



VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Nhat Truong

TOURNAMENT MANAGEMENT MOBILE APPLICATION

Technology and Communication
2024

ACKNOWLEDGEMENTS

I want to thank every VAMK teacher and staff. All of you have, directly or indirectly, accompanied me and led me to academic success.

I also thank Dr. Ghodrat Moghadampour, the Principal Lecturer in Software Engineering and my thesis advisor, for giving detailed instructions about the final thesis.

Finally, I want to give my best thankful sentences to my family and friends who financially and spiritually support me for almost four years of academic studies in Finland, where I have been far away from most of them most of the time.

ABSTRACT

Author	Nhat Truong
Title	Tournament Management Mobile Application
Year	2024
Language	English
Pages	74
Name of Supervisor	Dr. Ghodrat Moghadampour

The goal of this thesis project was to create a mobile application where users could manage multiple stages in multiple tournaments. In each stage, users could view and edit match results, then the overall results of that stage had to be calculated and updated automatically.

PostgreSQL relational database was used to store user information and credentials, and tournament data. A server, written in Python and using Flask module, was responsible for user data authentication and authorization. Another server, using ASP.NET Core Web API template, handled tournament, stage and match data. Finally, React Native was used for the client application.

The application works on both iOS and Android devices. With this application, tournament organizations can follow the progress and update the results of their tournament on their hand quickly while standing at the venue or beside the playground. They can also share their tournament results publicly.

Keywords Tournament, PostgreSQL, Python, ASP.NET Core, Expo, React Native, TypeScript

CONTENTS

ACKNOWLEDGEMENTS

ABSTRACT

1	INTRODUCTION	1
2	RELEVANT TECHNOLOGIES	3
2.1	PostgreSQL	3
2.2	Python & Relevant Modules	3
2.3	C# / .NET (Core) & Relevant Frameworks	4
2.4	JWT	5
2.5	React Native	6
2.5.1	Expo	7
2.5.2	JavaScript & TypeScript	7
2.5.3	React Native Libraries	7
3	APPLICATION DESCRIPTION	9
3.1	Quality function deployment	9
3.2	Use-case Diagram	11
3.3	Sequence Diagram	11
3.4	Architectural Diagram	15
3.5	Class Diagram	16
4	DATABASE DESIGN	19
5	GUI DESIGN	21
6	IMPLEMENTATION	37
6.1	Authentication Server	37
6.2	Tournament Data Server	43
6.3	The client	55
7	TESTING	62
8	CONCLUSIONS	64
	REFERENCES	65

LIST OF FIGURES, TABLES AND CODE SNIPPETS

Figure 1. JWT example [12].....	6
Figure 2. Tournament management system use cases.	11
Figure 3. Authentication sequence diagram.....	12
Figure 4. User information & password change sequence diagram.	13
Figure 5. Public data view sequence diagram.	13
Figure 6. Tournament sequence diagram.....	14
Figure 7. Stage sequence diagram.....	14
Figure 8. Match sequence diagram.	15
Figure 9. Tournament management application structure.	16
Figure 10. Tournament management application classes and relationships.....	17
Figure 11. Entity Relationship diagram for the database.	19
Figure 12. Bottom navigation bar.....	21
Figure 13. Sign in screen.	21
Figure 14. Password reset request form.....	22
Figure 15. Password reset form.....	22
Figure 16. Sign up form.....	23
Figure 17. Profile screen.	24
Figure 18. Change user information (left) and change password (right) forms...	25
Figure 19. Tournament list screen.	26
Figure 20. New tournament form.....	27
Figure 21. Tournament details screen.....	28
Figure 22. Edit tournament form.....	29
Figure 23. New stage form: stage information.....	30
Figure 24. New stage form: common stage configuration.....	30
Figure 25. New stage form: configuration for round-robin (left) and single-elimination (right) stages.	31
Figure 26. Edit stage order form.....	31
Figure 27. Stage details: stage information.....	32
Figure 28. Stage details: table results of round-robin stages.....	33

Figure 29. Stage details: match list of round-robin stages.....	33
Figure 30. Stage details: match list and overall results of single-elimination stages.	34
Figure 31. Round-robin (left) and single-elimination (right) match details.....	34
Figure 32. Round-robin (left) and single-elimination (right) edit team name form.	35
Figure 33. Round-robin (left) and single-elimination (right) edit match score form.	35
Figure 34. Edit match information form.....	36
Figure 35. Authentication server folder structure.....	37
Figure 36. Tournament data processing server folder structure.....	43
Figure 37. Client folder structure.....	55
Table 1. Application requirements.....	10
Table 2. Test results.....	62
Table 3. Test results (continue).....	63
Code Snippet 1. Environment variables in authentication server.....	38
Code Snippet 2. Flask configuration in app.py.....	38
Code Snippet 3. Psycopg2 database connection variables.....	39
Code Snippet 4. send_email.py file.....	39
Code Snippet 5. run.py file.....	40
Code Snippet 6. Example of the structure of authentication server endpoints. .	41
Code Snippet 7. launchSettings.json file.....	44
Code Snippet 8. appsettings.json file.....	44
Code Snippet 9. Program.cs file.....	45
Code Snippet 10. TokenValidation.cs file.....	46
Code Snippet 11. AppDbContext class.....	47
Code Snippet 12. Controller class example.....	49

Code Snippet 13. app.json configuration file.	56
Code Snippet 14. Example of using environment variables in the code.....	56
Code Snippet 15. App.tsx file.....	58
Code Snippet 16. Main.tsx file.....	59
Code Snippet 17. TournamentStack.tsx file.	60
Code Snippet 18. updateTournament function.	61

LIST OF ABBREVIATIONS

CLI	Command Line Interface
GUI	Graphical User Interface
SQL	Structured Query Language
API	Application Programming Interface
EF	Entity Framework
JSON	JavaScript Object Notation
JWT	JSON Web Token
MVCC	Multiversion Concurrency Control
WSGI	Web Server Gateway Interface
SMTP	Simple Mail Transfer Protocol
CORS	Cross-Origin Resource Sharing
REST	Representational State Transfer
LINQ	Language-Integrated Query
RFC	Request for Comments
IDE	Integrated Development Environment
URL	Uniform Resource Locator
TDD	Test-Driven Development

1 INTRODUCTION

This section explains why the tournament management application should be developed for mobile users, and what the main objectives are that the thesis project will achieve.

In professional tournaments, the tournament data and scoring system are specifically designed based on what sports the participants play, number of participants in the tournament and consideration from the organization. However, in semi-professional and amateur tournaments, such a complex system is not required. Instead, a quicker and more convenient solution is needed, since only the basic data is usually collected, and common formats are shared across different tournaments. Therefore, a generalized and customized tournament management application should be implemented.

The tournament management application needs to be handy so that the users do not need to stay next to a computer or hold a laptop. Moreover, accessing an installed application is much faster than opening a browser and searching for the application on a website. Thus, the best choice is to develop a cross-platform mobile application.

The application should provide a convenient and straightforward way to help users view and modify tournament data, as well as stages and matches inside the tournament. The tournament managers should not manually calculate and update the overall result of the stage. Instead, it will be automatically processed in the server based on the provided match results.

The supported stage formats that should be in the application are listed below:

- Single elimination: The participants are eliminated immediately if they lose any match in the stage. The single elimination format is valuable when the number of participants is large, time is short, and the number of locations is limited. [1]

- Round robin: This stage format consists of all participants playing against each other an equal number of times. When the number of participants is small and games are played quickly, this type of format is effective for a one-day tournament. When there are more participants and the games take longer to complete, then a round robin schedule is best suited for league play. [1]

2 RELEVANT TECHNOLOGIES

In this section, the main relevant technologies for the database, the servers and the client are described.

2.1 PostgreSQL

PostgreSQL, originally developed in 1986, is a modern and advanced relational database system which supports both non-relational and relational data types. PostgreSQL has great performance and scalability, which makes it extremely efficient when running deep, extensive data analysis across multiple data types. It also manages concurrency efficiently through its use of MVCC. Nowadays, many programming languages such as Python, JavaScript, C/C++, C#, etc. offer mature support for PostgreSQL. Moreover, PostgreSQL is a sustainable production database since it ensures high availability for both clients and developers. Last but not least, PostgreSQL is a free and open-source database. With all these benefits, PostgreSQL is one of the most compliant, stable, and mature relational databases available today. [2]

PostgreSQL databases can be accessed using a CLI tool called “psql” or a GUI tool called “pgAdmin”. PostgreSQL v16.1 and pgAdmin 4 v8.1 are used in this project.

2.2 Python & Relevant Modules

Python is one of the most popular programming languages in the world because it is a general-purpose and beginner-friendliness language. It can be used for many different tasks such as web development, data analytics, machine learning, automation, and the syntax is easy to learn. Moreover, it is open source, has a huge number of third-party modules and libraries, and has a large and active community. [3]

Flask is a lightweight WSGI web application framework supports minimal RESTful Web API. It is designed to make getting started quick and easy, with the ability to

scale up to complex applications. It began as a simple wrapper around Werkzeug and Jinja and has become one of the most popular Python web application frameworks. [4]

In this project, two support modules for Flask, Flask-Mail and Flask-CORS, were used. Flask-Mail was used for creating and sending auto-generated email messages by SMTP protocol, while Flask-CORS is used for handling CORS, which means it allows or blocks requests from some or all origins (including protocols, domains and ports).

Psycopg is the most popular PostgreSQL database adapter for Python. It completely implements the Python DB API 2.0 specification and provides thread safety (several threads can share the same connection). It was designed for heavily multi-threaded applications that create and destroy lots of cursors and make many concurrent INSERTs or UPDATEs. Psycopg 2 is both Unicode and Python 3 friendly [5]. In this project, this module helps executing database operations by writing dynamic SQL query strings in the code.

Argon2 is a secure password hashing algorithm designed to have both a configurable runtime as well as memory consumption. Argon2 has three variants. Argon2d can resist time–memory trade-offs, while Argon2i can resist side-channel attacks. Argon2id is the combination of the two variants above, so it is the best variant in practice. In 2015, Argon2 was the winner of the Password Hashing Competition [6]. Python has an Argon2 module called argon2-cffi.

2.3 C# / .NET (Core) & Relevant Frameworks

C# is a modern, innovative, open-source, cross-platform object-oriented programming language. It is the most popular language for .NET development. [7]

.NET (Core), developed and maintained by Microsoft, is a free, cross-platform, open-source developer platform for building many kinds of applications. .NET delivers productivity, performance, security, and reliability. Similar to Java, .NET

has its own garbage collector. Furthermore, it is type-safe and memory-safe, offers concurrency, includes a large set of libraries and has been optimized for performance on multiple operating systems and chip architectures [8]. .NET is faster than many frameworks such as Node.js and Java Servlet. (Microsoft, n.d., online) [7]

ASP.NET Core Web API is one of .NET Core templates which helps create RESTful web API. Latest .NET versions support two methods of creating the API: using ASP.NET Core Controller classes or minimal APIs. For medium and large applications, using controller classes is preferred.

Entity Framework is a modern object-relation mapper that lets developers build a clean, portable, and high-level data access layer with .NET across a variety of databases through plug-in libraries called database providers. It supports LINQ queries, change tracking, updates, and schema migrations. [9]

EF Core is a lightweight, extensible, open source and cross-platform version of the Entity Framework [10]. Since PostgreSQL is used in this project, Npgsql.EntityFrameworkCore.PostgreSQL database provider is also installed.

2.4 JWT

JWT is an open standard, which is defined in RFC 7519. They securely represent claims between two parties. Claims are typically used to represent an identity and its associations, so JWT is often used for authentication and authorization [11].

JWT is made up of three parts:

- The Header contains two pieces of information: token type (typ) and algorithm used (alg) [11]
- The Payload contains the claims, including registered and custom claims. Custom claims can be anything, but registered claims are standardized although they are not mandatory. Examples of registered claims are iss (issuer), exp (expiration time), sub (subject), and aud (audience). [11]

2.5.1 Expo

Expo is an open-source platform for making universal native apps for Android, iOS, and the web with JavaScript and React. It is a full ecosystem of tools that helps developers write, build, update, submit, and monitor mobile apps. [14]

Expo Go is a free, open-source sandbox for quick experimentation with building native Android and iOS apps. It is available for download on both iOS App Store and Android Play Store. [15]

2.5.2 JavaScript & TypeScript

JavaScript is the world's most popular programming language for web applications. It defines the behavior of web pages. The latest version of JavaScript is ES6 (2015), and it is still annually updated until now. [16]

TypeScript is a strongly typed programming language that builds on JavaScript. It can catch mistakes (errors and warnings) and display them in the editor. It also provides the ability to define primitive (“string” and “number”, for example) types and custom types (by using “type” and “interface” keywords) for variables. [17]

2.5.3 React Native Libraries

The React Navigation library helps developers implementing routing and navigation for Expo and React Native applications [18]. The library provides different types of navigators such as stack, bottom tabs, drawer, ... and supports deep linking.

React Native Paper, a third-party React Native library, is a collection of customizable and production-ready components for React Native, following Google’s Material Design guidelines [19]. It saves developers’ time in creating common complex components and a beautiful GUI.

Formik is another third-party library. It helps managing forms in React and React Native become easier by getting values in and out of the form state, handling input validation and error messages, and handling form submission [20]. Formik may be integrated with a small library called Yup. Yup is a collection of validation rules used for validating user input values.

In addition to those libraries above, other small libraries are also installed. In this project, the React Native Async Storage is used for storing and retrieving JWT token; the React Native Datetime Picker has interactive and customized datetime picker components for handling datetime states.

3 APPLICATION DESCRIPTION

The application is meant to help tournament managers build and run their tournaments. Also, anyone, including guest users, should be able to view the progress of the tournaments if managers share their tournaments publicly. The application should consist of four main parts:

A PostgreSQL database was used. The database has two schemas for two different purposes. One schema, named “auth”, stores user information and credentials, while the other one, named “data”, stores tournament, stage and match information.

The authentication server was written in Python and uses Flask library to create minimal REST API services. It also uses psycopg2 library to connect to the “auth” schema in the database.

The tournament data server runs a controller-based REST API using the ASP.NET Core Web API template and was written in C#. EF Core and Npgsql.EntityFrameworkCore.PostgreSQL libraries were installed on the server to connect to the “data” schema in the database.

The client was implemented using React Native and TypeScript. Users can interact with tournaments based on their role. They can also create and manage their account information.

The JWT technology was used across the client and the two servers for secure authentication and authorization. To store passwords securely, Argon2 password-hashing function was applied.

3.1 Quality function deployment

The table below shows the list of requirements implemented in the application. The priority scale has three levels. Priority 1 means must-have, 2 means should-have and 3 means nice-to-have requirements.

Table 1. Application requirements.

Reference	Description	Priority
F1	Sign in/Sign out & Sign up.	1
F2	Change & reset password.	1
F3	CRUD Tournament List (name, place, date, ...).	1
F4	CRUD Stage List (name, place, date, format, order, ...).	1
F5	While creating stage, matches are generated based on stage format and other configurations.	1
F6	Teams can be in one group or divided into many groups in all stage formats.	1
F7	Update match information & team names & match score.	1
F8	Match winner can be determined manually or automatically by final total score.	1
F9	In Single elimination format, the winner will be automatically advanced to next round.	1
F10	In Round robin format, the scores are calculated and ranked by predefined winning, drawing and losing points, as well as tiebreaking criteria.	1
F11	In Single elimination format, team names in the bracket are initially seeded in the first round (e.g. 'Team1' vs 'Team4', 'Team2' vs 'Team3').	2
F12	In Round robin format, users can add custom tiebreaking criteria. They can be ordered and sorted ascending or descending.	2
F13	Change user information.	2
F14	Help, contact channels and app info.	3
F15	In Single elimination format, there is an option for including third-place match(es).	3

3.2 Use-case Diagram

As previously mentioned, guest users can only view public tournaments and their content inside. To manage their own tournaments, they must sign up, if necessary, and sign in. In case of forgetting the password, they can reset it via their registered email addresses.

After signing in, they can change their user information and password, or delete user account. Some information has a restricted period, which means after changing that information, users must wait for a period of time to change it again. Moreover, users can create, edit and delete tournament, stage and match data. They can view both public and their own tournaments. If users want to sign in to another account, they must sign out first.

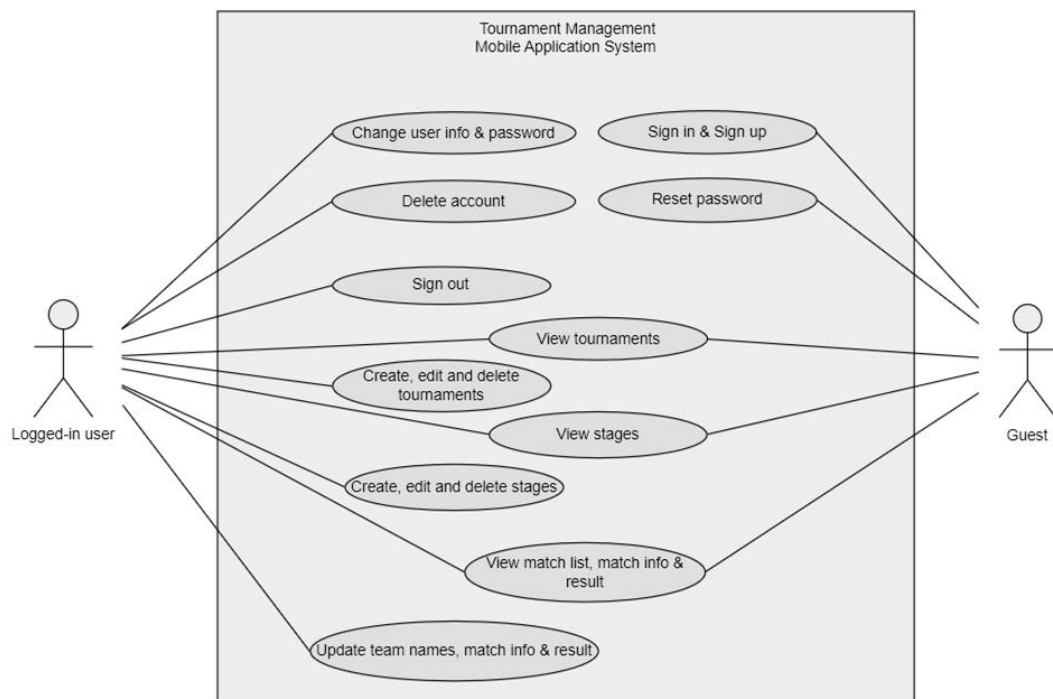


Figure 2. Tournament management system use cases.

3.3 Sequence Diagram

Some important activities are discussed using the application by the sequence diagrams below. It should be noted that these diagrams only show the successful

operations. If any error takes place during an operation, an error message will be sent back to the place where the first step of the operation is done.

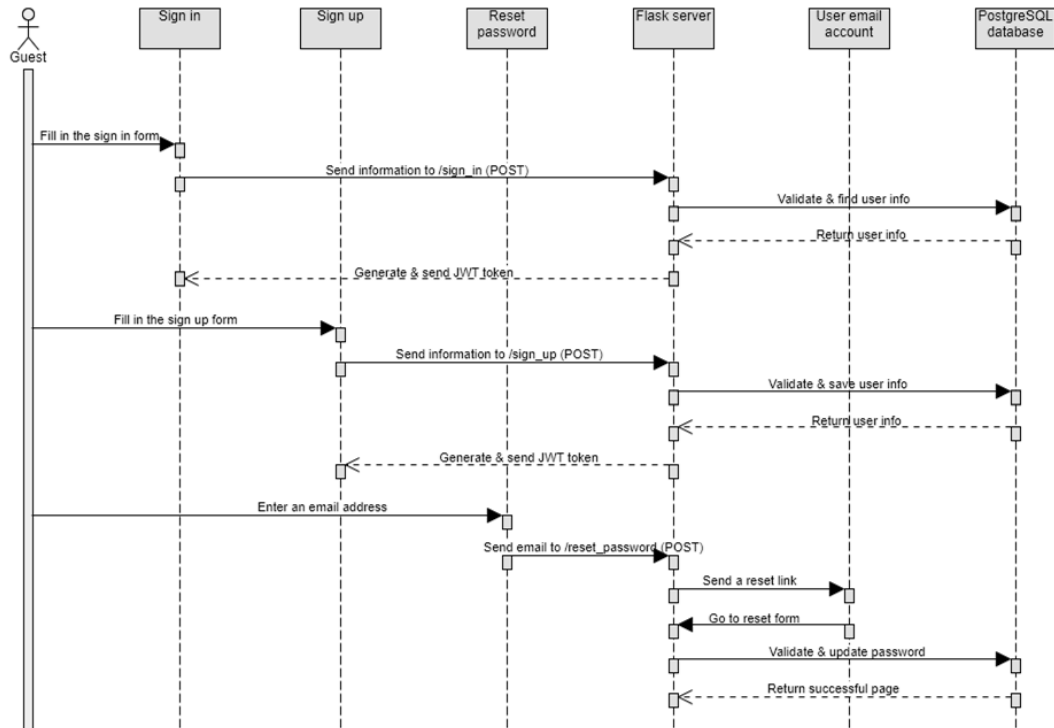


Figure 3. Authentication sequence diagram.

The first important activity is to get a user account and/or sign in. The process of signing in and signing up are quite similar: the guest sends the corresponding information to the Flask server, then the server will process the request data and send a JWT token back to the guest, and the guest will become a logged-in user. The only main difference is the sign-up function finds an existing user in the database, while the sign-in function saves processed data to the database.

If the users forget their password, they can request for a password reset link via email. When they receive and click the link, a form will be shown for them to submit their new password.

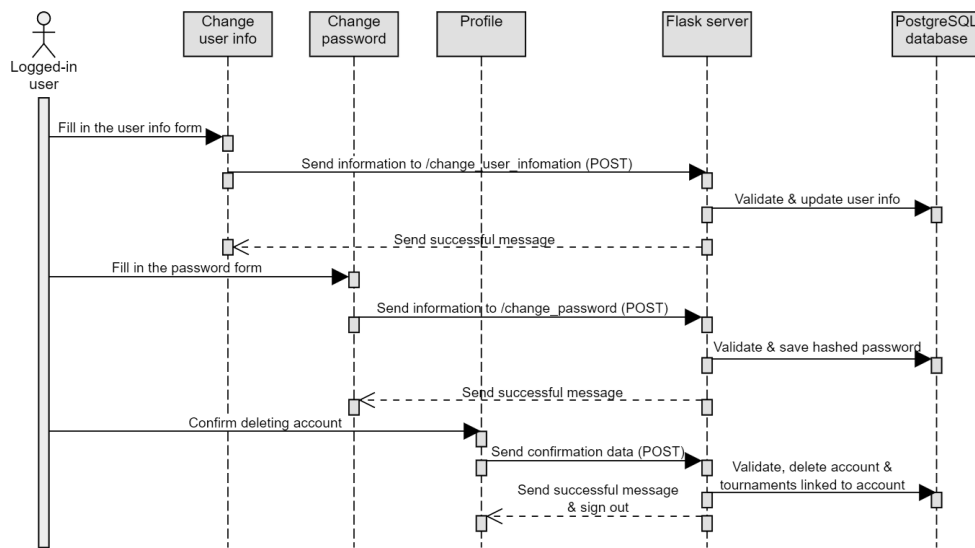


Figure 4. User information & password change sequence diagram.

Logged-in users can manage their account by changing user information and password, as well as deleting user accounts. The process is quite similar to signing in and signing up above, but only the successful status and message are returned.

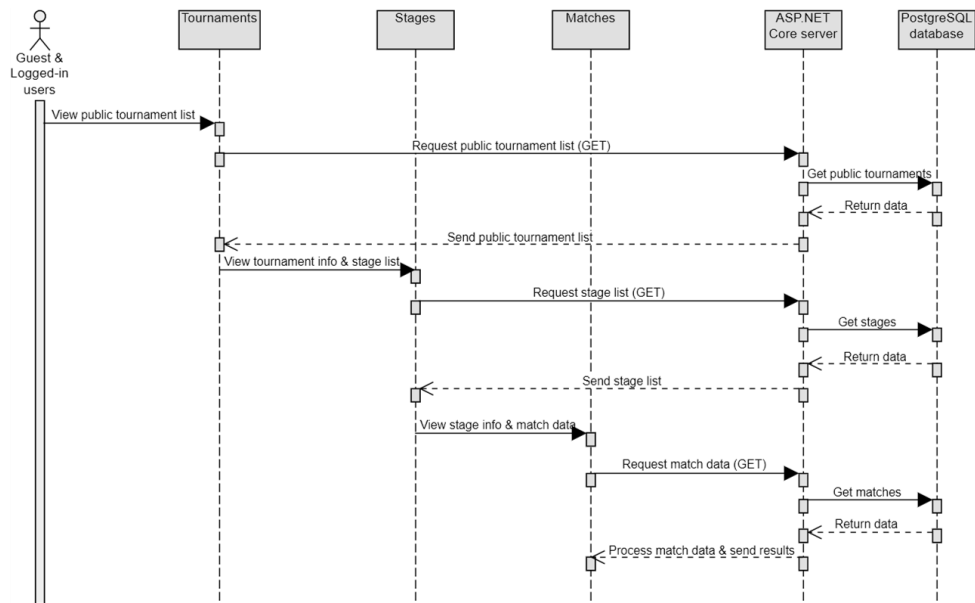


Figure 5. Public data view sequence diagram.

Whether users are logging in or not, they can always see a list of public tournaments and their content inside. Of course, they can only make GET requests for such information and cannot alter them.

The remaining diagrams describe the main tournament management activities for logged-in users. The process of fetching tournament, stage and match data are almost the same: the GET requests are sent to the ASP.NET Core server, then it collects corresponding data and returns the data as a JSON object to the client.

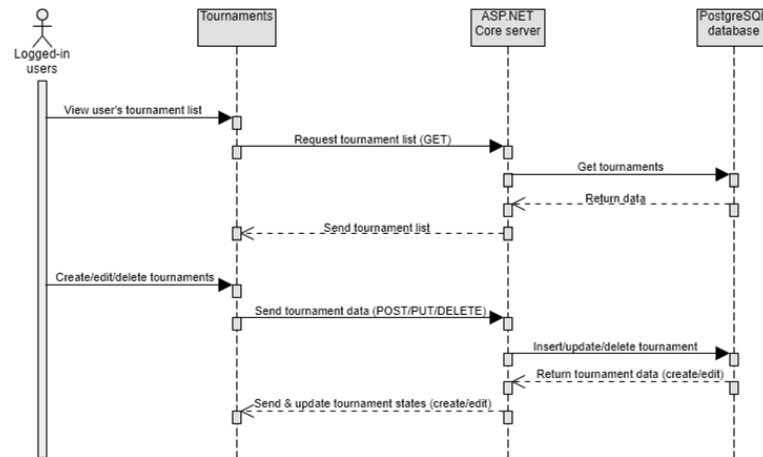


Figure 6. Tournament sequence diagram.

Users can also make POST, PUT and DELETE requests for creating, editing and deleting tournaments. The tournament data is returned to the client after the creation and edition, but nothing is returned after the deletion.

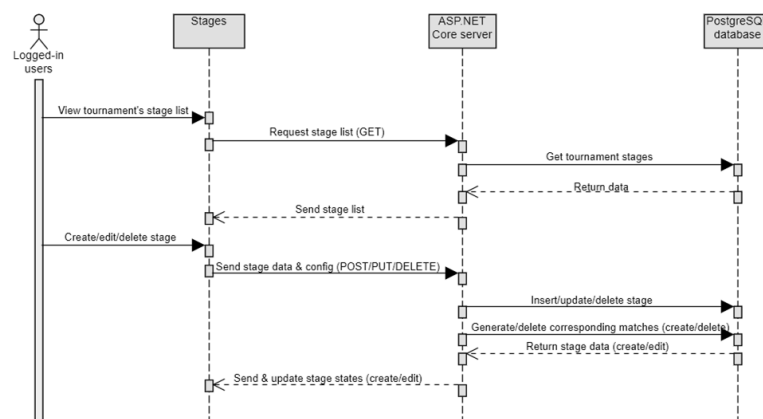


Figure 7. Stage sequence diagram.

The stage operations are almost the same as the tournament operations. In the stage creation and deletion, after inserting or deleting a stage, the corresponding matches will also be generated or deleted in another table in the database.

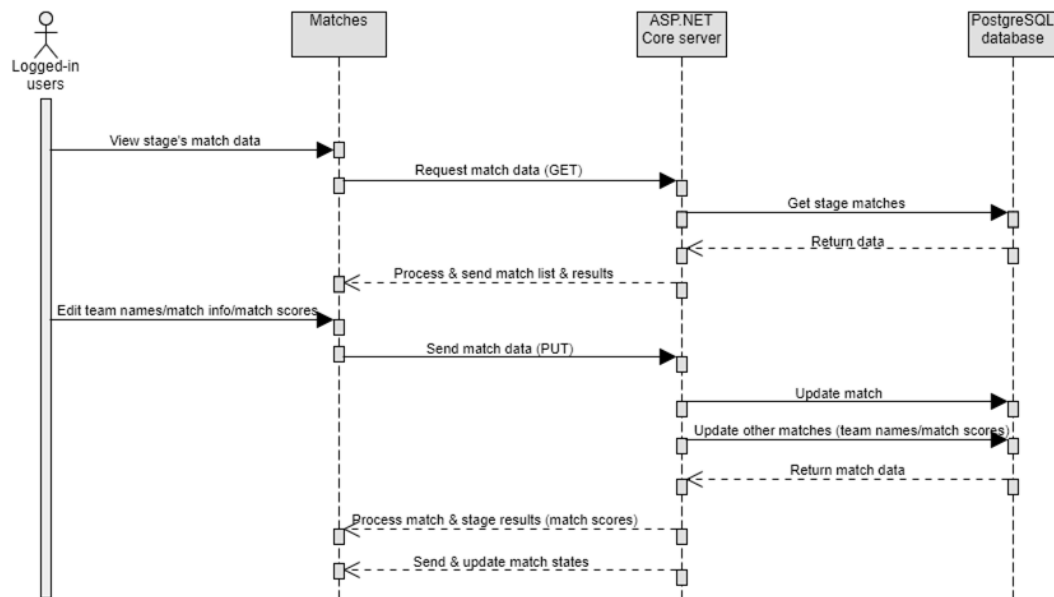


Figure 8. Match sequence diagram.

Regardless of stage formats, only PUT requests are allowed to alter match data. The requests are made to change team names, match information and scores. In some cases (editing team names and match scores), data in other matches in the same stage may also be changed. After changing match data in the database, the overall result of the stage will be calculated and returned to the client along with the new match data.

3.4 Architectural Diagram

As described in the previous section, the software architecture is made up of four parts: a PostgreSQL database, the backend with two servers (Python/Flask and C#/ASP.NET Core) and one React Native & TypeScript client.

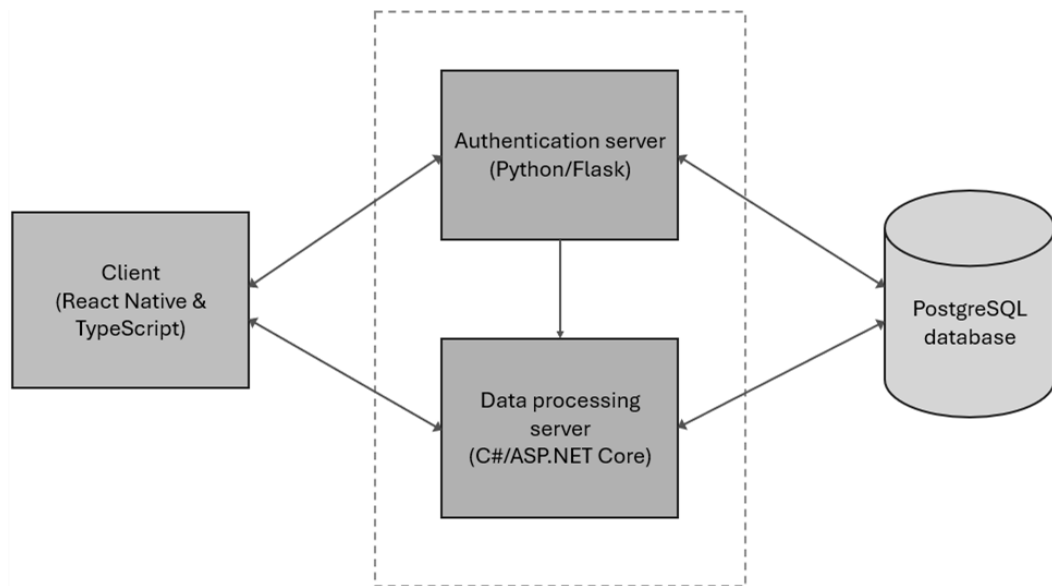


Figure 9. Tournament management application structure.

The two servers independently communicate with the database to get information and perform data modifications. Similarly, they independently connect to the client, and they are responsible for receiving and sending JSON data. There is a case where the authentication server communicates with the data processing server, but it will be discussed later.

3.5 Class Diagram

This class diagram below represents the data models that is implemented in the database as well as in the two servers. The four main classes are “User”, “Tournament”, “Stage” and “Match”.

Each user can have multiple tournaments. If a user account is deleted, all tournaments and their content connected to that account will also be deleted. This composition relationship is also applied to the relationship between “Tournament” and “Stage” classes, and between “Stage” and “Match” classes. The exception is a stage must have at least one match.

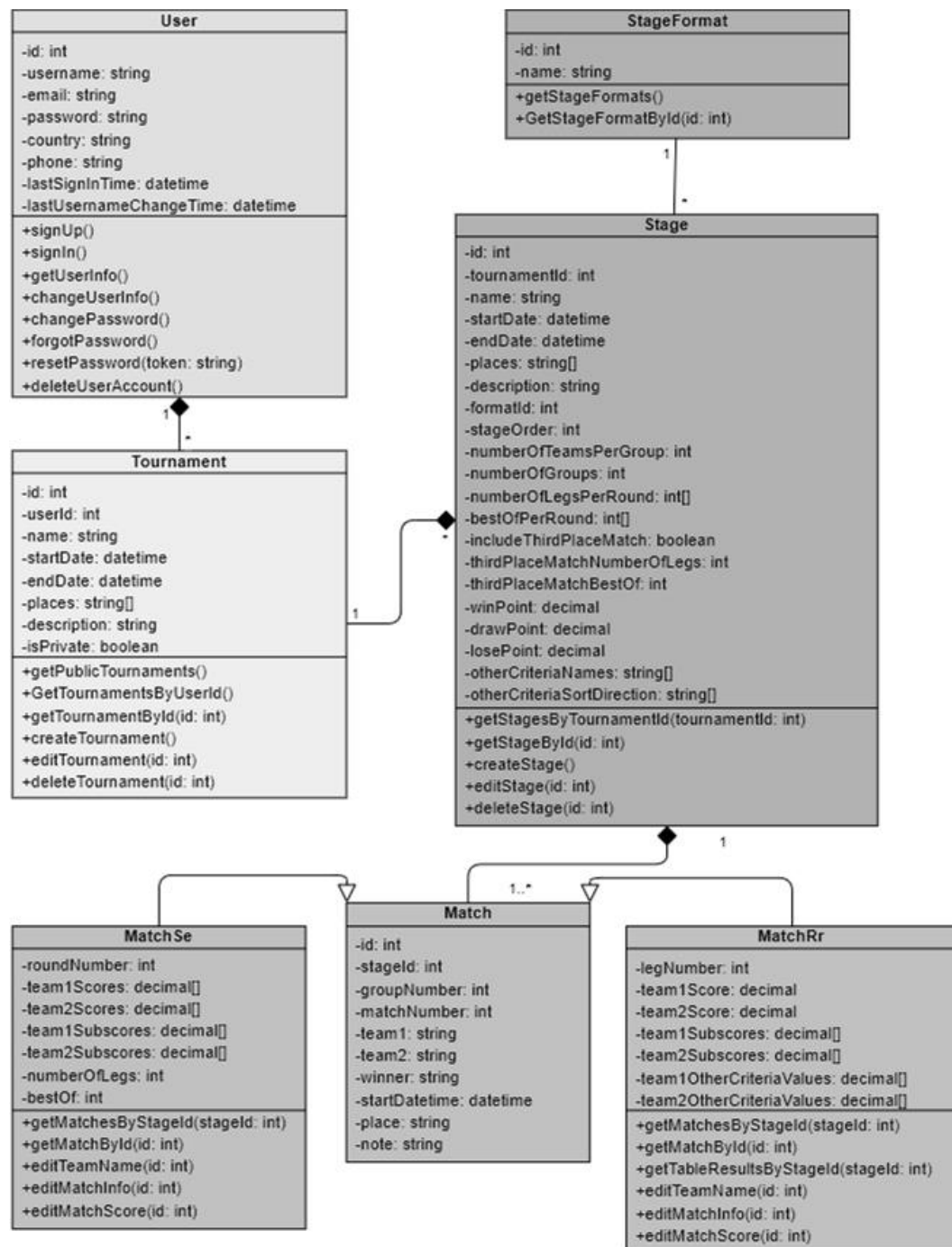


Figure 10. Tournament management application classes and relationships.

In the “User” class, the username, email and password attributes are sign-in credentials. Other attributes are informative or used for other purposes. All user data management operations are available in this class.

All attributes and operations for handling tournament data itself are in the “Tournament” class. The “userId” attribute is the foreign key of the id attribute in the “User” class.

In the “Stage” class, the “tournamentId” attribute is the foreign key of the id attribute in the “Tournament” class. The stage format is defined by the “formatId” attribute and the many-to-one associated relationship with the “StageFormat” class. Beside the attributes which describe the stage information, there are several attributes which store the stage configuration depending which stage format is chosen. In the diagram below, the attributes from “stageOrder” to “bestOfPerRound” are used for all stage formats. The three third-place match attributes are specifically for single-elimination format, and the last five attributes are for round-robin format. Like “Tournament” class, all stage data management are in the “Stage” class.

The basic information of every match is listed in the “Match” class. The “staged” attribute is the foreign key of the id attribute in the “Stage” class. From the “Match” class, two more classes are inherited to implement specific attributes and operations based on each stage format. The two classes are “MatchSe”, represents a single-elimination match, and “MatchRr”, represents a round-robin match.

4 DATABASE DESIGN

As mentioned in Chapter 3, the PostgreSQL database has two schemas: “auth” and “data”. The “auth” schema contains one table called “users”, and the “data” schema includes the remaining tables. All tables have the “id” column as the primary key.

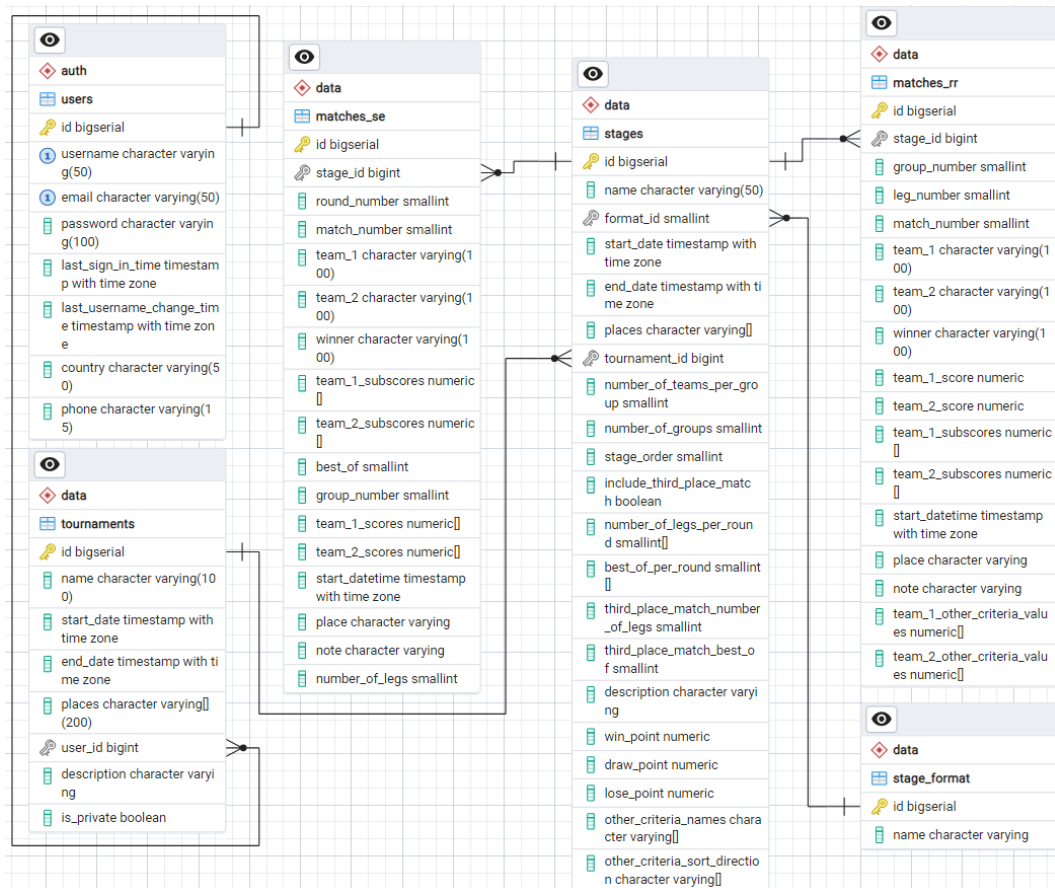


Figure 11. Entity Relationship diagram for the database.

The detailed description of every table is given next **auth.users** table store user credentials (username, email and password) and other information. Among informative columns, the “last_username_change_time” column is used for checking if changing username is allowed; the “last_sign_in_time” column might be used for deleting any account that has signed out for a long time.

The **data.tournaments** table stores the information about the tournament itself. It has a foreign key called “user_id” which connects to the “auth.users” primary key. There is also the “is_private” column for determining the visibility of the tournament in the application.

The **data.stages** table stores the information of the stage itself. It also stores the stage configuration which describes the stage structure, and how the matches should be generated and maintained in the stage. The table has two foreign keys, “tournament_id” and “format_id”. The “tournament_id” key links to the “data.tournaments” table, and the “format_id” key links to the “data.stage_format” table.

The **data.stage_format** table includes the full name of the “format_id” value in the “data.stages” table.

The **data.matches_se** is a collection of matches run in single-elimination format. Each match record contains metadata, editable match information and match results. The “stage_id” foreign key links to the primary key of the “data.stages” table.

The **data.matches_rr** is similar to the “data.matches_se” table, this table is a collection of matches, but they belong to the round-robin stages. It also has the “stage_id” foreign key, contains metadata, editable match information and match results. However, some metadata and match result columns are different from the columns in the “data.matches_se” table.

5 GUI DESIGN

The client application is made up of two tabs which can be navigated on the bottom navigation bar. These two tabs are “Tournaments” and “Profile”.



Figure 12. Bottom navigation bar.

In the “Profile” tab, if the user does not sign in, the sign in screen below will be shown. The screen includes a sign in form, a link to the forgot-password screen and a “Create account” button that navigates to the signup form.

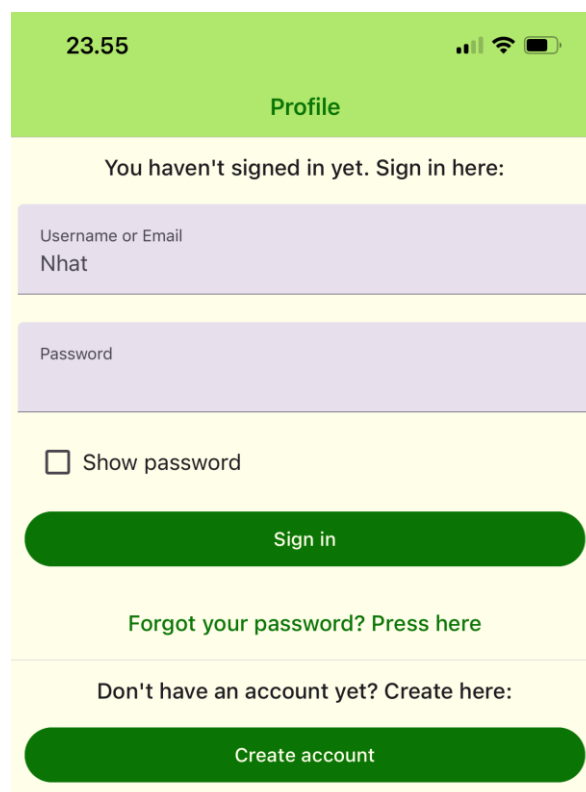
A mobile app screen titled "Profile" with a green header. The status bar at the top shows the time 23.55, signal strength, Wi-Fi, and battery icons. The main content area has a light yellow background. It starts with the text "You haven't signed in yet. Sign in here:". Below this are two input fields: "Username or Email" with the text "Nhat" and "Password". There is a checkbox labeled "Show password" which is currently unchecked. Below the input fields is a large green rounded button labeled "Sign in". Underneath the button is a green link that says "Forgot your password? Press here". At the bottom, there is another section with the text "Don't have an account yet? Create here:" and a large green rounded button labeled "Create account".

Figure 13. Sign in screen.

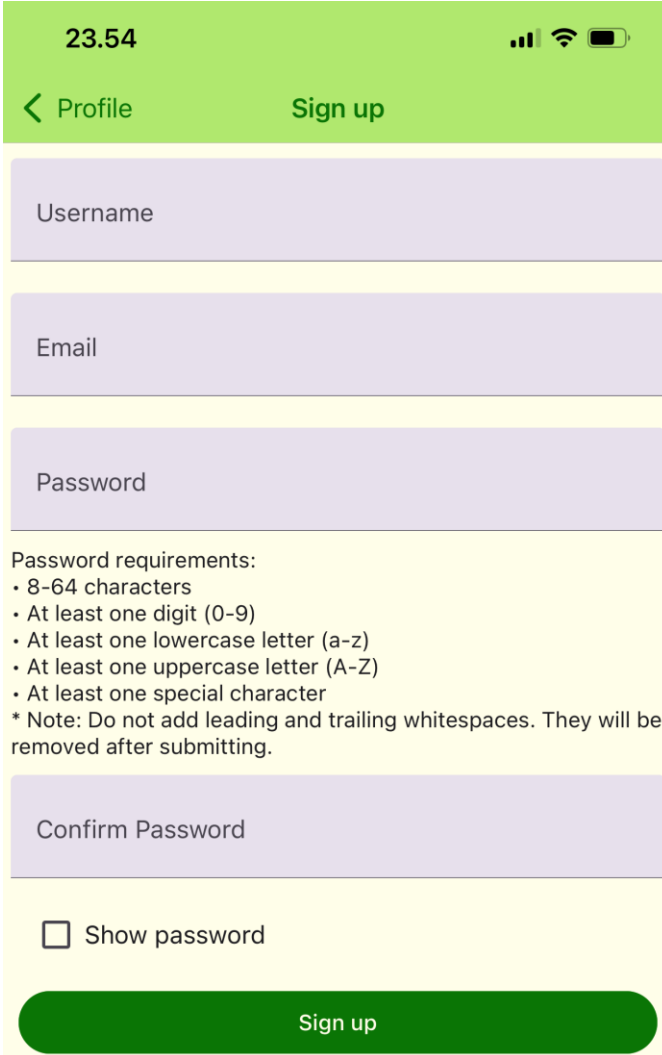
The forgot-password screen contains only an email input field and a button to submit the password reset request.

Figure 14. Password reset request form.

After requesting, the reset password action is done on a normal web browser when users click the link sent to their email address.

Figure 15. Password reset form.

The signup form contains necessary input fields for creating an account. A verbal password requirement list is also displayed on the screen.



The image shows a mobile application sign-up screen. At the top, the status bar displays the time 23:54, signal strength, Wi-Fi, and battery icons. Below the status bar is a green header with a back arrow and the text 'Profile' on the left, and 'Sign up' on the right. The main content area has a light yellow background and contains three purple input fields for 'Username', 'Email', and 'Password'. Below the password field, there is a list of password requirements: 8-64 characters, at least one digit (0-9), at least one lowercase letter (a-z), at least one uppercase letter (A-Z), and at least one special character. A note states: '* Note: Do not add leading and trailing whitespaces. They will be removed after submitting.' Below the requirements is a purple input field for 'Confirm Password'. At the bottom, there is a checkbox labeled 'Show password' and a green rounded button labeled 'Sign up'.

Figure 16. Sign up form.

After signing in, the users see the actual profile screen. They can view their detailed account information, modify user information, view application information, sign out or delete their account.

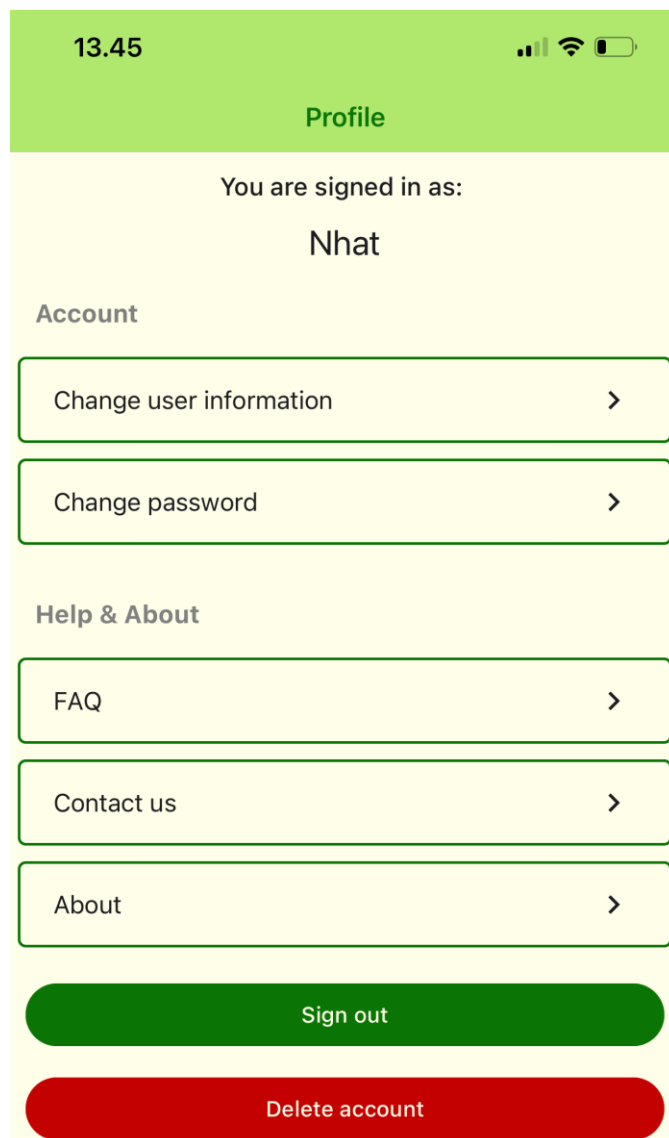


Figure 17. Profile screen.

The “Change user password” form lets users change their username and other information that is not available when signing up. The “Change password” form requires users to provide both current password and new password, as well as new password confirmation, for a secure password change.

The image displays two mobile application screens side-by-side. Both screens have a green header with the time '13.47' and status icons for signal, Wi-Fi, and battery. The left screen is titled 'Change user information' and contains the following fields: User ID (1), Username (Nhat), Email (minhnhat091101@gmail.com), Country (Finland), Dial code (+358), and Phone (3141592654). A green 'Change' button is at the bottom. The right screen is titled 'Change password' and contains fields for Current Password, New Password, and Confirm New Password. It also has a 'Show password' checkbox and a green 'Change' button. Password requirements are listed at the bottom right of the second screen.

Change user information (left) form fields:

- User ID: 1
- Username: Nhat
- Email: minhnhat091101@gmail.com
- Country: Finland
- Dial code: +358
- Phone (no leading zero): 3141592654

Change password (right) form fields:

- Current Password
- New Password
- Confirm New Password
- Show password

Password requirements:

- 8-64 characters
- At least one digit (0-9)
- At least one lowercase letter (a-z)
- At least one uppercase letter (A-Z)
- At least one special character

* Note: Do not add leading and trailing whitespaces. They will be removed after submitting.

Figure 18. Change user information (left) and change password (right) forms.

There are three other screens that are navigated from the “Help & About” section on the “Profile” screen. However, they only provide static application information, so they will not be shown here.

In the “Tournaments” tab, the first screen the users see is the tournament list screen. It has two parts: the upper part shows the user’s tournament list, while the lower part shows the public tournaments of all users. Each part has a search bar for filtering the corresponding list based on the tournament name. If the users do not sign in, they cannot see the upper part.

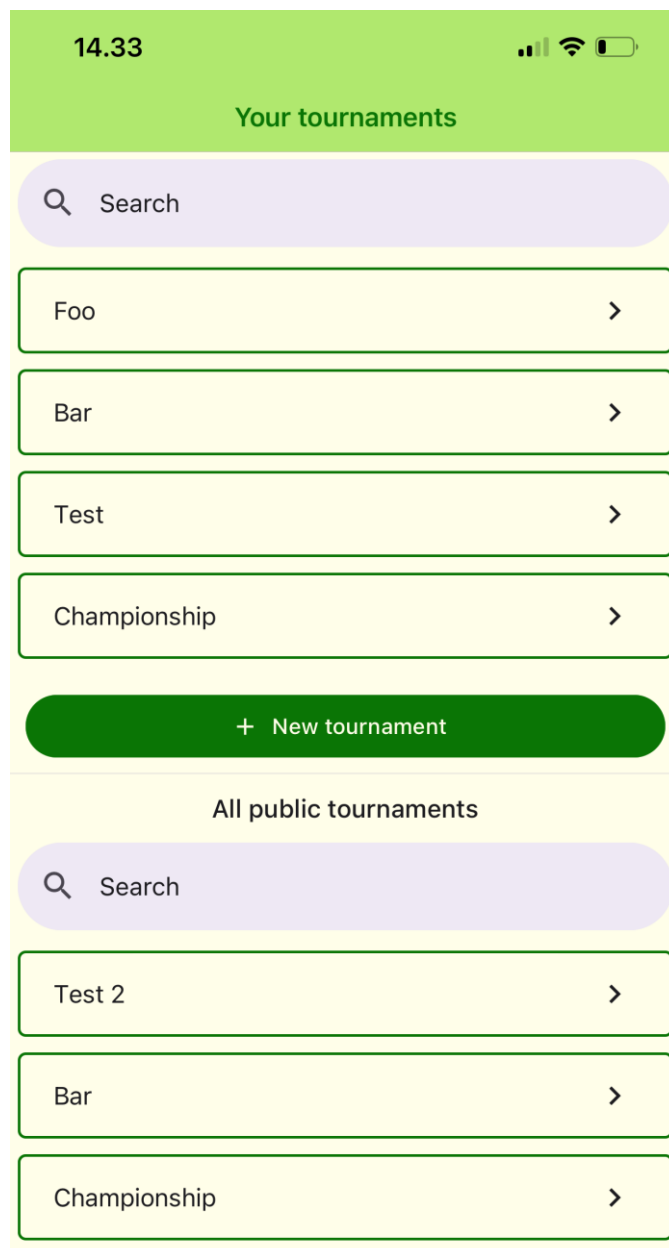


Figure 19. Tournament list screen.

The “New tournament” button navigates to a form to create a new tournament. Guest users cannot see that button.

23.57

< Your tournaments New Tournament

Name
New tournament

Start date
22 Apr 2024

End date
22 Apr 2024

Places
Add Places

Description

Set as private tournament

Create tournament

Figure 20. New tournament form.

One important note should be made: every button in the below figures below that is used for modifying the public tournaments and their stages and matches will not be shown to guest users (only the information is available for viewing).

Pressing any tournament will make the application navigate to the tournament details screen. It displays the tournament information and the stage list. In the tournament information section, the “Edit” and “Delete” buttons are used to edit the tournament information and delete the whole tournament. The stage list section contains pressable list of stages and two buttons for changing stage order and add a new stage. The “Change stage order” appears when the list has two or more stages.

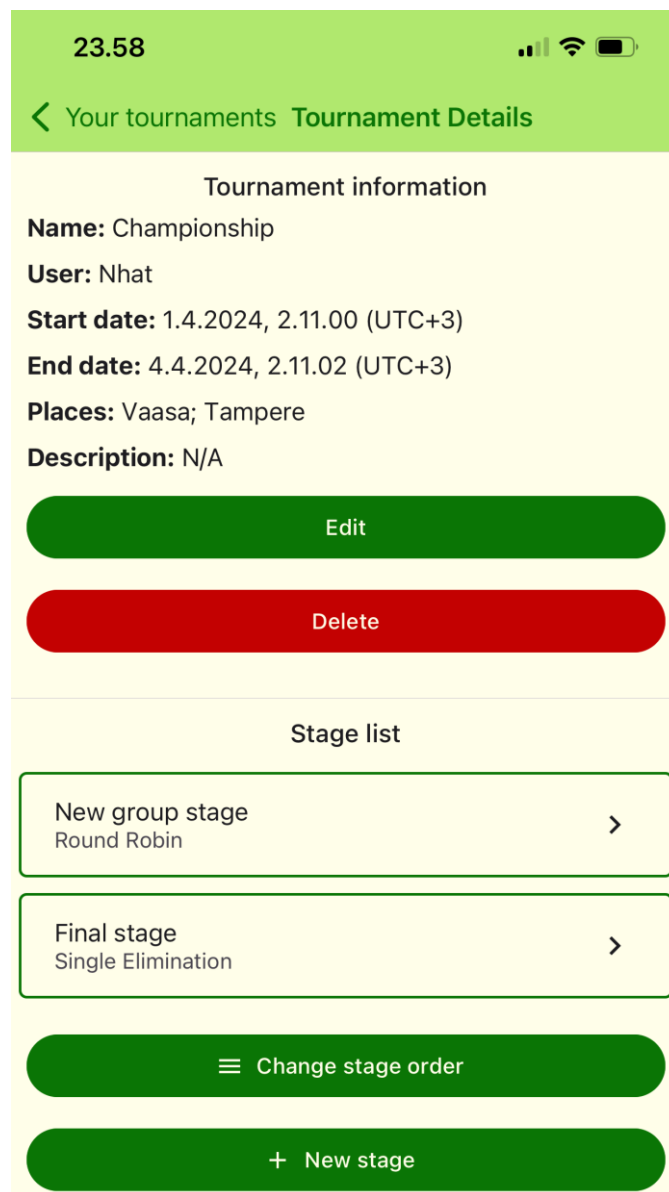


Figure 21. Tournament details screen.

To edit the tournament information, press the "Edit" button. The form is similar to the "New tournament" form, but current data will be in the input fields.

23.58

< Tournament Details Edit Tournament

Name
Championship

Start date
1 Apr 2024

End date
4 Apr 2024

Places

Place 1
Vaasa Remove

Place 2
Tampere Remove

Add Places

Description

Set as private tournament

Update

Figure 22. Edit tournament form.

The “New stage” button leads users to the stage creation form. The first section is the stage information form, which is similar to the “New tournament” form.

23.59

< Tournament Details New Stage

Stage information

Name
New stage

Start date
22 Apr 2024

End date
22 Apr 2024

Places
Add Places

Description

Figure 23. New stage form: stage information.

The second section is the stage configuration. In the second section, the first input fields are applied to all stage formats.

Stage configuration

Stage format
Single Elimination

Number of teams / group
4

Number of groups
1

Stage order
3

Figure 24. New stage form: common stage configuration.

After the first input fields, the other fields are displayed depending on which stage format is chosen.

The figure shows two side-by-side forms for configuring a new stage. The left form is for a round-robin stage, and the right form is for a single-elimination stage.

Left Form (Round-robin configuration):

- Number of legs / round (max. 3): Round 1: 1, Round 2: 1
- Best of / round: Round 1: 1, Round 2: 1
- Include third place match
- Third place match number of legs (max. 3): 1
- Third place match best of: 0
- Buttons: Create stage

Right Form (Single-elimination configuration):

- Number of legs (max. 3): 1
- Best of: 3
- Win Point: 0
- Draw Point: 0
- Lose Point: 0
- Other criteria:
 - 1. Name: Criteria 1, Sort direction: DESC, Remove button
- Buttons: Add Criteria, Create stage

Figure 25. New stage form: configuration for round-robin (left) and single-elimination (right) stages.

The “Change stage order” form is simple, as shown below. The order is changed by editing the number in the fields.

The figure shows a mobile application screen titled "0.02" with a status bar at the top. The screen is titled "Tournament Details Edit Stage Order". It displays a list of stages:

- 1 New group stage
- 2 Final stage

At the bottom of the screen is a green button labeled "Update".

Figure 26. Edit stage order form.

Pressing any stage will make the application navigate to the stage details screen. The screen shows the stage information, match list and overall result of the stage.

Some stage information, the match list and overall result of the stage are shown differently among the stage formats. The “Edit” button navigates the application to the “Edit Stage” form. It is similar to the “Edit Tournament” form, but there is no “Set as private tournament” checkbox.

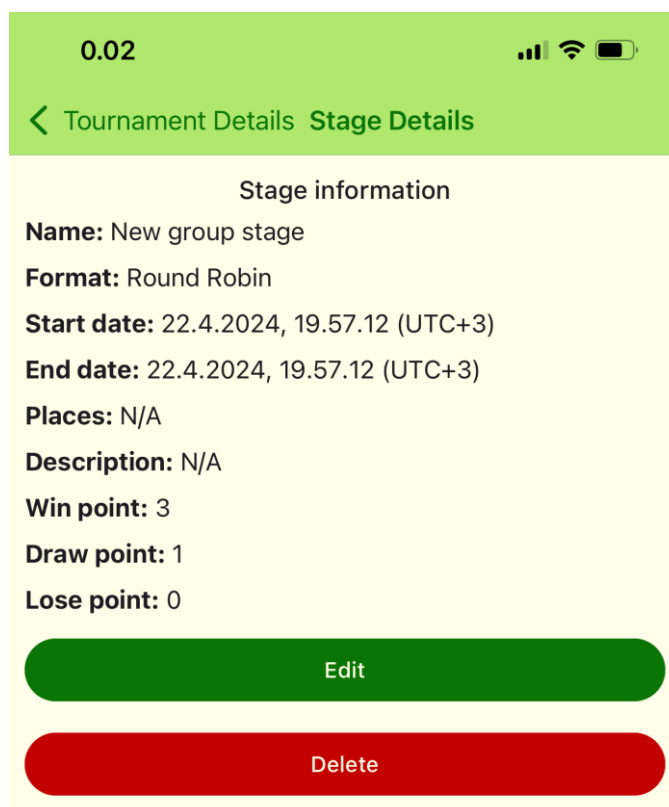


Figure 27. Stage details: stage information.

In round-robin stages, the table results and the match list in a selected group are shown. The columns in the table results are rank, team, points, difference, earned score and possibly other criteria.

Group 1						
Table results						
Rank	Team	Points	Difference	Earned score	IndexA	IndexB
1	G1-Team1	3	+1	2	2	1
2	G1-T3	0	0	0	0	0
3	G1-T4	0	0	0	0	0
4	G1-T5	0	0	0	0	0

Figure 28. Stage details: table results of round-robin stages.

The columns in the match list are leg (number), match (number), team 1 name and score, team 2 name and score

Match List					
Leg	Match	Team 1 Name	Team 1 Score	Team 2 Score	Team 2 Name
1	1	G1-Team1 *	2	1	G1-Team2
1	2	G1-Team1	0	0	G1-T3
1	3	G1-Team1	0	0	G1-T4
1	4	G1-Team1	0	0	G1-T5
1	5	G1-Team1	0	0	G1-T6
1	6	G1-Team1	0	0	G1-T7

Figure 29. Stage details: match list of round-robin stages.

In single-elimination stages, the match list in a selected group and a selected round is shown. The columns in the match list are similar to those in the round-robin stage, except that there is no leg column, and a column called “Winner's Next Round Match” is used to determine the connection between matches across the rounds. (For example, if the match is in round 1 and the column value is 3, the winner will be assigned to match number 3 in round 2).

Matches					
Group 1	Group 2				
Round 1	Round 2	3rd Place			
Match	Team 1 Name	Team 1 Score	Team 2 Score	Team 2 Name	Winner's Next Round Match
1	Team 5	0	4	Team 6 *	1
2	Team 7 *	3	0	Team 8	1

Figure 30. Stage details: match list and overall results of single-elimination stages.

The winner in every match in both types of match list is marked by a trailing star (*). By pressing a match in the match list, users can see the match details, as well as the three edit buttons. In single-elimination format, the team names can only be edited in the first round. Figure 31 below shows the “MatchDetails” screens of both stage formats.

0.04

< Stage Details Match Details

Group: 1
Leg: 1
Match: 1
Best of: 3
Start datetime: N/A
Place: N/A
Note: N/A
Score: G1-T1 0 - 0 G1-T2

Set	1	2	3
G1-T1	0	0	0
G1-T2	0	0	0

Other criteria:

	IndexA	IndexB
G1-T1	0	0
G1-T2	0	0

Edit team name

Edit match information

Edit match scores

0.08

< Stage Details Match Details

Group: 1
Round: 2
Match: 1
Legs: 2
Best of: 3
Start datetime: N/A
Place: N/A
Note: N/A
Total score: Team 1 3 - 2 Team 4

- Leg 1: Team 1 1 - 2 Team 4

Set	1	2	3
Team 1	8	13	9
Team 4	13	7	13

- Leg 2: Team 1 2 - 0 Team 4

Set	1	2	3
Team 1	5	11	0
Team 4	13	13	0

Edit match information

Edit match scores

Figure 31. Round-robin (left) and single-elimination (right) match details.

Here are the “EditTeamNames” screens of both stage formats.

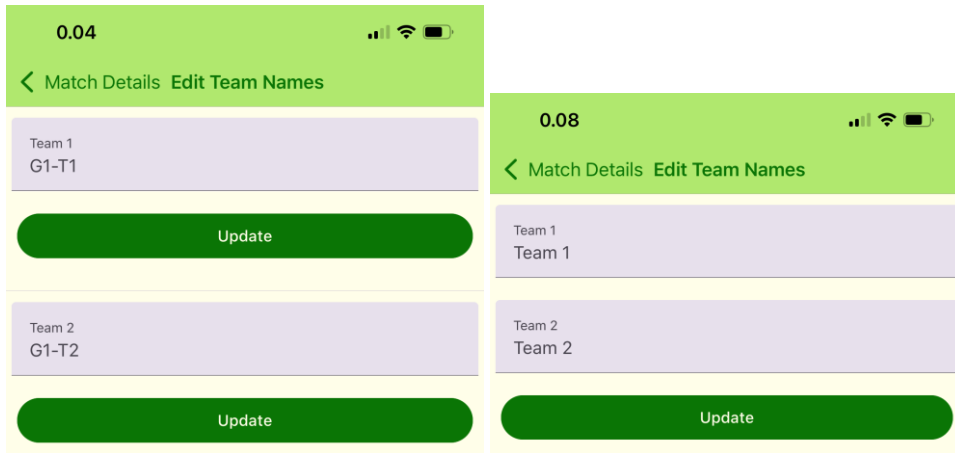


Figure 32. Round-robin (left) and single-elimination (right) edit team name form.

Then, here are the “EditMatchScores” screens of both stage formats.

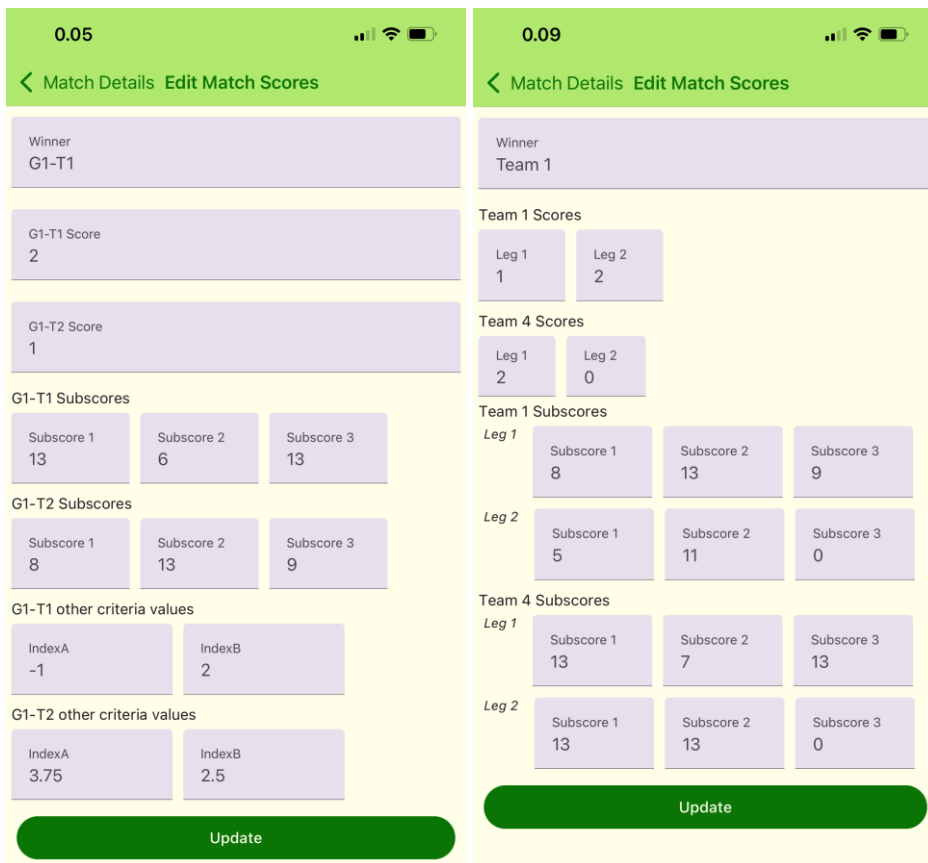
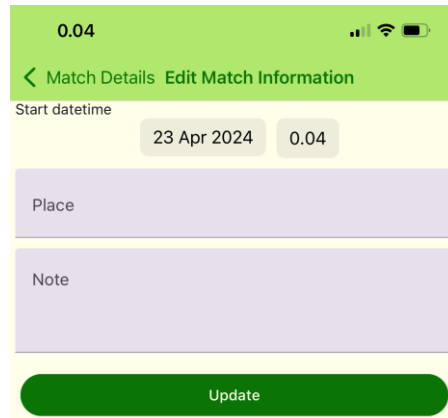


Figure 33. Round-robin (left) and single-elimination (right) edit match score form.

Finally, the figure below is the “EditMatchInformation” screen. The interface is applied the same way for both stage formats.



The screenshot shows a mobile application interface for editing match information. At the top, there is a green header bar with the text "0.04" on the left and signal, Wi-Fi, and battery icons on the right. Below the header, there is a navigation bar with a back arrow and the text "Match Details Edit Match Information". The main content area is light yellow and contains the following elements:

- A "Start datetime" label followed by two input fields: "23 Apr 2024" and "0.04".
- A large, empty text input area labeled "Place".
- A large, empty text input area labeled "Note".
- A green button at the bottom labeled "Update".

Figure 34. Edit match information form.

6 IMPLEMENTATION

The authentication server and the client were developed on Visual Studio Code, while the tournament data processing server was developed on Visual Studio 2022. All three code projects were managed by Git and pushed to GitHub as three separate repositories.

6.1 Authentication Server

The Python interpreter version used to run the server is Python 3.10.11. The figure below is the folder structure of the authentication server on Visual Studio Code IDE.

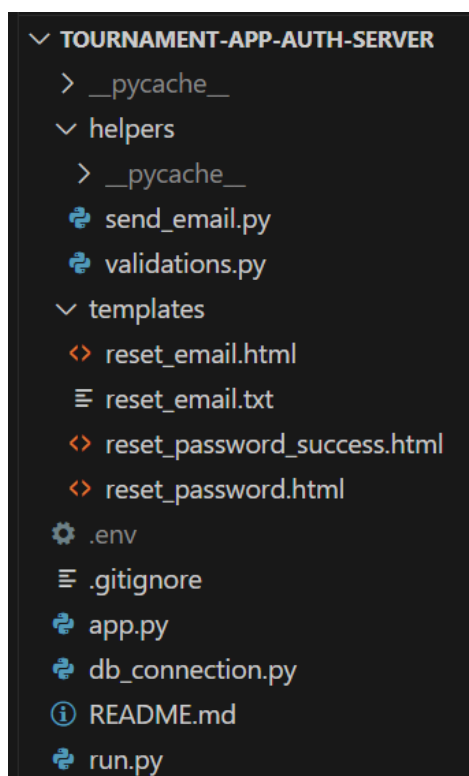


Figure 35. Authentication server folder structure.

The server needs to use some custom environment variables. One handy way to do this is to include those variables in the “.env” file in the root folder, and install

and import “dotenv” module to retrieve them. The environment variables that the server uses are listed below in the file:

```
FLASK_HOST_URL=0.0.0.0
POSTGRESQL_DATABASE_NAME=tournaments
POSTGRESQL_USERNAME=postgres
POSTGRESQL_PASSWORD=<insert_database_password>
POSTGRESQL_HOST=localhost
POSTGRESQL_PORT=5432
JWT_SECRET_KEY=<insert_a_secret_key>
MAIL_USERNAME=<insert_an_email_address>
MAIL_PASSWORD=<insert_the_email_password>
TOURNAMENT_DATA_SERVER_URL=<insert_the_public_url>
```

Code Snippet 1. Environment variables in authentication server.

The main code file which includes Flask configurations and endpoints is the “app.py” file. The configurations in this file are for mail service and CORS handling.

```
from flask_cors import CORS
from flask_mail import Mail

app = Flask(__name__)
app.config['MAIL_SERVER'] = 'smtp.gmail.com'
app.config['MAIL_PORT'] = 465
app.config['MAIL_USERNAME'] =
os.getenv("MAIL_USERNAME")
app.config['MAIL_PASSWORD'] = os.getenv("MAIL_PASSWORD")
app.config['MAIL_USE_TLS'] = False
app.config['MAIL_USE_SSL'] = True
mail = Mail(app)
CORS(app)
```

Code Snippet 2. Flask configuration in app.py.

The “db_connection.py” file has some variables retrieved from the “.env” file for psycopg2 database connection in the “app.py” file.

```
from dotenv import load_dotenv
import os

load_dotenv()
dbname = os.getenv('POSTGRESQL_DATABASE_NAME')
dbuser = os.getenv('POSTGRESQL_USERNAME')
```

```
dbpassword = os.getenv('POSTGRES_PASSWORD')
dbhost = os.getenv('POSTGRES_HOST')
dbport = os.getenv('POSTGRES_PORT')
```

Code Snippet 3. Psycopg2 database connection variables.

The templates folder contains some “.html” and “.txt” files, which is mainly used for password reset. The “helpers” folder has a few scripts where helper functions are defined and called in the “app.py” file. The “validations.py” file contains validation functions for different variables corresponding to the “auth.users” columns in the PostgreSQL database, such as username, email, password, and phone number. The “send_email.py” is responsible for sending automatic messages, using a thread to avoid delaying other services in the server.

```
from flask_mail import Message
from threading import Thread
from app import app, mail

def send_async_email(app, msg):
    with app.app_context():
        mail.send(msg)

def send_email(subject, sender, recipients, text_body,
html_body):
    msg = Message(subject, sender=sender,
recipients=recipients)
    msg.body = text_body
    msg.html = html_body
    Thread(target=send_async_email, args=(app,
msg)).start()
```

Code Snippet 4. send_email.py file.

To avoid circular import between the “app.py” and the “send_email.py” file, the run configurations for the Flask server will be stated in the “run.py” file. To run the server, run this file by this command: `py run.py`

```
import os
from app import app

if __name__ == '__main__':
```

```
app.run(debug = True, host =
os.getenv("FLASK_HOST_URL"), port = 5000)
```

Code Snippet 5. run.py file.

Here is the list of REST API endpoints in the server:

- /sign_up (POST)
 - Body keys: email (string), username (string), password (string)
- /sign_in (POST)
 - Body keys: username_or_email (string), password (string)
- /username/<user_id> (GET)
- /get_user_information (GET)
 - Header key: Authorization
- /change_user_information (POST)
 - Header key: Authorization
 - Body keys: new_username (string), country (string), phone (string)
- /change_password (POST)
 - Header key: Authorization
 - Body keys: current_password (string), new_password (string)
- /forgot_password (POST)
 - Body key: email (string)
- /reset_password/<token> (GET, POST)
- /delete_user_account (POST)
 - Header key: Authorization
 - Body key: password (string)

Every endpoint function starts with database connection and creating a cursor variable. Then, if an endpoint receives a request header and/or a request body, those values can be obtained by the request variable in the flask module. Next, if an endpoint requires a token, it can be retrieved from the “Authorization” header, validated and decoded. After that, the endpoint function does its main job. Finally, no matter whether the operation is successful or not, the database connection

should be closed and an object is returned. The return object always has an “isSuccess” Boolean value and may contain other information such as message or token. Thus, the structure of all endpoints is similar to the example below:

```

    conn = psycopg2.connect(dbname=dbname, user=dbuser,
password=dbpassword, host=dbhost, port=dbport)
    cur = conn.cursor()
    headers = request.headers
    request_body = request.get_json()
    try:
        token = headers["Authorization"].split("Bearer
", 1)[1]
        decoded_object = jwt.decode(token,
os.getenv("JWT_SECRET_KEY"), algorithms=["HS512"])

        # Main code

        conn.close()
        return {"isSuccess": True, "message":
"Success"} # Other key-value may be added here
    except jwt.exceptions.ExpiredSignatureError:
        conn.close()
        return {"isSuccess": False, "message": "Token
expired"}, 400
    except Exception:
        conn.close()
        return {"isSuccess": False, "message": "Bad
Request"}, 400

```

Code Snippet 6. Example of the structure of authentication server endpoints.

The list below is the detailed description of what each endpoint does on the server:

- **/sign_up:** The function validates the request body values. If they are valid, the password is hashed, the “last_sign_in_time” and “last_username_change_time” variables are initialized as current datetime variables and a new user is inserted into the database. Then a JWT token is generated and returned.
- **/sign_in:** The function checks if the username/email and password from the request body are empty. Then, it finds the user in the database. If it

finds one, it uses the saved hashed password to verify the provided password. If the verification is successful, it generates a JWT token and returns it to the client.

- **/username/<user_id>**: The function simply gets the username from the user ID provided in the URL.
- **/get_user_information**: The function gets almost every field of a user by the user ID retrieved from the decoded JWT token. It also includes some extra calculated values to the return object.
- **/change_user_information**: The function validates the request body values. Then, it checks if the difference between the current datetime and the last time the user changes the username is high enough and updates the username if the condition is satisfied. Finally, it changes other user information.
- **/change_password**: The function validates the request body values. Then, it finds the user who wants to change the password and verify the current password. If the verification is successful, the new password is hashed and updated in the database.
- **/forgot_password**: The function checks if the email exists in the database. If an email is found, an email message including a password reset token link is generated and sent to the email address.
- **/reset_password/<token>**: If the method is GET, a reset password web page is shown. If the method is POST, the function finds the user by the information in the decoded JWT token. Then, the new password provided is hashed and updated to the database.
- **/delete_user_account**: The function verifies the password sent from the client to make sure that the delete request is made by the correct authorized user. Then, it delete the user in “auth.users” table and all tournaments linked to the deleted user account in “data.tournaments” table by communicating with the tournament data processing server.

6.2 Tournament Data Server

.NET 8 is the framework version of the server. Figure 36 below shows the folder structure of the tournament data processing server on Visual Studio 2022 IDE.

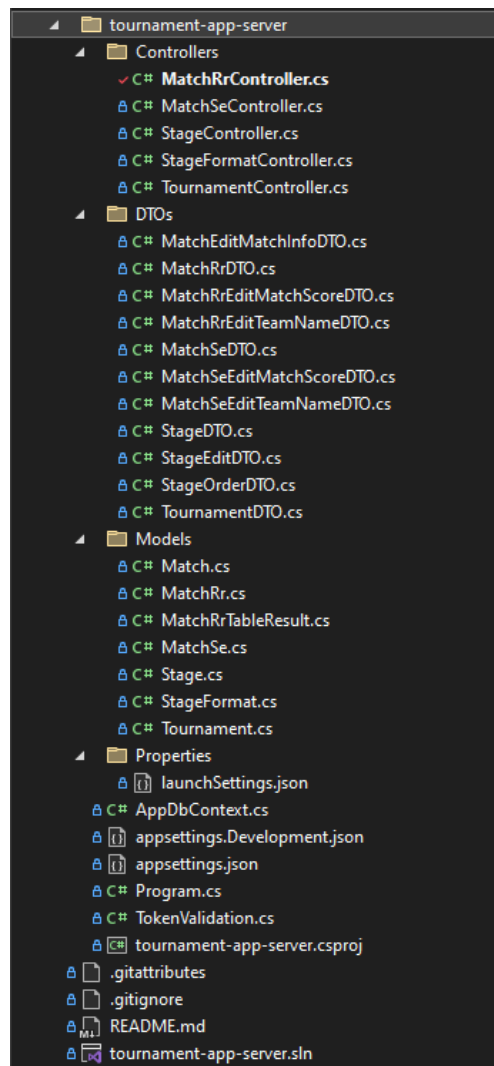


Figure 36. Tournament data processing server folder structure.

The project was initialized by the ASP.NET Core Web API template in the Visual Studio IDE. To run the server, use `dotnet run` command at the root folder or click the “Start” buttons on the Visual Studio toolbar (the green triangle icon). Some parameters are required for launching the server, and they can be adjusted in the “launchSettings.json” file. In this server, the environment variables and the application URL are added and modified.

```

{
  //Other setting parameters

  "profiles": {
    "tournament_app_server": {
      //Others
      "applicationUrl": "http://0.0.0.0:5244", //Any
unused registered port is fine
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development",
        "JWT_SECRET_KEY": "<insert_secret_key>"
      }
    },

    //Other profiles

  }
}

```

Code Snippet 7. launchSettings.json file.

A connection string must be prepared in the “appsettings.json” file in order to connect the server to the database. The connection string is named “DefaultConnection” and includes the database server URL and port, database name, and username and password of the database server.

```

{
  "ConnectionStrings": {
    "DefaultConnection":
"Server=localhost;Port=5432;Database=tournaments;;Usern
ame=postgres;Password=<insert_db_server_password>;"
  },

  //Other application settings

}

```

Code Snippet 8. appsettings.json file.

The “DefaultConnection” string will be used in the “Program.cs” file, where the database connection is established when the application starts running.

```
using Microsoft.EntityFrameworkCore;
```

```

using tournament_app_server;

var builder = WebApplication.CreateBuilder(args);

// Other builder services

var Configuration = builder.Configuration;
builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseNpgsql(Configuration.GetConnectionString(
        "DefaultConnection")));
var app = builder.Build();

// Other app configurations

app.Run();

```

Code Snippet 9. Program.cs file.

A helper function, called “ValidateToken” and located in the “TokenValidation.cs” file, is defined for later use in the controllers. The function decodes the JWT token string into an object whose values can be easily retrieved.

```

using Microsoft.IdentityModel.Tokens;
using System.IdentityModel.Tokens.Jwt;
using System.Text;

namespace tournament_app_server
{
    public class TokenValidation
    {
        public static JwtSecurityToken ValidateToken(string
token)
        {
            var handler = new JwtSecurityTokenHandler();
            var validationParameters = new
TokenValidationParameters
            {
                ValidateIssuerSigningKey = true,
                IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(Environment
.GetEnvironmentVariable("JWT_SECRET_KEY").PadRight(512
/ 8, '\0'))),
                ValidateIssuer = false,
                ValidateAudience = false,
                ValidAlgorithms =
[SecurityAlgorithms.HmacSha512]

```

```

        };
        handler.ValidateToken(token,
validationParameters, out var validatedToken);
        return handler.ReadJwtToken(token);
    }
}
}

```

Code Snippet 10. TokenValidation.cs file.

Three folders were created in the project: “Controllers”, “DTOs” and “Models”. The “Controllers” folder is the place where different endpoints are implemented; those endpoints use classes in the “DTOs” and “Models” folder for data processing. The “DTOs” folder has property-only classes that represent the request body objects sent from the client. The “Models” folder is similar to the “DTOs” folder, but the classes represent the data models of the database tables as well as the data objects returned to the client.

To determine which schema and tables the server accesses, the “AppDbContext” class is written in a C# file with the same name. It inherits the “DbContext” class from the EF Core library. By the “DbSet” properties, it also converts LINQ queries written in the controller classes into SQL queries.

```

using Microsoft.EntityFrameworkCore;
using tournament_app_server.Models;

namespace tournament_app_server
{
    public class AppDbContext : DbContext
    {
        public AppDbContext(DbContextOptions<AppDbContext>
options) : base(options) { }
        public DbSet<Tournament> Tournaments { get; set; }
        public DbSet<Stage> Stages { get; set; }
        public DbSet<StageFormat> StageFormats { get; set; }
        public DbSet<MatchSe> MatchSes { get; set; }
        public DbSet<MatchRr> MatchRrs { get; set; }

        protected override void OnModelCreating
(ModelBuilder modelBuilder)
        {

```

```

        modelBuilder.HasDefaultSchema("data"); //
        Configure table schema
        modelBuilder.Entity<Tournament>().ToTable("tournaments");
        modelBuilder.Entity<Stage>().ToTable("stages");
        modelBuilder.Entity<StageFormat>().ToTable("stage_format");
        modelBuilder.Entity<MatchSe>().ToTable("matches_se");
        modelBuilder.Entity<MatchRr>().ToTable("matches_rr");
        base.OnModelCreating(modelBuilder);
    }
}
}

```

Code Snippet 11. AppDbContext class.

Code snippet 12 below shows an example of the controller structure and how the “AppDbContext” is used in the controller. The **[Route]** tag indicates the starting part of the endpoints URL inside the class, and the rest part is defined in the **[HttpGet]** (or **[HttpPost]**, **[HttpPut]**, **[HttpDelete]**) tag, which defines the endpoint request method. The curly brackets in the URL string indicate the parameters of the endpoint functions. Each controller inherits the “ControllerBase” class of the “Microsoft.AspNetCore.Mvc” module. The database context is initialized before any endpoints. In every endpoint, the parameters are not only linked to the URL parameters. If any information from the request header and the request body is needed, the **[FromHeader]** and **[FromBody]** tags are used before the parameters themselves, respectively. The purposes of all endpoints will be discussed briefly later, but the application of the “AppDbContext” below can be easily understood.

```

using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using tournament_app_server.DTOs;
using tournament_app_server.Models;

namespace tournament_app_server.Controllers
{
    [Route("/stages")]
    [ApiController]

```

```

public class StageController : ControllerBase
{
    private readonly AppDbContext _dbContext;

    public StageController(AppDbContext dbContext)
    {
        _dbContext = dbContext;
    }

    [HttpGet("all/{tournament_id}")]
    public async Task<ActionResult<IEnumerable<Stage>>>
    GetStagesByTournamentId(long tournament_id,
    [FromHeader(Name = "Authorization")] string token = "")
    {
        if (_dbContext.Stages == null)
        {
            return NotFound();
        }

        try
        {
            if (token.Contains("Bearer "))
            {
                token = token.Split("Bearer ")[1];
            }
            var tournament = await
            _dbContext.Tournaments.FindAsync(tournament_id);
            if (tournament == null)
            {
                return NotFound();
            }

            if (token.Trim() == "")
            {
                if (tournament.is_private == true)
                {
                    throw new Exception("Cannot access or
modify these private stages without a valid token.");
                }
            }
            else
            {
                var decodedToken =
TokenValidation.ValidateToken(token);
                var payload = decodedToken.Payload;
                int userId = (int)payload["id"];
                if (tournament.user_id != userId &&
tournament.is_private == true)
            {

```



```

        throw new Exception("Cannot access or
modify these stages by your token.");
    }

    }

    return await _dbContext.Stages
        .Where(s => s.tournament_id == tournament_id)
        .OrderBy(s => s.stage_order)
        .ToListAsync();
    }
    catch (Exception ex)
    {
        return BadRequest(ex.Message);
    }
}

//Other endpoints
}

```

Code Snippet 12. Controller class example.

The tables of endpoints are implemented in the five controller classes. Any endpoints having the “Authorization” header key means that the user might be required a JWT token to get access to them.

- **TournamentController.cs:**
 - /tournaments/public (GET)
 - /tournaments/all (GET)
 - Header key: Authorization
 - /tournaments/<id> (GET)
 - Header key: Authorization
 - /tournaments (POST)
 - Header key: Authorization
 - Body keys: name (string), start_date (string), end_date (string), places (string[]), description (string), is_private (bool)
 - /tournaments/<id> (PUT)
 - Header key: Authorization

- Body keys: name (string), start_date (string), end_date (string), places (string[]), description (string), is_private (bool)
- /tournaments/<id> (DELETE)
 - Header key: Authorization
- **StageController.cs:**
- /stages/all/<tournament_id> (GET)
 - Header key: Authorization
- /stages/<id> (GET)
 - Header key: Authorization
- /stages (POST)
 - Header key: Authorization
 - Body keys: name (string), format_id (int), start_date (string), end_date (string), places (string[]), description (string), tournament_id (int), number_of_teams_per_group (int), number_of_groups (int), stage_order (int), include_third_place_match (bool), number_of_legs_per_round (int[]), best_of_per_round (int[]), third_place_match_number_of_legs (int), third_place_match_best_of (int), win_point (double), draw_point (double), lose_point (double), other_criteria_names (string[]), other_criteria_sort_direction (string[])
- /stages/<id> (PUT)
 - Header key: Authorization
 - Body keys: name (string), start_date (string), end_date (string), places (string[]), description (string), tournament_id (int),
- /stages/order (PUT)
 - Header key: Authorization
 - Body keys: List of: id (int), name (string), tournament_id (int), stage_order (int)
- /stages/<id> (DELETE)

- Header key: Authorization
- **StageFormatController.cs:**
 - /stage_format (GET)
 - /stage_format/<id> (GET)
- **MatchSeController.cs:**
 - /matches/se/all/<stage_id> (GET)
 - Header key: Authorization
 - /matches/se/<id> (GET)
 - Header key: Authorization
 - /matches/se/<id>/team_name (PUT)
 - Header key: Authorization
 - Body keys: team_1 (string), team_2 (string)
 - /matches/se/<id>/match_info (PUT)
 - Header key: Authorization
 - Body keys: start_datetime (string), place (string), note (string)
 - /matches/se/<id>/match_score (PUT)
 - Header key: Authorization
 - Body keys: winner (string), team_1_scores (double[]), team_2_scores (double[]), team_1_subscores (double[]), team_2_subscores (double[])
- **MatchRrController.cs:**
 - /matches/rr/all/<stage_id> (GET)
 - Header key: Authorization
 - /matches/rr/<id> (GET)
 - Header key: Authorization
 - /matches/rr/table_results/<stage_id>/<group_number> (GET)
 - Header key: Authorization
 - /matches/rr/<id>/team_name (PUT)
 - Header key: Authorization
 - Body keys: old_team_name (string), new_team_name (string)

- `/matches/rr/<id>/match_info` (PUT)
 - Header key: Authorization
 - Body keys: `start_datetime` (string), `place` (string), `note` (string)
- `/matches/rr/<id>/match_score` (PUT)
 - Header key: Authorization
 - Body keys: `winner` (string), `team_1_score` (double), `team_2_score` (double), `team_1_subscores` (double[]), `team_2_subscores` (double[]), `team_1_other_criteria_values` (double[]), `team_2_other_criteria_values` (double[])

The purpose of each endpoint are:

- **`/tournaments/public` (GET):** Return all tournaments where the “is_private” column in the database is false.
- **`/tournaments/all` (GET):** Return all tournaments of a user. The user ID can be retrieved in the JWT token.
- **`/tournaments/<id>` (GET):** Return tournament information by a tournament ID. Only the authorized user can access it if it is private.
- **`/tournaments` (POST):** Save a new tournament to the database. A valid token is required.
- **`/tournaments/<id>` (PUT):** Update a tournament in the database. Only the authorized user can modify it.
- **`/tournaments/<id>` (DELETE):** Delete a tournament as well as its stages and matches. Only the authorized user can delete it.
- **`/stages/all/<tournament_id>` (GET):** Return all stages of a tournament by a tournament ID. Only the authorized user can access it if the tournament is private.
- **`/stages/<id>` (GET):** Return stage information by a stage ID. Only the authorized user can access it if the tournament is private.

- **/stages (POST):** Save a new stage to the database and generate matches based on the selected stage format. The match generation process will be the same for every group.
 - If the format is single elimination, the number of teams in each group will be calculated to the upper nearest power of 2. Then matches will be generated in the first round; in the second round, the number of matches will be halved, and so on until the final match. A third-place match is optional for all groups.
 - If the format is round robin, a permutation (made by two nested loops of “number_of_teams_per_group”) is done to generate all match combinations. If the number of legs is larger or equal to two, the process is repeated corresponding times.
- **/stages/<id> (PUT):** Update a stage information (but not stage configuration) in the database. Only the authorized user can modify it.
- **/stages/order (PUT):** Change the order of stages in a tournament. Only the authorized user can modify it.
- **/stages/<id> (DELETE):** Delete a stage as well as its matches. Only the authorized user can delete it.
- **/stage_format (GET):** Return all stage formats.
- **/stage_format/<id> (GET):** Return a stage format by a stage format ID.
- **/matches/se/all/<stage_id> (GET):** Return all single-elimination matches of a stage by a stage ID. Only the authorized user can access it if the tournament is private.
- **/matches/se/<id> (GET):** Return single-elimination match information by a match ID. Only the authorized user can access it if the tournament is private.
- **/matches/se/<id>/team_name (PUT):** Change team names in a single-elimination stage. The names must be unique in that stage. The change must start in one of the first-round matches found by match ID. After that,

the old team names are searched in the remaining rounds. If they are found, they will also be changed. Only the authorized user can modify it.

- **/matches/se/<id>/match_info (PUT):** Update the information of a match (start_datetime, place, note). Only the authorized user can modify it.
- **/matches/se/<id>/match_score (PUT):** Update a single-elimination match result. Then the winning team name will be updated in the correct next round match and clear the scores of that match. Finally, the dependent matches after that next round will be reset (clear the team names and the scores). Only the authorized user can modify it.
- **/matches/rr/all/<stage_id> (GET):** Return all round-robin matches of a stage by a stage ID. Only the authorized user can access it if the tournament is private.
- **/matches/rr/<id> (GET):** Return round-robin match information by a match ID. Only the authorized user can access it if the tournament is private.
- **/matches/rr/table_results/<stage_id>/<group_number> (GET):** Calculate the table results of a group in a round-robin stage. This function calculates total points, difference, accumulated/earned score and other criteria values for each team in the group, then the results will be sorted by total points, then difference, then accumulated/earned score and finally other criteria values. Only the authorized user can access it if the tournament is private.
 - Total points: added if the team wins, draws or loses. “win_point”, “draw_point” and “lose_point” properties of the “Stage” class are used.
 - Difference: accumulated from the score difference of a team plays against each opponent in a group.
 - Accumulated/earned score: The sum of the score a team earned in every match in a group.

- Other criteria values: The sum of the criteria values a team gets in every match in a group. The values are sorted ascending or descending depending on the stage configuration.
- **/matches/rr/<id>/team_name (PUT):** Change team names in a round-robin stage. The names must be unique in that stage. They can be changed in any match, and if a team name is changed in a match, that name in other matches will also be changed. Only the authorized user can modify it.
- **/matches/rr/<id>/match_info (PUT):** Similar to “/matches/se/<id>/match_info” endpoint, but it works for round-robin matches. Only the authorized user can modify it.
- **/matches/rr/<id>/match_score (PUT):** Update a round-robin match result. Only the authorized user can modify it.

6.3 The client

In the client project, the React Native version is 0.72 (then upgraded to 0.74) and the Expo SDK version is 49 (then upgraded to 51). Figure 37 shows the folder structure of the React Native client on Visual Studio Code IDE.

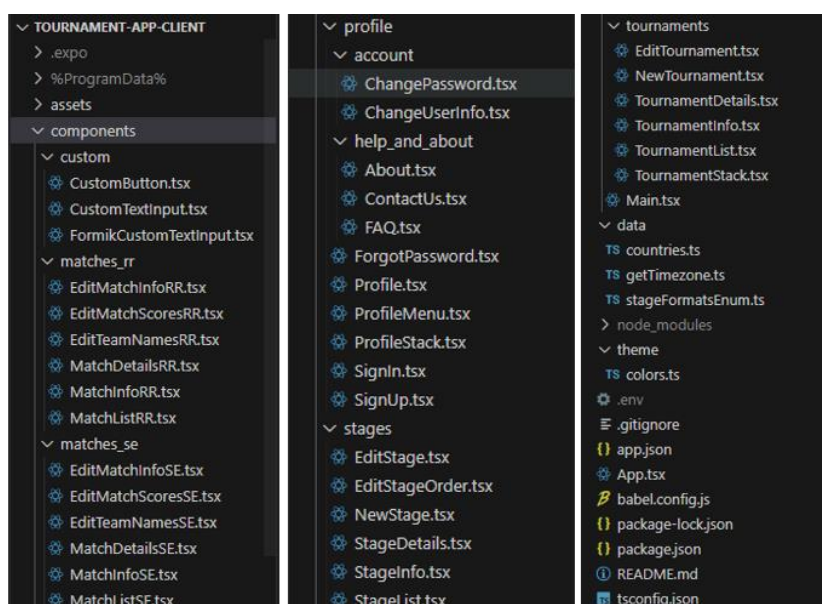


Figure 37. Client folder structure.

After the client project initialization, the orientation property in the “app.json” was “portrait”. This property was changed to “default”, which means the application is not locked to any orientation. There are other properties in this file, but they do not need to be changed now.

```
{
  "expo": {
    "orientation": "default",

    //Other properties
  }
}
```

Code Snippet 13. app.json configuration file.

The two environment variables used in the application are “EXPO_PUBLIC_AUTH_SERVER_URL” and “EXPO_PUBLIC_SERVER_URL”, which store the authentication and tournament data processing server base URLs, respectively (the variable name must start with “EXPO_PUBLIC_”, because Expo CLI loads those variables using this format). The variables are written in the “.env” file. In the code, they are used by the `process.env` object, for example:

```
const response = await
fetch(process.env.EXPO_PUBLIC_AUTH_SERVER_URL +
"/get_user_information")
```

Code Snippet 14. Example of using environment variables in the code.

The “colors.ts” file in the theme folder contains some hex-value color variables. In the data folder, the “countries.ts” code fetches information from all countries (API URL: <https://countriesnow.space/api/v0.1/countries/codes>); the “getTimezone” function in the “getTimezone.ts” file formats and returns the time zone string value from a given date object; and the “stageFormatsEnum.ts” file has an object that stores stage format ID as explicit names to avoid magic numbers in the code.

The “components” folder contains all the other components to render elements to the screen. This folder contains several subfolders. The “profile” folder stores components for authentication and managing user profiles. Application information and instruction are also rendered from the components in this folder. The “tournaments”, “stages”, “matches_se” and “matches_rr” folders store components for tournament, stage, single-elimination match and round-robin match data manipulation, respectively. Furthermore, custom components are also implemented in the “custom” folder and used in other places.

The root component is the “App.tsx” file. In this file, the “Main” component is rendered. The “SafeAreaProvider” component is wrapped around “Main” component so that the components inside the “Main” component are not covered by operation system elements such as status bar and home indicators. The “PaperProvider” component of the React Native Paper library is also needed for using the library’s UI components inside the “Main” component. Moreover, the “Main” component is wrapped by the “NavigationContainer” component of the React Navigation library for navigation components and states inside. Finally, for the Expo SDK 50 or above, the “react-native-reanimated” package must be installed and imported so that the application is not crashed while executing any navigation operation.

```
import { SafeAreaProvider } from 'react-native-safe-area-context';
import Main from './components/Main';
import { PaperProvider } from 'react-native-paper';
import { NavigationContainer } from '@react-navigation/native';
import 'react-native-reanimated';

export default function App() {
  return (
    <NavigationContainer>
      <PaperProvider>
        <SafeAreaProvider>
          <Main />
        </SafeAreaProvider>
      </PaperProvider>
    </NavigationContainer>
  );
}
```

```

        </NavigationContainer>
    )
}

```

Code Snippet 15. App.tsx file.

The “Main” component is in the “components” folder. In this component, the bottom navigation tabs are rendered. The component also initializes the “navigation” prop and the “token” state to be used in the subcomponents. Lastly, the “StatusBar” component is a quick and convenient way to style the operation system status bar.

```

import { createBottomTabNavigator } from '@react-
navigation/bottom-tabs';
import { StatusBar } from 'expo-status-bar';
import ProfileStack from './profile/ProfileStack';
import TournamentStack from
'./tournaments/TournamentStack';
import { secondary, tertiary } from '../theme/colors';
import { Ionicons } from '@expo/vector-icons';
import { useState } from 'react';
import { useNavigation } from '@react-
navigation/native';

const BottomTab = createBottomTabNavigator()

const Main = () => {
  const navigation = useNavigation()
  const [token, setToken] = useState<string>('')
  return (
    <>
      <StatusBar style="auto" />
      <BottomTab.Navigator
        screenOptions={({ route }) => ({
          headerShown: false,
          tabBarStyle: {
            backgroundColor: secondary
          },
          tabBarIcon: ({ color, size }) => {
            if (route.name === 'TournamentStack') {
              return (
                <Ionicons
                  name="trophy"
                  size={size}
                  color={color}

```

```

        />
      );
    } else if (route.name === 'ProfileStack') {
      return (
        <Ionicons
          name="person"
          size={size}
          color={color}
        />
      );
    }
  },
  tabBarActiveTintColor: tertiary,
))}
>
<BottomTab.Screen
  name="TournamentStack"
  children={() => <TournamentStack
navigation={navigation} token={token} />}
  options={{ title: 'Tournaments' }}
/>
<BottomTab.Screen
  name="ProfileStack"
  children={() => <ProfileStack
navigation={navigation} token={token}
setToken={setToken} />}
  options={{ title: 'Profile' }}
/>
</BottomTab.Navigator>
</>
);
}

export default Main

```

Code Snippet 16. Main.tsx file.

Pressing the tabs renders the “ProfileStack” and “TournamentStack” components. Those stacks are collections of navigated screens. The “ProfileStack” contains screens which render the components in the “profile” folder, while the “TournamentStack” contains screens which render the components in the “tournaments”, “stages”, “matches_se” and “matches_rr” folder. Code snippet 17 gives an example of implementation of those stacks.

```

import { createNativeStackNavigator } from "@react-
navigation/native-stack";
import { secondary, tertiary } from
"../../theme/colors";
import TournamentList from "./TournamentList";
import NewTournament from "./NewTournament";

{/*Other imports from 'components' folder*/}

const Stack = createNativeStackNavigator();

const TournamentStack = ({ navigation, token }: any) =>
{
  return (
    <Stack.Navigator
      screenOptions={{
        headerStyle: {
          backgroundColor: secondary
        },
        headerTintColor: tertiary
      }}
    >
    <Stack.Screen
      name="TournamentList"
      children={() => <TournamentList
navigation={navigation} token={token} />}
      options={{
        title: 'Your tournaments',
      }}
    />
    <Stack.Screen
      name="NewTournament"
      component={NewTournament}
      options={{
        title: 'New Tournament',
      }}
    />

    {/*Other Stack.Screen components*/}

    </Stack.Navigator>
  );
}
export default TournamentStack

```

Code Snippet 17. TournamentStack.tsx file.

The communications with the servers are made using the “fetch” API, which is the same as the “fetch” API in JavaScript. The API is implemented in the “useEffect” hooks and the Formik submit functions. Here is an example of a function that updates tournament information. The “values” parameter is an object of Formik input field values.

```
const updateTournament = (values: any) => {
  //Some input processing operations to 'values' (if
  necessary)

  fetch(`${process.env.EXPO_PUBLIC_SERVER_URL}/tourname
nts/${tournamentInfo.id}`, {
    method: 'PUT',
    headers: {
      'Content-Type': 'application/json',
      'Authorization': 'Bearer ' + token
    },
    body: JSON.stringify(values),
  })
  .then(async response => {
    if (response.ok) {
      return response.json()
    }
    else throw new Error(await response.text())
  })
  .then(data => {

    //State update and other operations

  })
  .catch((error: any) => {
    setServerErrorMessage(error.message)
  })
}
```

Code Snippet 18. updateTournament function.

7 TESTING

The application testing was conducted manually using the TDD method. After an endpoint in the servers or a component in the client is implemented, it is tested with sufficient test cases. Also, integration testing is done each time some backend routes and some corresponding frontend components are ready.

The servers themselves are tested by sending requests from Postman. The client, together with worked backend endpoints, is tested by the Expo Go application, mainly on the iPhone 12 Pro Max (sometimes on an Android emulator).

The tables below illustrate the description of the test cases and their results.

Table 2. Test results.

Expectation	Result
The sign in operation is successful if the credentials are correct. Otherwise, an error message must be displayed.	Pass
When the sign up operation is successful, a new account is created and the user is automatically signed in. If an error happens, an error message must be displayed. Password rules are applied.	Pass
When the user sends an email for password reset, he/she receives a unique reset link where he/she can submit a new password. Password rules are applied.	Pass
The user can change his/her user information and password. The username can only be changed again after a number of days since the last change. Password rules are applied.	Pass
When signing in, the user can sign out or delete his/her account. Deleting the account requires a password confirmation.	Pass
The user can see both his/her and public tournament lists if he/she is signing in. Otherwise, only the public list can be seen.	Pass
The user cannot modify any tournaments in the public list.	Pass

The test cases below will be applied to the user who has already signed in.

Table 3. Test results (continue).

Expectation	Result
The user can create, edit and delete tournaments and their stages inside.	Pass
After creating a new stage, the corresponding matches must also be generated based on stage configurations.	Pass (with not fully optimized results)
The order of stages in a tournament can be changed.	Pass (with average UX/UI)
When the tournaments and stages are deleted, confirmation must be made by typing their names. Deleting a tournament will delete its stages and matches, and deleting a stage will delete its matches.	Pass
The user can see the match list and overall results of the stages, as well as a specific match results by pressing a match in the match list	Pass
After editing team names, match information or match results, the match results, match list and overall results of the stage must be automatically updated.	Pass
Team names must be unique in the same stage.	Pass

The test cases in the two tables above cover the F1-F5, F7, F13 and F14 requirements in Table 1, section 3.1. The remaining requirements are also tested and passed.

8 CONCLUSIONS

The developed application meets most of the initial requirements. Each part of the system works and communicates with each other smoothly. The two servers provide different services, and both the servers and client code are well-organized so that error tracing is easier. The elements displayed on the screen are clear and concise, and navigation between screens is quick and easy.

The users can use the application conveniently and securely. They can manage multiple tournaments and customize multiple stages inside those tournaments. Moreover, the application automatically and immediately updates the overall stage results whenever the users update any match result.

Several challenges were met during the development process. First, the match generation in the stage creation depends on the stage format and many other stage configurations. Therefore, a complex algorithm is implemented to ensure the correct number of matches with correct initial values of each match, is generated. Second, in the team name edition process, the application does not only update the names in the selected match, but it also needs to find other matches in the same stage where the old names appear. Finally, updating the overall stage results based on match results takes quite a lot of time to implement, because each stage format has its own implementation, and the algorithm in each implementation is not simple.

The application is designed to be extensible, and some features can be developed in the future. One good example is that when creating a new stage, the user has more stage format choices such as double elimination. Another feature to be considered is to provide the user with a better customization of tiebreaking criteria in the round robin stages. Moreover, a visual bracket figure should also be implemented in the single elimination stages so that the user has an intuitive view of the overall stage result.

REFERENCES

- [1] Byl, J. (2014) Organizing Successful Tournaments, 4th edition. United Kingdom: Human Kinetics.
- [2] IBM (n.d.) What is PostgreSQL?. Retrieved April 7, 2024 from <https://www.ibm.com/topics/postgresql>
- [3] Coursera (2024) What Is Python Used For? A Beginner's Guide. Retrieved April 7, 2024 from <https://www.coursera.org/articles/what-is-python-used-for-a-beginners-guide-to-using-python>
- [4] Pallets (2023) Flask. Retrieved April 7, 2024 from <https://github.com/pallets/flask/>
- [5] Psycopg (n.d.) Psycopg – PostgreSQL database adapter for Python. Retrieved April 7, 2024 from <https://www.psycopg.org/docs/>
- [6] Schlawack, H. (2015) What is Argon2?. Retrieved April 7, 2024 from <https://argon2-cffi.readthedocs.io/en/stable/argon2.html>
- [7] Microsoft (n.d.) C#. Retrieved April 7, 2024 from <https://dotnet.microsoft.com/en-us/languages/csharp>
- [8] Microsoft (2024) Introduction to .NET. Retrieved April 7, 2024 from <https://learn.microsoft.com/en-us/dotnet/core/introduction>
- [9] Microsoft (n.d.) Entity Framework documentation hub. Retrieved April 8, 2024 from <https://learn.microsoft.com/en-us/ef/>
- [10] Microsoft (2021) Entity Framework Core. Retrieved April 8, 2024 from <https://learn.microsoft.com/en-us/ef/core/>
- [11] IBM (2024) JSON Web Token (JWT). Retrieved April 8, 2024 from <https://www.ibm.com/docs/en/cics-ts/6.1?topic=cics-json-web-token-jwt>
- [12] JWT.io (n.d.) JSON Web Tokens - jwt.io. Retrieved April 8, 2024 from <https://jwt.io/>

- [13] React Native (2023) Core Components and Native Components. Retrieved April 8, 2024 from <https://reactnative.dev/docs/intro-react-native-components>
- [14] Expo (n.d.) Expo. Retrieved April 8, 2024 from <https://expo.dev/>
- [15] Expo (n.d.) Expo Go. Retrieved April 8, 2024 from <https://docs.expo.dev/get-started/expo-go/>
- [16] W3Schools (n.d.) JavaScript Tutorial. Retrieved April 8, 2024 from <https://www.w3schools.com/js/>
- [17] TypeScript (n.d.) TypeScript is JavaScript with syntax for types. Retrieved April 8, 2024 from <https://www.typescriptlang.org/>
- [18] React Navigation (n.d.) React Navigation. Retrieved April 26, 2024 from <https://reactnavigation.org/>
- [19] React Native Paper (n.d.) Cross-platform Material Design for React Native. Retrieved April 8, 2024 from <https://callstack.github.io/react-native-paper/>
- [20] Formik (n.d.) Overview. Retrieved April 8, 2024 from <https://formik.org/docs/overview>