Design Description *version 3* *(January 7th, 2023)*

Project Code Defenders - Robo Tournament
Team Codebenders

POLITECNICO
MILANO 1863

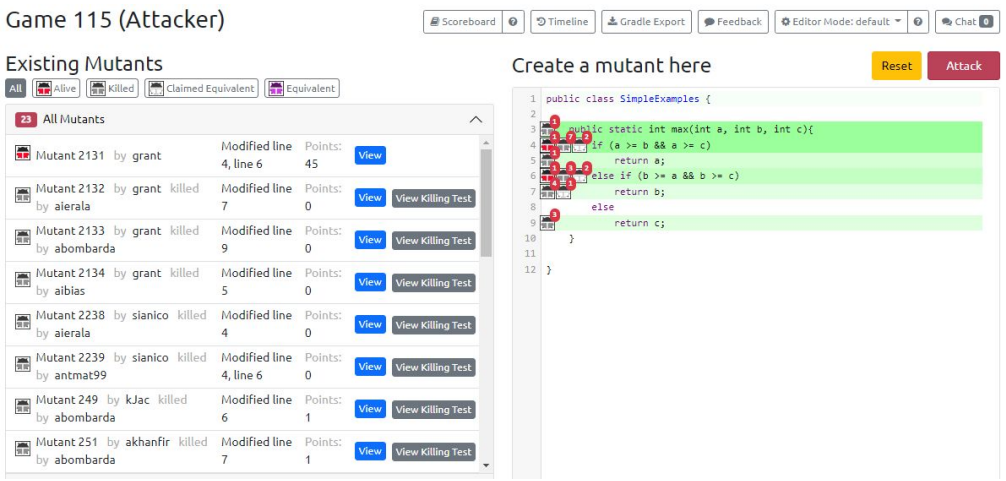
Mälardalen
University

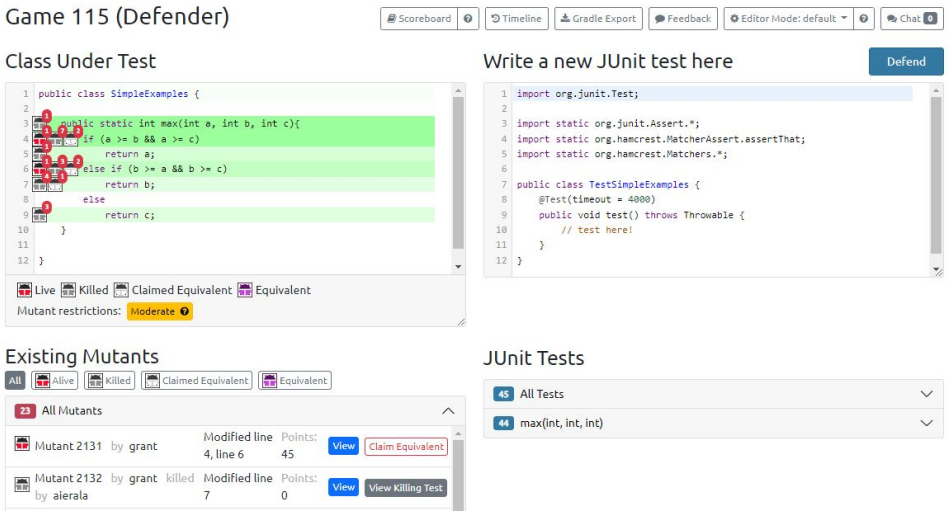
Table of content

Project vision

Code Defenders

- Software quality and testing are at the heart of software engineering, but they may not always get enough attention from software engineering education.
- CodeDefenders (web game) proposes the use of **gamification** to teach **mutation testing** and to strengthen code writing and testing skills.
- The game supports **team play and competition** by having Attackers - Defenders teams whose goal is to inject errors into code or write unit tests to catch them.
- The **"CodeDefenders: RoboTournament"** project aims at enriching the game by adding support for students tournaments and games against bots.

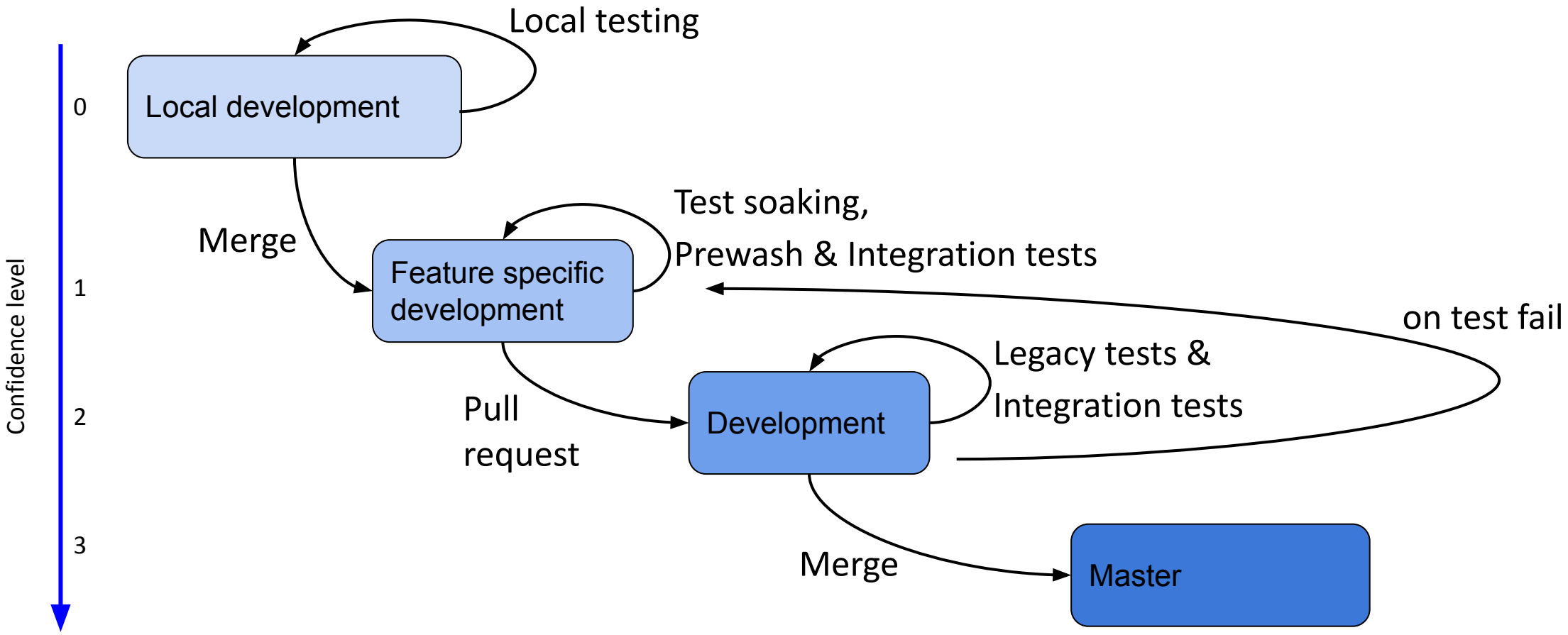




POLITECNICO MILANO 1863

Mälardalen University

Development and Testing workflow



Project requirements

- Implement a **tournament application**. This application must use CodeDefenders as a remote service (through APIs) and must include at least two tournaments modalities.
- Design and implement a set of **OpenAPIs for CodeDefenders** which can be used from the tournament application to manage games and players.
- Implement a **load balancing** mechanism which allows the tournament application to communicate with **multiple CodeDefenders servers** and to always create games on the less loaded server.
- Implement a **streaming** component which allows users to follow in progress games live. This component can optionally include an “overall tournament view” showing schedule, standings and other information for each tournament.
- Design and implement a set of **APIs** which allows users to train **bots** over past games data and to let those bots play CodeDefenders.

What the project is not going to address

- The tournament application will be an **external application**, developed separately. It won't be a plugin of CodeDefenders nor an application running on the same host.
- The tournament application will implement only the **tournament** and **streaming logic**. It won't reimplement or modify in any way the game logic, which is already coded in CodeDefenders and will be accessible through our APIs.
- We won't implement an **AI** playing CodeDefenders. This project requirement is **optional** and we are not planning to realize it because of the current lack of AI knowledge within our team.
- Streaming component will **not respect hard real-time** constraints

Desired functionalities (User Stories organized in Epics)

Tournament Management

CDF-32 Login/Register
CDF-41 Display tournaments info
CDF-33 Create Tournament
CDF-34 Join tournament
CDF-42 Matchmaking

Play tournament games

CDF-36 Starting games
CDF-38 Leave game and game end

Watch a streamed tournament game

CDF-39 View game stream
CDF-40 Notification of game stream updates

Team Management

CDF-35 Team creation
CDF-54 Team management
CDF-37 Join team

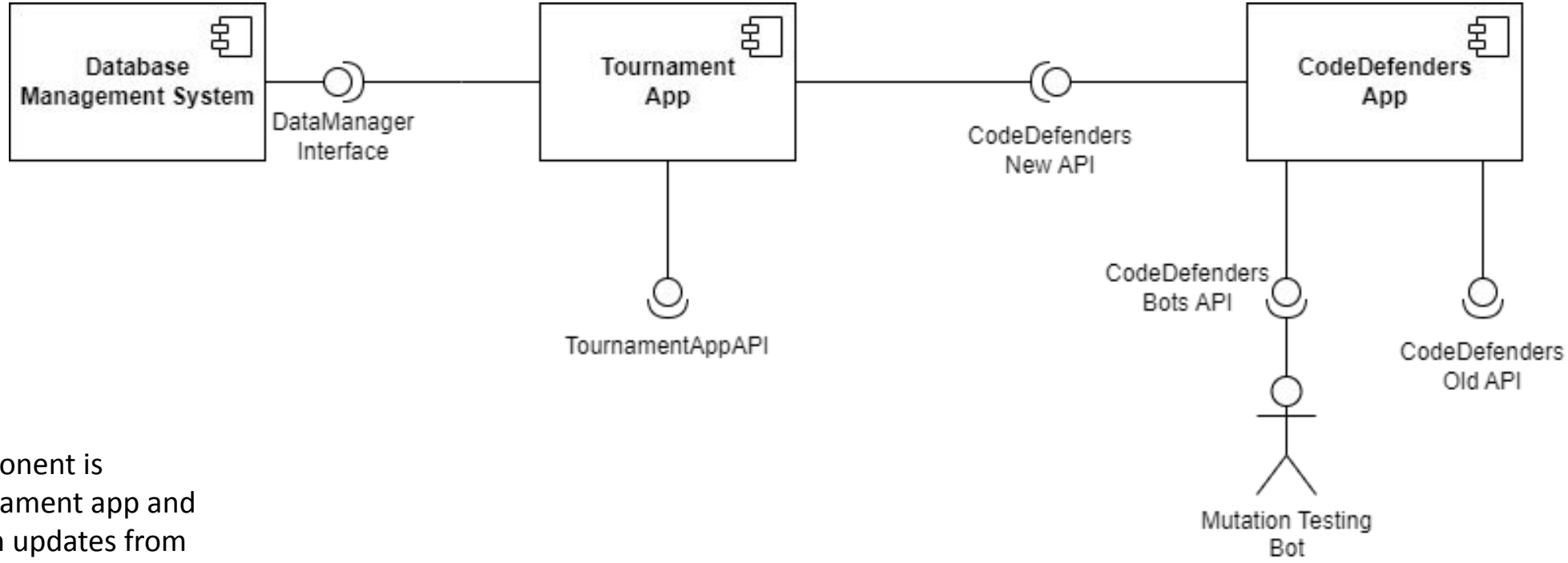
Play with bots

CDF-43 Bots can play
CDF-44 Bots can be trained

Avoid CodeDefenders overload

CDF-69 Efficient flow of updates
CDF-31 Load Balancing

High-level component diagram



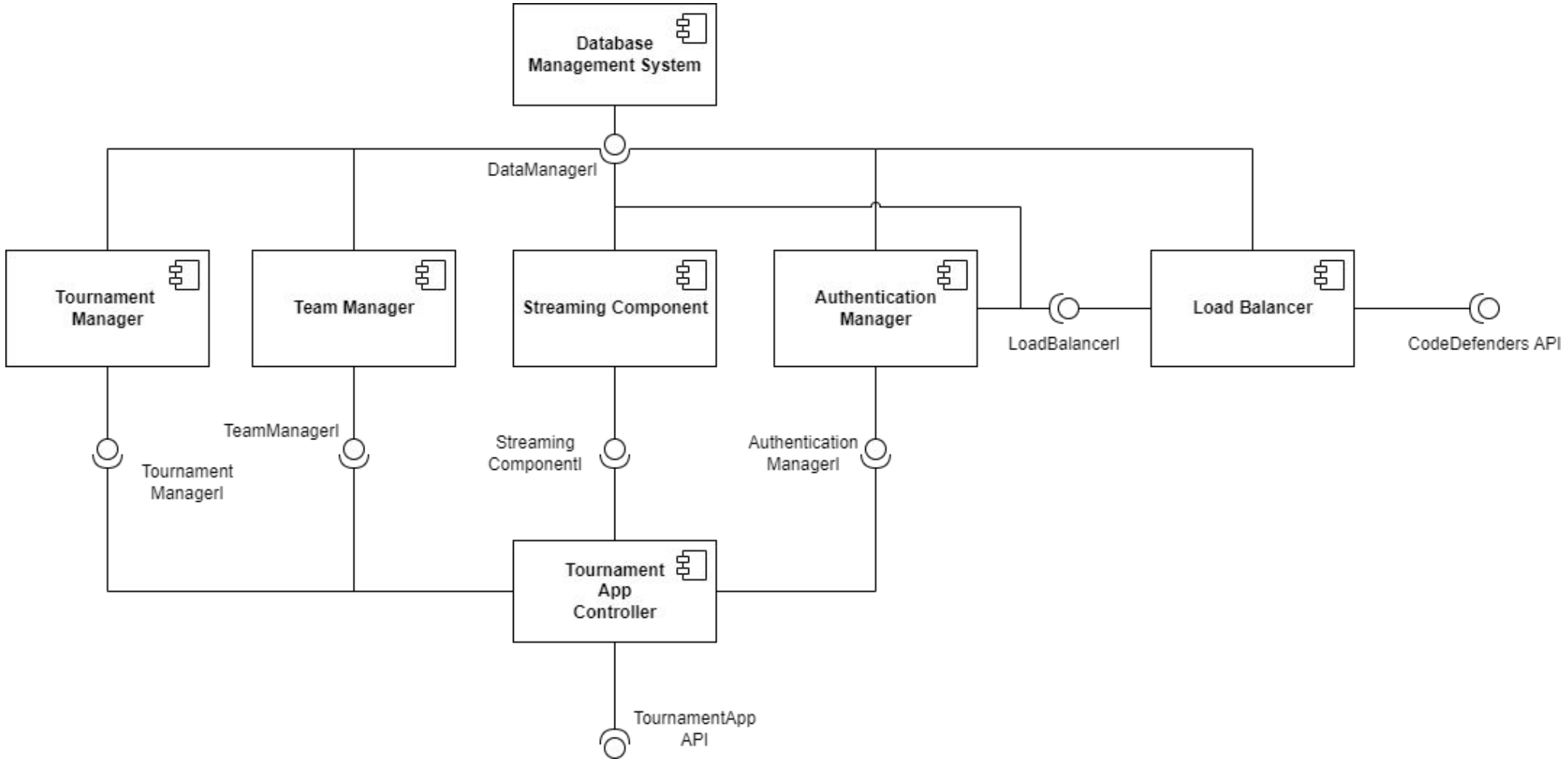
The streaming component is internal to the tournament app and exploits APIs to fetch updates from CodeDefenders

Software architecture

Each software component addresses the following user stories

- Tournament app:
  - CDF-41, CDF-33, CDF-34, CDF-42, CDF-35, CDF-37, CDF-54, CDF-36, CDF-38, CDF-39, CDF-40
- Code defenders APIs:
  - For bots:
    CDF-43, CDF-44
  - Other:
    CDF-32, CDF-39, CDF-36

Zoom on the tournament app



Software architecture

Each software component addresses the following user stories

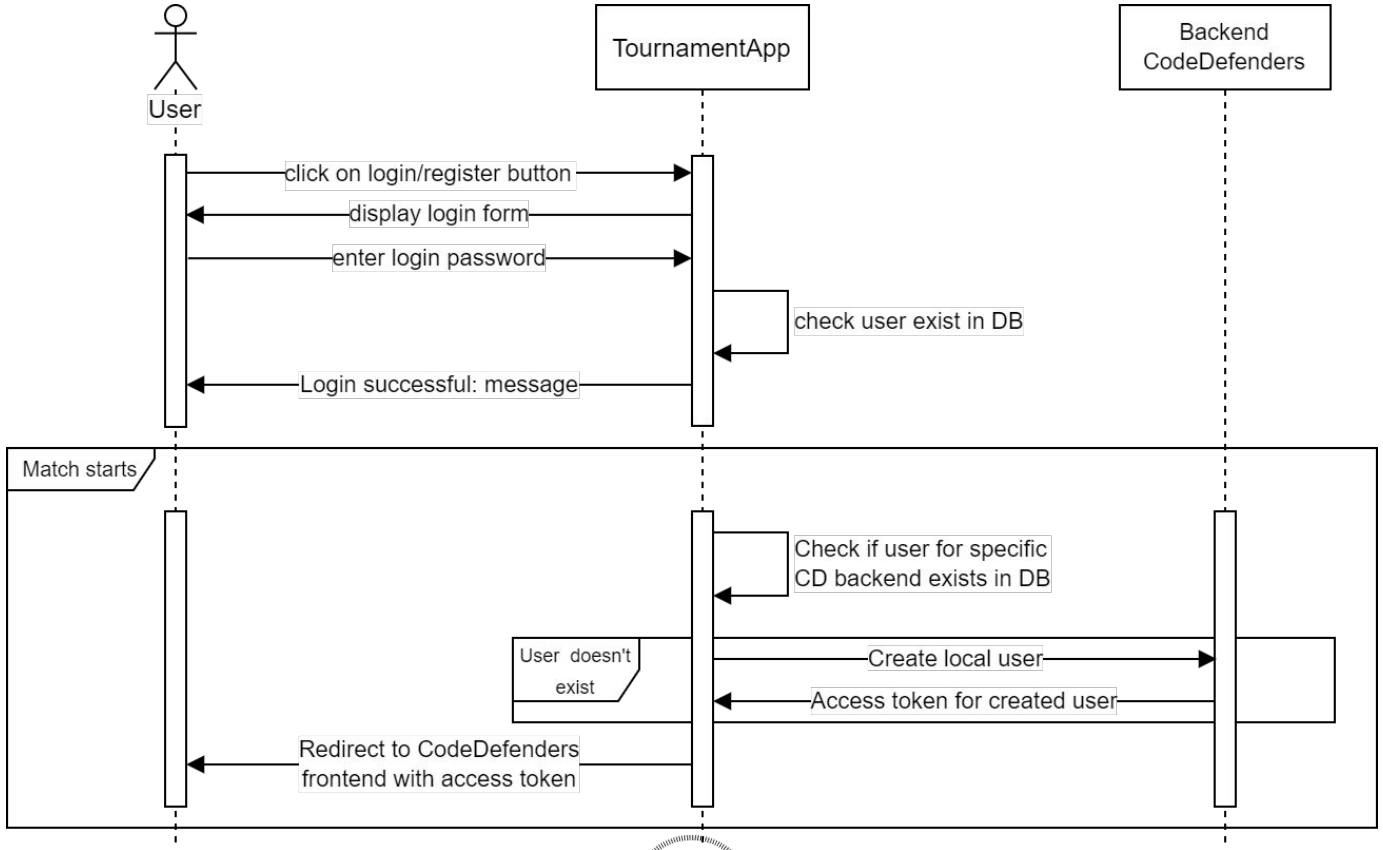
- Team manager:
  - CDF-35, CDF-37, CDF-54
- Tournament manager:
  - CDF-41, CDF-33, CDF-34, CDF-42
- Streaming component:
  - CDF-39, CDF-40
- Authentication manager:
  - CDF-32
- Tournament app, other:
  CDF-36, CDF-38

Important design decisions

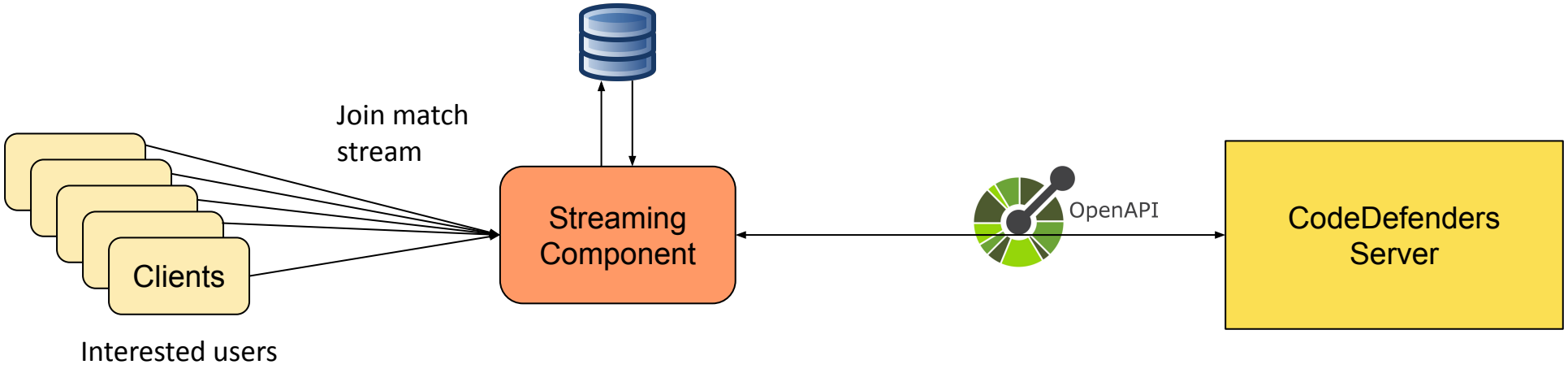
- Authentication sequence
- Streaming component
- Load Balancer
- CodeDefenders API
- Database design

Authentication sequence

The users register and login on our TournamentApp, with no interaction with the CodeDefenders backend.
When a match starts, we check if our user has a corresponding user on the CodeDefenders instance hosting the game. If it doesn't, we create it and get the access token; otherwise it already exists and we already have the token.
We redirect the user to the target game page on CodeDefenders specifying the authentication token, so that the CodeDefenders login page is skipped



Streaming component

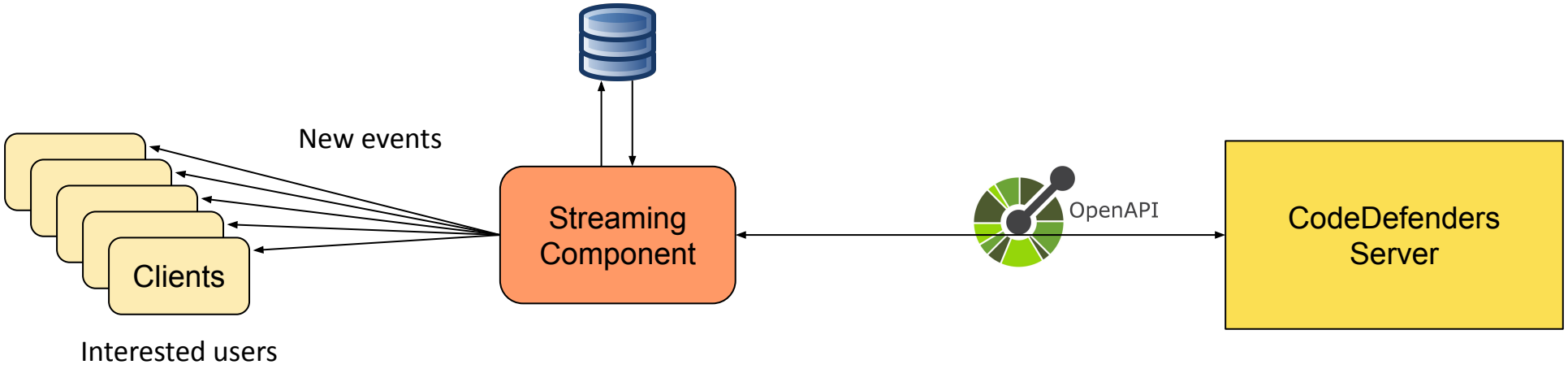


The streaming component communicates with CodeDefenders using OpenAPI as explicitly requested by the customer.

- The streaming component periodically (real time updates are not required) pulls events from CodeDefenders server instances
- It asks only for the new events happened from the last request

Clients interested in an ongoing game join the stream by sending a request to the Streaming Component and establishing a WebSocket communication channel.

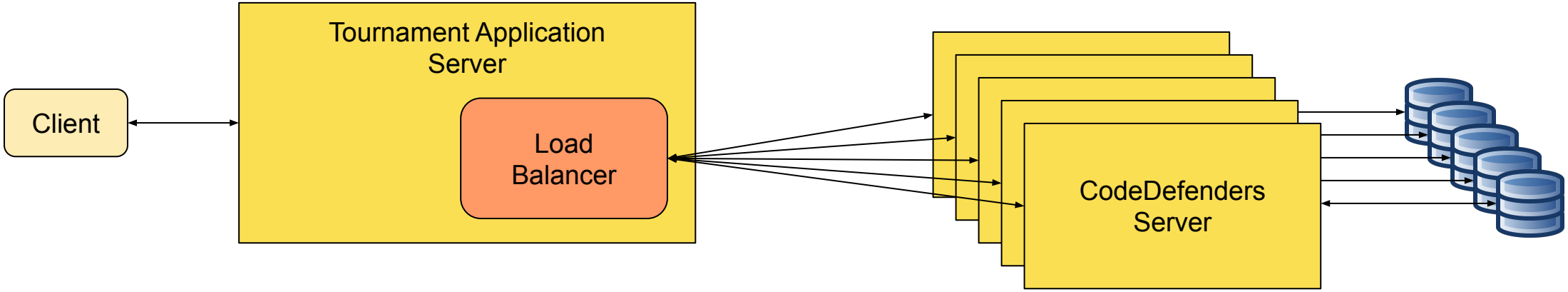
Streaming component



After requesting and receiving updates from CodeDefenders, the streaming component forwards the new events to interested clients.

- The new events are dispatched to the clients during a time interval of few second
- In this way the streaming is delayed by some seconds, but the events are received by the clients at the same rate they are happening and not all at once

Load Balancer



We will implement our own load balancer.

When a new game needs to be created the load balancer selects the CodeDefenders server with the minimum number of active games and issues a create game request to it.

The tournament app stores a mapping between active games and CodeDefenders instances in order to be able to redirect players and spectators requests to the correct server. The load balancer component addresses CDF-31.

Exposed CodeDefenders APIs

/api/class	GET	Get a class by its ID
/api/history	GET	Get the history of all played games
/api/player	GET	Returns a player's username and userId from its playerId
/api/user	GET	Returns a user's informations from its ID
/api/game	GET	Get the status of the game with the specified ID
/api/game/role	GET	Get own role in the game with the specified ID
/api/admin/class/upload	POST	Upload a class providing name and source
/admin/api/game	POST	Create a new game with the specified class, settings and teams
/admin/api/game/start	POST	Start a game
/admin/api/game/disable-uploads	POST	Disable class and mutant uploads in a game (advance it to the grace one phase)
/admin/api/game/disable-claims	POST	Disable mutant claims in a game (advance it to the grace two phase)
/admin/api/game/end	POST	End a game

Exposed CodeDefenders APIs

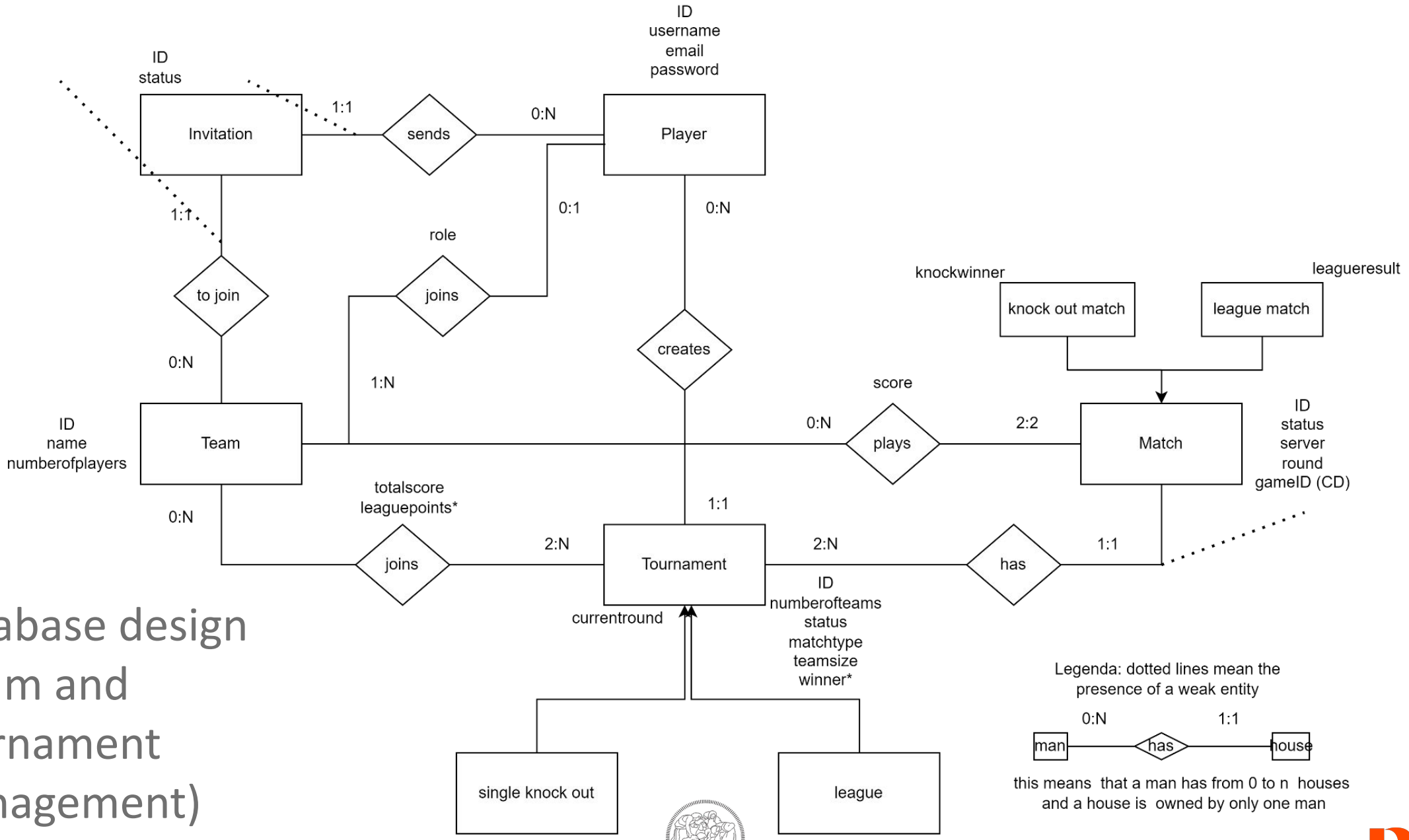
/api/game/settings	GET	Get the settings of the game with the specified ID
/api/game/test	GET	Get a test by its ID
	POST	Upload a test providing its code and target game
/api/game/test/template	GET	Get the test template for the game with the specified ID
/api/game/mutant	GET	Get a mutant by its ID
	POST	Upload a mutant providing its code and target game
/api/game/mutant/equivalences	GET	Get unresolved equivalence claims for the game with the specified ID
/api/game/mutant/equivalence/claim	POST	Claim the mutant with the specified ID as equivalent, meaning that it doesn't affect the behavior of the code
/api/game/mutant/equivalence/resolve	POST	Resolve the pending equivalence claim for a mutant, either by accepting it as equal or uploading the code for a killing test
/admin/api/auth/newUser	GET	Generate a new user with a name in the format `{creatorUsername}_{username}_{randomNumbers}`

Exposed CodeDefenders APIs

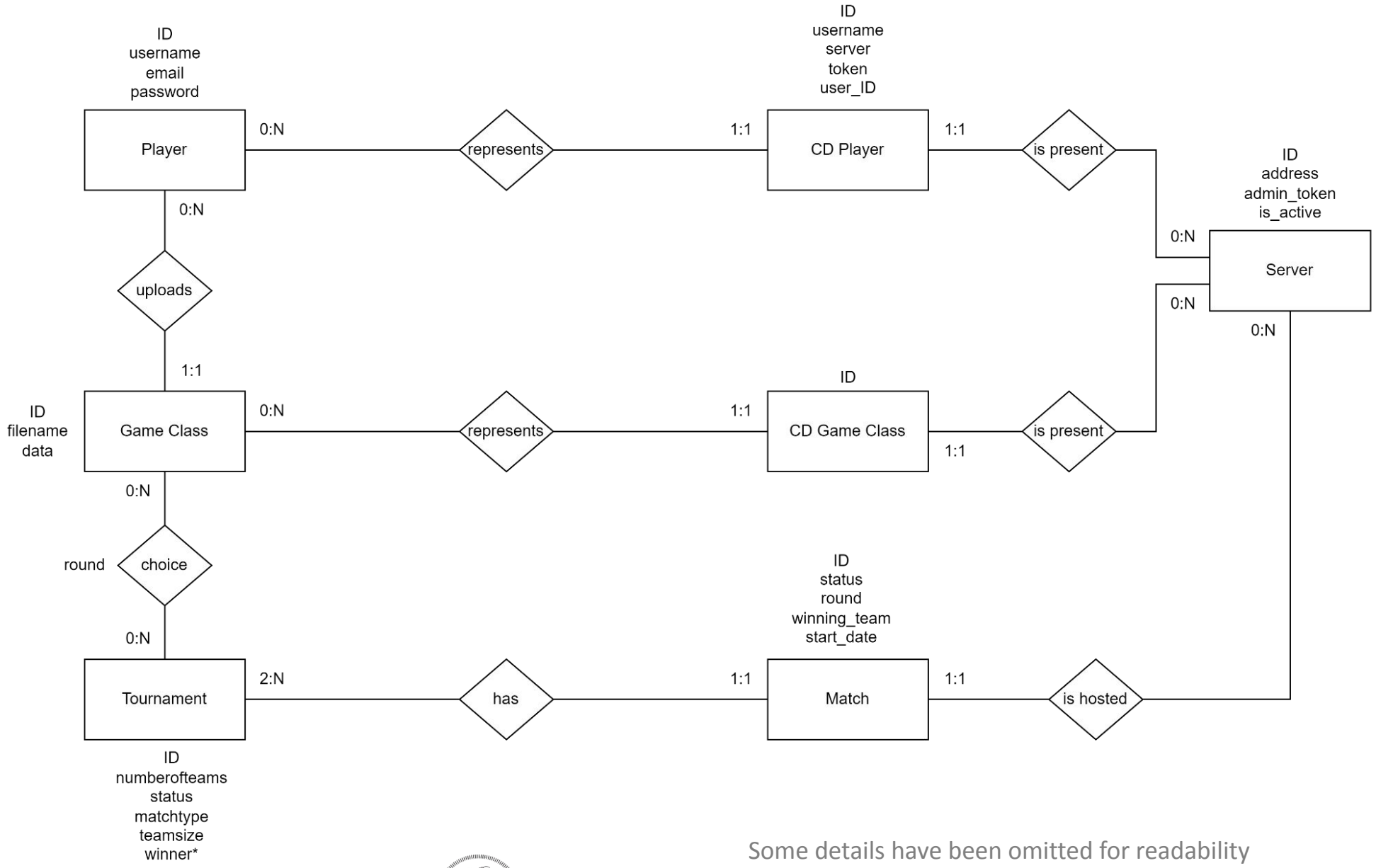
/admin/api/auth/token	GET	Get an authentication token for the Bot API, tied to a specific user
/api/auth/self	GET	Get own information (userId, username, API token)
/admin/api/auth/self	GET	Get own information (userId, username, API token). This API has the same functionality as getSelf, but it can be called only by admins, so it can be used as a permission check
/admin/api/events	GET	Get events and scores from all games created by the logged user, starting from the specified time\nOnly scores for games that also have events are returned
/admin/api/load	GET	Get server load (count of active battleground and melee games)

A formal description of exposed CodeDefenders OpenAPI can be found on our GitHub repository

Database design (Team and tournament management)



Database design (Match progress)

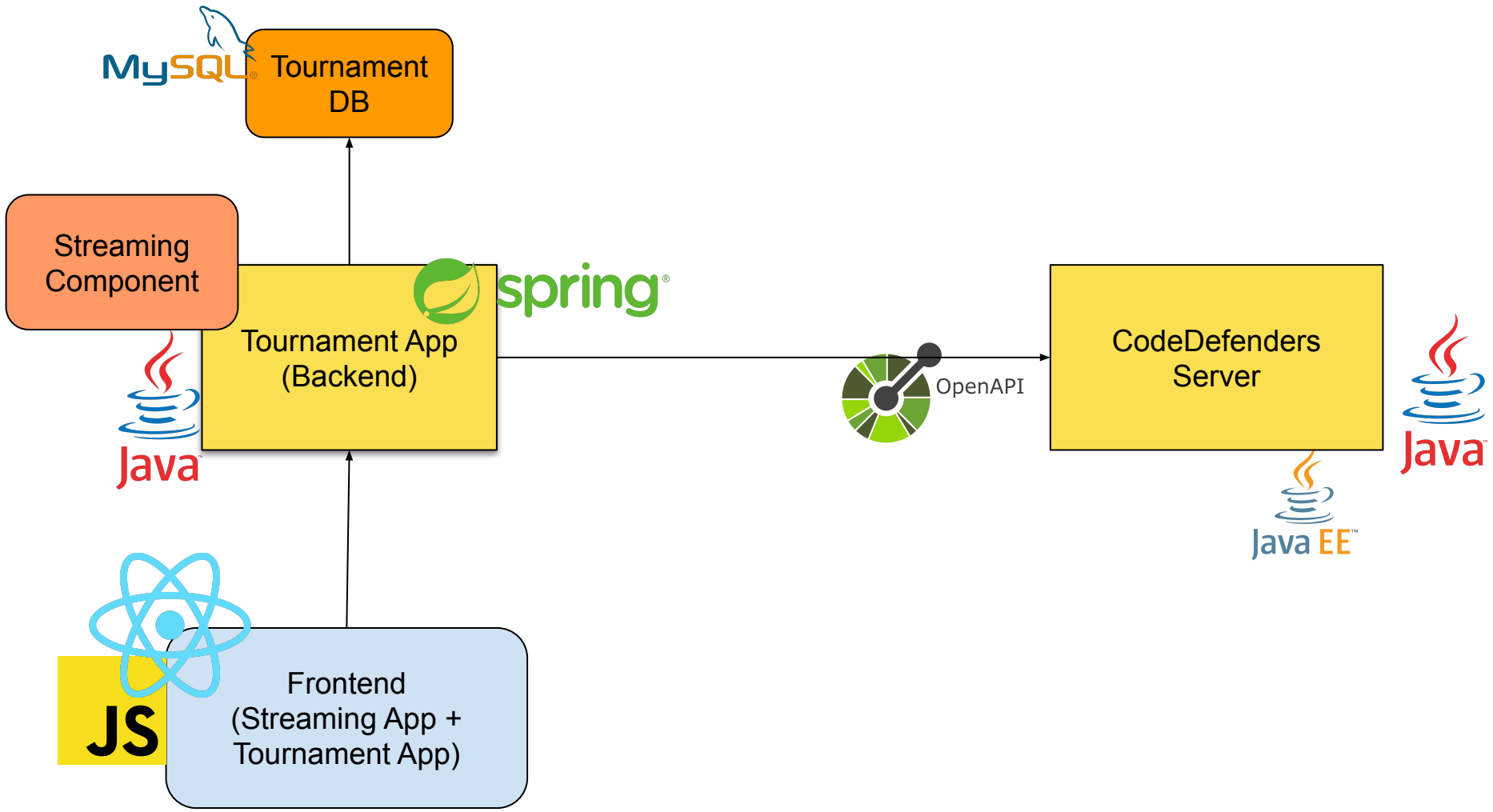


Some details have been omitted for readability

Database Design

- Players compete in matches through their teams
- Teams can be of variable size
  - this way tournaments with teams of max size of one player are possible, in other words, 1v1 matches are supported
  - tournament logic sets conditions for teams size
- Each match is assigned to exactly one tournament, and has exactly two teams as competitors (attackers vs defenders)

Technologies we are going to use



Motivations for technologies

Java + Spring:

- Java knowledge quite already spread among the team
- Integrates very well with concepts taken from JavaEE
- More support than JavaEE on forums and so on (also tutorials online)
- Allows to structure a Java Web application in an easy way

MySQL:

- The structure of the data we need will suit perfectly in a relational database.
- Team already familiar with this technology.

React:

- Free, open-source, and explanatory JavaScript library with simplistic learning curve
- Used for building simple or complex user interfaces, stable front-end framework
- Supports multi-purpose, clean architecture and platform-specific modules

Graphical User Interface

Tournament application should have user interface for

- Login and registration
  - CDF-32
- Team creation
  - CDF-35
- Team management (including joining a team)
  - CDF-37, CDF-54
- Tournaments overview
  - CDF-34, CDF-36, CDF-41
- Tournaments creation
  - CDF-33, CDF-42
- Watching live matches (streaming component)
  - CDF-39, CDF-40

Tournaments Overview

Hello user1! Welcome to tournament application of Code Defenders web game!

Home

Logout

Create Team

Manage Teams

Join Team

Tournament Creation

Tournaments

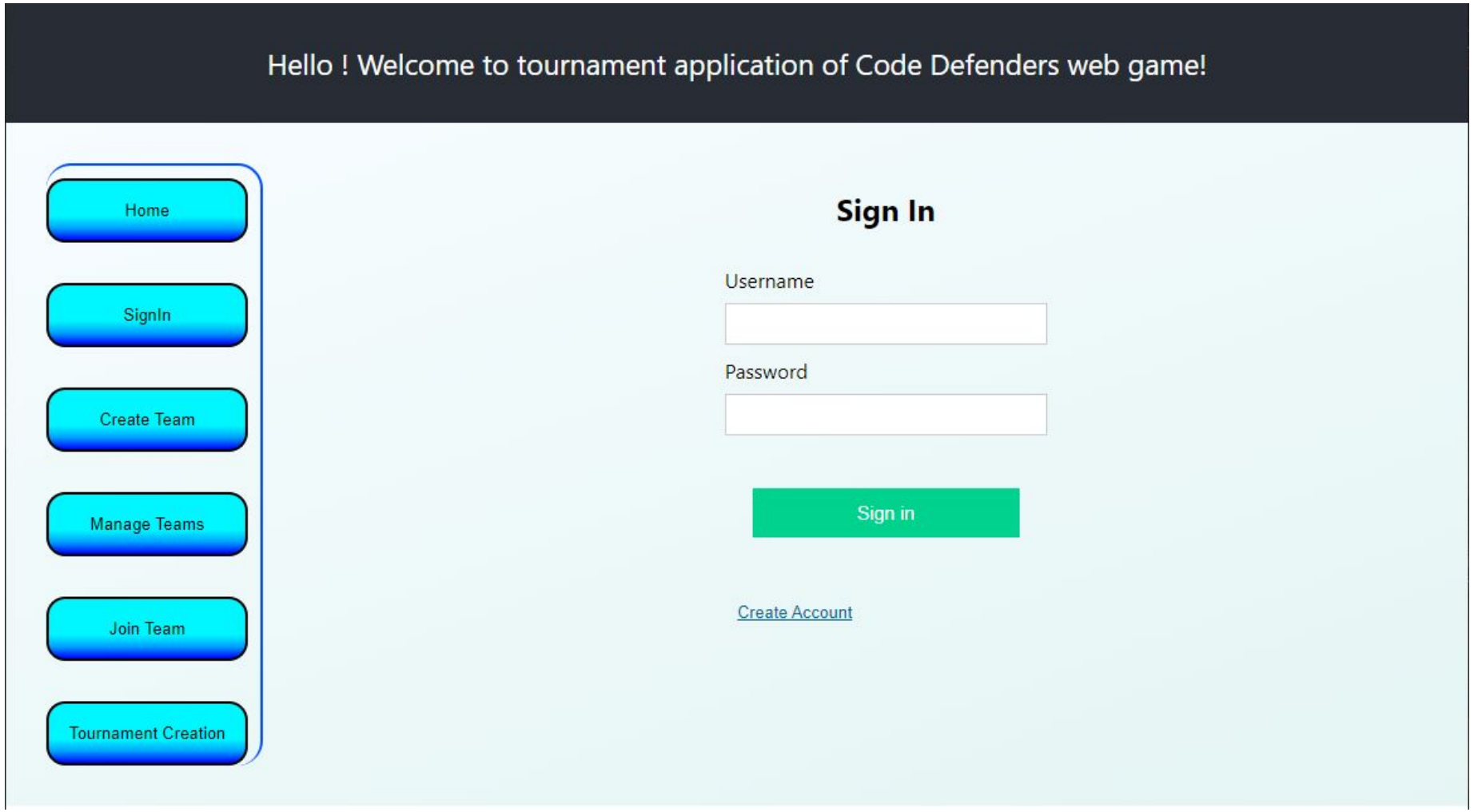
Search tournament by name

Name	Starting Date	Type	Status	Team Size	
tournament1	5/1/2023 14:12:54	KNOCKOUT	IN_PROGRESS	1	Joined / Scoreboard

Teams participating in the tournament:
team1 and team2

Team1	Team2	Round	Team1 Score	Team2 Score	Status	Winner	
team2	team1	1	0	0	IN_PHASE_ONE	-	Live Score / Play

Login


Hello ! Welcome to tournament application of Code Defenders web game!
Home
SignIn
Create Team
Manage Teams
Join Team
Tournament Creation
Sign In
Username
Password
Sign in
Create Account

Tournament Creation

Hello user1! Welcome to tournament application of Code Defenders web game!

Home

Logout

Create Team

Manage Teams

Join Team

Tournament Creation

Tournament creation

Name

tournament1

Type of tournament

Knockout

Size of teams

1

Number of teams

Must be a power of 2 between 2 and 16

2

Create tournament

Upload classes

Scegli file Nessun file selezionato

Upload

Select Game Classes

Hello user1! Welcome to tournament application of Code Defenders web game!

Home

Logout

Create Team

Manage Teams

Join Team

Tournament Creation

Tournaments

Search tournament by name

Name	Starting Date	Type	Status	Team Size	
tournament1	Still to be defined	KNOCKOUT	TEAMS_JOINING	1	Join

Classes selected so far:

Round	Class Name	Author
1	SimpleExample.java	admin

Teams participating in the tournament:
no team has joined yet

Number of teams required: 2

Remaining number of team slots: 2

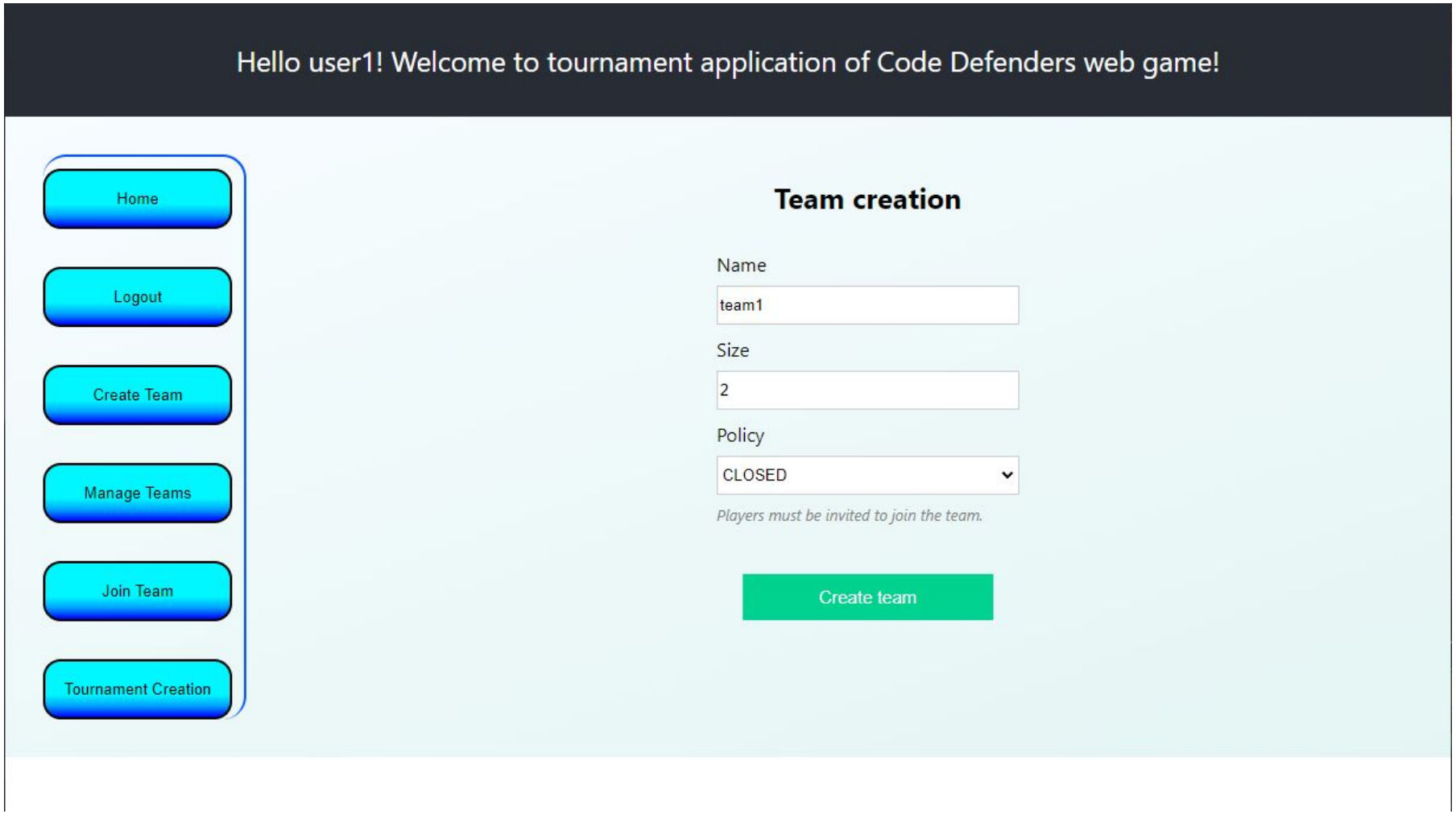
Choice of class for each round:

Round: 1

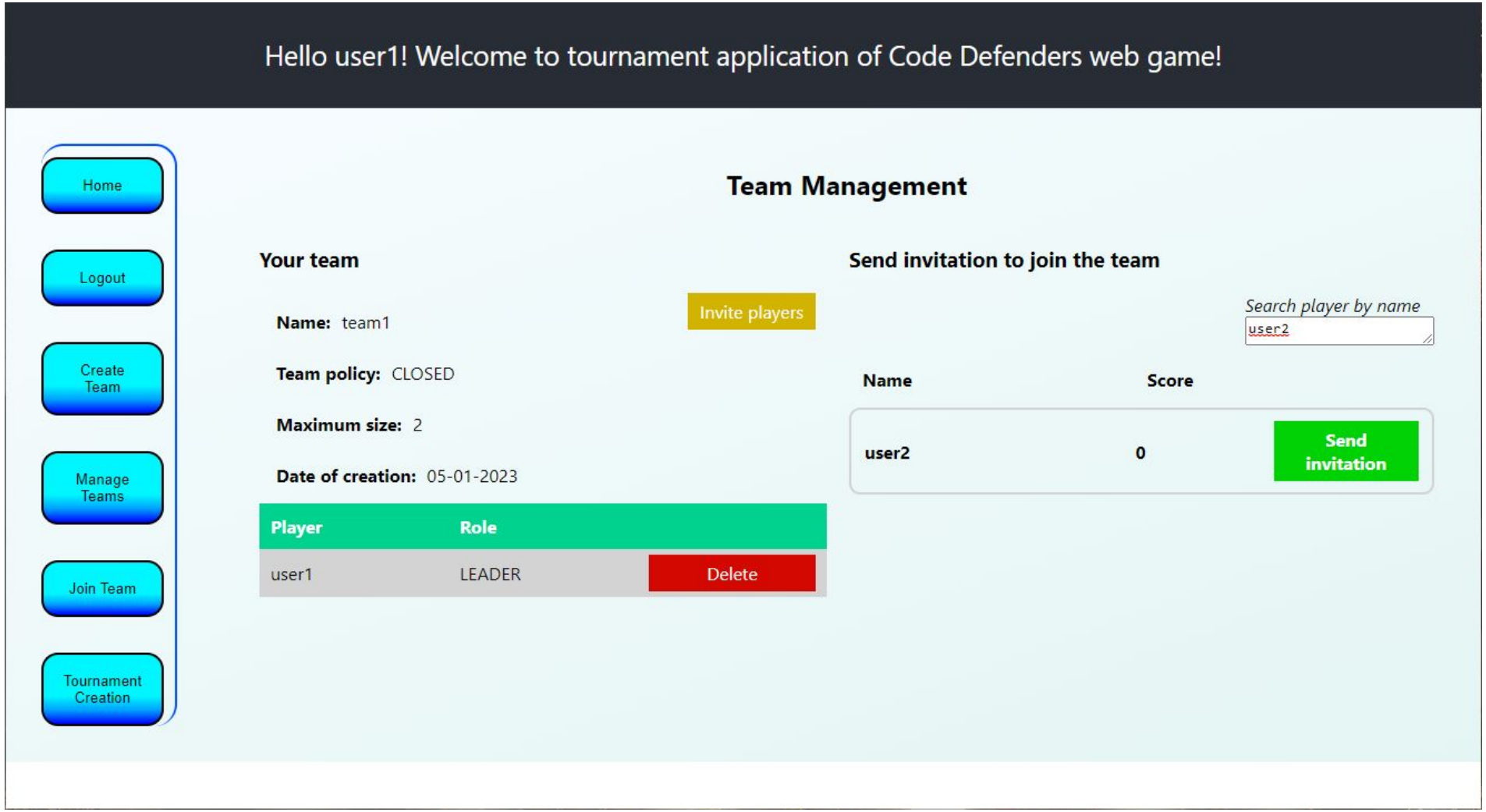
Class: SimpleExample.java

Select Class

Team Creation



Team Management



Join a Team


Hello user1! Welcome to tournament application of Code Defenders web game!
Home
Logout
Create Team
Manage Teams
Join Team
Tournament Creation
Check Invitations (0)
Teams
Search team by name
team1
Date of creation: 5.1.2023.
Members (1) of maximum size (2) :
user1 (leader)
The team is CLOSED to new players.
team2
Date of creation: 5.1.2023.
Members (1) of maximum size (2) :
user2 (leader)
The team is OPEN to new players.
Join Team

Play a Match on CodeDefenders

Back To Tournament Application Phase 1 D: 0 H: 0 M: 1 S: 59

Game 426 (Defender)

Scoreboard | Timeline | Gradle Export | Feedback | Editor Mode: default | Chat 0

Class Under Test

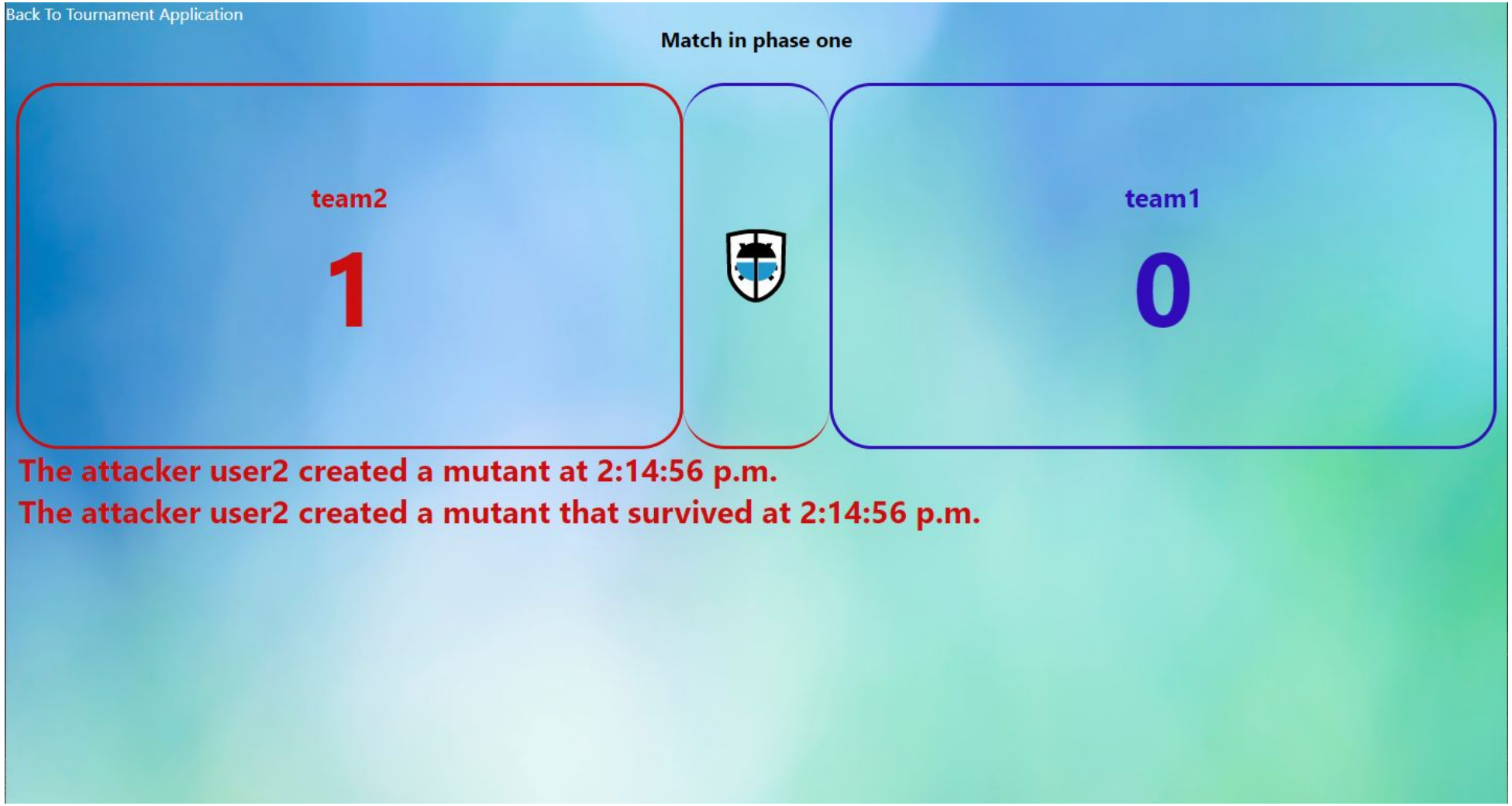
```
1  public class SimpleExample {
2
3      public static int max(int a, int b, int c){
4          if (a >= b && a >= c)
5              return a;
6          else if (b >= a && b >= c)
7              return b;
8          else
9              return c;
10     }
11
12 }
```

Write a new JUnit test here

Defend

```
1  import org.junit.Test;
2
3  import static org.junit.Assert.*;
4  import static org.hamcrest.MatcherAssert.assertThat;
5  import static org.hamcrest.Matchers.*;
6
7  public class TestSimpleExample {
8      @Test(timeout = 4000)
9      public void test() throws Throwable {
10         // test here!
11     }
12 }
```

Watch a Match Live



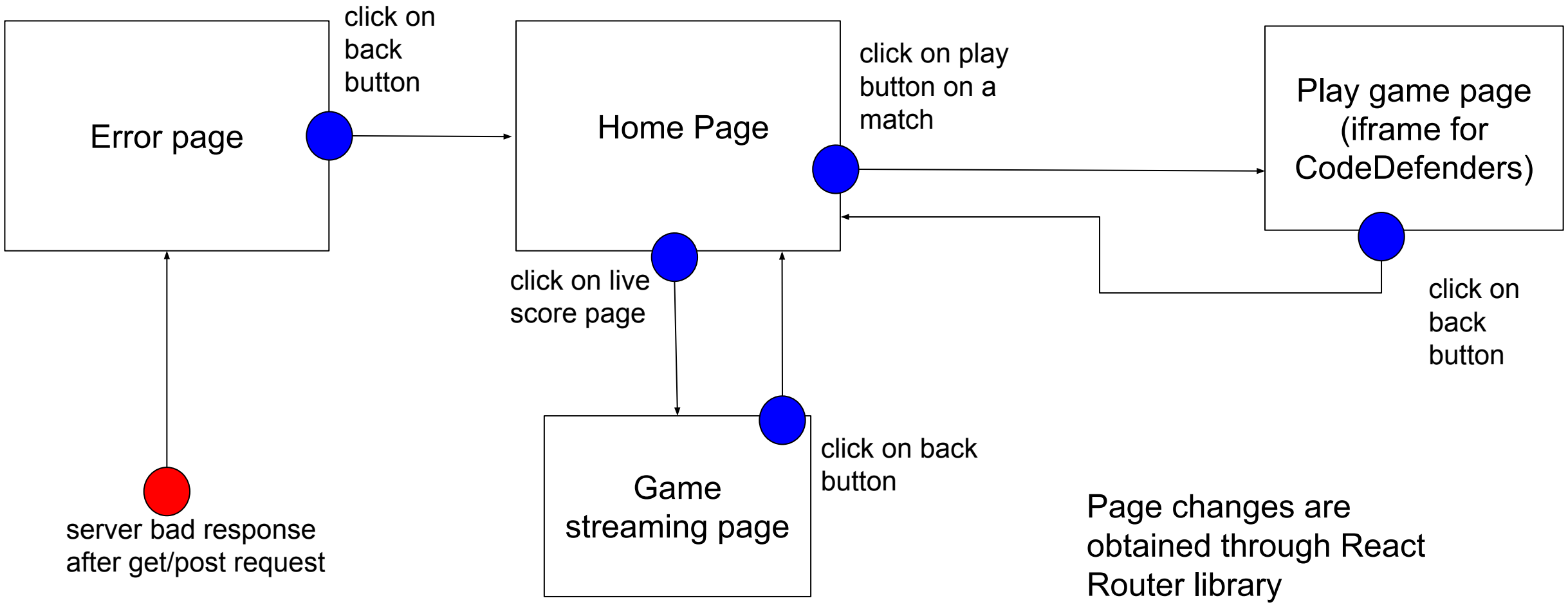
Detailed frontend software design

Frontend has been implemented using React and it's organized in three main pages:

- Home page: contains all the different sections of the Tournament Application (login and registration, tournaments, team creation and management)
- Play game page: embeds CodeDefenders interface to play the game with an overlay containing a timer and a back button
- Streaming page: shows streamed events and scores

An additional error page is used for unexpected errors.

Frontend high level pages division design



Page changes are obtained through React Router library

Frontend pages and sections

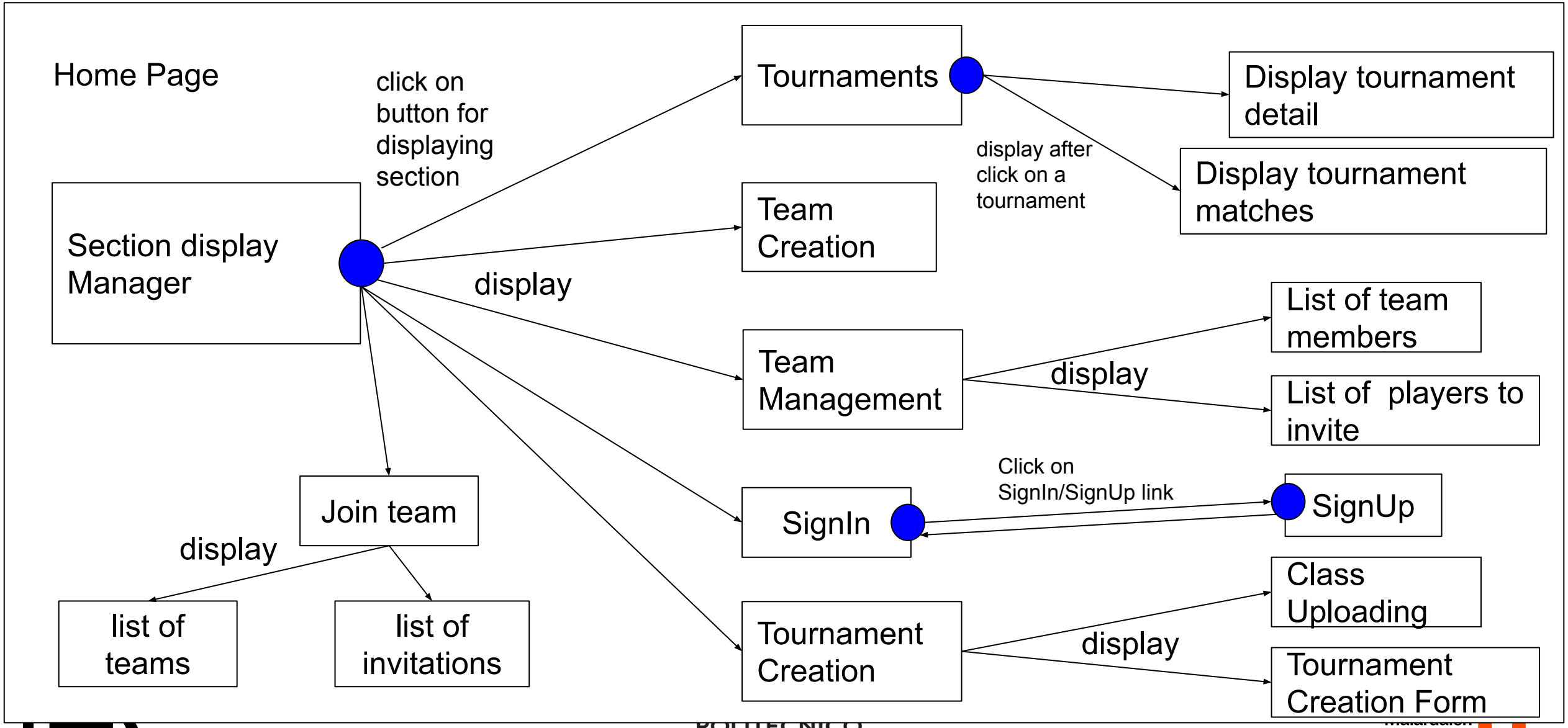
- HomePage
  - SignInSection
  - SignUpSection
  - TournamentSection
    - TournamentComponentSection
    - TournamentMatchSection
  - TeamManagementSection
    - KickPlayerSection
    - InvitePlayerSection
    - PromoteToLeaderSection
  - CreateTeamSection
  - JoinTeamSection
    - Accept/RefuseInvitationSection
    - JoinTeamFromListSection
  - CreateTournamentSection
    - UploadClassSection
    - CreateTournamentFormSection
- PlayPage
  - IFrameSection
  - MenuBar
- StreamingGamePage
  - StreamingEventSection
- ErrorPage

All green pages/sections are mapped by a main corresponding react component

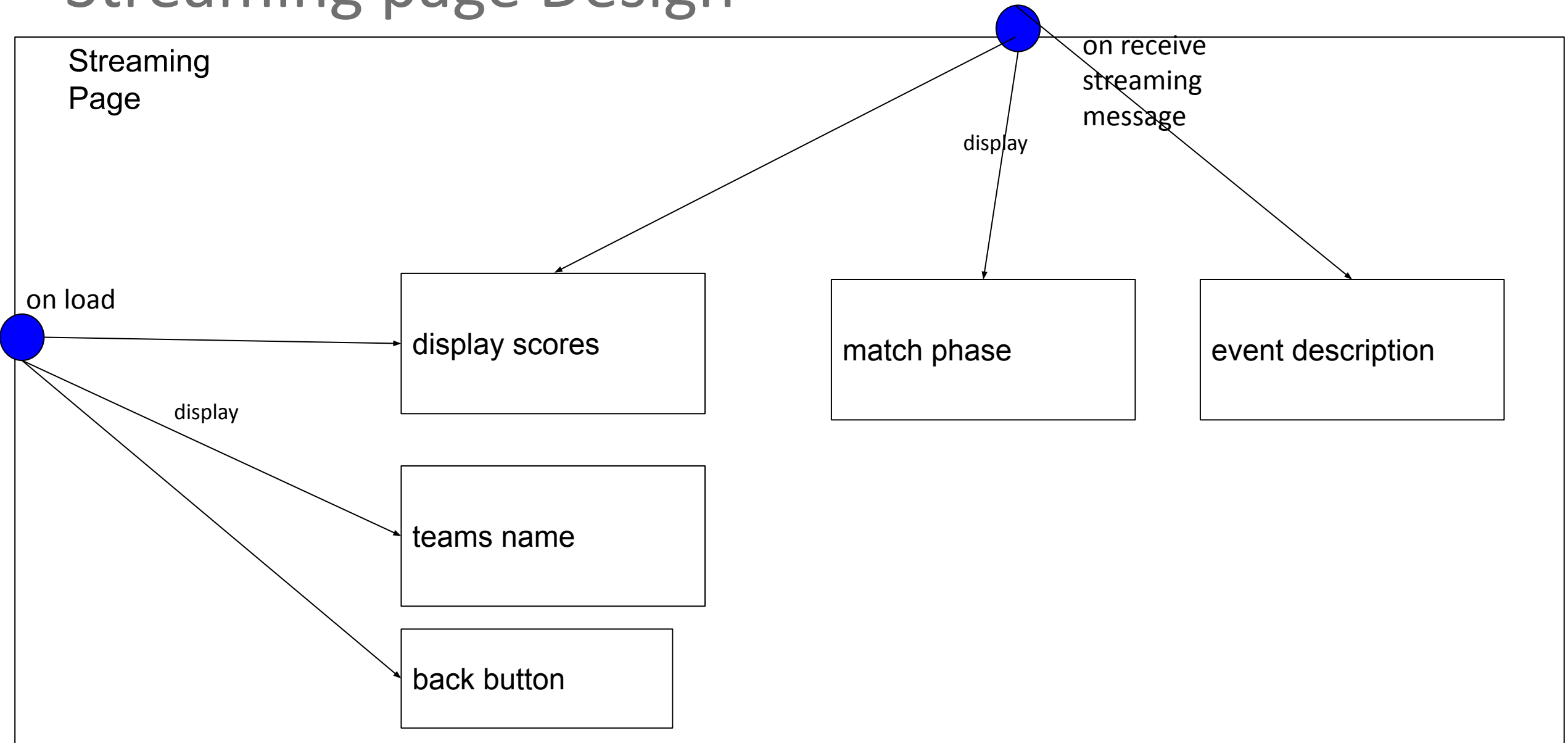
Frontend Actions and Events

- click on one of the home page display section buttons
- click on signin button
- click on sign up button
- click on logout button
- click on signup link
- click on tournament row to display informations
- click on join tournament button
- click on the button to select classes for each round
- click on the button to create tournament
- click on the button to create team
- click on the button to send invitations
- click on the button to accept/refuse invitation
- click on scoreboard button
- click on the button to promote a member to team leader
- click on the button to kick a member from a team
- click on the button to upload a class
- click on the button to play a game
- click on the button to show live score
- update received from streaming game component with partial page refresh
- click on labels to order by name/date/type for tournaments
- insert data in textfield to order data by the content of the textfield(for teams/tournaments/players)
- click on back button from play game iframe
- click on name header to sort tournaments by name
- click on back button in streaming page
- click on leave team button

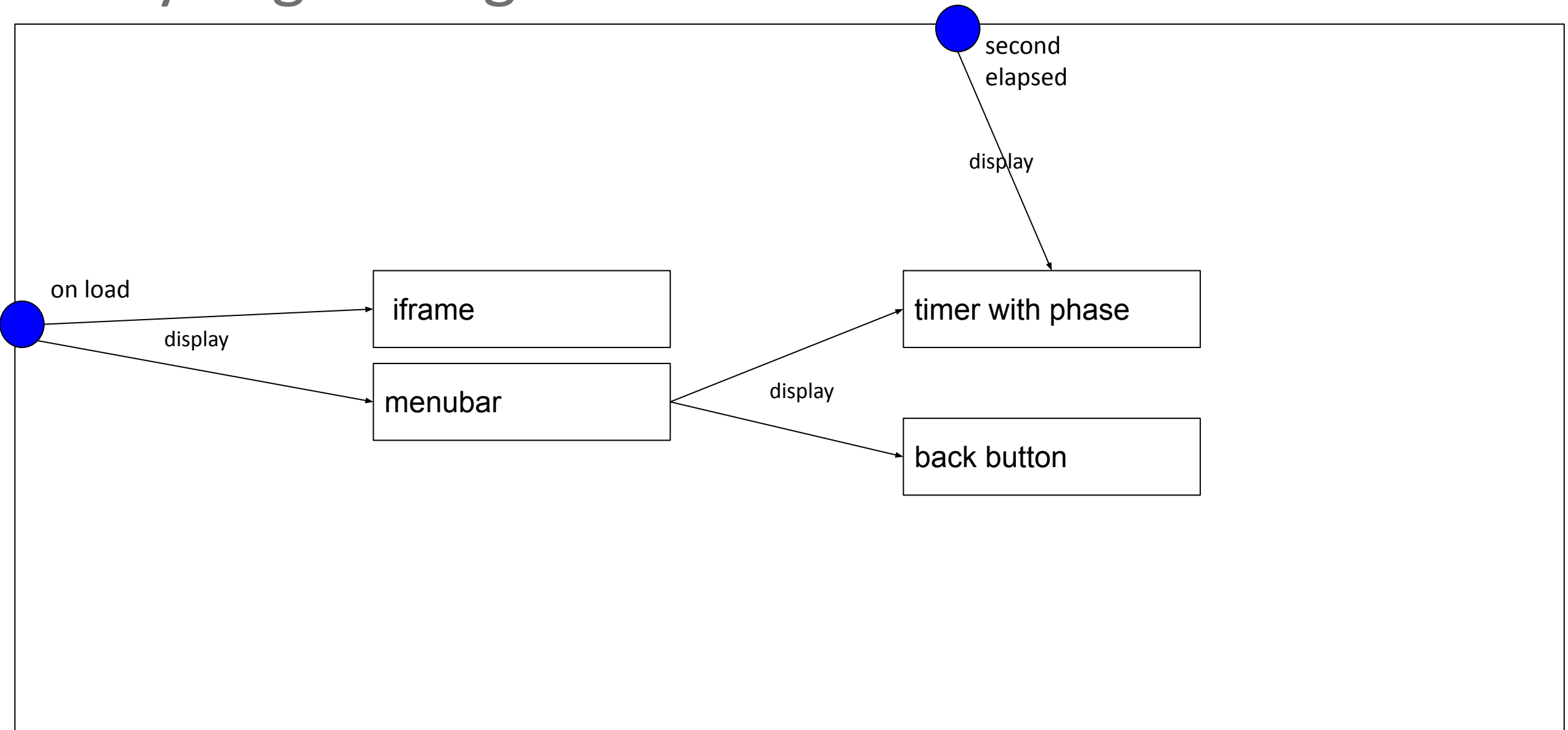
Homepage Design



Streaming page Design



Play Page Design



Frontend Main Components and Methods

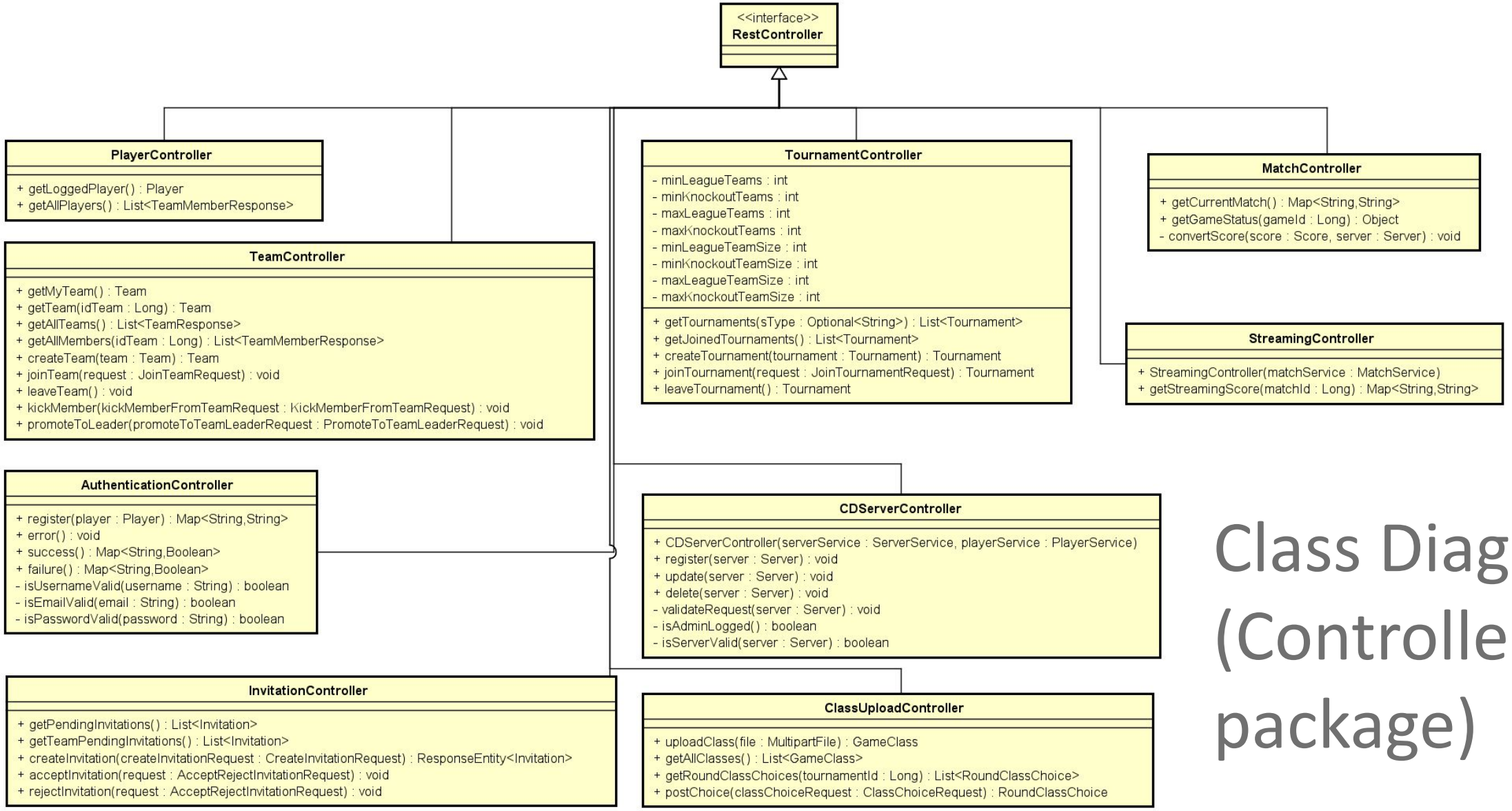
- Common main react methods
  - render() to display the content of each component
  - componentDidMount() to fetch data before rendering on section load
  - useEffect
- Home
  - mainPanel
- SignIn/Signup
  - renderErrorMessage
  - checkUsername
  - checkPassword
  - checkEmail
  - HandleSubmit
- TeamCreation
  - HandleSubmit
- TournamentCreation
  - handleSubmit
  - handleClassSubmit
- TeamManagement
  - handleClickInvite
  - handleClickLeave
  - handleClickKick
  - handleClickPromote
  - displayTeamInfo
  - displayTeamMember
- DisplayTournament
  - reloadPage
- ListTournamentComponent
  - refreshView
- TournamentEntry
  - SelectClass
  - DisplayClassUploading
  - JoinButton
  - DisplayTournamentInfo
  - JoinTournament
  - DisplayTeamForm
- MatchEntry
  - ShowCDFrame
- ListPlayers
  - handleSendInvite
- JoinTeam
  - TeamEntry
  - TeamMember
- InvitationEntry
- CdFrame
  - InitializeTimer
  - UpdateTimer
  - WaitStart
- Streaming
  - onMessage
- EventEntry
  - RetrieveEventDescription

Detailed backend software design

Backend has been implemented using Spring Boot and it's organized in three layers:

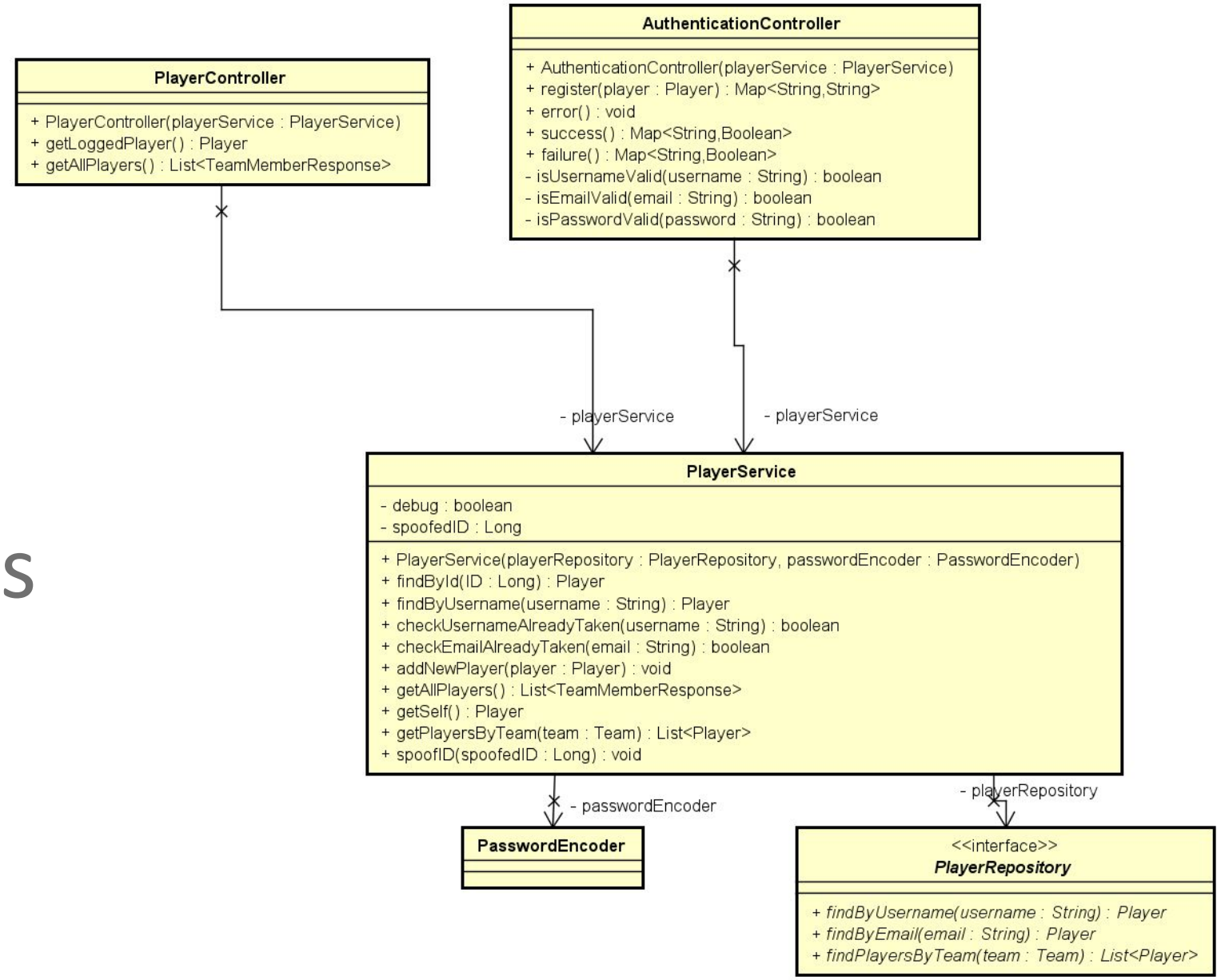
- Spring controllers: this layer interfaces with clients and receives their requests
- Spring services: this layer implements the business logic of the application
- JPA repositories and entities: this layer manages the communication with the database leveraging JPA functionalities

In the following slides we provide class diagrams describing backend components.

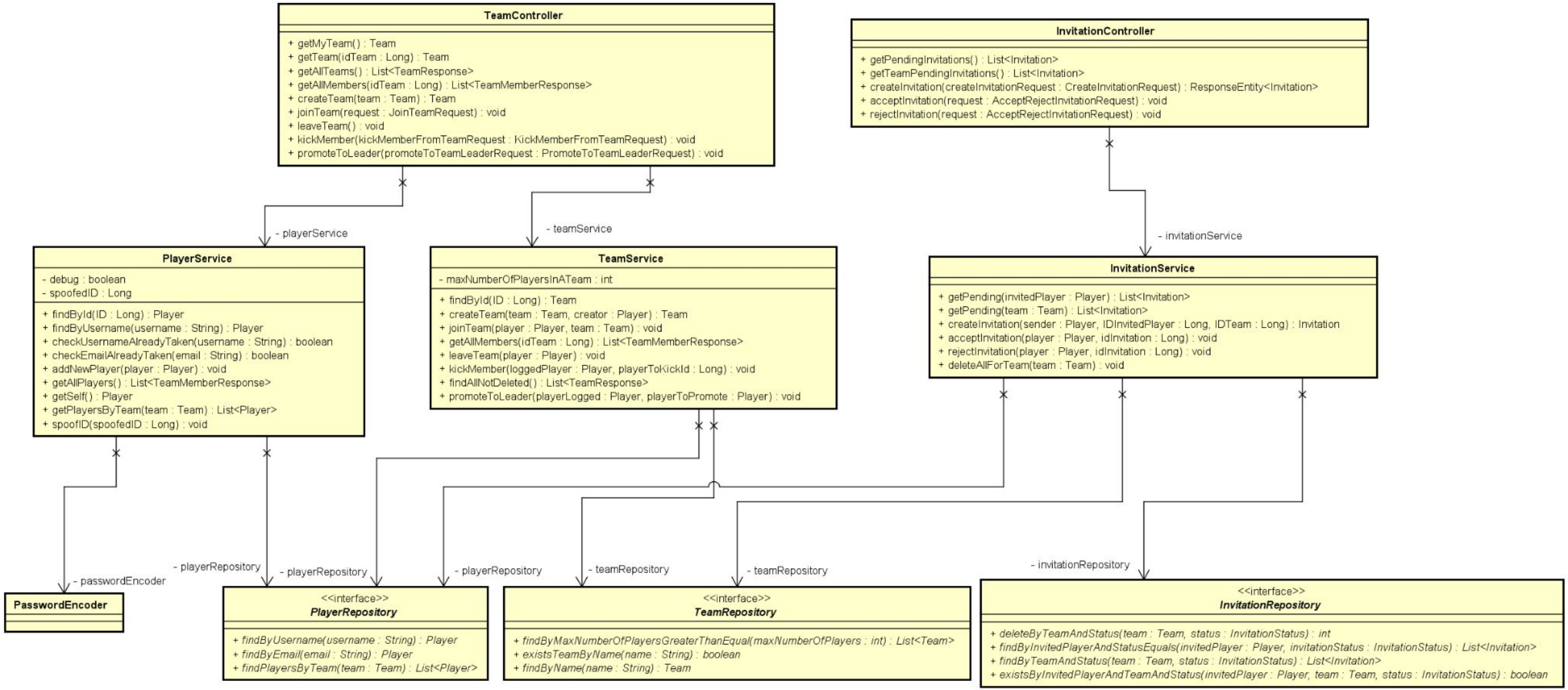


Class Diagrams (Controllers package)

Player and authentication class diagram



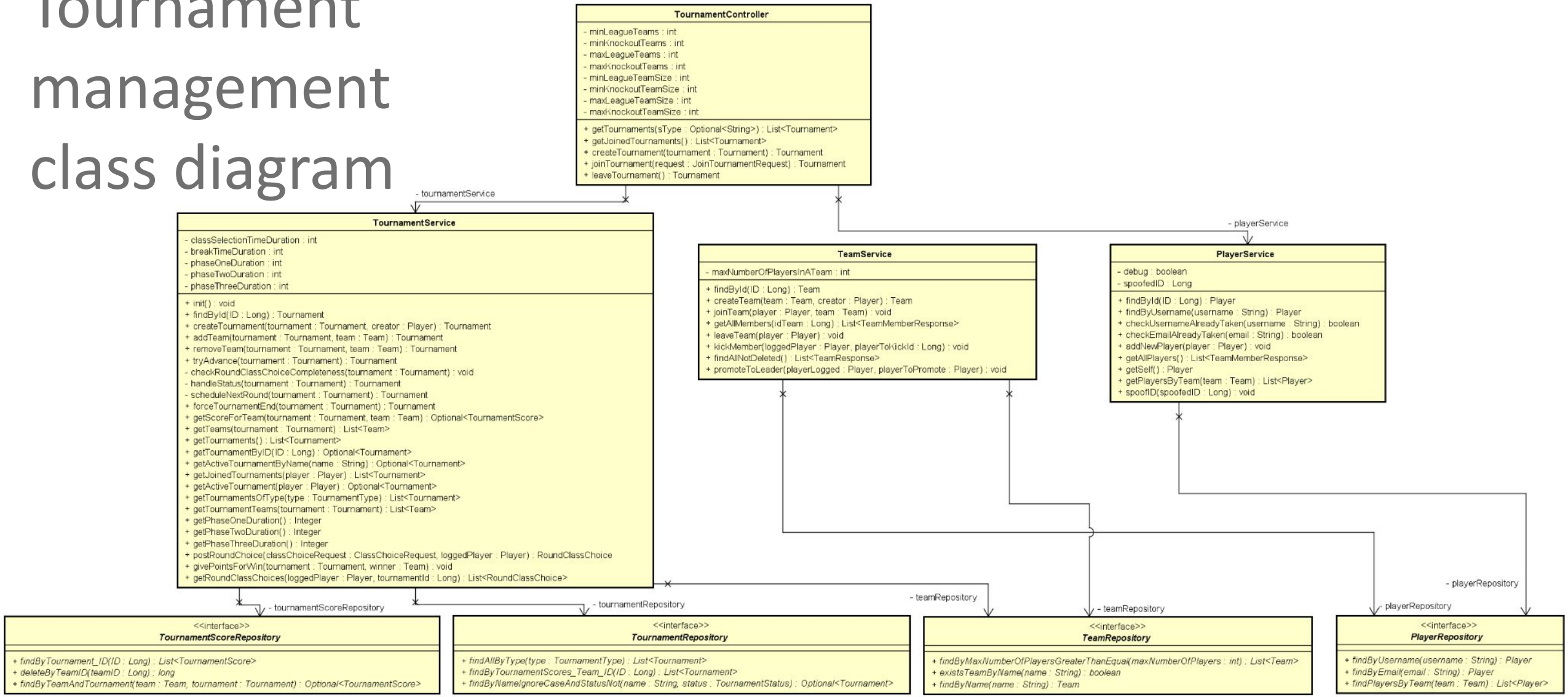
Some details have been omitted for readability



Team management class diagram

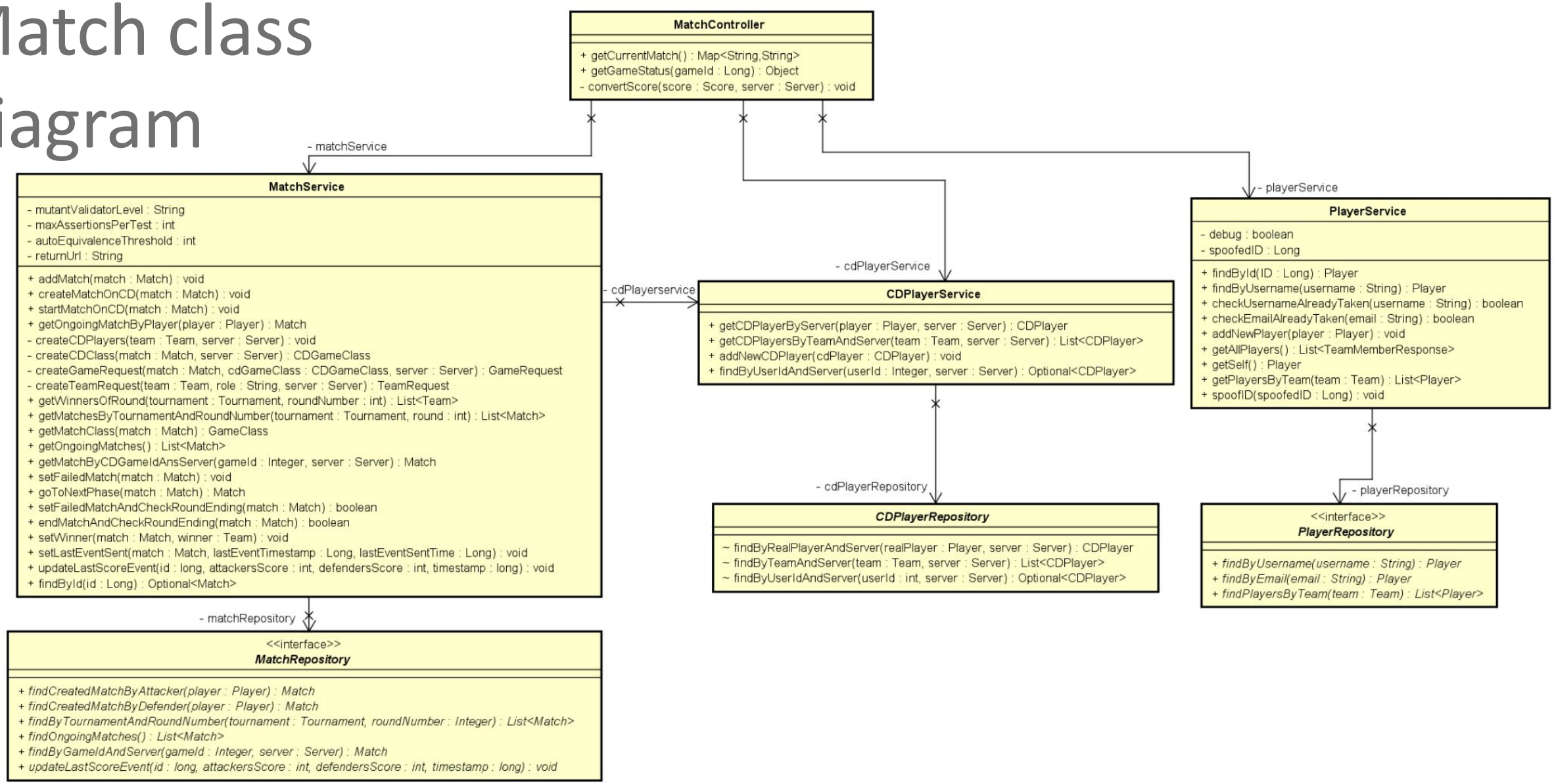
Some details have been omitted for readability

Tournament management class diagram



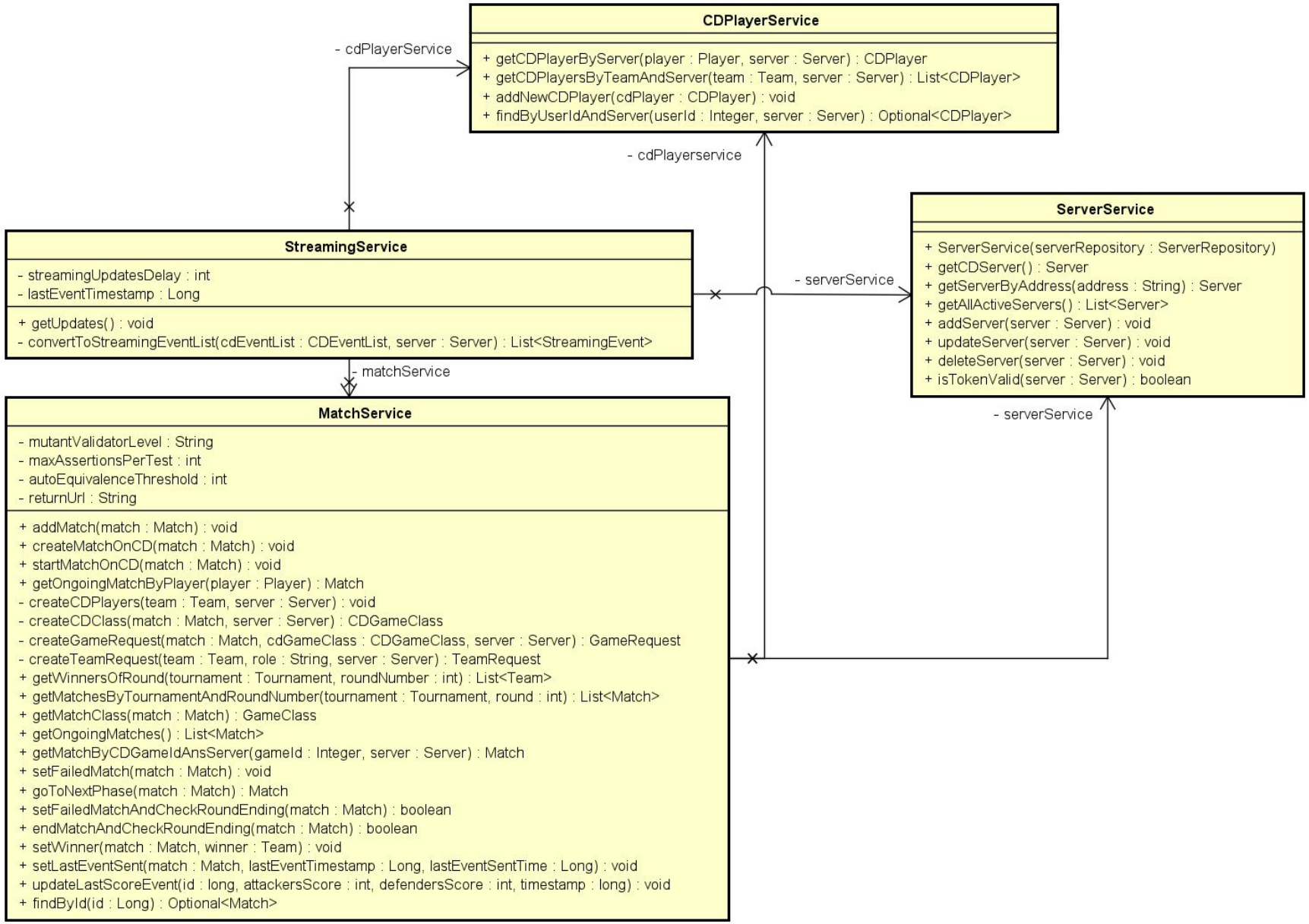
Some details have been omitted for readability

Match class diagram



Some details have been omitted for readability

Streaming component class diagram



Some details have been omitted for readability

Codebenders

contact info:

fanny.delnondedieu@fer.hr
dominik.brdar@fer.hr
hrvoje.rom@fer.hr
simone.mezzaro@mail.polimi.it
fabio.patella@mail.polimi.it
andrea2.restelli@mail.polimi.it