Sports Exercise Battle

Table of Contents

[1 Planning 1](#_Toc197442085)

[1.1 Reading the specifications and laying the groundwork (2.5h) 1](#_Toc197442086)

[1.2 Planning the database (2.5h) 1](#_Toc197442087)

[2 Building the Foundations 2](#_Toc197442088)

[2.1 Starting to code – creating the models (3h) 2](#_Toc197442089)

[2.2 Setting up the database and connecting it with the application (6h) 3](#_Toc197442090)

[2.3 HTTP Server and the router (7.5h) 3](#_Toc197442091)

[2.4 TCP Server for the Tournament (4.5h) 3](#_Toc197442092)

[3 Implementing Business Logic 3](#_Toc197442093)

[3.1 Login, Register and Token Authentication (7h) 3](#_Toc197442094)

[3.2 Get and Update Profile (3.5h) 4](#_Toc197442095)

[3.3 Get and Add History (5h) 4](#_Toc197442096)

[3.4 Fixing All the broken code and cleanup (2.5h) 4](#_Toc197442097)

[3.5 View Stats (1.5h) 4](#_Toc197442098)

# Planning

## Reading the specifications and laying the groundwork (2.5h)

The first thing I did was to carefully read through the specifications in order not to miss anything and risk having to scrap my ideas. After carefully reading through the text, I brainstormed and made thoughts about how the structure of the program and database should look like as well as what to implement as the additional feature. The reason I started planning the feature early is because implementing it after being finished would be harder, since I would have to dig through my code, find ways of implementing it and end up having to alter my code in a way that would allow the feature to function. By pre planning however, I can already write my code in a way that would make implementing that feature easier.

After then Initializing my repository, I started making prototypes for the database and also created another document in which I’m storing all my ideas.

## Planning the database (2.5h)

This is one of the first things I did. After reading the specifications I got to work and used a tool in order to visualize and plan my database. I ended up having to change the tables quite a lot, since at first, I wanted to go overboard, create everything as professional as possible, create tables to make retrieving data easier, to make implementing things later easier etc. At the end however, I decided on keeping things simple and only creating the tables that I actually needed.

This ended up being the Users and history table. At first, I created tables like session or tournament, but since those were not necessary according to the specifications, I ended up removing them and only kept the previously mentioned tables.

This is what the database ended up looking like:

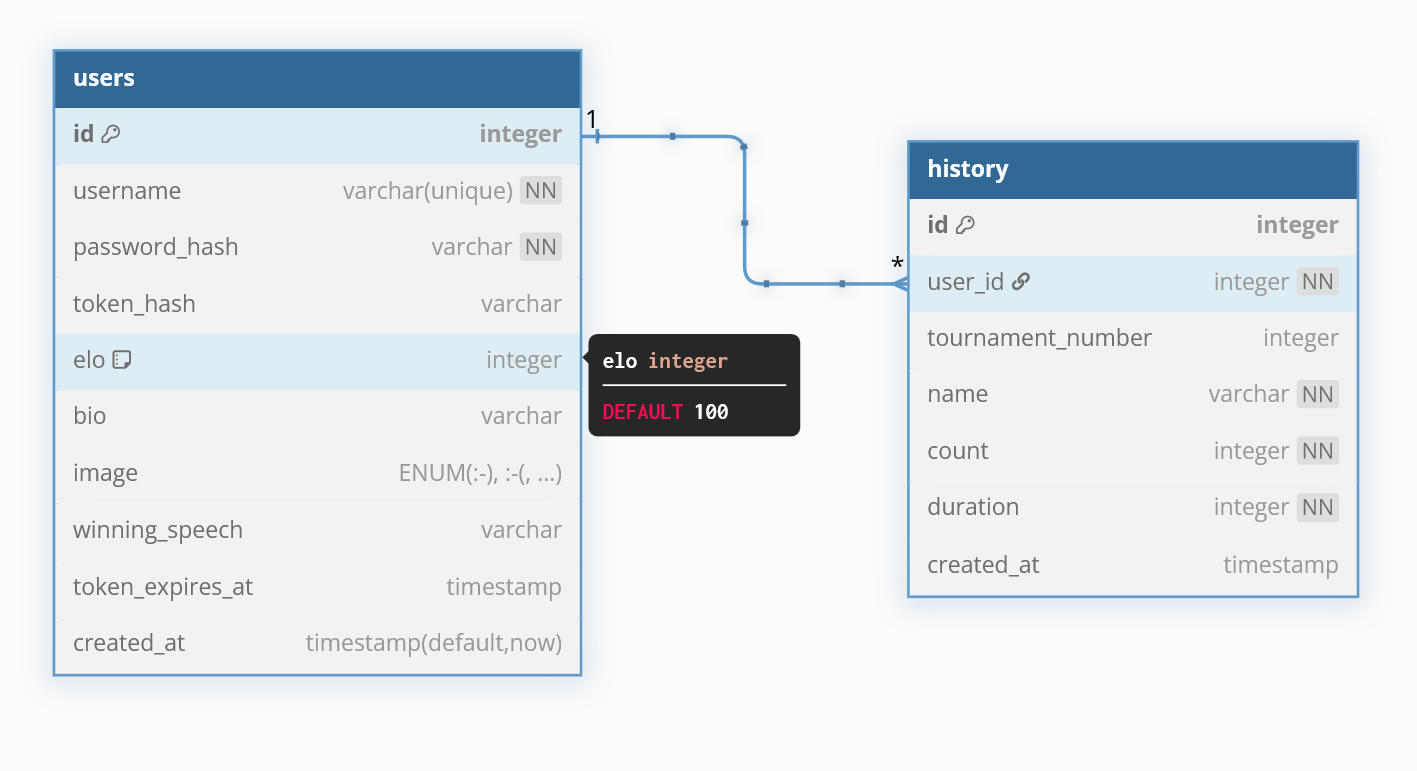


Figure 1: Vogue representation of database (https://www.dbdiagram.io/d)

# Building the Foundations

## Starting to code – creating the models (3h)

My first challenge was that I haven’t coded using C# in almost 5 months, since the university I currently am studying abroad at only offers Java, the good news however, is that we have worked a lot with TCP/WebSocket’s as well as REST, hence I feel more than confident starting this project. My first action will be to quickly revise our Software Engineering course in order to get fluent in C# again.

After going over the basics, I created the models “User” and “PushUpRecord”, which contain all the data needed for the program.

These Models contain:

|  |  |
| --- | --- |
| **User** | **PushUpRecord** |
| Int Id | Int Id |
| string UserName | Int UserId |
| string PasswordHash | Int TournamentNumber |
| Int Elo | String Name |
| String Token | Int Count |
| String? Image  Enum UserImage  {  Happy, // :-)  Sad. // :-(  …  } | Int Duration |
| String? Bio |  |
| DateTime? TokenExpiresAt |  |

I also implemented functions like HashValue, which turns the Token and Password into a hash and also created the functions CreateAndSetToken, SetImage and SetPassword.

## Setting up the database and connecting it with the application (6h)

I used the database template I made above to create the database including the tables inside a docker container. After that I started researching, since I was not too sure on how to create a decent Database connection file. After hours of researching and refining, the finished product is able to establish a connection, close it and dispose of it, which gets rid of unmanaged resources.

It also has functions to allow for CRUD operations. I forgot about CRUD when merging the branch, hence I was forced to return to the branch and create these functions before continuing with the http server.

Later one I also realized that functions like open, close or dispose are not really the best way to do it, so I decided on opening the database when a query is being run, after the query finished, it is being disposed. Everything is async.

## HTTP Server and the router (7.5h)

The HTTP consists of the Server and an router. The Http consists of the function StartAsync and HandleRequestAsync. The Http server triggers the Router, who is responsible for routing. RequestRouter consists of AddRoute, RouteReqeustAsync and a new class called RouteComparer that implements IEqualityComparer, in order to compare the routes efficiently.

## TCP Server for the Tournament (4.5h)

Since Im creating the TCP server ahead of the services which are supposed to send the tournament data I won’t be able to properly test it. I have to admit that this was not a wise choice, but since I already started, I will try to create the basic structure and do the actually implementation in the tournament branch. Suddenly stopping and continuing with the services would probably cause thing like conflicts which would take me time to fix it, which I most definitely not have.

# Implementing Business Logic

## Login, Register and Token Authentication (7h)

Since I already created the database aswell as the http server and the routes, only thing missing are the controllers, services and repositories, on which I will start working now in order to implement Login and Register. I also decided on adding the token now, this took way longer than expected however, since I kept running into issues.

## Get and Update Profile (3.5h)

I’m using the UserController, UserService and UserRepository for this. A get request to visit aour own profile and put to update it can be made. The only reason it took this long was because I overlooked an ! and ended up changing my code because of this and had to change it back afterwards.

## Get and Add History (5h)

I was totally wrong about this. Creating the functions only took like 30mins, everything worked till it suddenly didn’t. After 5 hours I finally finished this, but my entire code suffered from it, since I was forced to destroy it and delete many of the extra functions.

## Fixing All the broken code and cleanup (2.5h)

Even though the due date is over, I still want to finish this project, and since I destroyed most of my code in order to finish the history, I will take my time now to fix everything and make my code look decent again. I was able to fix everything and even improve on the code, I created new functions in order to make it easier for services like userService to check hashes/get the userId by creating new functions inside the authService, who is responsible for everything regarding security. I also changed the output, first the history outputted the total amount of pushups an the total duration, now I made it so that it simply shows the entire history instead of the total amount.

## View Stats (1.5h)

This was kind of tough, since I didn’t really know in which Controller/Service to put this into. I spent most of the time simply thinking about this and decided to create a new controller and service, which I will use for viewing stats, score and tournament, since all 3 of these need access to data from both of my tables, users and history(PushUpRecords).

## Refactoring and improving my code as well as improving the outputs (6h)

I first removed the hashing and token functions from authService and created new Services for both of them. After that, I refactored all the code that was effected by this. One of the reason I did this was because of the SRP. After that I also created a function in the tokenService which allowed me to verify if a token is valid by checking it against the tokens in the db and also checking the expire date, this function can either return (if valid) a bool saying that the token is valid, or when needed, a bool aswell as the tokenHash, which then can be used for things like searching a userId. I created the function this way in order to give the class as little information as possible, easily being able to leave out unnecessary information. I also refactored other code, added more comments and also improved the json messages being sent with the responses in order for them to be more informative.