IT 309 SOFTWARE ENGINEERING

PROJECT DOCUMENTATION

PocketPoint

Prepared by:
Bakir Mujkanović
Sead Smailagić

Proposed to:
Nermina Durmić, Assist. Prof. Dr.
Aldin Kovačević, Teaching Assistant

23.07.2023

TABLE OF CONTENTS

1. Introduction

As students with a passion for cue sports, we embarked on a project to create a user-friendly application called PocketPoint. Being novice developers, we approached this project with enthusiasm, despite our limited experience. In this paper, we will provide an overview of PocketPoint, highlighting its core functionalities and how it simplifies the organization of cue sport tournaments. This project serves as a learning experience for us, as we explore the challenges and opportunities of developing a software application from scratch, using technologies and principles we haven't used so far.

1.1. About the Project

PocketPoint: A Cue Sports Tournament Management App. This app aims to assist cue sports enthusiasts in managing tournaments, matches, and leagues effectively. A working demo can be viewed here.

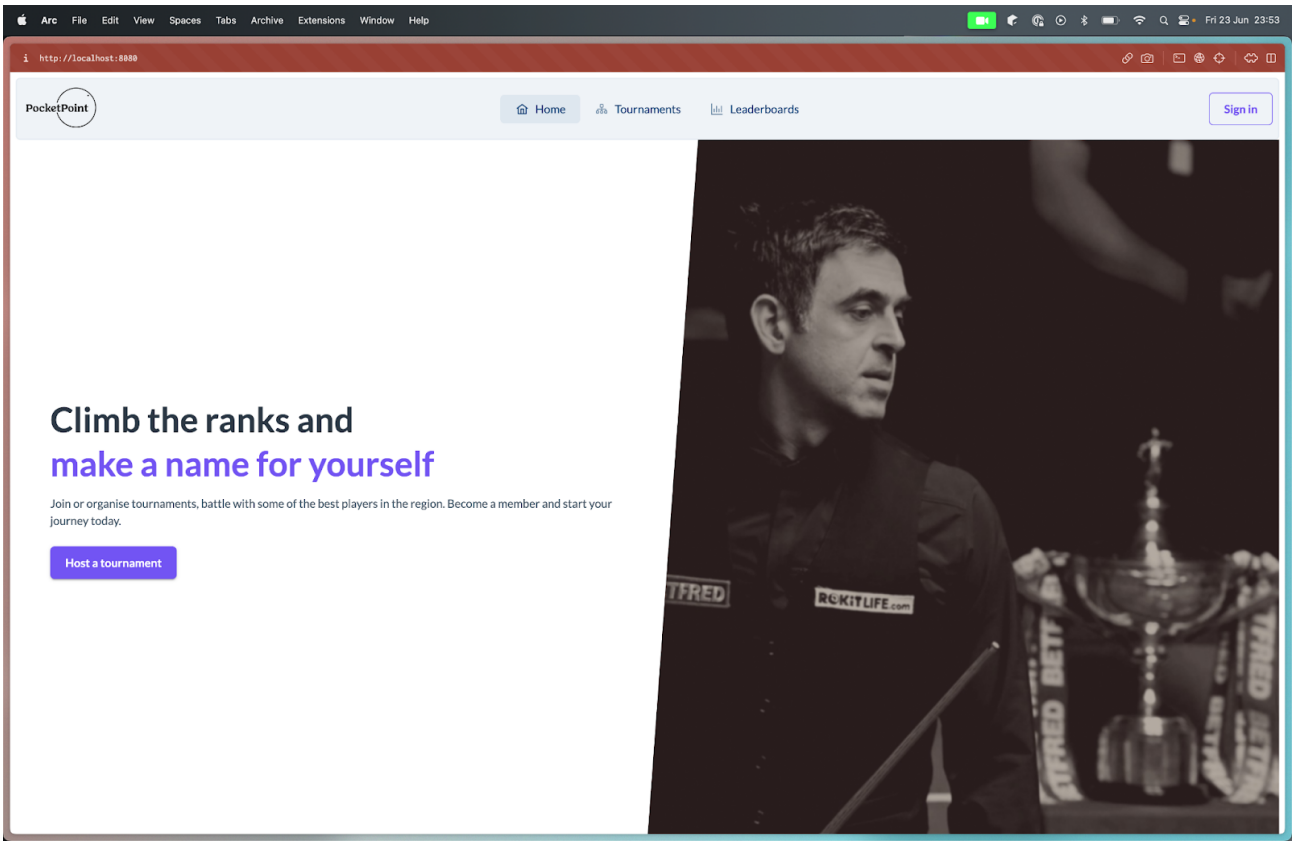
Github repositories:
https://github.com/mujkanovic01/pocketpoint
https://github.com/mujkanovic01/pocketpoint-fe

1.2. Project Functionalities and Screenshots

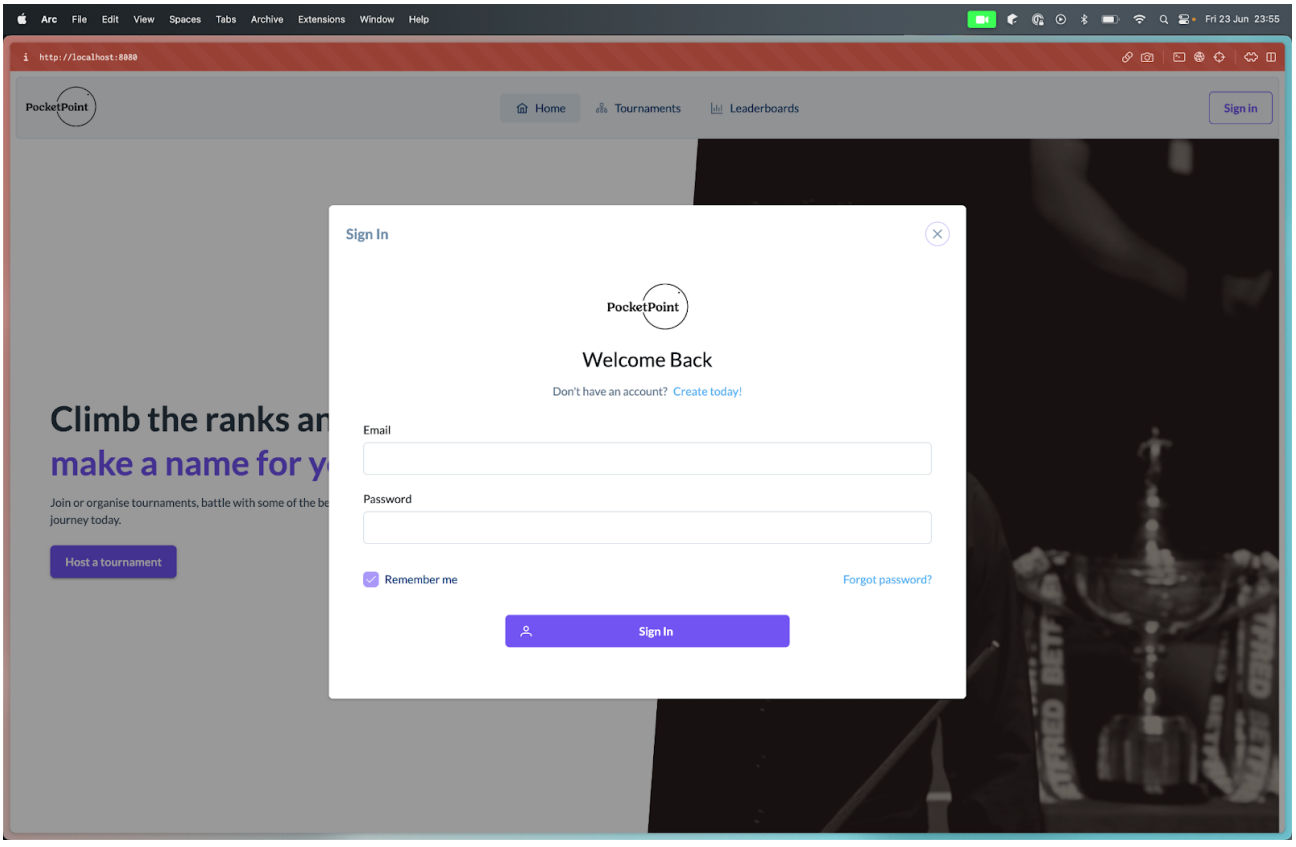
Some of the main features of the application:

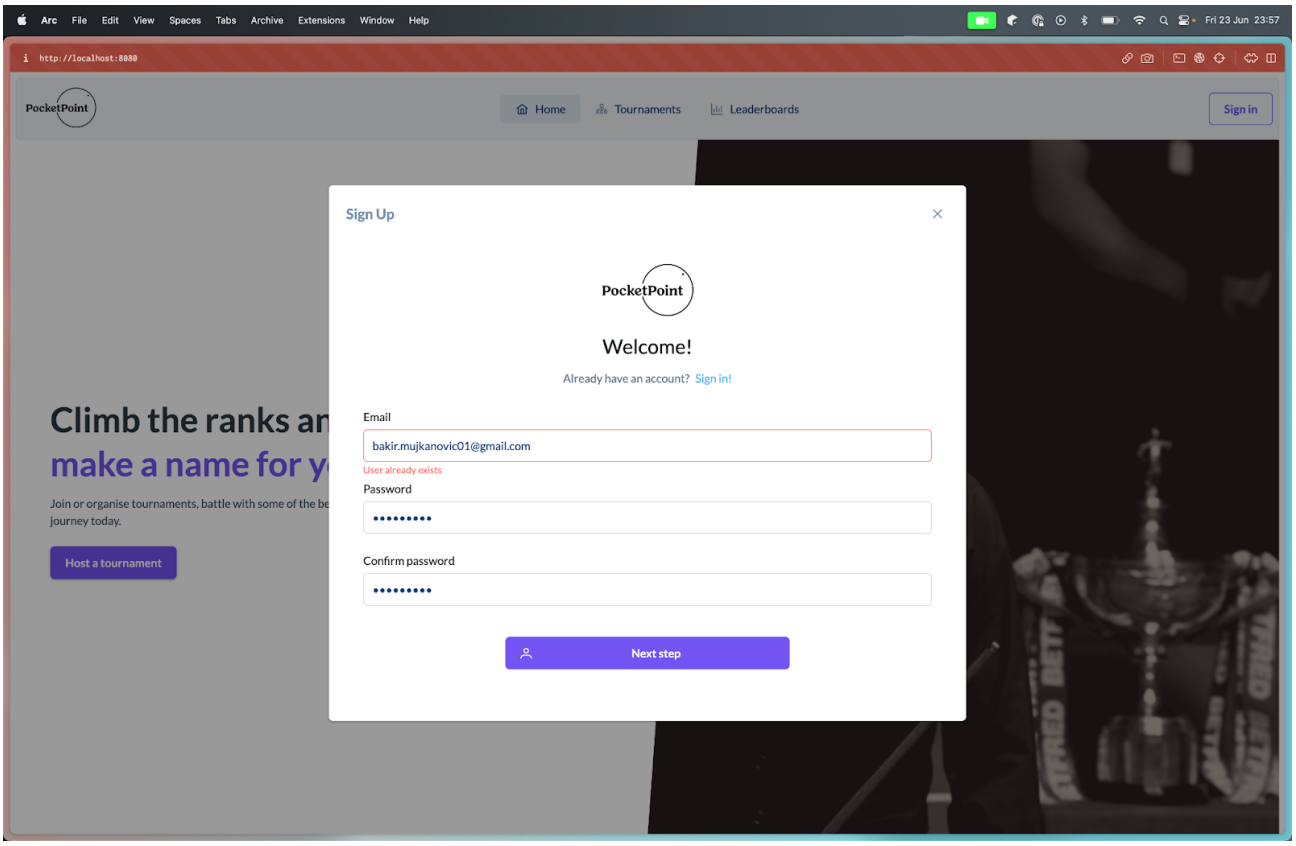
- Sign up/Sign in
- Frontend/Backend Validation
- Tournament Creation process
- Brackets generation for Single elimination tournaments with variable player counts

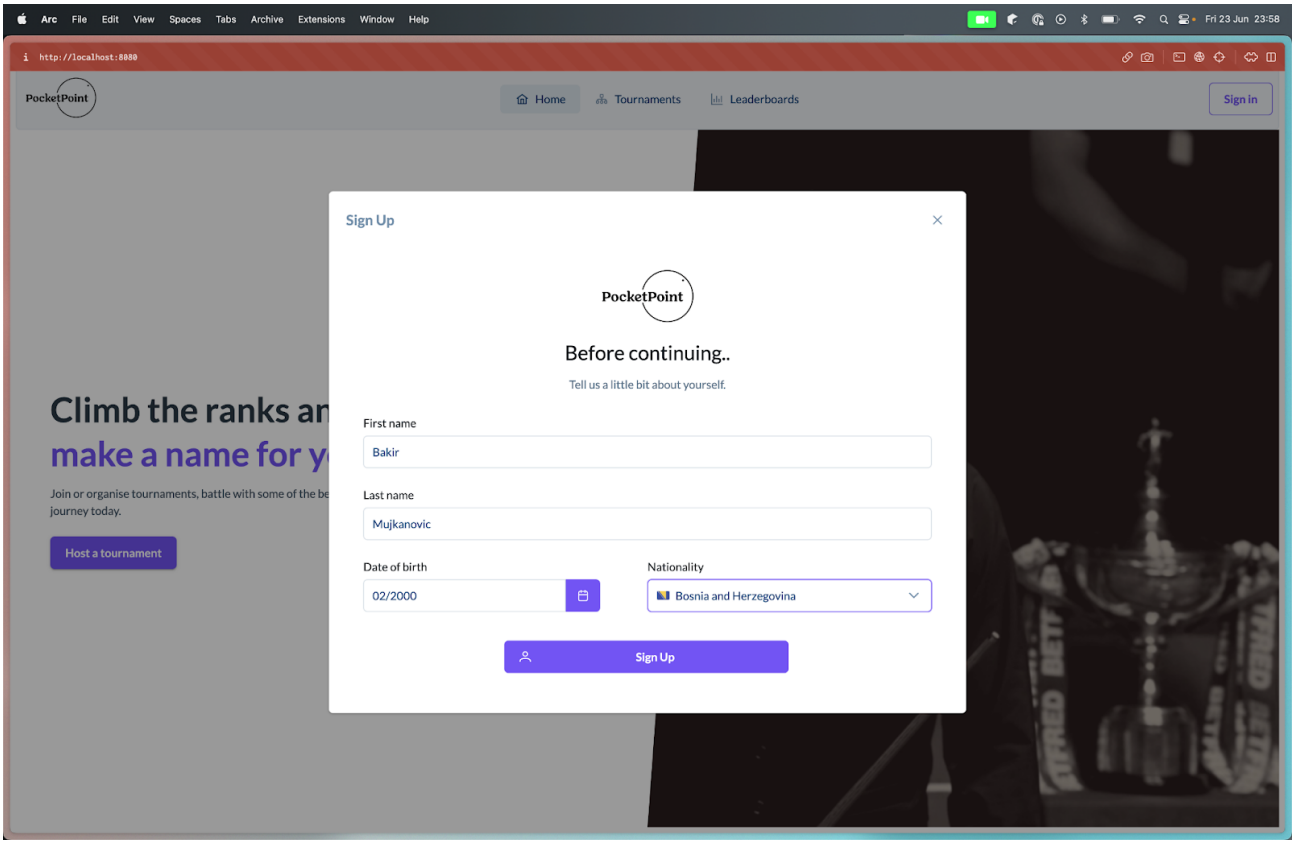
Landing page:



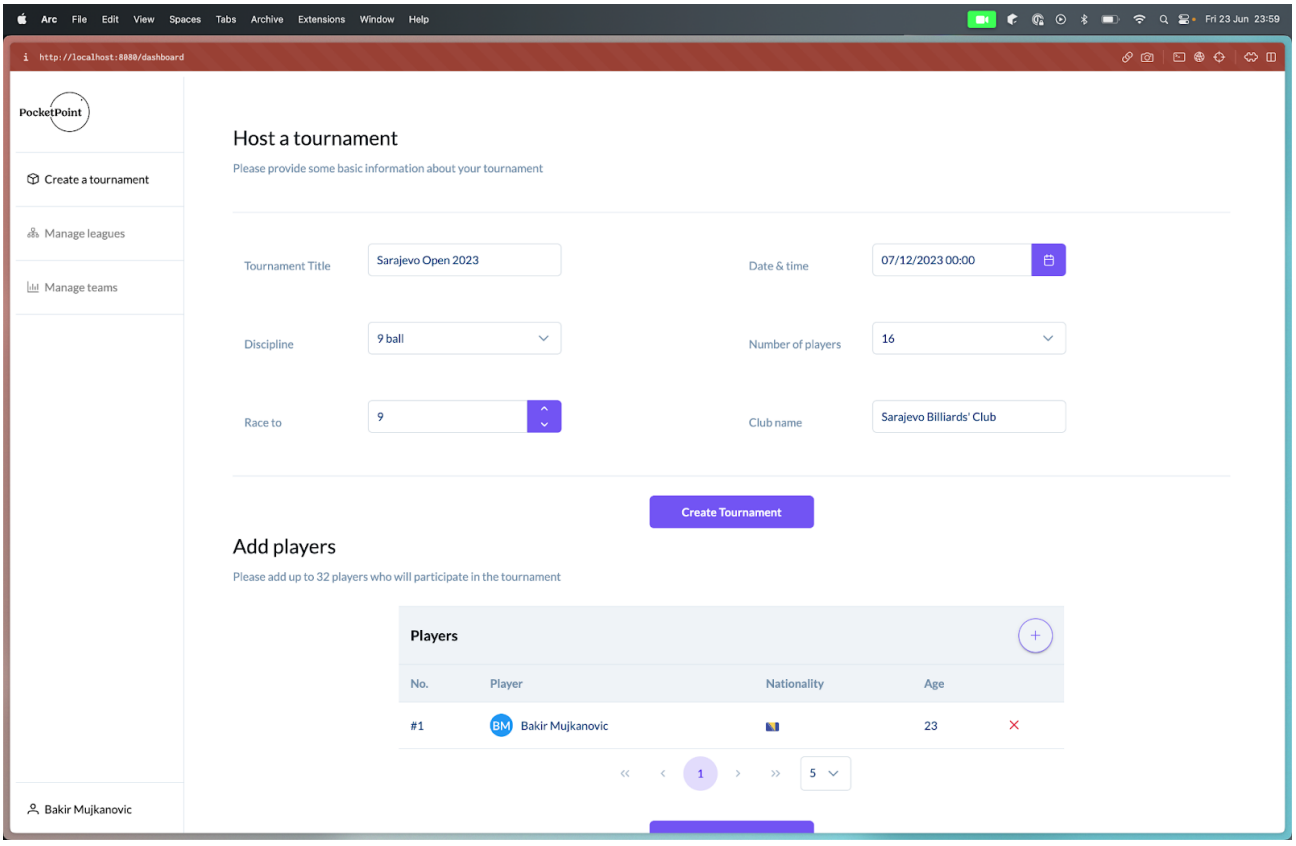
Sign in/Sign up process

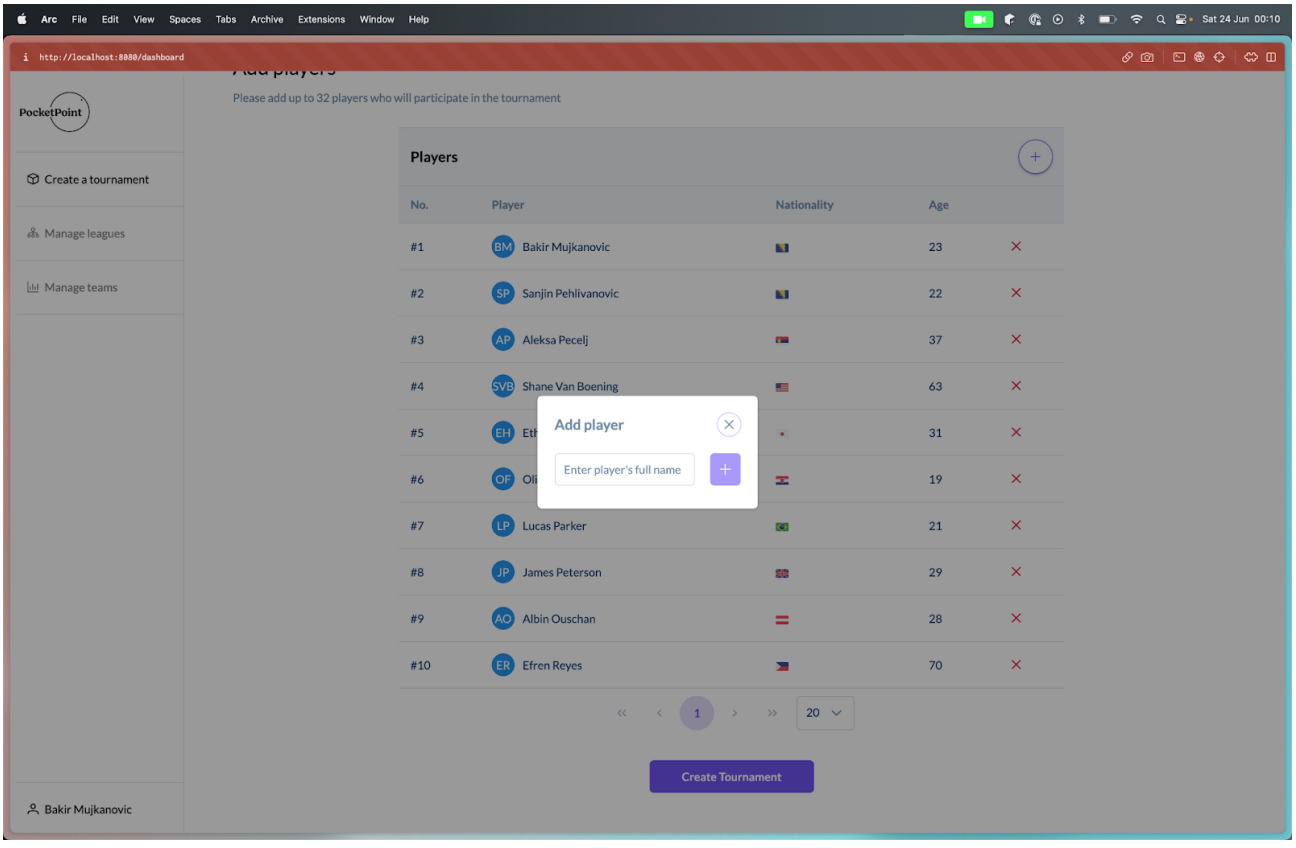



Arc File Edit View Spaces Tabs Archive Extensions Window Help
Fri 23 Jun 23:57
http://localhost:8080
PocketPoint
Home
Tournaments
Leaderboards
Sign in
Climb the ranks ar
make a name for y
Join or organise tournaments, battle with some of the be
journey today.
Host a tournament
Sign Up
PocketPoint
Welcome!
Already have an account? Sign in!
Email
bakir.mujkanovic01@gmail.com
User already exists
Password
Confirm password
Next step

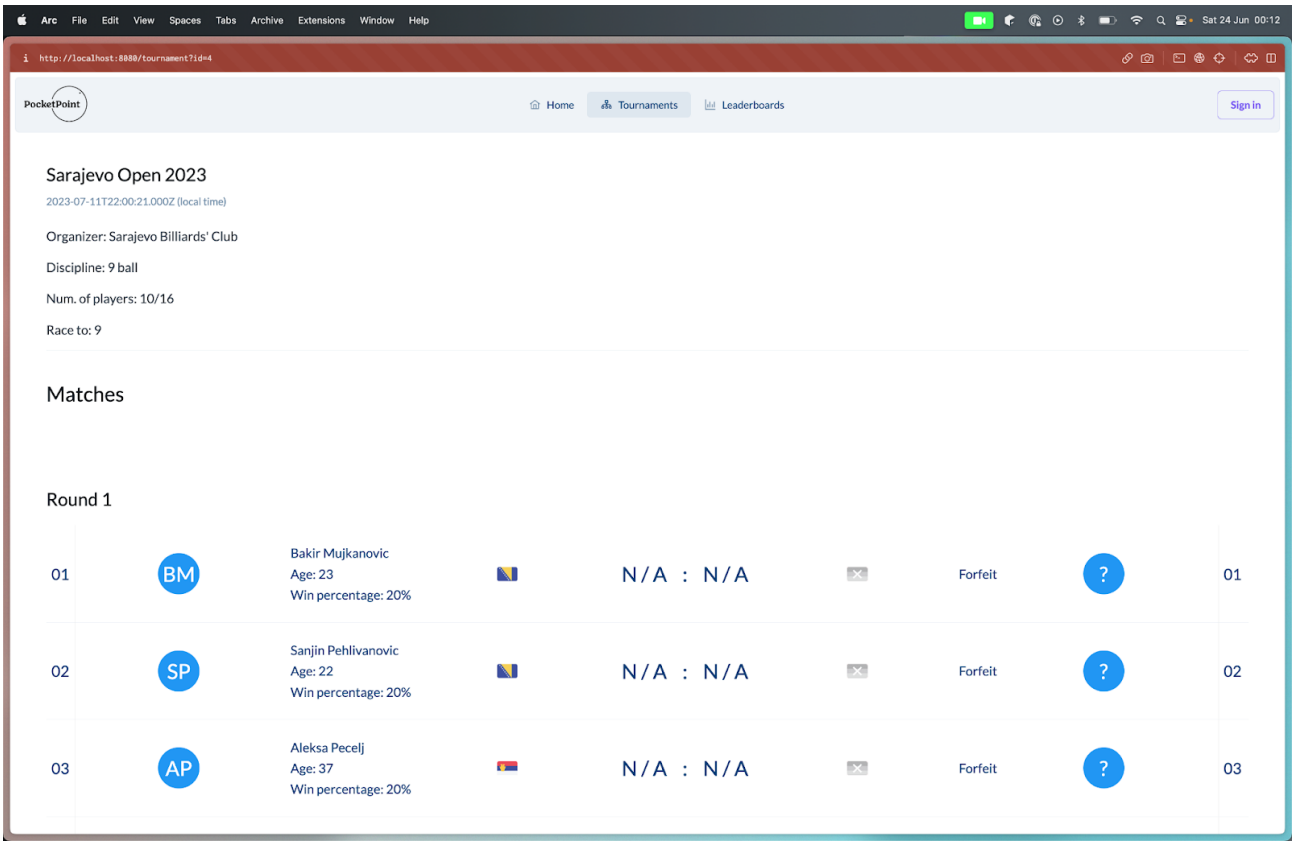

Arc File Edit View Spaces Tabs Archive Extensions Window Help
Fri 23 Jun 23:58
http://localhost:8080
PocketPoint
Home
Tournaments
Leaderboards
Sign in
Climb the ranks ar
make a name for y
Join or organise tournaments, battle with some of the be
journey today.
Host a tournament
Sign Up
PocketPoint
Before continuing..
Tell us a little bit about yourself.
First name
Bakir
Last name
Mujkanovic
Date of birth
02/2000
Nationality
Bosnia and Herzegovina
Sign Up

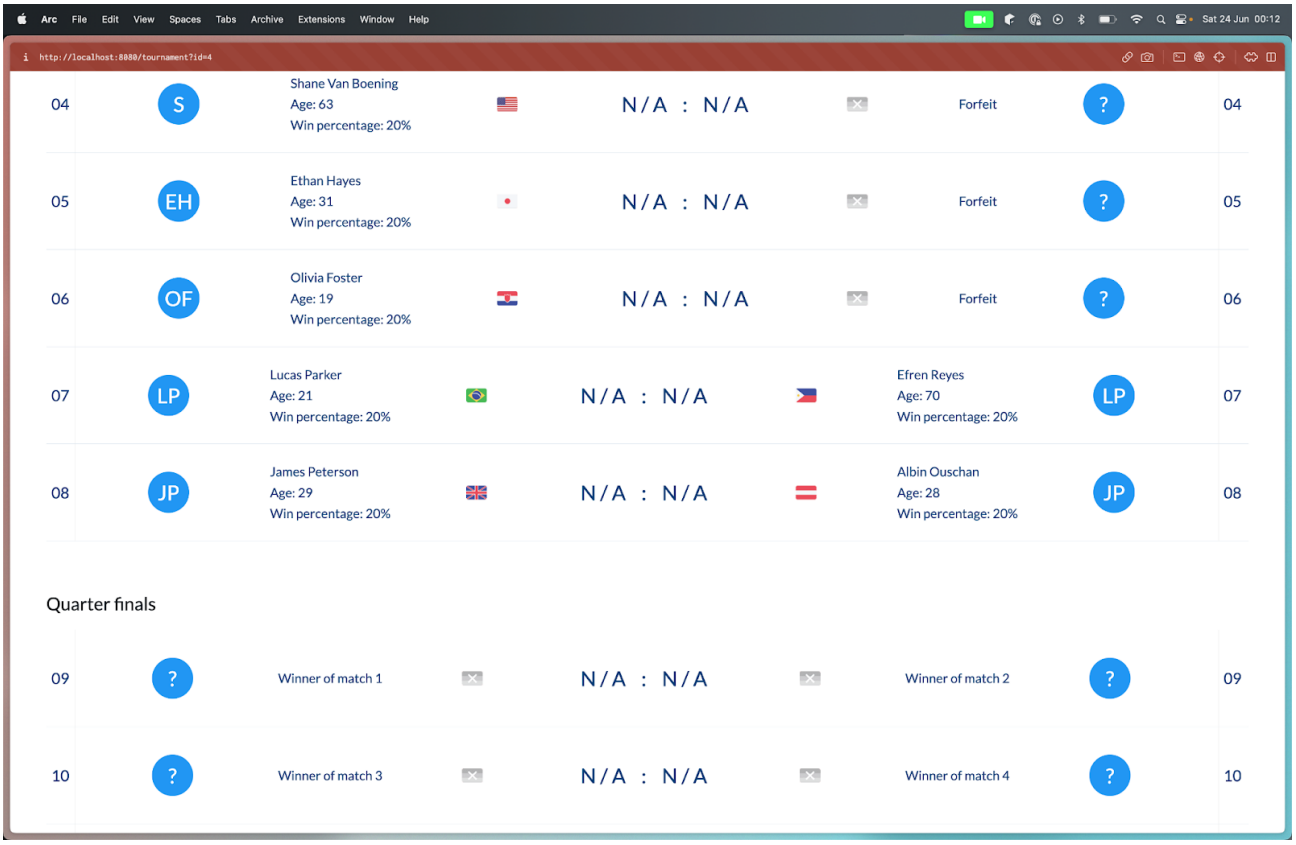
Tournament creation process





Brackets generation





2. Project Structure

2.1. Technologies

Frontend stack:

- HTML/CSS
- Vue.Js with PrimeVue

Backend stack:

- Node.js with Typescript
- Knex ORM
- MySQL

For the frontend side, we stuck to the Options API coding standard for writing Vue.js applications. Since Typescript was not used on the Frontend for simplicity's sake, there was no particular reason to choose the Composition API standard as it would make the project a lot more complex.

On the backend, we decided to use Typescript knowing very well it would take significantly more time to develop - which is why we ultimately didn't deliver on all intended features. Regardless, we decided using Typescript was still worth it since we expected the backend logic to get complex and it seemed worth the trade off.

2.2. Database Entities

Entities:

- Users
- Tournaments - Main entity, contains many Rounds
- Rounds - Contains multiple matches, ex. Semi-finals round
- Matches - Holds match information such as the score

2.3. Design Patterns

Since we decided to go the functional programming route on the backend for this project, it's not immediately obvious when some of the design patterns are used, since they are most easily noticeable when used with an OOP approach.

However, if we focus on some of the general principles of the patterns, we can identify some examples of where such patterns have been used.

For example:

The Singleton design pattern - Ensures that only one instance of a class exists and provides global access to it.

While it's not a class per-say, we do create only one instance of a Database connection and provide global access to it throughout the entire application. It's located inside the ./lib/db-client.ts file where the following two lines allow access to the same connection:

```
const db = knex(dbConfig);
export default db;
```

Whenever a request to the database needs to be made from one of the Services, this 'db' object is imported from the db-client.ts file and used exclusively to communicate with the database.

Another example of a design pattern being used in our application is the Chain of Responsibility Pattern.

Although not explicitly defined in the code, the errorMiddleware function resembles the Chain of Responsibility pattern. It exposes a series of handlers (handleDbError and handleDefaultError) where each of them can be used by the Service to handle different error types. If one of the Services throws an error, it's caught inside the Router files which then make use of the middleware to properly handle the Error.

Error thrown in the user Service

services/userService.ts

```ts
export const getByEmail = async (email: string): Promise<ValueError<User>> => {
  const [user, err] = await handlePromise(db(USER_TABLE).where(USER_COLUMNS.email, email).first());

  if (err !== null) {
    return [null, dbError(err)];
  }
  if (user === undefined) {
    return [null, messageError("User doesn't exist")];
  }

  return [user, null];
};
```

Error caught by the router and passed onto the middleware

routes/user/getByEmail.ts

```ts
const [user, err] = await UserService.getByEmail(email);
if (err !== null || user === null) {
  return next(err);
}
```

Error handled by the errorMiddleware

middleware/error.middleware.ts

```
const handleDbError = (err: ErrorResponse) => {
  // eslint-disable-next-line no-console
  console.error('DB Error', err.error);

  return {
    error: { message: 'Database error' },
    data: null,
  };
};
```

This pattern allows for flexible and decoupled error handling, as different handlers can be added or modified without affecting the overall error handling process.

2.4. Tests

The tests we implemented are located under ./src/services/user/userService.test.ts.

These tests utilize the Jest testing framework to verify the functionality of specific functions in the User Service. The tests cover the getById, getByEmail, and deleteById functions and validate their expected behavior.

In the getById test, it ensures that the getById function retrieves the user with the specified ID from the database. It mocks the database client (db) to return the expected user object and checks if the result matches the expected output.

Similarly, the getByEmail test verifies that the getByEmail function retrieves the user with the specified email from the database. It uses the mocked database client to simulate the retrieval of the user object based on the provided email and asserts the expected output.

The deleteById test ensures that the deleteById function deletes the user with the specified ID from the database. It sets up the mocked database client to return the number of deleted rows, and the test asserts the expected output.

These unit tests provide confidence in the correctness of the userService module by validating its core functionalities. By covering different scenarios and edge cases, these tests help identify potential issues and prevent regressions when making changes to the codebase. They can be executed as part of the automated test suite.

3. Conclusion

We learned a lot trying to develop such a backend-intensive application in quite a limited time span. While it's worth choosing a strict-type language on the backend in the long-run, perhaps it might have been smarter to keep it simple for projects like this. Generally, we would have loved to implement Match updates functionality, work on responsiveness more and try to implement leagues and teams. We would have loved to play with the idea of implementing WebSockets so that the users viewing a certain tournament could see the match updates real time.

Unfortunately, we started a bit and ran into some unforeseen complications and time delays. Generally, it was an enjoyable experience developing the app but we are disappointed we didn't have the chance to present it in the way that we had hoped.