Software Architecture Overview

Golf-Matcher

SWENG 894

December 9, 2019

Dan Castellucci

William Ganley

Peter Jester

Craig Roland

1. Introduction

1.1 Purpose

The Golf-Matcher application was created to allow multiple golf leagues to consolidate functionality into an easy to use user interface with high accessibility. We wanted to give users high availability access to add, edit, delete, and view the following:

- Scores
- Handicaps
- Points
- Standings
- Schedule

Prior to Golf-Matcher, golf leagues used very complex spreadsheets and required regular print outs of standings, handicaps, and schedules. It also required Admins to spend hours of time each week entering scores, and maintainability of the source code was extremely difficult.

The application is built using the Angular platform and framework that uses HTML and Typescript. The basic building blocks of the application are Angular Components and Services.

Angular Components are code grouped logically to control the display of a view. The component has a template that contains html code that tells Angular how the render the view. The final part of the Component is the styling of the view with Cascading Style Sheets (css).

Angular Services are typically a class with a single purpose to provide something back to the caller. Services are used to separate Components from other non-rendering functionality like data access.

The data for the Golf-Matcher application is stored in a Firebase database. Firebase is a cloud database that uses NoSQL to store data. Data is stored in JSON format and synchronized in real-time to all connected clients. Firebase supports multiple platforms like Android, iOS, and Javascript through several Software Development Kits (SDK).

1.2 Scope

The Golf-Matcher application is made up of several components that work together to provide the user with the correct view of the correct data. The components for the application are as follows.

- Authentication - Credential management to provide security
- Dashboard - Front page of the Golf-Matcher application
- Handicap - Calculate player handicaps from entered scores
- Leaderboard - Snapshot view of the current player/team standings
- Match - Create a competition between two teams at a scheduled time/date
- Player-Add - Add a player to the system
- Player-Edit - Edit an existing player information
- Players - Front page for player information and management
- Points - Calculate the points resulting from matches
- Schedule - Create a timeline for matches
- Scores -Edit - Edit existing scores information
- Sidebar - Navigation links displayed on the dashboard
- Team-Add - Add a team to the system
- Team-Edit - Edit an existing team information
- Teams - Front page for team information and management
- Util - Utility functions for handicap calculation algorithm

1.3 Definitions, Acronyms, and Abbreviations

HTML - Hyper Text Markup Language
CSS - Cascading Style Sheets
NoSQL - Not Only SQL alternative to relational databases which doesn't use a schema for data
SQL - Software Query Language
JSON - JavaScript Object Notation open-standard file format that uses human-readable text
SDK - Software Development Kit

1.4 References

1) Unknown Author. "Angular". Version: 8.2,.15-local+sha.bbeac0727b. 2019. https://angular.io

2. Architecture Goals and Constraints

Scalable Architecture
The Golf-Matcher application was developed using the Angular framework. The application is made up of several components and services that are responsible for a discrete view or data set. The application is easily scalable by adding additional components to support any needed views. New data can be accessed by adding a new service responsible for reading/writing the data.

Easy to Understand
The Golf-Matcher application is easy to understand. The component names match the data that will be displayed in the corresponding view. The service names match the data that is being read/written.

For example, the Players component is responsible for the Players view. This view allows the user to see all the information related to a Player. This information includes the following:

- Name
- Nickname
- Age
- Email
- Handicap
- League
- Scores

The Player data is read using the PlayersService which connects to the Firebase database and fetches the data. There are convenience functions to get player data, add a new player, update an existing player, delete an existing player, and find a player by name.

3. Use-Cases

Name: Register
Goal in Context: Users need a way to sign up to use the system.
Primary Actor: Users
Scenario: Users/Golfers will go to the home page, and be presented with login or register. Users will select register, and be able to register with the system.

Name: Register as Administrator

Goal in Context: Privileged users need a way to sign up to use the system.
Primary Actor: Privileged Users/Administrators
Scenario: Privileged Users/Administrators are able to register to gain access to the system.

Name: Sign In
Goal in Context: Users need a way to sign in to the system
Primary Actor: Users
Scenario: Users/Golfers are able to navigate to the website, and authenticate themselves with the server to gain the correct access to the system.

Name: Handicaps View
Goal in Context: Users will need a way to view handicaps
Primary Actor: Users/Admins
Scenario: Users and admins navigate to the Handicaps page, and will receive a view that displays the handicaps for all golfers in their league.

Name: Scores View
Goal in Context: Users will need a way to view scores
Primary Actor: Users/Admins
Scenario: Users and Admins will select the scores view. System will bring up new view that will show a list of all users and their current week scores if one is entered.

Name: Standings View
Goal in Context: Users will need a way to view their points for the season
Primary Actor: Users/Admins
Scenario: Users and Admins will select the standings view. System will bring up new view that will show a list of all teams, and their current standings, and total points for the season.

Name: Schedule View
Goal in Context: Users will need a way to view the schedule for the season
Primary Actor: Users/Admins
Scenario: Users and Admins will select the schedule view. System will bring up new view that will show a list of all teams, and their schedule. For each week.

Name: Add Golfer
Goal in Context: Admins will need to add a golfer
Primary Actor: Admins
Scenario: Admins will add a new golfer to the system. User will sign in, and see scores, handicaps, points, shot tracker for all golfers in league.

Name: Edit Golfer

Goal in Context: Admins will need to edit a golfer
Primary Actor: Admins
Scenario: Admins will edit a golfer already in the system.

Name: Delete Golfer
Goal in Context: Admins will need to Delete a golfer
Primary Actor: Admins
Scenario: Admins will Delete a golfer from the system.

Name: Add Team
Goal in Context: Admins will need to add a Team of 2.
Primary Actor: Admins
Scenario: Admins will add a new team to the system. Admins will select 2 users to create the team.

Name: Edit Team
Goal in Context: Admins will edit an already existing team.
Primary Actor: Admins
Scenario: Admins will select a team to edit. Admin will then be able to change the users who create the team, and team name.

Name: Delete Team
Goal in Context: Admins will need to delete a Team from the system.
Primary Actor: Admins
Scenario: Admins will select a team, and be prompted to remove the team from the system.

Name: Edit Schedule
Goal in Context: Admins will edit a schedule.
Primary Actor: Admins
Scenario: Admins will have a generated schedule already, and will be able to edit values. The admin will be responsible for solving any conflicts in scheduling if there are edits.

Name: Delete Schedule
Goal in Context: Admins will Delete a schedule.
Primary Actor: Admins
Scenario: Admins will need a way to Delete a schedule. Once the season is over we will way to soft delete the schedule, this way we can create a new one for the following season.

Name: Add Scores
Goal in Context: Admins will need a way to add scores
Primary Actor: Admins

Scenario: Admins will see a grid of golfers, and be able to quickly add scores for each golfer.

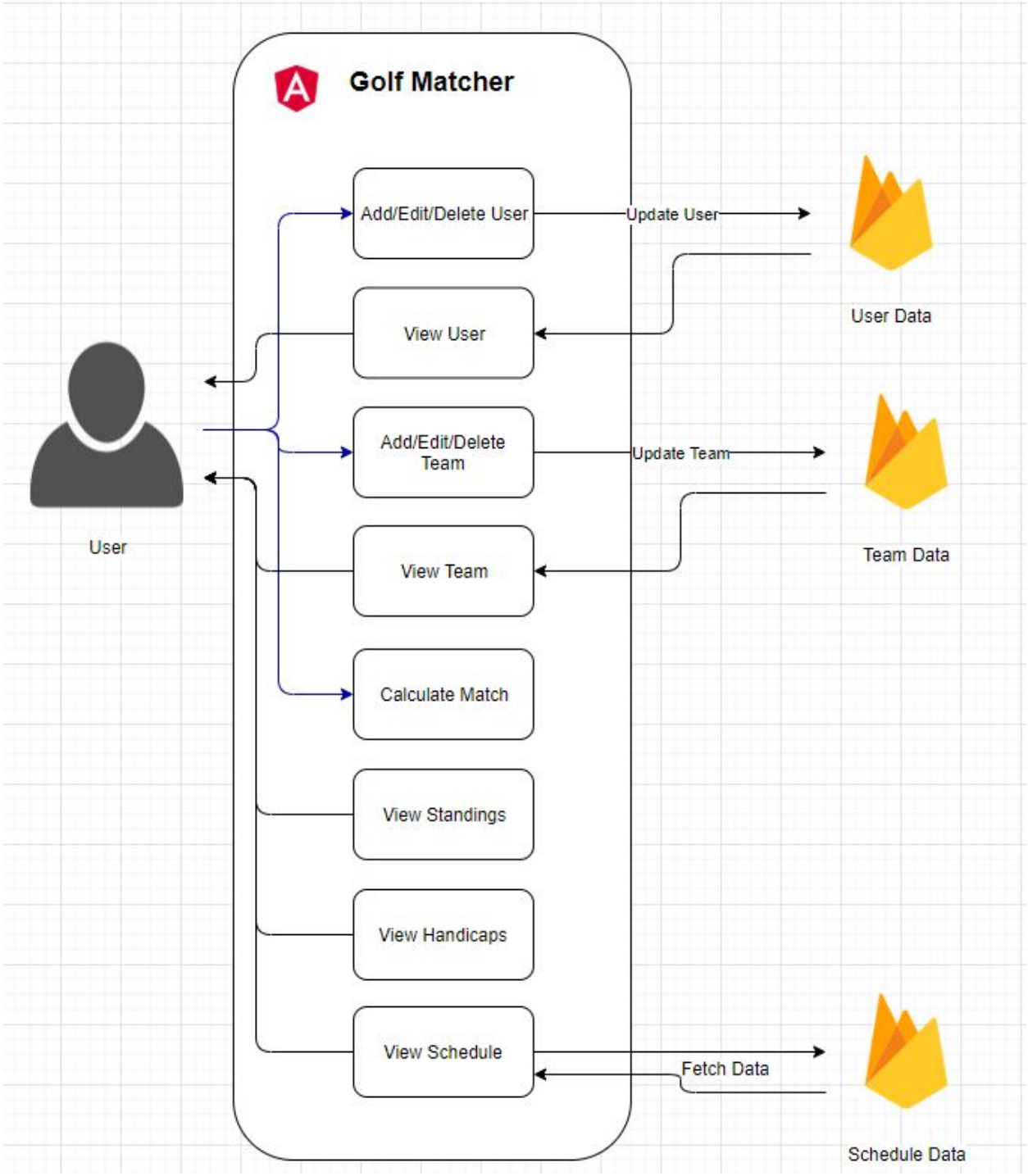
Name: Edit Scores
Goal in Context: Admins will need a way to edit scores
Primary Actor: Admins
Scenario: Admins will see a grid of golfers, if an error occurs we will need to make an adjustment. Admins should be able to select the week, and golfer who they want to edit. The system will then calculate points and handicaps based on the new values.

Name: Delete Scores
Goal in Context: Admins will need a way to delete scores
Primary Actor: Admins
Scenario: Admins will see a grid of golfers, if scores were entered for a user by mistake who did not play we want a quick way for admins to clear the scores for that week.

Name: Calculate Handicaps
Goal in Context: System shall calculate handicaps based on scores entered personal and league scores.
Primary Actor: System
Scenario: Whenever a new score record is entered for a user. The system shall calculate the handicaps for that user. System shall calculate a handicap for league play and personal play.

Name: Calculate points
Goal in Context: System shall calculate points based on league scores.
Primary Actor: System
Scenario: Whenever a new score is added for league play the system shall calculate points based on schedule. System shall read from schedule and determine points from scores entered for team x and team y.

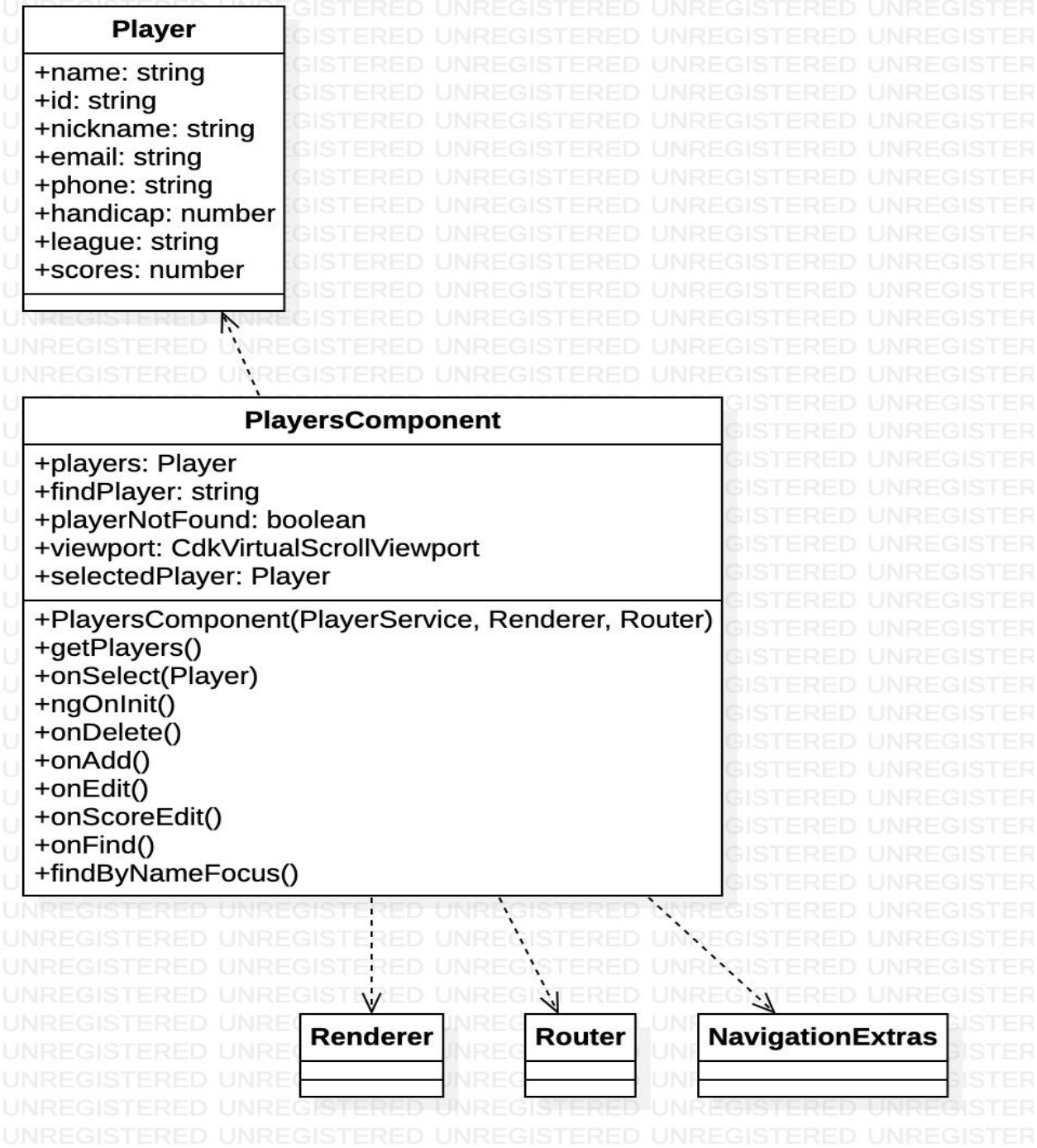
3.1 Use-Case View



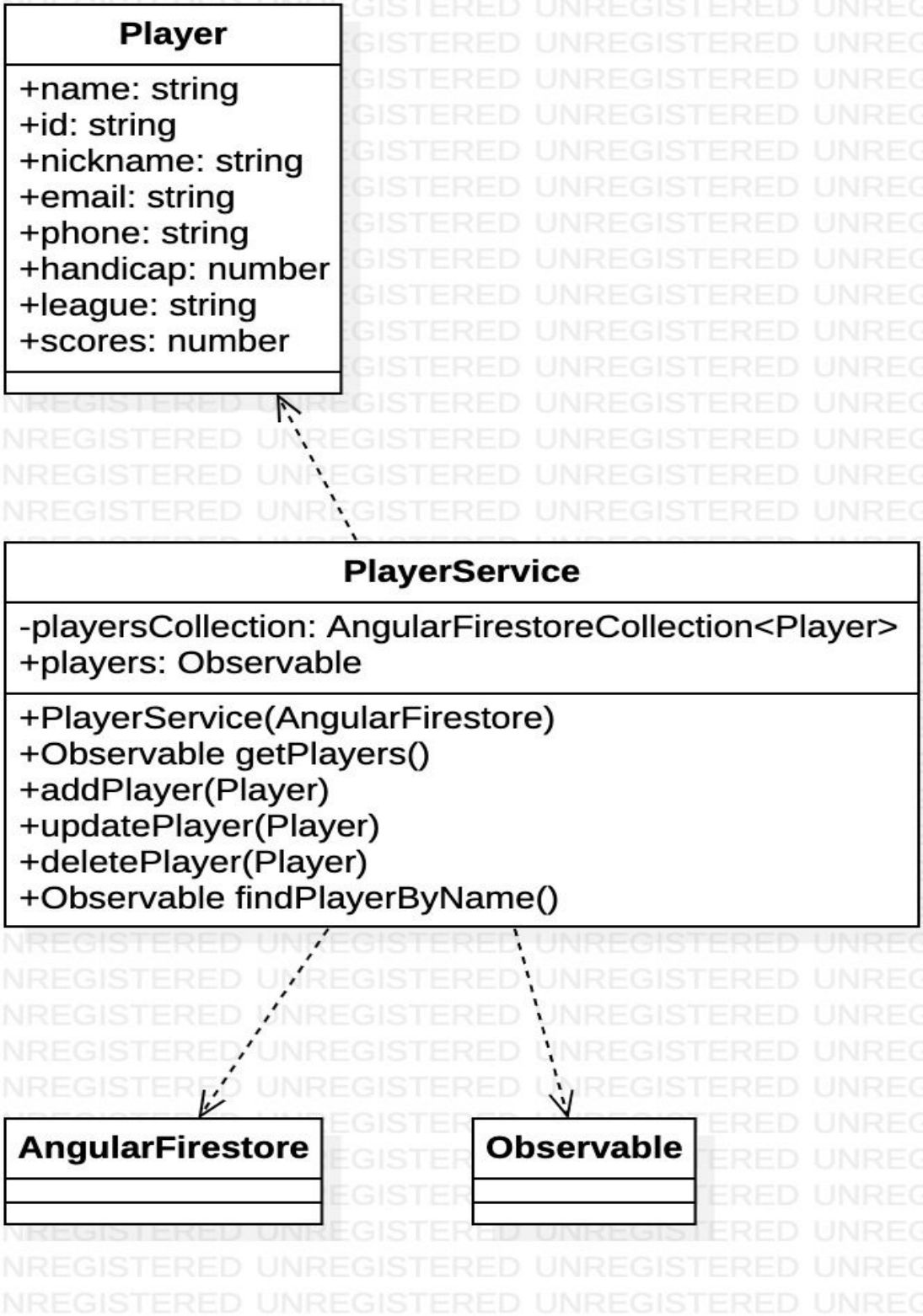
4. Logical View

CLASS DIAGRAMS

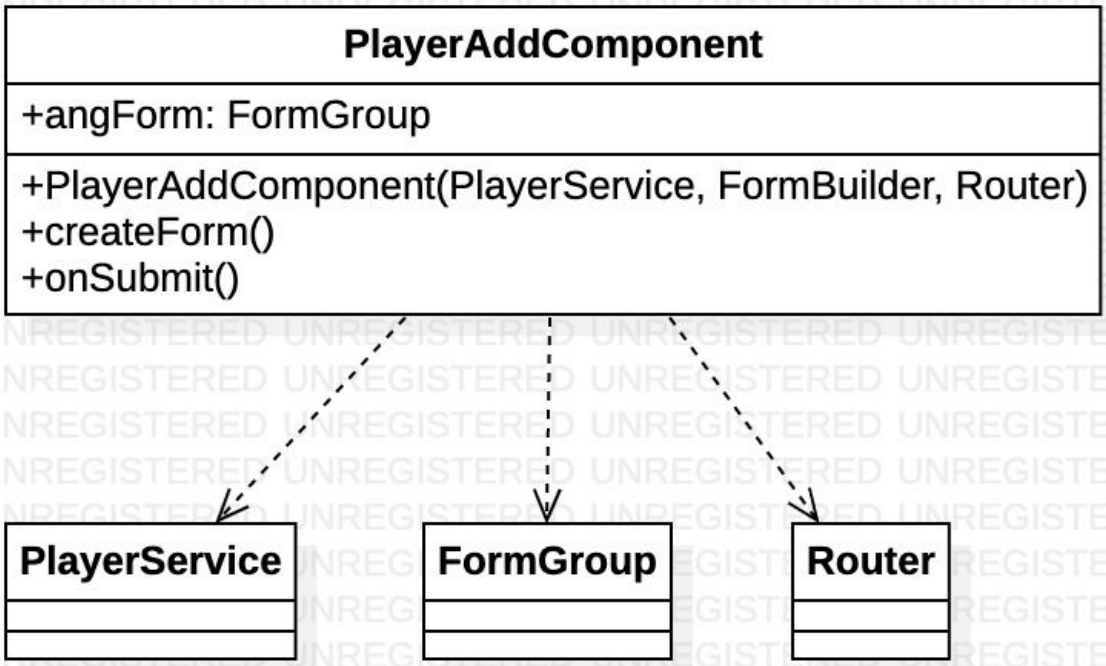
Players



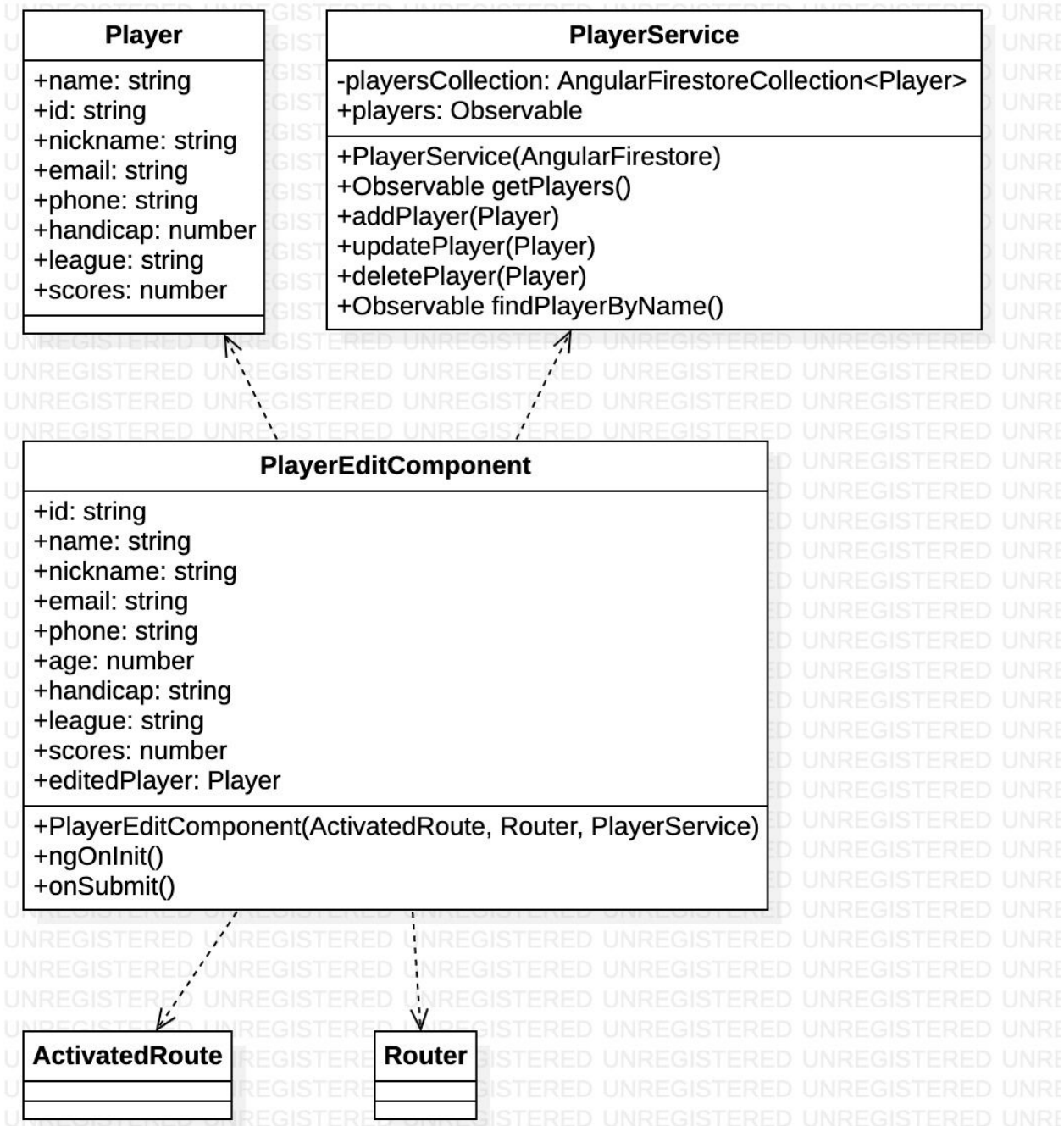
Player Service



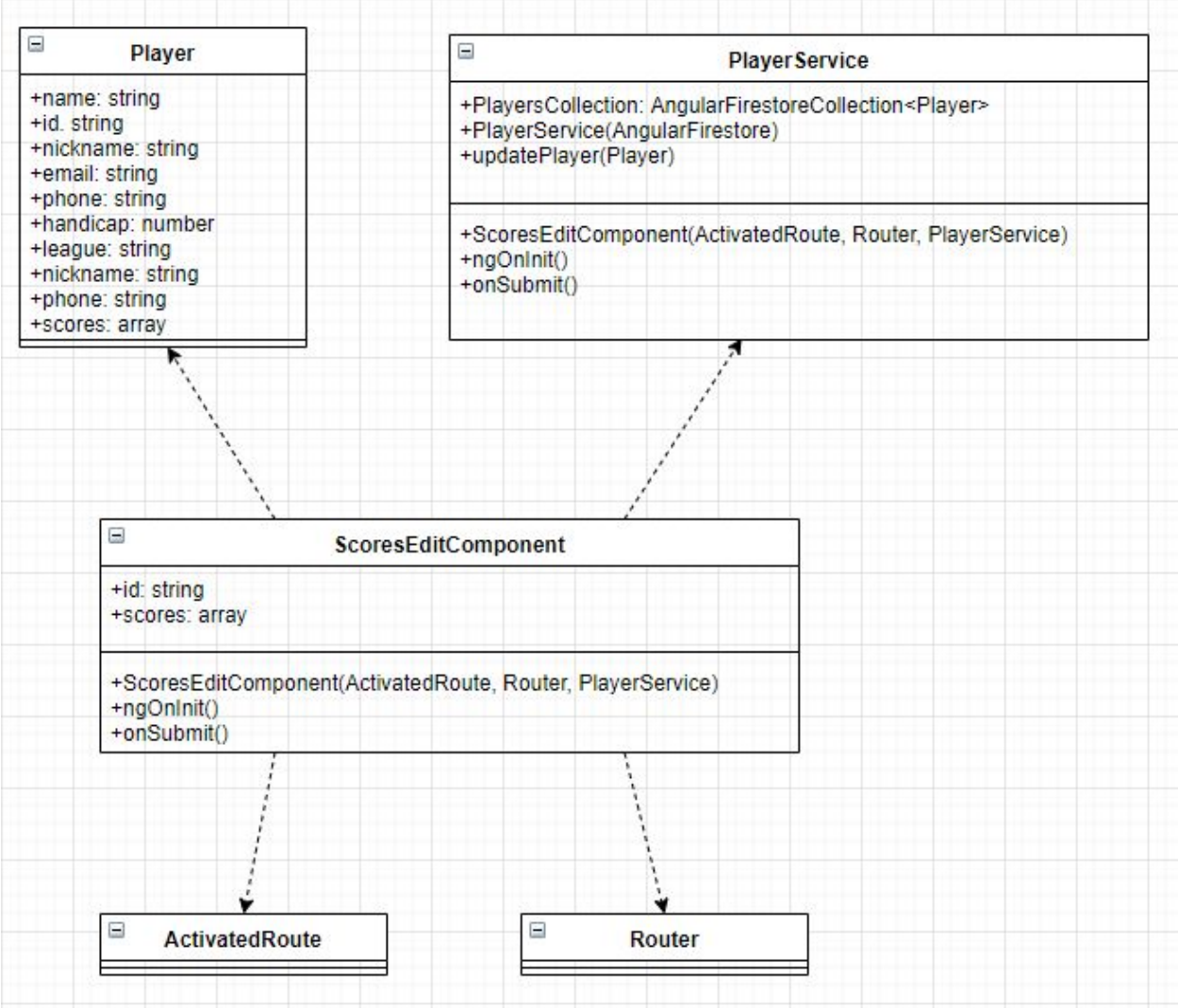
PlayerAddComponent



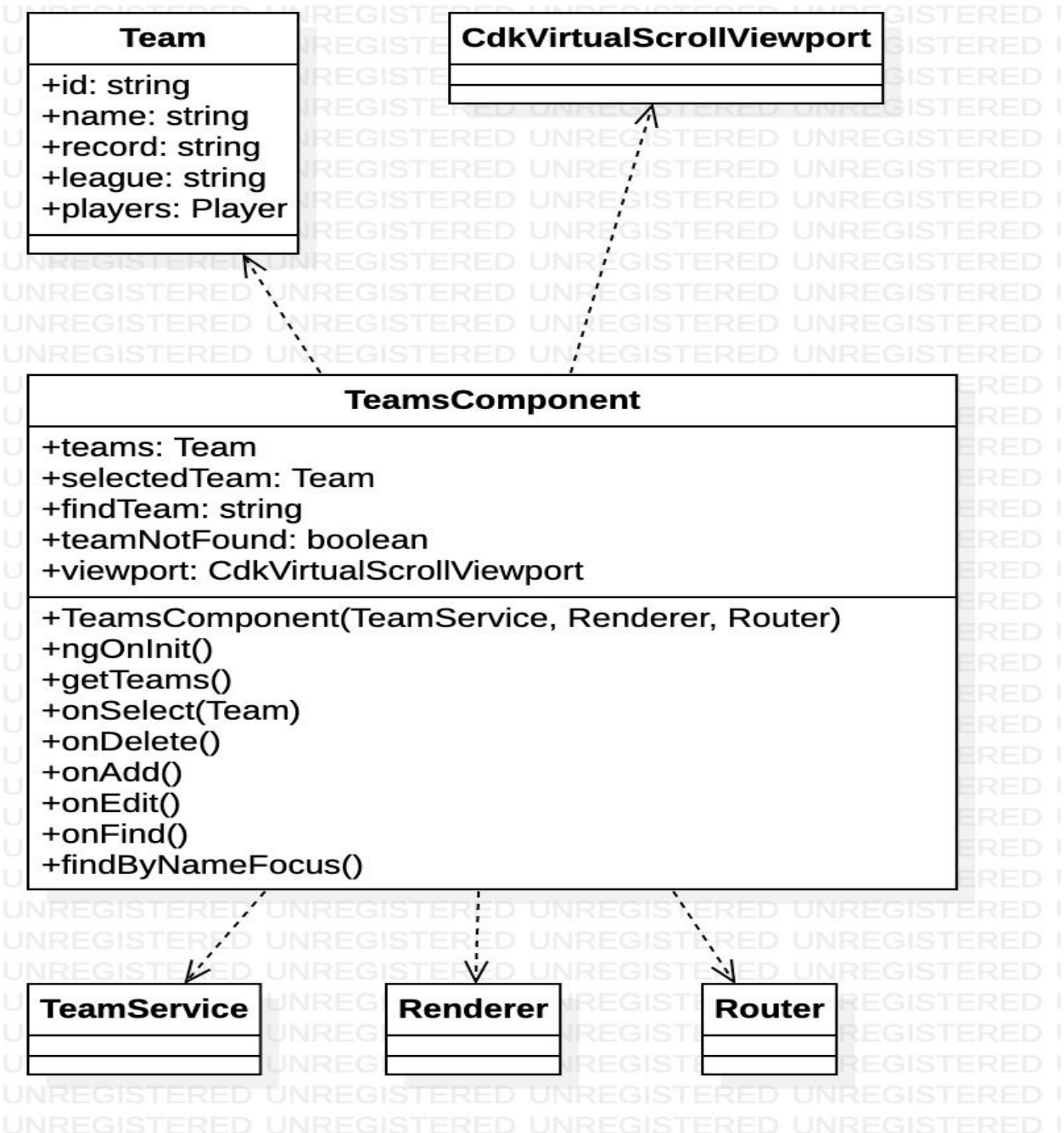
PlayerEditComponent



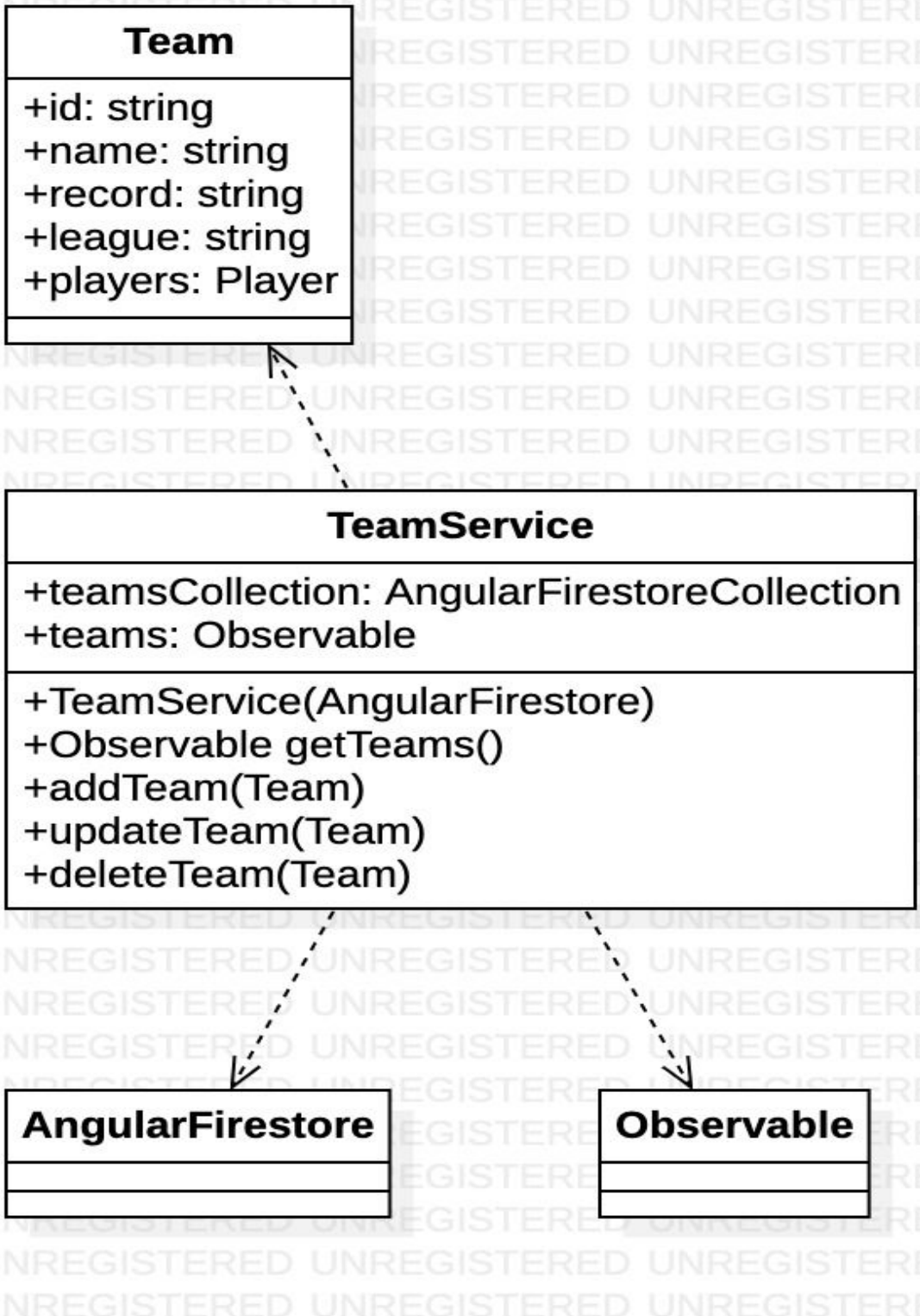
ScoresEditComponent



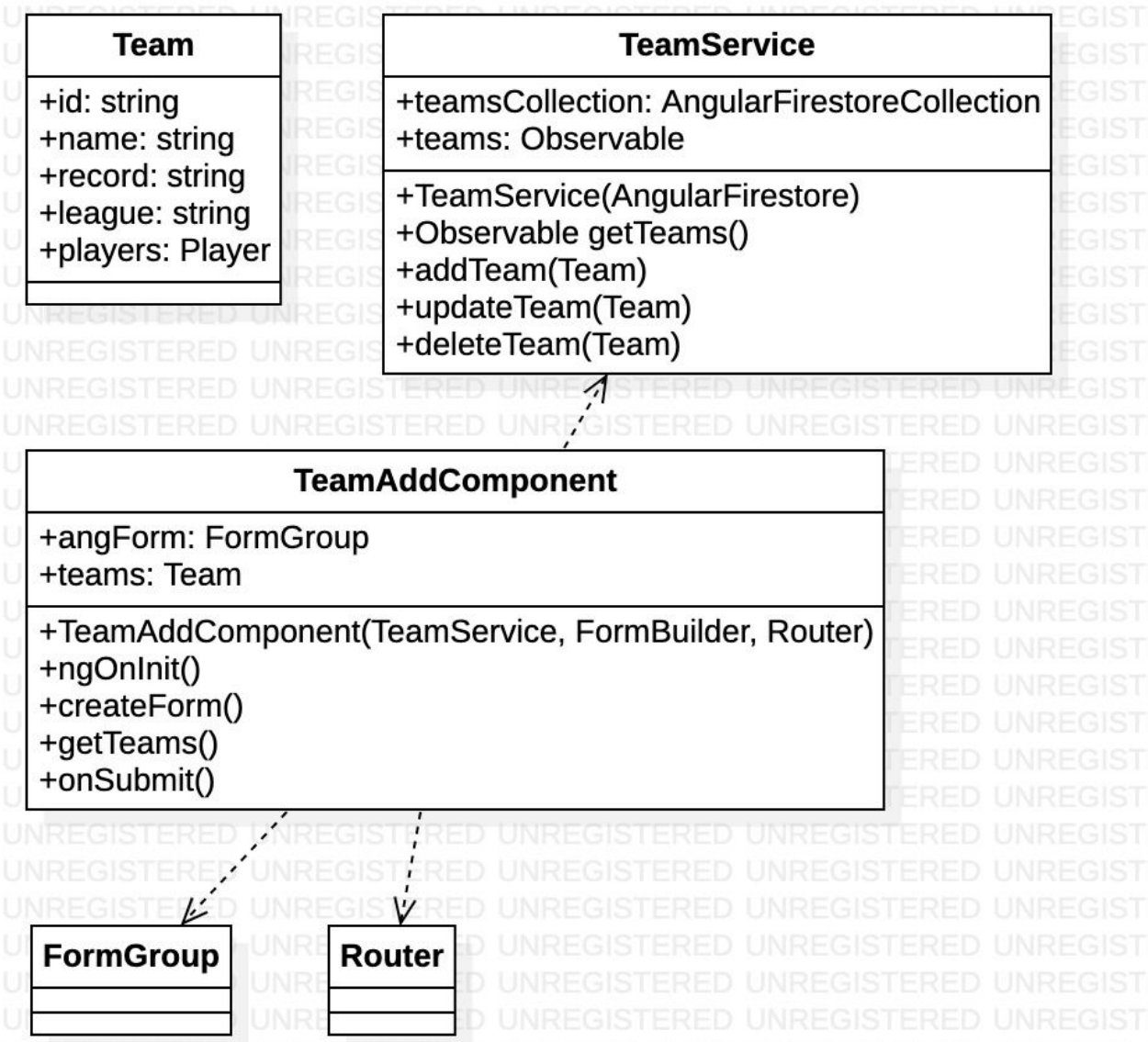
Teams


Team
+id: string
+name: string
+record: string
+league: string
+players: Player
CdkVirtualScrollViewport
TeamsComponent
+teams: Team
+selectedTeam: Team
+findTeam: string
+teamNotFound: boolean
+viewport: CdkVirtualScrollViewport
+TeamsComponent(TeamService, Renderer, Router)
+ngOnInit()
+getTeams()
+onSelect(Team)
+onDelete()
+onAdd()
+onEdit()
+onFind()
+findByNameFocus()
TeamService
Renderer
Router

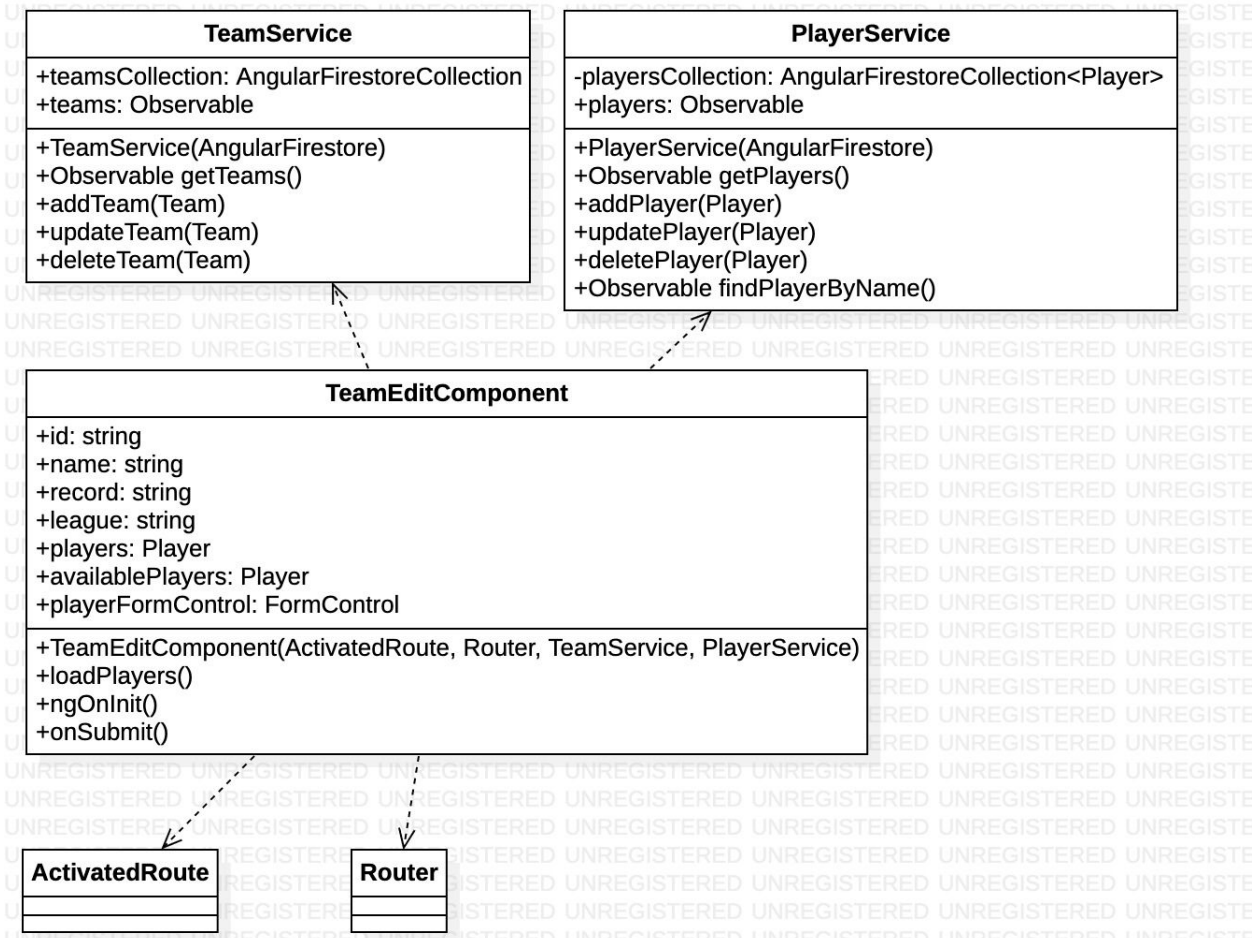
TeamService



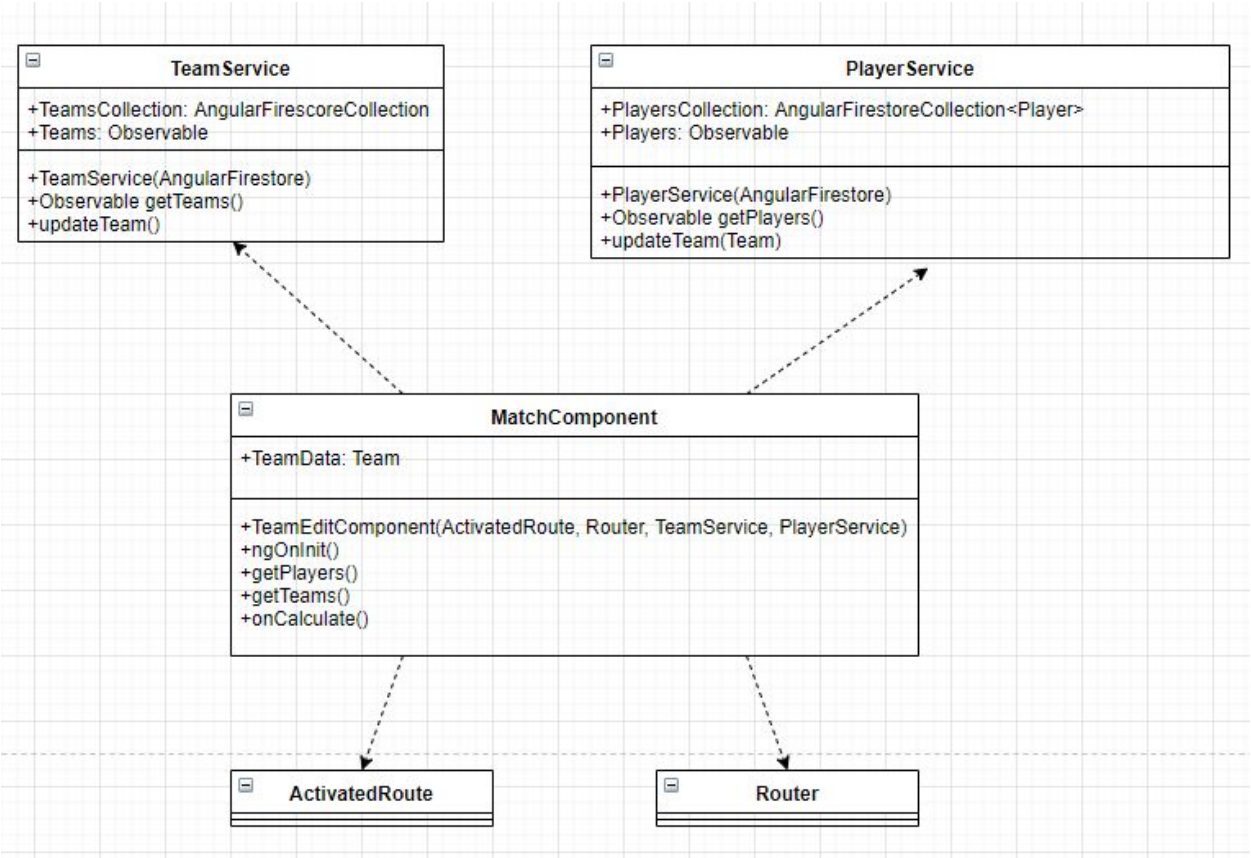
TeamAddComponent



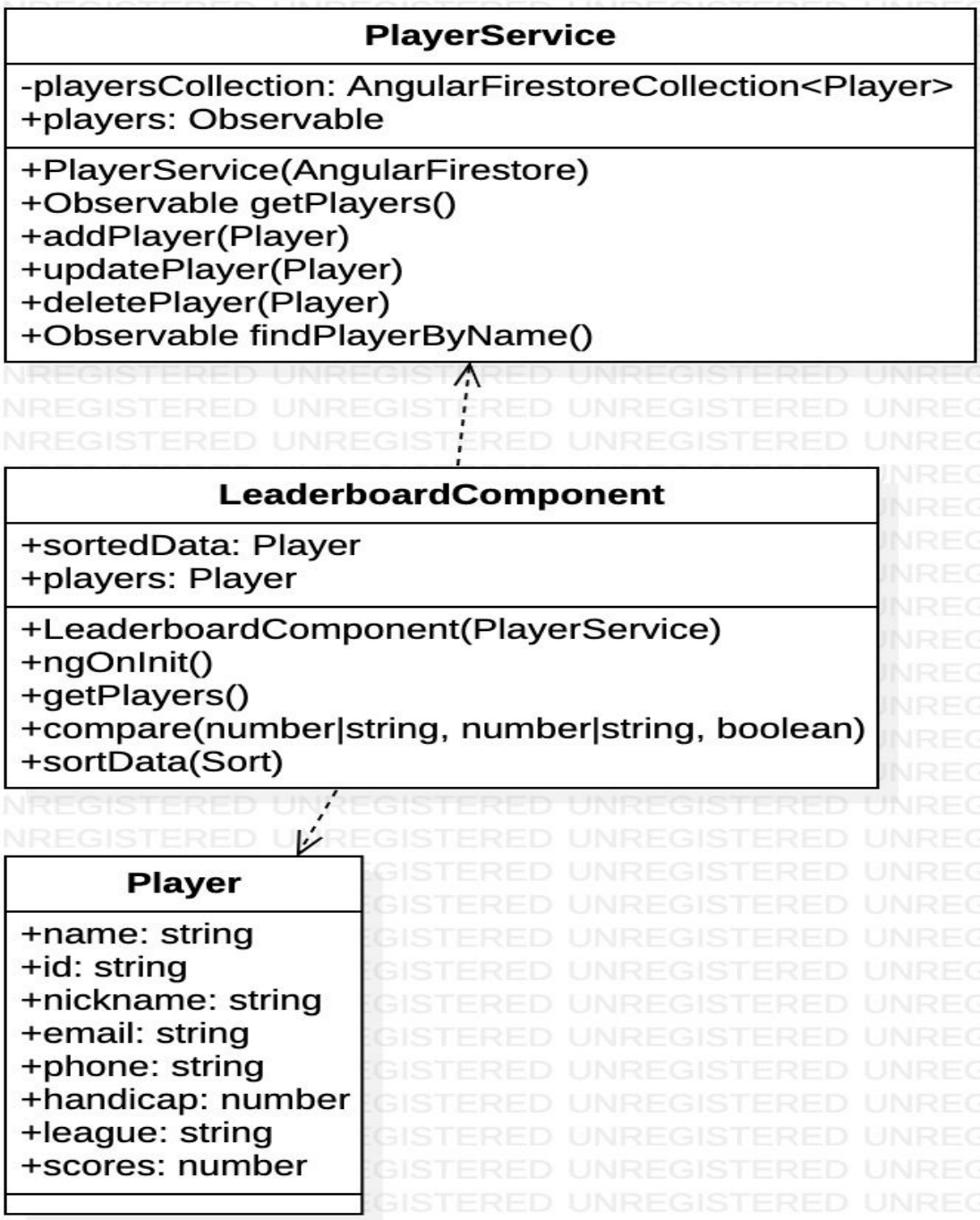
TeamEditComponent



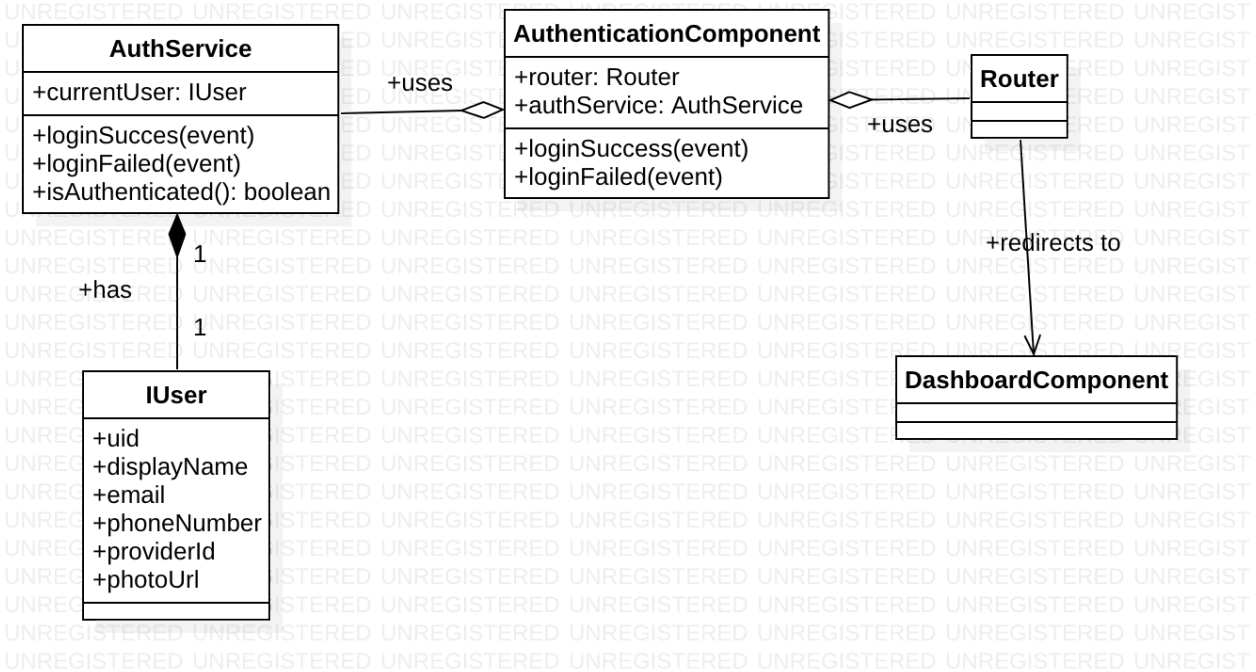
MatchComponent



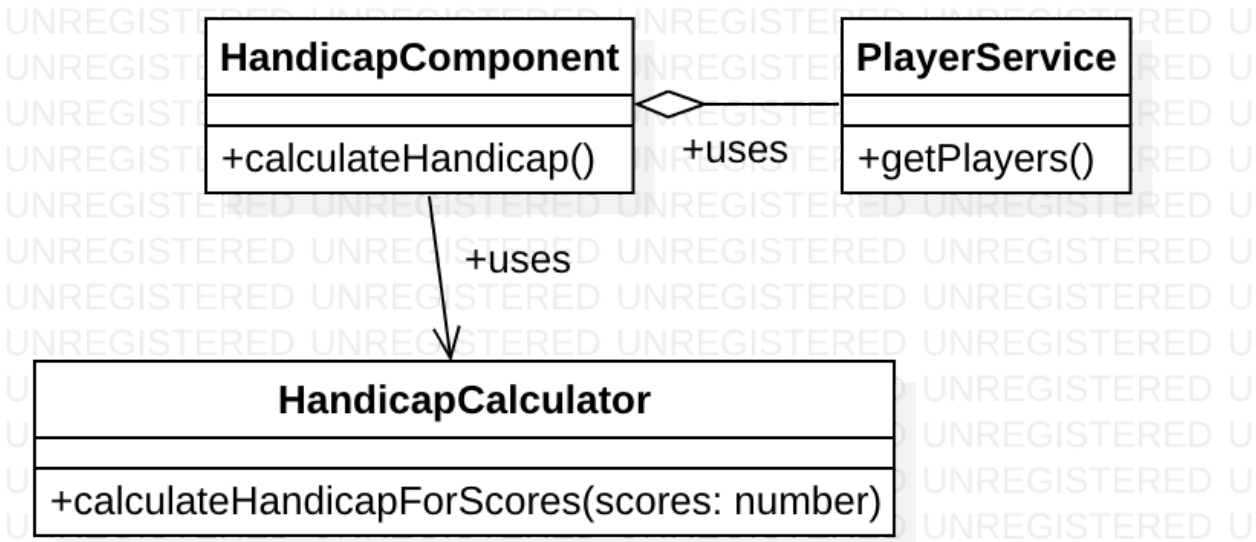
Leaderboard



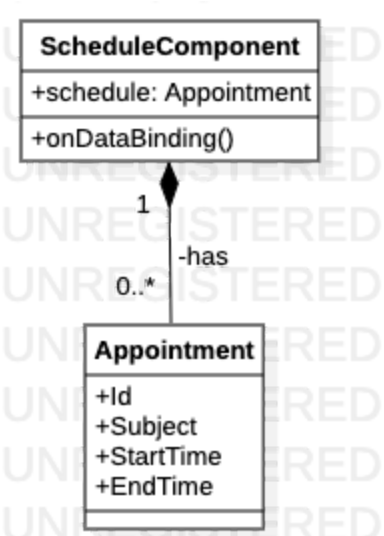
Authentication



Handicap

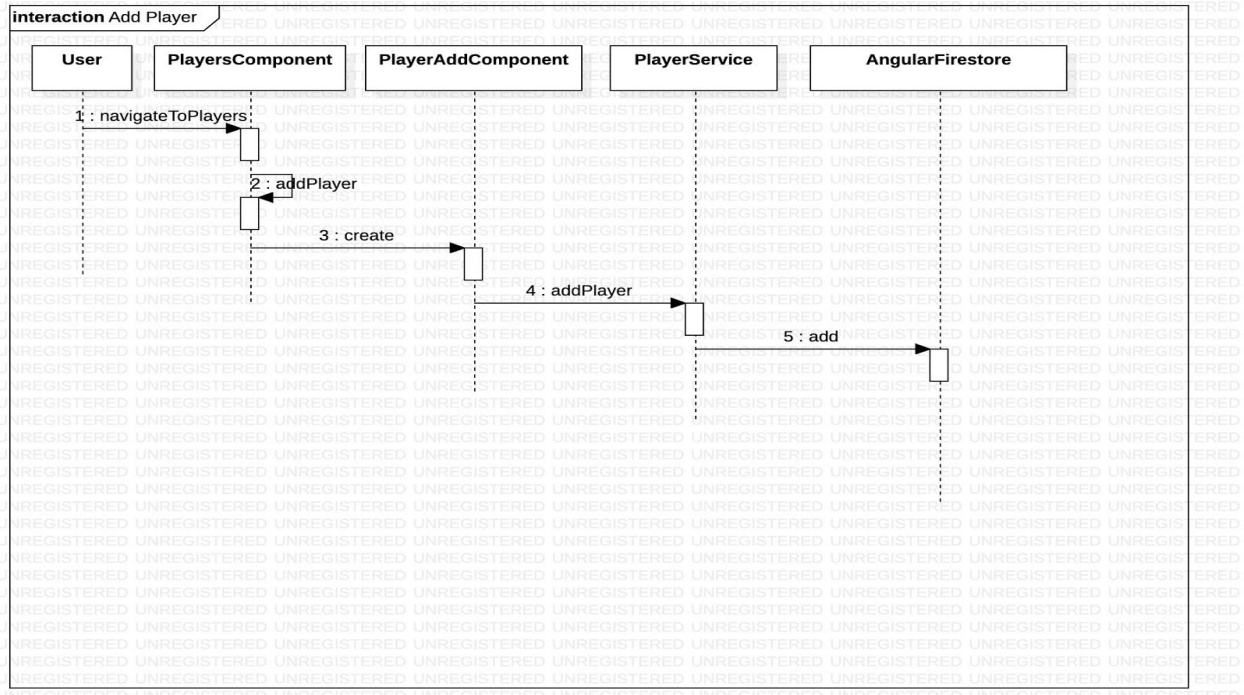


Schedule

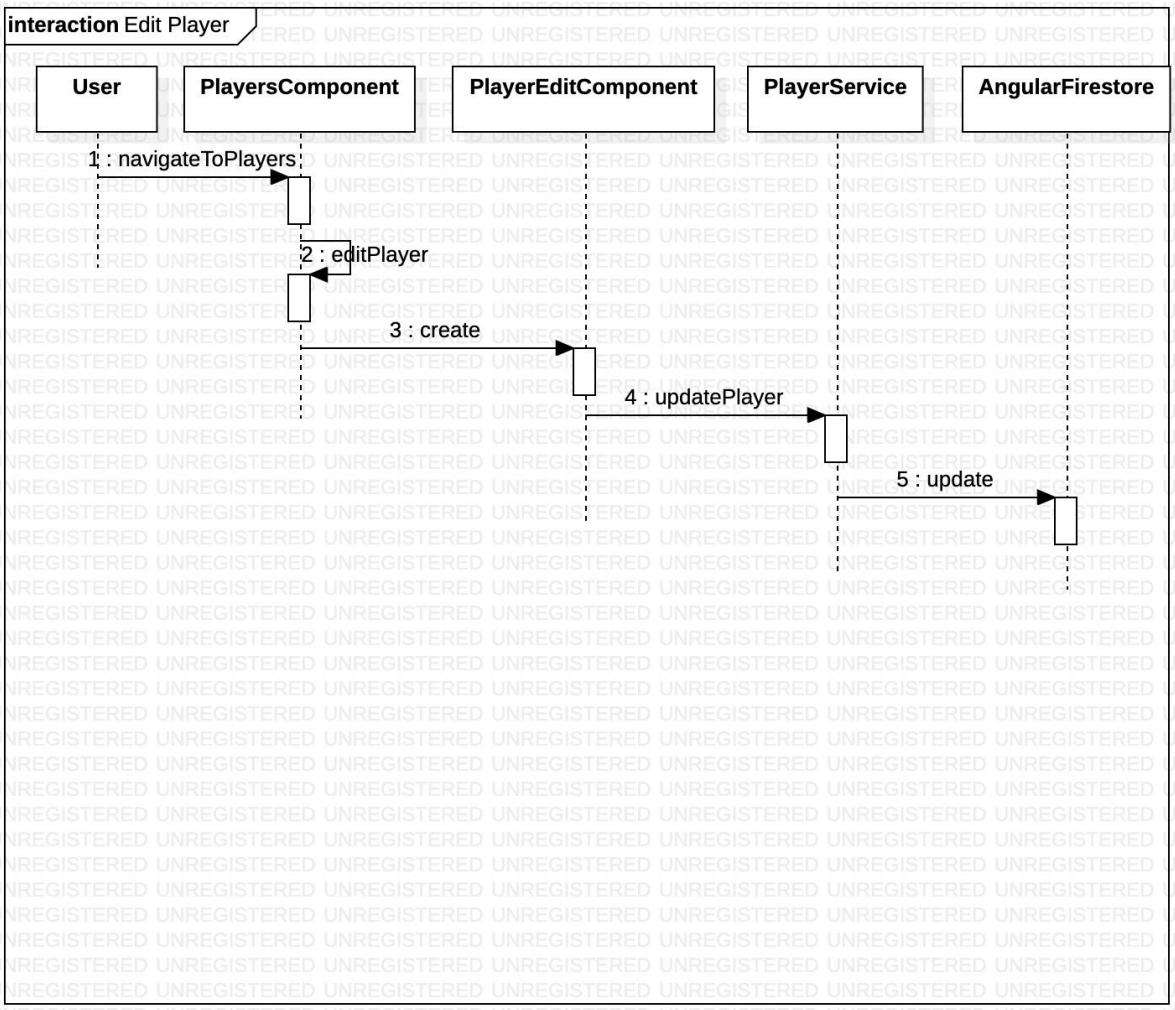


SEQUENCE DIAGRAMS

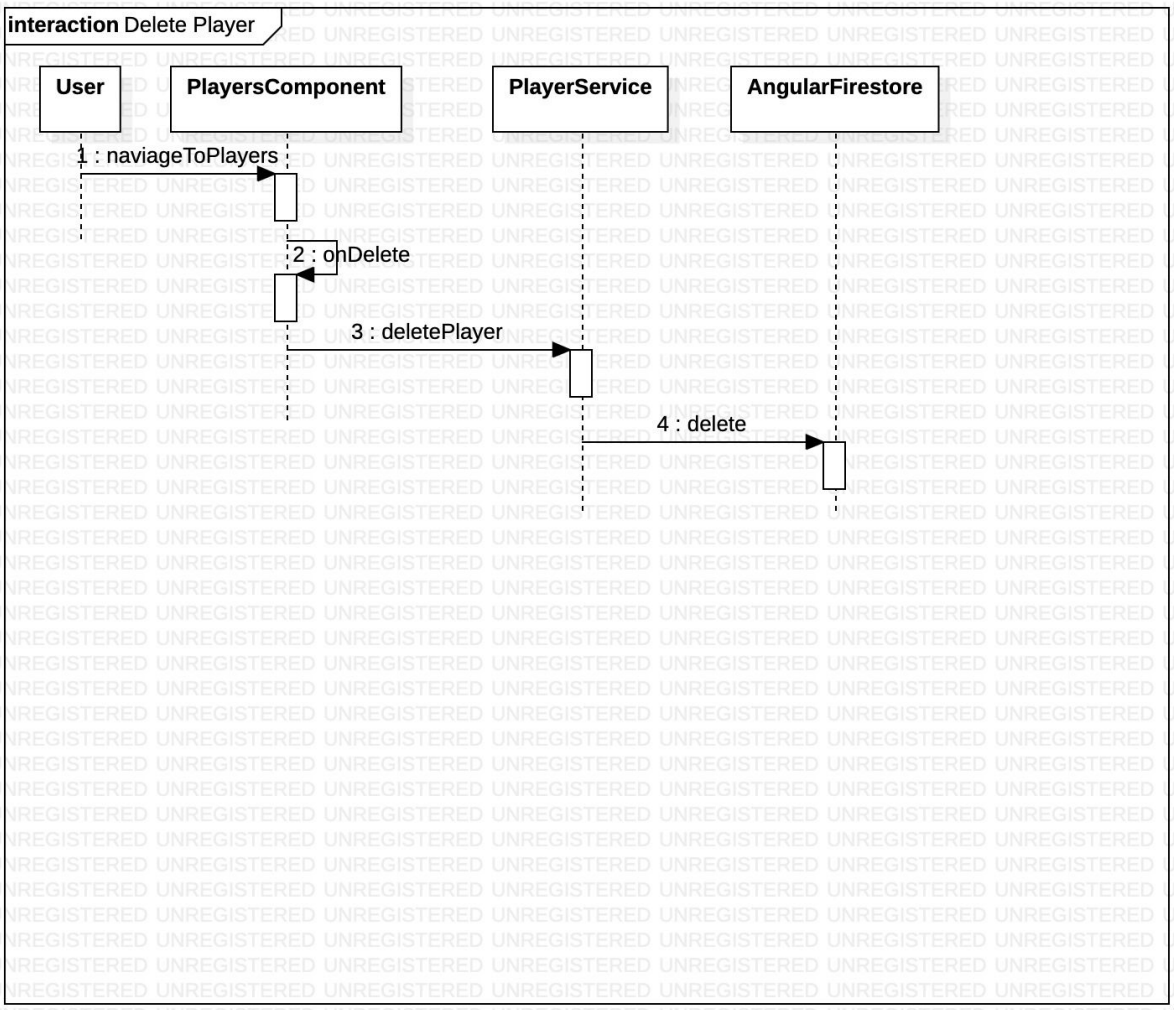
Add Player



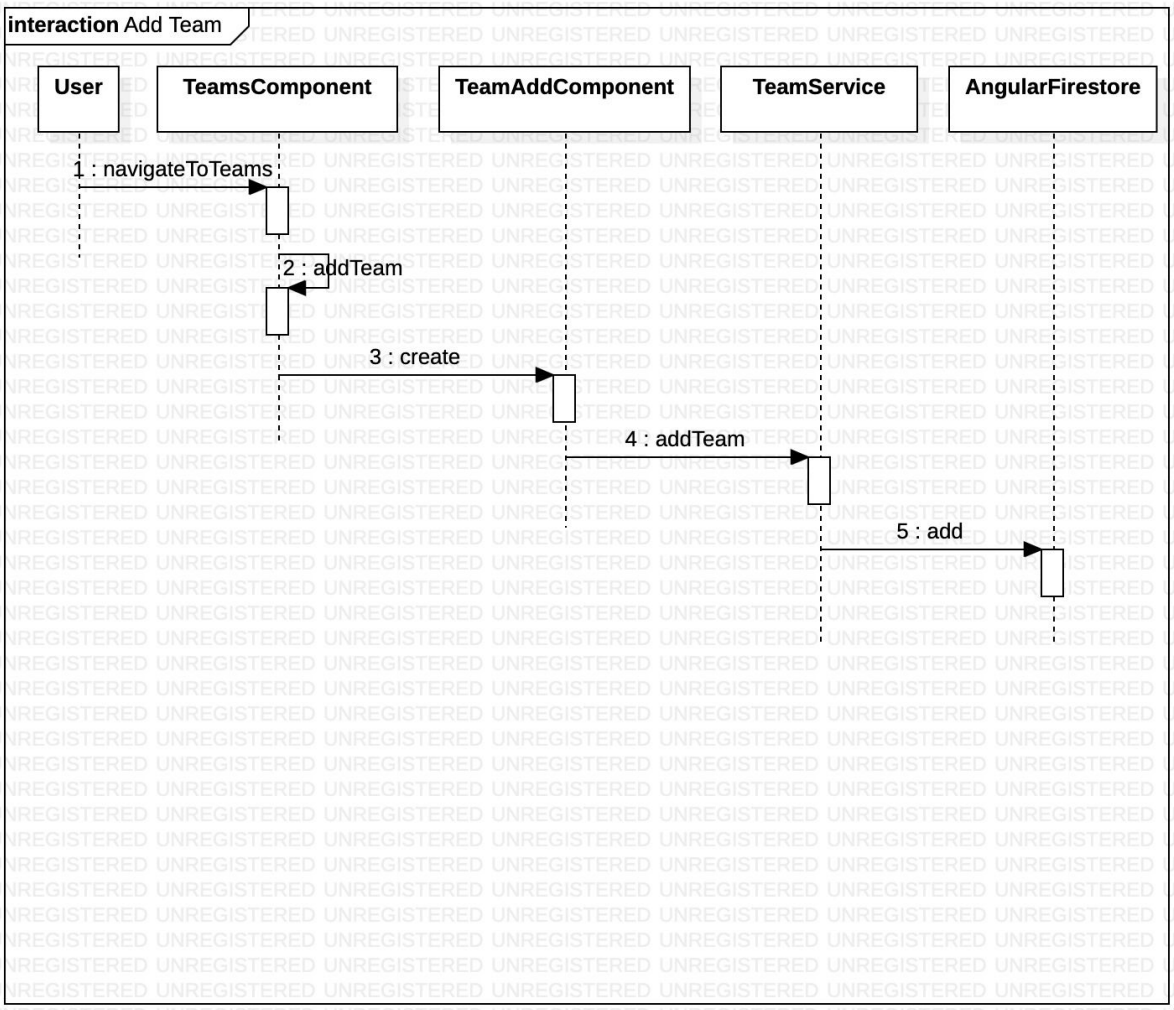
Edit Player


interaction Edit Player
User
PlayersComponent
PlayerEditComponent
PlayerService
AngularFirestore
1 : navigateToPlayers
2 : editPlayer
3 : create
4 : updatePlayer
5 : update

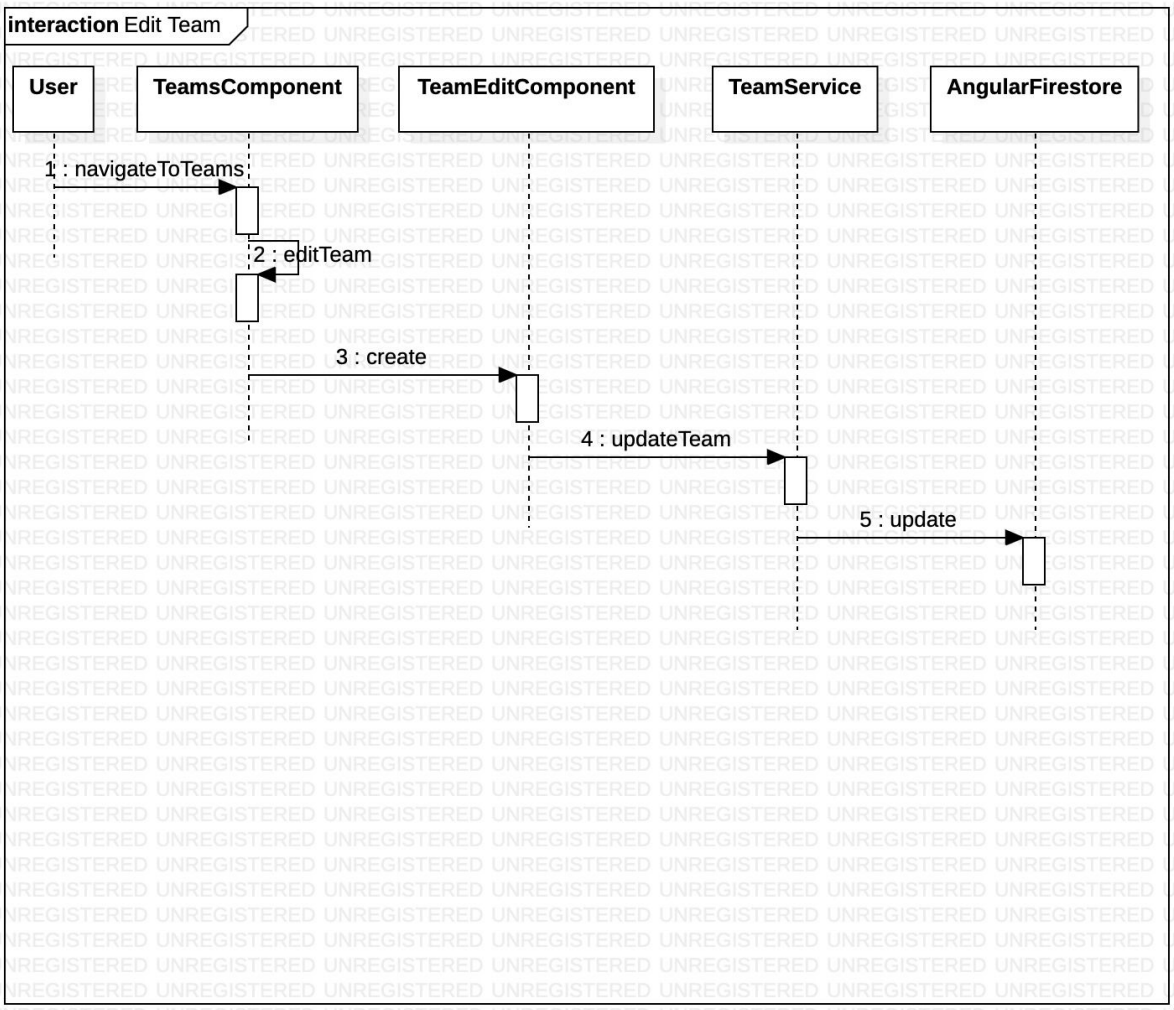
Delete Player



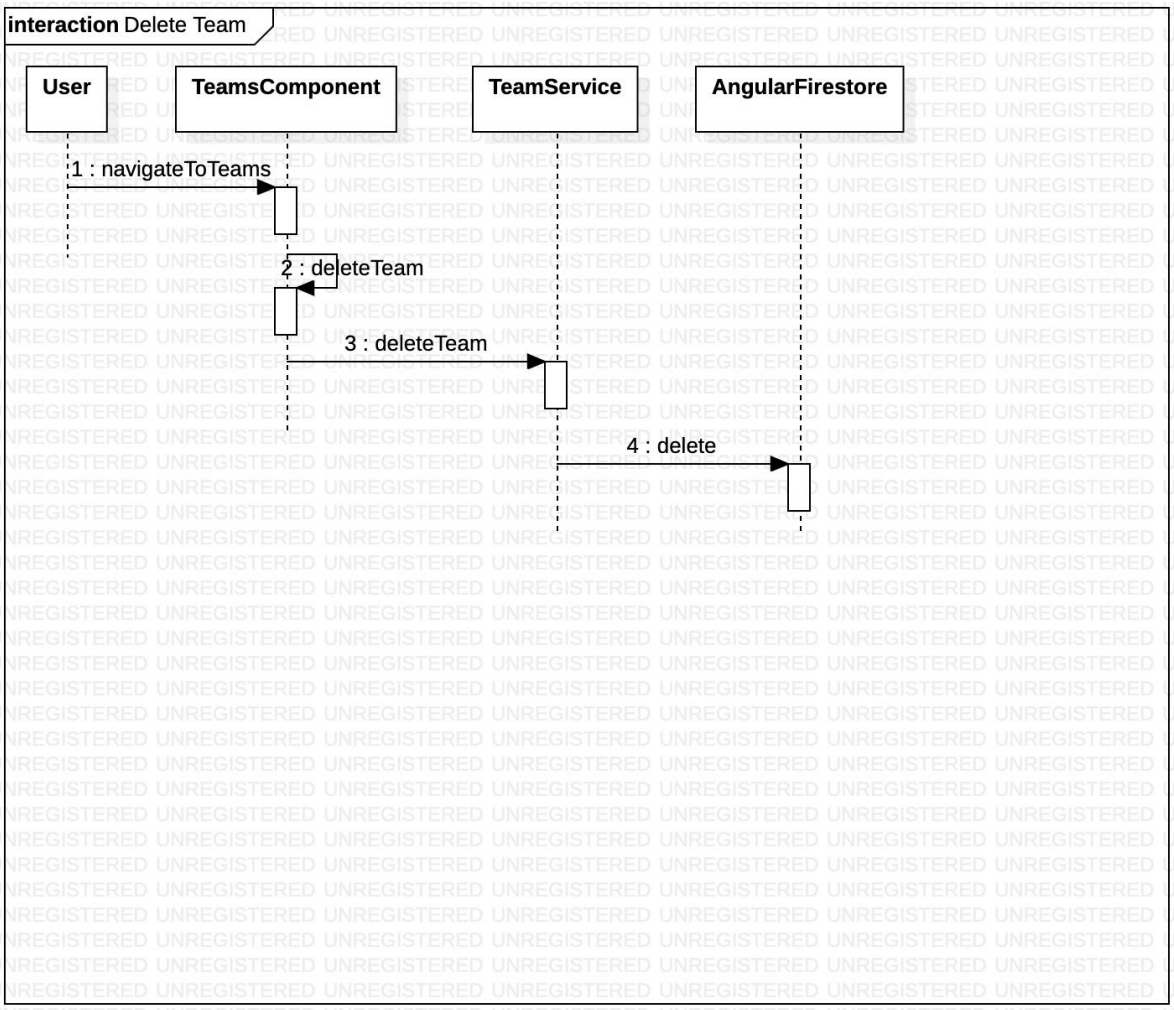
Add Team


interaction Add Team
User
TeamsComponent
TeamAddComponent
TeamService
AngularFirestore
1 : navigateToTeams
2 : addTeam
3 : create
4 : addTeam
5 : add

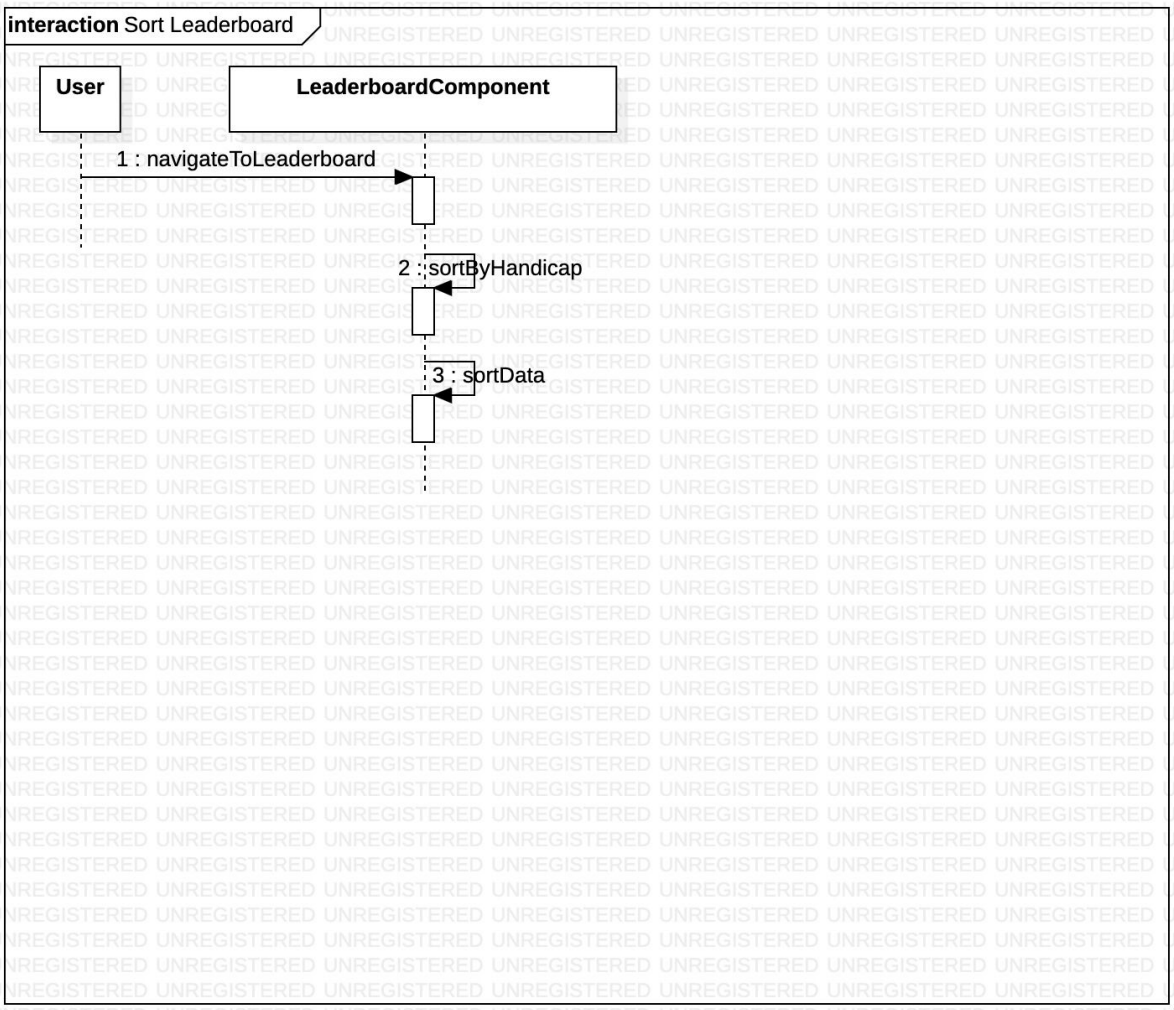
Edit Team


interaction Edit Team
User
TeamsComponent
TeamEditComponent
TeamService
AngularFirestore
1 : navigateToTeams
2 : editTeam
3 : create
4 : updateTeam
5 : update

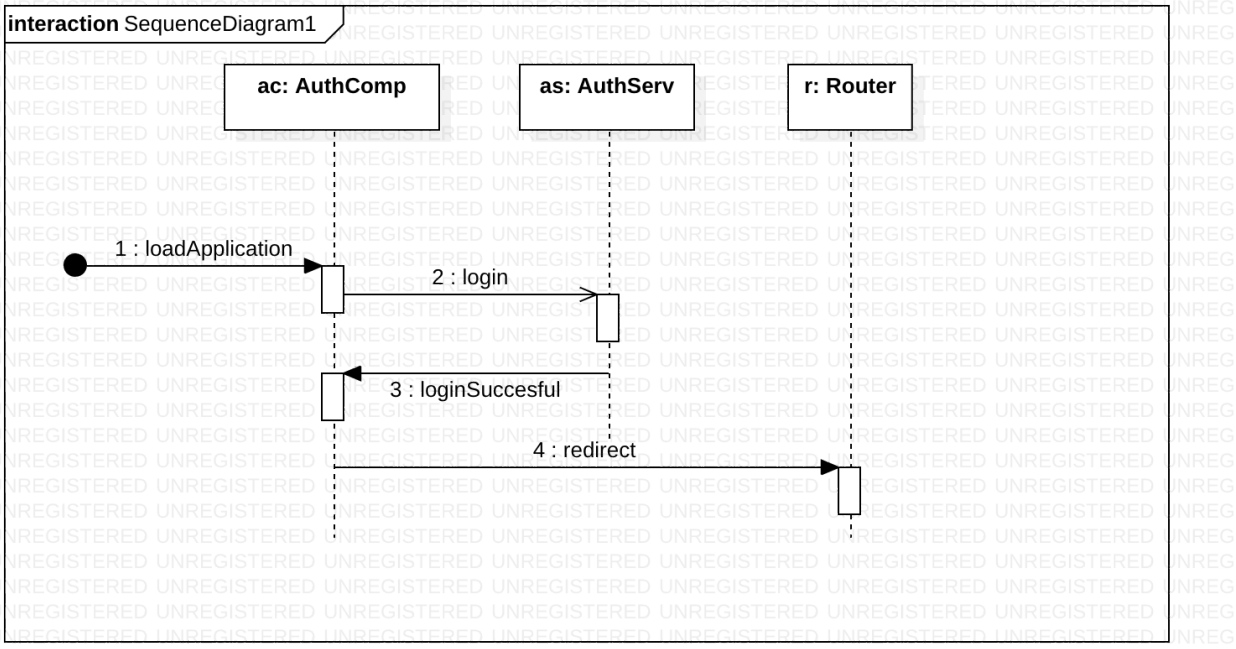
Delete Team


interaction Delete Team
User
TeamsComponent
TeamService
AngularFirestore
1 : navigateToTeams
2 : deleteTeam
3 : deleteTeam
4 : delete

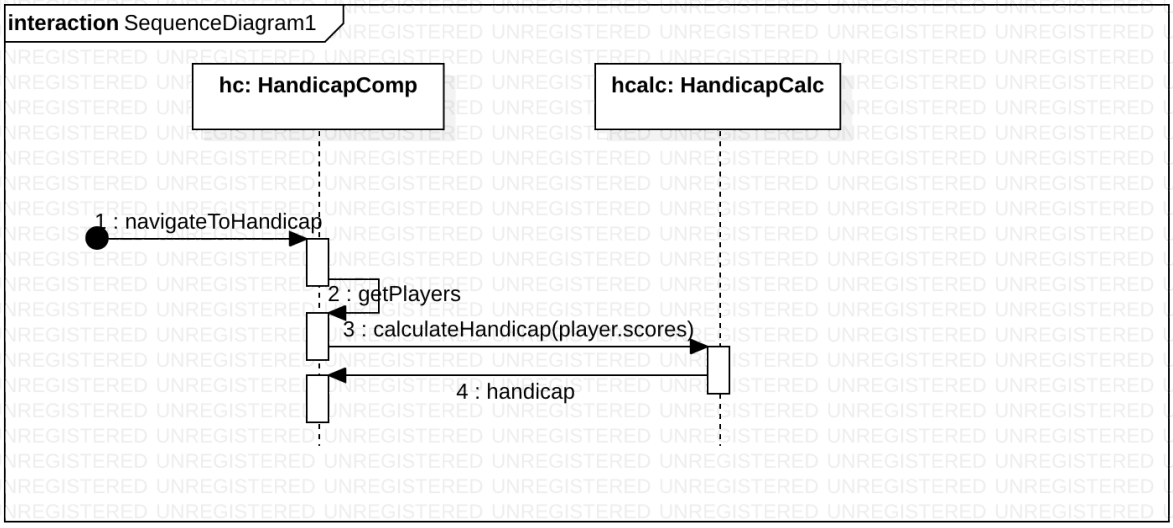
Sort Leaderboard



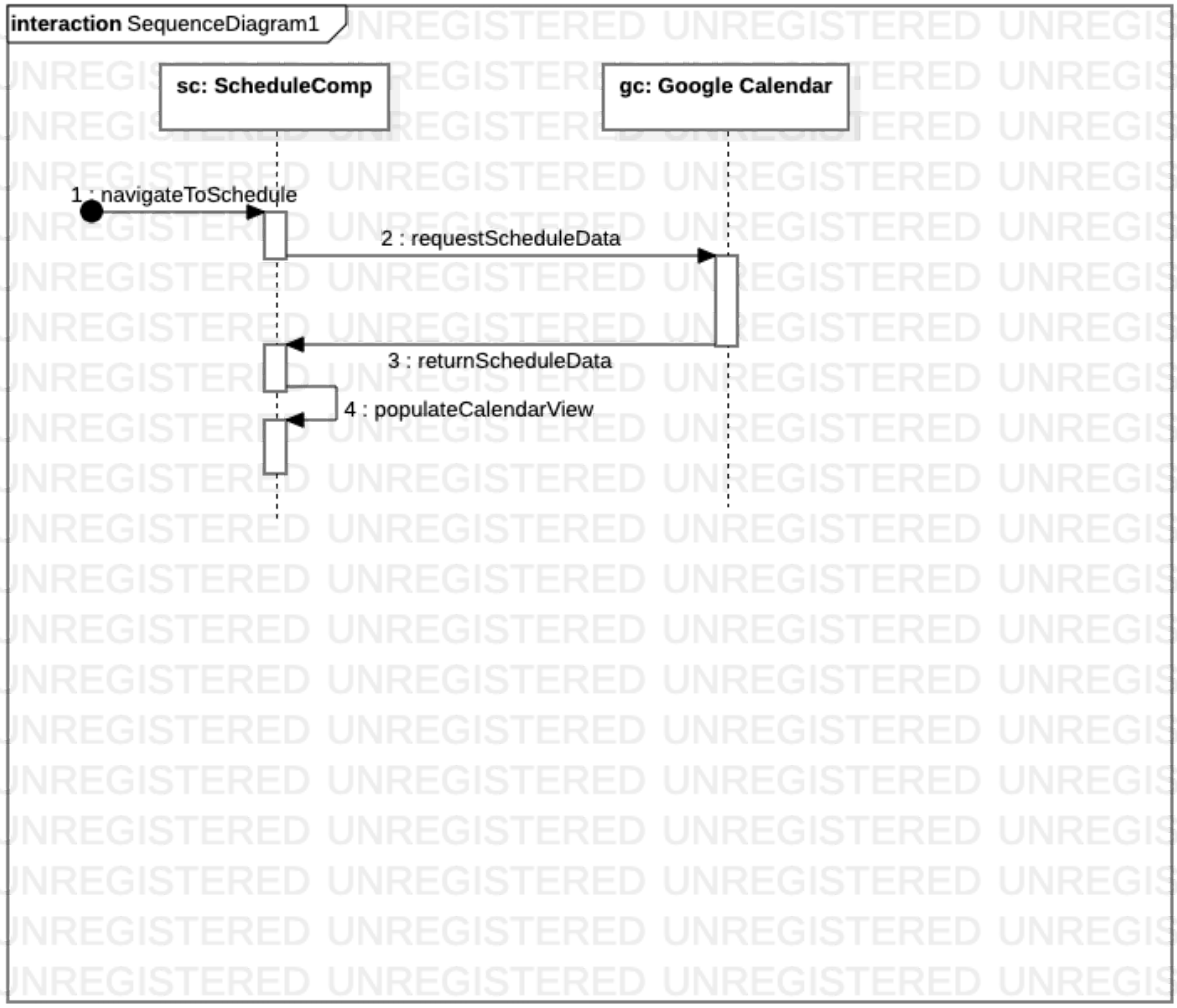
Authenticate



Calculate Handicap



Schedule



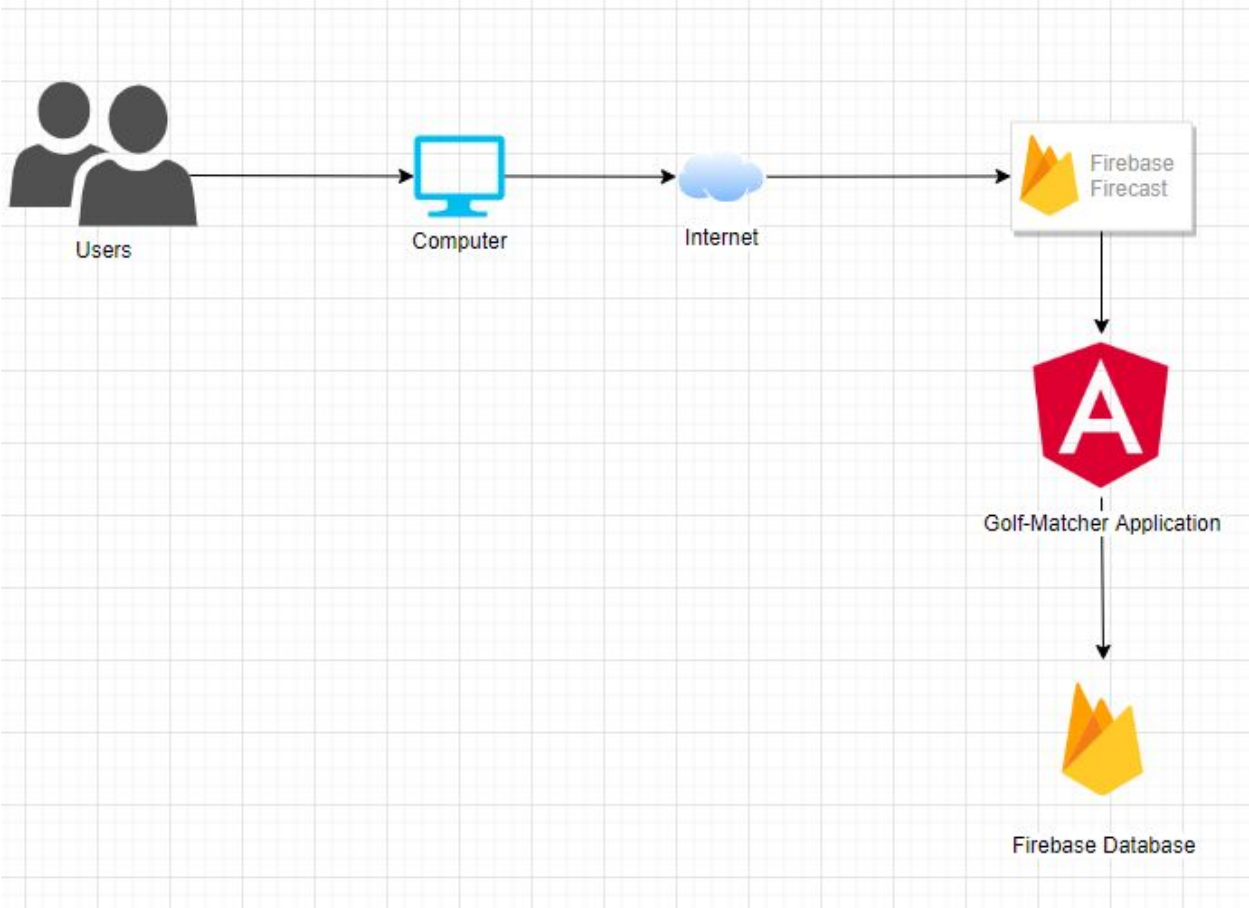
5. Process View

5.1 Processes

The Angular application, Golf-Matcher, bootstraps and runs using Javascript file. Angular loads the app-component.ts component which is the entry point of the application. The browser displays the components contained in the view of the app-component. Subsequent components are loaded based on the html statements.

There is a single browser process that runs the application. The Golf-Matcher application does not make use of other processes or threads to accomplish its tasks.

6. Deployment View



7. Size and Performance

Golf-Matcher will receive no more than 30 different users a day. We chose to host our website using Firebase Firecast because this allows for extremely high availability, and costs for such a small traffic website falls within their free tier. Firecast also allows for continuous integration for our application to be easily deployed. Firecast also allows collection of performance metrics to determine performance on requests, startup speed, and many others to make any necessary adjustments.

Firebase database also allows for our data to be highly accessible this way our data is always readily available.

8. Quality

The Golf-Matcher application was designed and built to meet the following quality attributes.

- Performance
- Interoperability
- Usability
- Reliability
- Security
- Maintainability
- Modifiability
- Testability
- Scalability
- Reusability
- Supportability

Performance
The application performance was measured via user response times. The user navigated to each of the views and noted the time to render each view. The application performance was limited by the time to read the Firestore database to retrieve application information. With a limited number of teams, players, scores, and handicaps the application was very performant. We increased the number of players to 50, which simulates a large golf league, and the application pages loaded in a timely manner. The user did not perceive any lag. We did similar testing with the number of teams and scores to stress the application.

Interoperability
The communication with external systems is encapsulated into each of the services: Player, Team, Authentication.

The Player service provides the interface to read/write Player data that each component uses. The Player service communicates with the Firebase database external system to add/edit/delete Player information from the Players table in Firebase.

The Team service provides the interface to read/write Team data that each component uses. The Team service communicates with the Firebase database external system to add/edit/delete Team information from the Teams table in Firebase.

The Authentication service provides the interface to the NgbAuthFirebaseUIModule which handles credential management. The Authentication service communicates with the Firebase authentication system to send/receive authentication credentials for the current user. The NgbAuthFirebaseUIModule provides the UI for the user to select credentials from the various credential services (Google, Facebook).

Usability

The Golf-Matcher application is designed to follow the mental model a golfer would use to participate in a golf league. The application is divided into components whose name matches their use (e.g. Handicaps, Scores). The user can easily navigate between components using the sidebar links. The components that do not display the sidebar links are data entry components that have a submit button. The submit button will navigate the user back to a component that displays the sidebar links to allow easy navigation.

The data entry components, such as Add Player, all have data validation which helps guide the user to enter the correct information. Any incorrect information the user enters will result in an alert that indicates the problem with the data.

Reliability

The Golf-Matcher application is designed to be reliable to the user. The application utilizes the Model-View-Controller paradigm which is a cornerstone of Angular development. As model data changes, each of the views dependent on the data are updated. This paradigm helps reduce dependency and increase application reliability.

Security

The application requires a user to authenticate before accessing any data. The Firebase NgbAuthFirebaseUIModule provides credential verification and management. Only the authenticated user can access any of the application data.

Maintainability

The application design has evolved over the semester. We have incorporated some stretch goals that were integrated easily at the end. The application componentization allowed the developers to easily add new functionality without modifying existing components. The system has evolved during the semester without affecting the core functionality.

Modifiability
The components that make up the application are well-defined and are responsible for a discrete portion of the data. The common data that needs to be updated upon change is the dashboard, sidebar, and routing. Any new component will need to update the dashboard, sidebar, and routing in order to allow the user to access the component. The new component can utilize existing services to access data such as Player, Team, Scores, or Handicaps without change to any other component. The new component will need a new service if it requires additional data not covered by the existing services.

Testability
The application has a robust set of unit and end-to-end tests to verify functionality. The unit tests run under Travis as part of the Continuous Integration workflow. Several unit tests cover each component to ensure internal component functionality. Travis provides a notification to all developers when a push is performed and tests run.

There are 50+ end-to-end tests that are run manually by a developer. The end-to-end tests run from the user perspective to verify visual components, links, and error-checks. The end-to-end tests are run using Protractor which is configured to generate a detailed report of test status with screenshots.

Scalability
The Golf-Matcher application required changes to support scaling up the number of players and teams. The visual components were wrapped in scrolled viewports to allow the user to interact with the components when the data size increases.

The data is stored in the Firebase database which supports exponentially more data than the application will create for a golf league.

Reusability
The application accesses data via the services such as Team and Player. These services are utilized by several components that require Team or Player information. The services encapsulate data access and provide a robust interface to interact with the data. Each service can be reused so that components don't have to duplicate code such as database connection.

Supportability
The data-entry components of the Golf-Matcher support data validation as the user interacts with the system. No invalid data is transmitted to the database because the

validation checks prevent the user from submitting. There are alerts for each widget that the user interacts with that will guide the user to enter valid information.

Conclusion

The Golf-Matcher application was designed and built to support a yearly golf league. It tracks players' and teams' schedules, scores, handicaps, and points. It is built using the Angular Framework and is accessible on any internet-capable device. It provides authentication for security using well-known credentials such as Google. The application will help golf league administrators and golf league players manage their information in a digital format.