Gym Planner

Create, share and enjoy gym exercises with simplicity!

K. van Heel
E. Ilgin
V. Ionescu
J. Ketelaar
Y. Noor

Research Report
Bachelor End Project

Preface

This report concludes the Bachelor project course, which is a compulsory course for obtaining the Bachelor of Science in Computer Science at the Delft University of Technology. This report contains all the information related to this 10-week project. The project has been finished in assignment for the start-up company Gymplanner. Gymplanner aims to create a forum where both teachers and students can create and share sport exercises. Right now there is no market where either teachers or students can find or create sport exercises. There are some websites that try to tackle the problem but they all have their own shortcomings. These will be discussed later in the report. Furthermore, the goal of this report is to inform the reader on the completed work and the process from idea to product. The report also contains recommendations for future work. The contactperson for Gymplanner is ir. H.J. Griffioen. The coach for this project was Prof.dr.ir. A. Bozzon and lastly, the general bachelor project coordinator has been Prof.dr.ir. H. Wang.

Summary

Gymplanner is a small company that offers students and teachers a forum where they can create and share exercises and training plans meant for any sport or physical education. For example, sports teachers such as high school teachers or elementary school teachers need to create a new sports plan for every class and Gymplanner tries to supply them with predefined training courses for every sport. All training courses are created by users of the system and then can be rated by other users based on a 5-star mark. Users can either choose to create exercises for exclusive use in a private group, or choose to share them with everyone on public pages. In order for exercises to be publicly shared, the content will be checked by verified users or system administrators. This way the company ensures high quality for all its public content.

The assignment was to build a system from scratch that was both maintainable and extendable. Furthermore, the system had to be straightforward and self-explanatory, in the sense that users should be able to find what they are looking for in the least amount of clicks possible. For this, we made sure that during design phases everything was kept as simple and clear as possible. After meeting with our client and hearing the requirements from the client and his stakeholders, we performed research into the system requirements. This requirement analysis helped us determine what needed to be implemented. In order to measure success, we set criteria that will help us determine if the project has succeeded or not. During the implementation process, we stumbled upon some inconsistencies towards the requirements that asked for a refactor of our designed system and database. At the end of week 6, the code got submitted and reviewed by the Software Improvement Group. In their feedback we were given a rating of 4.6 out of 5 for maintainability. They noted that some methods were too long and this hindered us from getting the perfect score. We processed this feedback and the result is a working, maintainable system.

Contents

1 Introduction

Gymplanner is a start-up company founded by Harm Griffioen. It is an online platform that aims to realize a forum for everyone inflicted with sports. The goal of the platform is to offer a medium where people can create and share exercises with each other to help improve the quality and accessibility of physical education. The platform's main focus are teachers and sport coaches that need to create a new sport plan every single time they are giving a course. The forum should be accessible and cover a wide variety of sport activities.

The motivation for creating such a platform is the research performed by the client, which decided that there was a large market available for people looking to find quality exercises. In the research, our client also encountered a couple of different platforms that attempt to fulfill this demand. However, these platforms lacked some important features in our eyes. Due to the lack of a complete platform that tackles all these issues, there is still a large portion of the market that is not satisfied.
The challenge for us is, thus, to design and implement a system that tackles the issue of not having a high-quality exercise sharing platform. The problem is not necessarily the internet being void of an exercise platform, but that all the available platforms do not conform to the market's needs. This is why we created a platform that takes the needs of the market into account and provides an elegant solution. Our platform is built on the core fundamentals of simplicity and quality assurance. We have created intelligent systems to make sure there are no difficult steps in achieving a user's goal when visiting the website.

The systems that are in place revolve around the smart search feature and text analysis based tag recommendation system. First off, our smart search feature allows users to find what they are looking for without requiring them to follow strict guidelines on how to search. Secondly, our text analysis-based tag recommendation system allows users to create exercises without having to put effort into coming up with good and relevant tags. These features ensure the first core fundamental of simplicity. Furthermore, the system is implemented in such a way that performance will not drop if we reach the full potential of the current available market, and even scaling beyond that should not have a noticeable impact. Finally, we designed the platform in such a way that quality assurance is done by crowdsourcing part of the task to dedicated users. This allows us to outperform the other platforms in this aspect as they require very active administration for a big user base. This design aspect assures the second core fundamental of quality assurance.

In the next chapter, the requirements for the application will be stated. These are needed to give a clear overview of what functionalities are required from the system and which requirements will not be implemented. The requirements are deduced in such a way that they are compliant with the wishes of our client. Then, an overview of the technologies and frameworks being used will be given and motivated. Chapter 4 will shed light on the system design choices. It gives a clear overview of the different components in the system and how they are constructed. Chapter 5 will give insight into how the project was structured, and how we made sure every week would contain more functionality. It also describes the development resources that were used. Furthermore, chapter 5 describes the feedback that was returned by the Software Improvement Group and how this feedback has been processed and implemented. Chapter 6 will give more insight into the actual implementation of the system. The project, and whether it has been a success will be evaluated in chapter 7. Chapter 8 will highlight recommendations and lastly, chapter 9 will contain a conclusion.

2

Requirements

The first step in any software project is to make clear what functionality the system requires. Before implementing anything, both the developers and the client needs to be aware of what the system must be able to do and what the system will not be able to do. This is achieved by creating functional and non-functional requirements for the application.

2.1 Constructing the requirements

The requirements are determined in such a way that they fit the MoSCoW method [3]. The MoSCoW method is a technique commonly used in fields such as project management, business analysis and software development. Reason for this is because it is a prioritization technique which gives priority to certain requirements over others. It thus gives a clear distinction between important requirements and less important requirements.

Using this method is also convenient for evaluating the system. The success of the system is measured by means of success metrics. Success metrics can be established by means of the requirements. For all the must-have requirements it is necessary that they are all implemented. The must-have requirements are the minimum requirements in order to call it a working system, also called MVP. Without some of these functions, the system will not operate, thus they all need to be implemented. For the should-have requirements we define the success metric as at least 75% of all functions implemented. Without the should-have requirements the system will still operate normally, however, it will miss some core functionality. The could-have requirements are requirements that could be added to the system when there is time left. However, they are not part of the fundamental functions of the system and without them, the users of the system will not experience missing functions. They are, therefore, not used in defining the success of the system. Lastly, the would-have requirements, commonly referred to as won't-have requirements, are requirements or functions that are not planned into the schedule for this project. Therefore, these will also be occluded in defining the succession of the system. In summary, the success metrics are defined as;

- All the must-have requirements are implemented and functional
- 75% of the should-have requirements are implemented and functional
- The system is online and functional
- 90% of the non-functional requirements are valid

2.1.1 Must have requirements

Most important requirements for the system. Without them, the system is not operational

- A visitor must be able to create an account, and delete this account
- A user must be able to log in and log out
- A user must be able to request a new password
- A user must be able to alter their e-mail address
- A group can be private or public
- A public group can be viewed by users and visitors without an account
- A public group can be joined by any user
- To join a private group, a user must send a request which must be accepted by at least the group moderator
- The contents of a private group is only visible to its members
- A group must be able to be found by searching for the group name
- A user must be able to create a new exercise using a title, image/schematic, description and optional remarks
- An exercise must be able to be found by searching on title, tags, rating and username
- A public exercise must be checked for quality by a system administrator before it is added
- A user creating an exercise will directly see how the exercise will be exported in PDF format
- A training must be part of a group.
- A user must be able to export a training (group of exercises) in PDF format
- An training must be able to be found by searching on title, tags, rating and creator
- The group administrator must be able to set the group to public/private
- The group administrator must be able to promote group members to *group moderator*
- The group administrator must be able to modify and delete exercises
- The group administrator must be able to modify and delete a group
- The system administrator must be able to log in and out to an administrator panel
- The system administrator must be able to edit the database through the administrator panel

2.1.2 Should haves

- A user should be able to rate an exercise or a training, based on a 1 to 5 star rating
- A user is able to recommend exercises or training courses in a group
- A training and exercise should contain a list of tags provided by its creator
- A tag should be an existing one or newly created
- A training and exercise should show the average rating it received
- A user should be able to communicate through voice with other users of the same group.
- A user should be able to communicate through text messaging with other users of the same group.
- A user should be able to see dynamically content changing by using synchronization

- A group should contain a list of available equipment
- Exercises/training courses that require equipment not present in the group should receive a mark to indicate it
- A group should contain zero or more templates
- A tag should be an existing one or newly created
- An exercise should contain a list of equipment required for the exercise
- A user should be able to export a training (group of exercises) in PDF format
- A user should be able to use a group's template when exporting a training course to PDF
- A training should contain a list of equipment required for the exercise

2.1.3 Could have requirements

- The group administrator could be able to assign/revoke permissions to/from the other ranks
- The user could be able to assign a tag to his account which has to be verified
- The user could be able to comment exercises
- The user could change text size of page
- A user could comment on an exercise/training
- The exercise creation tool could integrate the draw tool from gymplanner.nl
- Users could work together on the exercise creation tools by means of a collaboration framework such as TogetherJS
- Users could be able to communicate through private messages

2.1.4 Won't have requirements

- The system won't implement a payment system

2.2 Non-Functional Requirements

Below are the non-functional requirements described of the system that are also being evaluated at the end of the project in the success metrics. The discussion about these metrics can be found in 7.4.2

2.2.1 Development standards

- The back-end will be developed using NodeJS and TypeScript
- The front-end will be developed using React and TypeScript
- The main IDE used throughout the project will be Jetbrains WebStorm

2.2.2 Access Security

- The password shall never be stored at any given moment in time
- Payment information shall never be stored at any give moment in time
- Access permission for system data can only be modified by server administrators
- Personal information will not be mandatory

2.2.3 Usability

- A visitor shall require no more than 5 actions to access public information
- The main functions should be accessible within 5 actions from any state
- There will not be any functions hidden in the sub menus of drop-down menus

2.2.4 Accessibility/Availability

- The user is able to always create a new account as long as the system is running
- The user is always able to log in
- The system is able to handle 100 users simultaneously without creating server delay
- After creating an exercise/training/group, it will be accessible after at most 5 seconds
- Search results will be accessible within 5 seconds of searching

2.2.5 Confidentiality

- Documents sent for user-label validation will not be stored indefinitely
- Documents sent for user-label validation will not be accessible for non-admins
- Documents will be handled according to GDPR standards
- A user will be able to remove all their personal data from the website
- A user will easily be able to see all the data the website has on them

2.2.6 Maintainability

- The system will be built using an open source end-to-end software development platform with built-in version control, issue tracking, code review, CI/CD and more
- The code will be well documented to elaborate what functions do
- The code will be properly tested to make sure the functionality is in order

2.2.7 Efficiency

- Any interface between user and automated system will take no longer than 2 seconds
- The application should never be frozen waiting for a response/input

2.2.8 Portability

- The GUI is designed for responsiveness on modern devices supporting generic web browsers
- The application should not function differently between different devices

3

Research

3.1 Overview

This chapter includes the problem analysis and problem definition, as well as our research done for the project. This includes finding current solutions to our problem definition, a feasibility study and all research into current technologies needed for implementation. Most of the research has been done in the second week of the project. After meeting with the client and establishing the requirements for the system, we needed to verify the feasibility of the requirements given the time and budget. In section 3.3 we show several platforms in this area that already exist and in section 3.4 we describe how Gymplanner differs from these platforms and how we solve the main problem.

3.2 Problem definition

Sports club teachers and members, as well as gym school teachers and students, seek the option to share and retrieve training exercises from a central platform with ease. In the current situation, exercises are shared between these individuals through various means of communication that lack organization and control. A platform is required where these people are offered the ability to effortlessly host their exercises and possibly even offer them to a larger group of people than just those within their organization.

3.3 Available Solutions

To tackle the problem definition in the right way we first checked the currently available options that attempt to solve this and discuss what they lack. Below, we have listed the two solutions that came closest to achieving our goal.

3.3.1 Gymspiratie

Gymspiratie is a platform that provides gym exercises for primary- and high school teachers. It tackles a part of the problem for a very specific group of people. This site lacks the ability for teachers and/or students to create their own exercises, they only offer exercises made or added by the administrators of the website. Gymspiratie also does not offer exercises for members of sporting clubs or high school teachers and, thus, is not usable by the target audience described in the problem definition. Source: *https://www.gymspiratie.nl/*

3.3.2 Xpsnetwork

XPS Network is a platform on which personal trainers, coaches and athletes can host, create and share training exercises. It also provides a way to measure results such as body weight, nutrition intake, lap times, etc. It offers many things Gymplanner also plans to offer. However, it does not provide a fully web-based application, but rather a native client for 4 different platforms. The web app only offers limited functionality, which is a huge bottleneck for the general audience. Furthermore, XPS Network's user interface is very outdated and not intuitive. Lastly, training schemes do not have additional constraints such as materials that are required, which makes it difficult for gym teachers to find content they can use. Source: *https://www.xpsnetwork.com/*

3.4 Problem Analysis

In this section, we will briefly discuss the features Gymplanner offers that tackles the shortcomings of the previous stated alternatives.

3.4.1 Usage

Gymplanner aims to improve current solutions by expanding its target audience from organizations and professional trainers to include the general public, physical education teachers and smaller organizations. The platform should be relevant to all of these different groups while using the same user interface. This means that using the interface should require no prior training, and should not be considered as a time-consuming process. In other words, we strive for an intuitive design.

3.4.2 Privacy and Accessibility

In order to preserve privacy and accessibility, the user needs a way to select with whom they share their exercises. For example, big companies receive a part of their income from selling training courses and exercises. They would want to keep their data private, sharing it only with their customers. While non-profit organizations and individuals usually have no economic reasons to keep their data private. They could have their own reasons for choosing who they share it with.

3.4.3 Full text search

We want users to be able to find exercises and trainings easily through a simple search bar without having to follow strict search requirements. For this reason we want to implement an algorithm that allows the user to fully search based on different attributes of both trainings and exercises. These results are then shown based on the similarity with the search query.

3.4.4 Tag recommendation

To simplify creating an exercise we want to have a feature that analyzes the description and title of an exercise and suggest tags based on this. This feature allows the user to create an exercise and not have to think about the tags that might be relevant to their exercises. Secondly this feature also gives us the option to put a technical challenge inside of our system.

3.5 Course of action

When we concluded that the given assignment would be feasible we created a plan of action that aimed at looking at the big picture. As we had to write everything from scratch we set up some mayor guidelines that would help us with finishing the system on time. A clear overview of the global planning will be given at the end of this section. First, we needed to discuss different technologies and which would be better for our project. These technologies include whether we would want a relational database system which integrates SQL or non-relational database systems that would have noSQL. Furthermore, we needed to decide on what coding language to use for both back-end and front-end technologies. Once all of these technologies had been decided on and the initial system had been set up, we could start working on implementing functionality. Deadlines were usually set at the end of each week, marking the end of the sprints. At these deadlines, we concluded what was finished and what issues had to be pushed to the next week. Because the system was built from scratch, the end of the week would also mark a point in which we would tag our client such that he could see for himself what happened that week and whether he was satisfied with our choices. Other mayor deadlines were the end of week 5 in which we planned to have all must-have requirements, the end of week 6 which marked the point where the Software Improvement Group would evaluate our code. Midweek 7 as we planned on showing both our client and our coach a demo of the system so far. Week 9 for the second upload to the Software improvement Group and last but not least, week 10, the week in which the system has to be finished.

Week	Tasks
1	Meet with client and create requirements
2	Feasibility study and research report
3	Set up project, create backlog for first sprint, familiarize with new languages
4	Implement back-end, set-up database, design first pages
5	Integrate back-end to frond-end, design and add components on front-end
6	Front-end implementation and first SIG submission
7	Implement SIG-feedback, first demo deadline
8	Process feedback given by client and coach, deadline draft report
9	Last sprint, second SIG submission, deadline final report
10	Tweak the final product
11	Product demo

3.6 Back-end technologies

This chapter will discuss the technologies that are being used for the back-end of the system. In choosing a suitable programming language for the server-side development of the system, one must have a clear understanding of the differences between the alternatives. This chapter only discusses and motivates the technologies that we are using. The research into other technologies and their results can be found in Appendix A

3.6.1 NodeJS

The JavaScript run-time environment NodeJS is being used as infrastructure to build and run the web application. This open-source and cross-platform environment enabled us to build the server-side of the application supporting a wide range of packages through the NodeJS package manager (NPM). NodeJS is one of the most used back-end technologies and there is a wide arrangement of knowledge available on the internet. This was also the case for the other technologies we researched such as PHP and Java A. However, NodeJS has better scalability and we liked the idea of having the same syntax for client-side code and server-side code. Also, a large portion of the group had experience working with NodeJS, whereas no one had experience working with PHP.

3.6.2 Express

Express is a web application framework that provides features for web and mobile applications. Since we are building a web platform that is accessible through both web- and mobile- browsers, we chose to use Express features on top of NodeJS. Express allows us to design our back-end as an API to which our front-end connects. Express allows for easy set-up regarding HTTP calls that are made by the front-end to acquire information for the database or perform certain computations. This framework has become the standard for developing servers on NodeJS, which is also widely accepted by developers using the famous MEAN (Mon-

goDB, ExpressJS, AngularJS, NodeJS) software stack.

3.6.3 PassportJS

Since we decided in the early stages to use OAuth for authentication purposes, it was logical to use some kind of middleware since it is a waste of time to reinvent the wheel, so to speak. During our research period we struck upon PassportJS, an authentication middleware for NodeJS that has a set of strategies that supports authentication through Google, which is what we were aiming for. It also fits perfectly on our application due to its Express base. We, therefore, used the Google OAuth strategy provided by PassportJS for our authentication system.

3.6.4 Testing framework: Jest

Jest is a JavaScript testing framework that is included with React by default as it is also the framework Facebook uses for their code testing. It offers all the functionality that is desired from a good testing framework: different code coverage, easy mocking, extensive assertions, isolated testing and code coverage reports. These are basically all the tools we needed in order to test our features, which is why we found it illogical to use an external framework such as MochaJS. Jest is used in this project for both back-end, as it supports Node, and front-end testing purposes.

3.7 Front-end technologies

In choosing front-end technologies we again kept scalability and maintainability in mind. We needed a technology that would still be maintained in the future, thus we chose not to look for a super modern hyped technology that hasn't passed the test of time. The foundational languages of web applications being used widely are HTML (Hypertext Markup Language) and CSS (Cascading Style Sheets). HTML is used to describe the structure of the web page, whereas CSS describes the presentation of a page.

3.7.1 JavaScript

For dynamically changing the visible content several languages can be used. JavaScript, however, is by far the most widely used front-end technology for doing this. Some of the other languages just get transpiled into JavaScript and exist to extend the syntax of JavaScript into a more readable version. Because it is the industry standard, and everyone had some experience with JavaScript, JavaScript was the go-to language for front-end development.

3.7.2 TypeScript

TypeScript is a programming language that is developed by Microsoft. It is based on JavaScript but it adds the ability to specify types and automatically restricts types. TypeScript is compiled to normal JavaScript and is designed to be used for larger applications. The reason for using TypeScript in our project is that it makes the code more maintainable. Both for us as project members, but also for future developers. Being enforced to type variables, functions and classes makes it much easier to understand the code and what is happening. TypeScript only enables type checking during compilation and so it has no impact on performance post compilation. It is purely a measure to make sure that there are no type errors in the code and so that the code-base is more maintainable.

3.7.3 React

React is a JavaScript library that transforms the standard web development process. Normally you would create HTML files for pages and link JavaScript functionality through script tags. With React, JavaScript and HTML have been integrated into JSX, which is an XML-like syntax extension for JavaScript. This allows you to write HTML tags and directly input JavaScript between the tags. Together with React, JSX can be used to convert the standard web development programming paradigm into object-oriented programming. This allows for the abstraction of components on pages with their HTML and JavaScript being intertwined. Using this technology we can not only reduce the number of redundant lines of code but also allow for the entire project to be more maintainable.

Besides changing the way web development is approached, React also allows developers to make stateful components. These components hold certain data as a state and upon the state being changed the component will automatically be re-rendered without a refresh being required. This functionality makes it possible to have swift and interactive websites.

The way that React accomplishes stateful components is through a virtual Document Object Model (DOM). React uses an in-memory data structure that represents the DOM, this allows React to compare the current actual DOM with their virtual version and only re-render the components that need to be updated. This feature allows React to be very efficient and have a high performance. It is because of features like these, we decided to choose React over its counterparts Angular or Vue.

3.7.4 Redux

Redux is a state management tool served as a library that works with React, it allows applications to have a central store in which the state is managed. Any component that has stateful information will need to connect to the store and get or update data there. This library allows us to share the state between multiple different components with simplicity. Without this library, any two components that share state data require the state to live in a shared parent component, which can be extremely difficult to manage as the application grows in terms of components. In order to do so, we would need to create different callbacks in order to set the state from the child components. Redux makes this whole thing simple by allowing you to specify a store structure that you can easily request any component to have access to.

There are a few alternatives to Redux when it comes to state management tools. Most of the alternatives focus to improve the Redux workflow. However, these are preferred for developers who are already familiar with Redux and understand all of its inputs and outputs.

However, MobX is one of the alternatives worth to mention which uses an object-oriented programming approach whereas Redux follows a functional programming paradigm. MobX offers also a lot of abstraction which makes it easier to learn for the developers to catch on early in the development process. Despite some useful advantages of MobX over Redux when it comes to abstraction, multiple store, observable data(tracking changes automatically) and mutability.

Since we value scalability and maintainability very high in the front-end part of this project, we decided to choose Redux over MobX. Redux' functional programming principle and pure functions allow us to debug easily and maintain the code in a proper way.

3.7.5 Database

We use PostgreSQL for the database to connect to the back-end. It is a relational database management system supporting SQL. PostgreSQL is very similar to MySQL except for some performance differences and how easy it is to tweak the source. We researched multiple articles to find which database system to use for our needs and in most of them, it was said that PostgreSQL is a better out of the box solution. PostgreSQL, for example, is ACID compliant whereas for MySQL to be ACID compliant you need to run it with InnoDB. Besides better performance without tweaking and ACID compliance, PostgreSQL also offers more query features. Since our predicted database requirements for this system are not very advanced we decided to go for the most simple and elegant solution and so we came to PostgreSQL. A more extensive research into why we have chosen a relational database management system and in particular PostgreSQL can be found in A.

4

Design

This chapter will discuss different design choices. It will highlight the major components that make up the system and how they are constructed. Furthermore, for additional information we refer to Appendix A.5.7 where a detailed scheme for our database is shown.

4.1 User

A user is a big design requirement for this project. We want to give people the option to register as a user and join/create groups and do other actions that should be persisted to their user account.

MemberInformation is the interface that represents the data required to be a user on the web application. In figure 4.1 an overview of all the attributes in the interface is shown. When looking at the interface the first odd thing that stands out is that there is no entry for password. When designing the User component we discussed everything that was actually required to make a user and what information we might not want to store. We decided that a user only needed to have an email and a username to be able to join our website, and that they could upload an image if they wanted to. We left out privacy vulnerable information such as real life name, address information and a password. Storing sensitive information such as passwords brings mayor security risks and we wanted to avoid this. In order to do so, we decided to go with OAuth authentication. An additional advantage of using OAuth is that a user does not need to register and create an account to become a member. This contributes to the overall user experience in which the user does not have to go through a tiresome process of creating an account. Lastly, the MemberInformation interface has an email field and a contact_email field because we want the user to give the option to change their e-mail without having to change their log-in e-mail.

```
1 export interface MemberInformation {
2     email: string;
3     image: string;
4     contact_email: string;
5     username: string;
6 }
```

Listing 4.1: MemberInformation

4.2 Exercise

The exercise is the most essential part of our web application. Everything is built around the data that is held in the exercise object. A user will visit the website and create a training out of multiple different exercises. A different user will spend a lot of time in creating a quality exercise.

For this it's essential to look at the individual aspects that make up an exercise. In the database an exercise is defined as a table that has an id, a name, a description, duration and lastly an image. These attributes were chosen based on the requirements that the client gave to us. The ownership of the exercise is stored in a separate table that links the user id between the exercise id.

```
export interface ExerciseInformation  {
    id: number;
    image: string;
    duration: number;
    name: string;
    description: string;
    avg_rating: string;
    tags: string[];
}
export interface ExtendedExerciseInformation extends ExerciseInformation{
    ratings: Rating[];
    materials: string[];
    groups: string[];
}

export interface Rating {
    value: number;
    user: string;
}
```

Everything related to exercises

This object plays a big role on the web application and thus it should be fleshed out properly with all of its requirements. We wanted an exercise to have an image that the users can use to display what they mean with their description. An exercise should have a duration that defines in minutes how long the exercise should take to complete. A name should also be contained within the exercise object along with the tags/categories that users give to it. We chose these attributes based on the research phase and discussion with our client, because users want to have a refined amount of features to be able to add to their exercises.

We do not always require just that information on the web application however. During the exercise creation process a user should also be able to define the materials and the group the exercise should initially be placed in. This information is also required on the view exercise page, however it is not required for the other places on the web application where exercise information should be displayed. This is why we chose to have an ExtendedExerciseInformation interface where along with the standard information, the client will receive all the ratings, materials and the groups that this exercise is in. The rating consists of a value and the user who has given that rating.

4.3 Training

Another major component of the website are the trainings. They represent multiple exercises bundled together with a name so that other users can view them and follow them. People will go on the website and want to gather a few exercises to create their own training. We offer them them the feature to export the training to PDF format and download it easily to be printed out or viewed offline. We decided to give this feature because we know users will go on the site and want to go from creating/viewing a training to being able to print out an overview as fast as possible. We designed a training card that shows all the necessary data available as easily as possible.

To represent the data we have the following interface:

```
export interface TrainingInformation {
    id: number;
    image: string;
    name: string;
```

```
    description: string;
}
```

Training information

This interface along with some computation methods is used to create the training cards. Exercises are linked to trainings through the table exercise_training. When requesting training information for the PDF we just fetch all the entries in this table with the training id required. A training will have its own description and title as well as an optional image to represent the entire training. These attributes have been chosen after researching what users would want and discussing with the client what he thought users would want as their training data.

4.4 Groups

Last but not least are Groups. Groups are an important part of Gymplanner. Groups are a component that tries to make Gymplanner unique to other current solutions. They are the feature that makes Gymplanner a forum for sharing exercises. There is a distinction between private groups and public groups. It must be noted that joining and leaving groups is a feature that is only available for registered users. There is no criteria for joining public groups. We decided to give groups the ability to be private as we understand from our client's wishes that some groups will not want to share their exercises with everyone.

Once joined you can find every exercise and every training that belongs to this group. The idea behind the public groups is that all content in public groups is of high quality. For this reason a person can't create a training and automatically add this to a public group. The system administrator will get a request for the created exercise and can then choose whether or not to add the content to a group. This system makes sure that all content publicly available is of high quality.

For private groups there is a different standard. In order to join a private group, a request to the group administrator is sent. This request asks the administrator whether that particular person may join the group or not. Once accepted the user can then see all group related content. For creating an exercise in a private group no additional constraints are given. This creates the idea that all sorts of content can be shared in private groups and that it is not quality checked. We came to the decision to make a private group for every user in which their created content is posted. One of the main reasons behind this idea, is that all content created by one user can always be found in his or her personal group; content created is therefore easily accessible.

```
export interface ExRequest {
    id: number;
    exercise_id: number;
    group_id: string;
    status: RequestStatus;
    time_of_request: string;
    time_of_review: string;
    reviewer_id: number;
}


export interface GroupInformation {
    name: string;
    description: string;
    image: string;
    isprivate: boolean;
    administrator: string;
    ismember: boolean;
    grouprequest: GroupRequest | undefined;
}

interface GroupRequest {
    id: number;
    user_id: number;
    group_id: string;
}
```

As is visible in the interface structure above: A group consists of fields such as name, description, image, isprivate and administrator. When requesting group information from the back-end we will also receive whether or not the requesting user is a member and if there is an open group request. These attributes were chosen because the client wanted the groups to have a personal image and we obviously wanted the groups to be identified by a name.

The Group Request interface simply has a unique identifier with a user- and group id. This way we can give this data to the group administrator who can accept or deny the request.

Exercises also have a request structure that is similar, but a little more in depth. The ExRequest also has a status, a request time, a review time and the id of the reviewer. This is done to allow the system administrator to have more control over the exercises that get posted.

5 Process

This chapter describes the process of the project. It explains the development techniques that are used and also describes the different kind of tools. Lastly this chapter shows the feedback which was given by the Software Improvement Group, which can be found in section 5.4

5.1 Scrum

Scrum is a process that increases the probability of successfully developing software [5]. It is a process that has been used in every development course we've had and it seemed only logical to apply the method in this project as well. Scrum is a method that uses small development cycles that are called sprints to make sure every week new software is implemented. At the end of every week a plan is devised that will describe all plans for the upcoming week. Responsibilities are assigned and at the end of the next week a meeting is planned to see what has been finished and what has to be pushed to the next week.

Reflection

Scrum is a great way for managing projects in an efficient way and we experienced it exactly like this. Task assignment made it very clear who was responsible for specific requirements. Furthermore, everyone is aware of what has to be finished by the end of the week. Although our time estimation on specific tasks was not always accurate and some tasks had to be pushed to the next week, the method overall resulted in good progress.

5.2 Development Driven Development

Since our project had to be implemented for scratch we chose to take a development driven approach and focus on the testing of the system in later stages. Even though the entire system had been roughly designed during the research phase of the project, you still encounter problems or design flaws during development. When this happens, there's a good chance classes need to be redesigned, refactored or removed all together. We felt that for this reason a test driven development approach would not be optimal as this could make some tests redundant. This doesn't mean however that the no tests were written, just in a later stage of the project.

Reflection

The approach of focusing mostly on requirements worked out great. Since we had tight deadlines such as the upload for the Software Improvement Group and the first working demo for our client, a lot of time had to be spent in creating new functions. If we would have spent more time in creating tests that for some reason may have became redundant, there is a good chance we would have lost a lot of time that could have been spent more efficiently.

5.3 Development Resources

5.3.1 Gitlab

Version control is a crucial part for any software project that multiple people work on. Because we have been encouraged all throughout the bachelor to make use of Github we all agreed that Gitlab, which is a version of

Github, would be used. Gitlab has many advantages for working on projects with multiple people. It shows a clear history of all the changes that have been made to the code and it has a branching system. That means that whenever someone starts working on a new feature, a branch from the main code is made and changes are saved to this branch. When the feature is finished, the branch is then again merged with the main code. This branching systems eventually leads in less code breakage. However, in the case that the main code is not operational anymore, you can revert it back to an earlier point in time and see what breaks the code.
Another great feature Gitlab offers is a virtual board on which every feature of the system is visible. These features are created at the end of each week when features for the next week are discussed. During the week these features on the board are updated to show the status of every feature. This is extremely useful for working in groups, since you can see at any moment during the week what issue still needs to be completed and which ones have already been finished. An example of this virtual board can be seen at the end of this chapter.

Reflection
Version control has been a great help in managing this project. All of us were already expierenced with the use of Gitlab as it encouraged in all software courses. It allowed everyone to work on different features without breaking the master code. However, merging everything at the end of the week or before mayor deadlines often resulted in code-breakage. Fortunately, most of the times we expected this to happen and time was reserved to solve the code-breakage. In future projects we should consider merging different features earlier instead of every feature at the end of the week.

5.3.2 WebStorm
WebStorm is a powerful IDE for modern JavaScript development. WebStorm provides full support for JavaScript, TypeScript, HTML, CSS as well as for frameworks such as React, Angular, and Vue. It has intelligent code completion, on-the-fly error detection, powerful navigation and refactoring for JavaScript, TypeScript and all the most popular frameworks.[2]. Futhermore, WebStorm helps you develop server-side applications with Node.Js. Since during our research we had already established that we were going to use React and Node the choice for WebStorm was very straight forward. WebStorm in combination with ESLint would make sure that every code would be formatted the same way, enhancing readability and maintainability for other contributors. It also made sure that no unnecessary merge conflicts would occur because indents would not line up, and Gitlab would mark this as different pieces of code.

Reflection
WebStorm has been a great help throughout this project. If we were to work with JavaScript in future project we would definitely use WebStorm again as it offered great futures that helped us during development. The suggestions it provided were of great help since this helped programming in a, to us, new language.

5.3.3 PostgreSQL
As discussed in section 3.7.4 we chose to go for a relational database scheme because this matched our system design and fulfilled our needs in terms of scalability. Out of all the database options that use the SQL language we chose to go for PostgreSQL. PostgreSQL is a powerful, open source object-relational database system that uses and extends the SQL language combined with many features that safely store and scale the most complicated data workloads. [1]. Furthermore, it's ACID-compliant without additional plugins and seeing as it's free it was the perfect fit for this project.

Reflection
PostgreSQL performs well as our DBMS. Thanks to the syntax being almost native SQL it was easy to find documentation on all the functionalities we required, be it for PostgreSQL, MySQL or any other SQL based DBMS. Furthermore, PostgreSQL provides an user-friendly GUI named 'pgAdmin', which made for easy administration and debugging of the database.

5.4 Software Improvement Group
The upload from the Software Improvement Group marked a big deadline in this project. The first upload was set at the end of week 6, and the second upload at the end of week 9. The analysis of the code by the Software Improvement Group (SIG) was a requirement for the project. The purpose of the evaluation by SIG

is to provide an instrument for developers for guiding improvement of the products they create and enhance [6]. The evaluation criteria is limited to the internal quality of maintainability and its sub-characteristics of analyzability, modifiability, testability, modularity and reusability. [6]. These characteristics are assessed on software product properties such as volume, duplication, unit complexity, unit size, unit interfacing, module coupling, component balance and component independence.

5.4.1 Feedback

The first feedback from SIG was received on the 11th of June 2019, and the entire review (in dutch) can be found in Appendix D. for our first submission we received 4.6 stars on a 5 star scale. The score was exceptionally high and we did not expect to score this high on maintainability. The feedback states that we did not receive the highest score because we had one method which was on average too long and contained too much functionality. They also said that they couldn't find any test cases and strongly recommended we should create unit tests. At this point we didn't have any tests yet, since we started writing them at the beginning of week 7. However, all of the feedback was processed and implemented the week after. The method was broken into smaller pieces and the first tests were written at the beginning of week 7. The feedback from the second upload can not be stated in the report as the feedback will not be available before the deadline for this report. We feel however that we should score equally high or higher on the second upload as all code submitted after the first upload was written in the same style.

board.PNG

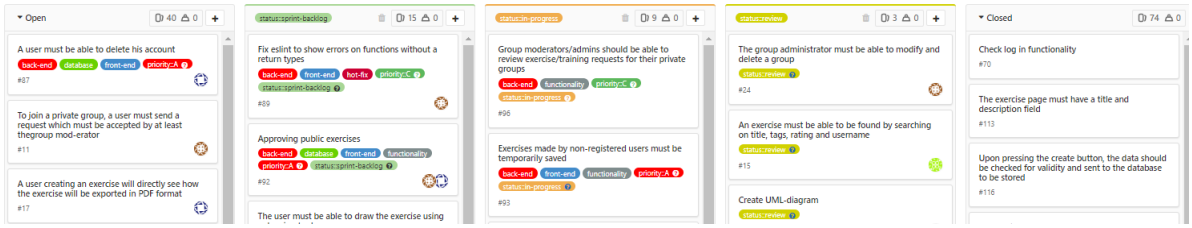


Figure 5.1: Virtual Gitlab board

6 Implementation

The design in chapter 4 is a good description for what the implementation of the application is going to look like. Most of the features work around the data models that are being described in that chapter. This chapter will discuss the specifics about implementing the design both on the front-end and the back-end side of the project.

6.1 Back-end Implementation

The back-end has as a job to listen to the clients for data they are requesting, connect to the database with queries to fetch the required data and package it all together in an understandable format to send back to the client. Besides this the back-end also tracks the user session and handles log-in requests.

6.1.1 Query structure

With this in mind we first set-up the database with our initial structure and connected it to the back-end. We performed some test queries to see if all the things we wanted to work were working.

To keep the back-end organized and structured we decided to create different routes for different purposes. For creating routes in Node.Js we used Express. For example, if we want to request training information we put that under the route: *db/training*. We envisioned this structure throughout the entirety of the back-end to maintain the concept of maintainability.

We then made functions inside a queries.ts file for the relevant route that would execute certain queries required to get certain data. An example of such a query is:

```
export async function getPopular(): Promise<Tag[]> {
    return await query(‘
        SELECT t.name, COUNT(*) as count
        FROM tags as t
        INNER JOIN exercise_tag as e
        ON e.tag_name = t.name
        GROUP BY name
        ORDER BY count DESC
        LIMIT 10
    ‘)
}
```

This query is located in the file: *db/tags/queries.ts*. These functions are exported so that a different file called *db/tags/routes.ts* could import the functions and specify the route at which this function should be accessed. The route will be defined in the following matter:

```
router.get('/popular', simpleGetResponse(getPopular));
```

When a connection requests the route popular the function getPopular will be returned. We wrote a function that takes an exported query function and collects all the requirement arguments from the request body and sends back the result of the query as a response to the connected client.

6.1.2 Exercise

For exercises we need the following functions:

```
getFromId(id);
searchInGroup(groupName, phrase);
search(requestingUser, phrase);
getTopFromPublic(amount);
getAllPublicExercises();
remove(id);
getFromTag(tagID);
getFromGroup(groupID);
getByUser(email);
create(image, hours ,minutes, name, description, name, tags, materials)
    ;
```

As mentioned during the design process the Exercise is the most important object throughout the website and because it is displayed on different pages with varying amounts of data we need multiple different methods to request exercises in different ways.

These are implemented on the back-end and used for different purposes all over the web application. For the create exercise page we for example need to have the create method, for the exercise display card we need to use getFromId and for the group page we need to be able to find all the exercises in a certain group.

6.1.3 Training

For trainings we need the following functions:

```
byGroup(groupID);
create(image, name, description, group, exercises[]);
addExercise(exercise, training);
getAllByEmail(requestingUser);
getPDF(trainingID);
```

The different parts of the web application where we need training data are the group page where all the trainings are displayed, the home page and the pdf export function.

6.1.4 Group

For groups we need the following functions:

```
isGroupPrivate(groupName);
checkGroupAdmin(requestingUser, groupName);
checkGroupMember(requestingUser, groupName);
getAllGroups();
getAllGroupsByEmail(requestingUser);
setPrivate(groupName, value);
getByName(groupName, requestingUser);
addExercise(id, groupName);
getGroupByPhrase(phrase);
deleteExercise(id, groupName, requestingUser);
deleteGroup(groupName, requestingUser);
create(name, image, description, requestingUser, isprivate);
removeUser(requestingUser, groupName);
addUser(requestingUser, groupName);
```

Groups require quite a few queries to be functional but most of the functionality of the website works around groups. Exercises need to be hosted in groups and so do training courses.

6.1.5 User session

We use PassportJS to handle the user serializing and deserializing along with OAuth2.0 to connect to Google Log-in services. We have routes for logging in and logging out.

The route for logging in route will first check the request object to see if the client is already authenticated and send an error message that the user is already logged in if so. If not, the route will return the URL that needs to be accessed to log in through Google. Upon going to that route Passport will handle the Google

authentication. Upon log in completion Passport will execute a callback in which we check the database for an already existing user, if this user does not exist we insert them into the database so we have an entry for the user that we can use to hang to the exercises they create and such.

To log out the client just requests the log out route and the user session is forgotten using Passport.

6.2 Front-end implementation

For the front-end we are mostly using React, JSX and SCSS. The front-end needs to communicate with the back-end to receive- and upload certain data. This is done by proxying requests starting with the /db/ route to the correct port on which the back-end is running.

6.2.1 Landing page

The landing page consists of two parts: The search bar with advanced search button and the exercise cards. The intention is to give users as easy as possible access to content on the website. When a user visits the landing page we want to give them the option to see the currently most popular exercises and to search for exercises or training courses.

The search bar uses full-text search to find matching properties on both the training and the exercises table. The full text search is performed on the back-end when a search query is received and it returns a set of options that match the full text input well enough.

The exercise cards simply make a database request for the currently most popular exercises based on the average rating and amount of ratings. These are then displayed on the landing page to allow users to quickly see what the most popular exercises currently are.

6.2.2 Home page

The home page was designed to be exactly what it's called; a home where users can find most of the information they require. We designed the home page to be made in such a way that upon the average visit, the user only needs to go to the homepage to see what's new.

For this we have two sections: The "My Trainings" section and the "My Groups" section.

The "My Trainings" section will contain all the training cards of groups the user is part of. The training cards will show all valuable information and have a download button. The download button makes a call to the back-end to download the PDF. The back-end then performs a query on the database containing all required data to create a training pdf (name, title, exercises, image). After this data is returned a PDF will be built out of this data and a download link will be sent back to the client.

6.2.3 Group page

The group page is a page where a collection of different data is showed. It consists of different sections that are navigable using the menu on the right side of the page. We wanted the group page URL to clearly show which group the user is on in case they wanted to navigate by directly going to the URL. This is why we decided to make the URL 'group/:groupName'.

We used react router to separate the different pages considering users might want to bookmark a certain section of the group page, for which it needs its own URL.

The components are as follows: Trainings, Members, Exercises and Materials. Each of these elements has an individual component. The trainings page is a page where all the group's trainings are displayed using the TrainingCard component, the members page shows all of the members using PersonCard components and the exercises page uses the DrillCard component to display all the group's exercises. Last but not least the materials are simply displayed as a list in plaintext.

When a user goes to the group page and they are not member of the group yet the content is not visible. A button is displayed in this situation where, depending on whether the group is public or private, the user can request access. This access is immediately granted when the group is public since any member should be able to join such a group. The implementation however is the same. The user will press the button and a request will be made to the back-end for the user to join a group. If the group is public this request will automatically be accepted. On the other hand if it is private the request will be put in the request database, which can be accepted by group administrators.

The group navigation menu also contains a section for the group administrator. The front-end will request information from the back-end that also has a field that will tell the front-end whether or not the user is the

group administrator. Based on this field a component is shown that allows the administrator to perform administrative tasks such as accept group join requests or remove certain members.

6.2.4 Creating an exercise

To create an exercise we require multiple different types for input from the user. We want the user to give a title and description, an exercise could have different materials and tags, it needs to be able to have an image and it needs to have a time and be assigned to a group.

We wanted to give the users a good overview and non-cluttered page where they could enter this data and know when they are done.We implemented the exercise creation page using multiple different creation stages that could all be made individually. These stages are implemented in a generic way such that validation of whether or not a component is finished can be done without hard-coding.

This way we constructed a list of finished stages on the left side of the page and a list of unfinished stages on the right side of the page. These lists are automatically updated whenever a stage is completed/uncompleted.For the input fields we created a component that has a limit on the amount of characters that can be put in. We do this to prevent the user from sending a title/description that goes over the varchar limit of the database.

The tags/materials stage has a generic input component that, upon pressing space, automatically converts text into tag components. This component is then used for both the materials and tags as they both are displayed using the same tag styling.

For the image stage we wanted the user to have live feedback on the image they are inputting. That is why the input box automatically shows the image if it can be loaded and gives an error if the image specified is not actually an image file. Upon finishing this process the exercise is posted into the database and then the user is redirected to the view exercise page.

6.2.5 Viewing an exercise

The page to view an exercise on needed to be non cluttered and easy to navigate. We wanted to show the key aspects required for a user to judge whether or not they liked the exercise. Therefore we made the title/description/image/duration largely sized on the center component and made it easy to see the groups this exercise was posted in.

Besides that we wanted to make it easy for the user to bookmark an exercise and navigate to them. We decided that a clear URL was important, so we made it in the following format: '/exercise/view/:id:'. This way a user can easily see where they are based on the URL.

The exercise component looks at the current path and uses the id to fetch the required data from the back-end and displays the data in the component.

6.2.6 Search implementation

We wanted to make the search implementation as extensive and advanced as we could. For this reason we looked for options when performing full-text search on the back-end. There were multiple options available for making this possible.

The first option was elastic search, a full-text search engine that gives the developer the option to store, load and search for data. The problem with elastic search was that it's a standalone database that required either a complete redesign of our database implementation or an inefficient bridge setup. We decided that this was not the right option for us.

Then we came across a fuzzy search algorithm. This is an algorithm that does not look for exact matches in text but instead looks for approximate matches. An example is when there is an exercise requiring the materials "Dumbell" and the user searches for "Dumbells" the algorithm will still find a match. When looking for the best way to implement this algorithm we found out that PostgreSQL has a built-in support for fuzzy queries.

When looking at the built-in support we noticed that PostgreSQL has functions to transform words into a list of distinct lexemes. We used this feature to build a full-text search support algorithm. To do this we construct a power-set of the search query that the user has entered. If the user enters "Shoulder Back workout" the powerset will be: {{Shoulder}, {Back}, {workout}, {Shoulder workout} {Back workout} {Shoulder Back Workout}}. This powerset is then passed to the query function to transform them into the search query notation to transform the query into a set of lexemes. This constructed query set of lexemes is then compared to the document to see how well they match. This match is given a score and based upon this score we decided whether or not to show the result.

6.2.7 Text-based Tag recommendations

Upon reaching the initial product, we decided to get into adding Quality of Life functionalities. After some thinking, we stumbled upon the idea to extract tags from the user-created content of an exercise.

First off we defined the content of an exercise as its title and its description. Then we decided that our recommendation system for tags would be based on text analysis. After some research, we decided to go for the TF-IDF[4] algorithm. TF stands for Term Frequency and IDF stands for Inverse Document Frequency.

The term frequency, or TF, which is the amount of times the term is used in the selected item, is multiplied by the inverse document frequency, which is the amount of items in which the term is used. To increase accuracy, we've created a small database of test data which consists of 50 exercises with title and detailed description, in Dutch. The TF is calculated as follows:

$$\mathrm{tf}(t,d) = \frac{n_{t,d}}{\sum_k n_{k,d}}$$

The inverse document frequency, or IDF, is a measure of rarity for the terms across documents. The corpus, collection of documents, is analyzed and if the a certain term appears in every single document, it is considered common. While a term that is frequent in one document and not in others, is a rare, important term. The IDF is calculated as follows:

$$\mathrm{idf}(t) = \log \frac{1+n}{1+\mathrm{df}(t)} + 1$$

Finally, the TF-IDF is the result of the multiplication of these two values.

$$\text{tf-idf(t,d)} = \text{tf(t,d)} \times \text{idf(t)}$$

So for each exercise that is being added, as soon as the user finished typing the title and description, our algorithm starts preprocessing the text by stemming words, which is the normalization of words that are the same word in different conjugations, and removing Dutch stop words. After this stage, the algorithm starts extracting the most important terms by calculating both the TF and IDF for the terms in the new exercise and sorting in order of TF-IDF rank. Finally, the top 5 terms are filled in automatically in the 'tags' field on the exercise creation screen.

The algorithm still needs some optimization, it takes a few seconds before results are returned, but this is mostly masked by the fact that the script is trigger upon finishing the title and description. Which means that the tags will be loaded in the background, regardless of the step the user is performing on the exercise creation screen.

7 Evaluation

After 10 weeks the project has to be evaluated to determine whether it has been completed successfully. In order to determine the success of the project, in Chapter 2 success criteria were established. This chapter will discuss whether these criteria have been met or not. This chapter will also show the test-results for the project to support the functionality of the system.

7.1 Requirements

In this section we will go over all the requirements we have designed for the system during the research phase and the features that were actually added to the system. There may be some features that were added during development that did not come up during the research phase of the project. First off we look at the must-have requirements, these can be separated into several categories. That is, users, groups, exercises and moderators. When looking at the system we can easily see that all of the must-have requirements are implemented with the exception of one: A user must be able to request a new password. During the design phase we did not extensively look in to ways of logging in yet. As we are using Google OAuth, we do not actually save passwords anywhere. Requesting a new password means a previous password has to be stored and verified somewhere, however this brings security issues which we wanted to avoid. Hence the requirement was not a must-have requirement after all. When looking at the should-have requirements we can see that most of the requirements are implemented as well. Requirements that were not needed anymore were requirements such as, using voice-chat to create an exercise or users recommending exercises to other users. Voice-chat was not needed as we did not implement an image-tool, making voice-chat completely redundant. Users recommending exercises to other users was simply not a feature that warranted the effort that would be required.

7.2 Testing

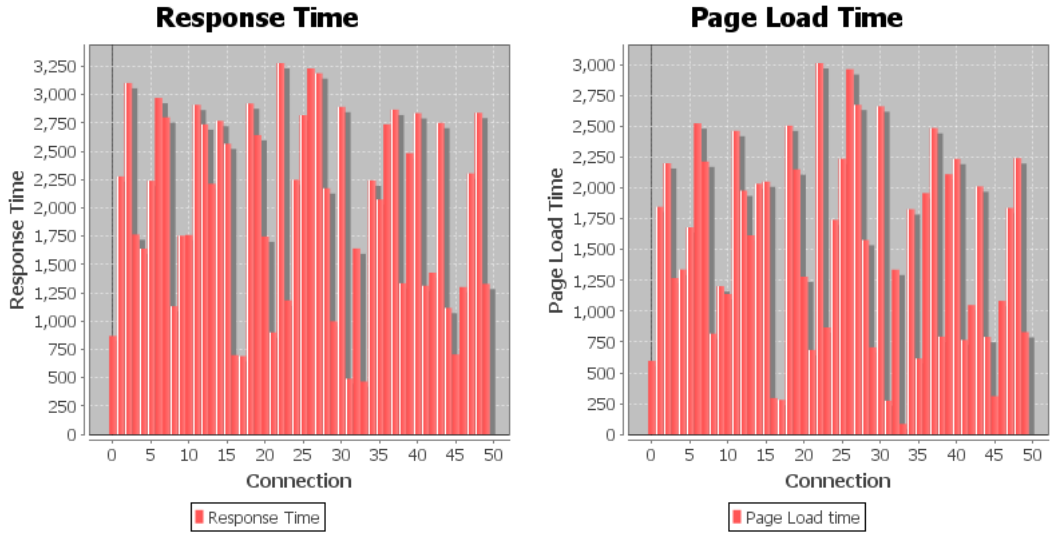
Our back-end can be summarized as a set of queries, routes and an authentication strategy implemented using OAuth2. To test these functionalities, we have written unit tests for every method/query that is available in the back-end triggered by Express' routes. Due to prioritization, we have been mainly testing good code smells and have ignored some alternative branches that catch errors. This can also been seen in Appendix C When looking over the table for the back-end we see very high coverage percentages for every route except src/routes/db. This route is only tested for 21.2% since it contains also the PDF generating methods which forms a greater part of the file being tested. We did not find it interesting to test these methods as the input to these methods are separately tested and the process of generating the PDF uses a well known npm library called pdfkit. All in all, including this route we have an average of 89% covered.

The unit tests are written and executed using the Jest library. Jest allowed us to run some tests in parallel and some in sequence. This paved the way for more complex tests that depended on multiple mutations, almost like integration tests. Also using the beforeAll and afterAll methods we were able to start with an empty database and end with an empty database, leaving no traces of the tests. Finally, Jest also allowed for testing of async methods using promises which fit very well on the structure of our queries which are also asynchronous methods that return promises.

For the front-end we decided to test the components that interact with data received from the back-end using sagas. All components making use of Redux are tested and this can also be seen in the Appendix C. In total we have a coverage for 99% for all Redux states. All components making use of Redux, exist to trigger events using action types; describing only what happened, whereas reducers describe how the application's state changes by fetching and setting the data as desired. By doing this, we made sure that the state of each component interacting with data correctly receives the data from the back-end and sets the state of each React component to the desired output. More information about code coverage of the front-end part can be found in Appendix C.

7.3 Load Testing

We used our own Selenium script written in Java to load test the system with 50 concurrent users. Our non-functional requirement stated that we needed to test 100 different users, but the script we created was not able to find enough available ports to perform this. We increased the amount of concurrent users from 10 to 50 in increments of 10. This was done to check if the system would perform worse, the bigger it scaled, which was not the case. Below are the graphs generated from our script:



7.4 Success Criteria

To measure the success of the project the success criteria have been defined in Chapter 2. These success criteria were:

- All the must have requirements are implemented and functional
- 75% of the should have requirements are implemented and functional
- The system is online and functional
- 90% of the non-functional requirements are valid.

7.4.1 Functional requirements

As stated in section 7.1 all of the must-have requirements have been implemented. Regarding the should-have requirements we can see that 3 out of the 16 should-have requirements have not been fulfilled. This results in 81.5% of the other should-have requirements. In addition we added technical features such as full-text search and text-analysis based tag recommendation. Hence we can conclude that we fulfill the success criteria and the project can be seen as successful.

7.4.2 Non-functional requirements

In this section we will highlight some of the non-functional requirements such as availability and efficiency by means of according figures. We will not discuss every non-functional requirement as some are clear enough

to state they have been successful. When looking at the non functional requirements stated in 7.4.2 we can see that all requirements from the section Development Standards and Access Security have been fulfilled. Most of the points have been discussed throughout the report and hence can be concluded as successful. In the section Availability we state that the system should be able to handle 100 concurrent users. We were not able to test a 100 simultaneous users as stated above, but we tested for 50 users and this was no problem. Furthermore, we state that search results should be accessible within 5 seconds of searching. 5 seconds can be regarded as very large, and we can easily fulfil this. The results for search time queries are depicted in Figure 7.1 and Figure 7.2

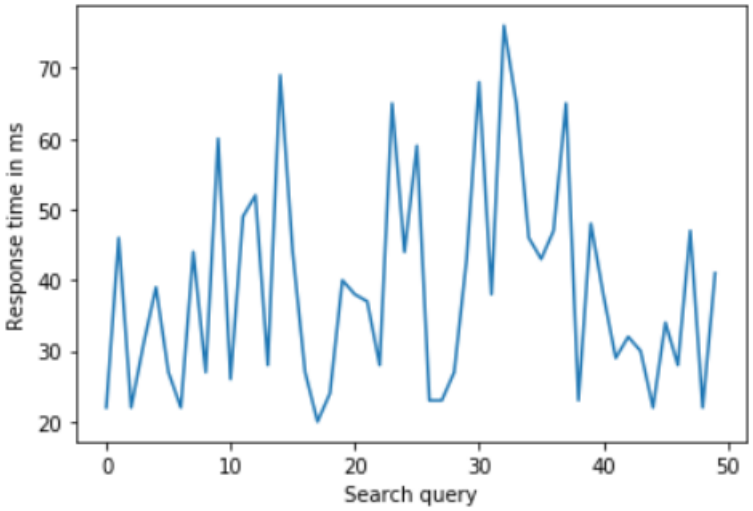


Figure 7.1: Response times for exercise search

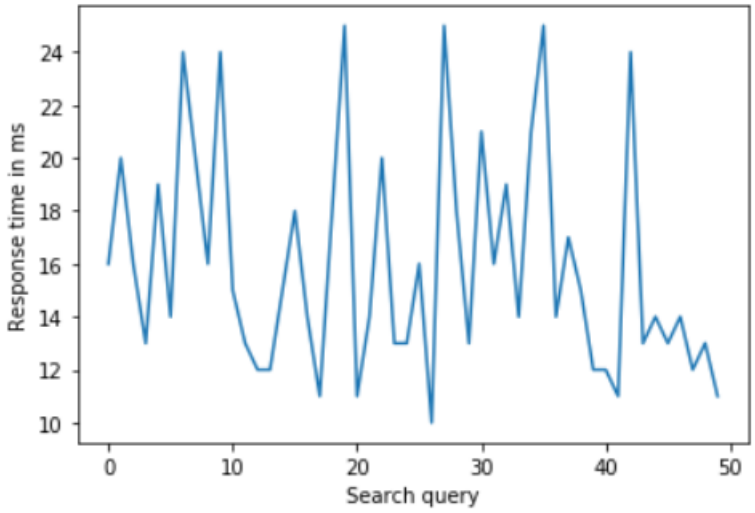


Figure 7.2: Response times for group search

Lastly, there is one section of the non-functional requirements that we did not complete. The section of Confidentiality. We stated that documents should not be stored indefinitely, a user should be able to see all data the application has stored and that documents should be handled according to GDPR standards. Not enough research has been concluded to state we handle our documents according to GDPR standards so we can't conclude this non-functional requirement. Furthermore, the system does not automatically remove content after it hasn't been seen or opened for some amount of time. Lastly, we did not create an automatic option where a user could see all the data the website has saved from that user. The user is however able

to contact a system-administrator who in turn is able to see what data on the user has been collected. The other 2 non-functional requirements we did complete. In numbers this would mean we fulfill 24 out of the 27 non-functional requirements. Fulfilling 88.8 % of all the requirements. We feel this is close enough to 90 % to say it is succesful.

7.5 User evaluation

For a complete evaluation we asked our client to show the system to people who would eventually become users of the platform. This evaluation is particular important as it shows what other people are thinking of the system. Whereas we may feel everything is intuitive en simple, this may not always be the case for people not familiar with the system. The client asked a series of a question and we will summarize and comment their reactions below.

7.5.1 Do you need extra information for creating an exercise or group

Overall the feedback on this question was very good. Everyone stated that once you were on the right page that everything was intuitive. However, one person noted that it is not self-explanatory that you need to navigate to the home page in order to start creating content. For this reason we adjusted the landing page a little bit such that it shows an extra button for navigating to the home page.

7.5.2 Are the information cards clear and intuitive?

Feedback here was that they liked our designs for the information card and the information depicted was sufficient and clear. However, they noted that the ratings were given in too many decimals and redirects were not fully working. This seemed to be bugs in our code and we adjusted this accordingly.

7.5.3 Is the process of joining a group clear?

The general feedback on this question was positive. Certain testers found it a little difficult to find a group because the group page showed so many groups. We feel this difficulty might be because it was not clear enough in the search bar that any feature can be used for searching purposes. To fix this issue we added a more clarifying placeholder that explains what features the user can search on and makes it easier to find the group they want.

7.5.4 Do you understand the difference between public and private groups?

Everyone stated that they understood what the difference was, however as both groups are visible on the group page we do not know which groups are public and which ones are private. For this we adjusted the design for the group cards such that private groups show a lock to the name. This indicates the privacy of a group.

7.5.5 What is your opinion on the navigation

The general consensus was that the navigation for the most part was clear but some users said the distinction between home page and landing page was not clear enough. Besides that a complaint was that the search button did not work, this demo was before the search bar was completed and therefore after this feedback the search bar is now functional.

7.5.6 Is any functionality missing

The feedback here was very useful as it shows the needs of the market. The feedback stated that certain buttons such as the search bar on the landing page was not working. That materials and tags were not saved when creating an exercise and searching in general was difficult or tiresome. We took all of these comments into advice and solved them accordingly. We feel most of the remarks have been solved.

7.5.7 What would you like to be able to search on

The users wanted to be able to search on all exercise features. They also said they found the search bar on the landing page enough functionality to be able to navigate through the website. The feedback was primarily positive with the exception that the search bar was currently not functional, however as said in a previous subsection this has now been fixed.

7.6 Process Evaluation

In order to make future projects run more smoothly, project evaluation is essential. This gives a clear overview of all the things that went well, and which things are worth reconsidering in future projects.

As stated in Chapter 5, this project made use of the Scrum methodology. The organization of the project by means of the Scrum methodology has been very helpful. It gives room for a lot of flexibility and ensures that in every week new software is developed. In addition, the Gitlab board tool discussed in Chapter 5 was a great way of keeping track of the issues. However in future projects we should work with this tool more organized. All of the issues were made at the beginning of the project, some were added during the sprint backlogs and this caused the board to become cluttered. Some issues were duplicated and some issues were never changed from status. This is definitely something that should be reconsidered for future projects.

Secondly, although a planning was made every week for the required features, we experienced that we were sometimes too optimistic in this planning. We planned for the end of week 4, the second week of development, that every must-have requirement should be finished and we could show the first demo to our client. In reality this was too optimistic and we experienced that not enough must-haves were finished for the demo. In the future we should think in more depth what the must-have requirements are, and what they all consist of. Some of the must-have requirements were made up out of several other components that were not yet implemented, and hence difficult to complete all at once. The idea for next projects is that must-have requirements are constructed in a more detailed manner.

Lastly, communication between different parts of the project is something which should be considered for next projects. Even though we had a very good distinction between who would work on what part of the project, we experienced that communication between two parts (i.e. back-end and front-end) was not always optimal. This resulted in a lot of functional code in the back-end and in the front-end with no real connection between them. We've experienced this ourselves during the project and took it upon ourselves to communicate better. Overall we feel as if we eventually fixed this communication problem.

8

Recommendations

Since the project is now finished and will be handed over to the client, this chapter will discuss future work and general recommendations. We'll divide the chapter in two sections, recommendation on the implementations and general recommendations for Gymplanner.

8.1 Implementation Recommendations

All of the code has been written in such a way that it should be easily scalable and maintainable. This gives room to add additional functionality. Unfortunately not all of our initial ideas for the website could be implemented, because of time constraints. This does however not mean that they shouldn't be added anymore. The biggest recommendation is, when the website starts to grow, to implement a payment system. This was a requirement which was beyond the scope of this project but is essential in order to make the website profitable. Obviously you need to give paying members advantages over those of non-paying members otherwise there is no incentive to become a paying member. Some of these advantages could be the maximum amount of groups you can join, or the maximum amount of training courses you can save and export to PDF format. These constraints would have to be implemented in the code, as there are currently no such constraints. Other recommendations would include expansion of the create exercises functionality. The system would benefit from an image editing tool, in which users can make illustrations for the description. Eventually pairing this with a collaboration framework from React would give users to option to work on exercises together. This could greatly help in providing high quality exercises

8.2 General Recommendations

In order to get Gymplanner really off the ground it might be a good idea to think about marketing. First start by test-casing the website to a close group of friends or a small group of the targeted audience. Make note of their recommendations for the website and consider changing them accordingly. After that promoting the website might be a good idea. An example is buying targeted advertisements on Facebook. For a last recommendation, it could be beneficial to hire a graphical designer. Although we are very satisfied with the design choices we made, they may not be compliant with the wishes of the targeted users. In designing every component we kept simplicity and clarity in mind and we feel this is strongly reflected in the end design. However, as the design has never been tested by actual users, we can't guarantee that it is what users of the system might want or expect. In the end, user experience if the most import aspect of a website.

9 Conclusion

The project has been done for the start-up company Gymplanner. The web application it had did not fulfill any of the actual wishes of the client. It also wasn't very attractive to sign up for the platform as most of the content seemed to be absent. Our job was to create a new system that would fulfill the basic functionality the client expected the website to have. The result we have delivered is a system that offers all basic functionality and has a much more attractive look for new users.

Furthermore, the system is well documented and well tested which ensures that most of the functionality does what it is supposed to be doing. Of course, this is not a guarantee that the system is without bugs, but it should definitely make it more maintainable and scalable. Most of the code has been written in small chunks of data to keep functionality as separated as possible, which in turn also helps preserve the maintainability. The company should experience no issues if it chooses to add extra functionality.

This project was not the most technical challenging project regarding our skill-set. Most of the system was designing the database, and developing functionality in a new language. One of the project members had already worked with the react language and thus had some experience in web-development. The other group members had little to no experience with the react language and this made some of the features challenging. In order to make sure the web application would have features the competition did not have we looked for technical challenging features that would be more compliant with our skill-set. This is when we started looking in to text-based recommendations and full text search functionality. These were technically challenging aspects of the system, as they required certain algorithms to work. With these features we are satisfied as they add a little extra technical prowess to the system.

Overall, we as a group are very satisfied with the final product. The code is clean, well-tested, scalable and open for adding new functionality. The system itself is launch ready and we are hoping to see it successfully launch. We feel that there is definitely an application that is able to compete with the other solutions and has features that seem to be missing in the competition

A

Framework Research

This section will contain all the research we've conducted during the research phase of the project. Most of the research has also been included in the research rapport.

A.1 Back-end Framework Research

A.1.1 PHP

PHP is a sever-side scripting language. Created in 1994 and primarily meant for web development. PHP is processed by an interpreter, and can be embedded into HTML markup. It's very flexible and this contributed a lot in the popularity of the language. It has great support for relational databases and database management systems such as MySQL and PostgreSQL. However, PHP has major security issues and does not score as well on maintainability and readability as the alternatives

- + Efficient working with relational database
- + Large community with a lot of support
- - Not designed for many concurrent users
- - Outdated for the current web model

A.1.2 Java

Java is an object-oriented programming language developed by Oracle. It uses static type-checking at compiling and at runtime which makes it a highly secure language. It is platform independent and this contributes a lot to portability. Just as it's competitors also provides a lot of open-source libraries

- + Platform independence
- + High performance for CPU-intesive applications
- - Less scalable

A.1.3 NodeJS

NodeJS is an open-source cross-platform environment that enables building the server-side of the application by providing a wide range of packages. These packages can be easily installed through the NodeJS package manager(NPM). NodeJS supports multi-threading and this makes it very scalable. By providing single syntax for client and server side it also contributes to re-usability of code. Furthermore, NodeJS can be integrated with TypeScript and type safety contributes to maintainability.

- + Better scalability because of multi-threading support
- + Simultaneous request handling by means of a non-blocking IO system.
- - Relatively young

A.1.4 Testing framework: MochaJS

The most obvious and used choice when it comes to JavaScript testing seems to be MochaJS, due to its flexibility and possibilities to allow developers to choose third party options as to which assertion, mocking and spy libraries they want to use. It is the most popular framework supporting both back-and and front-end testing.

- + NodeJS debugger support
- + Providing a clean base to develop tests
- + Object mocking

A.2 Front-end Framework Research

A.2.1 Vue.js

Vue.js is a JavaScript framework, launched in 2013, which perfectly fits for creating highly adaptable user interfaces and sophisticated Single-page applications. However, since it's market share is relatively small it does not have the resources angular or react have, it is not desirable for bigger applications that require more external resources.

- + Large scaling and very lightweight
- + Great for single page applications
- - Lack of resources

A.2.2 Angular.js

Angular is a JavaScript MVVM framework, which is great for building highly interactive web applications. It is mostly maintained by Google and thus can be expected to be around for a long time. Angular is a complete MVC framework which means it has by default a lot of functionality. This also makes it a more heavyweight framework with functionality you eventually may not need.

- + Exceptional support for Typescript
- + Detailed documented and a lot of support
- - Heavyweight framework with functionality you may not use
- - Entry level slightly higher

A.2.3 ReactJS

ReactJS is a JavaScript library, open sourced by Facebook and can be used for building any modern application regardless of size and scale. Since it's developed and maintained by Facebook this framework can be expected to be around for a long time. Furthermore, React makes use of a virtual DOM which makes it incredibly fast in performance.

- + High performace
- + Well documented with a lot of support
- + Easy to learn due to simple design
- - React moves away from class-based components, which may be barrier for Object Oriented Programmers

A.3 Integrated Development Research

Once we established what languages we were going to use, we had to decide on an Integrated Development Environment (IDE). This section contains the research we've done into different IDE options

A.3.1 Microsoft Visual Studio Code

VS code is not really an IDE, but more of an editor. It is considerably faster, more lightweight and highly customizable. However some quick research showed that the learning curve is considerably higher if you're programming in languages you are not yet familiar with and it strongly recommended that you install plugins such as Code Spell Checker, prettier goto-last-edit-location and TODO highlights. These are all already integrated in WebStorm

- + faster
- + lightweight
- - requires more plugins to make code maintainable

A.3.2 WebStorm

Stated by JetBrains as the smartest JavaScript IDE. WebStorm has a lot of predefined functionality which helps you keep your code maintainable. It is perfectly equipped for building complex client and server-side development with NodeJS. Furthermore it offers great support to some type annotation frameworks and testing frameworks.

- + type annotation framework support
- + test framework support
- - heavier than VS code

A.4 Type Annotation Framework research

When researching for Type Annotation Frameworks that would comply with our previous stated choices, two options came up.

A.4.1 Flow

Flow is a static type checker designed for JavaScript. It is specialized in finding errors in JavaScript applications. Just like Typescript it is compiled to plain JavaScript. However, since it only became available for Windows in 2016, the support community for it is considerably smaller

A.4.2 TypeScript

TypeScript is a super-set of JavaScript which provides static typing, classes and interfaces. It provides a rich environment and may help you find errors in large pieces of code. TypeScript has great build in support for WebStorm and React. Because the support for it is considerably larger, and the integration into WebStorm is already there, we decided to work with TypeScript.

A.5 Database Management System research

A.5.1 Relational Databases

Relational databases are databases based on the relational model of data. The data is stored in tables in which each row is an entry and each column represents information. Relations between tables are represented by Primary Keys and Foreign Keys. The SQL systems discussed below compose is by no means an exhaustive list of those available.

Advantages

Due to its consistency in model, relational databases allow for complex queries, transactions and routine analysis of data. It also supports the so called ACID properties:

- **A**tomic, a transaction is either performed completely or not at all.
- **C**onsistent, the database is always in a correct state, both before and after a transaction.
- **I**solated, concurrent transactions do not influence each other directly.

- **D**urable, a transaction cannot be undone in the future.

Disadvantages
Relational databases are not ideal for the storage of large images, numbers and other types of multimedia.

A.5.2 SQLite
SQLite is a decentralized file-based database that does not require a central server for the database usage. All reads and writes are performed locally on the file and the transactions adhere to the ACID property.

Advantages
SQLite requires very little disk-space, its size is easily under 600 KB. It is best used for low-traffic websites with little data or IoT devices due to its small size and RDBMS nature. It can also be used as an addition to other RDBMS to cache data client-side. This would increase the speed of queries significantly.

Disadvantages
As stated above, due to its local nature, SQLite lacks multi-user capabilities which are requires by most systems regarding the sharing and mutation of data. Another disadvantage is that the write operations are serialized, which creates a bottleneck during concurrent writes.

A.5.3 MySQL
Unlike SQLite, MySQL has serves client/server architecture that consists of an SQL server with multiple threads. MySQL stands for scalability, security and replication.

Advantages
Due to its multi-threaded nature, MySQL can easily be run across multiple CPU's, hence it's very scalable. Unlike SQLite, MySQL supports multi-user transactions. And it is well optimized for read-heavy instructions, better than PostgreSQL in throughput and performance.

Disadvantages
The performance of MySQL degrades quickly on concurrent write instructions and heavy select operations. Also on concurrent read-write operations it lacks performance.

A.5.4 PostgreSQL
PostgreSQL is an open source project heavily focused on SQL compliance and extensibility. The architecture of PostgreSQL is also of a client/server type. The server is called 'postgres'.

Advantages
Because PostgreSQL processes a lot in parallel, it outperforms MySQL in select operations and concurrent write instructions. Furthermore, it carries the same avantages as MySQL.

Disadvantages
PostgreSQL is power hungry, a lot more than MySQL. For every client connection it forks of for up to 10MB of memory.

A.5.5 Non-Relational Databases
Non-relational databases are databases that come in any other shape than that of the tabular relation described in A.5.1. Non-relational databases are also called NoSQL databases. The NoSQL systems discussed below compose by no means an exhaustive list of those available.

Advantages
NoSQL databases work on structured, semi-structured, and unstructured data. The architecture is also efficient as it scales with your database rather than being predefined from the start.

Disadvantages
There is not a lot of documentation on NoSQL systems compared to SQL systems, since they are not that mature and they mostly are open-source. Thus, it requires more skill and time to get into a certain type of

NoSQL.

A.5.6 MongoDB

MongoDB is an open source platform written in C++ and is easy to set up. It stands for high performance, high availability and auto scaling. It is also described best as a cross-platform, document-oriented NoSQL database.

Advantages

MongoDB is a schema-less database and is, therefore, very flexible. It also supports sharding, which is a mechanism that divides and replicates the database on different servers to ensure resilience. Besides, it also has very fast query responses due to the accessing of documents by indexing.

Disadvantages

MongoDB has no join operations, in contrary to relational database systems. It is possible to write such an operation in code, but it will not be as optimal as the SQL join operations. Due to the lack of joins, there is data redundancy which leads to increased memory usage without usage.

A.5.7 Cassandra

Cassandra is a key-value based DBMS. It is similar to MongoDB but it stores single rows instead of documents. And these rows don't follow the same pattern as they must in SQL databases.

Advantages

Cassandra is open source, which means it is free to use and this increased the community over time. There is a lot of documentation and information available. Besides, unlike the master-slave architectures of most SQL systems, Cassandra uses a peer-to-peer architecture. Meaning that there is no single point of error, every node can take over the job of a failing node.

Disadvantages

Transitions from any other DBMS to Cassandra does not work well, it is better to either use Cassandra from the start or not at all. Besides, it is not very good in handling many-to-many requests.

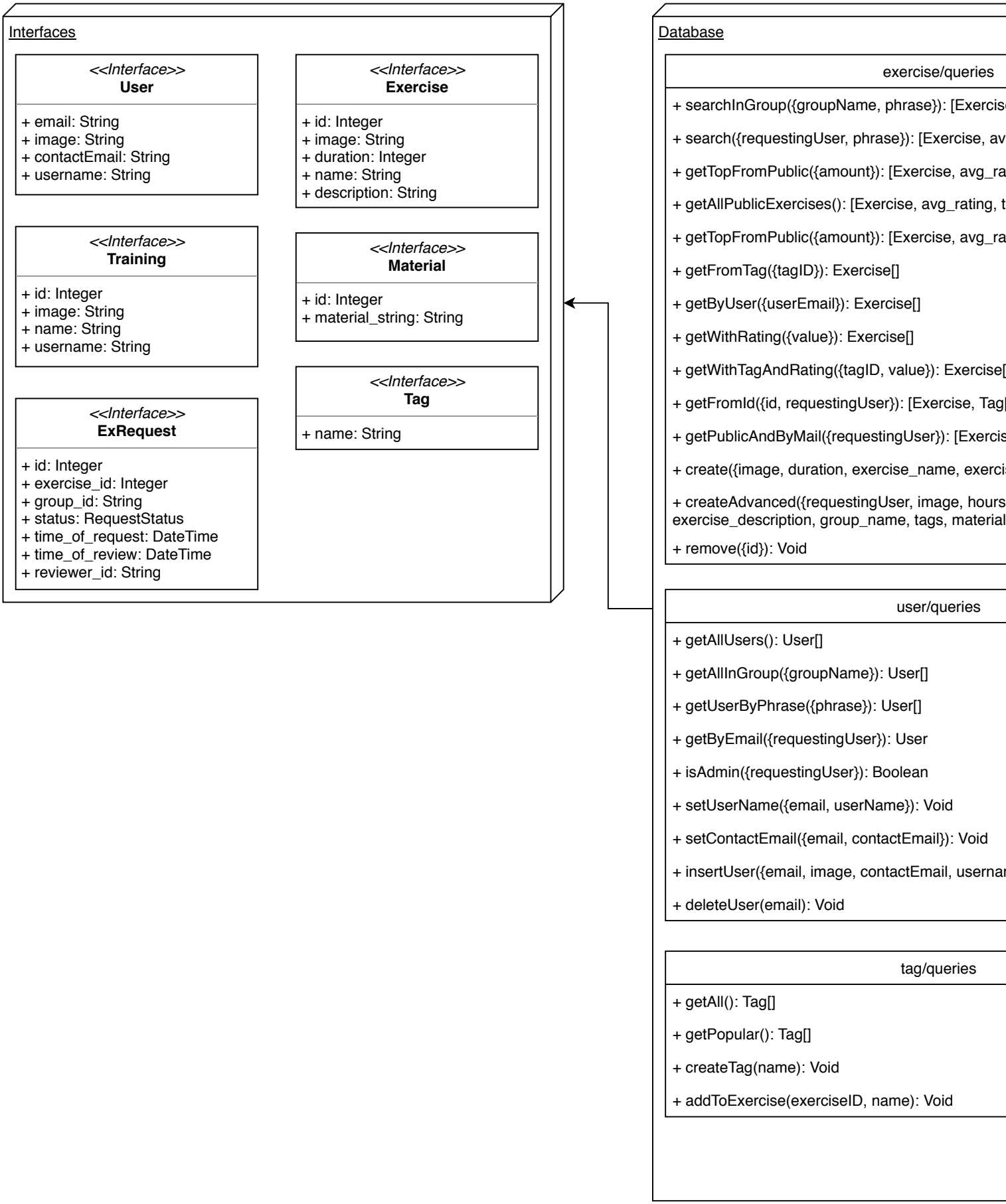
B

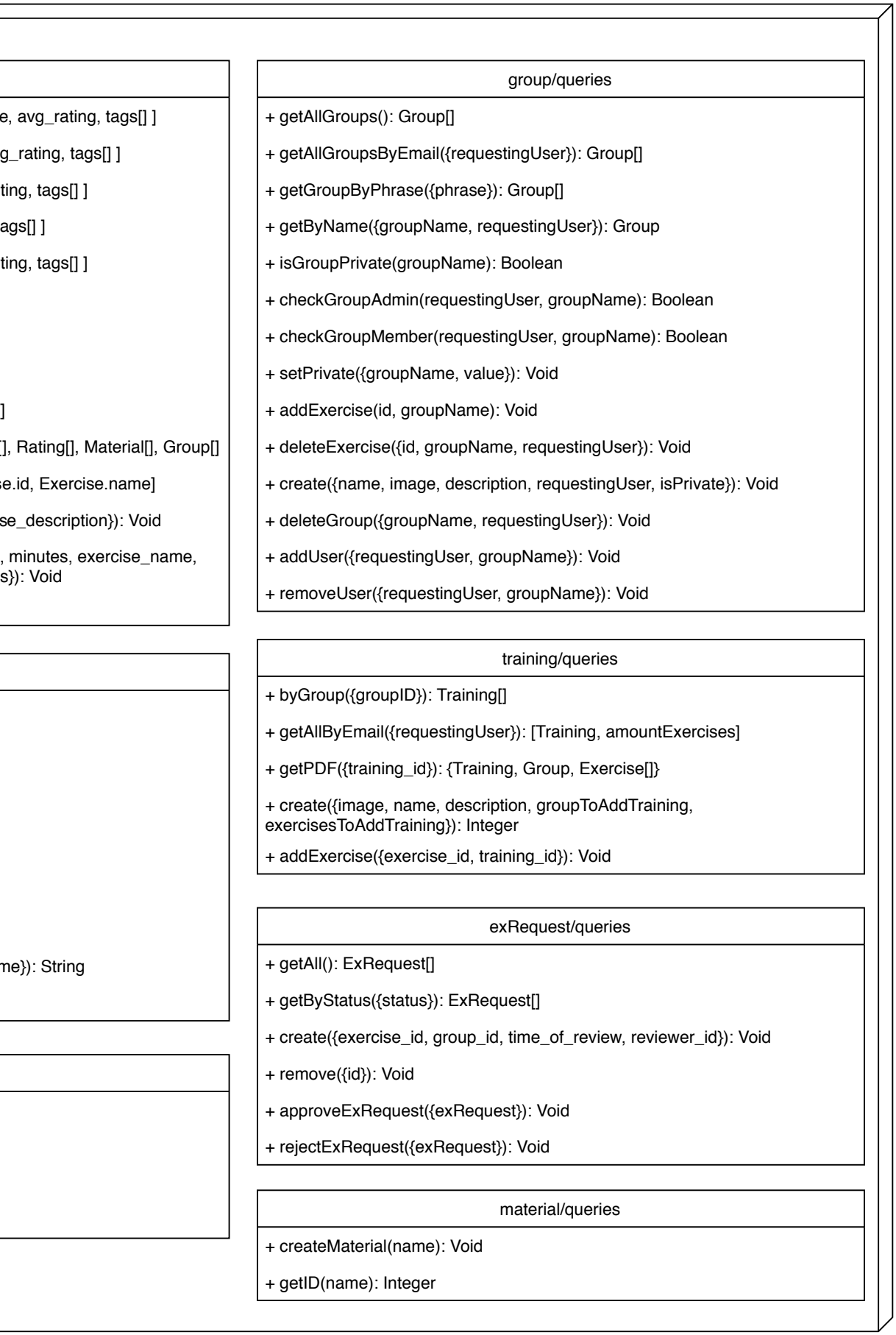
UML

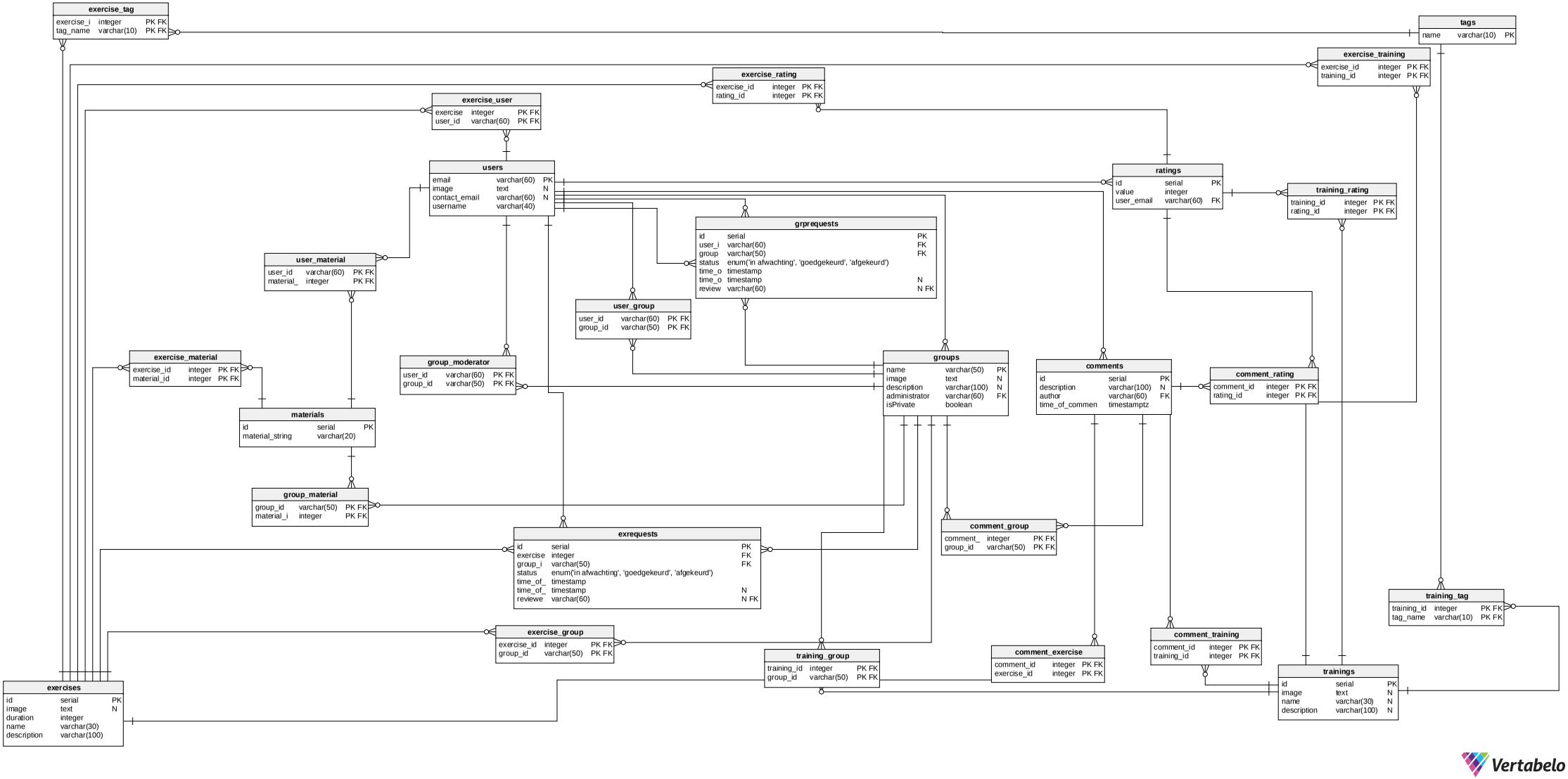
The first diagram is that of the back-end. It spans two pages and is best viewed if those pages are set side by side. It is a class diagram that shows the two main components of the back-end. A group of interfaces and their specifications for our custom objects and a group of classes that contain the methods for querying the database. The classes are divided according to the objects they perform queries on.

The second diagram is an ER diagram of the Postgres database. It is displayed in landscape for best visibility. The notation used for relations is the crow-foot notation. Default styling is used for the boxes, including the identifiers PK for primary keys and FK for foreign keys.

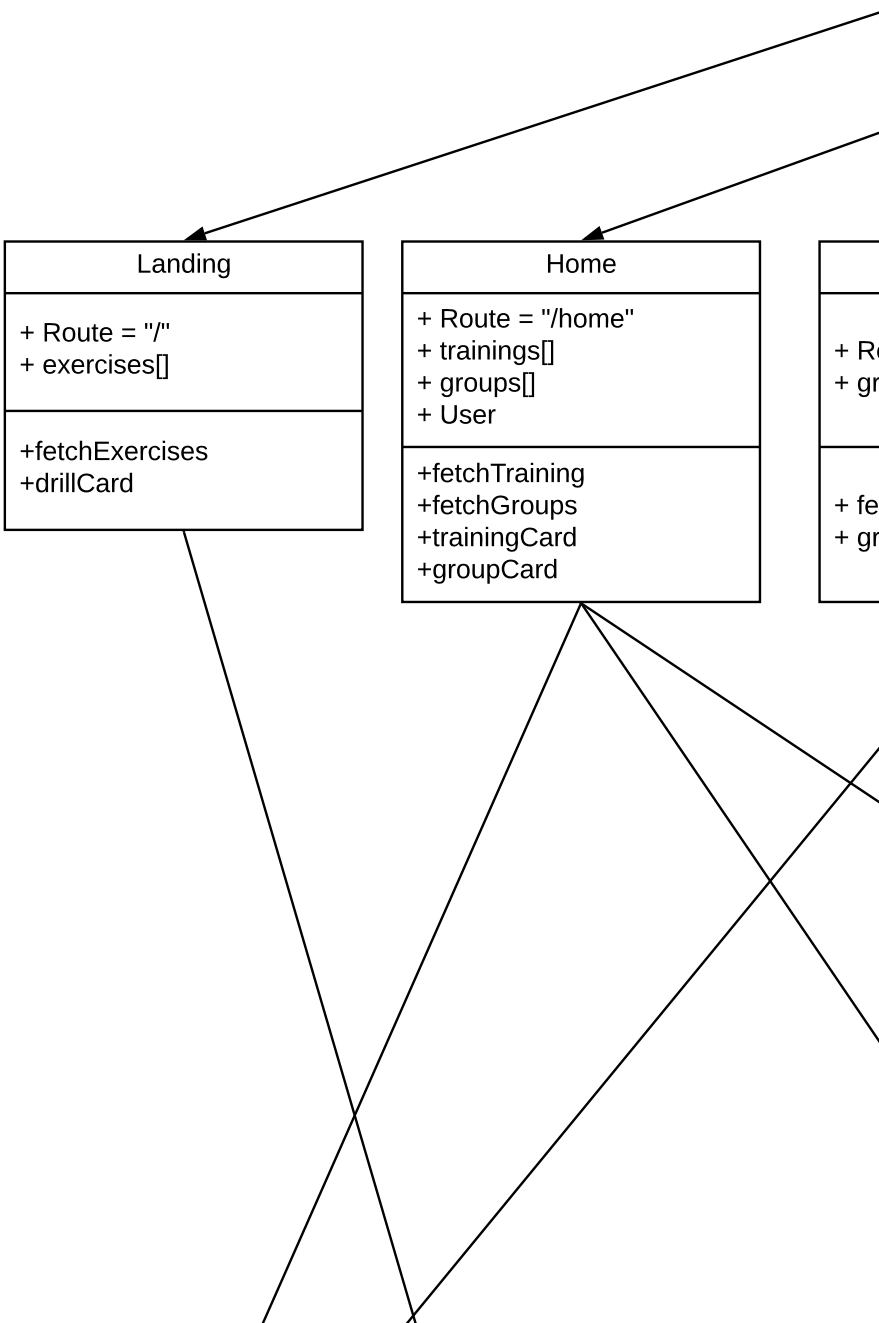
The third and last diagram is that of the front end. It consists of 7 pages and is viewed best as follows. The first 4 pages are set side by side from left to right and the final 3 pages are set in the same order below the first 3 of the first 4. It is again a class diagram, this time displayed as a hierarchy according to their relations.

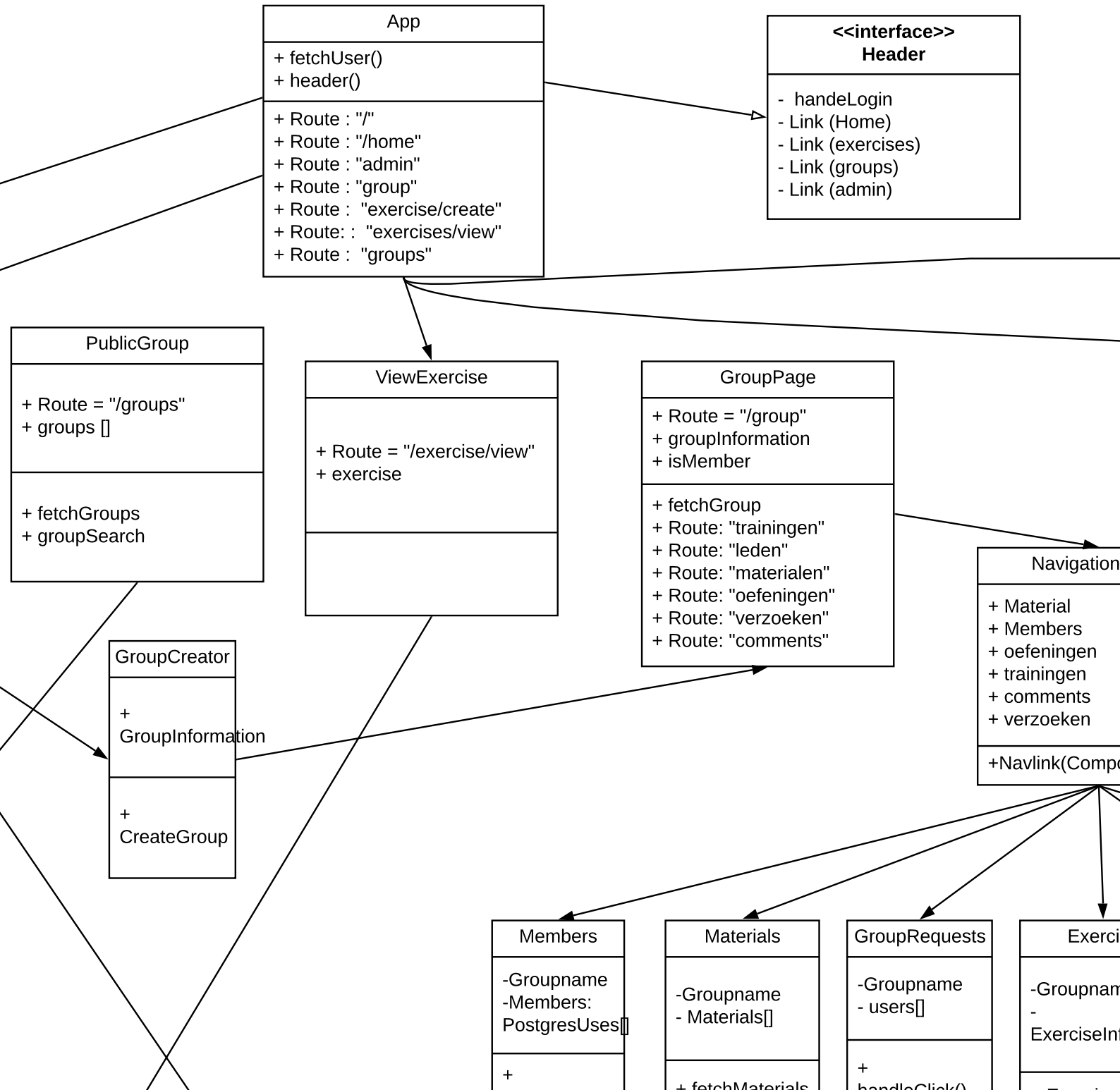

Interfaces
<<Interface>>
User
+ email: String
+ image: String
+ contactEmail: String
+ username: String
<<Interface>>
Exercise
+ id: Integer
+ image: String
+ duration: Integer
+ name: String
+ description: String
<<Interface>>
Training
+ id: Integer
+ image: String
+ name: String
+ username: String
<<Interface>>
Material
+ id: Integer
+ material_string: String
<<Interface>>
Tag
+ name: String
<<Interface>>
ExRequest
+ id: Integer
+ exercise_id: Integer
+ group_id: String
+ status: RequestStatus
+ time_of_request: DateTime
+ time_of_review: DateTime
+ reviewer_id: String
Database
exercise/queries
+ searchInGroup({groupName, phrase}): [Exercis
+ search({requestingUser, phrase}): [Exercise, av
+ getTopFromPublic({amount}): [Exercise, avg_ra
+ getAllPublicExercises(): [Exercise, avg_rating, t
+ getTopFromPublic({amount}): [Exercise, avg_ra
+ getFromTag({tagID}): Exercise[]
+ getByUser({userEmail}): Exercise[]
+ getWithRating({value}): Exercise[]
+ getWithTagAndRating({tagID, value}): Exercise[
+ getFromId({id, requestingUser}): [Exercise, Tag
+ getPublicAndByMail({requestingUser}): [Exercis
+ create({image, duration, exercise_name, exerci
+ createAdvanced({requestingUser, image, hours
exercise_description, group_name, tags, material
+ remove({id}): Void
user/queries
+ getAllUsers(): User[]
+ getAllInGroup({groupName}): User[]
+ getUserByPhrase({phrase}): User[]
+ getByEmail({requestingUser}): User
+ isAdmin({requestingUser}): Boolean
+ setUserName({email, userName}): Void
+ setContactEmail({email, contactEmail}): Void
+ insertUser({email, image, contactEmail, usernar
+ deleteUser(email): Void
tag/queries
+ getAll(): Tag[]
+ getPopular(): Tag[]
+ createTag(name): Void
+ addToExercise(exerciseID, name): Void

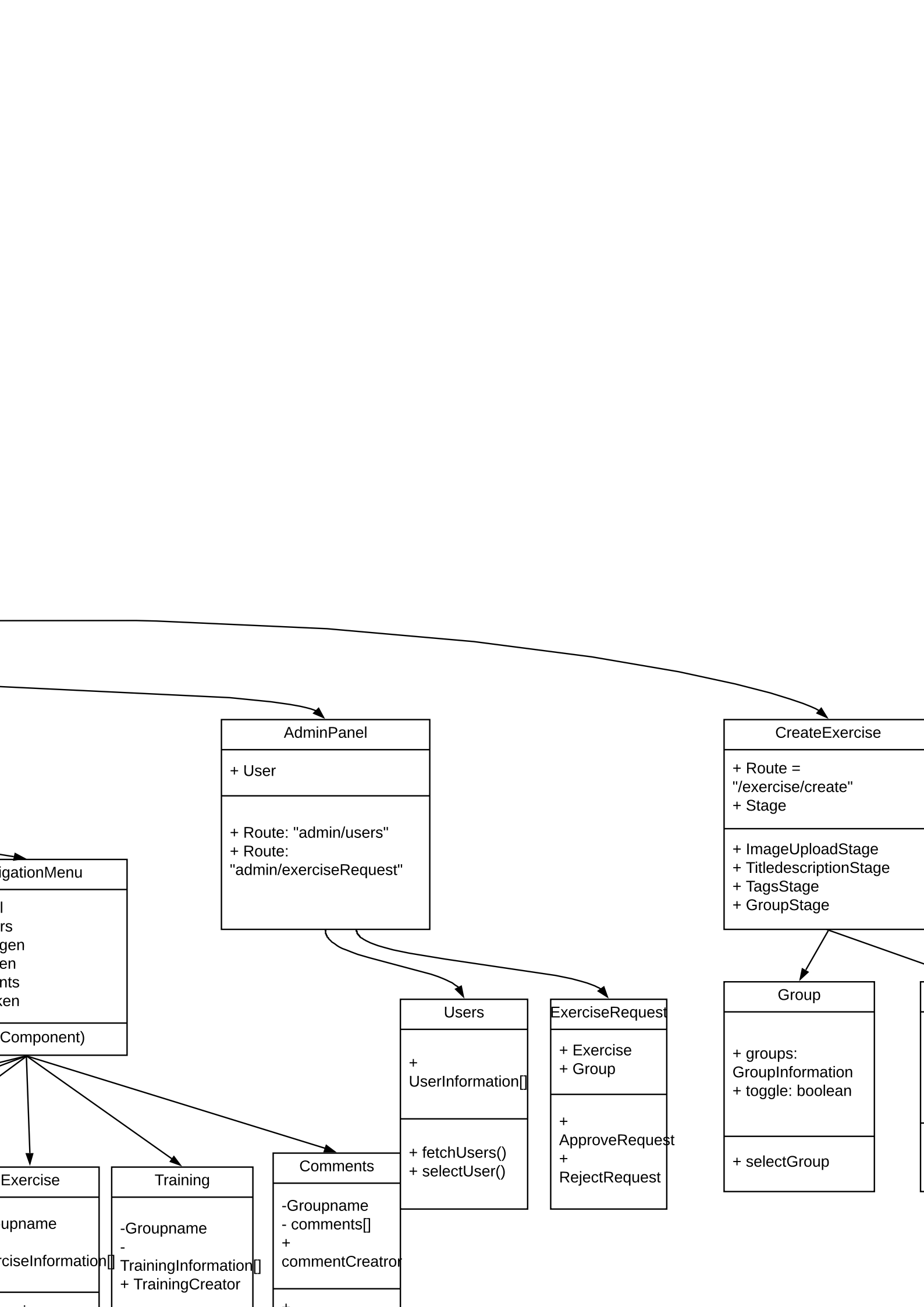

e, avg_rating, tags[]]
g_rating, tags[]]
ting, tags[]]
ags[]]
ting, tags[]]
]
], Rating[], Material[], Group[]
se.id, Exercise.name]
se_description}): Void
, minutes, exercise_name,
s}): Void
me}): String
group/queries
+ getAllGroups(): Group[]
+ getAllGroupsByEmail({requestingUser}): Group[]
+ getGroupByPhrase({phrase}): Group[]
+ getByName({groupName, requestingUser}): Group
+ isGroupPrivate(groupName): Boolean
+ checkGroupAdmin(requestingUser, groupName): Boolean
+ checkGroupMember(requestingUser, groupName): Boolean
+ setPrivate({groupName, value}): Void
+ addExercise(id, groupName): Void
+ deleteExercise({id, groupName, requestingUser}): Void
+ create({name, image, description, requestingUser, isPrivate}): Void
+ deleteGroup({groupName, requestingUser}): Void
+ addUser({requestingUser, groupName}): Void
+ removeUser({requestingUser, groupName}): Void
training/queries
+ byGroup({groupID}): Training[]
+ getAllByEmail({requestingUser}): [Training, amountExercises]
+ getPDF({training_id}): {Training, Group, Exercise[]}
+ create({image, name, description, groupToAddTraining,
exercisesToAddTraining}): Integer
+ addExercise({exercise_id, training_id}): Void
exRequest/queries
+ getAll(): ExRequest[]
+ getByStatus({status}): ExRequest[]
+ create({exercise_id, group_id, time_of_review, reviewer_id}): Void
+ remove({id}): Void
+ approveExRequest({exRequest}): Void
+ rejectExRequest({exRequest}): Void
material/queries
+ createMaterial(name): Void
+ getID(name): Integer

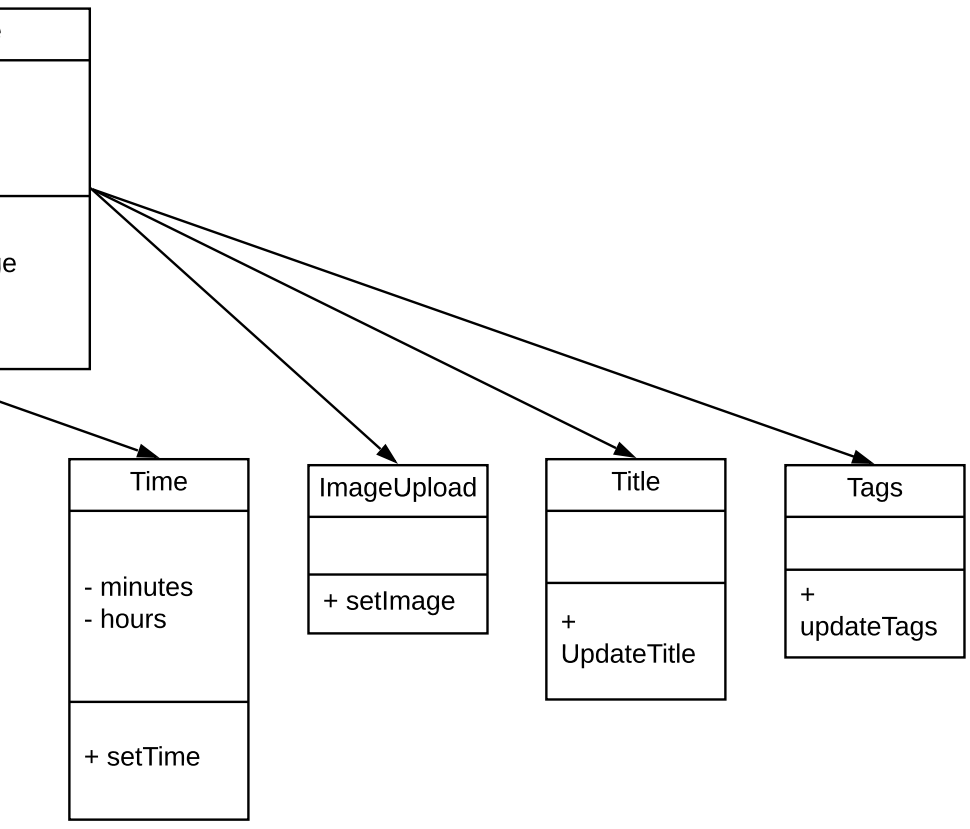

exercise_tag
exercise_i integer PK FK
tag_name varchar(10) PK FK
tags
name varchar(10) PK
exercise_training
exercise_id integer PK FK
training_id integer PK FK
exercise_rating
exercise_id integer PK FK
rating_id integer PK FK
exercise_user
exercise integer PK FK
user_id varchar(60) PK FK
users
email varchar(60) PK
image text N
contact_email varchar(60) N
username varchar(40)
ratings
id serial PK
value integer
user_email varchar(60) FK
training_rating
training_id integer PK FK
rating_id integer PK FK
grprequests
id serial PK
user_i varchar(60) FK
group varchar(50) FK
status enum('in afwachting', 'goedgekeurd', 'afgekeurd')
time_o timestamp
time_o timestamp N
review varchar(60) N FK
user_material
user_id varchar(60) PK FK
material_ integer PK FK
user_group
user_id varchar(60) PK FK
group_id varchar(50) PK FK
exercise_material
exercise_id integer PK FK
material_id integer PK FK
group_moderator
user_id varchar(60) PK FK
group_id varchar(50) PK FK
groups
name varchar(50) PK
image text N
description varchar(100) N
administrator varchar(60) FK
isPrivate boolean
comments
id serial PK
description varchar(100) N
author varchar(60) FK
time_of_commen timestamptz
comment_rating
comment_id integer PK FK
rating_id integer PK FK
materials
id serial PK
material_string varchar(20)
group_material
group_id varchar(50) PK FK
material_i integer PK FK
exrequests
id serial PK
exercise integer FK
group_i varchar(50) FK
status enum('in afwachting', 'goedgekeurd', 'afgekeurd')
time_of_ timestamp
time_of_ timestamp N
reviewe varchar(60) N FK
comment_group
comment_ integer PK FK
group_id varchar(50) PK FK
training_tag
training_id integer PK FK
tag_name varchar(10) PK FK
exercise_group
exercise_id integer PK FK
group_id varchar(50) PK FK
training_group
training_id integer PK FK
group_id varchar(50) PK FK
comment_exercise
comment_id integer PK FK
exercise_id integer PK FK
comment_training
comment_id integer PK FK
training_id integer PK FK
trainings
id serial PK
image text N
name varchar(30) N
description varchar(100) N
exercises
id serial PK
image text N
duration integer
name varchar(30)
description varchar(100)

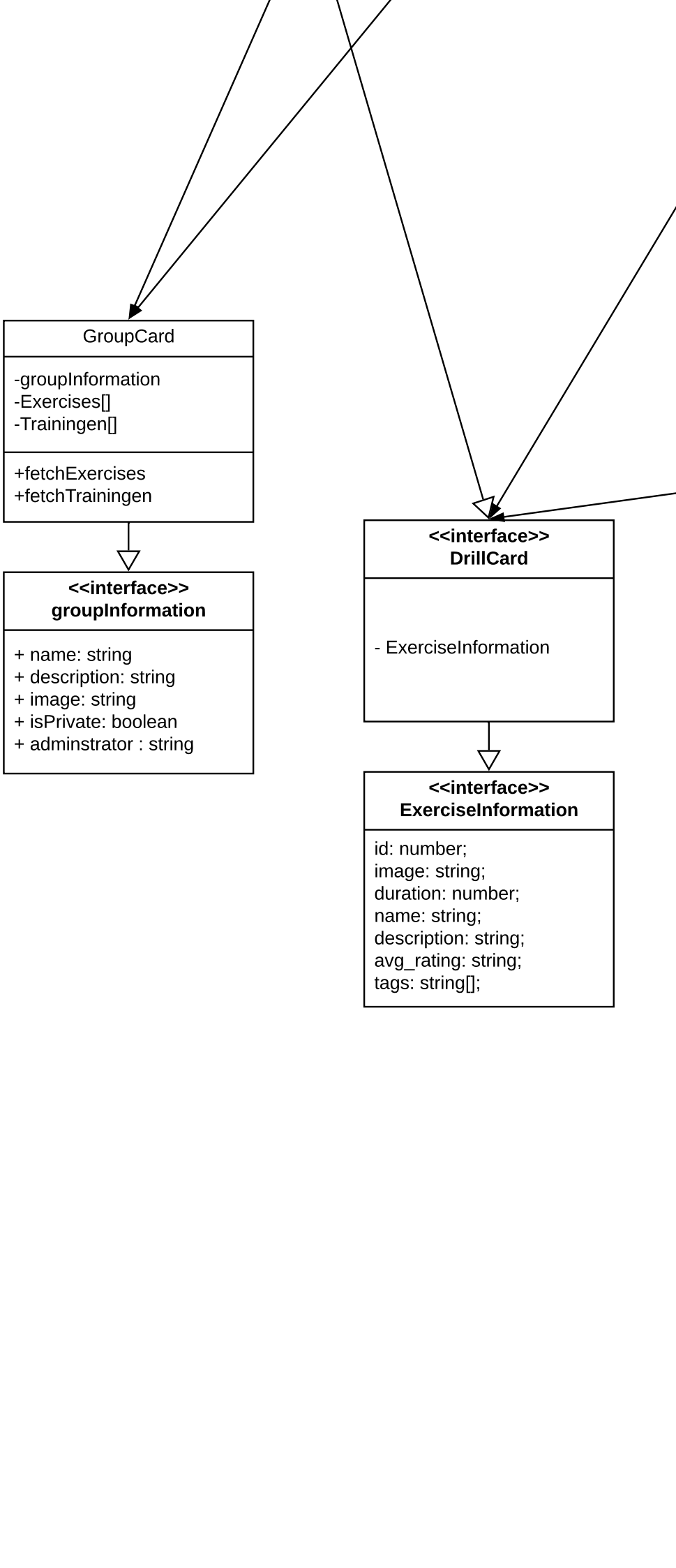
Vertabelo

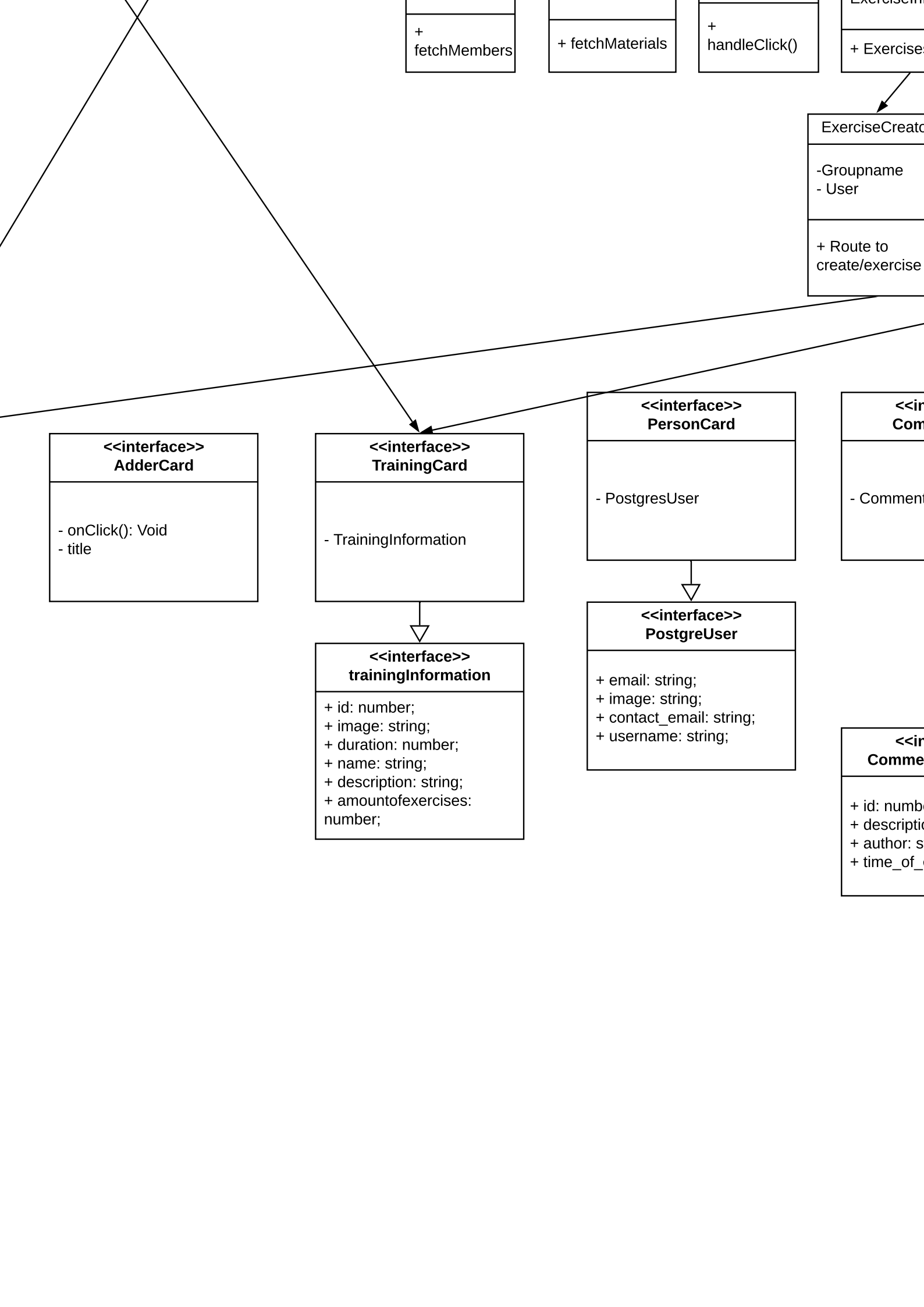

Landing
+ Route = "/"
+ exercises[]
+fetchExercises
+drillCard
Home
+ Route = "/home"
+ trainings[]
+ groups[]
+ User
+fetchTraining
+fetchGroups
+trainingCard
+groupCard

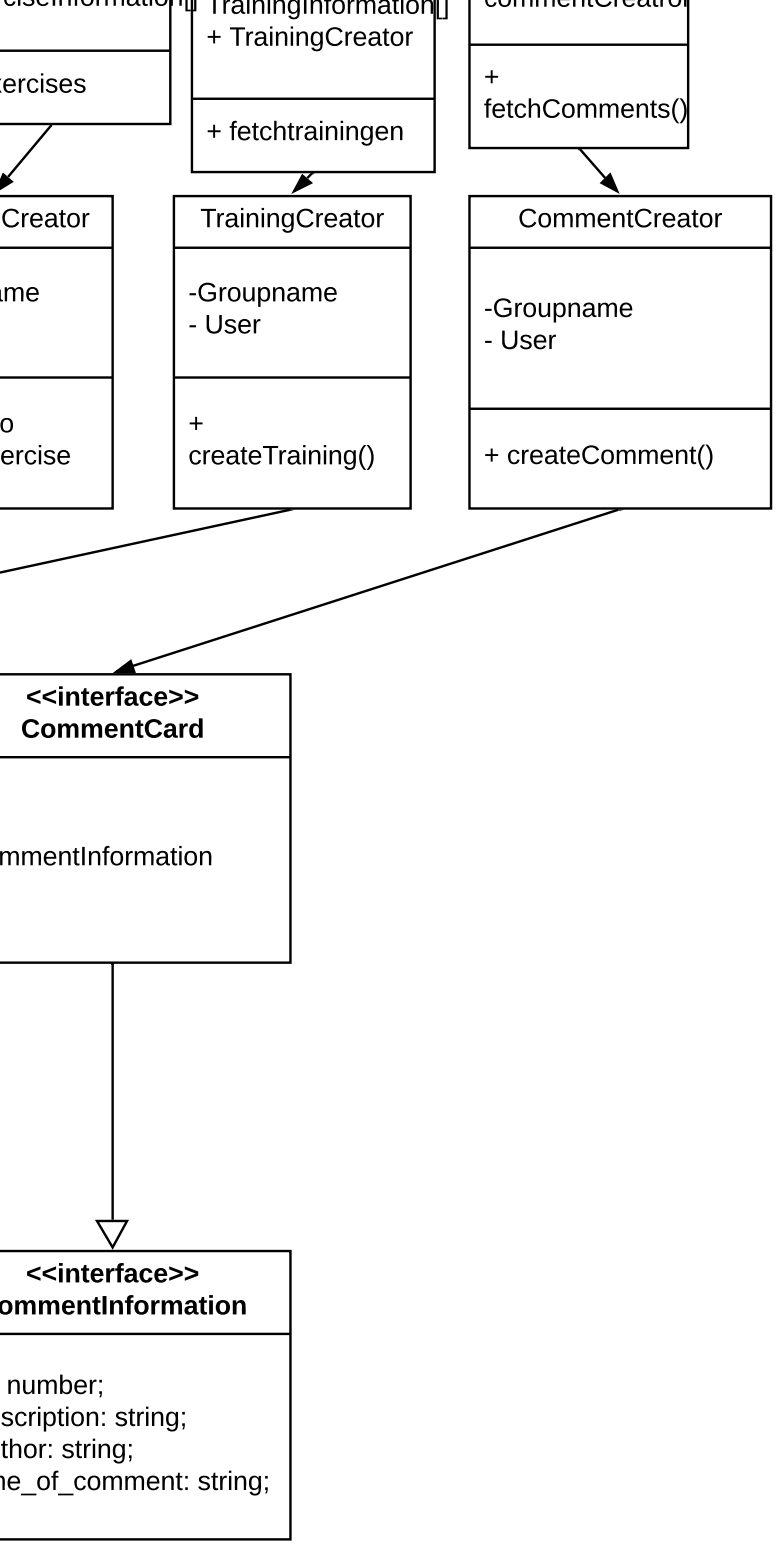

App
+ fetchUser()
+ header()
+ Route : "/"
+ Route : "/home"
+ Route : "admin"
+ Route : "group"
+ Route : "exercise/create"
+ Route: : "exercises/view"
+ Route : "groups"
<<interface>>
Header
- handeLogin
- Link (Home)
- Link (exercises)
- Link (groups)
- Link (admin)
PublicGroup
+ Route = "/groups"
+ groups []
+ fetchGroups
+ groupSearch
ViewExercise
+ Route = "/exercise/view"
+ exercise
GroupPage
+ Route = "/group"
+ groupInformation
+ isMember
+ fetchGroup
+ Route: "trainingen"
+ Route: "leden"
+ Route: "materialen"
+ Route: "oefeningen"
+ Route: "verzoeken"
+ Route: "comments"
Navigation
+ Material
+ Members
+ oefeningen
+ trainingen
+ comments
+ verzoeken
+Navlink(Compo
GroupCreator
+ GroupInformation
+ CreateGroup
Members
-Groupname
-Members: PostgresUses[]
Materials
-Groupname
- Materials[]
GroupRequests
-Groupname
- users[]
+ handleClick()
Exerci
-Groupnam
-
ExerciseInf


AdminPanel
+ User
+ Route: "admin/users"
+ Route:
"admin/exerciseRequest"
CreateExercise
+ Route =
"/exercise/create"
+ Stage
+ ImageUploadStage
+ TitledescriptionStage
+ TagsStage
+ GroupStage
igationMenu
l
rs
gen
en
nts
ken
Component)
Users
+
UserInformation[]
+ fetchUsers()
+ selectUser()
ExerciseRequest
+ Exercise
+ Group
+
ApproveRequest
+
RejectRequest
Group
+ groups:
GroupInformation
+ toggle: boolean
+ selectGroup
Exercise
upname
rciseInformation[]
Training
-Groupname
-
TrainingInformation[]
+ TrainingCreator
Comments
-Groupname
- comments[]
+
commentCreatror


Time
- minutes
- hours
+ setTime
ImageUpload
+ setImage
Title
+
UpdateTitle
Tags
+
updateTags


GroupCard
-groupInformation
-Exercises[]
-Trainingen[]
+fetchExercises
+fetchTrainingen
<<interface>>
groupInformation
+ name: string
+ description: string
+ image: string
+ isPrivate: boolean
+ adminstrator : string
<<interface>>
DrillCard
- ExerciseInformation
<<interface>>
ExerciseInformation
id: number;
image: string;
duration: number;
name: string;
description: string;
avg_rating: string;
tags: string[];


+
fetchMembers
+ fetchMaterials
+
handleClick()
+ Exercises
ExerciseCreatc
-Groupname
- User
+ Route to
create/exercise
<<interface>>
AdderCard
- onClick(): Void
- title
<<interface>>
TrainingCard
- TrainingInformation
<<interface>>
trainingInformation
+ id: number;
+ image: string;
+ duration: number;
+ name: string;
+ description: string;
+ amountofexercises:
number;
<<interface>>
PersonCard
- PostgresUser
<<interface>>
PostgreUser
+ email: string;
+ image: string;
+ contact_email: string;
+ username: string;
<<in
Com
- Comment
<<in
Comme
+ id: numb
+ descriptio
+ author: s
+ time_of_


+ TrainingCreator
+ fetchtrainingen
+
fetchComments()
Creator
ame
o
ercise
TrainingCreator
-Groupname
- User
+
createTraining()
CommentCreator
-Groupname
- User
+ createComment()
<<interface>>
CommentCard
mmentInformation
<<interface>>
ommentInformation
number;
scription: string;
thor: string;
me_of_comment: string;

B.1 ER-Diagrams

Gymplanner DB ER diagram

Database model documentation

Table of contents

1. Model details

Model name:

Gymplanner DB ER diagram

Version:

2.3

Database engine:

PostgreSQL

Description:

2. Tables

2.1. Table trainings

2.1.1. Columns

Column name	Type	Properties	Description
id	serial	PK	
image	text	null	
name	varchar(30)	null	
description	varchar(100)	null	

2.2. Table exercises

2.2.1. Columns

Column name	Type	Properties	Description
id	serial	PK	
image	text	null	
duration	integer		
name	varchar(30)		
description	varchar(100)		

2.3. Table users

2.3.1. Columns

Column name	Type	Properties	Description
email	varchar(60)	PK	
image	text	null	
contact_email	varchar(60)	null	
username	varchar(40)		

2.4. Table ratings

2.4.1. Columns

Column name	Type	Properties	Description
id	serial	PK	
value	integer		

user_email	varchar(60)		

2.5. Table tags

2.5.1. Columns

Column name	Type	Properties	Description
name	varchar(10)	PK	

2.6. Table groups

2.6.1. Columns

Column name	Type	Properties	Description
name	varchar(50)	PK	
image	text	null	
description	varchar(100)	null	
administrator	varchar(60)		
isPrivate	boolean		

2.7. Table materials

2.7.1. Columns

Column name	Type	Properties	Description
id	serial	PK	
material_string	varchar(20)		

2.8. Table exrequests

2.8.1. Columns

Column name	Type	Properties	Description
id	serial	PK	
exercise_id	integer		
group_id	varchar(50)		
status	enum('in afwachting', 'goedgekeurd', 'afgekeurd')		
time_of_request	timestamp		
time_of_review	timestamp	null	

reviewer_id	varchar(60)	null	

2.9. Table grprequests

2.9.1. Columns

Column name	Type	Properties	Description
id	serial	PK	
user_id	varchar(60)		
group_id	varchar(50)		
status	enum('in afwachting', 'goedgekeurd', 'afgekeurd')		
time_of_request	timestamp		
time_of_review	timestamp	null	
reviewer_id	varchar(60)	null	

2.10. Table comments

2.10.1. Columns

Column name	Type	Properties	Description
id	serial	PK	
description	varchar(100)	null	
author	varchar(60)		
time_of_comment	timestamptz		

2.11. Table comment_exercise

2.11.1. Columns

Column name	Type	Properties	Description
comment_id	integer	PK	
exercise_id	integer	PK	

2.12. Table comment_training

2.12.1. Columns

Column name	Type	Properties	Description
comment_id	integer	PK	

training_id	integer	PK	

2.13. Table comment_group

2.13.1. Columns

Column name	Type	Properties	Description
comment_id	integer	PK	
group_id	varchar(50)	PK	

2.14. Table comment_rating

2.14.1. Columns

Column name	Type	Properties	Description
comment_id	integer	PK	
rating_id	integer	PK	

2.15. Table exercise_tag

2.15.1. Columns

Column name	Type	Properties	Description
exercise_id	integer	PK	
tag_name	varchar(10)	PK	

2.16. Table exercise_material

2.16.1. Columns

Column name	Type	Properties	Description
exercise_id	integer	PK	
material_id	integer	PK	

2.17. Table user_material

2.17.1. Columns

Column name	Type	Properties	Description
user_id	varchar(60)	PK	
material_id	integer	PK	

2.18. Table exercise_group

2.18.1. Columns

Column name	Type	Properties	Description
exercise_id	integer	PK	
group_id	varchar(50)	PK	

2.19. Table exercise_user

2.19.1. Columns

Column name	Type	Properties	Description
exercise_id	integer	PK	
user_id	varchar(60)	PK	

2.20. Table training_group

2.20.1. Columns

Column name	Type	Properties	Description
training_id	integer	PK	
group_id	varchar(50)	PK	

2.21. Table training_tag

2.21.1. Columns

Column name	Type	Properties	Description
training_id	integer	PK	
tag_name	varchar(10)	PK	

2.22. Table exercise_training

2.22.1. Columns

Column name	Type	Properties	Description
exercise_id	integer	PK	
training_id	integer	PK	

2.23. Table exercise_rating

2.23.1. Columns

Column name	Type	Properties	Description
exercise_id	integer	PK	
rating_id	integer	PK	

2.24. Table training_rating

2.24.1. Columns

Column name	Type	Properties	Description
training_id	integer	PK	
rating_id	integer	PK	

2.25. Table user_group

2.25.1. Columns

Column name	Type	Properties	Description
user_id	varchar(60)	PK	
group_id	varchar(50)	PK	

2.26. Table group_moderator

2.26.1. Columns

Column name	Type	Properties	Description
user_id	varchar(60)	PK	
group_id	varchar(50)	PK	

2.27. Table group_material

2.27.1. Columns

Column name	Type	Properties	Description
group_id	varchar(50)	PK	
material_id	integer	PK	

3. References

3.1. Reference FK_0

users	0..*	ratings
email	<->	user_email

3.2. Reference FK_1

users	0..*	groups
email	<->	administrator

3.3. Reference FK_2

exercises	0..*	exrequests
id	<->	exercise_id

3.4. Reference FK_3

groups	0..*	exrequests
name	<->	group_id

3.5. Reference FK_4

users	0..*	grprequests
email	<->	user_id

3.6. Reference FK_5

groups	0..*	grprequests
name	<->	group_id

3.7. Reference FK_6

users	0..*	comments
email	<->	author

3.8. Reference FK_7

comments	0..*	comment_exercise
id	<->	comment_id

3.9. Reference FK_8

exercises	0..*	comment_exercise
id	<->	exercise_id

3.10. Reference FK_9

comments	0..*	comment_training
id	<->	comment_id

3.11. Reference FK_10

trainings	0..*	comment_training
id	<->	training_id

3.12. Reference FK_11

comments	0..*	comment_group
id	<->	comment_id

3.13. Reference FK_12

groups	0..*	comment_group
name	<->	group_id

3.14. Reference FK_13

comments	0..*	comment_rating
id	<->	comment_id

3.15. Reference FK_14

ratings	0..*	comment_rating
id	<->	rating_id

3.16. Reference FK_15

exercises	0..*	exercise_tag
id	<->	exercise_id

3.17. Reference FK_16

tags	0..*	exercise_tag
name	<->	tag_name

3.18. Reference FK_17

exercises	0..*	exercise_material
id	<->	exercise_id

3.19. Reference FK_18

materials	0..*	exercise_material
id	<->	material_id

3.20. Reference FK_19

users	0..*	user_material
email	<->	user_id

3.21. Reference FK_20

materials	0..*	user_material
id	<->	material_id

3.22. Reference FK_21

exercises	0..*	exercise_group
id	<->	exercise_id

3.23. Reference FK_22

groups	0..*	exercise_group
name	<->	group_id

3.24. Reference FK_23

exercises	0..*	exercise_user
id	<->	exercise_id

3.25. Reference FK_24

users	0..*	exercise_user
email	<->	user_id

3.26. Reference FK_25

trainings	0..*	training_group
id	<->	training_id

3.27. Reference FK_26

groups	0..*	training_group
name	<->	group_id

3.28. Reference FK_27

trainings	0..*	training_tag
id	<->	training_id

3.29. Reference FK_28

tags	0..*	training_tag
name	<->	tag_name

3.30. Reference FK_29

exercises	0..*	exercise_training
id	<->	exercise_id

3.31. Reference FK_30

trainings	0..*	exercise_training
id	<->	training_id

3.32. Reference FK_31

exercises	0..*	exercise_rating
id	<->	exercise_id

3.33. Reference FK_32

ratings	0..*	exercise_rating
id	<->	rating_id

3.34. Reference FK_33

trainings	0..*	training_rating
id	<->	training_id

3.35. Reference FK_34

ratings	0..*	training_rating
id	<->	rating_id

3.36. Reference FK_35

users	0..*	user_group
email	<->	user_id

3.37. Reference FK_36

groups	0..*	user_group
name	<->	group_id

3.38. Reference FK_37

users	0..*	group_moderator
email	<->	user_id

3.39. Reference FK_38

groups	0..*	group_moderator
name	<->	group_id

3.40. Reference FK_39

groups	0..*	group_material
name	<->	group_id

3.41. Reference FK_40

materials	0..*	group_material
id	<->	material_id

3.42. Reference exrequests_users

users	0..*	exrequests
email	<->	reviewer_id

3.43. Reference grprequests_users

users	0..*	grprequests
email	<->	reviewer_id

C

Coverage

coverage.png

All files

84.5% Statements 447/529 **68.48%** Branches 63/92 **83.33%** Functions 70/84 **82.93%** Lines 306/369

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

File	Statements		Branches		Functions		Lines	
src	100%	5/5	100%	0/0	100%	1/1	100%	5/5
src/db	87.5%	14/16	100%	2/2	50%	1/2	84.62%	11/13
src/db/exercise	93.38%	127/136	75%	9/12	86.36%	19/22	94.19%	81/86
src/db/exrequest	100%	18/18	100%	0/0	100%	4/4	100%	12/12
src/db/group	95.31%	122/128	80%	28/35	100%	14/14	95.18%	79/83
src/db/material	100%	9/9	100%	4/4	100%	2/2	100%	5/5
src/db/rating	83.33%	10/12	100%	0/0	66.67%	2/3	85.71%	6/7
src/db/tags	100%	27/27	100%	10/10	100%	5/5	100%	18/18
src/db/training	100%	46/46	100%	7/7	100%	5/5	100%	30/30
src/db/user	100%	46/46	100%	3/3	100%	8/8	100%	37/37
src/routes/db	21.25%	17/80	0%	0/19	40%	6/15	25%	17/68

Figure C.1: Back-end test coverage

coverage.png

All files

99.11% Statements 222/224 **91.23%** Branches 52/57 **98.33%** Functions 59/60 **99.06%** Lines 211/213

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

File	Statements		Branches		Functions		Lines	
redux	100%	24/24	100%	11/11	100%	6/6	100%	23/23
redux/adminPanel/exRequests	100%	40/40	100%	2/2	100%	12/12	100%	38/38
redux/adminPanel/users	100%	28/28	100%	8/8	100%	8/8	100%	27/27
redux/groupPage	96.55%	56/58	66.67%	10/15	93.33%	14/15	96.23%	51/53
redux/homePage	100%	41/41	100%	11/11	100%	11/11	100%	40/40
redux/public	100%	33/33	100%	10/10	100%	8/8	100%	32/32

Figure C.2: Front-end test coverage

D

SIG-feedback

Beste,

Hierbij ontvang je onze evaluatie van de door jou opgestuurde code. De evaluatie bevat een aantal aanbevelingen die meegenomen kunnen worden tijdens het vervolg van het project. Bij de evaluatie van de tweede upload kijken we in hoeverre de onderhoudbaarheid is verbeterd, en of de feedback is geaddresseerd. Deze evaluatie heeft als doel om studenten bewuster te maken van de onderhoudbaarheid van hun code, en dient niet gebruikt te worden voor andere doeleinden.
Let tijdens het bekijken van de feedback op het volgende: - Het is belangrijk om de feedback in de context van de huidige onderhoudbaarheid te zien. Als een project al relatief hoog scoort zijn de genoemde punten niet 'fout', maar aankopingspunten om een nog hogere score te behalen. In veel gevallen zullen dit marginale verbeteringen zijn, grote verbeteringen zijn immers moeilijk te behalen als de code al goed onderhoudbaar is. - Voor de herkenning van testcode maken we gebruik van geautomatiseerde detectie. Dit werkt voor de gangbare technologieën en frameworks, maar het zou kunnen dat we jullie testcode hebben gemist. Laat het in dat geval vooral weten, maar we vragen hier ook om begrip dat het voor ons niet te doen is om voor elk groepje handmatig te kijken of er nog ergens testcode zit. - Hetzelfde geldt voor libraries: als er voldaan wordt aan gangbare conventies worden die automatisch niet meegenomen tijdens de analyse, maar ook hier is het mogelijk dat we iets gemist hebben.
Mochten er nog vragen of opmerkingen zijn dan horen we dat graag.
Met vriendelijke groet, Dennis Bijlsma

[Feedback]
De code van het systeem scoort 4.6 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code bovengemiddeld onderhoudbaar is. De hoogste score is niet behaald door een lagere score voor Unit Size.
Bij Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Dit kan verschillende redenen hebben, maar de meest voorkomende is dat een methode te veel functionaliteit bevat. Vaak was de methode oorspronkelijk kleiner, maar is deze in de loop van tijd steeds verder uitgebreid. De aanwezigheid van commentaar die stukken code van elkaar scheiden is meestal een indicator dat de methode meerdere verantwoordelijkheden bevat. Het opsplitsen van dit soort methodes zorgt er voor dat elke methode een duidelijke en specifieke functionele scope heeft. Daarnaast wordt de functionaliteit op deze manier vanzelf gedocumenteerd via methodenamen.
Voorbeelden in jullie project: - simpleResponses.ts.pdfPostResponse
Als laatste nog de opmerking dat er geen (unit)test-code is gevonden in de code-upload. Het is sterk aan te raden om in ieder geval voor de belangrijkste delen van de functionaliteit automatische tests gedefinieerd te hebben om ervoor te zorgen dat eventuele aanpassingen niet voor ongewenst gedrag zorgen. Op lange termijn maakt de aanwezigheid van unit tests je code ook flexibeler, omdat aanpassingen kunnen worden doorgevoerd zonder de stabiliteit in gevaar te brengen.
Over het algemeen scoort de code dus bovengemiddeld, hopelijk lukt het om dit niveau te behouden tijdens de rest van de ontwikkelfase.

Bibliography

[1] The PostgreSQL Global Development Group. PostgreSQL global development group, 1996. URL https://www.postgresql.org/about/.

[2] Jetbrains. WebStorm, 2000. URL https://www.jetbrains.com/help/webstorm/getting-started-with-webstorm.html.

[3] Agile Project. The dsdm agile project framework (2014 onwards), Jun 2017. URL https://www.agilebusiness.org/content/moscow-prioritisation.

[4] SAX-VSM. TF-IDF, 2016. URL https://jmotif.github.io/sax-vsm_site/morea/algorithm/TFIDF.html.

[5] Ken Schwaber. *Agile project management with Scrum*. Microsoft press, 2004.

[6] Joost Visser. Sig/tÜvit evaluation criteria trusted product maintainability, 2016. URL https://www.softwareimprovementgroup.com/news-knowledge/sig-models/.