POPA EDUARD ȘTEFAN
Ultimate EURO 2024 Manager

Functionalities description

The implemented functionalities are presented below:

- the possibility to create a new account or to login if the user has registered before;
- the selection of one country participating in the knockout rounds (from that moment the selected team will be the user's team);
- the generation of a new random draw everytime a new game begins;
- the presentation of the selected team's members with all their details to ensure a better choice of the participating players in order to win the round;
- if the user's team wins, the draw will be updated in the web page and the user will select once again the team's members until the team is eliminated or it wins the tournament;
- in the case of winning/losing a proper message will be displayed;
- the game will not start running if the user doesn't specify 1 player as a goalkeeper, 4 players as defenders, 4 players as midfielders and 2 players as attackers for the team;
- the restriction of access to certain pages if some conditions are not met, for example:

1. if the user is not yet logged in and they try to enter another URL in address bar, other than those specific for login/register pages, then the login/register web resource will be loaded;
2. if a user has begun a game and hasn't finished it yet and tries to enter the page which presents all the countries, then they will be redirected back to the draw page in order to finish the game;
3. if a user currently hasn't an active game, then the access to the draw page will not be permitted until they select a country;

Technology stack

I have developed a full stack solution in order to solve this problem. The graphical interface was implemented by using the **React library** and a few additional npm packages. The web client runs on port 3000 and sends HTTP requests to the server and renders the received response data in an organized manner. For the backend side I made use of the **Node.js** platform based on JavaScript programming language. The server runs on port 8080 and receives HTTP requests from the client in order to perform a certain task involving the access to the database. All the users and games data is stored in a **SQLite relational database** which is stored locally.

Project’s installing requirements

In order to be able to run the solution, the hosting system must have installed Node.js and npm (Node Package Manager). Both of them can be installed simultaneously from the Node official website (**https://nodejs.org/en/download/package-manager**) by downloading and executing the kit.

To run the backend server, use the next command in the “backend” folder: **node index.js**

To run the client, use the next command in the “frontend” folder: **npm start**

Below are presented the installed packages which are required to ensure both backend and frontend functionalities:

Backend project

- SQLITE package (for database management) - **npm install sqlite3 --save**
- Express package (to implement a HTTP backend server) - **npm install express --save**
- Body Parser package (to parse the incoming HTTP requests from the client side) - **npm install body-parser**
- CORS package (to permit the acceptance of HTTP requests coming from another domain, as the frontend and backend runs on different ports) - **npm install cors**

Frontend project

Important: Firstly, please run the command **npm install** (to restore the “node_modules” folder)

- MUI Material package (offers GUI components) - **npm install @mui/material @emotion/react @emotion/styled**
- React Router package (to ensure the passing from one page to another) - **npm install react-router-dom**

Solution’s architecture

The backend project (Node.js)

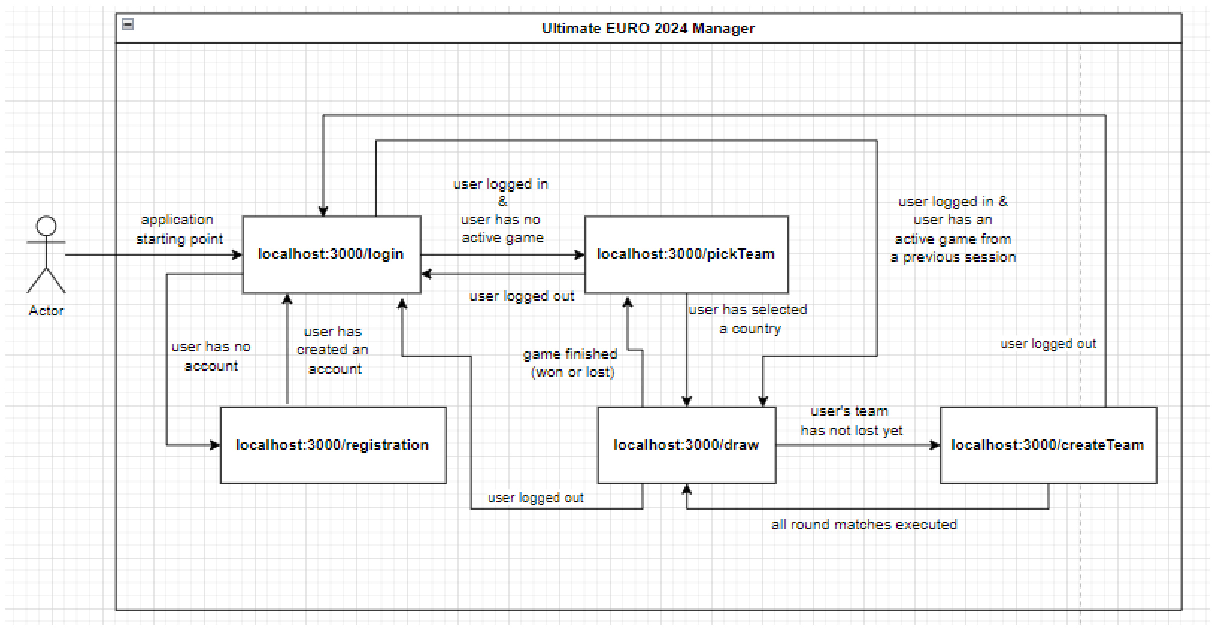
The backend project consists of the next structure:

- node_modules folder – has all the required packages in order to run the application;
- connectDB.js file – the script responsible for the connection with the database;
- controller.js file – the script with the database manipulation functions;
- databaseEURO2024.db file – the database file with all the stored information;
- index.js file – the starting point of the application;
- package.json / package-lock.js files – configuration files;
- router.js file – the file with the mapping of the endpoints and their actions;
- validator.js file – file with middleware functions that ensure the data consistency before accessing the endpoint (do not allow the registration of two users with the same username);

The backend server’s implemented endpoints are:
POST /registration – to insert a new user in the database;
POST /login - to authenticate into the application;
POST /game - to insert a new game registration in the database;
PATCH /game/:id - to update the game state attribute (set the game as finished) for the game with the ID specified in the parameter;
GET /game - to display all the games regardless of their status;
POST /draw - to insert the latest updates of the draw after a round of matches has finished;
GET /draw/:id - to display the current state of the draw for the current player specified by the ID present in the parameter;
GET /draw - to get all the draws evolutions for all the games regardless of the user;
PATCH /draw/:id - to update the score attribute after a round of matches for the row with the ID specified în the parameter;
GET /player/:country - to get all the players competing for a specified country;

The frontend project (React)

After opening the web application the user will be able to access the following pages:
localhost:3000/login – the page has a form containing 2 inputs for username and password;
localhost:3000/registration – the page has a form containing 3 inputs (username, password and repeat password - the last two inputs have to be equal in order to proceed with a request to the backend server) for creating a new account ;
localhost:3000/pickTeam - a list of 16 countries is presented in order to select one for the tournament;
localhost:3000/draw - the initial random draw of 16 countries is presented and it evolves as the game goes on and the scores are updated;
localhost:3000/createTeam - a page in which a table of all the players playing for the selected country is presented along with 4 dropdown lists which have to contain players for the all 4 specified roles in the team;

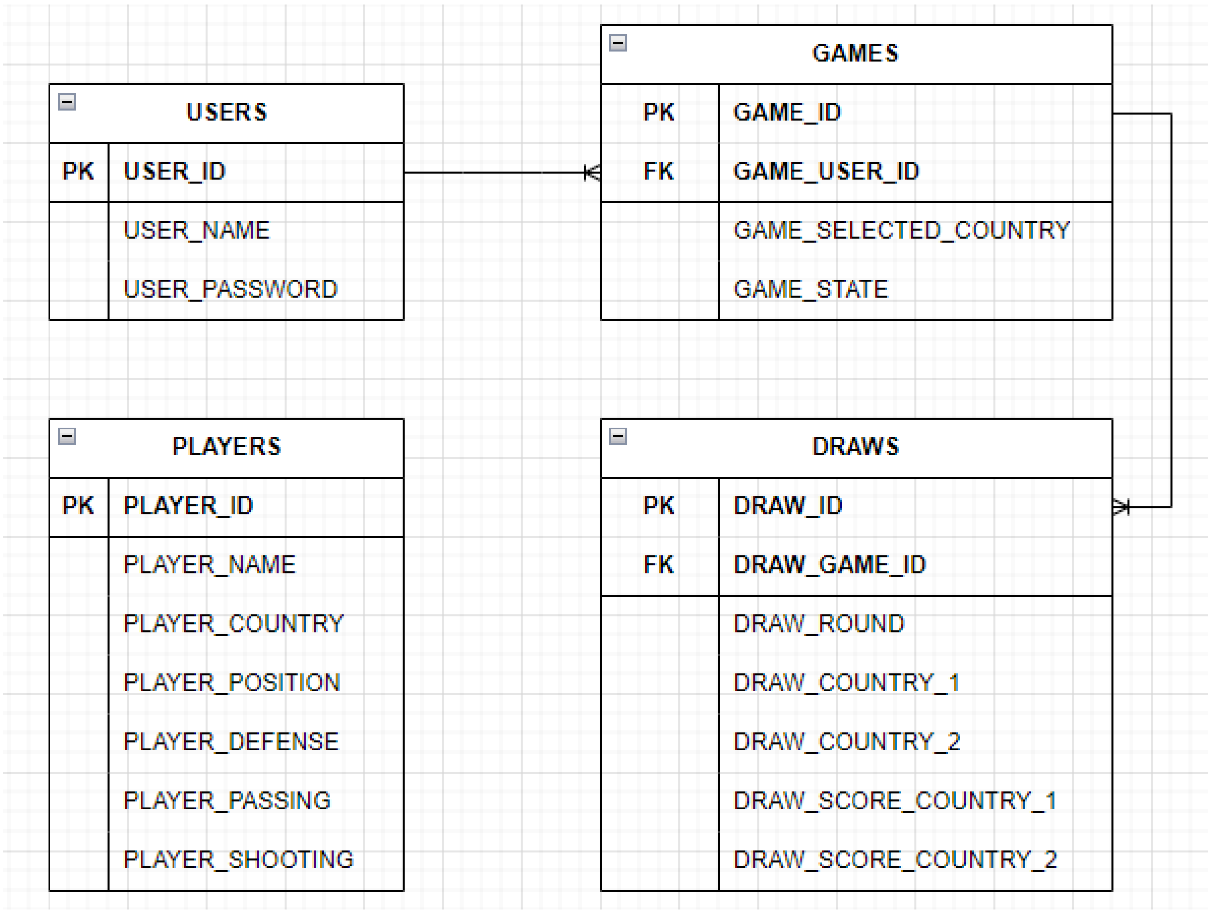

Ultimate EURO 2024 Manager
Actor
application starting point
localhost:3000/login
user logged in & user has no active game
localhost:3000/pickTeam
user logged in & user has an active game from a previous session
user logged out
user has no account
user has created an account
user has selected a country
game finished (won or lost)
user logged out
localhost:3000/registration
localhost:3000/draw
user's team has not lost yet
localhost:3000/createTeam
user logged out
all round matches executed

The routing flow presented in the above diagram shows how the user is redirected from one web page to another according to some conditions. A user cannot access the "/pickTeam" resource to choose a country with which is going to compete if there is a game linked with the current user which has the status "Started". A check is made every time a user is logging in the application to see if there is such an entity. After a game is started, a user can swing only between the "/draw" and "/createTeam" pages until the game is over or the user chooses to log out.

The database (SQLite)

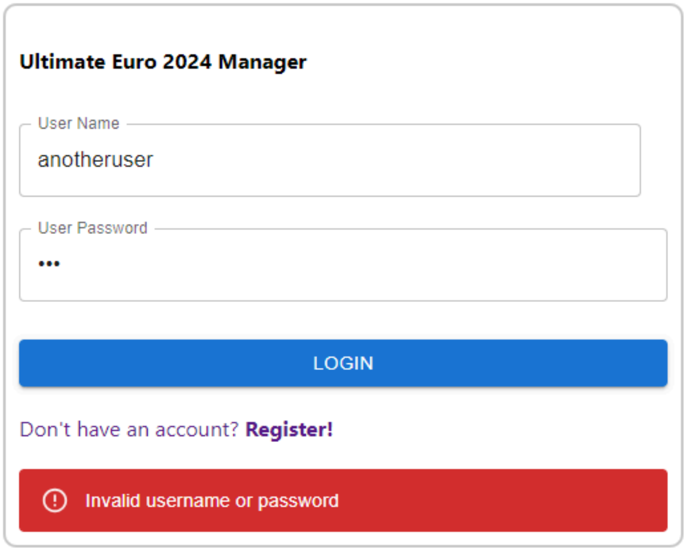
The SQLite database is composed by 4 tables:

1. **Users** - contains the data required for users in order to log in the web application;
2. **Players** - contains all the necessary information about all the participating teams players that will be used to compute the scores for the matches;
3. **Games** - a user can have only one active game (reflected in the GAME_STATE attribute) associated with their own user ID;
4. **Draw** - a game has a draw. A row in the Draw table corresponds to a single match and contains the 2 contesting teams and their scores.



The "Games" table acts as an intermediate entity between a certain user and the generated draw of the game which was started by that user. The "Games" table is correlated with a user by the use of the foreign key field "GAME_USER_ID" whose value is found in the “USER_ID” field from the “USERS” table. Also, the draw of a competition is associated with only one game entity by the attribute “DRAW_GAME_ID”.

Solution’s interface

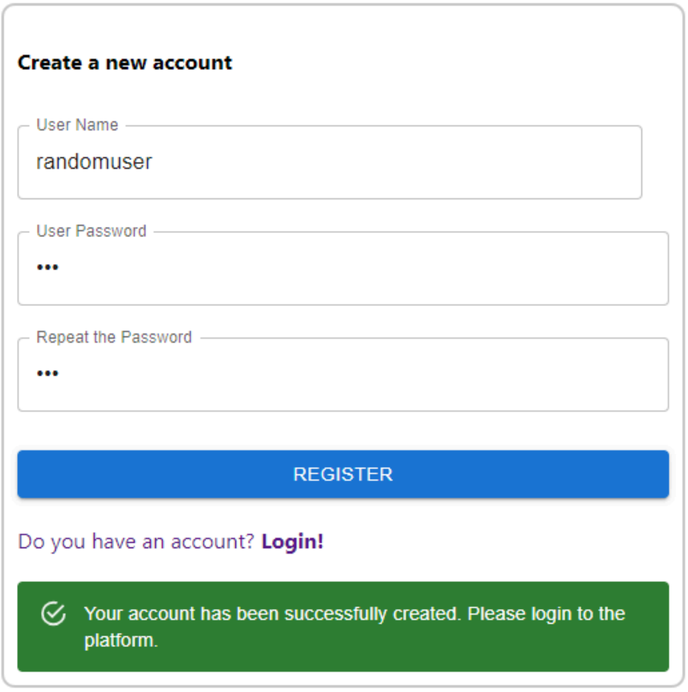


anotheruser

•••

Invalid username or password

localhost:3000/login



randomuser

•••

•••

Your account has been successfully created. Please login to the platform.

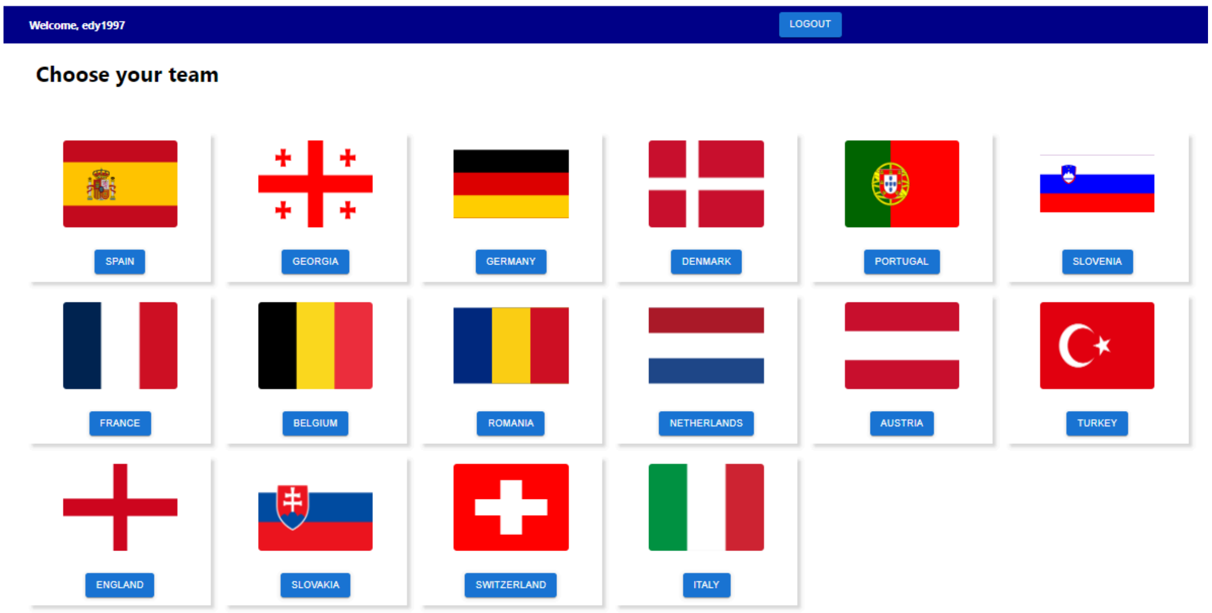
localhost:3000/registration

When a user opens the web application, the first page that is presented is the login page. Firstly, the opening of a new account is necessary if the user doesn’t own one already. Some alerts are implemented in the case that the user offers wrong date, for example:

- if the user wants to create an account with a username that is already taken by another user;
- if the user wants to create an account and the fields “user password” and “repeat password” are not equal;
- if a user already has an account and completes the fields with the wrong credentials;
- if the user creates a new account and all the passed values are valid;

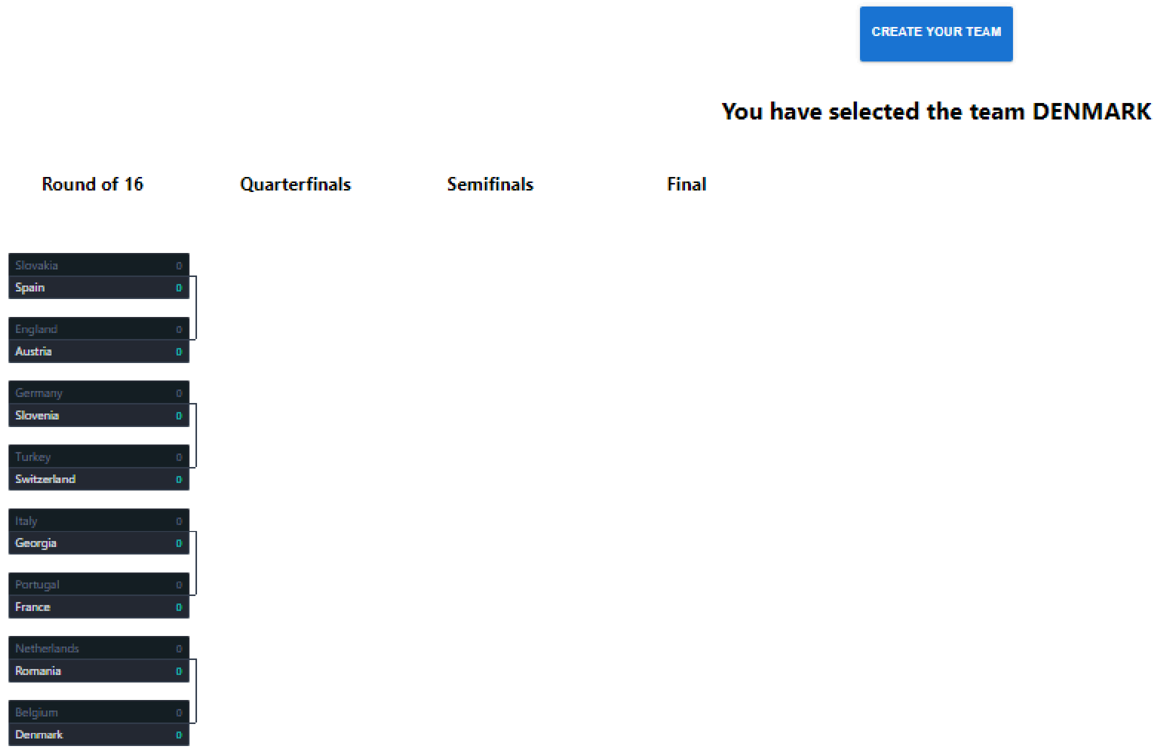
If the user moves on and they haven’t an associated game which is not finished, then the “/pickTeam” page is shown. There is a list of all the 16 contesting countries along with their flags. The name of the countries are integrated within a button whose action is to set the name of the selected country in order to be accessed during the data flow.

Also, if a user has logged in then in every interface there will be a top navbar showing the username of the logged user and a “Logout” button which will delete the current session.

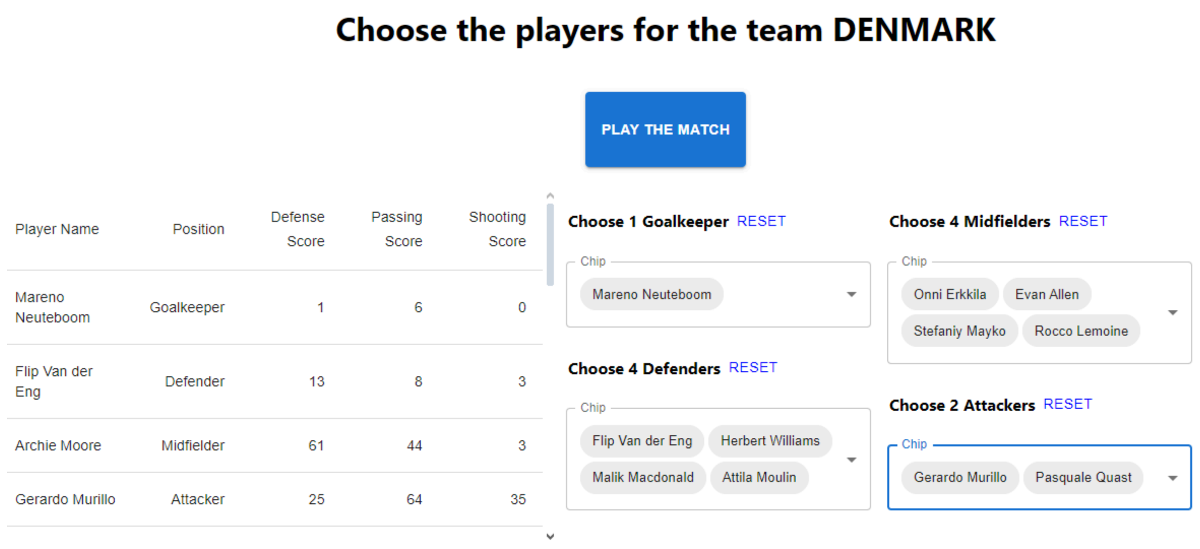


localhost:3000/pickTeam

After picking up a country, a random draw will be generated and shown in a web where the user will be redirected to. In this “/draw” page will be specified the team that was chosen by the user. Here, as the game evolves, the draw will be presented in its entirety. The game has 4 rounds: round of 16, quarterfinals, semifinals and the final. Above the draw, there is a “Create your team” button which will send the user to a page where they can select the players for the chosen country.



localhost:3000/draw



Choose the players for the team DENMARK

Player Name	Position	Defense Score	Passing Score	Shooting Score
Mareno Neuteboom	Goalkeeper	1	6	0
Flip Van der Eng	Defender	13	8	3
Archie Moore	Midfielder	61	44	3
Gerardo Murillo	Attacker	25	64	35

Choose 1 Goalkeeper RESET
Chip: Mareno Neuteboom

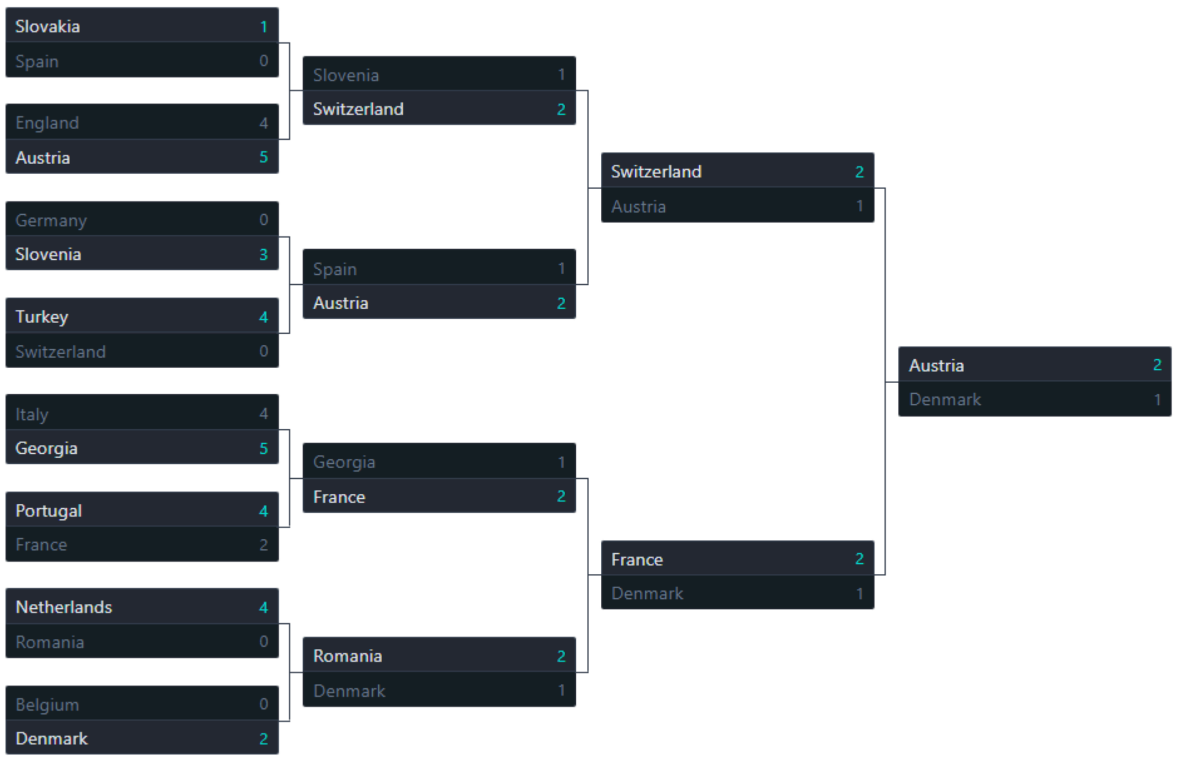
Choose 4 Defenders RESET
Chip: Flip Van der Eng, Herbert Williams, Malik Macdonald, Attila Moulin

Choose 4 Midfielders RESET
Chip: Onni Erkkila, Evan Allen, Stefaniy Mayko, Rocco Lemoine

Choose 2 Attackers RESET
Chip: Gerardo Murillo, Pasquale Quast

localhost:3000/createTeam

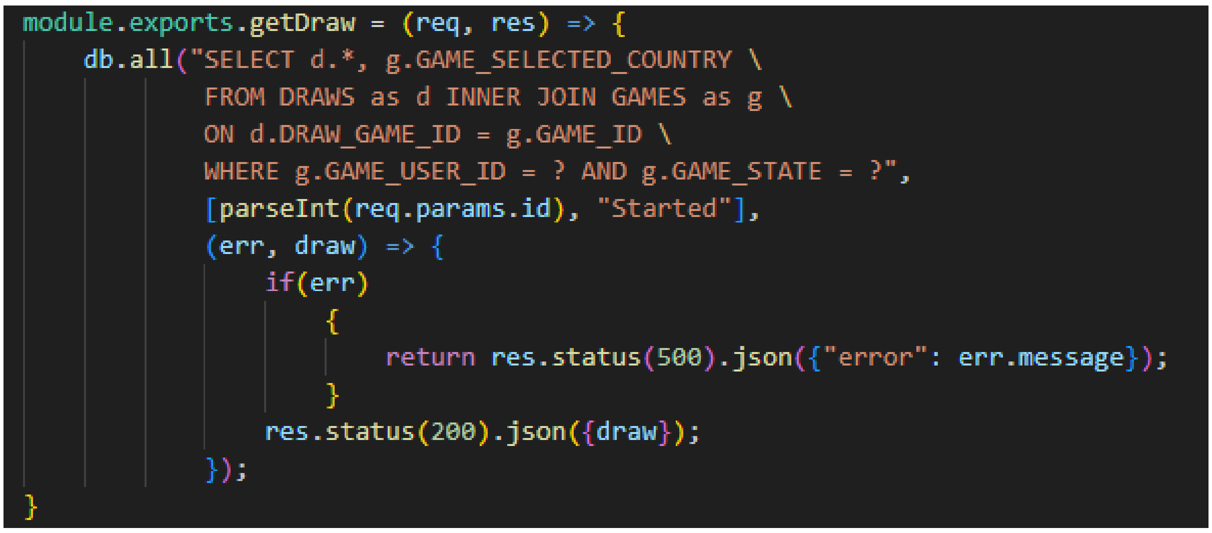
For each round of the tournament, the user has to select 11 players for the selected team (1 goalkeeper, 4 defenders, 4 midfielders, 2 attackers). In the left part of the page is a table with all 22 players competing for the selected team and their attributes in order to give the user a chance to make a good selection. If any of the 4 dropdown lists don't meet the requirement of the number of selected players for the respective role (there is no check performed to see if the player chose the players correctly according to their role in the team), then an alert will be released when the button "Play the match" is clicked. If the requirement is respected then the user will be redirected to the draw page with the updated results of the current round. And these steps are repeated until the user wins or loses the game.



Implementation details

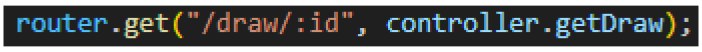
Endpoint implementation

An endpoint is a backend server's resource that maps a function which is executed when the endpoint receives a HTTP request from the client side (frontend). For example, in the capture presented below is implemented a function which executes a SQL query. This is responsible for searching all the draws for an active game which is associated with a user. The function receives 2 parameters, a request and a response object. The query has 2 variables indicated by the question marks signs. These variables will take their value from the second parameter of the db.all() call, which is an array of 2 elements. The first element is the ID of the user for which the SQL search is made and the second element is the status of the game. All the rows returned by the query will be sent in JSON format as a response to the client who made the request.



```
module.exports.getDraw = (req, res) => {
    db.all("SELECT d.*, g.GAME_SELECTED_COUNTRY \
            FROM DRAWS as d INNER JOIN GAMES as g \
            ON d.DRAW_GAME_ID = g.GAME_ID \
            WHERE g.GAME_USER_ID = ? AND g.GAME_STATE = ?",
            [parseInt(req.params.id), "Started"],
            (err, draw) => {
                if(err)
                    {
                        return res.status(500).json({"error": err.message});
                    }
                res.status(200).json({draw});
            });
}
```

In order to connect the function to an endpoint (in this case the endpoint is "localhost:8080/draw/:id) it is needed to make use of a Router object provided by the Express package which is already installed. The Router object covers all the HTTP methods which are specified as the name of the function as it is shown below. The first argument is the name of the endpoint which, also, specifies a parameter (variable) passed through the resource. The second one is the name of the function which was described.

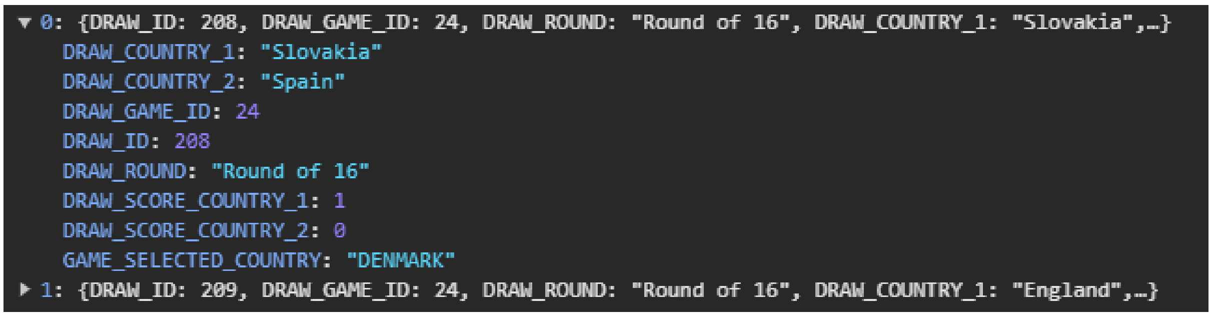


```
router.get("/draw/:id", controller.getDraw);
```

This endpoint is used to check if the logged user has an unfinished game in order to not allow the generating of a new one.

Persistence, user management and sessions

The progress of the game is saved in the database after each round is played. If a user logged out before the game is finished, then they can continue to play after logging back in. This is where the function explained in the previous page is involved. In this case, the progress of the game and the selected country is saved in local storage in order to be accessed by multiple web pages. The progress of the draw is stored as an array of objects, like it is shown in the next capture.

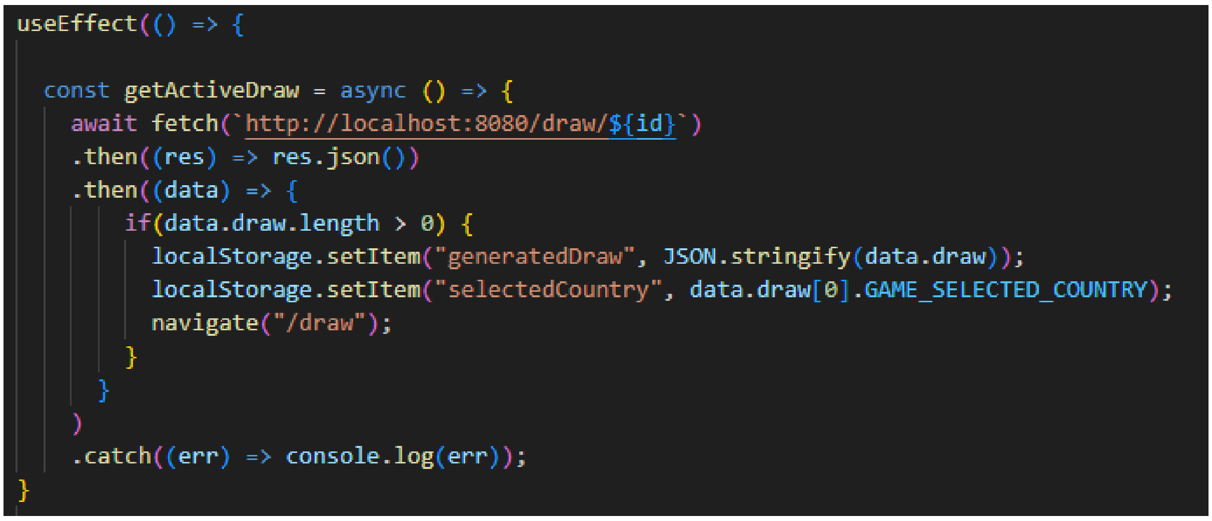


```
▼ 0: {DRAW_ID: 208, DRAW_GAME_ID: 24, DRAW_ROUND: "Round of 16", DRAW_COUNTRY_1: "Slovakia",…}
    DRAW_COUNTRY_1: "Slovakia"
    DRAW_COUNTRY_2: "Spain"
    DRAW_GAME_ID: 24
    DRAW_ID: 208
    DRAW_ROUND: "Round of 16"
    DRAW_SCORE_COUNTRY_1: 1
    DRAW_SCORE_COUNTRY_2: 0
    GAME_SELECTED_COUNTRY: "DENMARK"
▶ 1: {DRAW_ID: 209, DRAW_GAME_ID: 24, DRAW_ROUND: "Round of 16", DRAW_COUNTRY_1: "England",…}
```

An array element contains data about the name of the two countries which are due to play or played the match, the round in which the match takes place and the scores for each country. The “GAME_SELECTED_COUNTRY” attribute is not a part of the draw entity. It comes from the joining of the Draws and Games tables to get the name of the country which was selected by the user for the current game. This piece of data is important in order to send a request to query the database for all the users of the selected country.

There is no need to store the local storage data after the user logs out (the log out function actually clears all the local storage), as it will be provided again when the user decides to log in.

Each user is delimited from the other ones, as the database is modeled in such a way that each game is linked with a single user.



```
useEffect(() => {

  const getActiveDraw = async () => {
    await fetch(`http://localhost:8080/draw/${id}`)
    .then((res) => res.json())
    .then((data) => {
        if(data.draw.length > 0) {
          localStorage.setItem("generatedDraw", JSON.stringify(data.draw));
          localStorage.setItem("selectedCountry", data.draw[0].GAME_SELECTED_COUNTRY);
          navigate("/draw");
        }
      }
    )
    .catch((err) => console.log(err));
}
```

In the page that follows the login page, as the page is loaded, a check is performed to see if a player has an active game started before. The useEffect hook in React is used to perform an action according to a dependency, which is a variable or set of variables. If the hook finds a change of the dependency value, then it will trigger the action. In this case there is no dependency specified, so the hook will be triggered at the page loading. The action consists of sending a GET HTTP request to the “localhost:8080/draw/:id” endpoint. In the case of which the data returned by the server has at least one row / element, the local storage variables for the draw evolution and the selected country are setted and the user will skip the page for selecting a country and will be redirected to the draw page.

Score generating algorithm

The way the score is computed is as it follows:

1. after completing the name of the selected players in the “/createTeam” page, a search will be performed in order to create an array with the complete information of the selected players and the number of role mismatches will be calculated;
2. for every element of the draw (equivalent of a match) which belong to the current round:
    1) if one of the teams are the selected team then players team information will be stored in the local storage, and for the other one a call to the endpoint that returns all the players for a specified country will be made and its results will be stored in local storage;
    2) for the teams that differ from the selected country, a reduction of the team will be performed to reach from 22 players to 11 (the array will be shuffled randomly and then will be parsed until an array of 1 goalkeeper, 4 defenders, 4 midfields and 2 attackers will be created)
    3) for both teams, the scores of the players attributes will be added;
    4) if one country is the selected one, then the score will be penalized by the next formula (for 0 mismatches the score will not be affected):

    $$new\ score\ = (1\ - \frac{number\ of\ role\ mismatches}{1 + number\ of\ role\ mismatches})\ \times\ current\ score$$

    5) the country with the higher score will get a match score according to the ratio of the two values;
    6) a PATCH HTTP request will be made in order to update the score for the match;
3. after parsing the matches array, a GET HTTP request will me made to retrieve the draw with the updated scores;
4. an array of winners will be creating after comparing all the match scores;
5. for all the elements of the winners array, a POST HTTP request will me made to create new matches for the winning teams (by default the scores are all equal to 0);