with his candidates.

`/candidate/feedback` `post` `user`

Sets the users preferences for a candidate. This can only take place when the user is authenticated.

Param	Type	Description
id	Number	The id of the candidate.
liked	Boolean	The boolean if the user liked the candidate.

`/candidate/populate` `post` `user`

Gets the users candidates. This can only take place when the user is authenticated.

Param	Type	Description
-	-	-

Dating Feedback

The route `/dating-feedback` houses all the data requests concerning the feedback of the candidates of the user.

Auth	Type	Route	Description
admin	get	`/dating-feedback/amount`	Gets the amount of feedback of the dates.
user	post	`/dating-feedback/accept`	Accepts the date a user got.
user	post	`/dating-feedback/cancel`	Cancels the date a user got.
user	post	`/dating-feedback/report`	Report the user after the date has taken place.
user	post	`/dating-feedback/message`	Sends a message after the date was successful.
user	post	`/dating-feedback/incorrect-match`	Sets the date as unsuccessful after the date has taken place.

`/dating-feedback/amount` `get` `admin`

Basic get request to retrieve the amount of dating feedback. Only admin authentication is required, no further params are needed.

`/dating-feedback/accept` `post` `user`

Sets the date as accepted from this user. This can only take place when the user is authenticated.

Param	Type	Description
date_id	Number	The id of the date.

`/dating-feedback/cancel` `post` `user`

Sets the date to cancelled from this user. This can only take place when the user is authenticated.

Param	Type	Description

Param	Type	Description
date_id	Number	The id of the date.

`/dating-feedback/report` `post` `user`

Report the other user after the date, because he did something wrong. This can only take place when the user is authenticated.

Param	Type	Description
date_id	Number	The id of the date.
reasons	array<Object>	The reasons the user has to report the candidate. Can contain a reason that does not exist yet.

`/dating-feedback/message` `post` `user`

Sends a message to the user after both decided to send the message. This can only take place when the user is authenticated.

Param	Type	Description
date_id	Number	The id of the date.
message	String	The message that will be send after both thought the date was successful.

`/dating-feedback/incorrect-match` `post` `user`

Sets the date to unsuccessful for the user. This can only take place when the user is authenticated.

Param	Type	Description
date_id	Number	The id of the date.

Establishment Availability

The route `/establishment/availability` houses all the data requests concerning the establishment availabilities.

Auth	Type	Route	Description
admin	get	`/establishment/availability/get`	Get all the establishment availabilities.
admin	post	`/establishment/availability/create`	Create an establishment availability.
admin	post	`/establishment/availability/update`	Update an establishment availability.
admin	post	`/establishment/availability/archive`	Archive an establishment availability.

`/establishment/availability/get` `get` `admin`

The route to get the establishment availabilities. This can only take place with admin privileges.

`/establishment/availability/create` `post` `admin`

The route to creating a new establishment availability. This can only take place with admin privileges.

Param	Type	Description
datingEstablishmentId	Number	The id of the dating establishment.
timeSlotId	Number	The id of the time slot.

Param	Type	Description
places	Number	The number of tables available at that timeslot in that establishment.

`/establishment/availability/update` `post` `admin`

The route to update a establishment availability. This can only take place with admin privileges.

Param	Type	Description
datingEstablishmentId	Number	The id of the dating establishment.
timeSlotId	Number	The id of the time slot.
places	Number	The number of tables available at that timeslot in that establishment.

`/establishment/availability/archive` `post` `admin`

The route to update a establishment availability. This can only take place with admin privileges.

Param	Type	Description
datingEstablishmentId	Number	The id of the dating establishment.
timeSlotId	Number	The id of the time slot.
archived	Boolean	The boolean if the dating establishment availability is archived.

Gender

The route `/gender` houses all the data requests concerning the genders.

Auth	Type	Route	Description
none	get	`/gender/get`	Gets all the genders.
admin	get	`/gender/amount`	Gets the amount of genders.
admin	get	`/gender/interest-amount`	Gets the amount of interests.

`/gender/get` `get`

Gets all the genders

`/gender/amount` `get` `user`

Gets the amount of genders by the amount of matches. This can only take place with admin privileges.

`/gender/interest-amount` `get` `user`

Gets the amount of gender interests by the amount of matches. This can only take place with admin privileges.

Image

The route `/image` houses all the data requests concerning the image of the user.

Auth	Type	Route	Description
user	get	`/image/all`	Gets all the images of the user.
admin	get	`/image/amount`	Get the amount of images.

Auth	Type	Route	Description
user	get	`/image/:name`	Get the image with a specific name.
admin	get	`/image/:id/:name`	Get the image with a specific name and id.
user	get	`/image/candidate/:candidateId/:file`	Get the images with a specific name from a candidate.
user	post	`/image/upload`	Upload a photo.
user	post	`/image/delete/:photoObject`	Delete a photo.
user	post	`/image/ordering`	Order the photos.

`/image/all` `get` `user`

Gets all the images a user has. This can only take place when the user is authenticated.

`/image/amount` `get` `admin`

Gets the amount of images. This can only take place with admin privileges.

`/image/:name` `get` `user`

Gets the image with a specific name. This can only take place when the user is authenticated.

`/image/:id/:name` `get` `user`

Gets the image with a specific name and id. This can only take place when the user is authenticated.

`/image/:id/:name` `get` `admin`

Gets the image with a specific name and id. This can only take place with admin privileges.

`/image/candidate/:candidateId/:file` `get` `user`

Get the images from a candidate This can only take place when the user is authenticated.

Param	Type	Description
candidateId	Number	The id of the candidate
file	String	The filename of the image of the candidate.

`/image/upload` `post` `user`

Uploads an image to the server. This can only take place when the user is authenticated.

Param	Type	Description
files	array<File>	The files uploaded.

`/image/delete/:photoObject` `post` `user`

Deletes a specific photo. This can only take place when the user is authenticated.

Param	Type	Description
photoObject	String	The file name to be deleted.

`/image/ordering` `post` `user`

Orders the images in the way he set them. This can only take place when the user is authenticated.

Param	Type	Description
files	array<{name: String, order: Number}>	The files with their new ordering.

Language

The route `/language` houses all the data requests concerning the languages of the system.

Auth	Type	Route	Description
none	get	`/language/get`	Gets the languages that are defined in the database

`/language/get` `get`

Gets the languages of the system from the database.

Location

The route `/location` houses all the data requests concerning the locations of the system.

Auth	Type	Route	Description
none	get	`/location/get`	Gets the locations that are defined in the database

`/location/get` `get`

Gets the locations of the system from the database.

Matching

The route `/matching` houses all the data requests concerning the matching of the system.

Auth	Type	Route	Description
admin	get	`/matching/match`	Run the matching algorithm
admin	get	`/matching/logs`	Get the matching logs

`/matching/get` `get` `admin`

Start the matching algorithm. This can only take place with admin privileges.

`/matching/logs` `get` `admin`

Get the matching logs. This can only take place with admin privileges.

Report

The route `/report` houses all the data requests concerning the report about users.

Auth	Type	Route	Description
user	get	`/report/get`	Get the report if the user is banned.
admin	get	`/report/pending`	Get all the pending reports, without the banned users.
admin	get	`/report/case`	Get all the information about the reported user
admin	post	`/report/feedback`	Report a user with a specific report id.

Auth	Type	Route	Description
user	post	`/report/case-feedback`	Possibility to ban a user after they are reported.

`/report/get` `get` `user`

Basic get request to retrieve the report reason if the user is banned. This can only take place when the user is authenticated.

`/report/pending` `get` `admin`

Get the pending reported users that are not banned yet. This can only take place with admin privileges.

`/report/case` `get` `admin`

Get all the information about the reported user. This can only take place with admin privileges.

Param	Type	Description
reportedUserId	Number	The id of the user.

`/report/feedback` `post` `admin`

Report a user. This can only take place when the user is authenticated.

Param	Type	Description
reportedUserId	Number	The id of the user.
reportType	Number	The id of the report why the user is reported.

`/report/case-feedback` `post` `user`

Taken action for a reported user. This can only take place with admin privileges.

Param	Type	Description
reported_user_id	Number	The id of the user.
banned_report_id	Number	The id of the report why the user is banned.
latest_report	Date	The time the user was last reported.

Report Type

The route `/report-type` houses all the data requests concerning the report types.

Auth	Type	Route	Description
admin	get	`/report-type/get`	Get the report types from the database.
user	get	`/report-type/get-non-archived`	Get the report types that are not archived and allowed to see by the user.
admin	post	`/report-type/create`	Create a new report type.
admin	post	`/report-type/update`	Update an existing report type.
admin	post	`/report-type/archive`	Archive an existing report type.

`/report-type/get` `get` `admin`

Basic get request to retrieve all the report types. This can only take place with admin privileges.

`/report-type/get-non-archived` `get` `user`

Basic get request to retrieve all the report types that are not archived and that are applicable for the user. This can only take place when the user is authenticated.

`/report-type/create` `post` `admin`

Create a new report type. This can only take place with admin privileges.

Param	Type	Description
description	String	The description of the report type.

`/report-type/update` `post` `admin`

Update an existing report type. This can only take place with admin privileges.

Param	Type	Description
id	Number	The id of the report type.
description	String	The description of the report type.

`/report-type/archive` `post` `admin`

(Un)Archive an existing report type. This can only take place with admin privileges.

Param	Type	Description
id	Number	The id of the report type.
archived	Boolean	Whether the report type is archived or not.

Time Slot

The route `/time-slot` houses all the data requests concerning the time slots.

Auth	Type	Route	Description
none	get	`/time-slot/get`	Get the time slots from the database.
none	get	`/time-slot/get-next`	Get the time slots that are in the future.
admin	get	`/time-slot/partition`	Get the time slot partitions of the establishments. These partitions are the amount of entries and the total amount of places.
admin	post	`/time-slot/create`	Create a new time slot.
admin	post	`/time-slot/update`	Update an existing time slot.
admin	post	`/time-slot/archive`	Archive an existing time slot.

`/time-slot/get` `get`

Basic get request to retrieve all the time slots.

`/time-slot/get-next` `get`

Basic get request to retrieve all the time slots that are in the future.

`/time-slot/partition` `get` `admin`

Get the time slot partitions of the establishments. These partitions are the amount of entries and the total amount of places. This can only take place with admin privileges.

`/time-slot/create` `post` `admin`

Create a new time slot. This can only take place with admin privileges.

Param	Type	Description
time	String	The starting time of the time slot.

`/time-slot/update` `post` `admin`

Update an existing time slot. This can only take place with admin privileges.

Param	Type	Description
id	Number	The id of the time slot.
time	String	The starting time of the time slot.

`/time-slot/archive` `post` `admin`

(Un)Archive an existing time slot. This can only take place with admin privileges.

Param	Type	Description
id	Number	The id of the time slot.
archived	Boolean	Whether the time slot is archived or not.

Bibliography

[1] Monica Anderson, Emily A. Vogels, and Erica Turner. *The Virtues and Downsides of Online Dating*. May 2020. URL: https://www.pewresearch.org/internet/2020/02/06/the-virtues-and-downsides-of-online-dating/ (visited on June 16, 2020).

[2] Brenda K Wiederhold. *Internet Dating: Should You Try It?* 2020.

[3] Roberto Ferdman. *How well online dating works, according to someone who has been studying it for years*. Mar. 2016. URL: https://www.washingtonpost.com/news/wonk/wp/2016/03/23/the-truth-about-online-dating-according-to-someone-who-has-been-studying-it-for-years/.

[4] Judy A. McCown et al. "Internet Relationships: People Who Meet People". In: *CyberPsychology & Behavior* 4.5 (2001). PMID: 11725652, pp. 593–596. DOI: 10.1089/109493101753235188. eprint: https://doi.org/10.1089/109493101753235188. URL: https://doi.org/10.1089/109493101753235188.

[5] Janelle Ward. "Swiping, matching, chatting: Self-Presentation and self-disclosure on mobile dating apps". In: *Human IT: Journal for Information Technology Studies as a Human Science* 13 (2016), pp. 81–95.

[6] Maarten H. W. Van Zalk et al. "Who Benefits From Chatting, and Why?: The Roles of Extraversion and Supportiveness in Online Chatting and Emotional Adjustment". In: *Personality and Social Psychology Bulletin* 37.9 (2011). PMID: 21673194, pp. 1202–1215. DOI: 10.1177/0146167211409053. eprint: https://doi.org/10.1177/0146167211409053. URL: https://doi.org/10.1177/0146167211409053.

[7] Elisabeth Timmermans and Cédric Courtois. "From swiping to casual sex and/or committed relationships: Exploring the experiences of Tinder users". In: *The Information Society* 34.2 (2018), pp. 59–70. DOI: 10.1080/01972243.2017.1414093. eprint: https://doi.org/10.1080/01972243.2017.1414093. URL: https://doi.org/10.1080/01972243.2017.1414093.

[8] Jonathan D. D'Angelo and Catalina L. Toma. "There Are Plenty of Fish in the Sea: The Effects of Choice Overload and Reversibility on Online Daters' Satisfaction With Selected Partners". In: *Media Psychology* 20.1 (2017), pp. 1–27. DOI: 10.1080/15213269.2015.1121827. eprint: https://doi.org/10.1080/15213269.2015.1121827. URL: https://doi.org/10.1080/15213269.2015.1121827.

[9] Yang Xiaojie. "Analysis of DBMS: MySQL Vs PostgreSQL". In: *Kemi-Tornio University of Applied Sciences Technology* (2012), p. 66.

[10] Neoteric. *Single-page application vs. multiple-page application*. Mar. 2018. URL: https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58.

[11] Pramod Sadalage and Martin Fowler. *Evolutionary Database Design*. May 2016. URL: https://martinfowler.com/articles/evodb.html.

[12] Alberto Gimeno. *Node.js multithreading: What are Worker threads, and why do they matter?* Jan. 2019. URL: https://blog.logrocket.com/node-js-multithreading-what-are-worker-threads-and-why-do-they-matter-48ab102f8b10/.

[13] NodeJS. *Don't Block the Event Loop (or the Worker Pool)*. Feb. 2020. URL: https://nodejs.org/en/docs/guides/dont-block-the-event-loop/.

[14] Jessie Leung. *The Test Pyramid*. May 2019. URL: https://medium.com/better-programming/the-test-pyramid-80d77535573 (visited on June 16, 2020).

[15] Ekaterina Novoseltseva. *8 Benefits of Unit Testing*. Jan. 2017. URL: https://dzone.com/articles/top-8-benefits-of-unit-testing (visited on June 16, 2020).

[16] *What are browser developer tools?* URL: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_are_browser_developer_tools (visited on June 16, 2020).

[17] World Health Organization. *Coronavirus*. URL: https://www.who.int/emergencies/diseases/novel-coronavirus-2019 (visited on June 19, 2020).

[18] Pekka Abrahamsson et al. "Agile Software Development Methods: Review and Analysis". In: *VTT publication* (2002).

[19] Pete Pizzutillo. *Static analysis: Leveraging source code analysis to reign in application maintenance cost*. Dec. 2013. URL: https://www.castsoftware.com/blog/static-analysis-leveraging-source-code-analysis-to-reign-in-application-maintenance-cost (visited on June 16, 2020).

[20] Xiaoyun Yang. *JavaScript is a loosely typed language, meaning you don't have to specify what type of information*. Jan. 2018. URL: https://medium.com/@xiaoyunyang/javascript-is-a-loosely-typed-language-meaning-you-dont-have-to-specify-what-type-of-information-137408d54fc7 (visited on June 16, 2020).

[21] Cory Kapser and Michael Godfrey. ""Cloning Considered Harmful" Considered Harmful". In: *2006 13th Working Conference on Reverse Engineering* (2006). DOI: 10.1109/wcre.2006.1.

[22] *What is duplicate code?* May 2020. URL: https://www.codegrip.tech/productivity/what-is-duplicate-code/ (visited on June 16, 2020).

[23] Software Testing Help. *Top 6 BEST Python Testing Frameworks*. Feb. 2020. URL: https://www.softwaretestinghelp.com/python-testing-frameworks/ (visited on Apr. 22, 2020).

[24] Robert J Brym and Rhonda L Lenton. "Love online: A report on digital dating in Canada". In: *MSN. ca, February* 6 (2001).

[25] Jeana H Frost et al. "People are experience goods: Improving online dating with virtual dates". In: *Journal of Interactive Marketing* 22.1 (2008), pp. 51–61.

[26] Amjad Hudaib et al. "Requirements prioritization techniques comparison". In: *Modern Applied Science* 12.2 (2018), p. 62.

[27] *Klanten betalen het liefst online: CCV*. Sept. 2019. URL: https://www.ccv.eu/nl/2018/hoe-betalen-klanten-liefst-online-ideal-paypal-tot-creditcard/ (visited on Apr. 28, 2020).

[28] *Stripe*. URL: https://stripe.com/en-nl/about (visited on Apr. 28, 2020).

[29] *Mollie*. URL: https://www.mollie.com/nl/developers (visited on Apr. 28, 2020).

[30] Sarah T Roberts. *Content moderation*. 2017.

[31] Andreas Veglis. "Moderation techniques for social media content". In: *International Conference on Social Computing and Social Media*. Springer. 2014, pp. 137–148.

[32] Dan Gusfield and Robert W Irving. *The stable marriage problem: structure and algorithms*. MIT press, 1989.

[33] Francesco Ricci, Lior Rokach, and Bracha Shapira. "Recommender systems: introduction and challenges". In: *Recommender systems handbook*. Springer, 2015, pp. 1–34.

[34] Richi Nayak, Meng Zhang, and Lin Chen. "A social matching system for an online dating network: A preliminary study". In: *2010 IEEE International Conference on Data Mining Workshops*. IEEE. 2010, pp. 352–357.

[35] Eli J Finkel et al. "Online dating: A critical analysis from the perspective of psychological science". In: *Psychological Science in the Public interest* 13.1 (2012), pp. 3–66.

[36] Austin Carr. "I found out my secret internal Tinder rating and now I wish I hadn't". In: *Fast Company* (2016).

[37] *Powering Tinder® - The Method Behind Our Matching*. Mar. 2019. URL: https://blog.gotinder.com/powering-tinder-r-the-method-behind-our-matching/ (visited on Apr. 22, 2020).

[38] Lukas Brozovsky and Vaclav Petricek. “Recommender System for Online Dating Service”. In: *CoRR* abs/cs/0703042 (2007). arXiv: cs/0703042. URL: http://arxiv.org/abs/cs/0703042.

[39] P. Xia et al. “Reciprocal recommendation system for online dating”. In: *2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. 2015, pp. 234–241.

[40] Anand Rajaraman and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2011.

[41] Karl Pearson. “LIII. On lines and planes of closest fit to systems of points in space”. In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (1901), pp. 559–572.

[42] A.E. Elo. *The Rating of Chess Players, Past and Present*. Ishi Press International, 2008. ISBN: 9780923891275. URL: https://books.google.nl/books?id=syjcPQAACAAJ.

[43] *Angular*. URL: https://angular.io/ (visited on Apr. 22, 2020).

[44] *TypeScript*. URL: https://www.typescriptlang.org/ (visited on Apr. 22, 2020).

[45] *React*. URL: https://reactjs.org/ (visited on Apr. 22, 2020).

[46] Nitin Pandit. *What and Why React.js*. URL: https://www.c-sharpcorner.com/article/what-and-why-reactjs/ (visited on Apr. 22, 2020).

[47] *Introducing JSX*. URL: https://reactjs.org/docs/introducing-jsx.html (visited on Apr. 22, 2020).

[48] *Vue.js*. URL: https://vuejs.org/ (visited on Apr. 22, 2020).

[49] *Angular vs React vs Vue: Which is the Best Choice for 2019?* Aug. 2019. URL: https://hackernoon.com/angular-vs-react-vs-vue-which-is-the-best-choice-for-2019-16ce0deb3847 (visited on Apr. 22, 2020).

[50] *Build Truly Native Mobile Apps with Vue: NativeScript*. URL: https://www.nativescript.org/vue (visited on Apr. 22, 2020).

[51] Youssef Bassil. *A Comparative Study on the Performance of the Top DBMS Systems*. 2012. arXiv: 1205.2889 [cs.DB].

[52] Steven A Gabarró and Steven A Gabarrâo. *Web Application Design and Implementation: Apache 2, PHP5, MySQL, JavaScript, and Linux/UNIX*. Wiley-Interscience, 2007.

[53] Ivan Zoratti. “MYSQL security best practices”. In: (2006).

[54] *Node.js*. URL: https://nodejs.org/ (visited on Apr. 29, 2020).

[55] S. Tilkov and S. Vinoski. “Node.js: Using JavaScript to Build High-Performance Network Programs”. In: *IEEE Internet Computing* 14.6 (2010), pp. 80–83.

[56] K. Lei, Y. Ma, and Z. Tan. “Performance Comparison and Evaluation of Web Development Technologies in PHP, Python, and Node.js”. In: *2014 IEEE 17th International Conference on Computational Science and Engineering*. 2014, pp. 661–668.

[57] *Jest · Delightful JavaScript Testing*. URL: https://jestjs.io/ (visited on Apr. 24, 2020).

[58] *the fun, simple, flexible JavaScript test framework*. URL: https://mochajs.org/ (visited on Apr. 24, 2020).

[59] *Top 5 Javascript Testing Frameworks*. Oct. 2019. URL: https://www.browserstack.com/guide/top-javascript-testing-frameworks (visited on Apr. 24, 2020).

[60] *Snapshot Testing · Jest*. URL: https://jestjs.io/docs/en/snapshot-testing (visited on Apr. 24, 2020).

[61] *Headless Browser Testing Awesomeness Pros and Cons*. Nov. 2018. URL: https://testguild.com/headless-browser-testing-pros-cons/ (visited on Apr. 24, 2020).

Glossary

Admin App The front-end VueJS application that is used by the Administrator, with the Vuetify UI framework as the main framework. 15–17, 25, 31, 32, 35, 47–49, 95, 109–112, 135, 137

Administrator A stakeholder that will use the Admin App. vii, 5–7, 16, 17, 25, 30–32, 35, 49

Adobe XD Adobe XD is the design tool that was used to create the initial designs for the application. 9–11

Angular Angular is a researched framework for this project, like VueJS. Angular did not seem like a good fit for this project and was not selected to work with. 7, 81, 82

API Application Programming Interface; Software that acts as middle-ware. It allows two independent applications to send and retrieve data of each other. 12, 15, 17, 18, 35, 42

BookshelfJS A JavaScript ORM for NodeJS, it utilizes KnexJS for its functionality. 18

BootstrapVue BoostrapVue is the bootstrap library for VueJS. It allows the creation of properly styled pages to a professional standard. 16, 47, 135, 137

CircleCI Continuous Integration & Deployment Service that the project used in the early stages for the CI. It was however dropped and changed to GitHub Actions. 24

Client App The front-end VueJS application that is used by the client, with the BootstrapVue UI framework. 6, 9, 12, 15, 16, 25, 31, 35, 47, 49, 137

CSS Cascading Style Sheets; Is the styling sheet for pages written in markup language like HTML. 137

Dating Establishment The establishment at which dates could take place, these establishments are owned by the external Dating Establishment. vii, 4, 5, 7, 20, 32, 34, 35, 37, 49, 102–104, 106, 135, 137

DEO Dating Establishment Owner; a stakeholder that connects to the application to provide dating locations with their Dating Establishment. 6, 7

Discord Application that allows voice calls and screen sharing. Discord was used to allow the developers work virtually together. No physical meetings could take place due to COVID-19. 39, 44

ES6 ECMAScript 6; A programming languages that is inspired on JavaScript. 15

ESLint Linter for locating static errors in JavaScript code. 43

Establishment App The front-end VueJS application that is used by the Establishment, this is not an actual component, but could be implemented in the future. 49

Express The web server framework for NodeJS, that allows the web pages to be hosted on a server. 18, 19

Firebase Platform that allows implementation of several features. The feature that this project uses is the Authentication Services. 15, 16, 18, 24, 25, 37, 48, 49

FireFly The company that ordered the creation of the entire application. v, vii, 1, 37, 45, 56, 60, 135

FireFly Dating The name of the application that was commissioned by the FireFly company. vii, 1, 3, 5–7, 9, 15, 17, 19, 24, 26–31, 37, 45, 48, 49, 56, 60, 70, 78, 80, 84, 102, 106

Git Git is a version-control system that is used for software development. A key feature is that it tracks changes in the code base. 136

GitHub The hosting platform on which the application version control is hosted. This hosting platform uses Git. 10, 24, 36, 41, 136

GitHub Actions Continuous Integration Software for GitHub Projects. 24, 135

GitHub Boards Project board of GitHub. These boards help organize and distribute different tasks within the team. The usage of boards lead to efficient and effective team cooperation. 41

Google Drive Storge platform in the cloud that houses many different notes taken by the team. 39, 136

Google Hangouts Video conferencing, which was used to have video calls instead of physical meetings, due to COVID-19. 39, 44

Google Sheets Google Sheets is a spreadsheet tool that is accessable within Google Drive. 41

HTML HyperText Markup Language; The standard markup language for pages of your web browser. 15, 18, 135, 136

JavaScript The main programming language that is being used within this application. It has first-class citizens and is object-oriented. 8, 12, 13, 18, 22, 43, 48, 81, 85, 135–137

Jest JavaScript testing framework that is being used in the front-end. Initially Mocha was being used as the testing framework in the front-end, but was unsuitable and was dropped. 8, 22, 47, 85

JSDoc Markup language of JavaScript. It allows to explain what each function does and what the parameters are. 43

JSX It stands for JavaScript XML. It allows to write HTML and JavaScript properly together. 81, 82

KnexJS A SQL query builder that allows the creation of maintainable queries in JavaScript for MySQL and other databases. 18, 19, 135

linter A static analysis tool, that checks if the code base is conform to the set of rules. 43, 135

MariaDB A MySQL relational DBMS, that allows a Graphical User Interface instead of Command Line. 18

McCabe Complexity Cyclomatic complexity, is a tool that indicates how complex a function is. Functions with a large McCabe Complexity are complex and difficult to test, and should thus be avoided if possible. 93, 94

Mocha Test framework for JavaScript that focuses on NodeJS programs. Initially it was used in the front- and back-end, but was later changed to only be included in the back-end. 8, 21, 22, 85, 136

MoSCoW These are the requirements for the applications, separated in Must have, Should have, Could have and Won't have. 25, 40

MySQL MySQL is the DBMS that is being used within the application. MySQL uses SQL. 8, 18, 136

NodeJS JavaScript runtime environment that allows the creation of a dynamic server. It runs the script outside the web browser that created requests. 8, 13, 17, 18, 20, 21, 85, 135, 136

Nodemailer A library for NodeJS that allows applications to send emails. This is a more basic mail client in comparison to SendGrid. 48

OAuth A standard for authentication by using third party accounts. 15, 18

PM2 PM2 is a tool that enforces the up-time of the JavaScript runtime NodeJS. 36

PostgreSQL PostgreSQL as a DBMS like MySQL. PostgreSQL is not being used in this application.. 8

Product Owner The stakeholder that owns the entire application. vii, 4, 6, 9–13, 23, 32, 35, 39–42, 49

Python A programming language that has many efficient libraries and excellent tools available. It is used in the application for the matching of users. 13, 20, 22, 48

React React is a researched framework for this project, like VueJS. React did seem like a good fit for this project, but was not selected as VueJS was slightly better. 7, 81, 82

Scrum Scrum is a practise to allow developing of systems on an uniform manner. It relies on the agile principles. 39–42, 44

SCSS Sassy CSS is a superset of CSS. It introduces variables and conditional styling. 10, 15

SendGrid Mail Service that allows sending of email with many additional functionalities. 18, 48, 136

Time Slot A Time Slot reserved at a Dating Establishment at which a date could take place. vii, 5, 6, 20, 29, 32, 34, 49, 103, 106, 110

TypeScript TypeScript is a superset of JavaScript. It tries to make the language stricter. 81

user A stakeholder that will use the Client App. vii, 1, 3–7, 9, 12, 15, 18–20, 23–32, 35–37, 45, 49, 102–107, 137

VueJS The model-view framework that utilizes JavaScript framework. It is used for building the entire front-end side of the application. 7, 10, 15, 21–23, 35, 82, 85, 94, 135, 137

Vuetify A VueJS UI library that is similair to BootstrapVue. This library houses more functionality for working with data in comparison to BootstrapVue. It is used on the Admin App. 16, 47, 135

WhatsApp WhatsApp is a communication tool that was used in this project to communicate after the work hours. 39

Abbreviations

ACID	Atomicity, Consistency, Isolation, and Durability. 82
BEP	Bachelor End Project. v, 1, 12, 13, 25, 32, 35, 37, 39, 41, 42, 44, 48, 70
CCM	Commercial Content Moderation. 77
CF	Collaborative Filtering. 7, 78–80, 86
CI	Continuous Integration. 22, 24, 43, 135
CRUD	Create, Read, Update and Delete. 16, 17
DBMS	DataBase Management System. 18, 83, 85, 136
E2E	End to End. 22
EJS	Embedded JavaScript templating. 18
FAQ	Frequently Asked Questions. 25, 26, 34, 35
GDPR	General Data Protection Regulation. 35, 76
JSCPD	JavaScript Copy/Paste Detection. 43, 44
JSON	JavaScript Object Notation. 82
MVP	Minimal Viable Product. 1, 5, 6, 12, 42, 43, 47, 70, 77, 86, 88, 89
ORM	Object-Relational Mapping. 18, 104, 135
PSP	Payment Service Provider. 6, 7, 32, 76, 77, 86
SIG	Software Improvement Group. 35, 43, 44, 76, 93, 94
SQL	Structured Query Language. 22, 82, 83, 86, 136
TOS	Terms of Service. 35, 37
UGC	User-Generated Content. 77
UI	User Interface. 9, 10, 40, 47, 135, 137
UML	Unified Modeling Language. 98, 99
UX	User Experience. 9