Web Applications A.Y. 2023-2024

Homework 1 – Server-side Design and Development

Master Degree in Computer Engineering

Deadline: 29 April, 2024

Group Acronym	**DAM**	
Last Name	**First Name**	**Badge Number**
Basaglia	Alberto	2119289
Bruttomesso	Andrea	2120933
Corrò	Alessandro	2125034
Popovic	Milica	2119069
Seghetto	Davide	2122548
Stocco	Andrea	2108885

1 Objectives

The primary objective of our web application is to provide a comprehensive platform for organizing football tournaments and following match schedules, results, and rankings effortlessly. Our solution offers an opportunity for tournament organizers to take care of tournament management, allowing them to create and set up tournaments, schedule matches, and manage teams and players. On the other hand, the basic users can easily access to tournaments information, such as match schedules, results, and statistics.

2 Main Functionalities

Our system accommodates three distinct user types:

- Guest: a non-logged-in user.
- Team Administrator: a registered user responsible for managing his own teams.
- Tournament Administrator: a registered user responsible for managing his own tournaments. Additionally, tournament administrators can manage the teams enrolled in their tournaments.

A registered user may hold ownership of both tournaments and teams. Every of those users can view past and present tournaments along with their related match schedules, team standings, and other statistics.

More in-depth, the main functionalities are the following:

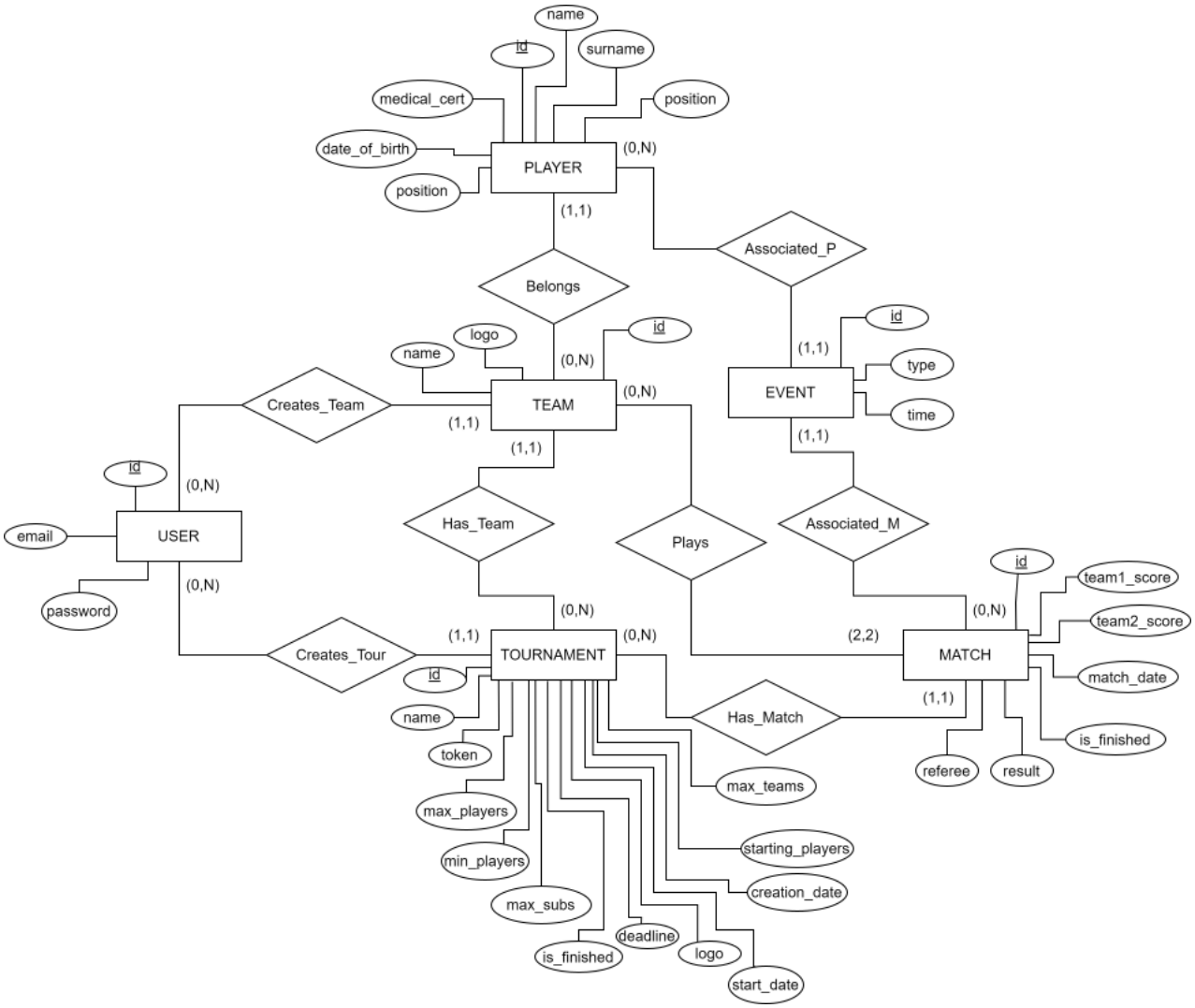
- **Creation of a new tournament:** This functionality offers the opportunity to each logged in users to create a football tournament (and become its administrator). On the main page of the web app, it is possible to create an account which is required for the tournament creation. On the other hand, if the user already has an account he can proceed with the login step. After successfully completing the sign up or sign in process, there are other steps which should be taken such as providing basic data about the tournament (name, logo, deadline for the team creation, number of starting players, maximum number of players, maximum number of teams, etc.). Upon providing all essential data, the tournament is created and saved into the database.
- **Edit/deletion of a tournament:** This feature enables a tournament administrator to edit data of the tournament or delete it.
- **Creation/edit/deletion of a team for the tournament:** Once the user is logged in, he can select the tournament of his interest in order to enroll his team by providing all data needed, such as a team name, logo, and players. Only the tournament and team administrator are allowed to edit the team data or delete the team.
- **Addition/edit/deletion of players:** Only the tournament and team administrator are allowed to add, edit, or delete players to/from the team.
- **Creation of the draw:** Tournament administrators have the ability to make a draw for the tournament. There are two options for making a draw:
  1. After the team creation deadline, the system will automatically make the tournament draw at midnight. This draw will then be available on the page.
  2. If the tournament administrator deems the tournament ready to start, he can choose to manually close subscriptions, at which point the system will promptly generate the matches.
- **Updating the info of a tournament's matches:** Only the tournament administrator is able to insert the result of a match that has been played, along with the main events like the scorers and the sanctions (yellow/red cards).

- **Following the tournament's matches:** Another feature of the webapp is displaying results of tournament matches and current statistics like team standings and top scorers' ranking. Users have the option to filter and view both concluded and upcoming matches. That feature is available for every active and past tournament, after choosing the particular tournament from the app's main page.

3 Data Logic Layer

3.1 Entity-Relationship Schema

The following ER schema contains six entities which are going to be described in details in this section.



1. **User:** Entity Type User contains ID (INT), Email (CHAR) and Password (CHAR) for every user. The primary key is ID, while Email is a unique column. Every user can create one or more tournaments as well as one or more teams.

2. **Tournament:** Entity Type Tournament contains ID (INT), Name (CHAR), Token (CHAR), Creator_User_ID (INT), Max_Teams (INT), Max_Players (INT), Min_Players (INT), Starting_Players (INT), Max_Substitutions (INT), Deadline (DATETIME), Start_Date (DATETIME), Creation_Date (DATETIME), Logo (BYTEA) and Is_Finished (BOOL). The primary key is ID, while Creator_User_ID is a foreign key constraint to the User table. Logo contains the file which represents the logo for a particular tournament. Max_Players and Min_Players are maximum and minimum numbers of players for each team in the tournament, while Starting_Players is a number of players in each match of the tournament and defines the type of tournament that is being organized (football, futsal etc.). Deadline represents the date until which teams can be enrolled. Starting_Date is the date of the first match, while Creation_Date indicates the date of the creation of the tournament. Is_Finished indicates whether a tournaments is over or not. Each tournament must be created by exactly one user and has zero or more teams and matches.

3. **Team:** Entity Type Team contains ID (INT), Name (CHAR), Logo (BYTEA), Creator_User_ID (INT) and Tournament_ID (INT). The primary key is ID, while Creator_User_ID is a foreign key constraint to the User table and Tournament_ID is a foreign key constraint to the Tournament table. Every Team must belong to exactly one tournament and must be created by a user. This Entity Type can have multiple players and can play multiple matches.

4. **Player:** Entity Type Player contains ID (INT), Name (CHAR), Surname (CHAR), Team_ID (INT), Position (ENUM), Medical_Certificate (BYTEA) and Date_Of_Birth (DATETIME). The primary key is ID while Team_ID is a foreign key constraint to the Team table. A medical certificate must be saved in the database for each player. Every player must belong to a team and can be participant in more events.

5. **Match:** Entity Type Match contains ID (INT), Team1_ID (INT), Team2_ID (INT), Tournament_ID (INT), Team1_Score (INT), Team2_Score (INT), Result (CHAR), Referee (CHAR), Match_Date (DATETIME) and Is_Finished (BOOL). The primary key is ID while Team1_ID and Team2_ID are foreign keys constraint to the respective teams. Tournament_ID is a foreign key constraint to the respective tournament. Every match must belong to only one tournament, have always two teams and can have multiple events associated to it.

6. **Event:** Entity Type Event contains ID (INT), Match_ID (INT), Player_ID (INT), Type (Enum) and Time (DATETIME). The primary key is ID while Match_ID is a foreign key constraint to the Match table and Player_ID is a foreign key constraint to the Player table. Every event belongs to one match and has one Player who made the event. Type of event can be goal, yellow card or red card.

3.2 Other Information

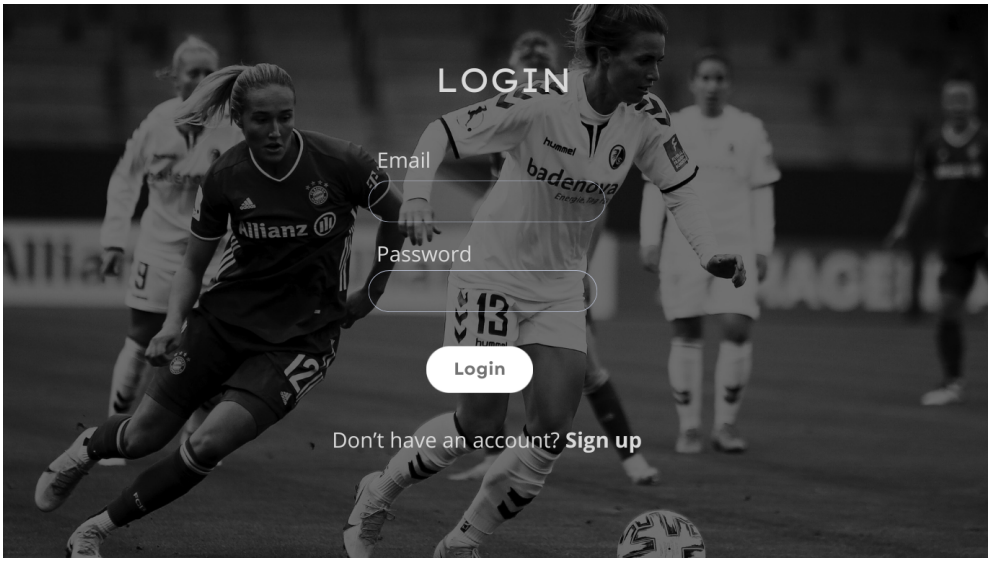
Aside from the tables created in the database, the following three enums are also created: `event_type`, `match_result` and `player_position`. `event_type` is one of the fields in the Events table and it serves to describe a type of event, while `match_result` is one of the fields in the Matches table and it serves to describe whether team1, team2 won or if it was a draw. Lastly, `player_position` is one of the fields in the Players table and it serves to describe a position of each player in a team.

4 Presentation Logic Layer

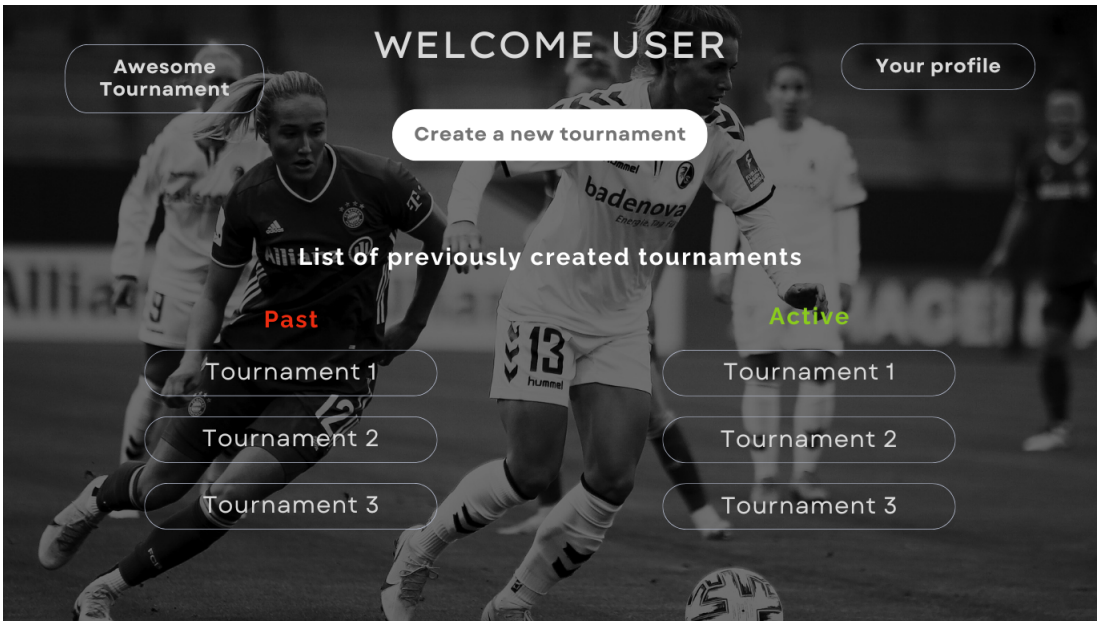
There will be present approximately 10 pages. The default one will be a page showing all the past and active tournaments, that every user can check even if not logged in. A page for the login and the sign up. Once logged in, the user will be able to see also the past and active tournaments that he created. Finally, there will be the pages dedicated to displaying the information and the ones dedicated for the creation of the teams and the tournaments and for the updating of the matches.
Below are some examples of pages implementing the functionalities described before.

- **Homepage:** it displays both past and active tournaments even if the user is not logged in. By clicking on a past tournament all the details are shown (results, both tables, etc.); by clicking on an active one it is displayed the same thing but in real time (to be updated). In the top right corner there is a button that allows the login for the user.

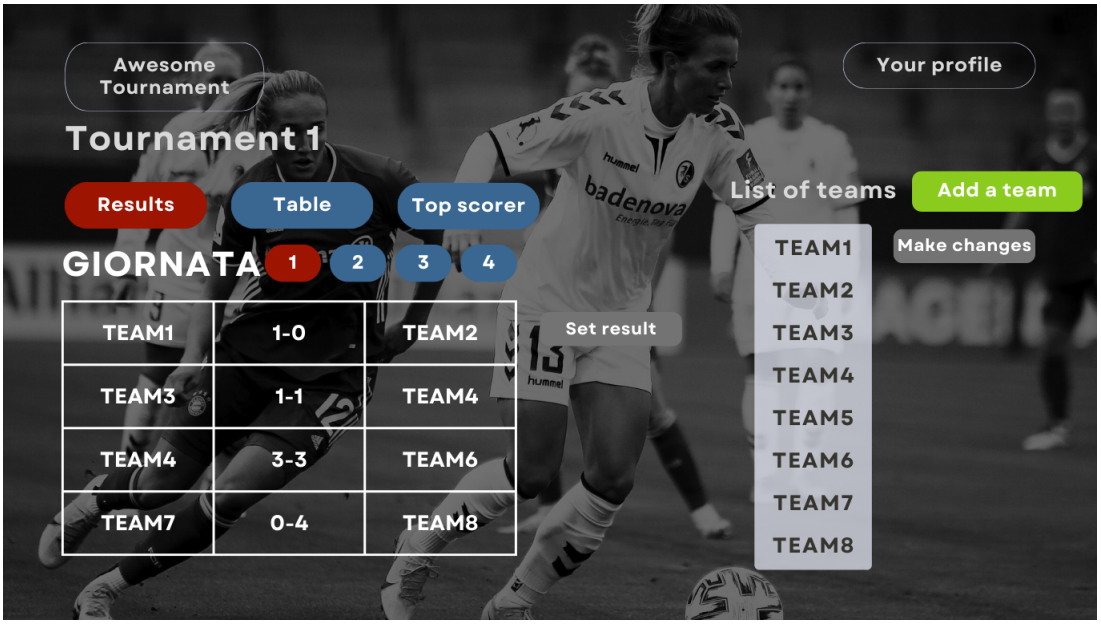
- **Login in page:** it is requested to provide a valid email and a secure password. By clicking on "Sign up" the user can register on the site.



- **Homepage after login:** after the login all the past and active tournaments created by the user are displayed. From here the user can start the process to create a new tournament.



- **Page of a tournament:** the page displays the infos about a tournament. The user can decide what to check from the three button (Results, Table and Top scorer). It is also possible to choose the matchday to check. For the admin of the tournament and the creator of that team it is also possible to manage the team.

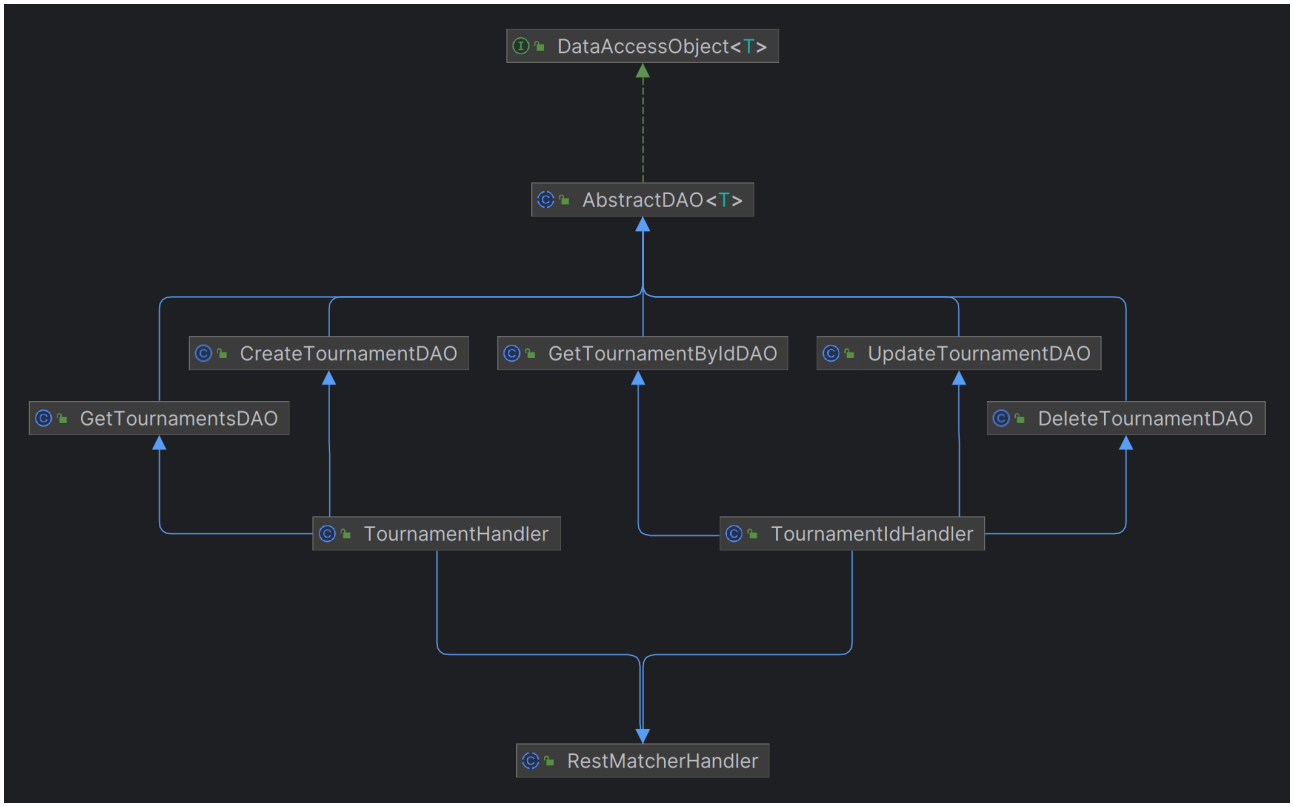


- **Page of a specific match:** the page displaying the infos about a specific match of a tournament. Only the admin of the tournament is allowed to fill in the events for a match by clicking on "update result and scorers".

- **Process of creating a tournament:** the image below display the process of creating a tournament. Only the last required info are displayed, like the starting date, the deadline and the logo.

5 Business Logic Layer

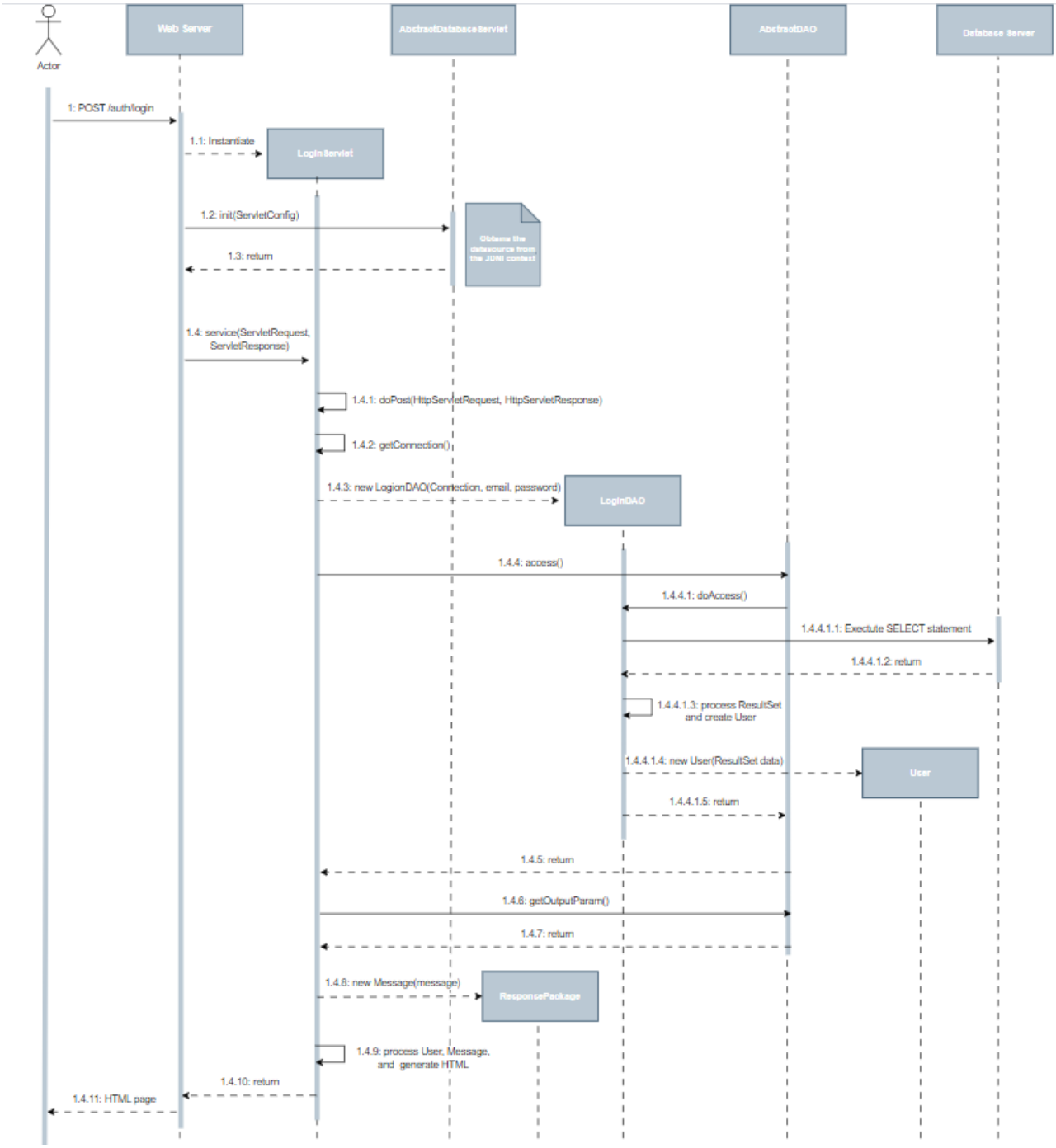
5.1 Class Diagram



The image above illustrates a sample of the project's class diagram. We utilize a distinct servlet called *RestMatcherServlet* (not depicted), to manage all the REST endpoints we've established. Furthermore, servlets were integrated to oversee the non-REST endpoints for each database entity.

Specifically, the image showcases the Java classes responsible for handling the REST endpoints related to tournaments. We employed two distinct handlers, both extending the *RestMatcherHandler* class, to manage all the tournament's REST endpoints. The *TournamentHandler* handles requests concerning the retrieval of the entire list of active tournaments and the creation of new ones. Meanwhile, the *TournamentIdHandler* addresses requests related to a single tournament specified by its ID. Database access is facilitated through specific DAO classes for each type of request. All of these DAOs extend the *AbstractDAO* class, implementing a customized version of the *doAccess()* method. Similarly, we can handle all the CRUD operations for every entity stored in the database.

5.2 Sequence Diagram



Here is reported the sequence diagram for the operation about the login of a simple user. The user initiates the login process by sending a POST request to the web server. The web server instantiates the LoginServlet and gets the information about the datasource from the init method of the AbstractDatabaseServlet. After that it calls the doPost() method of the LoginServlet, passing the HttpServletRequest and the HTTPServletResponse. The LoginServlet analyzes the request and recognizes that it is a SELECT operation,indicating a login attempt, thus it verifies the user's credentials using the request parameters. After that, a ResponsePackage is used to encapsulate the response from the API, providing a structured format for conveying the response status and message, along with any associated data. In the end, the UserServlet replies to the web server, indicating the outcome of the login process using methods like setStatus() and getWriter(), which notify whether the operation was successful or not.

5.3 REST API Summary

Here is the list of REST API implemented.

URI	Method	Description
/auth/login	POST	Sends email and password to be checked
api/users	POST	Creates a new user
api/users/[user]	GET	Returns [user] fields
api/users/[user]	PUT	Updates [user] fields
api/users/[user]	DELETE	Deletes [user]
api/tournaments	GET	Returns a list of all tournaments
api/tournaments	POST	Creates a new tournament connected to the user making the request
api/tournaments/[tournament]	GET	Returns [tournament] fields
api/tournaments/[tournament]	PUT	Updates [tournament] fields
api/tournaments/[tournament]	DELETE	Deletes [tournament]
api/tournaments/[tournament]/teams	GET	Returns a list of all teams in [tournament]
api/tournaments/[tournament]/teams	POST	Create a team connected to [tournament]
api/tournaments/[tournament]/matches	GET	Returns the list of all matches in [tournament]
api/tournaments/[tournament]/ranking	GET	Returns the ranking of [tournament]
api/tournaments/[tournament]/scorers_ranking	GET	Returns the scorers ranking of [tournament]
api/matches/[match]	GET	Returns [match] fields
api/matches/[match]	PUT	Updates [match] fields
api/matches/[match]/events	GET	Returns a list of all the events in [match]
api/matches/[match]/events	POST	Creates a new event connected to [match]
api/events/[event]	PUT	Updates an event
api/events/[event]	DELETE	Deletes an event
api/teams/[team]	GET	Returns [team] fields
api/teams/[team]	PUT	Updates [team] fields
api/teams/[team]	DELETE	Deletes [team]
api/teams/[team]/players	POST	Creates a new player in [team]
api/teams/[team]/players	GET	Returns a list of all players in [team]
api/players/[player]	GET	Returns the fields of [player]
api/players/[player]	PUT	Updates [player] fields
api/players/[player]	DELETE	Deletes [player]

Table 2: REST API Summary

5.4 REST Error Codes

Here is the list of errors defined in the application.

Error Code	HTTP Status Code	Description
200	OK	The request has succeeded
201	CREATED	A new resource has been created
204	NO_CONTENT	The server does not need to return any content in the response body
302	FOUND	The requested resource has been temporarily moved to a different URL
400	BAD_REQUEST	The server cannot process the request due to invalid request structure
401	UNAUTHORIZED	The request lacks proper authentication credentials or the credentials provided are invalid
403	FORBIDDEN	The user is not authorized to perform that particular action
404	NOT_FOUND	The server cannot find the requested resource
405	METHOD_NOT_ALLOWED	The server does not support the HTTP method used in the request for the specified resource
500	INTERNAL_SERVER_ERROR	The server encountered an unexpected condition that prevented it from fulfilling the request
501	NOT_IMPLEMENTED	The server does not support the functionality required to fulfill the request
503	SERVICE_UNAVAILABLE	The server is currently unable to handle the request

Table 3: REST Error Codes

5.5 REST API Details

We report here 3 different type of resources that our web applications handle.

Creation of a player

The following endpoint allows the team owner to add a new player to the team.

- URL: `api/teams/[team]/players`
- Method: `POST`
- URL Parameters: - [team]: the identifier of the team to which the player will belong
- Data Parameters: {`"name": player_name, "surname": player_surname, "team_id": player_team_id, "position": player_position, "medical_certificate": player_medical_certificate, "date_of_birth": player_date_of_birth`}
- Success Response:
  - Code: 201
    - Content: { `"result": "Resource created"` }

- Error Response:
  - Code: 302
    - Content: { "error": { "code": 302, "message": "User not logged in" } }
  - Code: 400
    - Content: { "error": { "code": 400, "message": "Wrong request format" } }
  - Code: 403
    - Content: { "error": { "code": 403, "message": "User doesn't have the necessary permissions" } }
  - Code: 503
    - Content: { "error": { "code": 503, "message": "Server not ready" } }

Get a list of all the players in a team
The following endpoint returns the list of all the players belonging to a team.

- URL: `api/teams/[team]/players`
- Method: `GET`
- URL Parameters: - [team]: the identifier of the team
- Data Parameters: no data params needed
- Success Response:
  - Code: 200
    - Content: [{"ID": player1_ID, "name": player1_name, "surname": player1_surname, "team_id": player1_team_id, "position": player1_position, "medical_certificate": player1_medical_certificate, "date_of_birth": player1_date_of_birth}, ...]
- Error Response:
  - Code: 204
    - Content: { "error": { "code": 204, "message": "No content" } }
  - Code: 503
    - Content: { "error": { "code": 503, "message": "Server not ready" } }

Get player fields
The following endpoint returns all the fields of a player.

- URL: `api/players/[player]`
- Method: `GET`
- URL Parameters: - [player]: the identifier of the player
- Data Parameters: no data params needed
- Success Response:
  - Code: 200
    - Content: {"ID": player_ID, "name": player_name, "surname": player_surname, "team_id": player_team_id, "position": player_position, "medical_certificate": player_medical_certificate, "date_of_birth": player_date_of_birth}

- Error Response:
  - Code: 404
    - Content: { "error": { "code": 404, "message": "Not found" } }
  - Code: 503
    - Content: { "error": { "code": 503, "message": "Server not ready" } }

6 Group Members Contribution

Alberto Basaglia Alberto was part of the design process of the ER diagram. He wrote the logic that handles the dispatching of REST requests. Additionally he wrote the part of the system that handles the management of the session and the authentication. He reviewed the report and wrote the Presentation Logic Layer section.

Andrea Bruttomesso Andrea took part in designing the ER diagram and REST API endpoints to meet established standards. Additionally, he assisted in crafting the REST request handling process. Then played a role in developing Data Access Objects (DAOs) and endpoints for the Match entity. Moreover, Andrea implemented an automated system for generating matches, tailored for real-world applications. Throughout the project, he actively engaged in code reviews, addressing minor issues, and suggesting improvements to enhance code style and readability.

Alessandro Corrò Alessandro worked on the initial setup of the presentation layer and gave a strong contribution in writing the report. He wrote all the DAOs, Servlet and Handlers related to the Events and all the endpoints related to Match-Event handling.

Milica Popovic Milica wrote the first two sections of the report: Objectives and Main Functionalities. Additionally, she helped with creating the ER diagram. Furthermore, she created the sequence diagram, as well as all DAOs related to the Entity Type Team, and all the endpoints related to Team handling.

Davide Seghetto Davide wrote the first two parts of the report together with Milica, and also the third and fourth part. He produced the presentation layer and the sample for the class diagram. Finally, he wrote the code for the RankingScorerServlet and for the DAOs and the handlers related to the Tournament.

Andrea Stocco Andrea contributed to the design of the ER diagram and wrote sections on REST API details and REST Error Codes. He played a role in developing DAOs and endpoints to manage the Player entity and the tournaments ranking. Finally, Andrea implemented the logic for uploading binary files to the database and reviewed the report.