CSE 232 Advanced Software Engineering

NAZAMLY

The Ultimate sports tournament manager

PROJECT DOCUMENTATION

Submitted to:

Prof. Dr. Gamal Abdel Shafy

Eng. Sally Shaker

Submitted by:

Youssef George Fouad	19P9824
Mostafa Nasrat Metwally	19P4619
Kerollos Wageeh Youssef	19P3468
Anthony Amgad Fayek	19P9880

0. Abstract

The name alongside the logo truly reflects the webapp main function, which will be discussed in the document, in addition to the youthful identity of the project written in a fresh font to represent the dynamic, competitive energy of the sports industry.

Figure 1: Nazamly Logo

It is a webapp that is set to be able to help in planning, organizing in addition to following up with any sport tournament of any organizational plan, sport, or number of participants.

Table of Contents

Table of Figures

1. Introduction

1.1. Purpose

This document is created so that there is no miss communication between the software development team and the intended party that requested the development of the software product, as it entails all the details of the software which are explained and visualized using various tools like UML Diagrams and charts. In addition to that, this document contains details of the plan going forward with development which includes all the steps that will be taken as well as the financial aspect and the different ways the user can practically use the product.

1.2. List of Definitions

- **API** (Application Programming Interface)

A specific method prescribed by an application program by which a programmer writing an application program can make requests of the operating system or another application.

- **Gantt Chart:**

A horizontal chart frequently used in project management that provides a graphical illustration of a schedule that helps to plan, coordinate, and track specific tasks in a project.

- **IDE** (Integrated Development Environment)

a programming environment that has been packaged as an application program, typically consisting of a code editor, a compiler, a debugger, and a GUI builder.

- **KLOC** (Thousands of Lines of Code)

A traditional measure of how large a computer program is or how long or how many people it will take to write it, sometimes used as a rough measure of programmer productivity.

- **SDK** (Software Development Kit)

A set of programs used by a computer programmer to write application programs.

- **UI** (User Interface)

Everything designed into an information device with which a human being may interact -- including display screen, keyboard, mouse, light pen, the appearance of a desktop, illuminated characters, help messages, and how an application program or a Web site invites interaction and responds to it.

1.3. Scope

Nazamly is an online webapp that is set to be able to help in planning, organizing in addition to following up with any sport tournament of any organizational plan, sport, or number of participants. The software is a tournament organization tool that can be used by various parties in communicating tournament details among many people that include match dates, contestants, scores, and venues. It also has the option of letting users buy or sell tickets if a venue is available for use.

Tournify is a website that has similar features and goals like that of Nazamly. They have over 30,000 users ranging from amateurs to professionals. They have also partnered with famous football clubs such as Ajax and Chelsea. This goes to show the potential of the Nazamly app in the Egyptian market. In addition to that, what separates Nazamly from Tournify is the follow up feature, as Tournify only allows the participants of a tournament to see its progress unlike Nazamly which will show any interested user any tournament's progress and any match results.

1.4. List of References

1) Angular 13

Angular is a TypeScript-based free and open-source web application framework led by the Angular Team at Google and by a community of individuals and corporations.

2) Ng Bootstrap Libraries

Angular widgets built from the ground up using only Bootstrap 4 CSS with APIs designed for the Angular ecosystem.

3) GitHub

A provider of software development, distributed version control, and source code management functionality using Git.

Project source code available at: Nazamly GitHub Repository

4) **MS Visio**

Microsoft Visio is a diagramming and vector graphics application and is part of the Microsoft Office family.

5) Lucidchart

A web-based proprietary platform that allows users to collaborate on drawing, revising and sharing charts and diagrams.

6) Visual Studio Code

An IDE made by Microsoft for Windows, Linux and macOS, and was used to write this application code.

1.5. Overview

This document is a part of the full Nazamly project. It aims to discuss the detailed, full process and help you understand the architecture, interface, capability, and ability of the Nazamly software system.

It helps answer questions of the software architect, engineer, and developer concerning how the journey of building the system took place from day one till the deployment and delivery of the web application. In addition, it gives business-related people and investors some insights about the plan, cost estimation, and potential profit and growth. Moreover, it provides the end-user with the know-how of using the system to have the best possible experience.

2. General Description

2.1. Product Perspective

The product acts as the bridge that merges the gap between casual sports players, tournament organizers, and fans away from the professional athletic world. It offers a solution platform where an organizer can schedule a community tournament and allow the fans to follow their favorite teams playing in their community tournaments and buy tickets too to attend these matches, paying in cash or using cards.

2.2. General Capabilities

The system can generate matches schedule, calculate teams points, organize tickets selling, and allow fans to sign up for a free-to-create account to customize their experience as true fans of their community teams.

2.3. General Constraints

The web application should have a user-friendly graphical interface that allows him to interact easily with the system and enjoy the best possible customized experience in simple, appropriate English language.

2.4. User Characteristics

In this section, the different user roles are described where, Nazamly webapp is expected to have three type of users, anonymous users that have not signed up for an account in addition to the logged in users that have already sign up to the platform, besides the admin users that have chosen to have the organizers' privileges when creating their Nazamly app.

Firstly, anonymous users can sign up to a make a new account or view any details about any tournament ranging from scores to player details without being able to buy tickets or follow certain team.

Secondly, registered users, who have an account but are not organizers, have added benefits than an anonymous user that include, selecting their favorite teams for a more personalized experience, and the ability to purchase tickets to any available match.

Last but not least, organizing users will be able to use all available features of the application that include creating new tournaments and managing them by entering matches' scores and players info.

2.5. Environment Description

The webapp's operational environment is intended to be like any other web application software system that requires a web browser and a stable internet connection in order to fully operate and function properly.

2.6. Assumptions and Dependencies

The system assumes the regular user has some basic knowledge about how sports tournaments work in terms of match making, scoring, and pointing system in addition to general sports related English terms. It is also assumed that the user has access to internet on any device with an internet browser already installed.

3. Specific Requirements

3.1. Capability Requirements

1- **Sign up**

The website visitor shall register for one of two account types, normal user or a tournament organizer user to enjoy special features.

2- **Create a new tournament**

A tournament organizer shall be able to create a new tournament and the system should generate a matches' timetable as per the input conditions.

3- **Add/ Update match final score**

A tournament organizer shall be able to update a match final score.

4- **Select favorite teams to follow**

Registered users shall select their favorite teams to follow their latest and upcoming matches on their home screen.

5- **View matches**

All users shall view matches of any team played in any tournament in any time.

6- **Filter viewed matches**

All users shall be able to filter viewed matches by date, team or tournament.

7- **Buy tickets**

Registered users shall be able to buy tickets for any upcoming match.

8- **Generate sales report**

Tournament organizers shall be able to generate sales report.

9- **View players' statistics**

Any user shall view players' goals statistics and filter them according to tournament.

10- **View tournaments league standings**

All users shall view a tournament standing of any tournament in any time.

11- **Sort tournament's view**

All users shall be able to sort viewed tournament standings, ascendingly or descending, by several filters, including position, goals for count, goals against count, goals difference, and matches played count.

3.2. Constraint Requirements

1- The system needs an internet connection to be functional.

2- The program shall have a user-friendly interface so that its easier for any user to be able use.

3- The program shall have a 50-millisecond response time.

4- The program shall not contain any inappropriate content; sexual, violent, bullying, or racist.

5- The passwords should be securely saved and will not be accessed by anyone.

6- The program shall be safe from any attacks or attempts of hacking.

4. Use-Case Diagram

A use case diagram establishes the capability of the system from the user's point of view including several components:

1- **Actors**: anonymous user, registered normal user, registered organizer user.

2- **Use cases**: represented by blue ovals in figure 1.

3- **System boundary**: represented by a rectangular frame.

4- **Use cases' relationships**: extend, include, inheritance.

5- **Actors' relationships.**

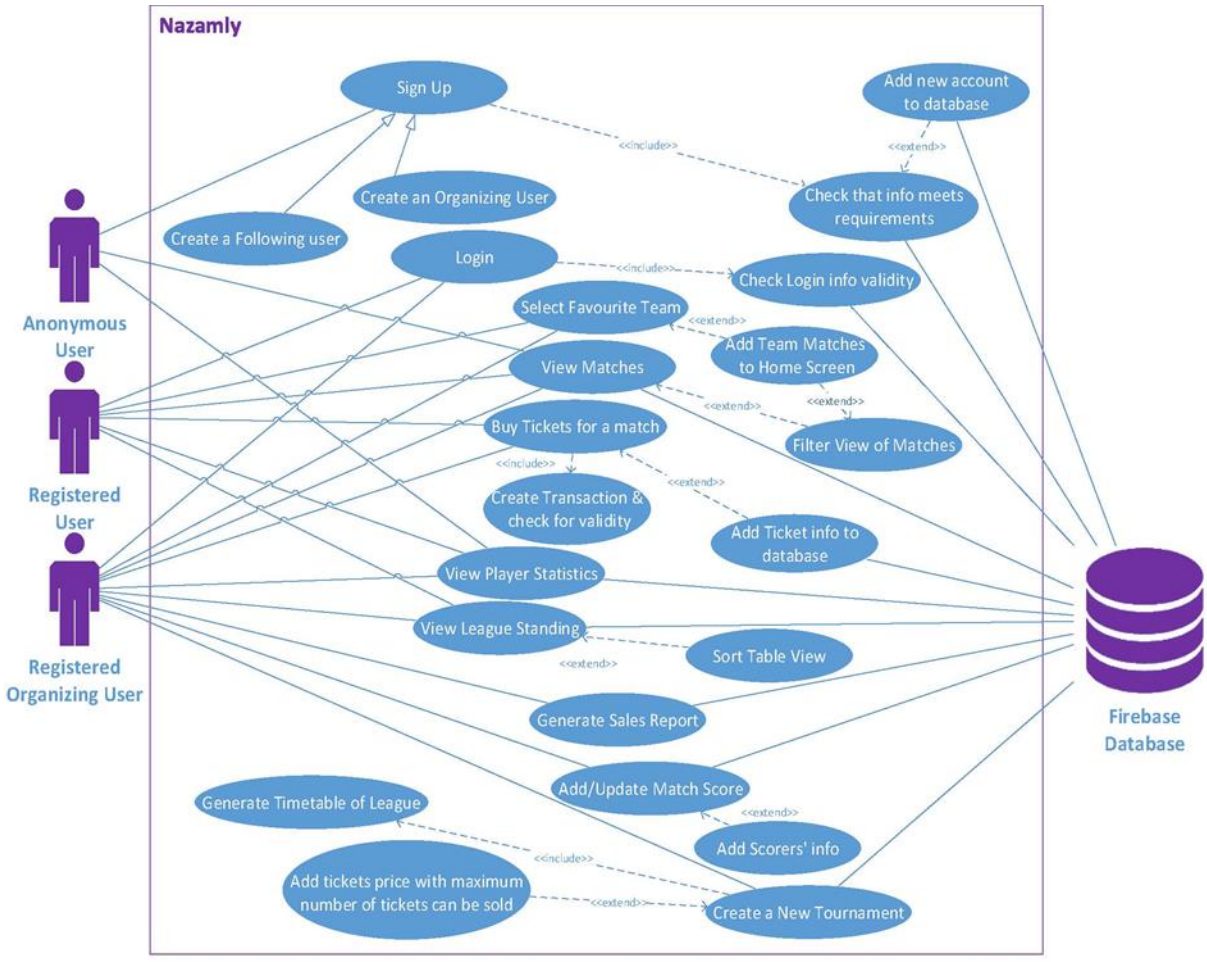


Figure 2: Use-Case Diagram

1- Sign Up

Use Case Name	Sign up	
Related Requirements	Requirement 1.	
Goal In Context	A new user registers for a new user account.	
Preconditions	The user registers with a new email.	
Successful End Condition	A new user account is created.	
Failed End Condition	The request is rejected.	
Primary Actors	Anonymous user.	
Secondary Actors	Firebase Database	
Trigger	The user fills in the new account form.	
Included Cases	Check that provided information meets requirements.	
Main Flow	Step	Action
	1	The user requests to register for a new account.
	2	The user fills in the required personal information.
	3 **include:: check info meets req.**	The user's details are verified using the credentials' authentication database system.
	4	A new account is created.
	5	A success message pops up.
Extensions	Step	Branching Action
	3.1	The credentials' authentication database system refuses the user's personal information.
	3.2	The request is rejected, and an error message pops.

2- Check that provided information meets requirements.

Use Case Name	Check that provided information meets requirements.	
Related Requirements	Requirement 1.	
Goal In Context	A new account details need to be checked and verified.	
Preconditions	The user submits a new user request.	
Successful End Condition	A new user account is verified.	
Failed End Condition	The new account request is rejected.	
Primary Actors	Firebase Database.	
Secondary Actors	None.	
Trigger	The new user information is provided to the system.	
Included Cases	Add new account to database.	
Main Flow	Step	Action
	1	The new user information is provided to the system.
	2	The credentials' database authentication system verifies the details are not of another existing account and that all requirements meet the specifications.
	3	The user's details are verified.
	4 **include:: Add new account to the database.**	A new account is added to the users' database.
	5	A success message pops up.
Extensions	Step	Branching Action
	3.1	The details are rejected for duplication or inaccuracy.
	4.1	No new account added due to rejection of user's information.

3- Add new account to database

Use Case Name	Add new account to database.	
Related Requirements	Requirement 1.	
Goal In Context	Add new account information to database records.	
Preconditions	The authentication system verifies the new user information.	
Successful End Condition	A new user account is added.	
Failed End Condition	The new account request is not added.	
Primary Actors	Firebase Database.	
Secondary Actors	None.	
Trigger	The new user information is verified and accepted by the database.	
Base Case(s)	Check that provided information meets requirements.	
Main Flow	Step	Action
	1	The new user information is accepted by the credentials' database authentication system.
	2	All user information details are added to the database records.

4- Login

Use Case Name	Log in	
Related Requirements	Requirement 2.	
Goal In Context	A registered user logs in to his account.	
Preconditions	The user has an account.	
Successful End Condition	The user logs in his account.	
Failed End Condition	The user does not log in his account.	
Primary Actors	Registered user – Registered organizer account.	
Secondary Actors	Firebase Database	
Trigger	The user fills in his login credentials.	
Included Cases	Check login information validity	
Main Flow	Step	Action
	1	The user fills in his username and password.
	2 **include:: check login validity**	The provided credentials are verified using the credentials' authentication database system.
	3	The user logs in his account.
	4	The user returns to his customized home page.
Extensions	Step	Branching Action
	2.1	The credentials' authentication database system refuses the provided credentials.
	3.2	Login fails, and a specified error message pops.

5- Check log in validity

Use Case Name	Check login validity	
Related Requirements	Requirement 2.	
Goal In Context	A registered user logs in to his account.	
Preconditions	The user attempts to log in.	
Successful End Condition	Login credentials verified.	
Failed End Condition	Login credentials verification failed.	
Primary Actors	Firebase Database	
Secondary Actors	None	
Trigger	The user fills in his login credentials.	
Main Flow	Step	Action
	1	The database system is provided user credentials.
	2	Check whether email and password match.
	3	Return success message
Extensions	Step	Branching Action
	3.1	Email and password do not match.
	3.2	Return error message.

6- Select favorite team

Use Case Name	Select favorite team.	
Related Requirements	Requirement 5.	
Goal In Context	Add a team to user's favorite teams list to customize his experience.	
Preconditions	The user is logged in to his account.	
Successful End Condition	User's favorite teams list updated.	
Failed End Condition	User's favorite teams list not updated.	
Primary Actors	Registered user – Registered organizer account.	
Secondary Actors	Firebase Database.	
Trigger	The user selects an existing team to add to his list.	
Included Cases	Add team matches to home screen.	
Main Flow	Step	Action
	1	The user selects team name from a team list including the teams that are not already in his custom favorite teams list.
	2 **include:: Add team matches to home screen**	The team is added to his list.
	3	Home page matches are updated
Extensions	Step	Branching Action
	2.1	Retrieve added team matches and add to home screen.

7- Add team matches to home screen

Use Case Name	Add team matches to home screen.	
Related Requirements	Requirement 6.	
Goal In Context	Add teams matches to the application “Home” or “All matches” screen.	
Preconditions	Teams’ list received is not empty.	
Successful End Condition	Matches are retrieved from database.	
Failed End Condition	No matches found in database.	
Primary Actors	Firebase database.	
Secondary Actors	Any user	
Base Case(s)	Add team matches to home screen – Filter view of matches.	
Main Flow	Step	Action
	1	Teams’ list is provided to fetch for their matches.
	2	Matches’ list is retrieved from the database according to the teams’ list.
	3	Matches’ list is sent to be viewed on screen.
Extensions	Step	Branching Action
	2.1	No matches are found for any of the list’s teams.

8- Filter view of matches

Use Case Name	Filter view of Matches.	
Related Requirements	Requirements 6, 7.	
Goal In Context	Choose specific filters for matches to view.	
Preconditions	None.	
Successful End Condition	Found matches conforming user's filters.	
Failed End Condition	No matches found conforming user's filters.	
Primary Actors	Any user.	
Secondary Actors	Firebase Database	
Trigger	The user edits the view matches' filter.	
Included Cases	View matches	
Main Flow	Step	Action
	1	The user edits any of the date, team, or tournament filters.
	2	Database system fetches the match conforming the received filters.
	3	The matches' list is retrieved from the database to be viewed on the user's screen.
	4	Viewed matches are updated.
Extensions	Step	Branching Action
	2.1	No matches found according to the set filters.

9- View matches

Use Case Name	View matches.	
Related Requirements	Requirement 6.	
Goal In Context	View match cards including date, time, teams name and score.	
Preconditions	None.	
Successful End Condition	Matches viewed in the form of match cards list.	
Failed End Condition	No matches are viewed.	
Primary Actors	Any user.	
Secondary Actors	Firebase Database	
Included Cases	Filter view matches	
Main Flow	Step	Action
	1	Matches' list received from any other module.
	2	Matches data are formatted to a match card format including teams name, score, date, and time.
	3	Match cards are listed on user's screen.

10- Buy tickets for a match

Use Case Name	Buy tickets for a match.	
Related Requirements	Requirement 8.	
Goal In Context	Sell match tickets to a registered user.	
Preconditions	User logged in his account.	
Successful End Condition	User gets his ticket(s).	
Failed End Condition	Ticket not sold.	
Primary Actors	External bank servers, registered user.	
Secondary Actors	None.	
Included cases	Create transaction & check validity	
Main Flow	Step	Action
	1	User selects match and number of tickets
	2	Database check if number of tickets required is more than or equal available tickets for this match.
	3	Total price is viewed on the screen.
	4 **include:: Create transaction & check validity**	User chooses cash or credit card payment.
	5	Payment validated and success payment message received.
	6	User gets a PDF file includes all tickets details.
Extensions	Step	Branching Action
	2.1	Tickets required are more than available.
	2.2	Error message that the quantity chosen is not available.
	2.3	User reattempts a smaller number.
	4.1	Transaction rejected.
	4.2	User reattempts payment using another payment method.

11- Create transaction and check for validity

Use Case Name	Create transaction and check for validity.	
Related Requirements	Requirements 8.	
Goal In Context	Complete the payment process to buy tickets.	
Preconditions	User chooses to pay online by credit/debit card.	
Successful End Condition	Transaction completed.	
Failed End Condition	Transaction refused.	
Primary Actors	External bank servers, registered user.	
Secondary Actors	None.	
Trigger	The user attempts tickets purchase and provides credit/debit card info.	
Base Case(s)	Buy tickets for a match.	
Main Flow	Step	Action
	1	Credit/debit card information received.
	2	Information sent to bank servers to complete transaction.
	3	Success message received.
	4	Success message sent to the user.
Extensions	Step	Branching Action
	3.1	Transaction refused message received.

12- View player statistics

Use Case Name	View players statistics.	
Related Requirements	Requirements 10.	
Goal In Context	View goal scorers' statistics to the users.	
Preconditions	None.	
Successful End Condition	Players' details displayed.	
Failed End Condition	No data displayed.	
Primary Actors	Any user.	
Secondary Actors	Firebase database.	
Trigger	None.	
Included cases	None.	
Main Flow	Step	Action
	1	Fetch players' data from database.
	2	Players' data are formatted to a player card format including his name, number of goals, team, and tournament names.
	3	Match cards are listed on user's screen.
Extensions	Step	Branching Action
	1.1	No players' data found.

13- View league standings

Use Case Name	View league standings.	
Related Requirements	Requirements 11.	
Goal In Context	View league standings to the users.	
Preconditions	None.	
Successful End Condition	League standings displayed.	
Failed End Condition	League standings not displayed.	
Primary Actors	Any user.	
Secondary Actors	Firebase database.	
Trigger	User chooses league name to view its standings.	
Included cases	None.	
Main Flow	Step	Action
	1	Fetch teams' list of the selected tournament from database.
	2	Sort teams' list according to points account, if two of them have equal number of points, refer to the goals difference.
	3	Display all teams' details in a table view.
Extensions	Step	Branching Action
	1.1	No teams found in database.
	1.2	Display error message.

14- Generate Sales Report

Use Case Name	Generate sales report.	
Related Requirements	Requirement 9.	
Goal In Context	Generate sales report for a tournament organizer.	
Preconditions	The user organizes a tournament.	
Successful End Condition	A sales report is generated.	
Failed End Condition	A sales report is not generated.	
Primary Actors	An organizing user.	
Secondary Actors	Firebase database.	
Included cases	None.	
Main Flow	Step	Action
	1	Tournament organizer selects a tournament.
	2	Retrieve matches and tickets lists of the selected tournament from the database.
	3	Calculate sales for each match in addition to total tournament sales
	4	Create a sales formatted PDF files.
	5	Tabulate calculated sales info.
	6	Open the PDF file for the user to view or save.

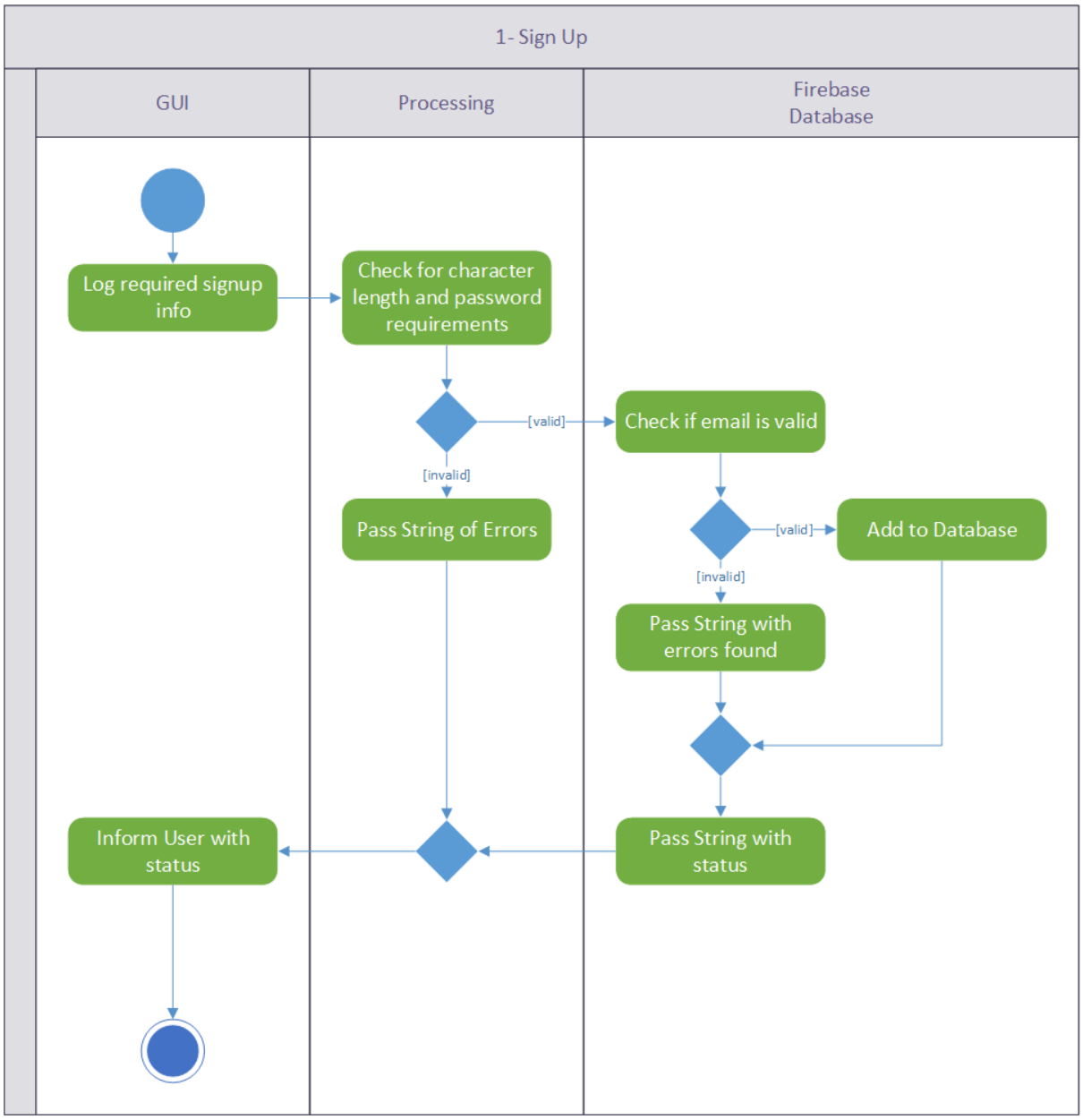
15- Add/update match score

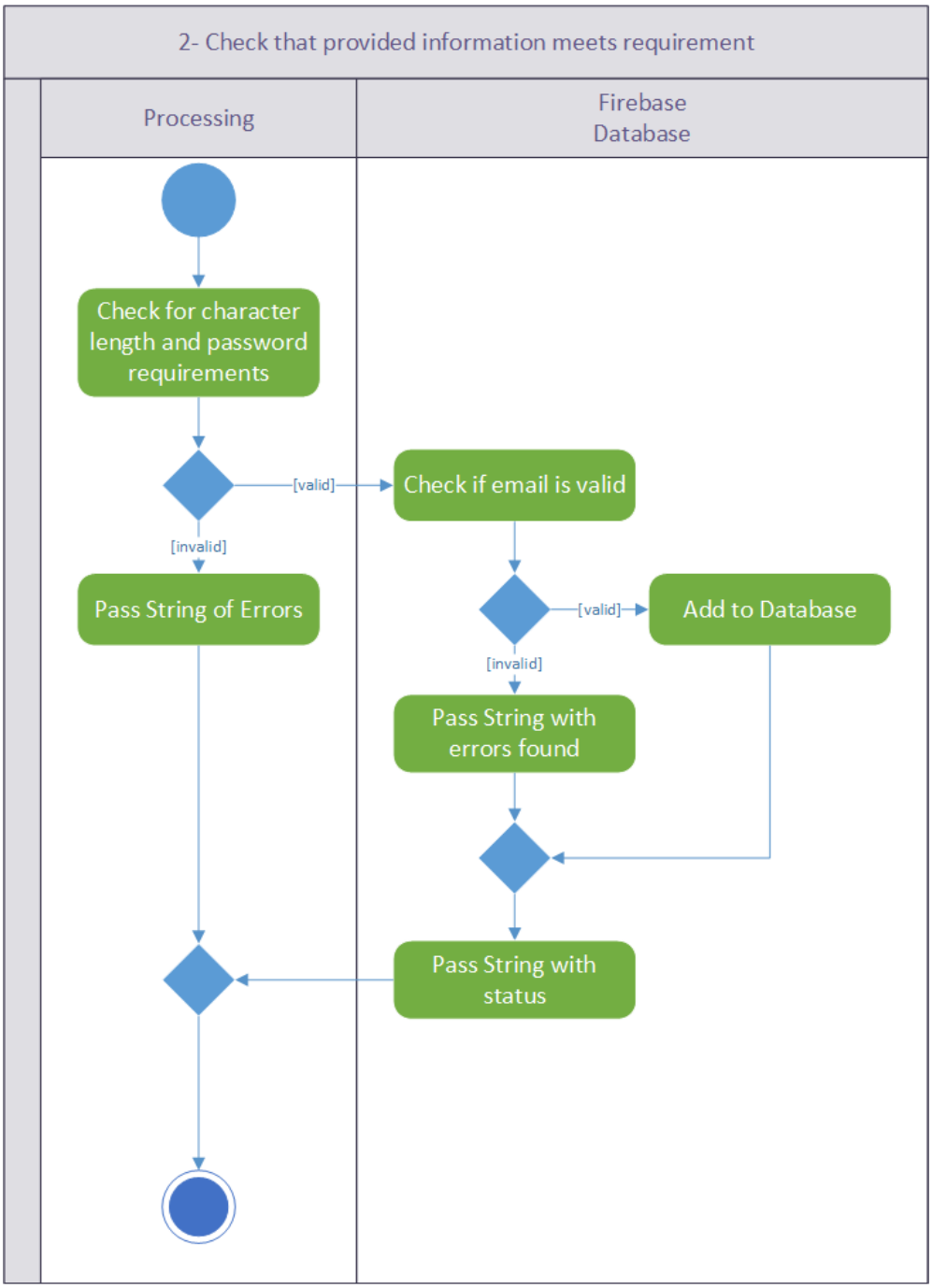
Use Case Name	Add/update match score.	
Related Requirements	Requirement 4.	
Goal In Context	Update match details after it has been played.	
Preconditions	The user organizes a tournament.	
Successful End Condition	Match final score updated.	
Failed End Condition	Match final score not updated.	
Primary Actors	An organizing user.	
Secondary Actors	Firebase database.	
Included cases	Add scorers' info.	
Main Flow	Step	Action
	1	Tournament organizer selects a tournament.
	2	Retrieve matches and players names lists of the selected tournament from the database.
	3	Organizer chooses the match he wants to update.
	4 **include:: Add scorers' info**	Organizer enters the match final score.
	5	Match final score updated.
Extensions	Step	Branching Action
	1.1	Selected Tournament does not have any match left not played.
	2.1	Selected match has a final score assigned to it.
	4.1	User selects of the already existing players or add a new one to assign a goal to him/her.

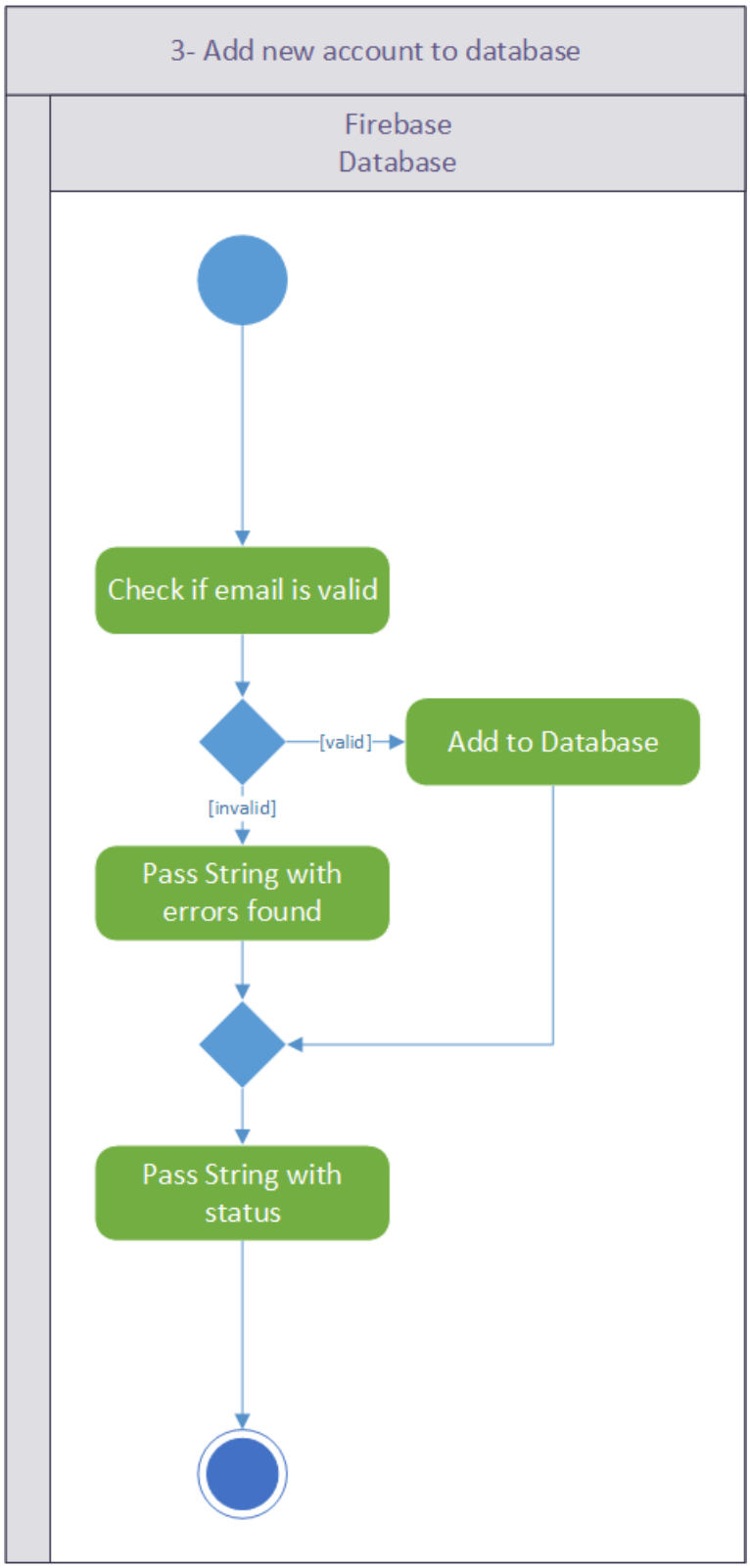
16- Create new tournament

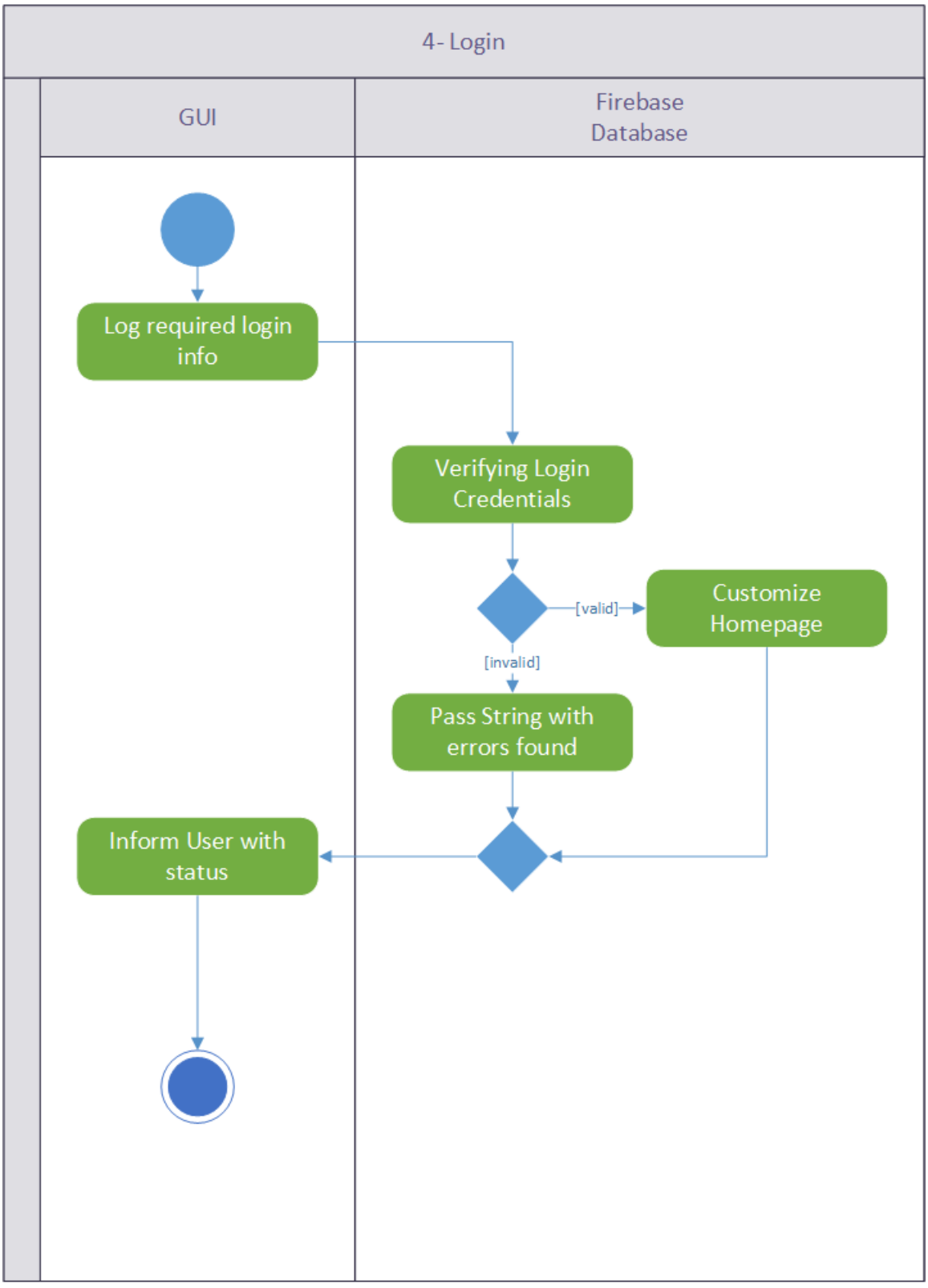
Use Case Name	Create new tournament.	
Related Requirements	Requirement 3.	
Goal In Context	Organizer add a new tournament to the system.	
Preconditions	The user account is an organizer account.	
Successful End Condition	A new tournament added to the system.	
Failed End Condition	No tournaments added to the system.	
Primary Actors	An organizing user.	
Secondary Actors	Firebase database.	
Included cases	Generate timetable of league – Add tickets info	
Main Flow	Step	Action
	1 **include:: Add tickets info**	Tournament organizer enters tournament details including, tournament name, teams count, teams' names, time frame, and number of legs.
	2 **include:: Generate timetable**	A timetable is generated specifying matches weeks, opponents, date, and time.
	3	Tournament and matches details added to the database.
	4	Success label pops up indicating the creation of a new tournament.
Extensions	Step	Branching Action
	1.1	Error message due to similarity between new tournament and other tournaments names.
	2.1	Error message due to time interval is negative where the start date comes after the finish date

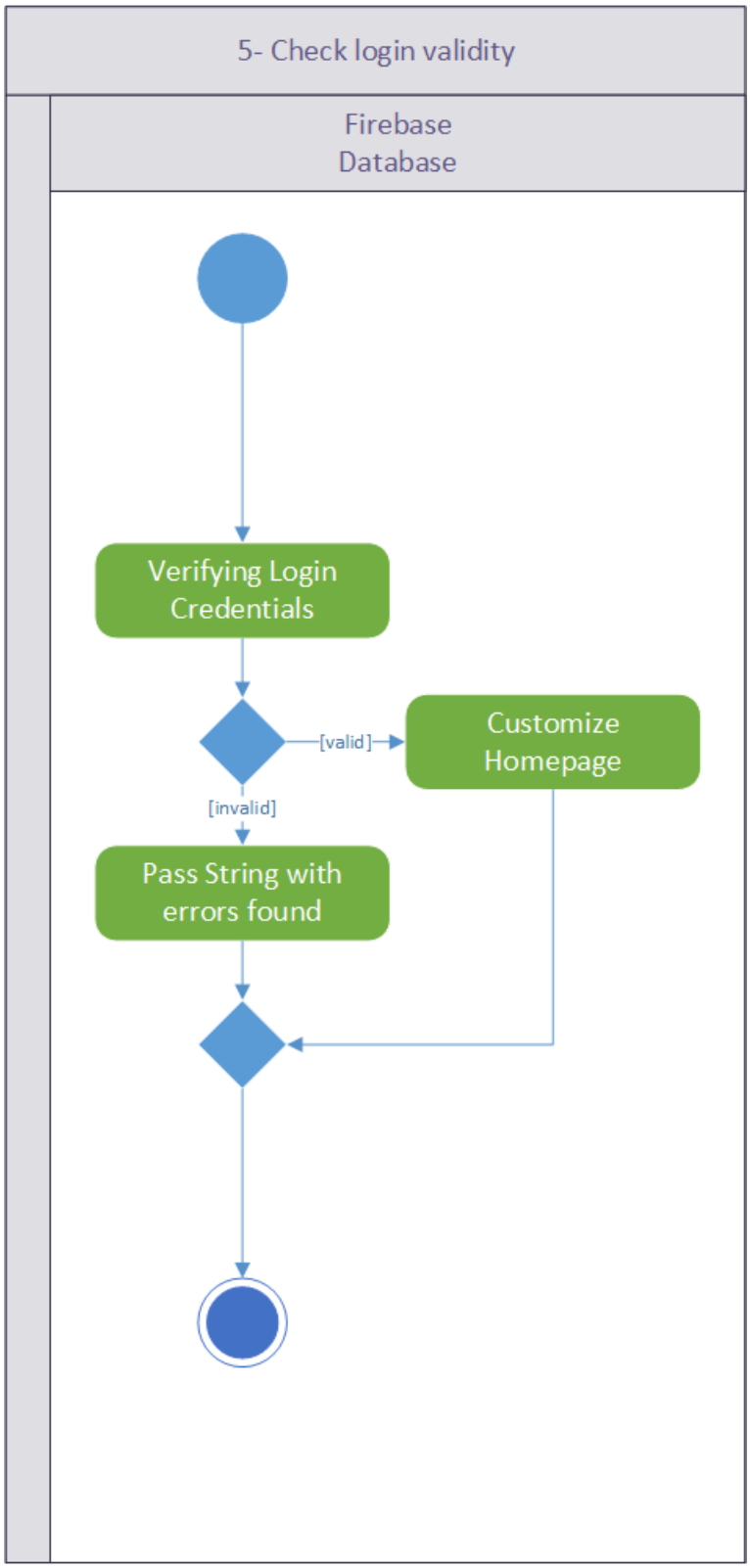
5. Swimlane Diagrams

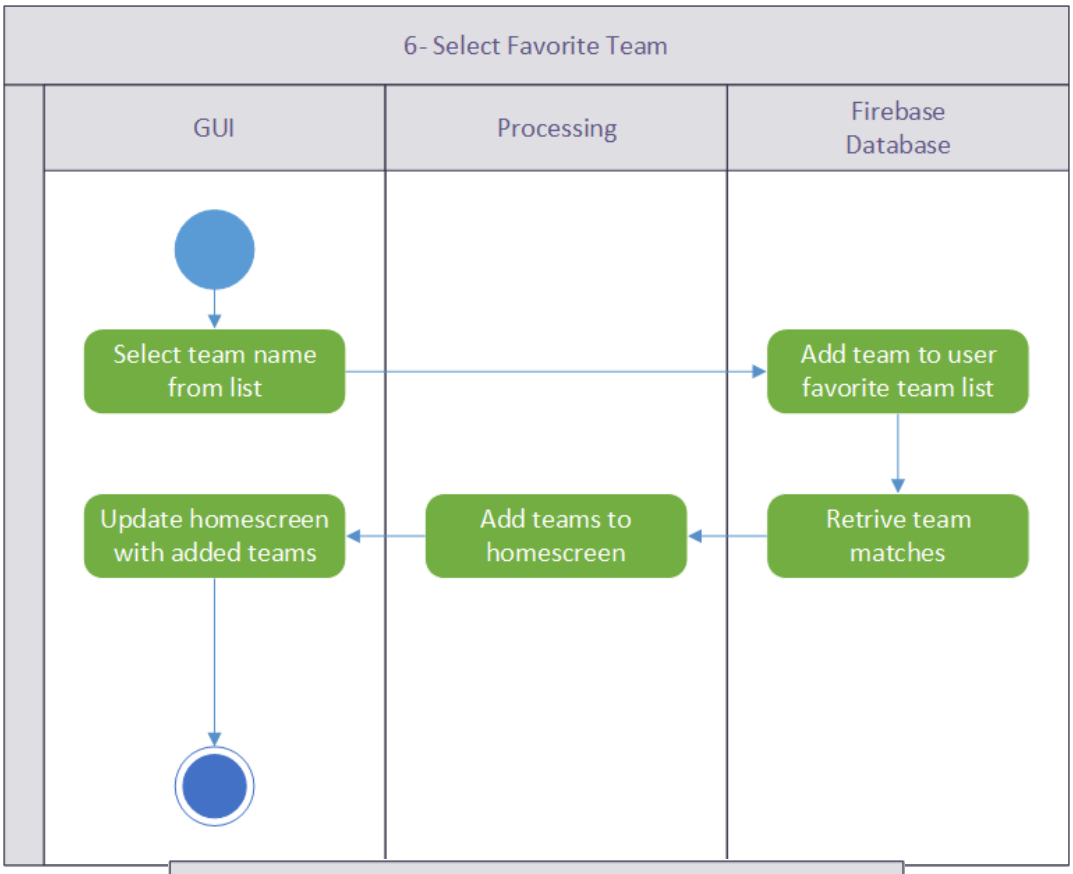


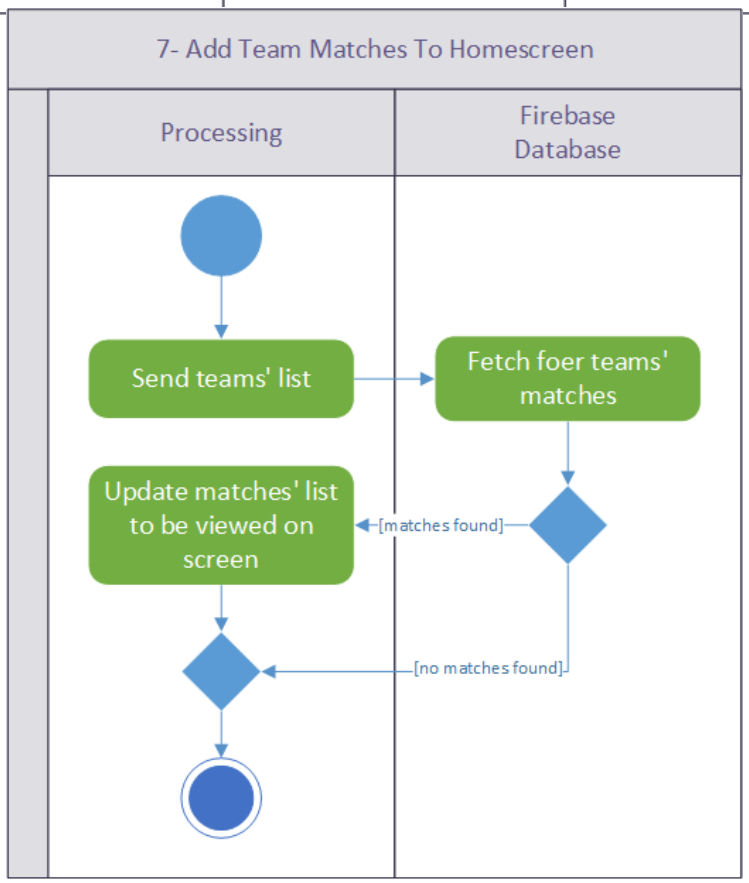

2- Check that provided information meets requirement
Processing
Firebase Database
Check for character length and password requirements
[valid]
Check if email is valid
[invalid]
Pass String of Errors
[valid]
Add to Database
[invalid]
Pass String with errors found
Pass String with status

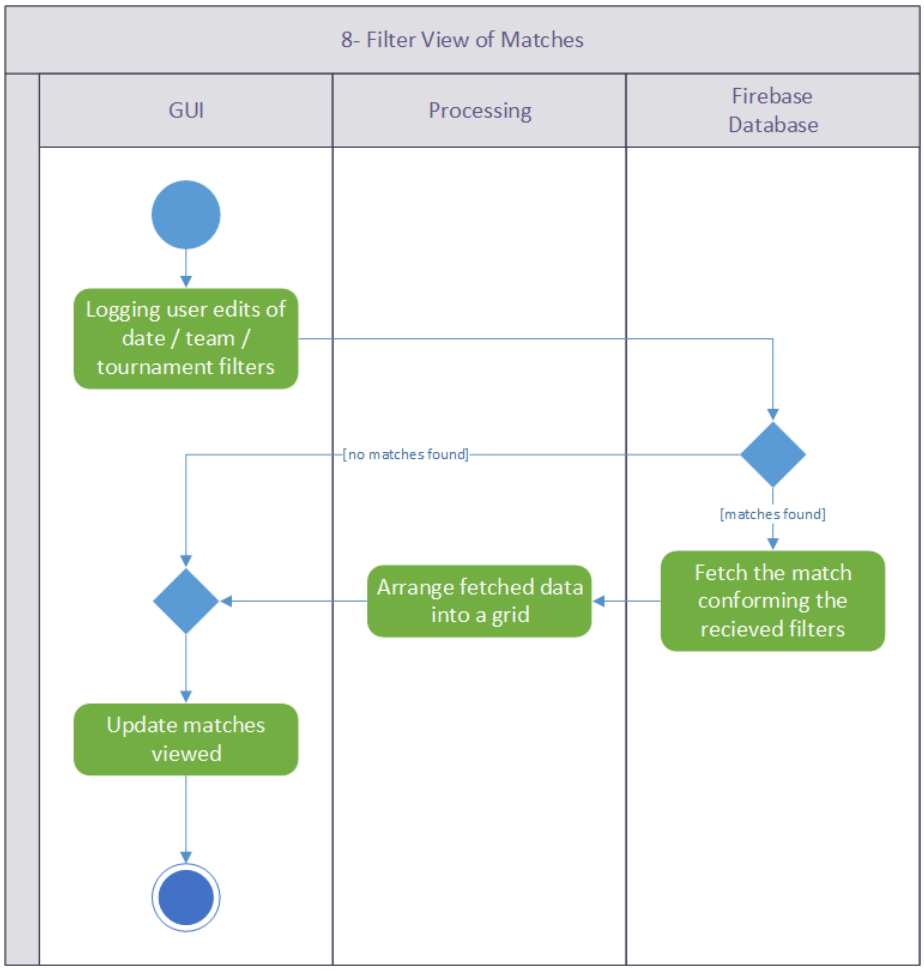

3- Add new account to database
Firebase
Database
Check if email is valid
[valid]
Add to Database
[invalid]
Pass String with
errors found
Pass String with
status

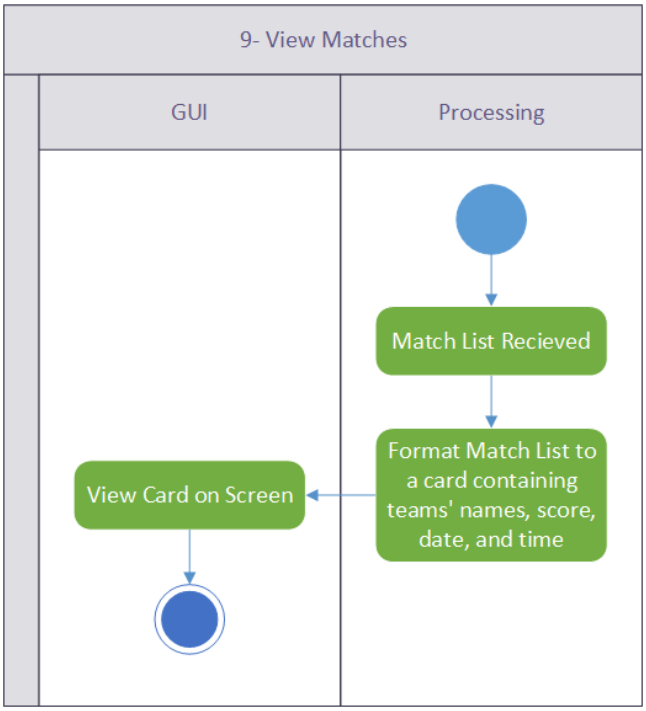

4- Login
GUI
Firebase Database
Log required login info
Verifying Login Credentials
[valid]
Customize Homepage
[invalid]
Pass String with errors found
Inform User with status

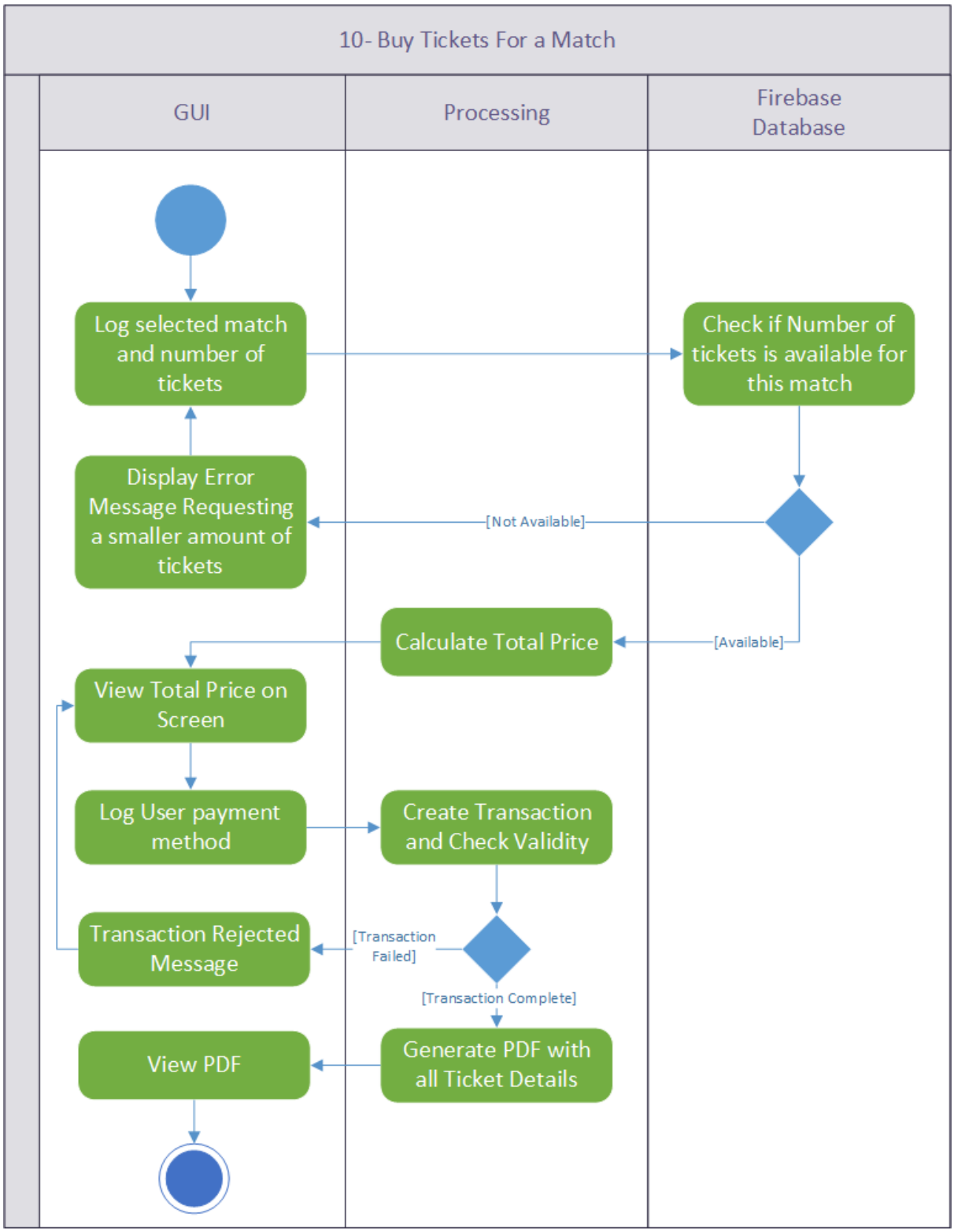

5- Check login validity
Firebase
Database
Verifying Login
Credentials
[valid]
Customize
Homepage
[invalid]
Pass String with
errors found

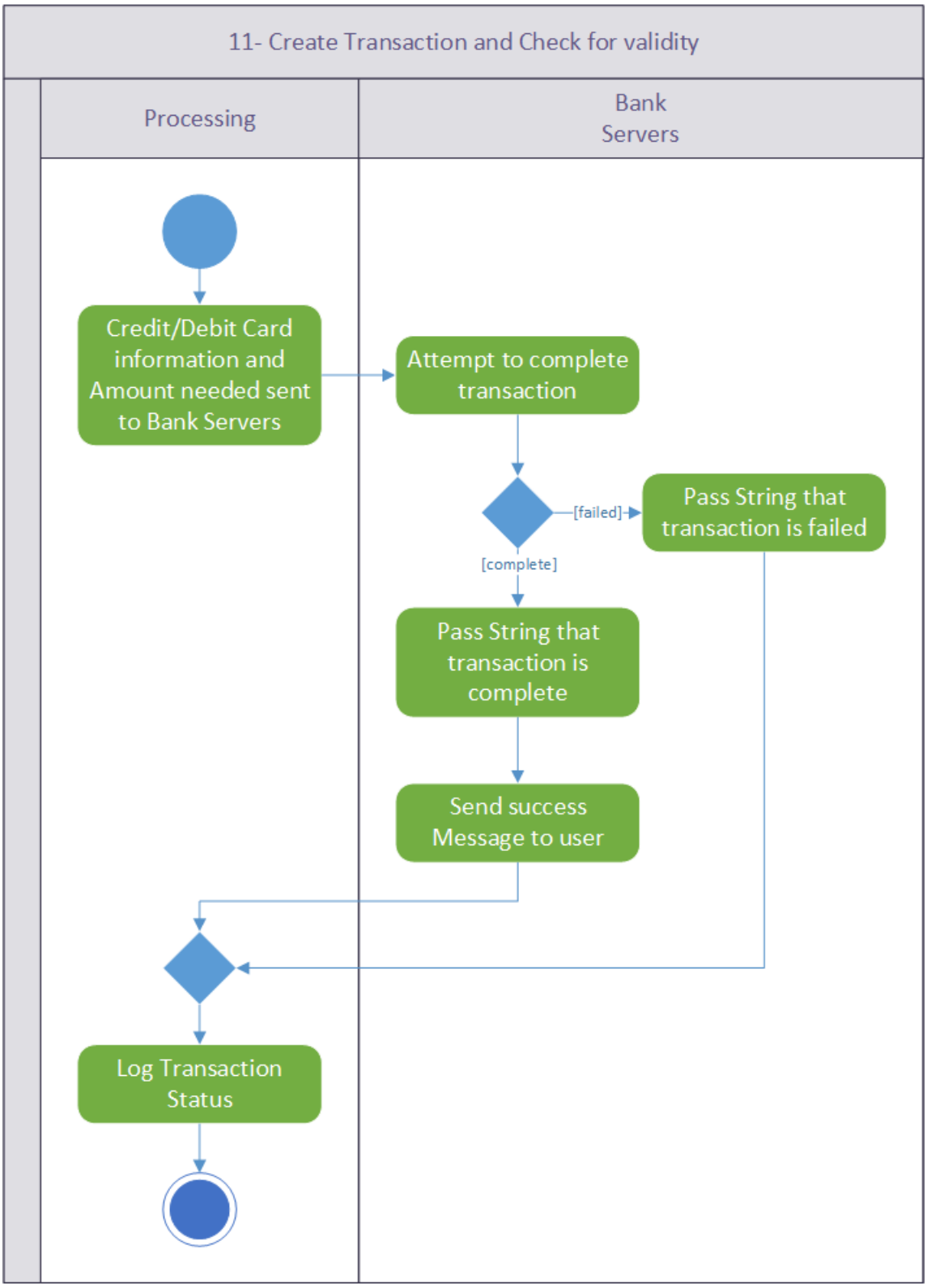

6- Select Favorite Team
GUI
Processing
Firebase Database
Select team name from list
Add team to user favorite team list
Update homescreen with added teams
Add teams to homescreen
Retrive team matches

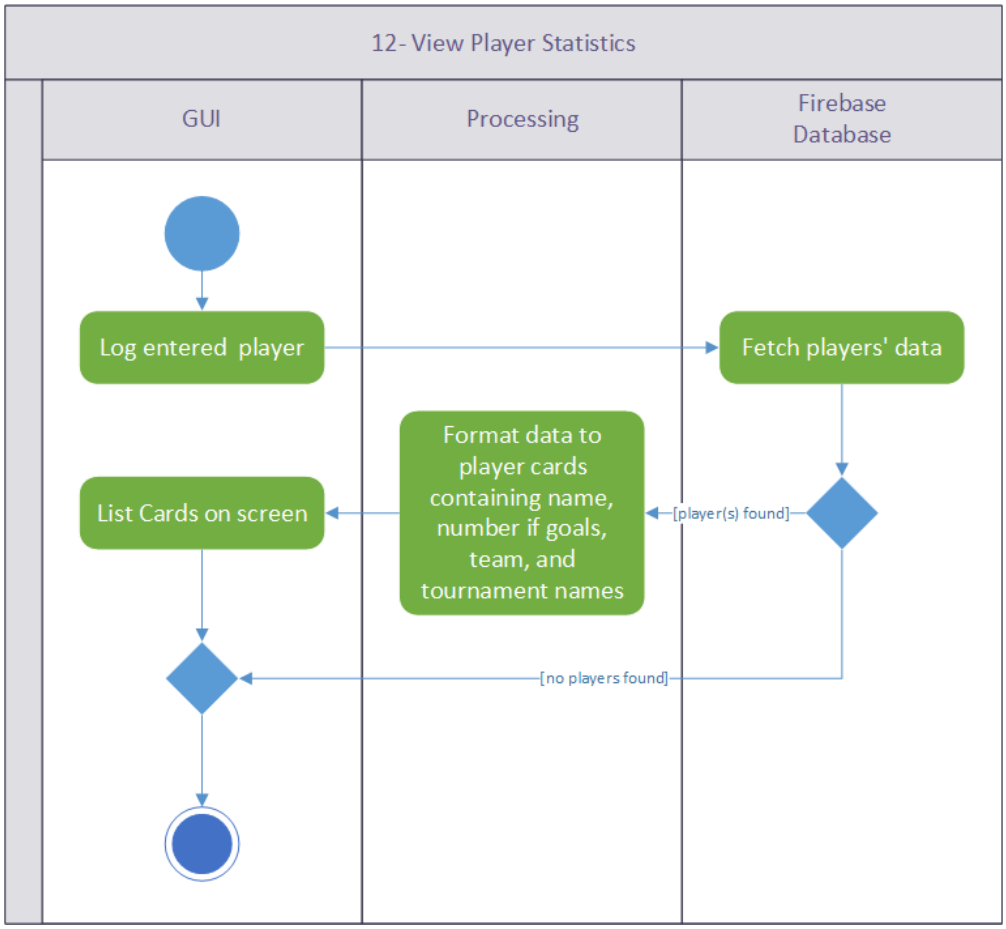

7- Add Team Matches To Homescreen
Processing
Firebase Database
Send teams' list
Fetch foer teams' matches
Update matches' list to be viewed on screen
[matches found]
[no matches found]

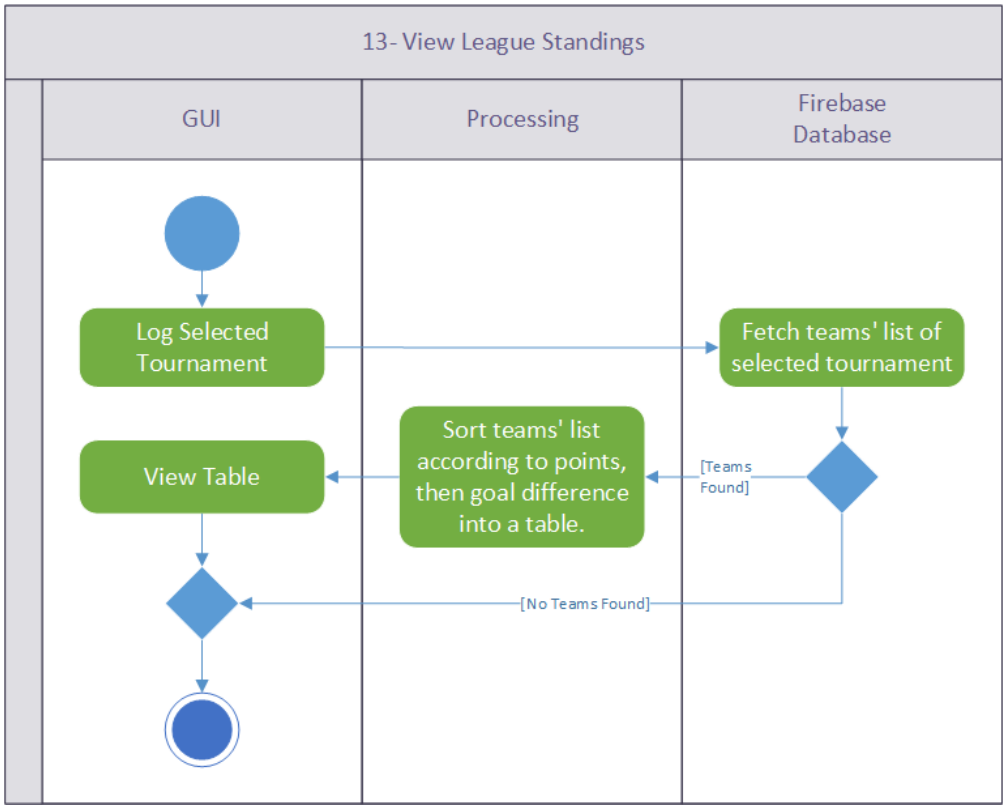

8- Filter View of Matches
GUI
Processing
Firebase Database
Logging user edits of date / team / tournament filters
[no matches found]
[matches found]
Fetch the match conforming the recieved filters
Arrange fetched data into a grid
Update matches viewed

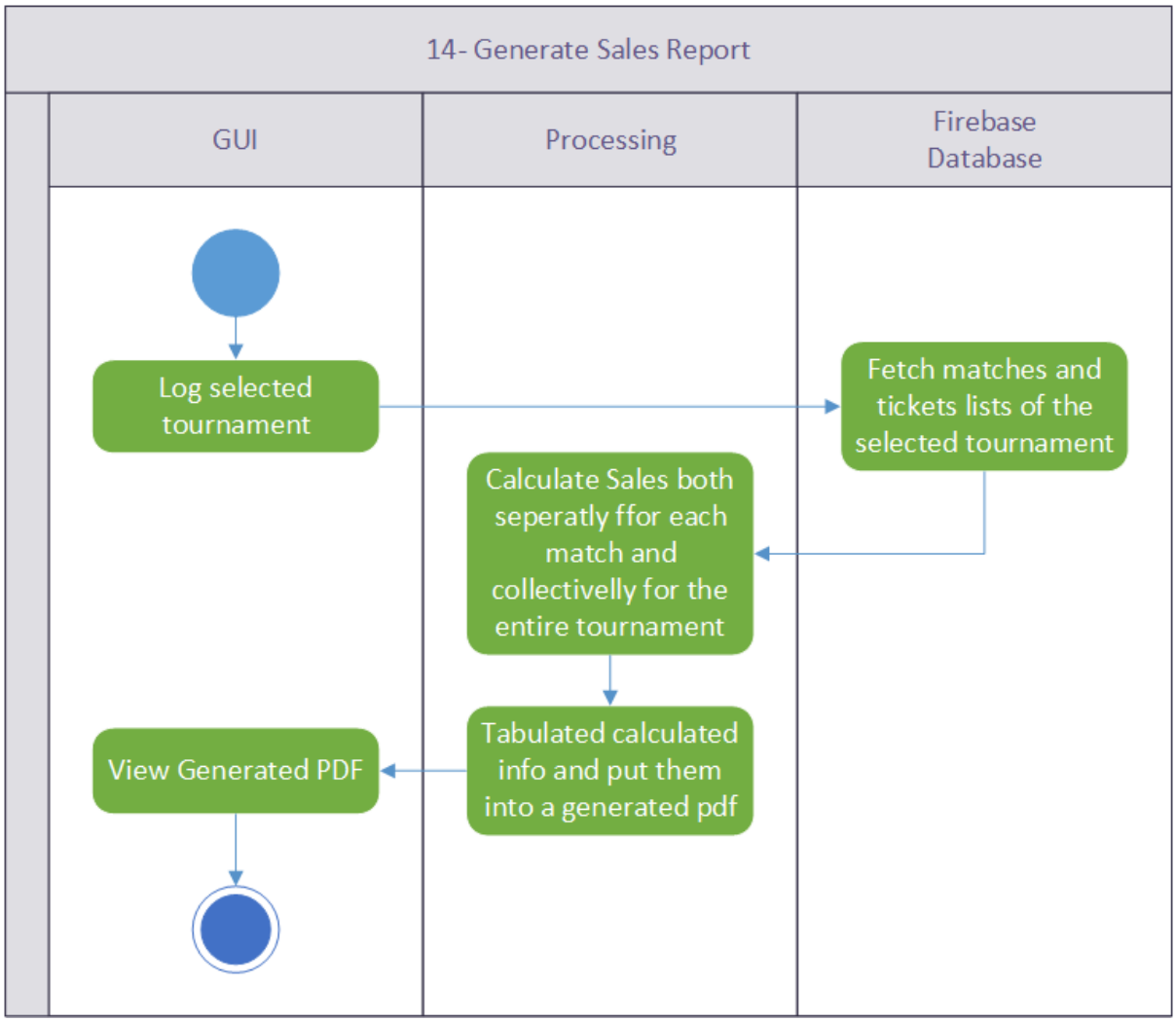

9- View Matches
GUI
Processing
Match List Recieved
Format Match List to a card containing teams' names, score, date, and time
View Card on Screen

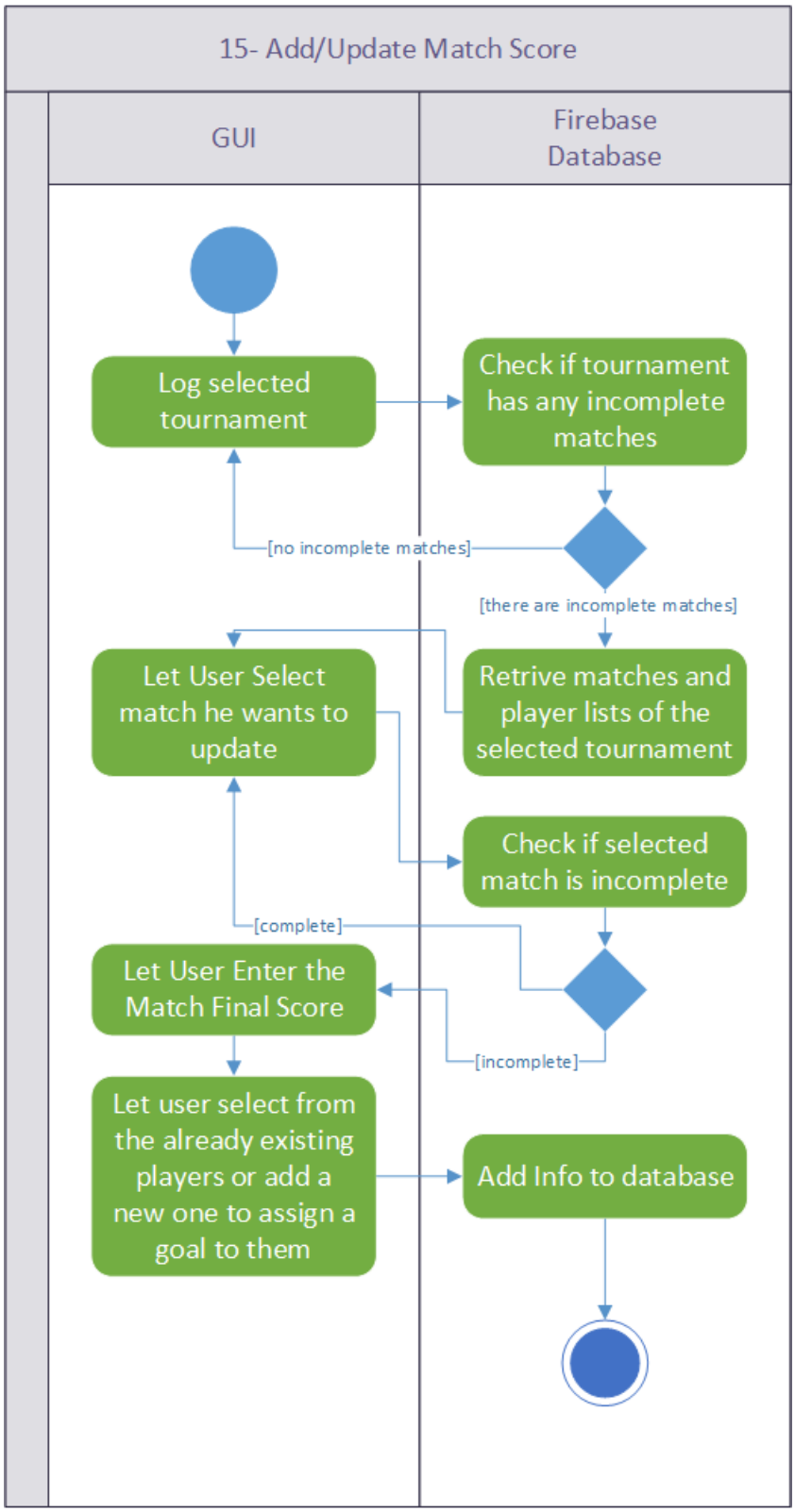

10- Buy Tickets For a Match
GUI
Processing
Firebase Database
Log selected match and number of tickets
Check if Number of tickets is available for this match
Display Error Message Requesting a smaller amount of tickets
[Not Available]
[Available]
Calculate Total Price
View Total Price on Screen
Log User payment method
Create Transaction and Check Validity
Transaction Rejected Message
[Transaction Failed]
[Transaction Complete]
View PDF
Generate PDF with all Ticket Details

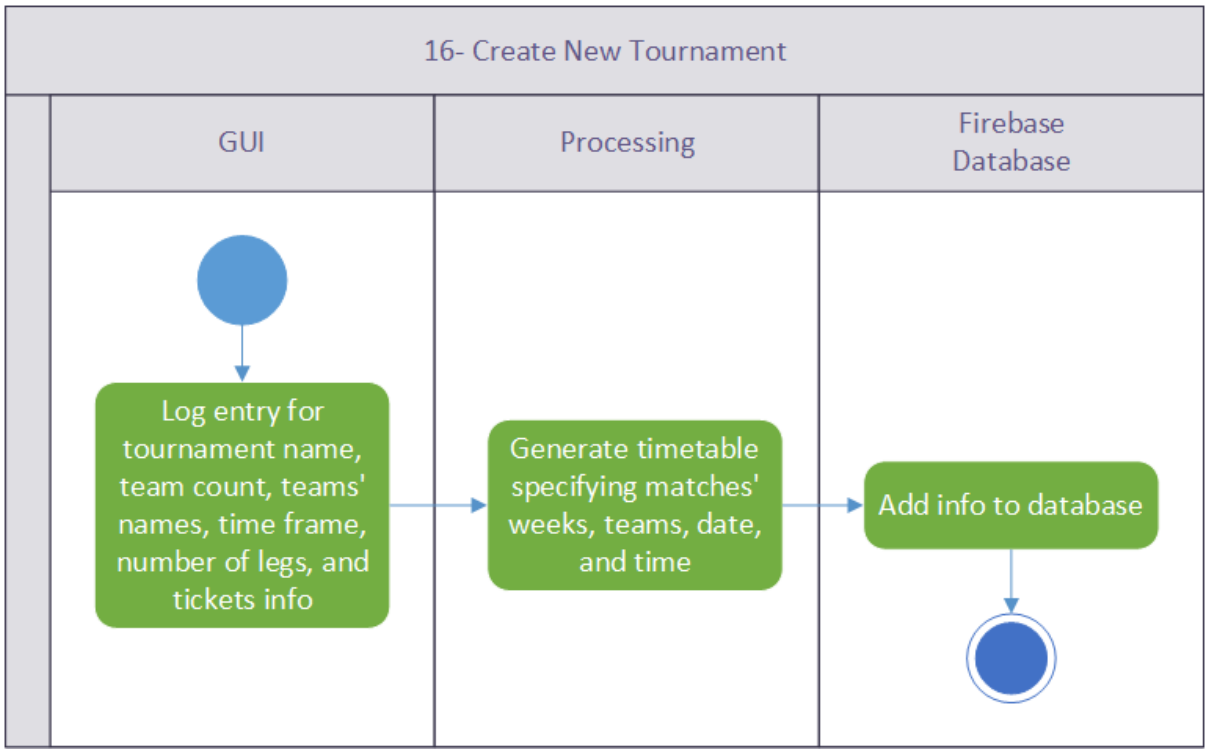

11- Create Transaction and Check for validity
Processing
Bank Servers
Credit/Debit Card information and Amount needed sent to Bank Servers
Attempt to complete transaction
[failed]
Pass String that transaction is failed
[complete]
Pass String that transaction is complete
Send success Message to user
Log Transaction Status


12- View Player Statistics
GUI
Processing
Firebase Database
Log entered player
Fetch players' data
Format data to player cards containing name, number if goals, team, and tournament names
List Cards on screen
[player(s) found]
[no players found]


13- View League Standings
GUI
Processing
Firebase Database
Log Selected Tournament
Fetch teams' list of selected tournament
Sort teams' list according to points, then goal difference into a table.
View Table
[Teams Found]
[No Teams Found]


14- Generate Sales Report
GUI
Processing
Firebase Database
Log selected tournament
Fetch matches and tickets lists of the selected tournament
Calculate Sales both seperatly ffor each match and collectivelly for the entire tournament
Tabulated calculated info and put them into a generated pdf
View Generated PDF


15- Add/Update Match Score
GUI
Firebase Database
Log selected tournament
Check if tournament has any incomplete matches
[no incomplete matches]
[there are incomplete matches]
Let User Select match he wants to update
Retrive matches and player lists of the selected tournament
Check if selected match is incomplete
[complete]
Let User Enter the Match Final Score
[incomplete]
Let user select from the already existing players or add a new one to assign a goal to them
Add Info to database


16- Create New Tournament
GUI
Processing
Firebase Database
Log entry for tournament name, team count, teams' names, time frame, number of legs, and tickets info
Generate timetable specifying matches' weeks, teams, date, and time
Add info to database

6. Noun Extraction and CRC Cards

6.1. Noun Extraction Method

Problem Definition

"Admin users can create and manage tournaments, add teams, and edit match scores; while normal users can follow teams and tournaments, view match scores and player stats, and buy tickets. You must be a registered user to add your favorite teams, view tournament standings, or buy match tickets either by cash or credit card. When an organizer user adds a team to his tournament, the team admin must accept this invitation to be added to this tournament. All users including anonymous can view match scores, goal scorers, and player stats. Finally, all registered users can edit some of their account info."

All Identified Nouns:

Admin User, Tournament, Team, Match Score, Player Stats, Tickets, Cash, Credit Card, Registered User, Match Tickets, Team Admin, Invitation, Goal, Account Info

Excluded Nouns:

Cash, Credit Card as they lie outside the problem boundary.

Invitation, Goal has no physical existence.

Admin User, Normal user can be subclasses for an abstract class *User*

Match Score, Player Stats, Match Ticket, and Team Admin are actually attributes for the classes *Match, Player, Match, and Team* respectively.

Extracted Nouns:

User (Admin User & Registered User), Tournament, Team, Match, Player, Ticket

6.2. CRC Cards Method

Figure 3: CRC cards

7. Class Model

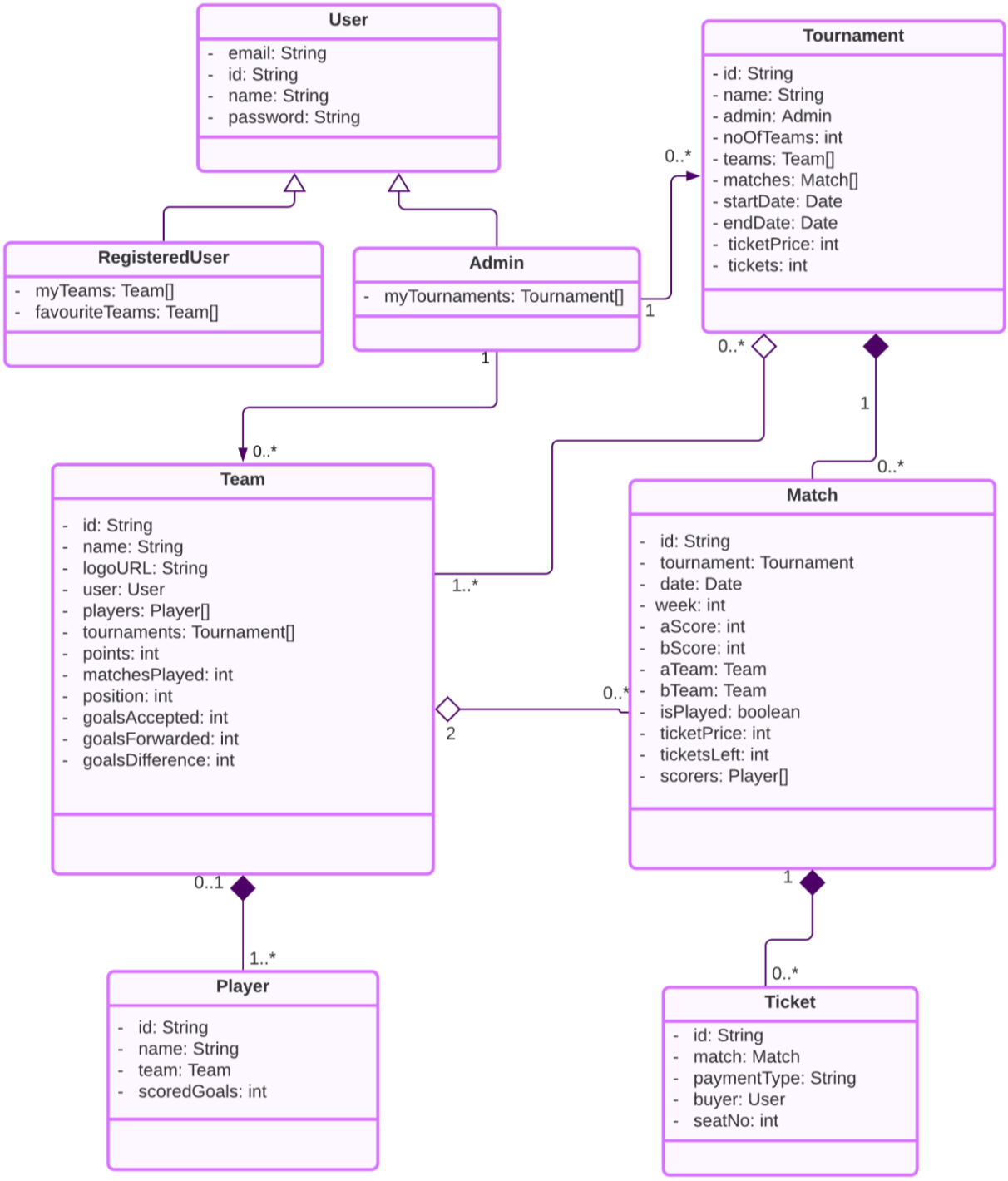


Figure 4: Class Model

The UML class model of this system, shown in Figure 4 clearly demonstrates all the classes, their relationships, and their attributes. Initially, noun extraction method was used to determine the potential classes for this system, then CRC cards was used to determine possible class members and the relations between them.

8. State Diagram

The detailed state diagram for the whole system, provided in Figure 23, shows a sequence of states the NAZAMLY system can assume during its lifetime, together with the stimuli that cause those changes between every one of the different states starting from the initial start of the system, symbolled by a circle at the top left of the diagram, until the different possible end states, symbolled using 2 centered circles.

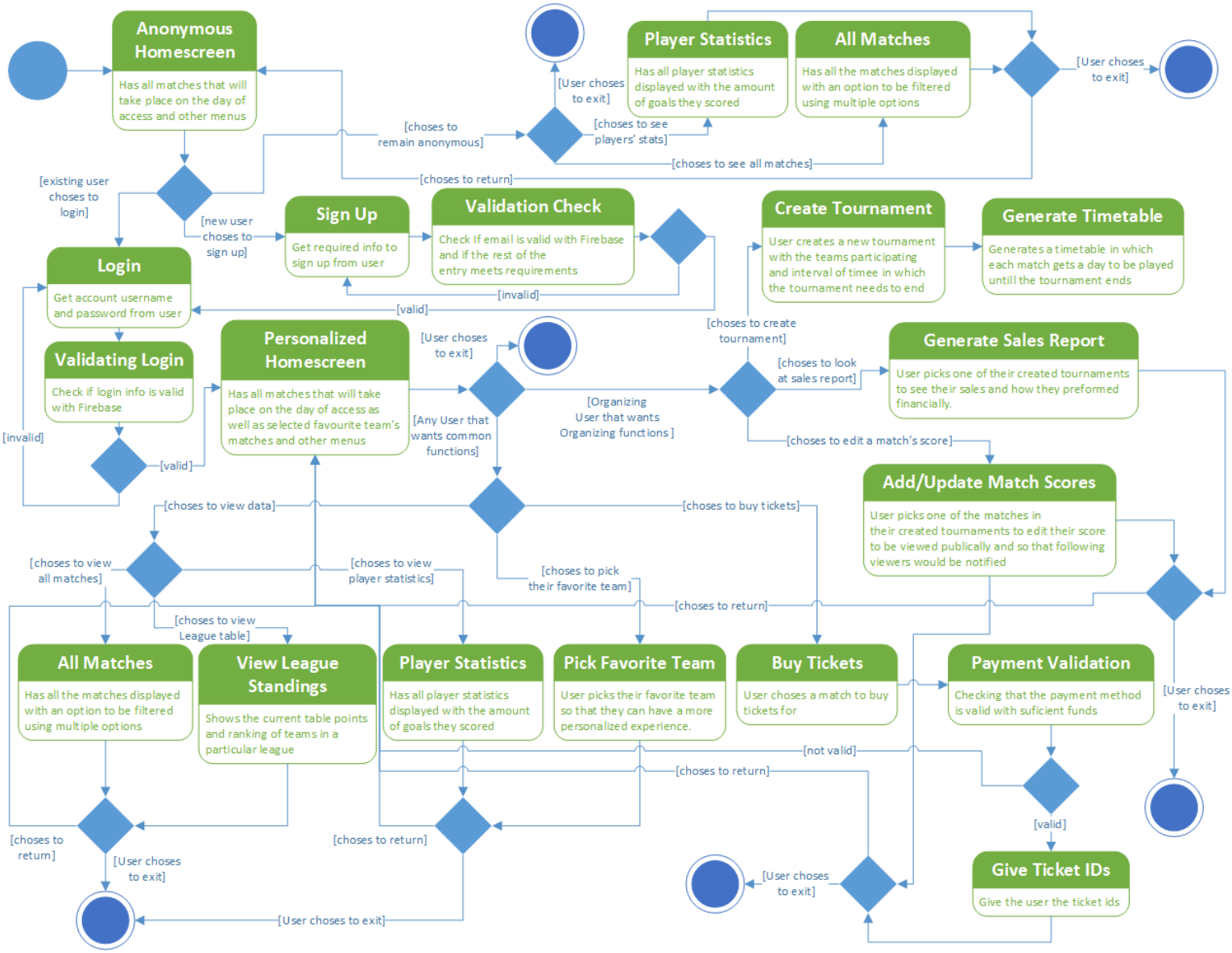
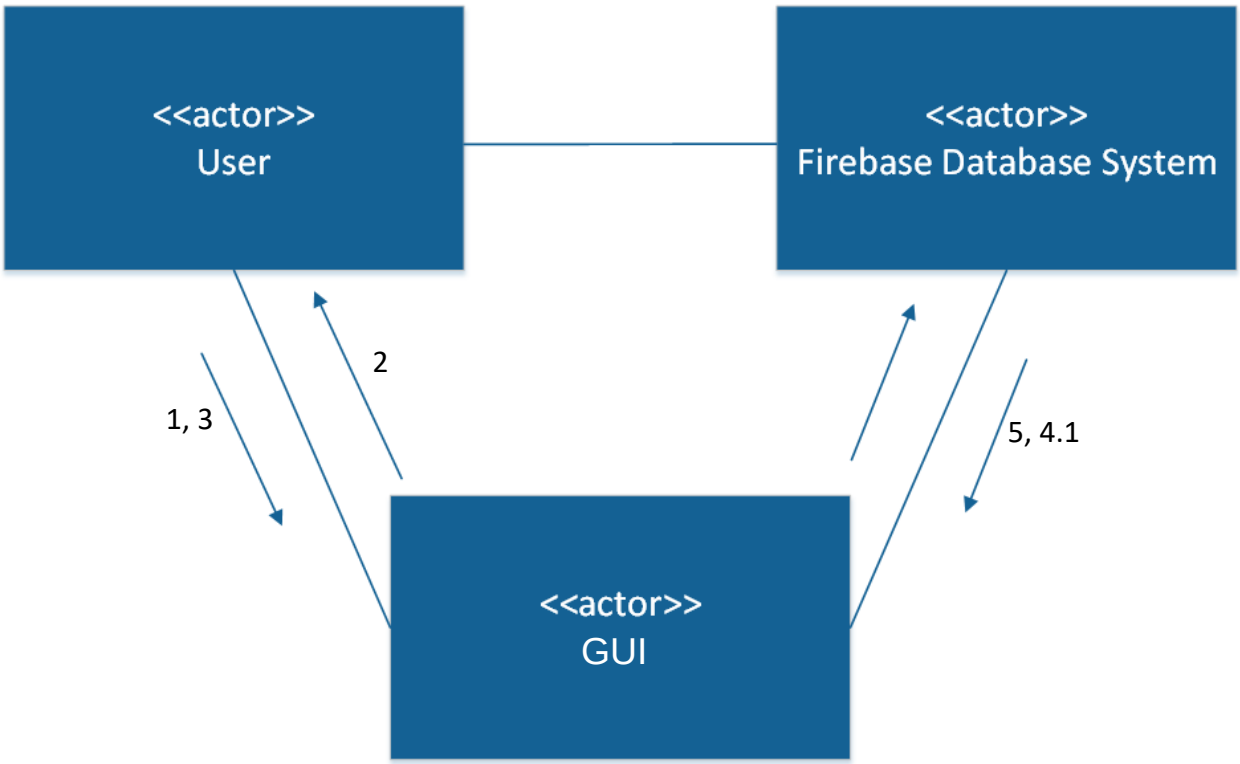


Figure 5: State Diagram

9. Interaction (Collaboration) Diagrams

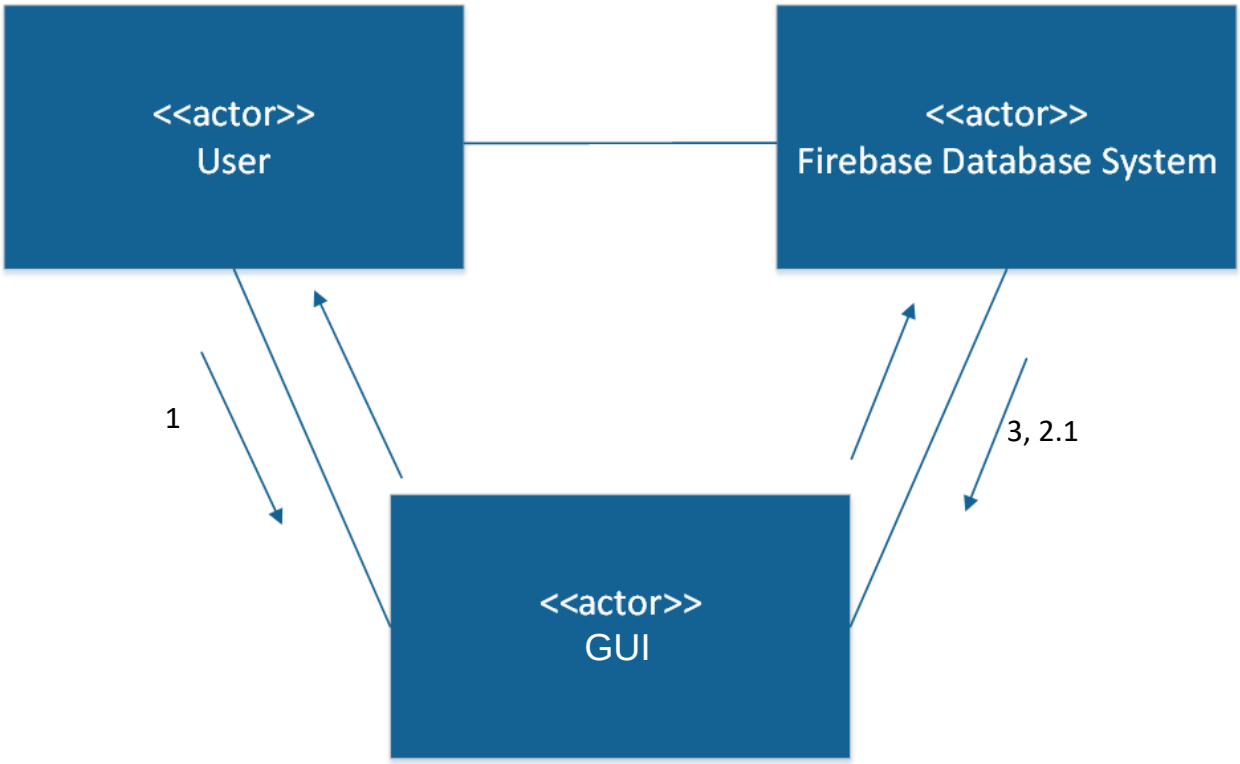
In this section, collaboration diagram was adopted as an interaction diagram showing the interaction between the different actors of the system. Below, a collaboration diagram is provided for each one of the aforementioned use cases.

1- WWeSign Up



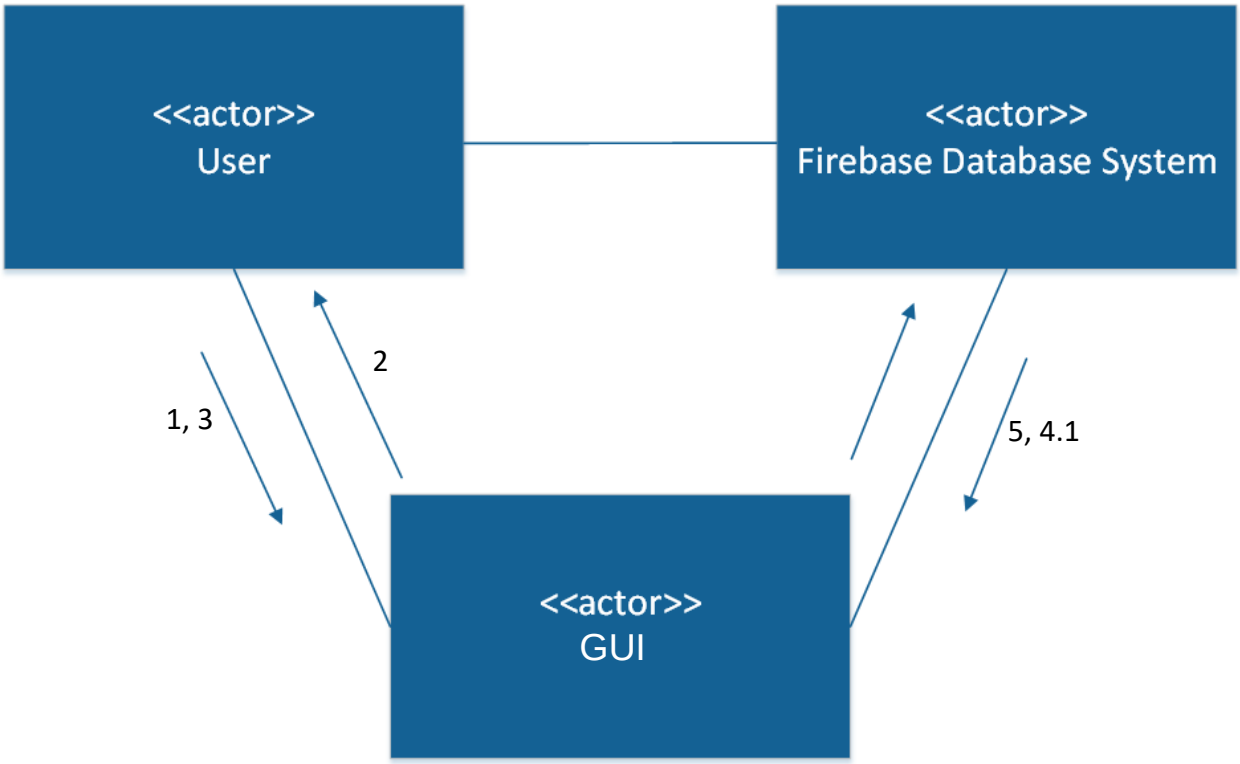
1.	The user requests to register for a new account.
2.	GUI brings up sign up form
3.	The user fills in the required personal information.
4.	The user's details are sent to be verified using the credentials' authentication database system.
5.	A success message pops up.
4.1	The request is rejected, and an error message pops.

2- Check that provided information meets requirements.



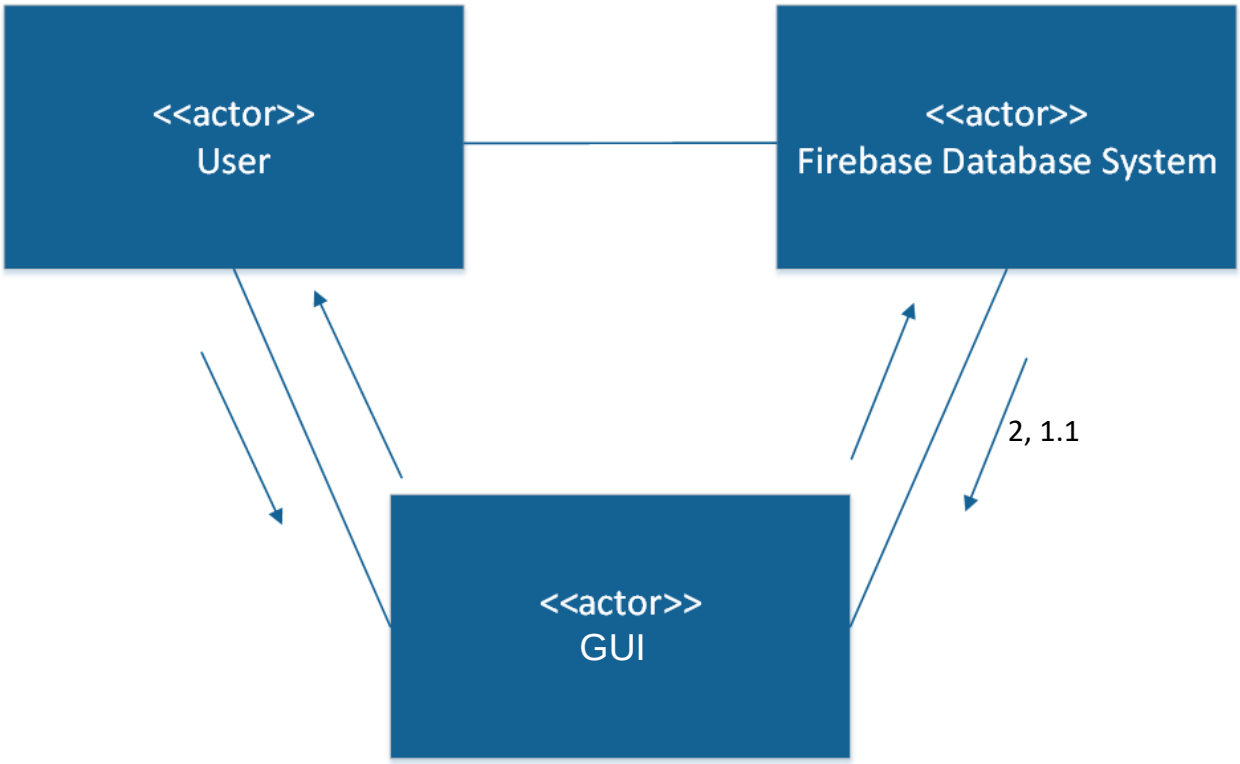
1.	The user fills in the required personal information.
2.	The user's details are sent to be verified using the credentials' authentication database system.
3.	Account is added, and success message pops up.
2.1	The request is rejected, and an error message pops.

3- Login



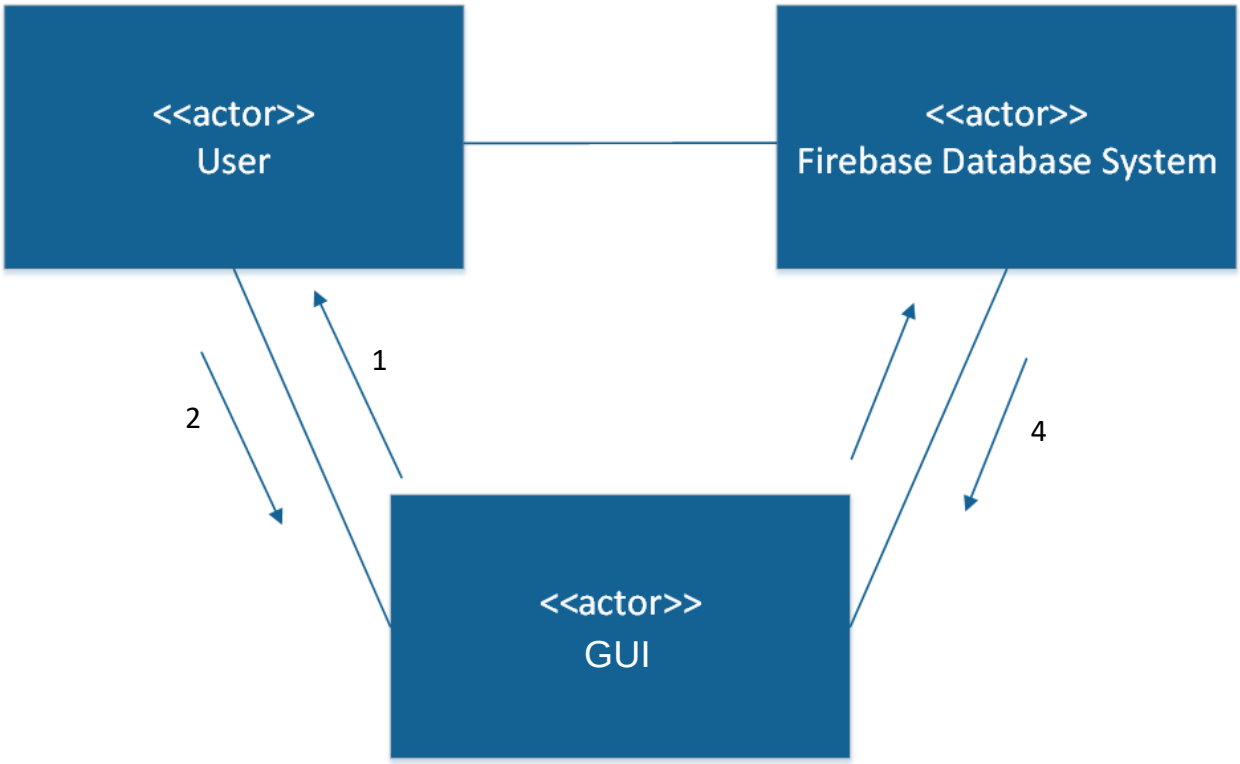
1.	The user requests to login to his account.
2.	GUI brings up login form
3.	The user fills in his username and password.
4.	The provided credentials are verified using the credentials' authentication database system.
5.	The user logs in and return to his customized home page
4.1	Login fails, and a specified error message pops.

4- Check login validity



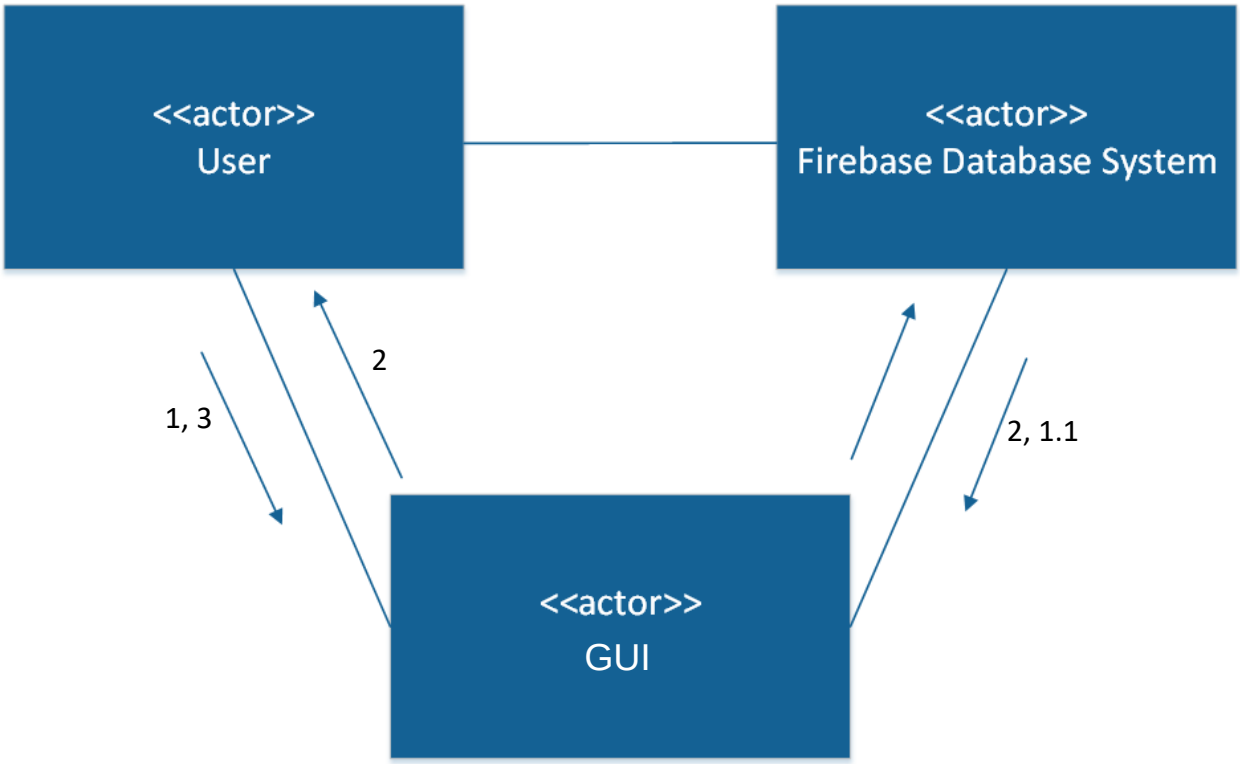
1.	The database system is provided user credentials.
2.	Return success message
1.1	Return error message.

5- Select Favorite Team



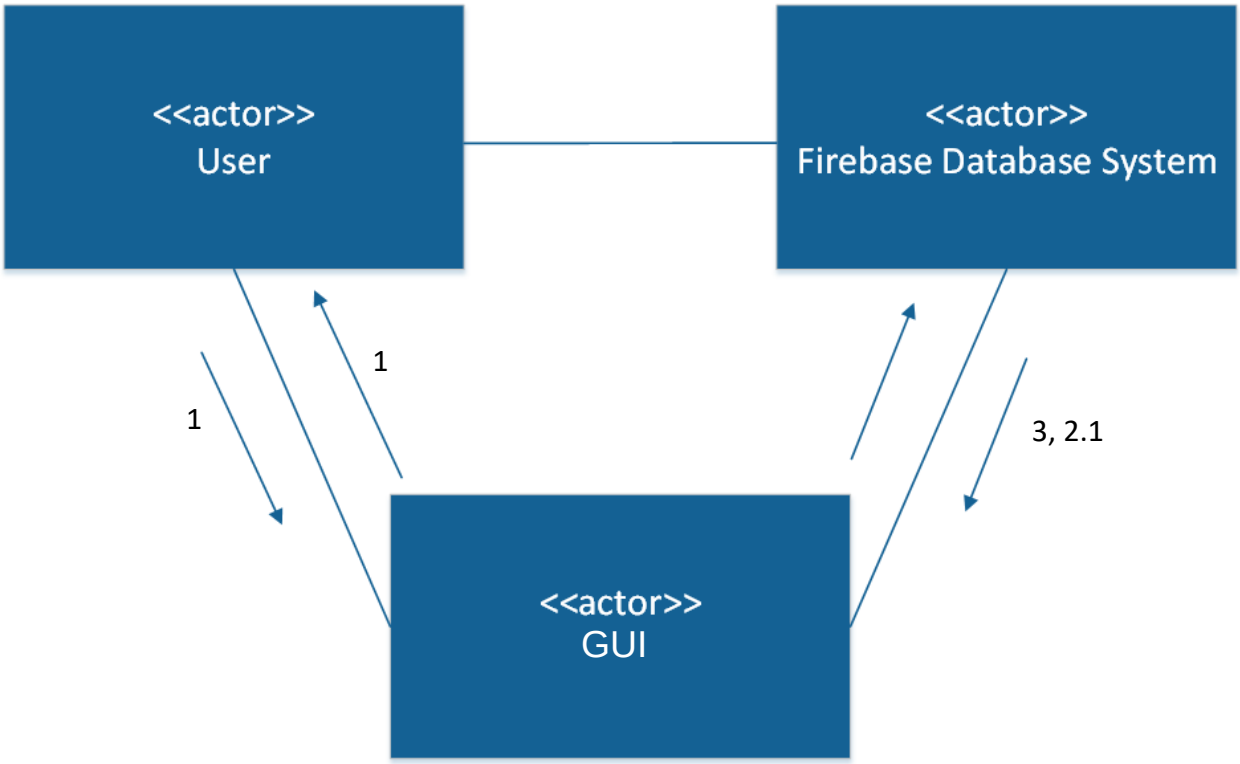
1.	A team list is provided with all the included teams
2.	User selects team name from list
3.	The team is added to his list
4.	Home page matches are updated with the team matches

6- Add Team Matches to Home screen



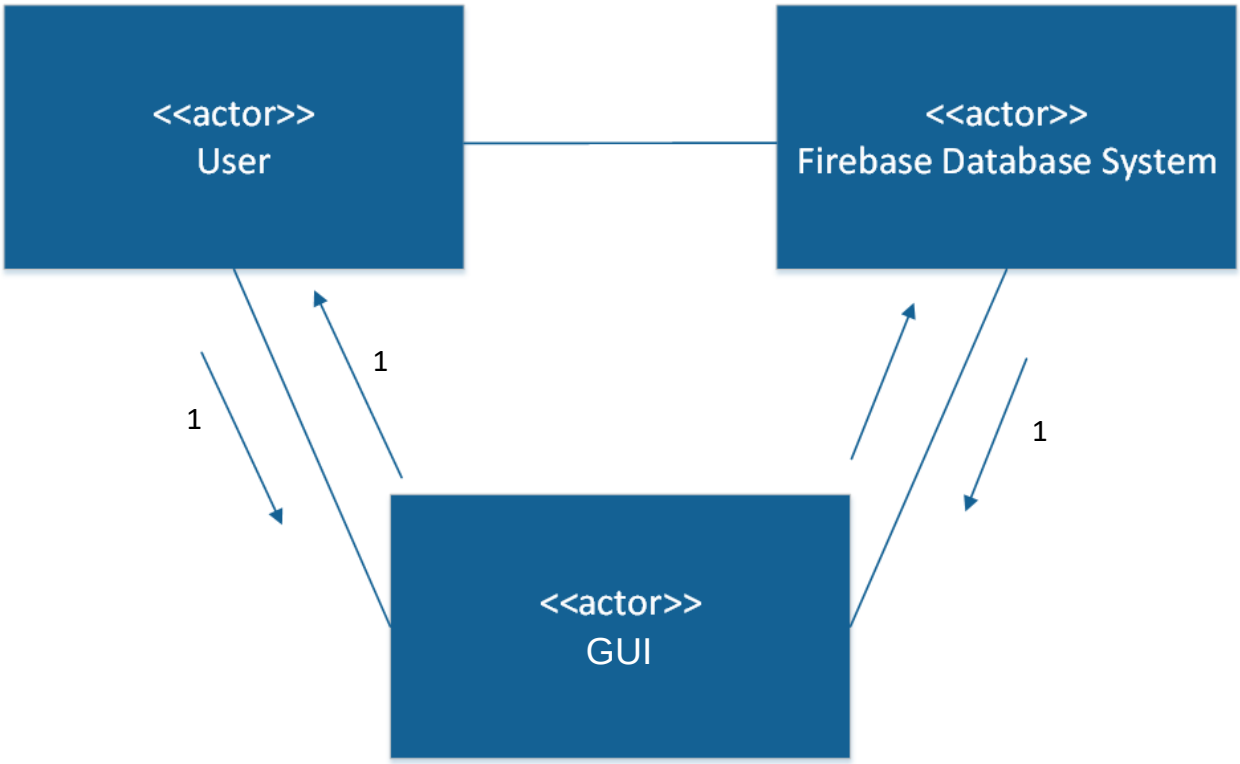
1.	Teams' list is provided to fetch for their matches.
2.	Matches' list is retrieved and sent to be viewed on screen.
1.1	No matches are found, and a message is viewed on screen

7- Filter view of Matches



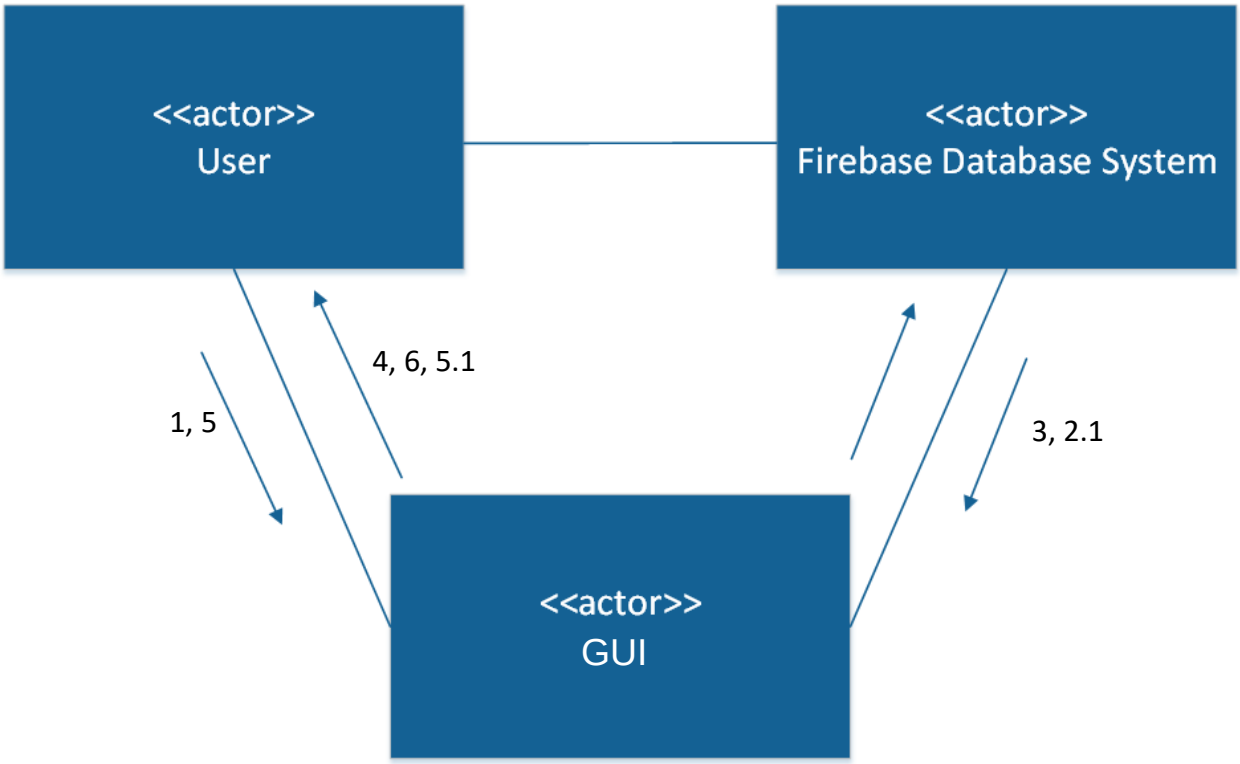
1.	The user edits any of the date, team, or tournament filters.
2.	Database system fetches the match conforming the received filters.
3.	The matches' list is retrieved from the database to be viewed on the user's screen.
2.1	No matches are found according to the set filters, and a message is viewed on screen

8- View Matches



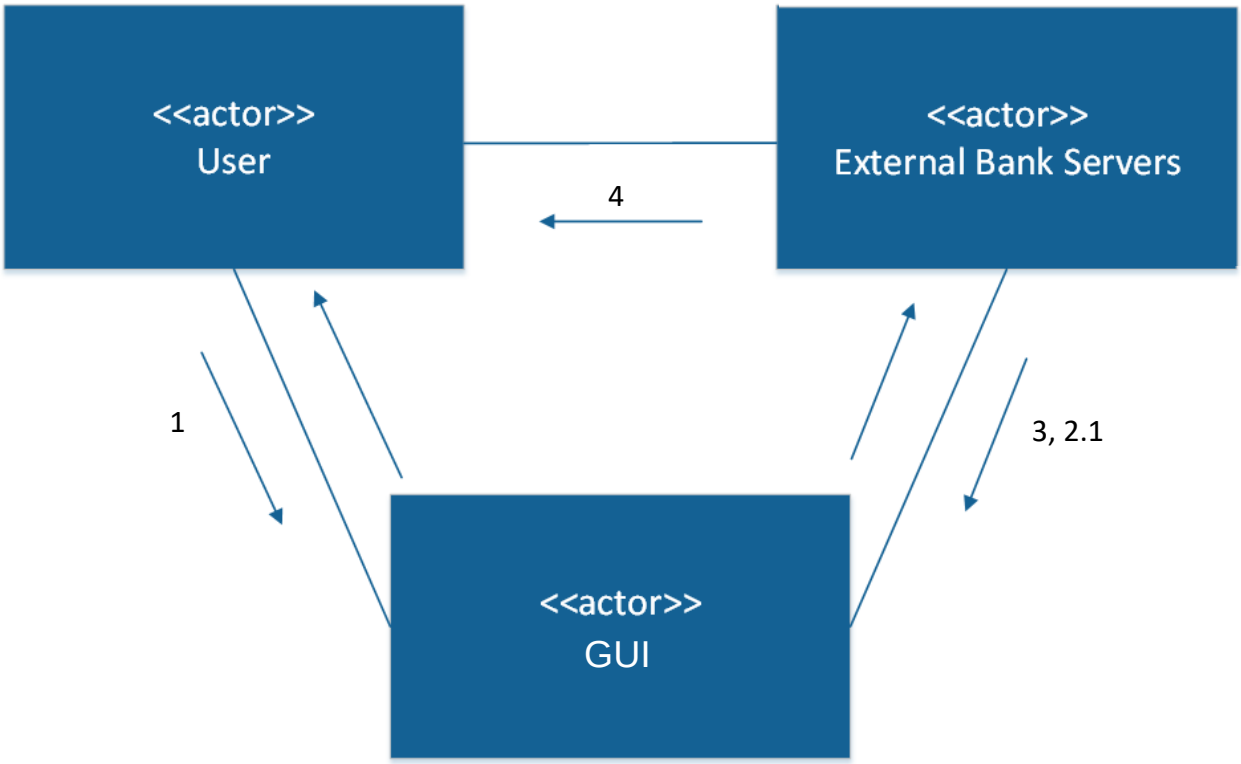
1. Matches' list received, formatted to cards that include team names, scores, fate and time, and list them on the user's screen.

9- Buy Tickets for a Match



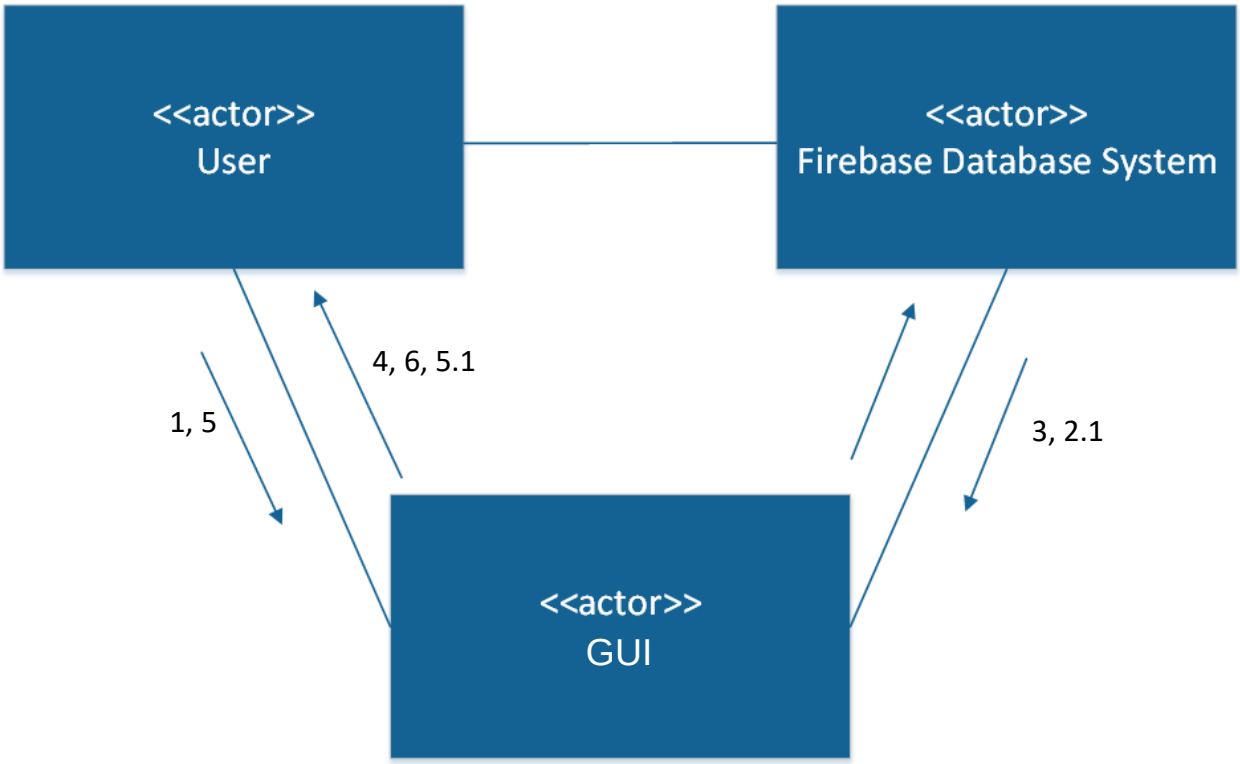
1.	User selects match and number of tickets
2.	Database check if number of tickets required is more than or equal available tickets for this match.
3.	Total price is viewed on the screen.
4.	User is asked to choose a payment method
5.	User chooses cash or credit card payment.
6.	Payment validated and success payment message received, and user gets a PDF file includes all tickets details.
2.1	Error message that the quantity chosen is not available.
5.1	Transaction rejected and failed payment message received.

10- Create transaction & check for validity.



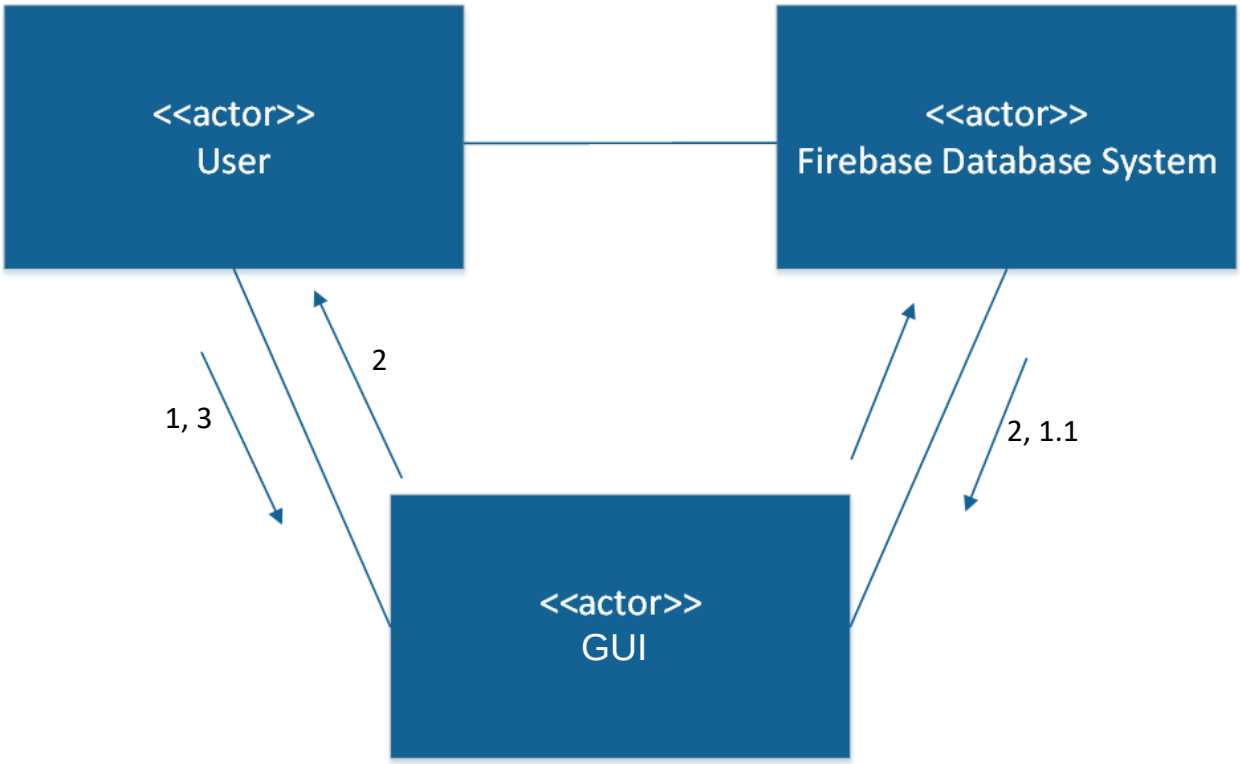
1.	Credit/debit card information received.
2.	Information sent to bank servers to complete transaction.
3.	Success message received.
4.	Success message sent to the user.
2.1	Transaction refused message received.

11- View players statistics.



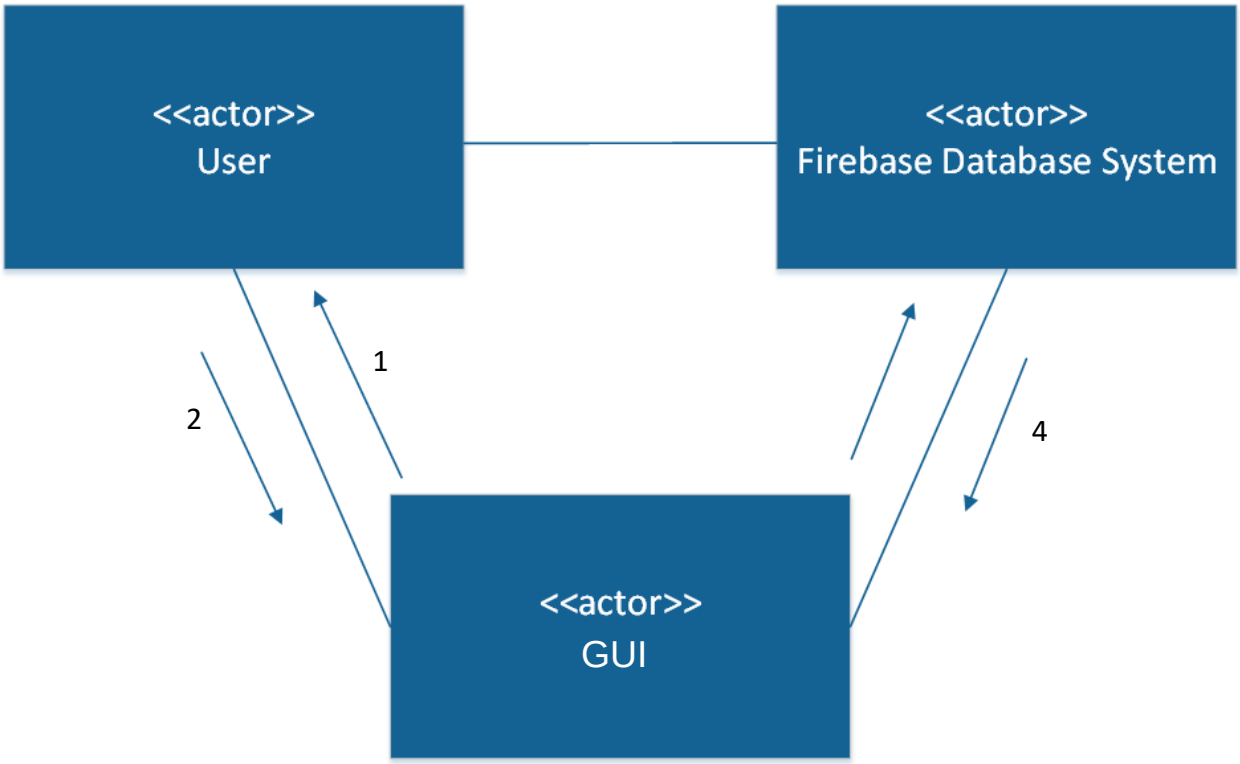
1.	Fetch players' data from database.
2.	Players' data are formatted to a player card format including his name, number of goals, team, and tournament names. Cards are then listed on user's screen.
3.	Total price is viewed on the screen.
4.	User is asked to choose a payment method
5.	User chooses cash or credit card payment.
6.	Payment validated and success payment message received, and user gets a PDF file includes all tickets details.
2.1	Error message that the quantity chosen is not available.
5.1	Transaction rejected and failed payment message received.

12- View league standings.



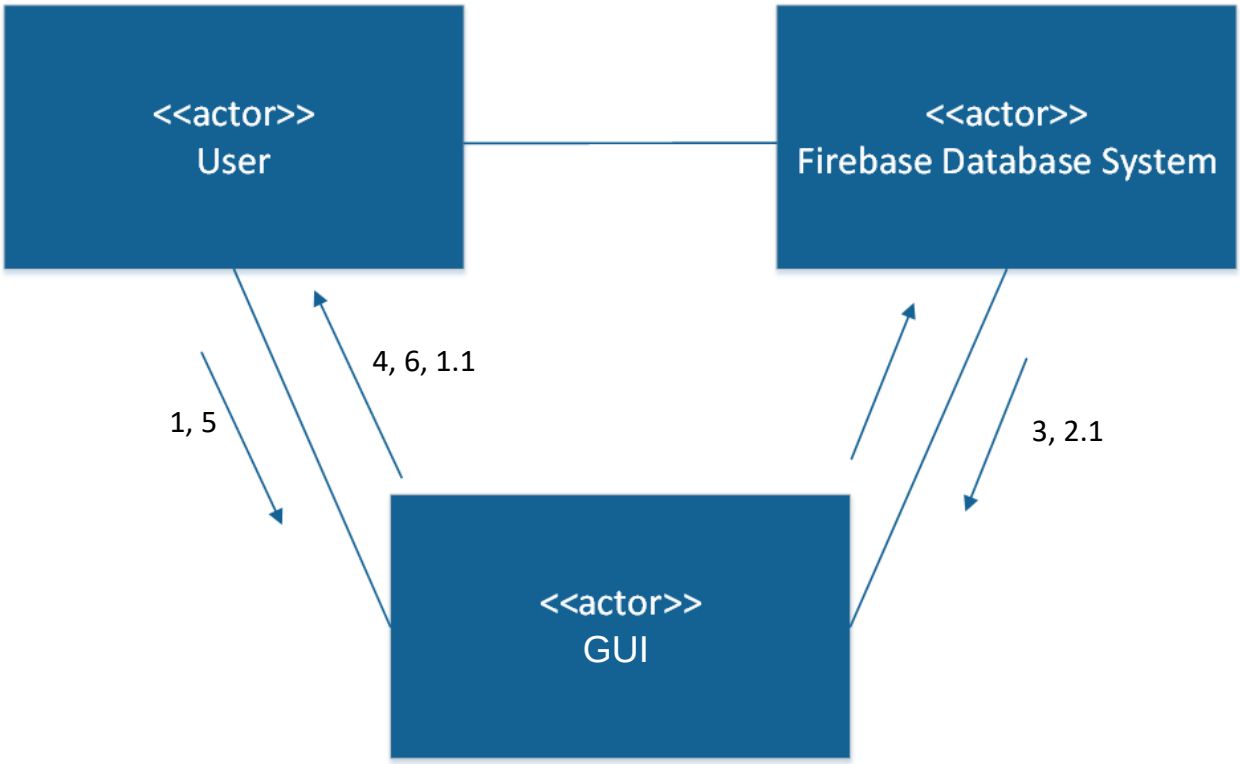
1.	Fetch teams' list of the selected tournament from database.
2.	Sort teams' list according to points account, if two of them have equal number of points, refer to the goals difference and display all teams' details in a table view.
1.1	No teams are found, and an error message is viewed on screen

13- Generate sales report.



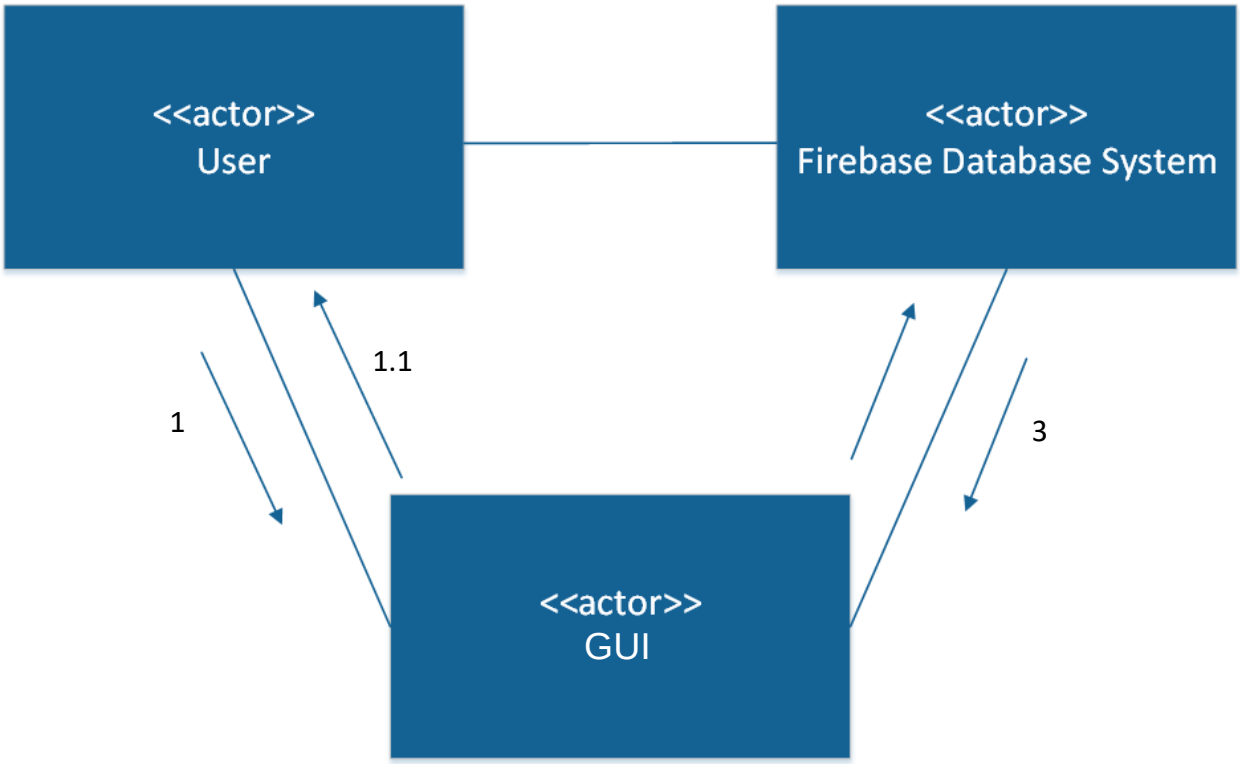
1.	A tournament list is provided with all the available tournament
2.	Tournament Organizer selects tournament from list
3.	Retrieve matches and tickets lists of the selected tournament from the database.
4.	Create a sale formatted PDF file to view and open

14- Add/update match score.



1.	Tournament organizer selects a tournament.
2.	Retrieve matches and players names lists of the selected tournament from the database.
3.	Lists are returned.
4.	User is asked to select a match from the list provided.
5.	Organizer chooses the match he wants to update and adds the match final score.
6.	Match final score updated message appears.
1.1	Error message appears stating that selected tournament does not have any match left not played.
2.1	Error message that the selected match has a final score assigned to it.

15- Add/update match score.



1.	Tournament organizer enters tournament details including, tournament name, teams count, teams' names, time frame, and number of legs.
2.	A timetable is generated specifying matches weeks, opponents, date, and time. All details are then added to the database.
3.	Success label pops up indicating the creation of a new tournament.
1.1	Error message due to similarity between new tournament and other tournaments names or start date comes after the finish date.

10. Class Diagram

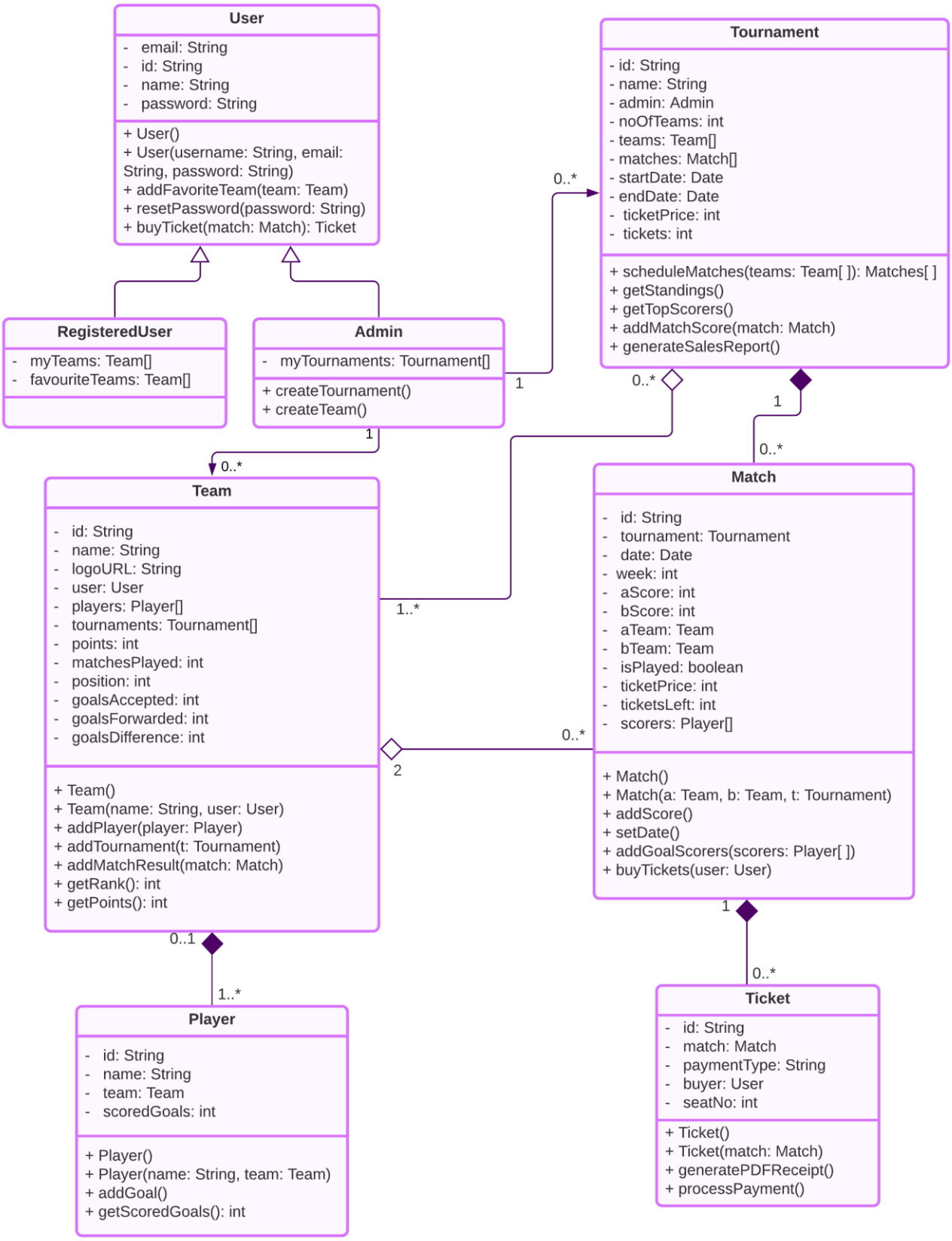


Figure 6: Class Diagram

- **Assumptions & Explanations:**
    - All implemented setters and getters for class attributes are not explicitly shown in the class diagram.
    - Class ***User*** is abstract as any user is assumed to be either an **Admin** or a regular **RegisteredUser**.
    - Any tournament must contain at least 1 team, where a team can participate in more than one tournament.
    - Any **Player** can be only registered in one **Team**, so there is no option for a registered player to be added to two different Teams.
    - All matches within a specific tournament have the same ticket price, so this attribute is added to the aggregating class, **Tournament.**

11. Client-Object Relation Diagram

The below figure shows the client-object relation diagram for this system. An arrow heading from object A to object B means that object B is a client of object A and able to create an instance of it. **AppController** is the only class object that will be initiated by the main application object (**NazamlyApplication**), because if initiated, all other objects can be initiated accordingly.

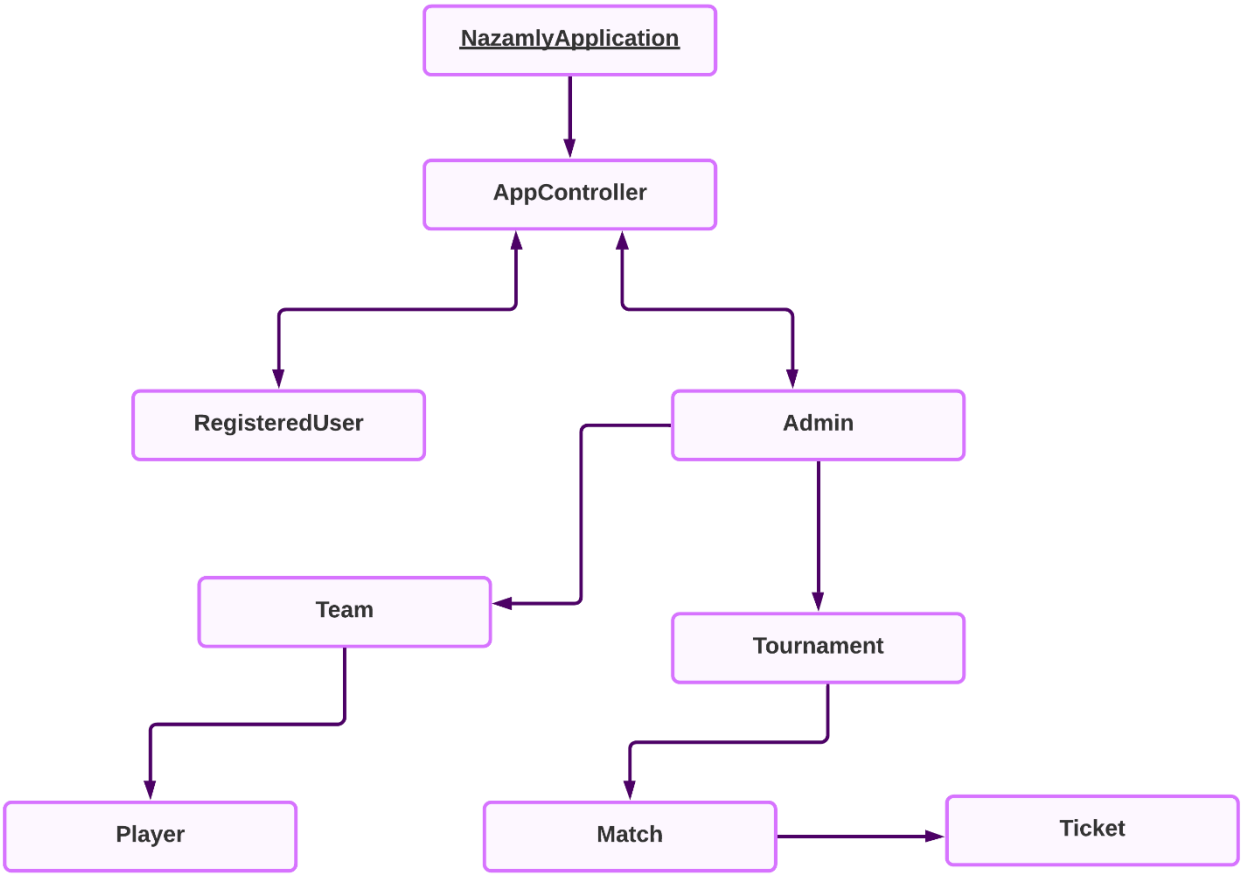


Figure 7: Client-Object Relation Diagram

12. OOAD Methodologies

The main goal of the Object-Oriented Analysis and Design (OOAD) is to reach the detailed specifications of all classes including its attributes, methods in addition to the relationships between these classes. In the analysis phase, classes names, attributes, and relationships between them are identified. Later in the design phase, class methods are identified and verified.

Over the years multiple methodologies have appeared and developed by several people. For instance, Beck and Cunningham came up with Class-Responsibility-Collaboration (CRC) cards in 1986, which was applied on our system and discussed in Section 6 of this document.

During the analysis and design of our Nazamly system we followed and applied 2 well-known methodologies, the Object modeling technique (OMT) and the Booch methodology.

12.1. Object Modeling Technique (OMT)

OMT describes a method for the analysis, design, and implementation of a system using an object-oriented technique.

In the analysis phase, we try to identify objects and relationships between them based on the goal and domain of the problem, without giving much attention to the attributes or the methods of these objects.

1. ***Analysis***

In this phase the preparation of precise and correct modelling of the real-world problems is prepared. Analysis phase starts with setting a goal i.e., finding the problem statement.

1.1. Problem statement of our system was as follows:

“Admin users can create and manage tournaments, add teams, and edit match scores; while normal users can follow teams and tournaments, view match scores and player stats, and buy tickets. You must be a registered user to add your favorite teams, view tournament standings, or buy match tickets either by cash or credit card. When an organizer user adds a team to his tournament, the team admin must accept this invitation to be added to this tournament. All users including anonymous can view match scores, goal scorers, and player stats. Finally, all registered users can edit some of their account info.”

Problem statement is further divided into object, dynamic and functional model.

1.2. ***Object model:***

Presented by the object model that contains the suggested classes names and the relationships between them. (Class diagram with no attributes or methods)

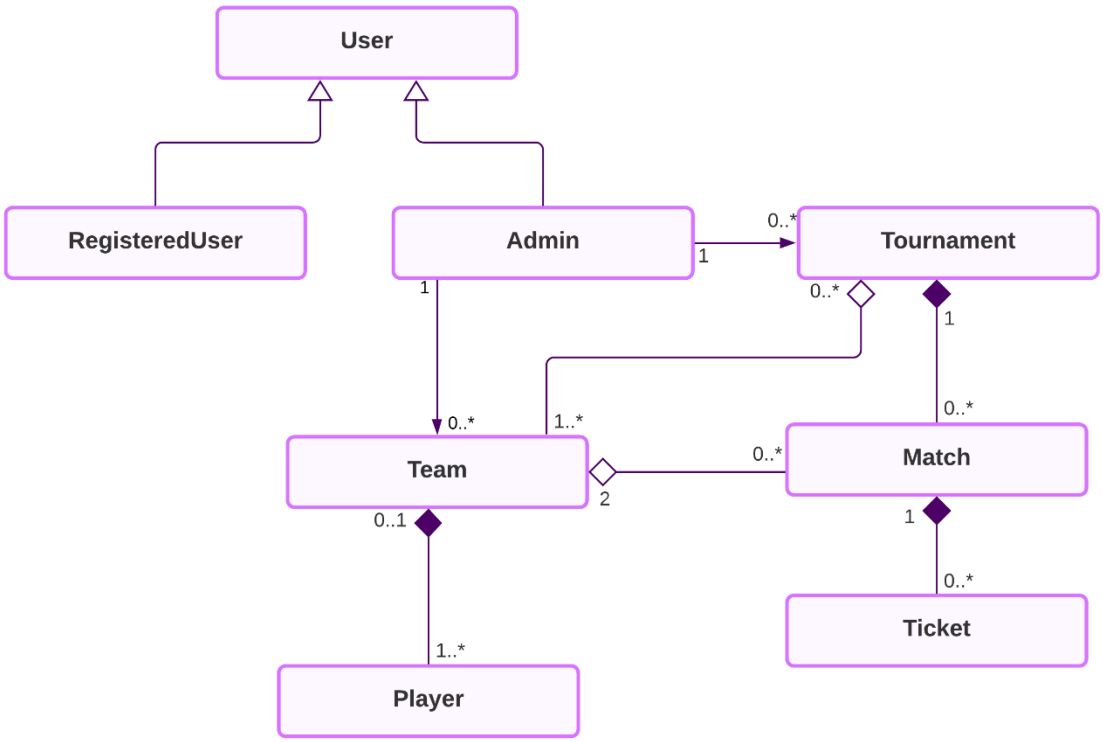


Figure 8: Object Model

1.3. ***Dynamic model*:**

Presented by the state diagram, describing the behavior of the system in a finite number of states that the system goes through as a whole. State diagram was earlier discussed and elaborated in section 8, figure 5.

1.4. ***Functional model:***

This model describes the whole processes and actions using (DFD). It focuses on how data is flowing, where data is stored (data stores) and how it is processed. DFD starts with a context level before iterating to increase the details level from level 0 until reaching to a point where sufficient details are elaborated.

1.4.1. Context diagram

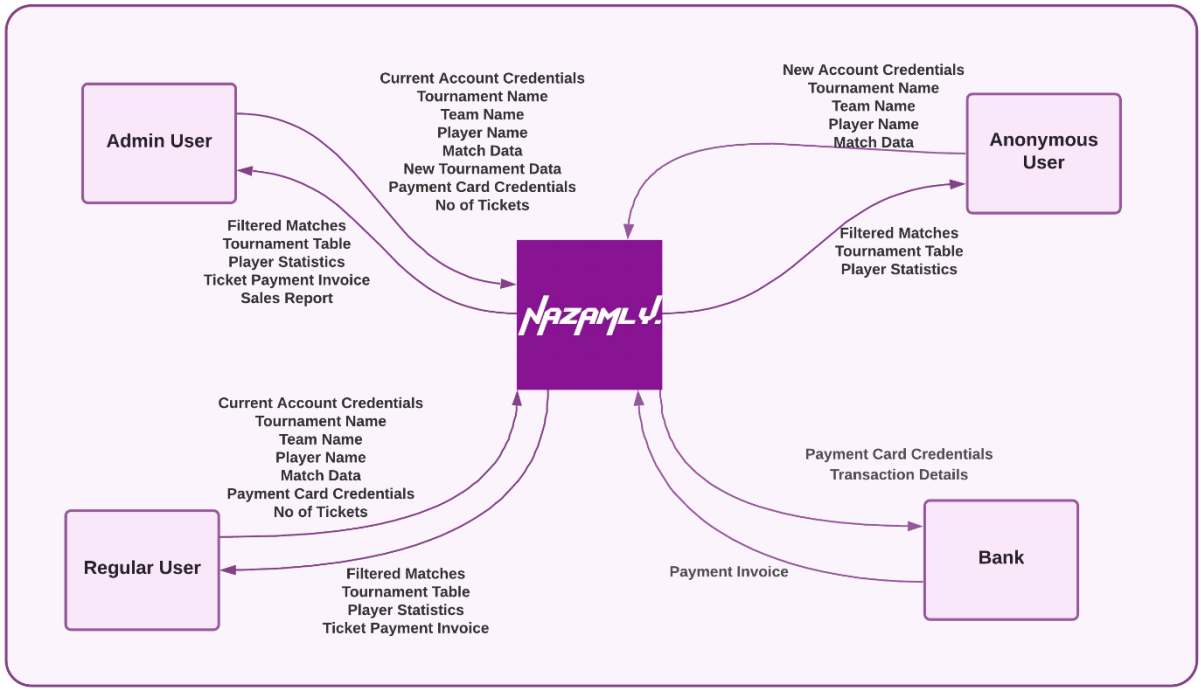


Figure 9: Context diagram

1.4.2. Level 0 DFD

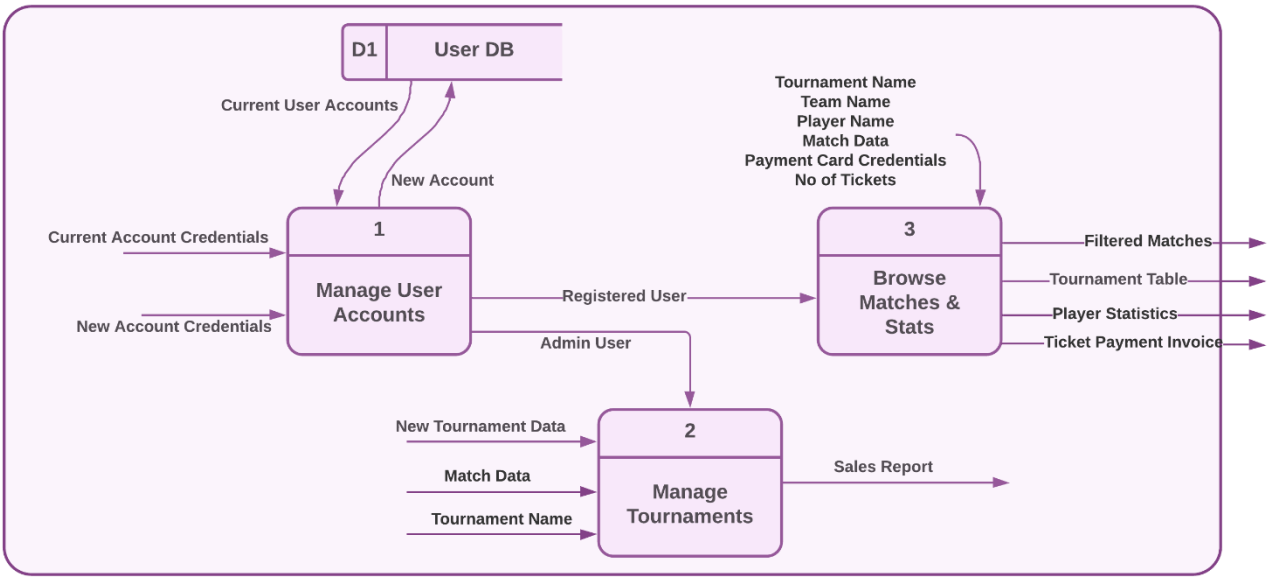


Figure 10: DFD Level 0

1.4.3. Level 1 DFD

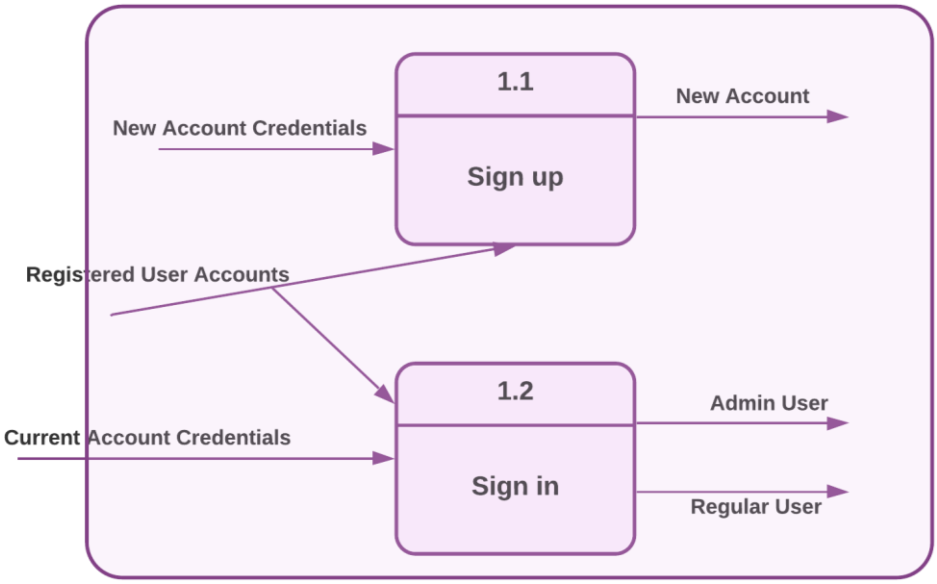


Figure 13: Process 1 Level 1 DFD

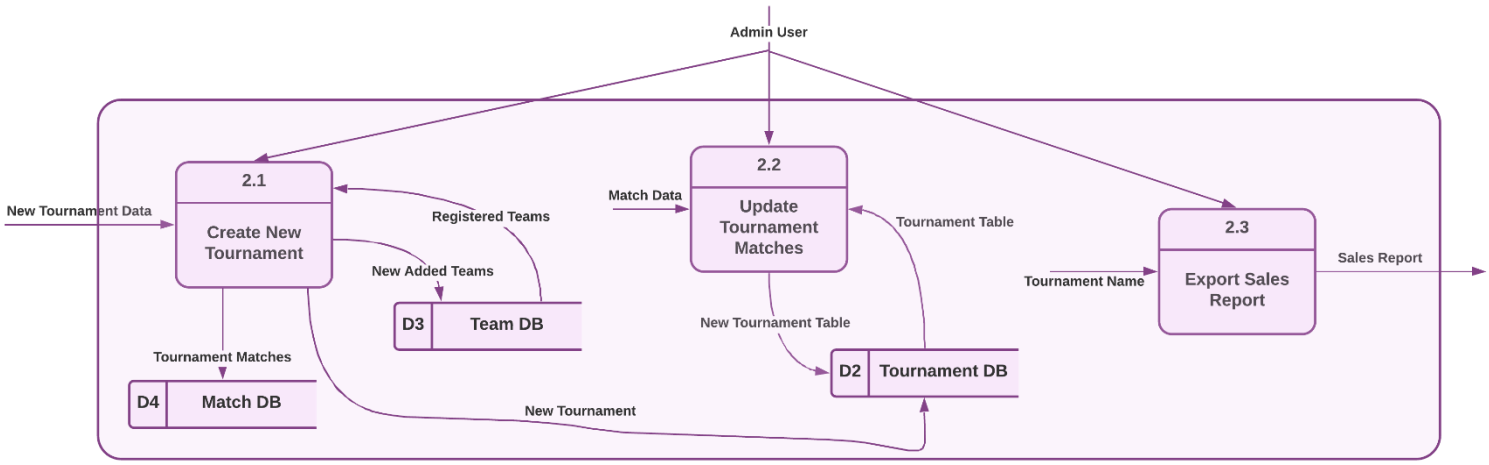


Figure 12: Process 2 Level 1 DFD

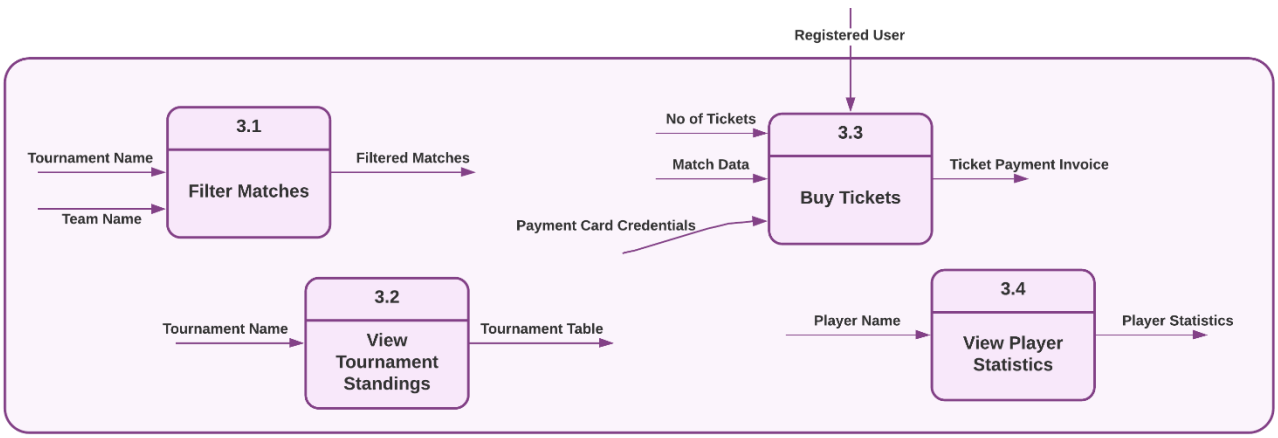


Figure 11: Process 3 Level 1 DFD

1.4.4. Level 2 DFD

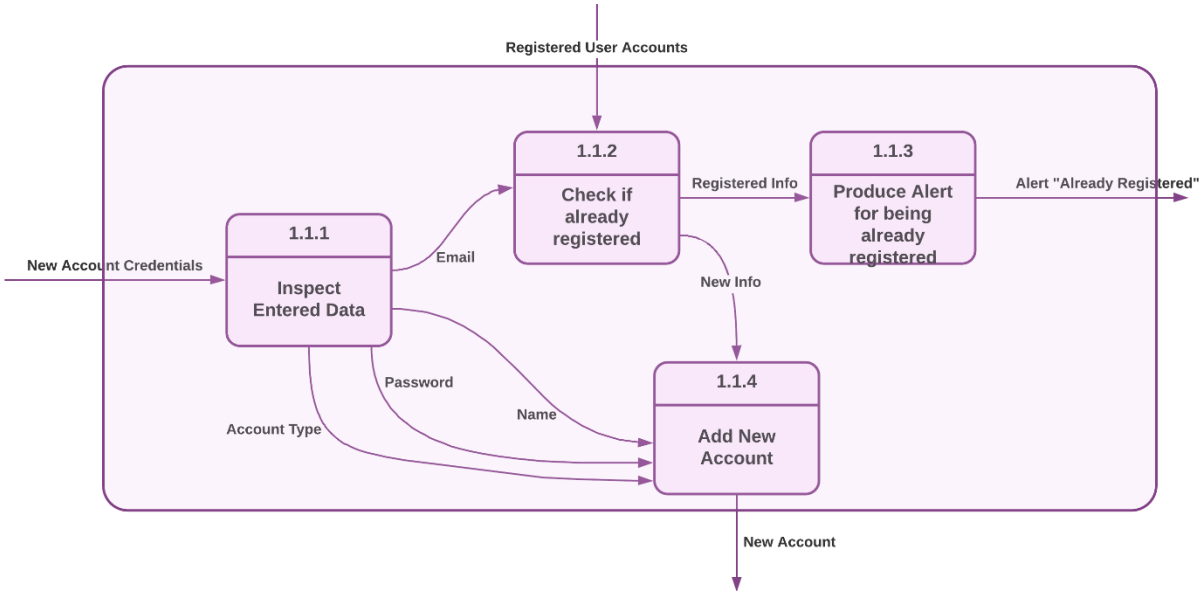


Figure 14: Process 1.1 Level 2 DFD

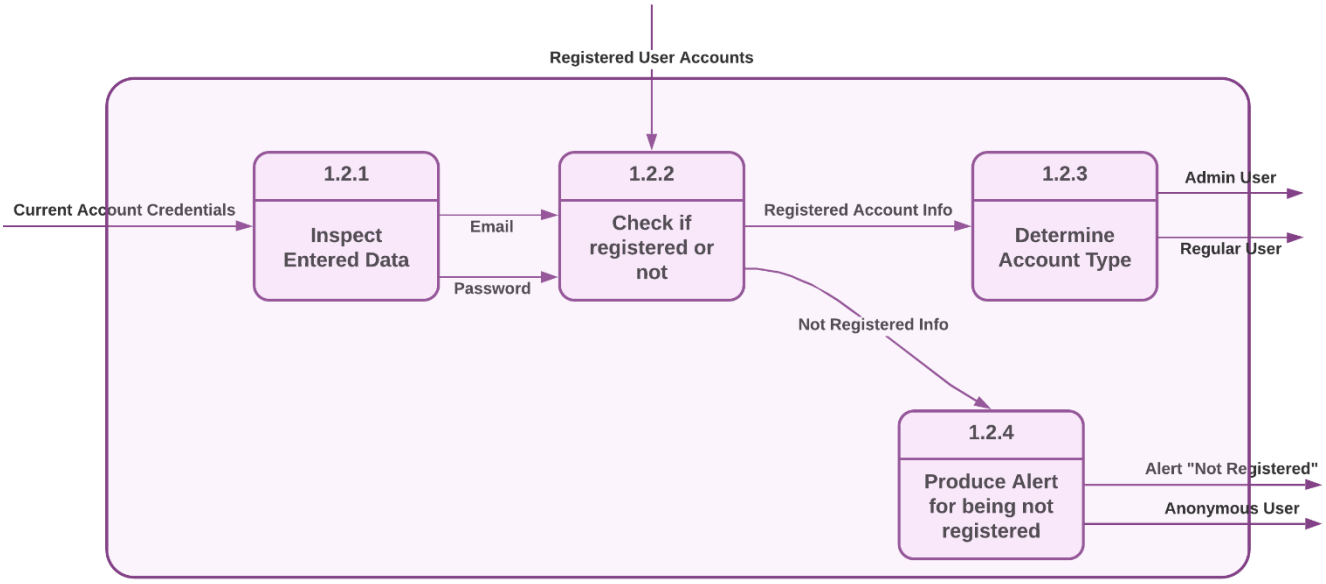


Figure 15: Process 1.2 Level 2 DFD

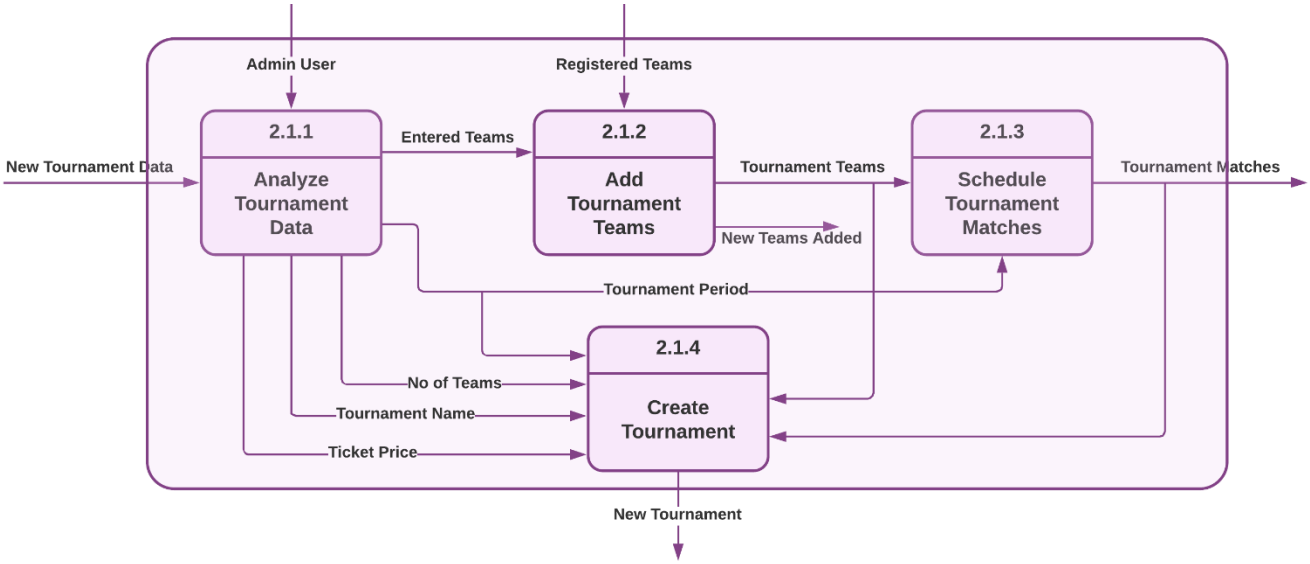


Figure 18: Process 2.1 Level 2 DFD

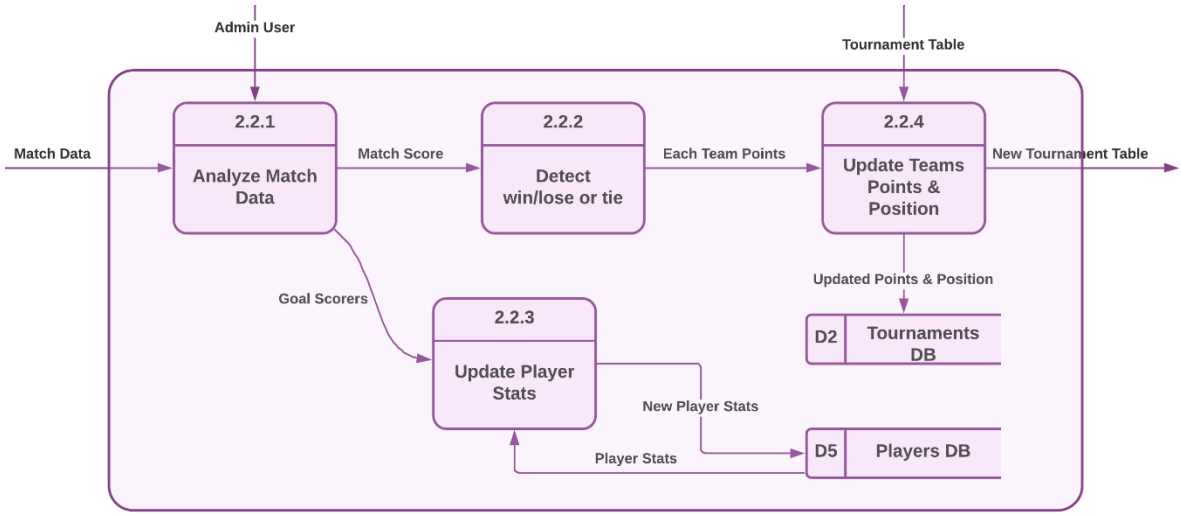


Figure 16: Process 2.2 Level 2 DFD

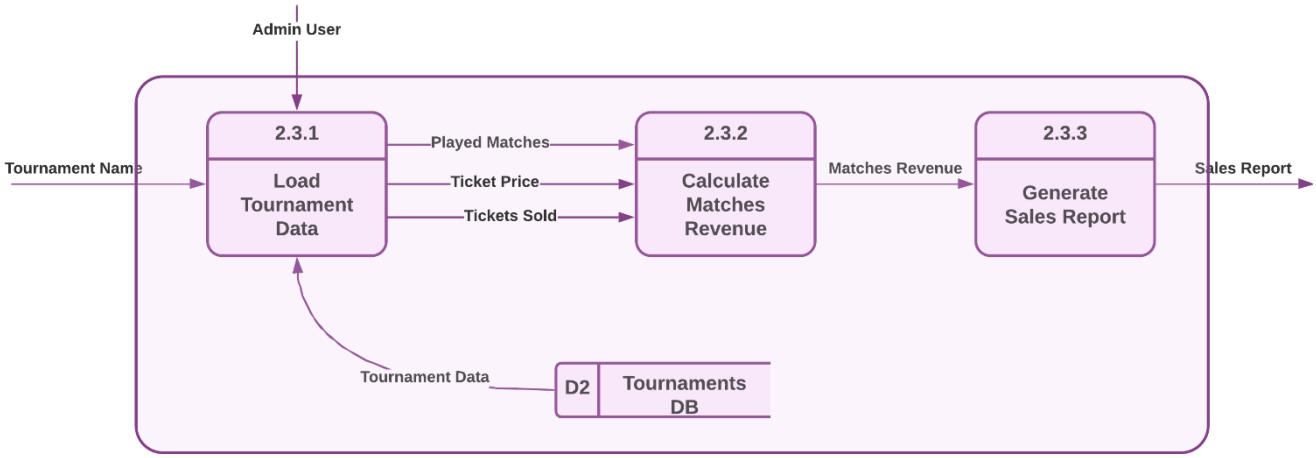


Figure 17: Process 2.3 Level 2 DFD

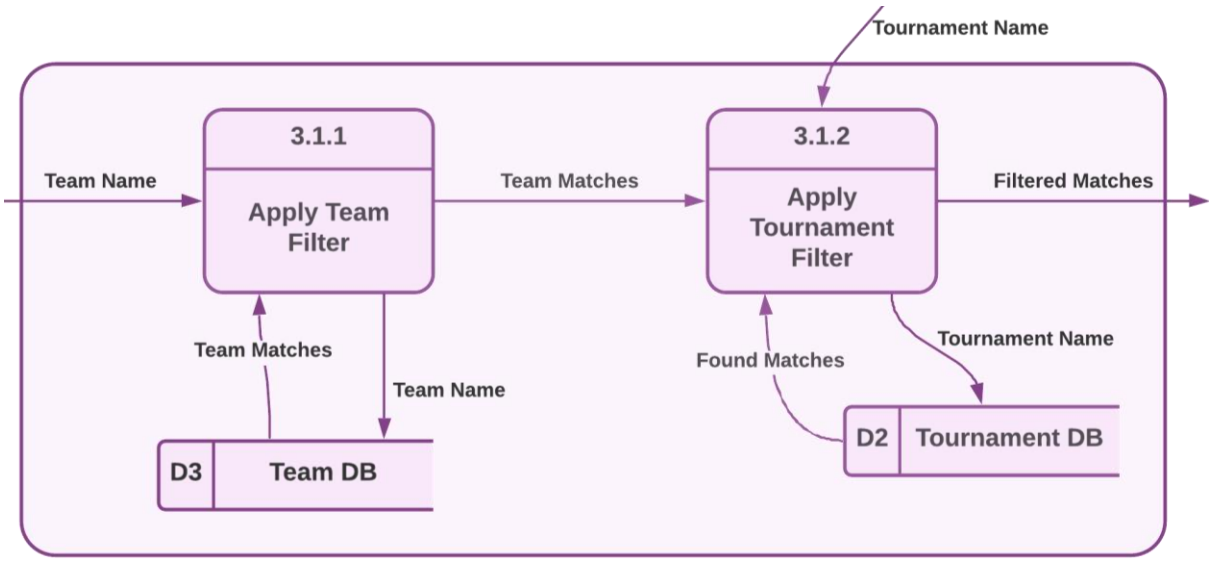


Figure 21: Process 3.1 Level 2 DFD

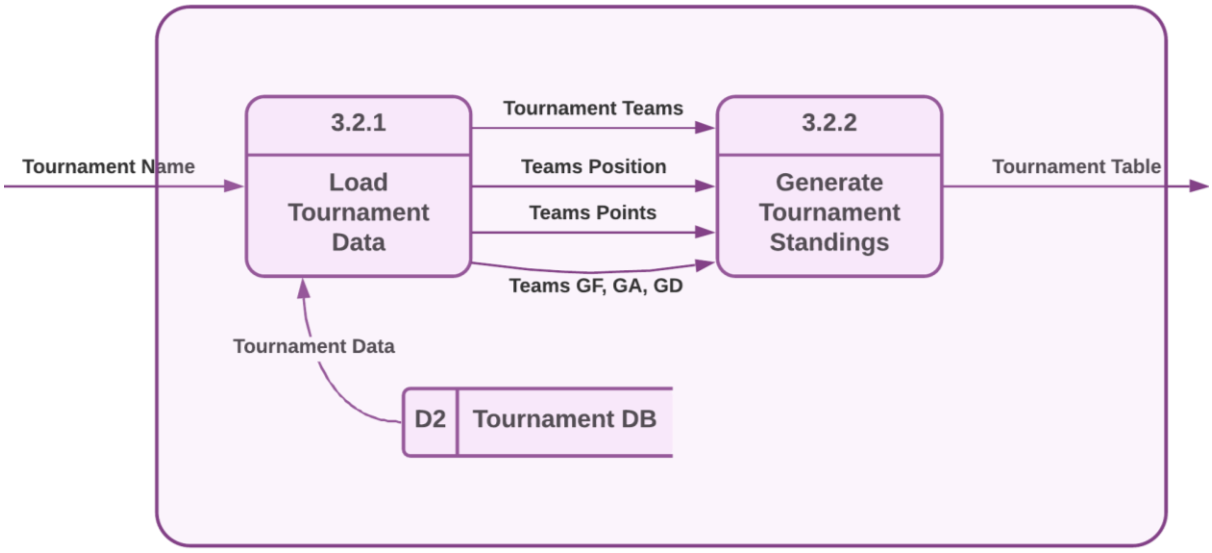


Figure 20: Process 3.2 Level 2 DFD

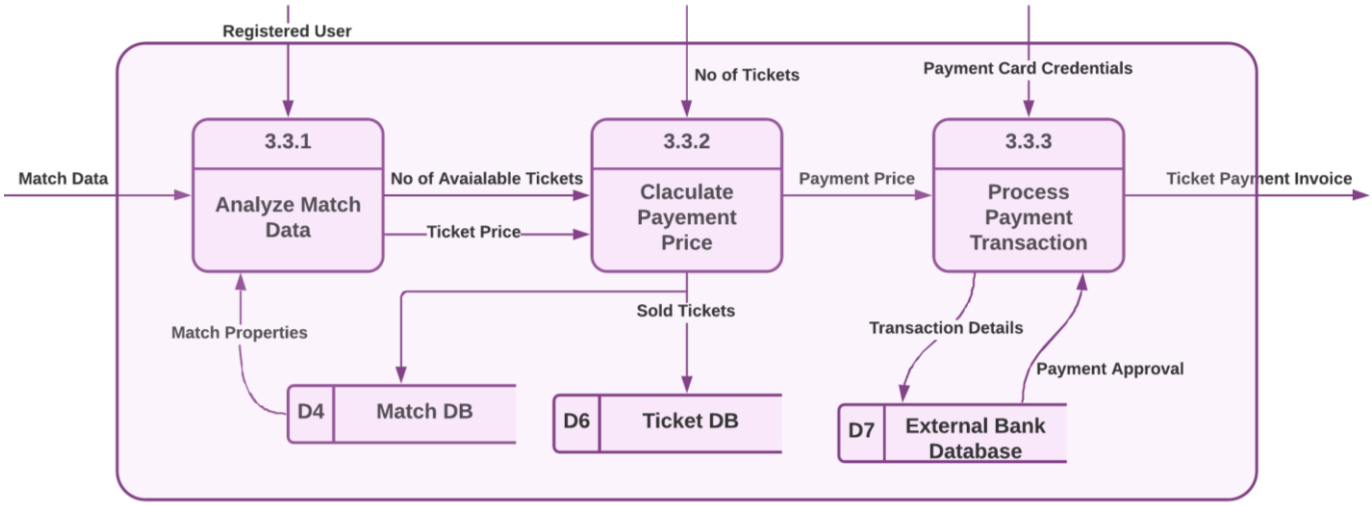


Figure 19: Process 3.3 Level 2 DFD

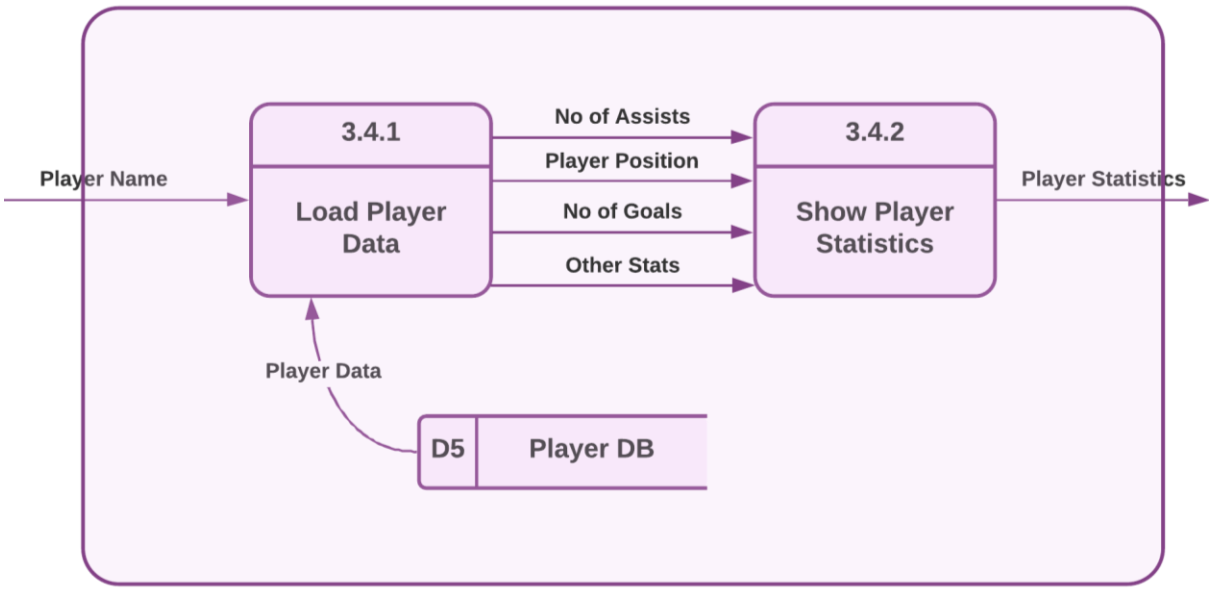


Figure 22: Process 3.4 Level 2 DFD

2. ***System design*.**

System design determines all system architecture and data storage. High level architecture of the system is designed during this phase that decomposes the system into a set of sub-systems and/or packages. This process is thoroughly described, discussed, and diagramed in section 14, later in this document.

3. ***Object design*.**

This phase produces a design document, consisting of detailed objects and dynamic and functional models. It is concerned with classification of objects into different classes and about attributes and necessary operations needed. Class diagram was provided earlier in section 10 of this document.

4. ***Implementation*.**

During this phase, the design is translated into the software. Project full source code is attached with the document.

12.2. Booch methodology

This method only covers the analysis and design phases of object-oriented software systems development, and it can be divided into 2 main processes:

1- **A macro development process** treating the OOAD in a more abstract look, focusing on the big picture, and consisting of the following steps:
   1.1. Conceptualization
   1.2. Analysis and development of the model
   1.3. Design the system architecture.
   1.4. Evolution
   1.5. Maintenance

2- **A micro development process** that dives deeper into the classes' details generating a refined version of the class diagram after each of the following steps:
   2.1. Identify classes and objects
   2.2. Identify class and object semantics
   2.3. Identify class and object relationships
   2.4. Identify class and object interfaces and implementation

The Booch methodology consists of the following diagrams:

- ***Class diagram***: Discussed and diagramed in section 10.
- ***Object diagram***: Mainly focuses on the lifecycle of the object. Same as class diagram but class names are underlined.
- ***State transition diagrams***: Same as state diagrams which are used to identify the methods needed, section 8.
- ***Module diagrams***: Same as component diagram, section 15, grouping each set of classes that interact together to reach a certain goal and are named modules, components or packages.
- ***Interaction diagrams***: Sequence or collaboration diagrams, discussed in section 9. After identifying methods in the state diagram. They help in setting a specific procedure of assigning each method to a specific class, where the method shall be a part of the class at the arrowhead in the interaction diagram.

13. Comparative Analysis

13.1. Object Modeling Technique (OMT)

OMT was one of the earliest methods adopted for object-oriented analysis and design of software systems, but it had many cons beside its pros. From its main disadvantages is:

- Objects identified in the analysis phase can't be verified as the objects' definition is highly subjective to the analyst problem understanding, opinions, and experience.
- Lack of step-by-step procedure to identify objects increases the error possibility.
- Many iterations required to reach an optimum analysis and design to the system.
- Considering a forth step for implementation was not necessary as OOAD only focuses on the analysis and design before the software implementation in terms of code.
- Data flow diagrams (DFD) used are not a UML standardized diagram, which make it inconvenient to use.

13.2. The Booch methodology

The Booch methodology avoided many of the criticized specifications of OMT where:

- It removed the implementation phase as it mainly focuses on the analysis and design phases only.
- This methodology also helps in filling the gaps found in the OMT, where Booch differentiated between classes and objects.
- Unlike OMT which leaves the methods classification to the analyst judge, interaction diagrams adopted by Booch help in setting a specific procedure of assigning each method to a specific class, where the method shall be a part of the class at the arrow head in the interaction diagram.

14. ARCHITECTURAL MODEL

Building a software system needs a study of several architecture styles to wisely choose the most suitable style that correctly defines the system's structure, which is most commonly a merged style of more than one style.

Therefore, designing Nazamly was no different as it is built of a merged style that is a combination of 3 different well-known styles, MVC, data-centered, and object-oriented architecture styles, which will be discussed in the following sections.

14.1. Object-Oriented Architecture

In this style, the system components encapsulate data and only allow manipulating them through methods in units called objects, as shown in the UML class diagram (provided earlier in section 10) (Figure 3), where communication between them is accomplished via message passing.

14.2. Blackboard (Data-centered) Architecture

Using this architecture style allows the system to have a datastore, a Firebase database in our situation, that resides at the center of this architecture, and it is accessed frequently by the system users to add, modify, update, and delete data through database services.

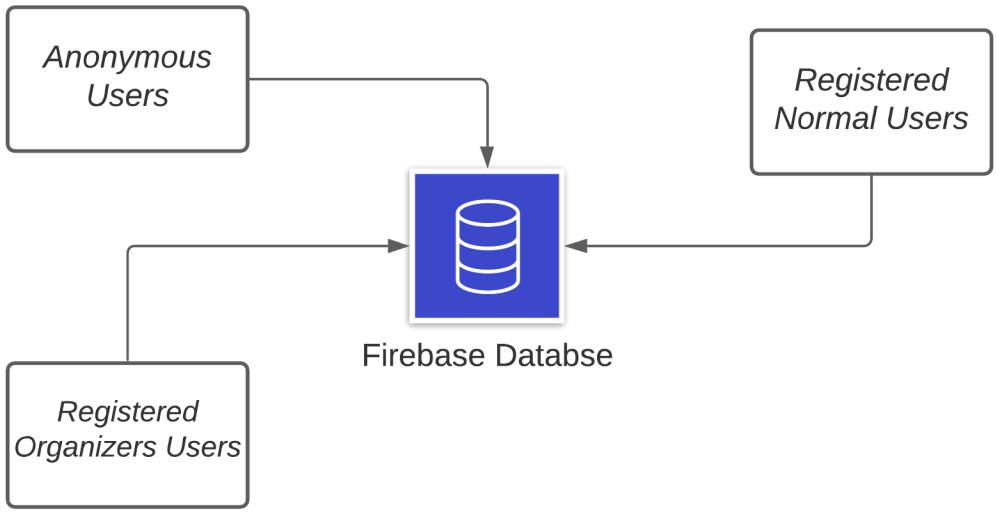


Figure 23: Blackboard Architecture

14.3. Model-View-Controller Architecture

Known as MVC, model-view-controller architecture is a widely used one in web applications, consists of 3 main parts:

- *Model*: includes the data and its processing.
- *View*: includes representing data to the end-user as the graphical user interface of the webapp, provided sample screenshots on section 19 (User Guide), later in this document.
- *Controller*: includes input handling, business logic in addition to coordinating between the model and view components.

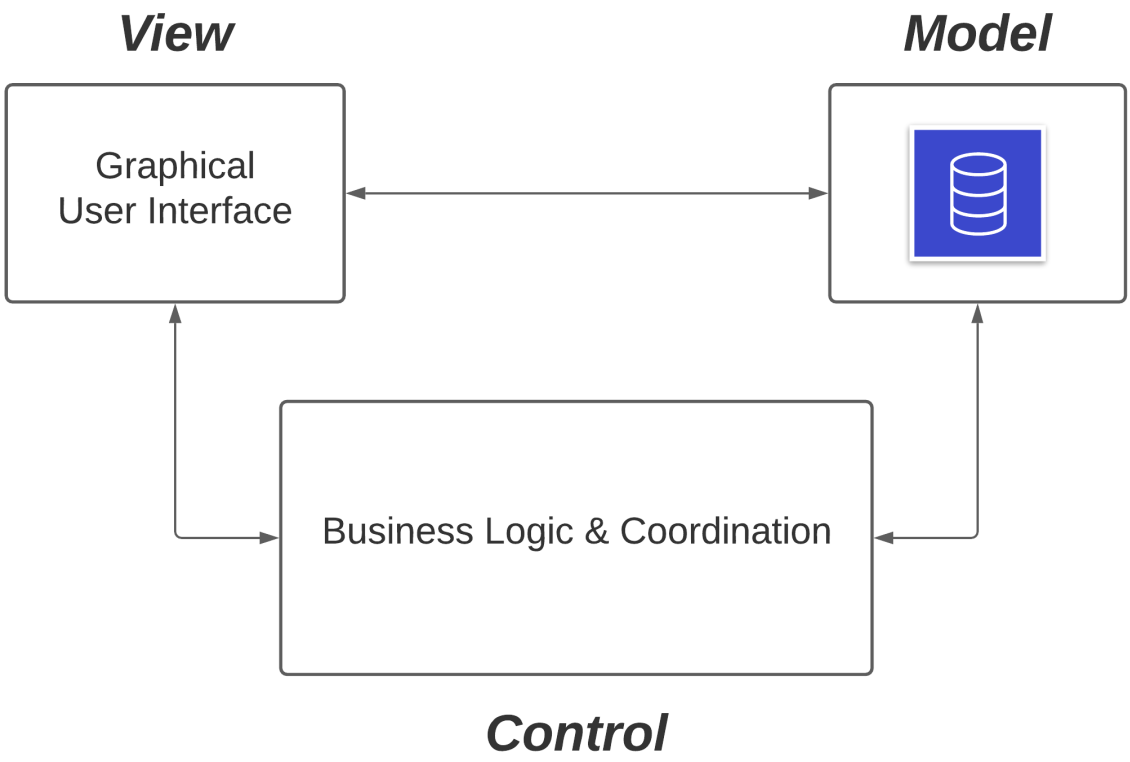


Figure 24: MVC Architecture

14.4. Merged Architecture Style

As mentioned before, building Nazamly requires a merged architecture style consisting of three of the discussed architectures. The MVC architecture is the framework of the design with the Control part utilizing the object-oriented architecture style, and the Model part making use of the data-centered style. The merged style designed by the Nazamly team is shown in the figure below.

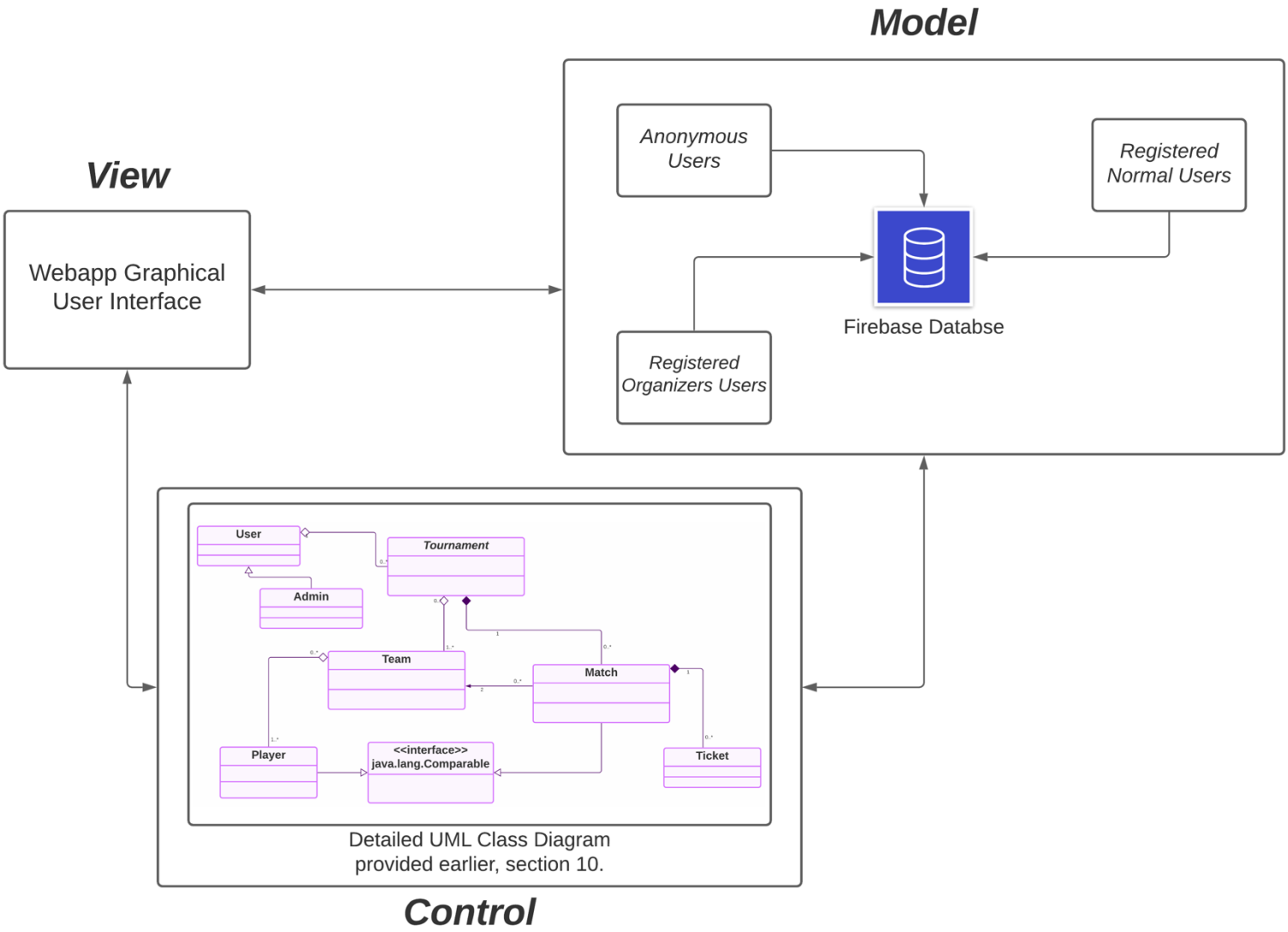


Figure 25: Merged Architecture Style

15. Component Diagram

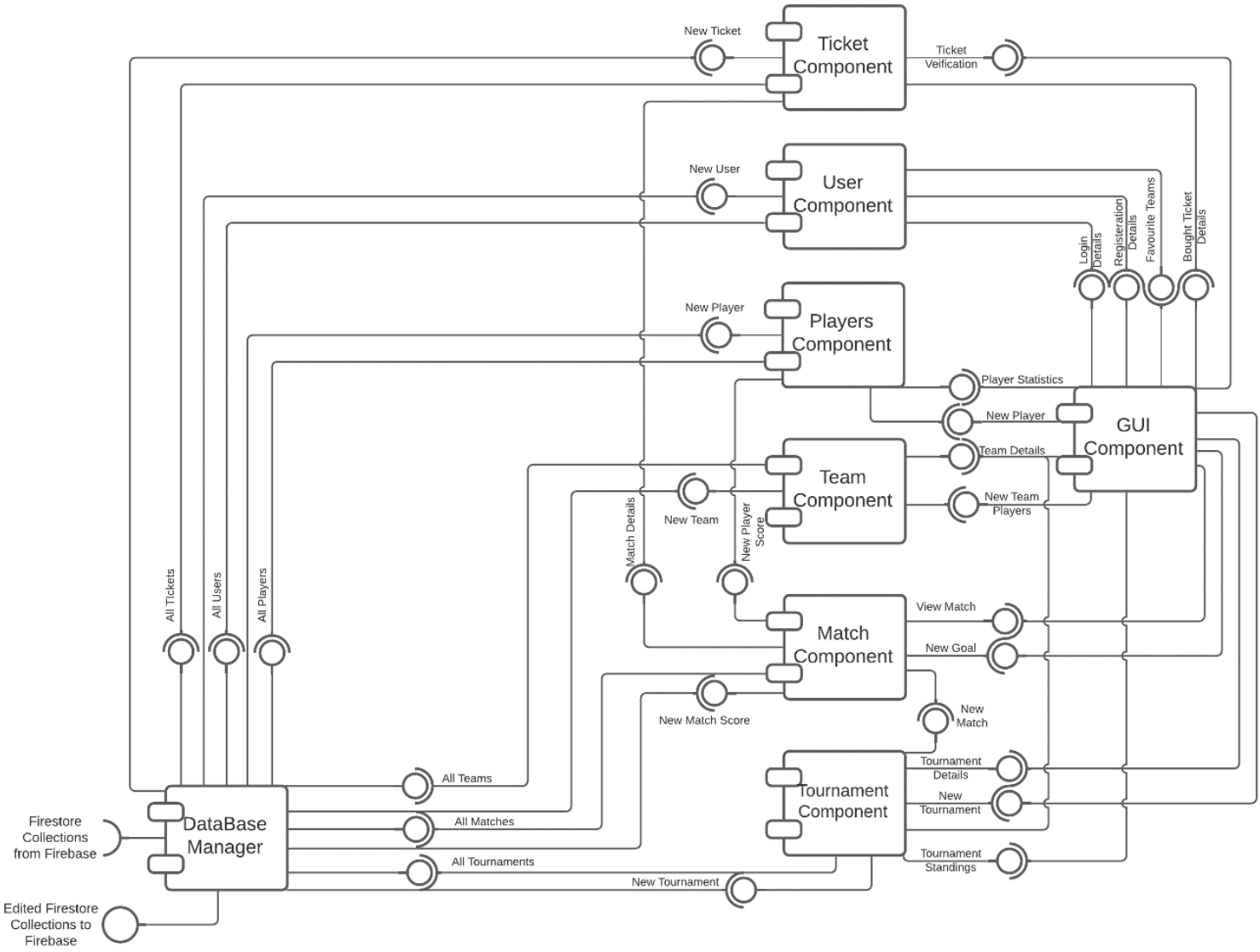


Figure 26: Component Diagram

Narrative Description:

- The **DataBase Manager** component is responsible for retrieving all our data from Firestore as Observales Collections, and then converting them to JavaScript Arrays to be provided in different interfaces to the remaining components.
- The **Tournament** component receives all stored tournaments and process it to display required tournament details to the **GUI** component. It also receives new tournament data from **GUI** component and then creates tournament matches that are sent to the **Matches** component.
- The **Matches** component also receives all stored matches and filters them in order to be viewed accordingly with the applied filters. It also receives match score and new goals from **GUI** component and sends these data to **Player** and **Team** component.
- The **User** component receives sign up data from **GUI** and provides an interface with this new account data to the **Database Manager** to be stored in our Firestore. Also, it receives user login data from **GUI** and searches all registered user data that are received from **Database Manager**.
- The **Ticket** component receives any requested ticket from **GUI**, and provides an interface with the new ticket data to the **Database Manager**. It also retrieves all tickets data from **Database Manager**, and allows a tournament admin to verify if a certain ticket id is valid or not.

16. Testing

16.1. *Unit Testing*

As mentioned earlier, Nazamly web application is developed using Angular framework. And it is important to be noted that, in Angular, the application structure is mainly divided into modules that are composed of smaller related components. An angular component actually represents a single web page in the application and hence comprises a TypeScript (.ts) file, HTML (.html) file, and CSS (.scss) file for this single component.

In addition, the system classes, represented earlier in the class diagram, are generally implemented into two files: a model and a service. The class model (*.model.ts*) actually comprises the class data members, while the service (*.service.ts*) provides all the required functionalities related to this class. For example, the ***Match*** class, shown before in the class diagram, will be implemented in two files: *match.model.ts* and *match.service.ts*, as shown in the below figure.

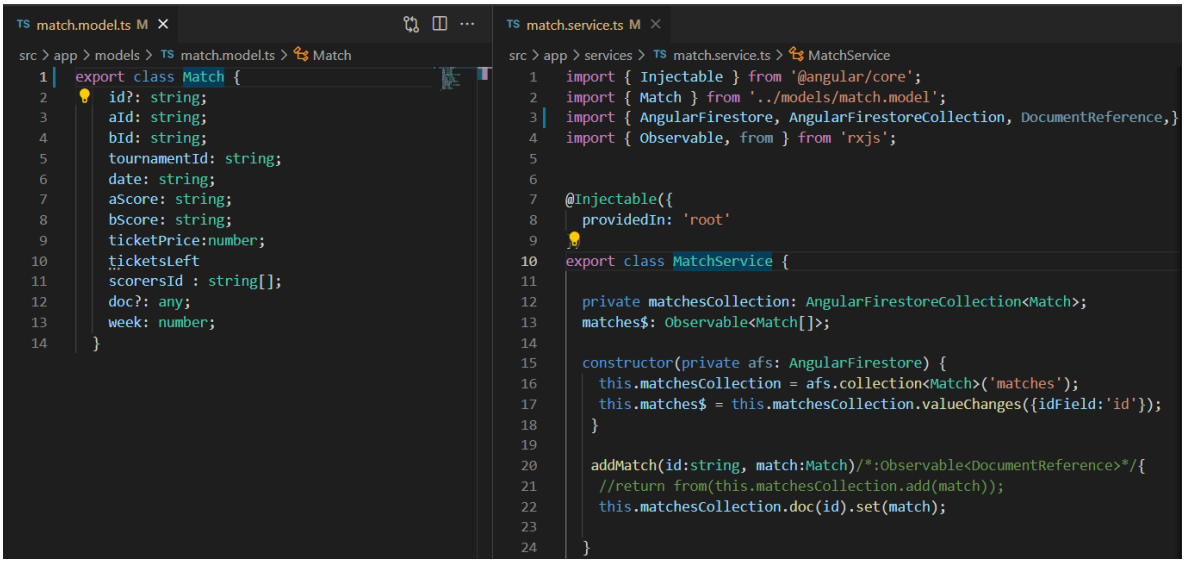


Figure 27: Model and Service for Match class

Unit testing was applied after the implementation of each component, where the code is compiled first assuring there are no syntax errors or runtime exceptions, and then the whole application is served internally on a local host (http://localhost:4200) in order to test the web page that this recently-created component represents.

If there are any errors, exceptions, or visual mis-implementation in the page formatting; the code is modified, compiled, and served again until the desired implementation is achieved.

16.2. Integration Testing

This phase of testing begins after applying unit testing on each of the system components, where each component is added and integrated with the already developed part of the system and then tested as a whole. Integration testing is conducted to evaluate the compliance of this component with specified functional requirements, so all the functionalities that are shared between it and any other components or modules are tested, and all the external provided services that it may depend on are also tested if they are integrating well together.

For instance, after implementing and unit-testing the ***create-tournament*** component (that models the page for creating a tournament), integration testing was applied to test the proper integration of this component with the system. This was done by manually creating a tournament from the ***create-tournament*** page, and then testing the ***matches*** page whether it will show the matches of this newly created tournament or not.

16.3. System Testing

This phase was applied after the end of the system development, in which all the system components and modules are integrated together, and the system is tested as a whole. In this phase, all the previously specified functional requirements are tested using a set of test cases which are shown in detail in the next section.

16.4. Test Cases

In order to test all the functionalities that Nazamly application provides, a large set of test cases is applied to cover a wide range of possibilities that may happen during using the system. These test cases are grouped below according to the functionality they test.

A detailed description for each test case is provided along with the test result showing the application response in a figure.

16.4.1. ***Sign Up***

The sign-up form includes 3 fields for Username, Email, and Password, along with a checkbox for the user to select whether he is a tournament organizer or not.

These fields have some restrictions: the username and email should not be registered previously, the email also should be a valid one, formatted as "name@example.com", while the password should have minimum length of 6 characters. The password typically can be any mix of lowercase or uppercase alphabet, numbers, or special characters.

Below are all the test cases performed on the sign-up form, along with the application response.

- **Entry of proper data with no violation for restrictions**

When entering proper data in the register page and clicking "Create Account" button, the account will be created and the user will be directed automatically to his profile page, as shown in below figure.

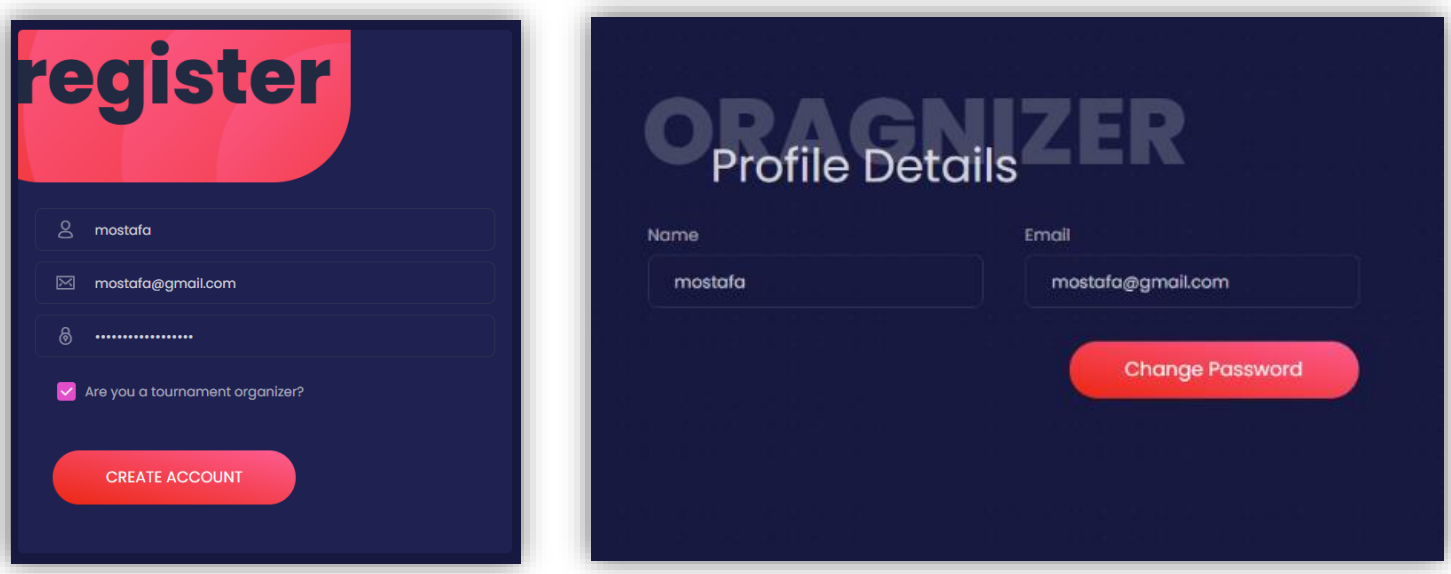


Figure 28: Sign up - Entry of proper data

- **Entry of an invalid email**

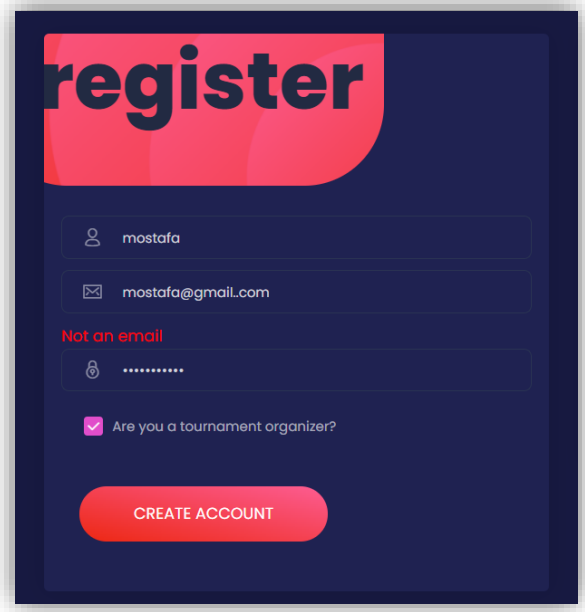


Figure 29: Sign up - Entry of an invalid email

- **Entry of a password with length <6**

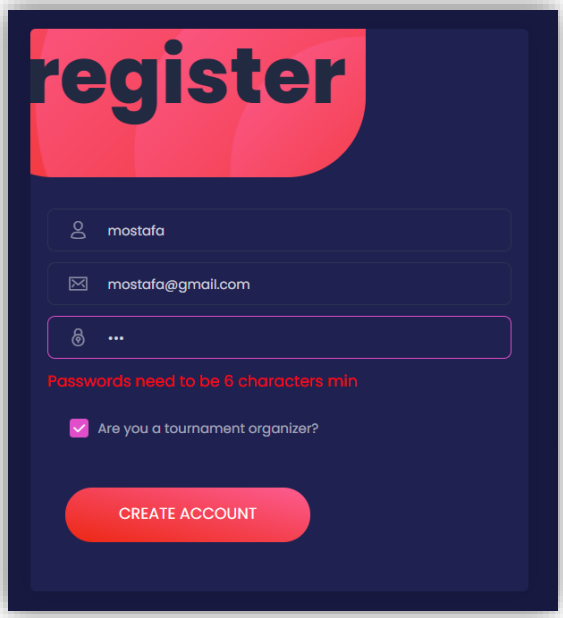


Figure 30: Sign up - Entry of a short password

- **Entry of an already registered username**

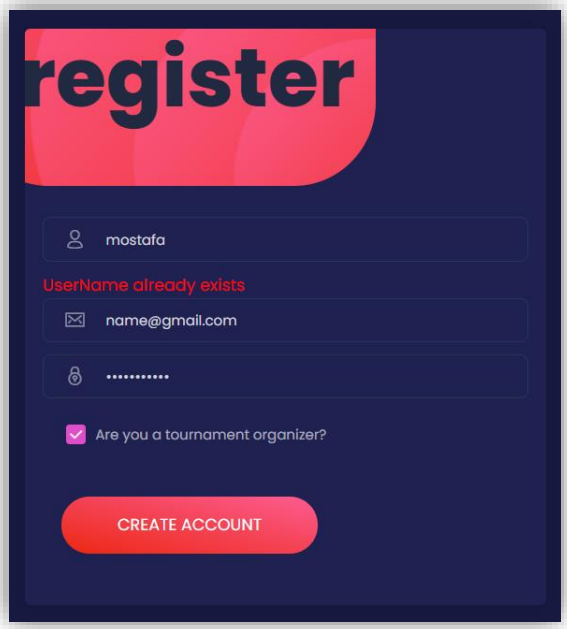


Figure 31: Sign up - Entry of a registered username

- **Entry of an already registered email**

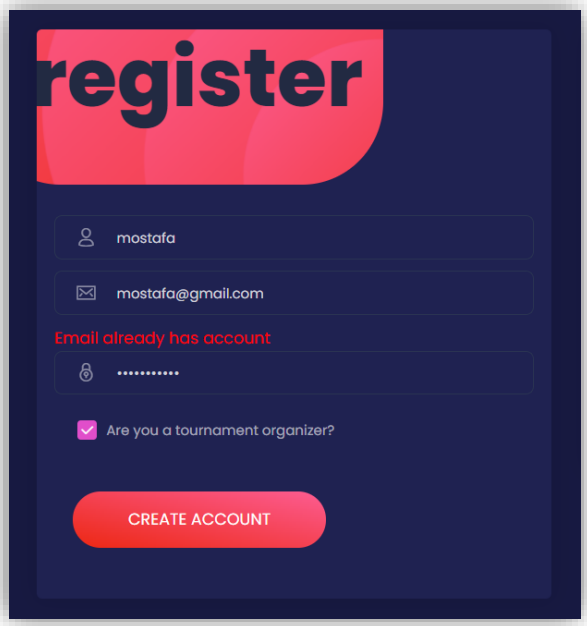


Figure 32: Sign up - Entry of a registered email

16.4.2. ***Sign In***

The sign in form simply includes email and password field***s. If a user has already registered for an account, he can enter the email-password pair that with which he registered to login to his account and enjoy the features available to him.***

The user can only log in if he entered a matching pair of email and password. Otherwise, he will be prompted with a ***message conveying the problem. The email may be invalid, the password may be short, or the email and password are not matching.***

Below are all the test cases performed on the sign in form, along with the application response.

- **Entry of a correct email and password**

Once the user enters a matching pair of email and password and clicking “Login” button, he is directed automatically to his personalized home page, as shown in these figures.

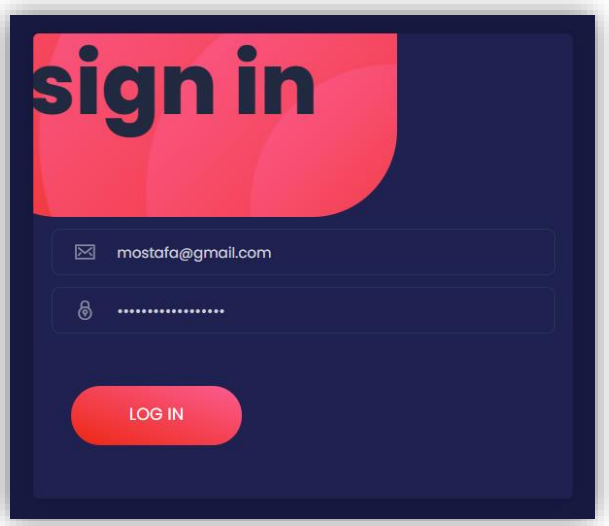


Figure 33: Sign in - Entry of correct email and password

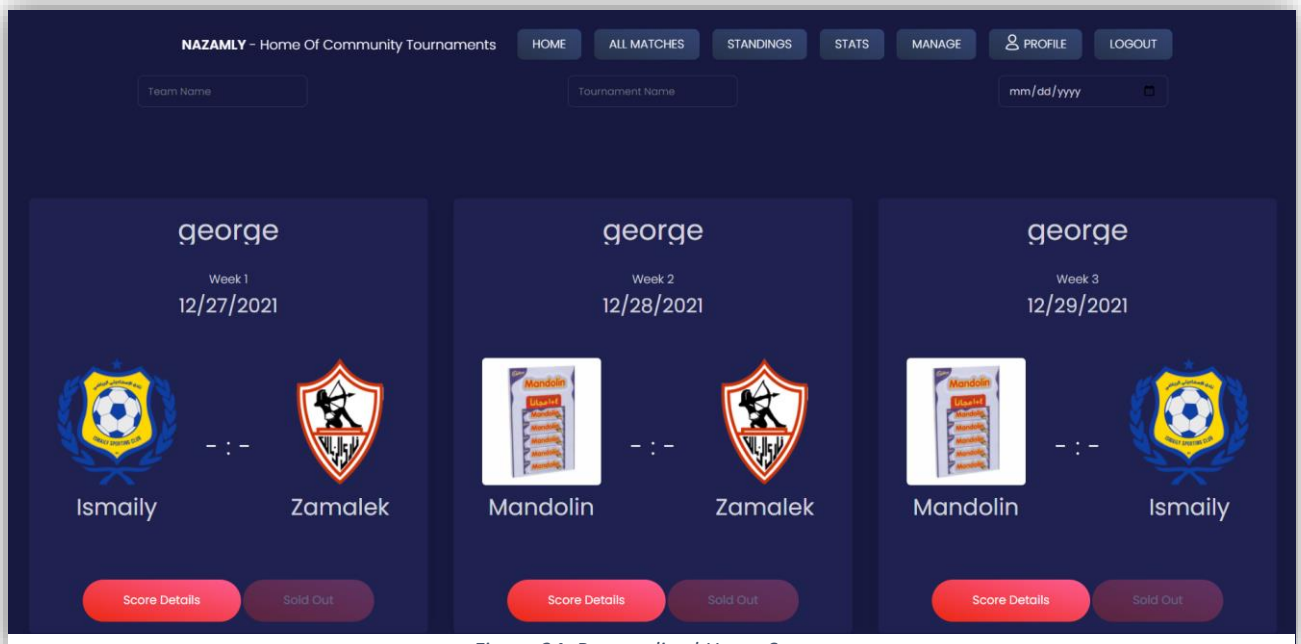


Figure 34: Personalized Home Screen

- **Entry of incorrect email**

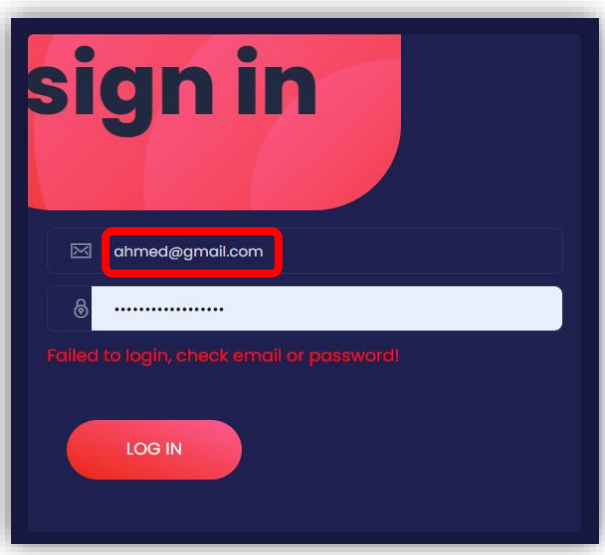


Figure 35: Sign in - Entry of incorrect email

- **Entry of incorrect password**

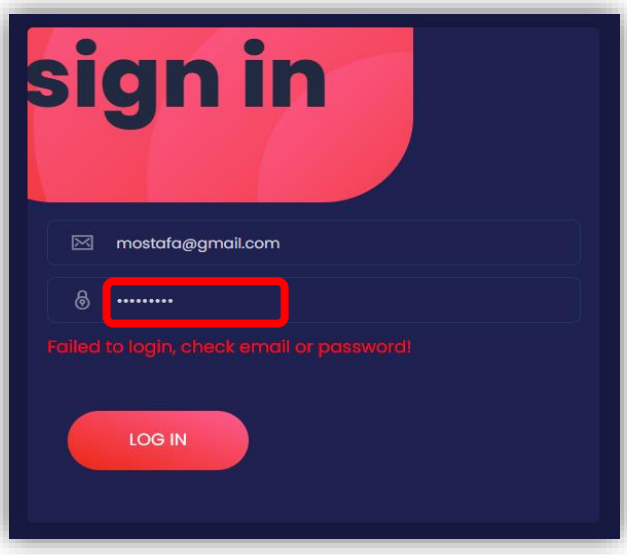


Figure 36: Sign in - Entry of incorrect password

- **Entry of an invalid email**

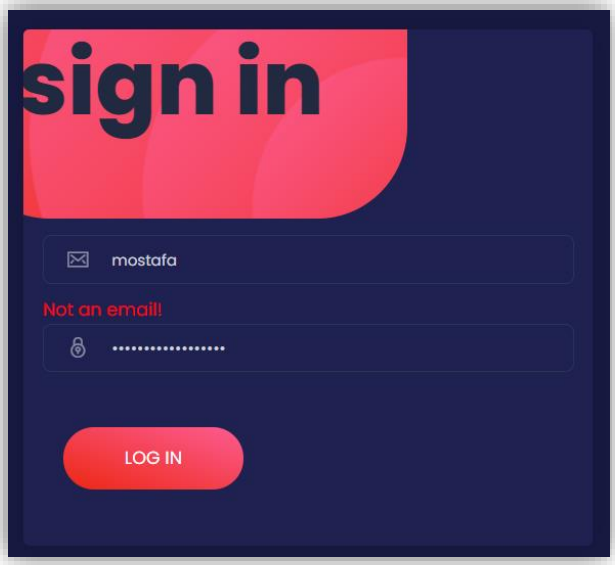


Figure 37: Sign in - Entry of invalid email

- **Entry of short password**

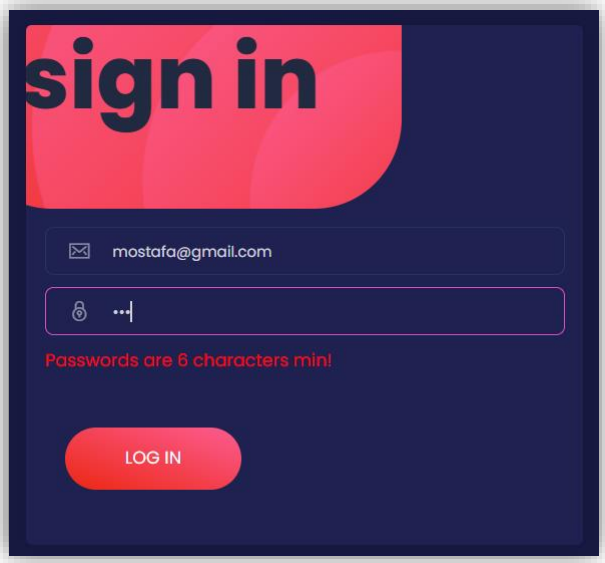


Figure 38: Sign in - Entry of short password

16.4.3. Create Tournament

Only organizing users are eligible to use the feature of **creating and managing new tournaments**. To create a new tournament, the admin enters some data for his tournament such as name, start and end date, ticket price, no. of tickets/match, and finally he adds all the tournament teams. However, there are some validations and restrictions on these input fields which are shown in the below test cases.

For instance, **the tournament name, start date, and end date fields cannot be left empty. The selected end date should - logically - come after the selected start date with at least 1 day in between. And trivially, the tournament should have at least 2 participating teams. The tournament name cannot be the same as an already existing one.**

Moreover, it is assumed that each team can play at most 1 match in a day, so the no. of days between the start and end dates should at least equal the no. of tournament rounds or more, i.e., a tournament with n teams should be at least n days long.

All these test cases are shown below along with the application response to each of them.

- **Adding a name of an already existing tournament**

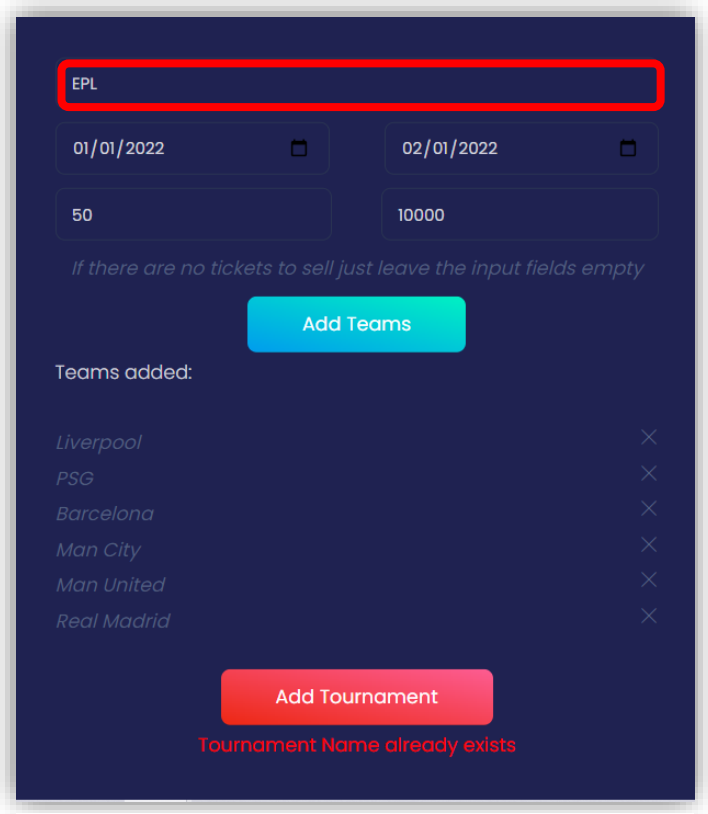


Figure 39: Create Tournament - Adding an existing tournament name

- ***Leaving name or date fields empty***

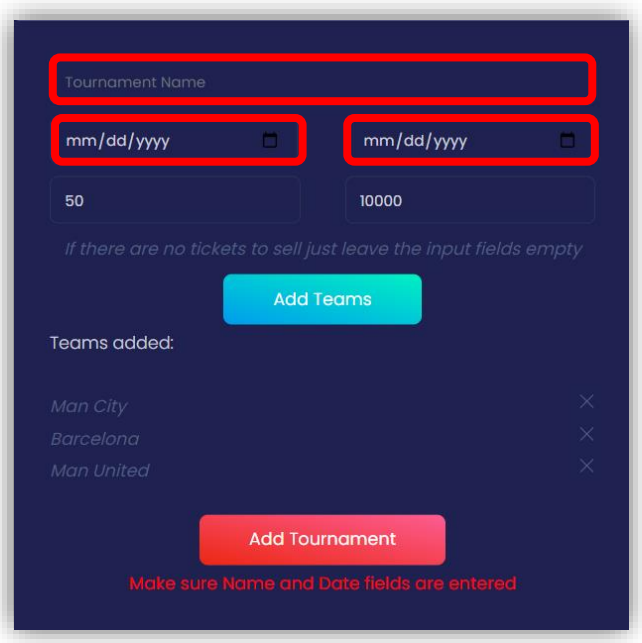


Figure 40: Create Tournament - Leaving fields empty

- ***Selecting end date that go before start date***

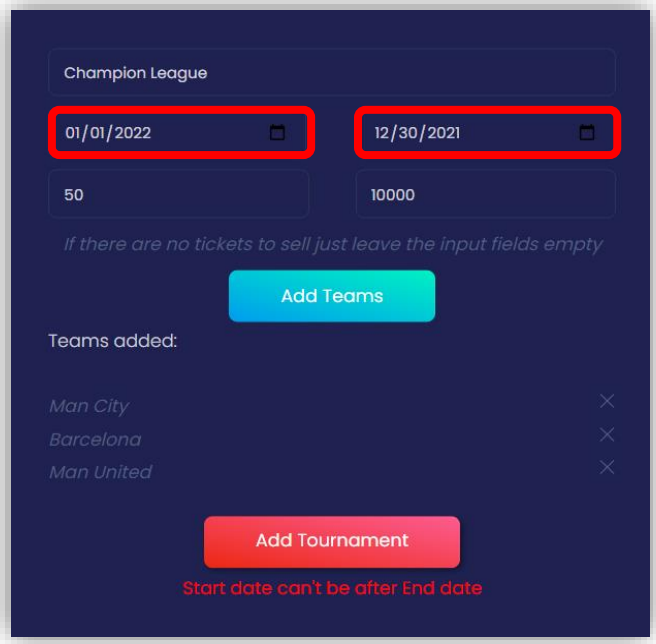


Figure 41: Create Tournament - End date is prior start date

- ***Adding no or only 1 team to the tournament***

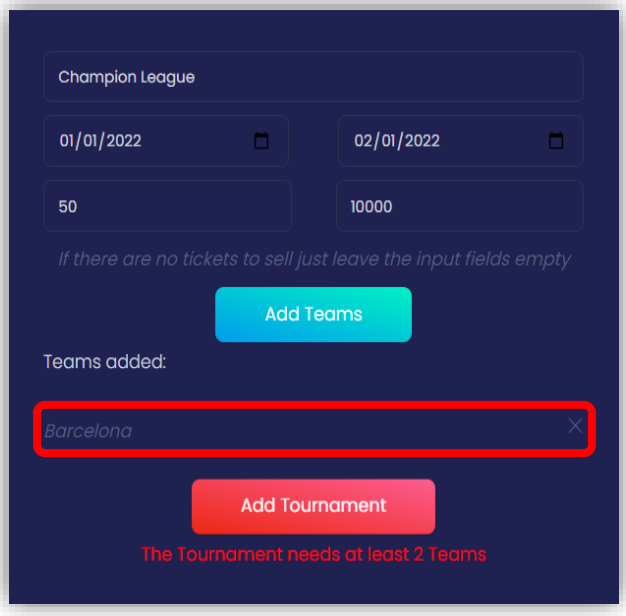


Figure 42: Create Tournament - Adding no or only 1 team

- ***Providing a short tournament period (no. of days < no. of teams)***

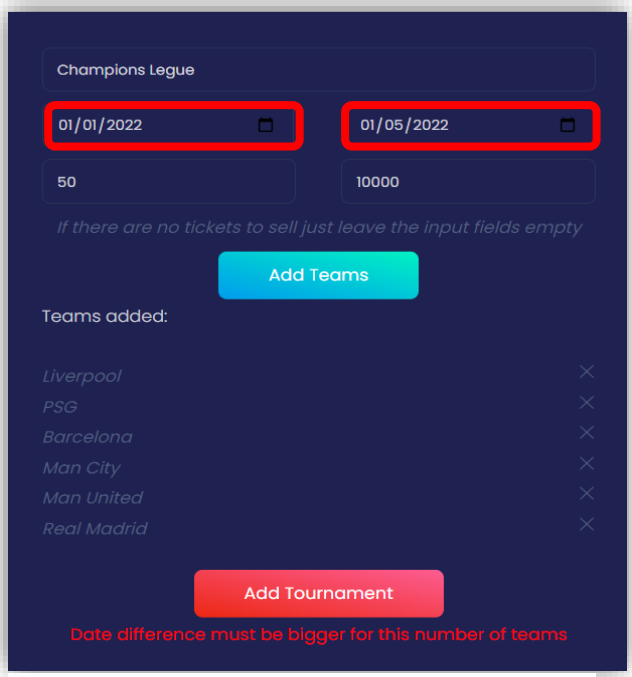


Figure 43: Create Tournament - Short tournament period

Typically, the organizer can add any no. of teams to his tournament. The algorithm used to schedule the matches between the tournament teams assumes that the no. of team is even, and equal no of matches are held in each round. Therefore, a tournament with even (N) teams will have (N-1) rounds and (N-1)*(N/2) matches.

However, if the user added odd no. of teams to the tournament, the application adjusts the algorithm slightly in order to for each team to have played the same no. of matches at the end of tournament. In this case, one team will be left out and don't play a match in each round. Therefore, a tournament with odd (N) teams will have (N) rounds and (N/2)*(N-1) matches.

Two more test cases for these 2 possibilities are given below.

- **Creating tournament with EVEN no. of teams**

  In this test case, a tournament with 6 teams is created as shown below. For all teams to face each other, the tournament should have 5 rounds and a total of 15 matches which are indeed created as shown in the below figure.

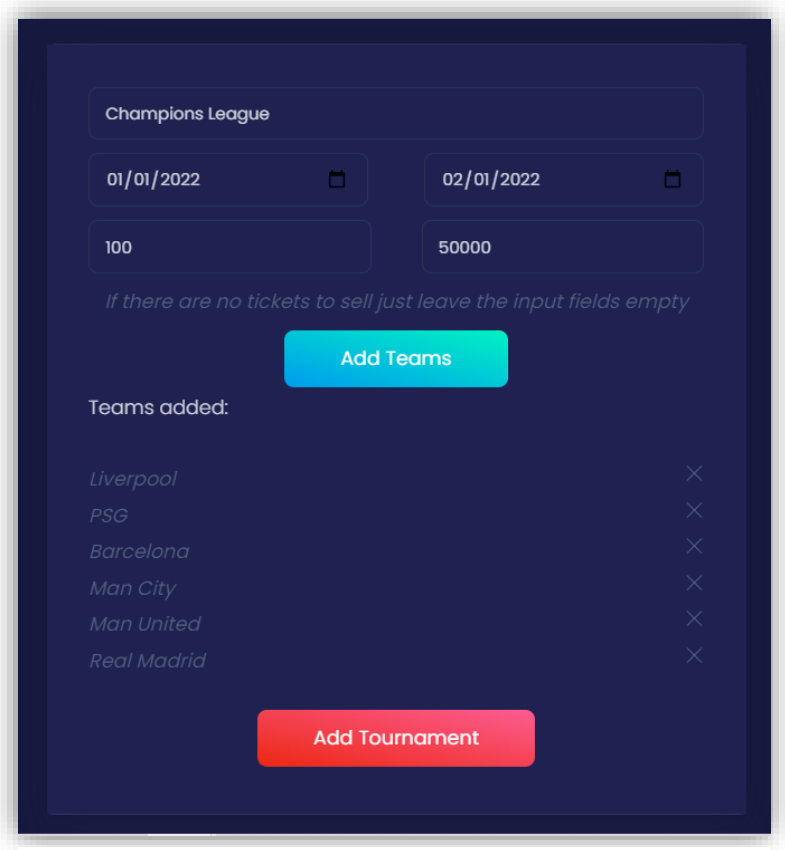


Figure 44: Create tournament - EVEN no. of teams

These are the 15 matches created for the EVEN tournament:

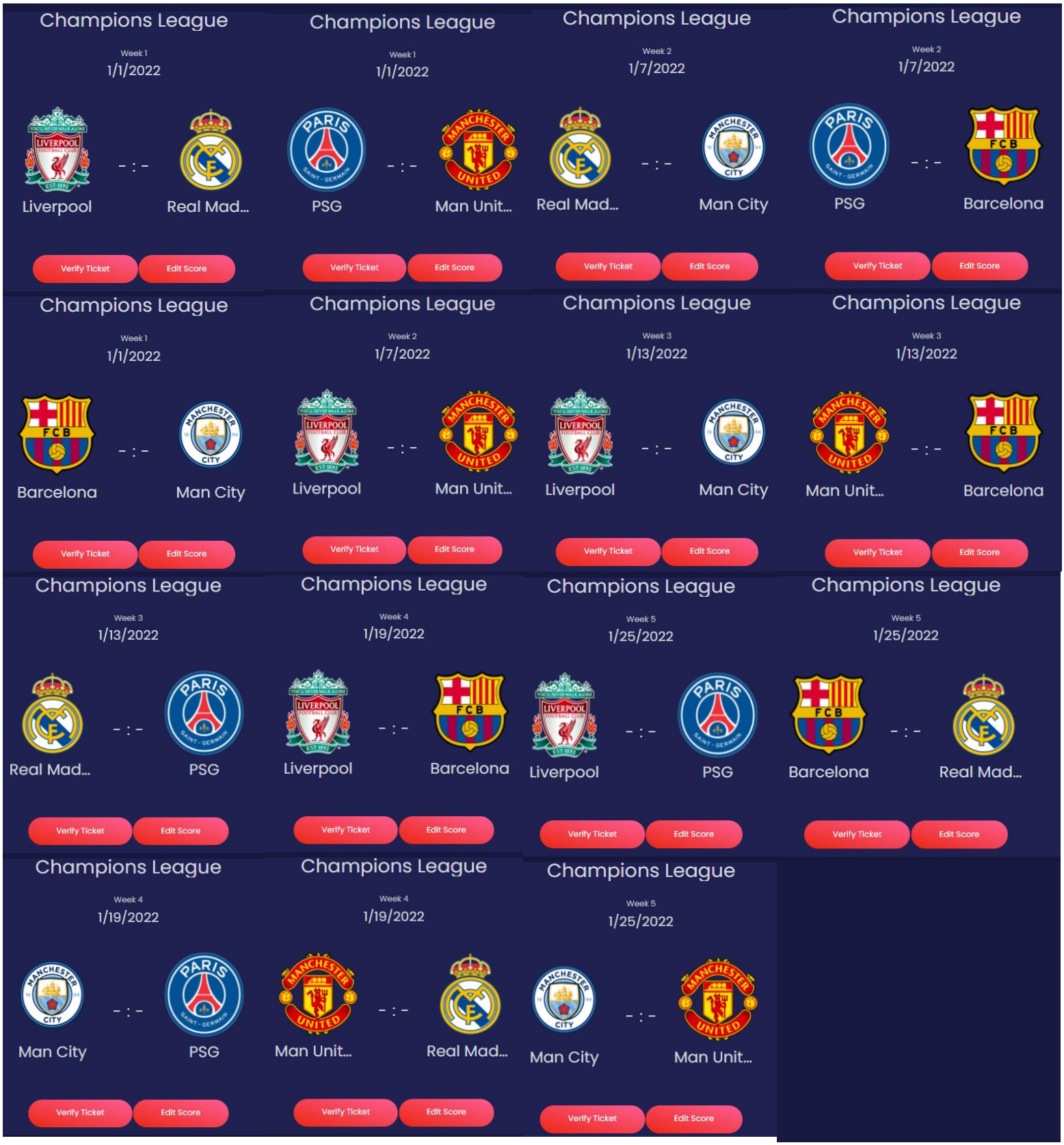


Figure 45: Created matches for EVEN no. of teams

- **Creating tournament with ODD no. of teams**

  In this test case, a tournament with 5 teams is created as shown below. For all teams to face each other, the tournament should have 5 rounds and a total of 10 matches which are indeed created as shown in the next page.

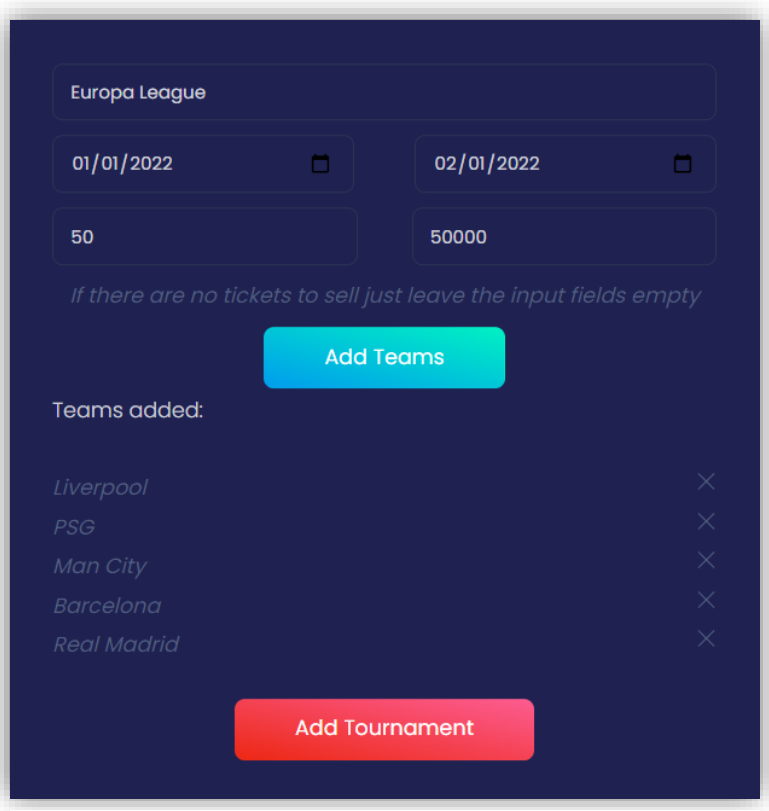


Figure 46: Create tournament - ODD no. of teams

These are the 10 matches created for the ODD tournament:

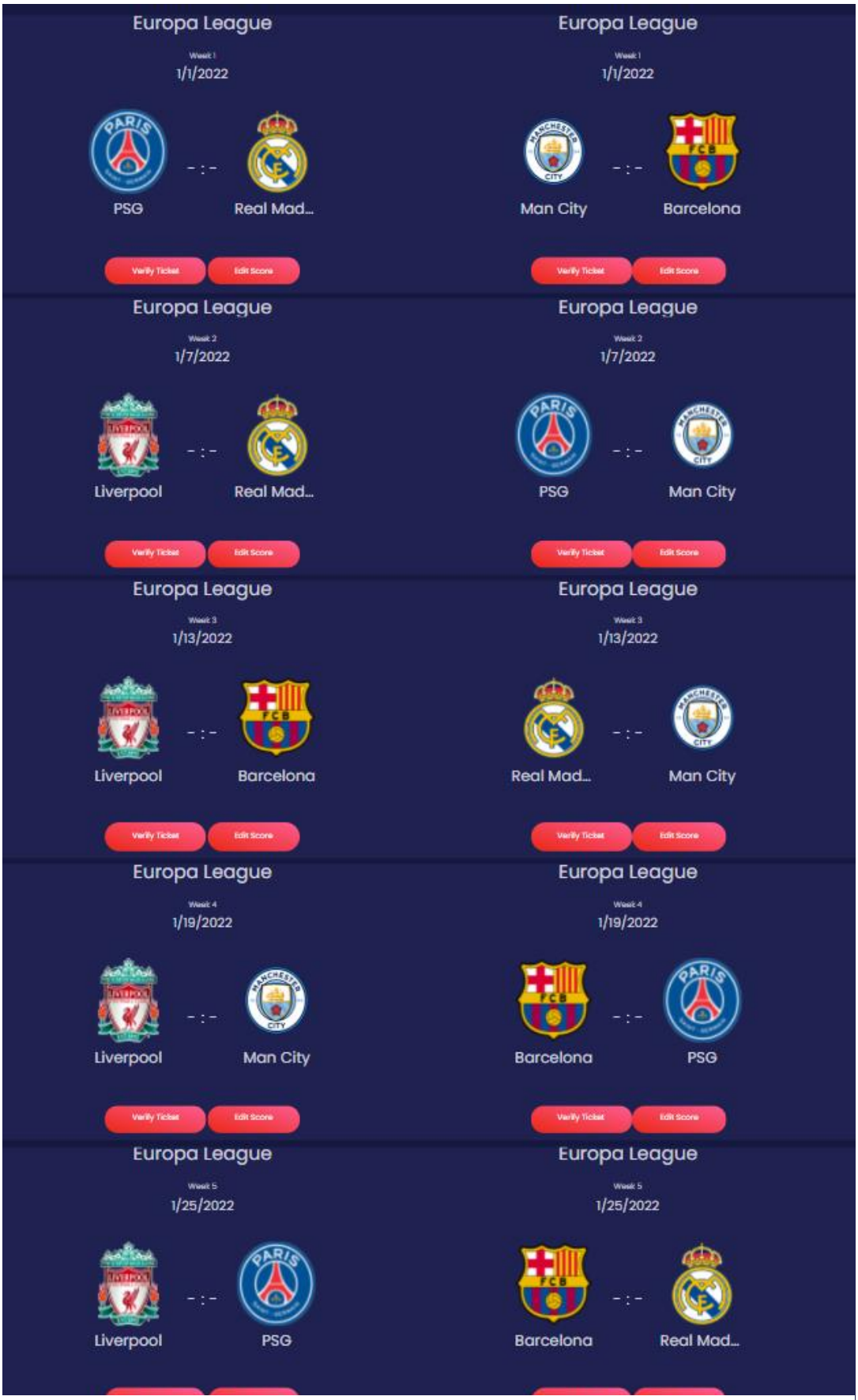


Figure 47: Created matches for ODD no. of teams

16.4.4. ***View Matches***

This feature is available for all types of users: organizers, registered, or anonymous. They can access their favorite teams' and favorite tournaments' matches through "Home" tab, or they can view all available matches through "All Matches" tab.

Displayed matches can be made more focused by using one of the three filters at the top of the page: Team Name, Tournament Name, and Date. With the first, the user can view all matches for a specific team, while the second filter can display all matches in a specific tournament, and finally the date filter shows all the matches that are gonna be played in the picked date.

Initially, as figure .. shows, all available upcoming matches for all teams in any tournaments are displayed with no filters. Using filters, they are reduced to match only the applied filters, as shown in the upcoming 3 test cases.

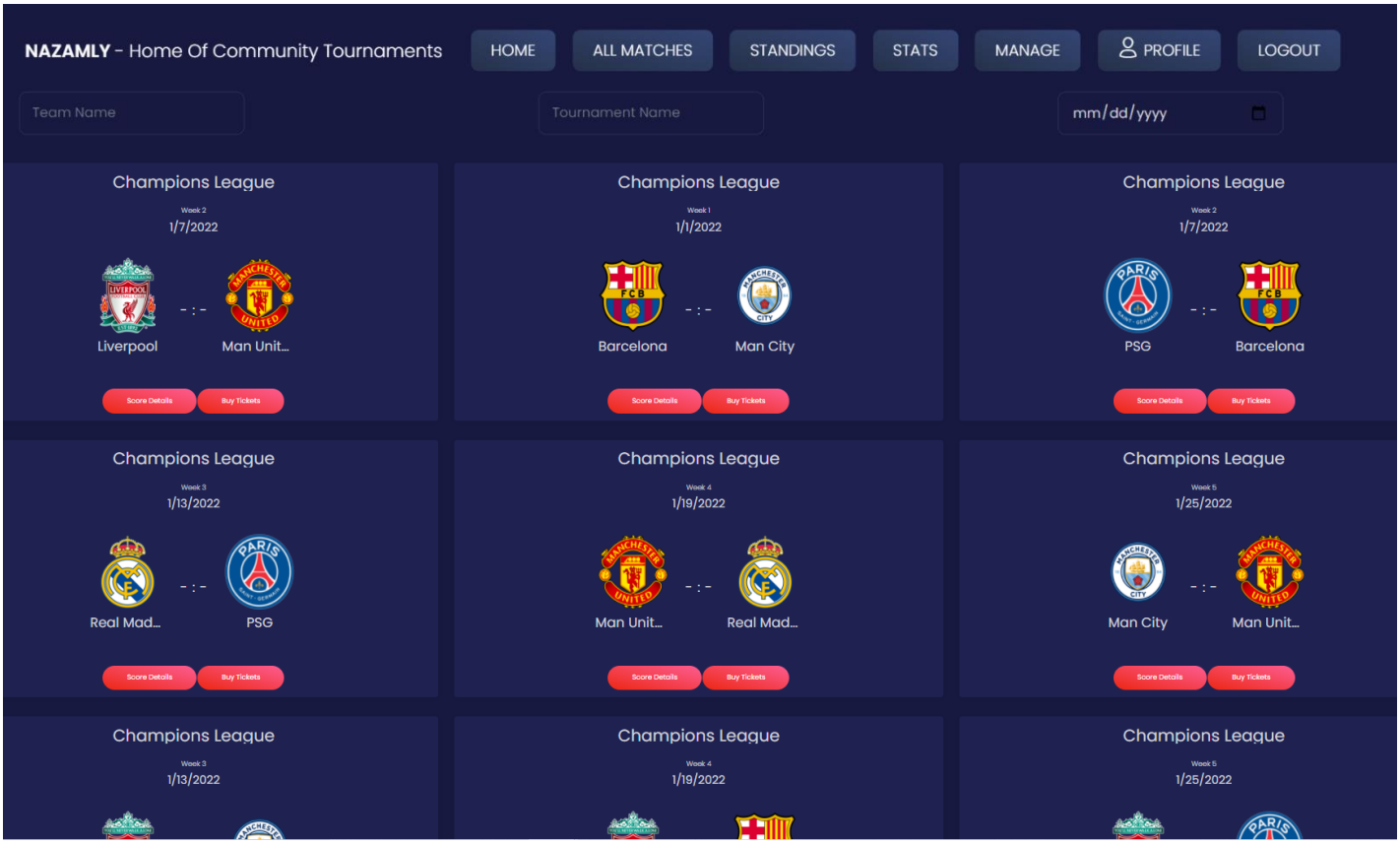


Figure 48: Default view of "All Matches" tab

- **Filter matches by Team Name**

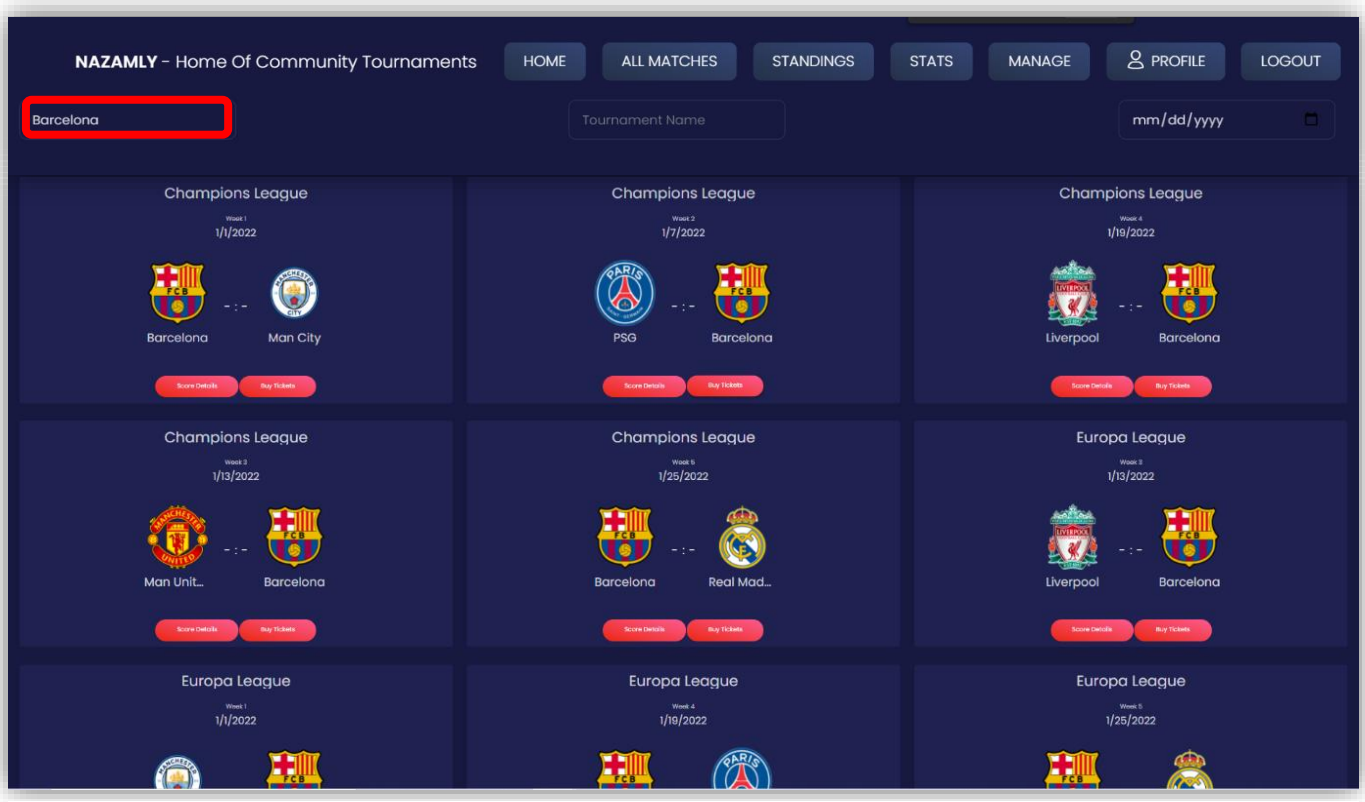


Figure 49: Filter matches by Team Name

- **Filter matches by Tournament Name**

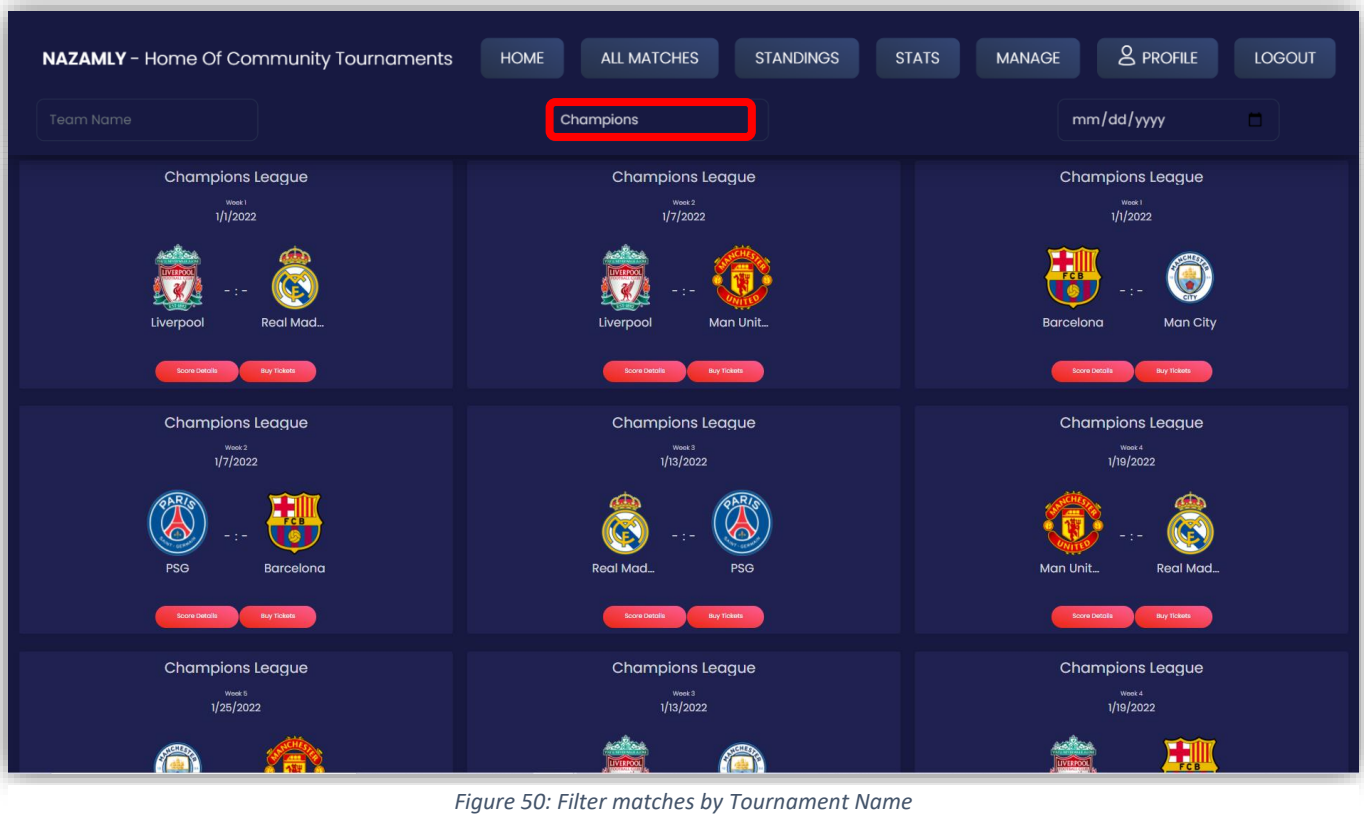


Figure 50: Filter matches by Tournament Name

- **Filter matches by Date**

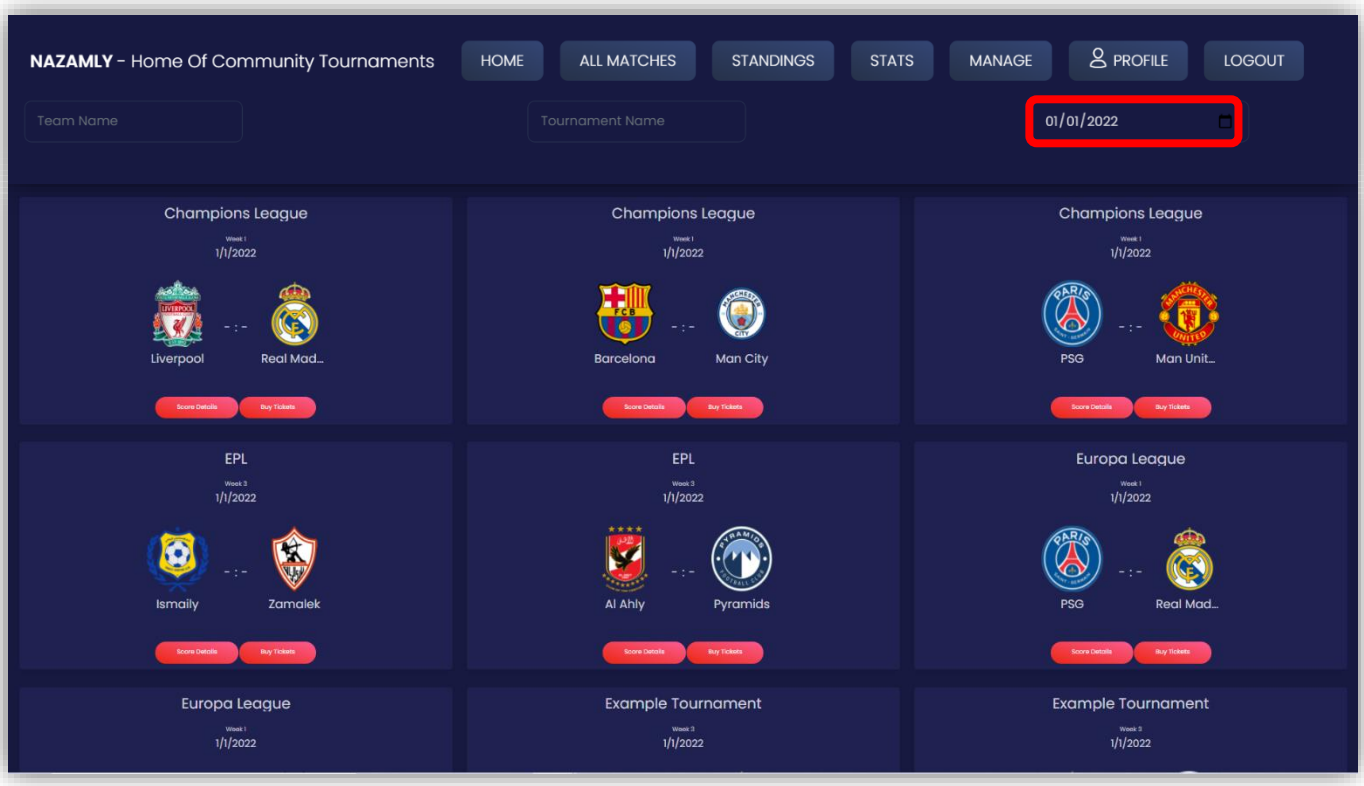


Figure 51: Filter matches by Date

16.4.5. ***Buy Tickets***

One of the features that are available to users is to buy match tickets, which can be accessed ***by clicking the "Buy Tickets" button displayed on any match card. The user can buy one or more tickets with two payment options: Cash payment and Credit Card payment. Once the order is confirmed, a pdf receipt will be automatically available for the user to download.***

However, there is a limit for the no. of ***available tickets for a match which is specified from the organizer when creating the tournament.***

When buying tickets, it is eligible if the ***ordered tickets don't exceed the available ones. Otherwise, a message will be displayed showing the available no. of tickets.***

Moreover, if ***all the match tickets are sold out or there are no tickets originally for this match (the tournament organizer can specify this), the "Buy Tickets" button will show "Sold out" instead.***

Some test cases for all these possibilities are provided below.

- ***Entry of available no. of tickets***

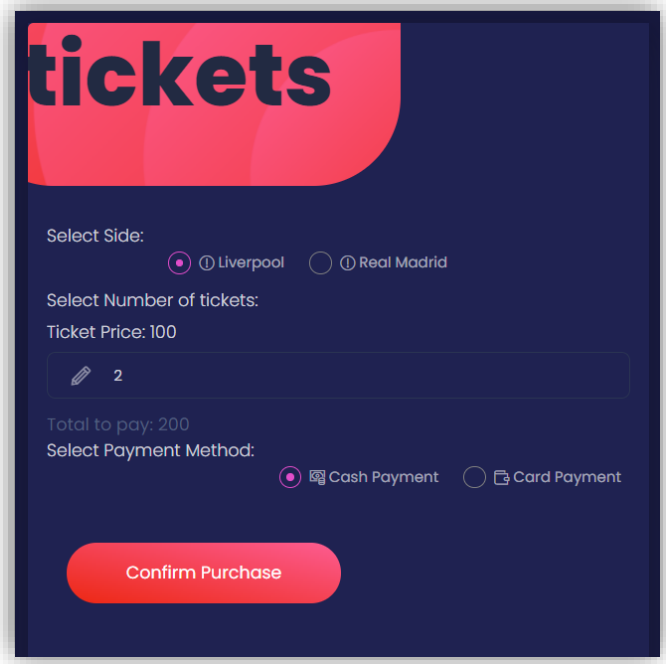


Figure 53: Buy Tickets - Entry of availble no. of tickets

Figure 52: Buy Tickets - Generated e-ticket

- ***Entry of exceeding no. of tickets***

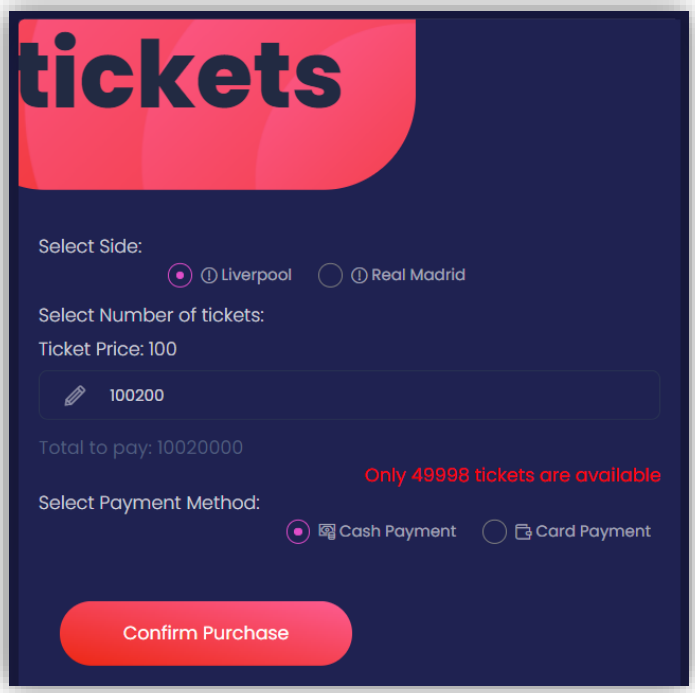


Figure 54: Buy Tickets - Entry of exceeding no. of tickets

- ***No tickets are offered, or all tickets are sold out***

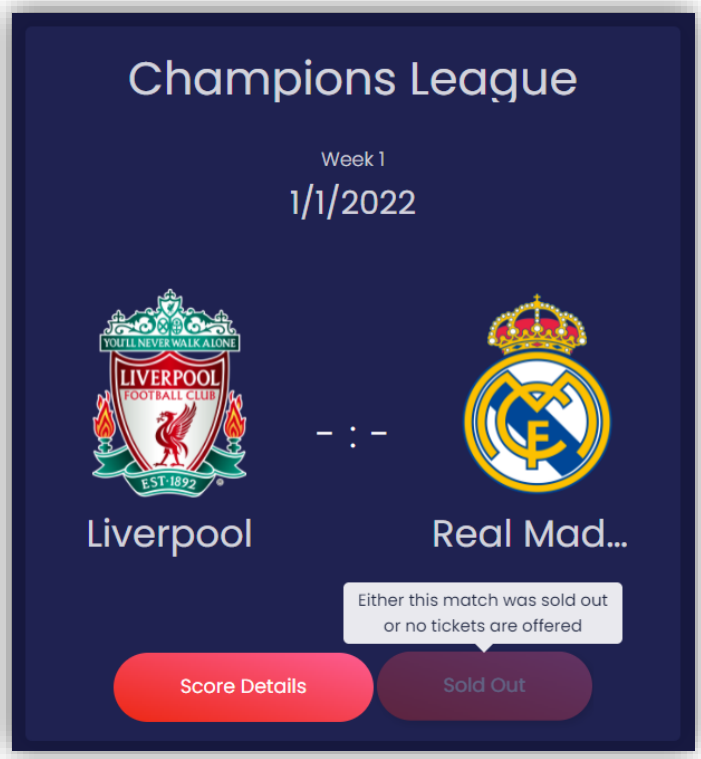


Figure 55: Buy Tickets - No available tickets

17. Estimated Project Cost

The Function Point method is used in the process of cost estimation of this project. It is used as it depends mainly on the detailed specification of the program user interface and hence producing a nearly accurate estimate for the cost of program development. The user input, output, and inquiry screens of the UI; the database files; and interfaces with external systems are counted; then each one is assigned with the appropriate complexity as the below table indicates. Based on this, the Unadjusted Function Point (UFC) is calculated as shown below.

Table 1: Measurement Parameters Details

Parameter	Parameter Elements	Complexity
User Input Screens	1- New Account Registration	Simple
	2- Log in	Simple
	3- Manage Profile	Simple
	4- Buy Tickets	Average
	5- Create New Tournament	Complex
	6- Update Match Result	Complex
User Output Screens	1- Reset Password Alert	Simple
	2- Bought Tickets Report	Simple
	3- Sales Report	Average
User Inquiries Screens	1- Default Home Screen	Simple
	2- Personalized Home Screen	Average
	3- All Matches	Average
	4- League Standings	Complex
	5- Players Statistics	Complex
Database Files	1- Users File	Simple
	2- Admins File	Simple
	3- Tickets File	Simple
	4- Players File	Simple
	5- Teams File	Average
	6- Matches File	Average
	7- Tournaments File	Complex
External Interfaces	1- External Bank Database	Simple
	2- Email Service API	Average

17.1. Unadjusted Function Point Calculation

Table 2: Unadjusted Function Point

Measurement Parameter	Count			Weighting Factor			Sum
	Simple	Average	Complex	Simple	Average	Complex	
Number of User Inputs	3	2	1	3	4	6	23
Number of User Outputs	2	1	---	4	5	7	13
Number of User Inquiries	1	2	2	3	4	6	23
Number of Files	4	2	1	7	10	15	63
Number of External Interfaces	1	1	---	5	7	10	12
Unadjusted Function Point (UFC) count							134

17.2. Complexity Factor Ratings

Table 3: Complexity Factor

i	Complexity Factor	Rate/F (0-5)
1	Backup and recovery	1
2	Data communication	5
3	Distributed processing functions	3
4	Is performance critical?	4
5	Existing operating environment	3
6	On-line data entry	5
7	Input transaction built over multiple screens	0
8	Master files updated on-line	5
9	Complexity of inputs, outputs, files, inquiries	3
10	Complexity of processing	3
11	Code design for re-use	2
12	Are conversion/installation included in design?	0
13	Multiple installations	1
14	Application designed to facilitate change by the user	5
Total Count		**40**

Adjusted FP Calculation:

$$FP = UFC * \left(0.65 + 0.01 * \sum_{i=1}^{14} F_i \right)$$

$$FP = 134 * (0.65 + 0.01 * 40) = 134 * 1.05 \cong 141$$

1.1. LOC Calculation:

Since the software will be developed mainly using JavaScript, and HTML the expected number of Lines Of Code (LOC) is calculated as follows, considering JavaScript has an average LOC/FP of 53, and HTML has 40 LOC/FP averaging 46.5.

$$\boldsymbol{Expected\ LOC = FP * \left(\frac{LOC}{FP}\right) = 141 * 46.5 = 6556 = 6.556\ KLOC}$$

1.2. Effort & Time Calculation Using CoCoMo Model:

Nazamly software application can be classified as an organic type as the team size required is adequately small, the problem is well understood and has been solved in the past, and also the team members have a nominal experience regarding the problem. Hence, the values for the constants that will be used in calculations are:

$$\boldsymbol{a = 2.4, \quad b = 1.05}$$

The Basic COCOMO Model can be used for quick and slightly rough calculations of Nazamly software costs by estimating required effort and time as follows:

$$\boldsymbol{Effort = a * (KLOC)^{b} = 2.4 * (6.556)^{1.05} = 17.33\ Person/Month}$$

The time required for developing the software is about 2 months.

$$\boldsymbol{Persons\ Required = \frac{Effort}{Time} = \frac{17.33}{2} \cong 9\ Persons}$$

Finally, an approximate estimate for the project cost can be calculated by assuming an average salary of 15,000 L.E./Person.

$$\boldsymbol{Total\ Cost = 9 * 15000 = 135,000\ L.E.}$$

18. USER GUIDE

18.1. Getting Started

This is your first time using the Nazamly webapp, an application that offers a solution to you if you need to follow matches results of the teams you would love to see.

The system can be used anonymously or while logged into your free-to-create Nazamly account for a personalized experience and some extra features.

Starting the application for the first time will open to the default home screen in the following figure.

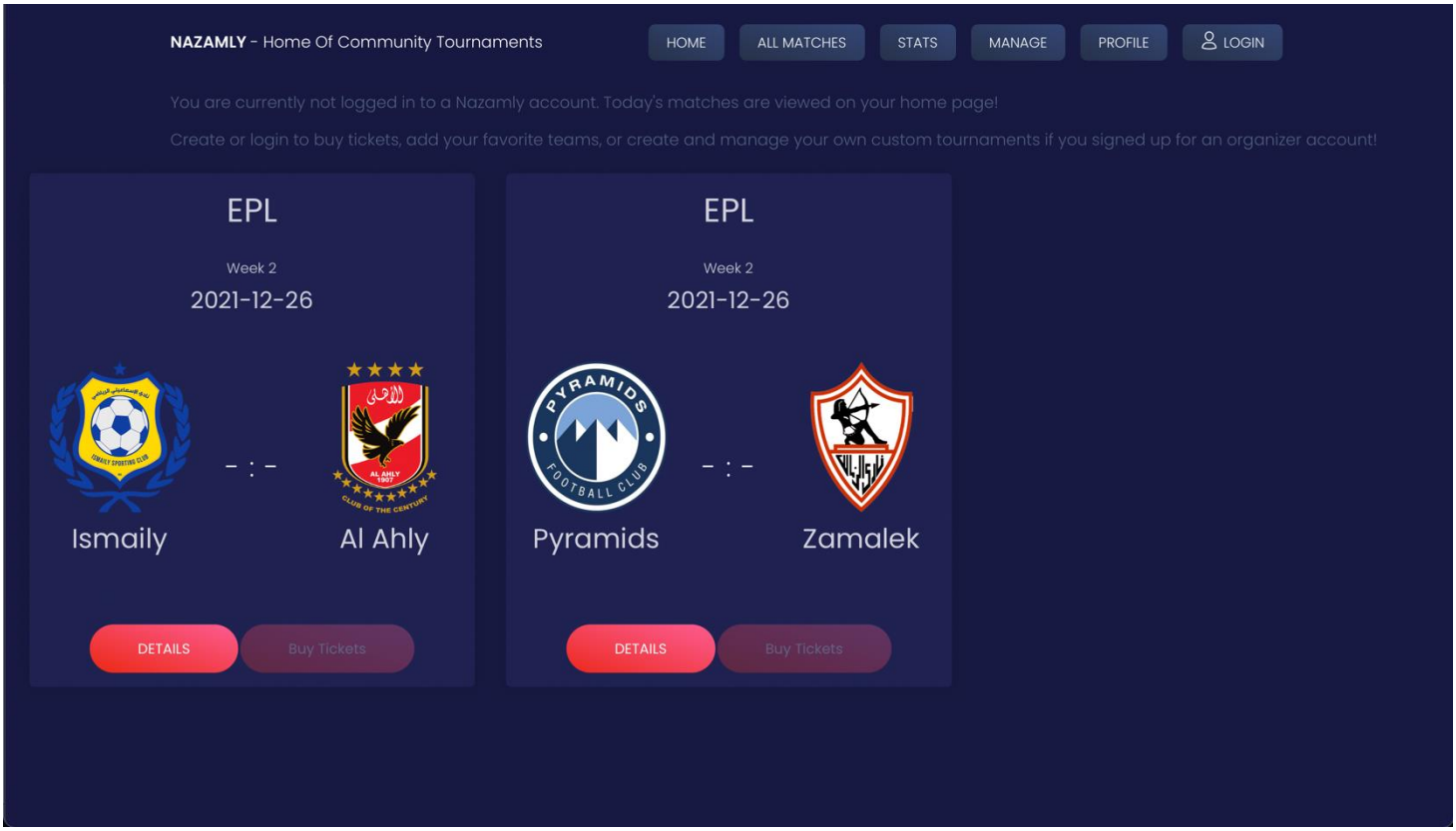


Figure 56: Getting Started

*18.1.1. **Creating a new Nazamly account***

It is highly recommended to create a new free-to-create Nazamly account, which is a very simple operation where you will have your own account within a few steps.

1- Press on the “Login” button on the top right of your home screen.

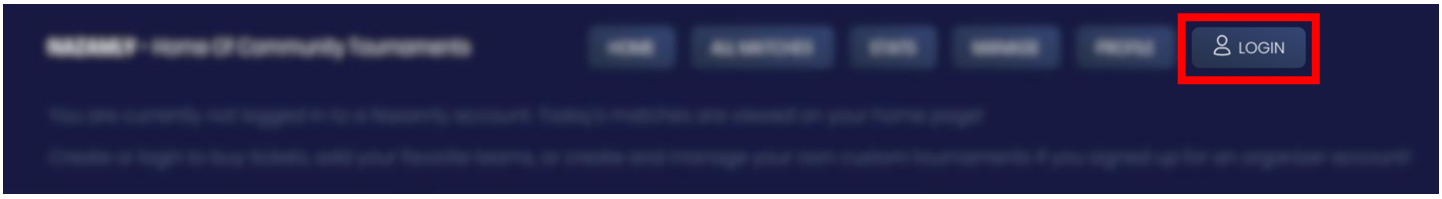


Figure 57: Login Button

2- Fill in your personal information in the “register” box as follows, name, email, a password that is not shorter than 6 letters long, and only check the last checkbox if you are a tournament organizer or willing to be one.

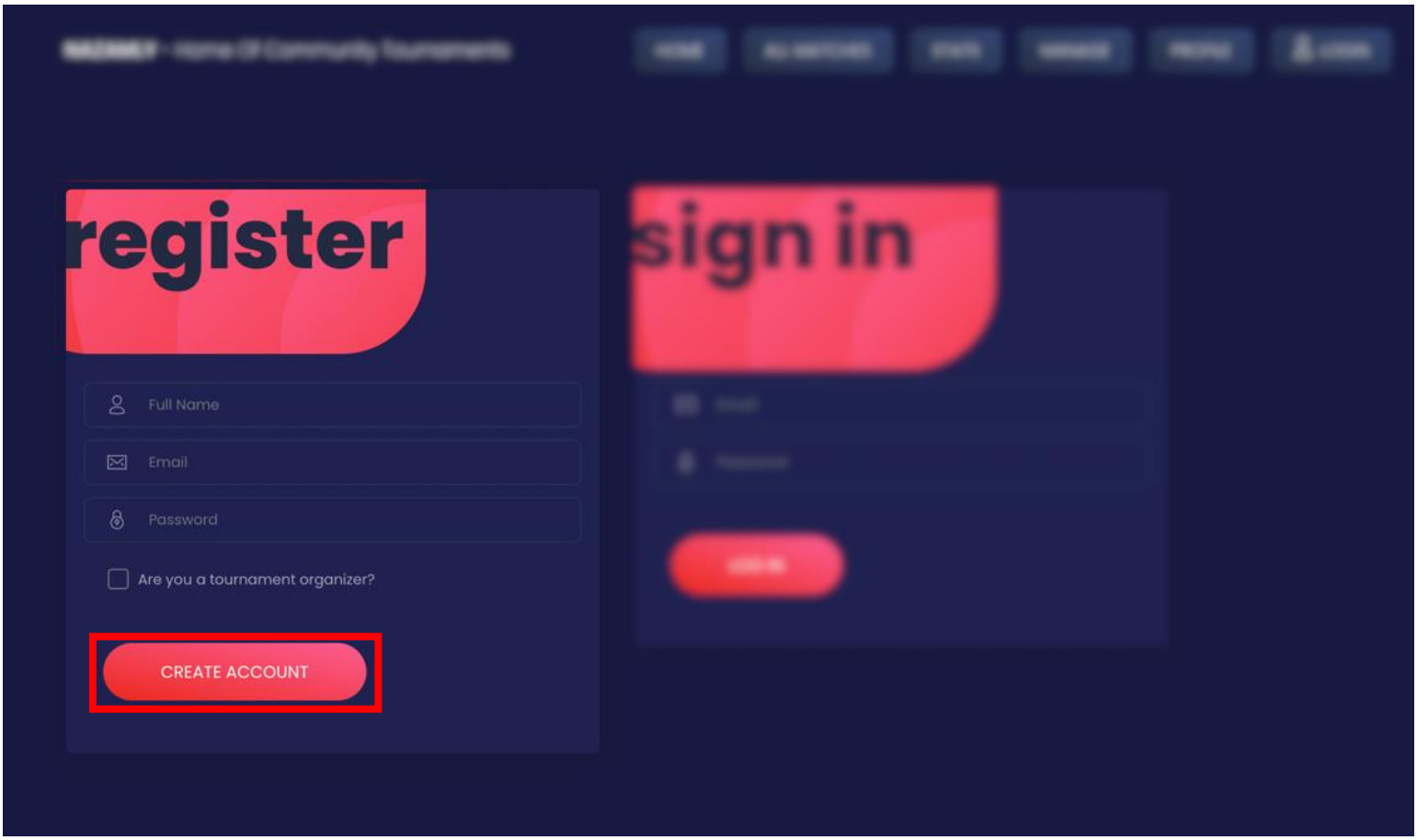


Figure 58: Register new account

3- Press the “CREATE ACCOUNT” button.

*18.1.2. **Logging into your Nazamly account***

When you sign in to your Nazamly account, you start your very own customized experience.

1- Fill in your account credentials in the “sign in” including the email you used in creating your account and your account password.
2- Press the “LOG IN” button.

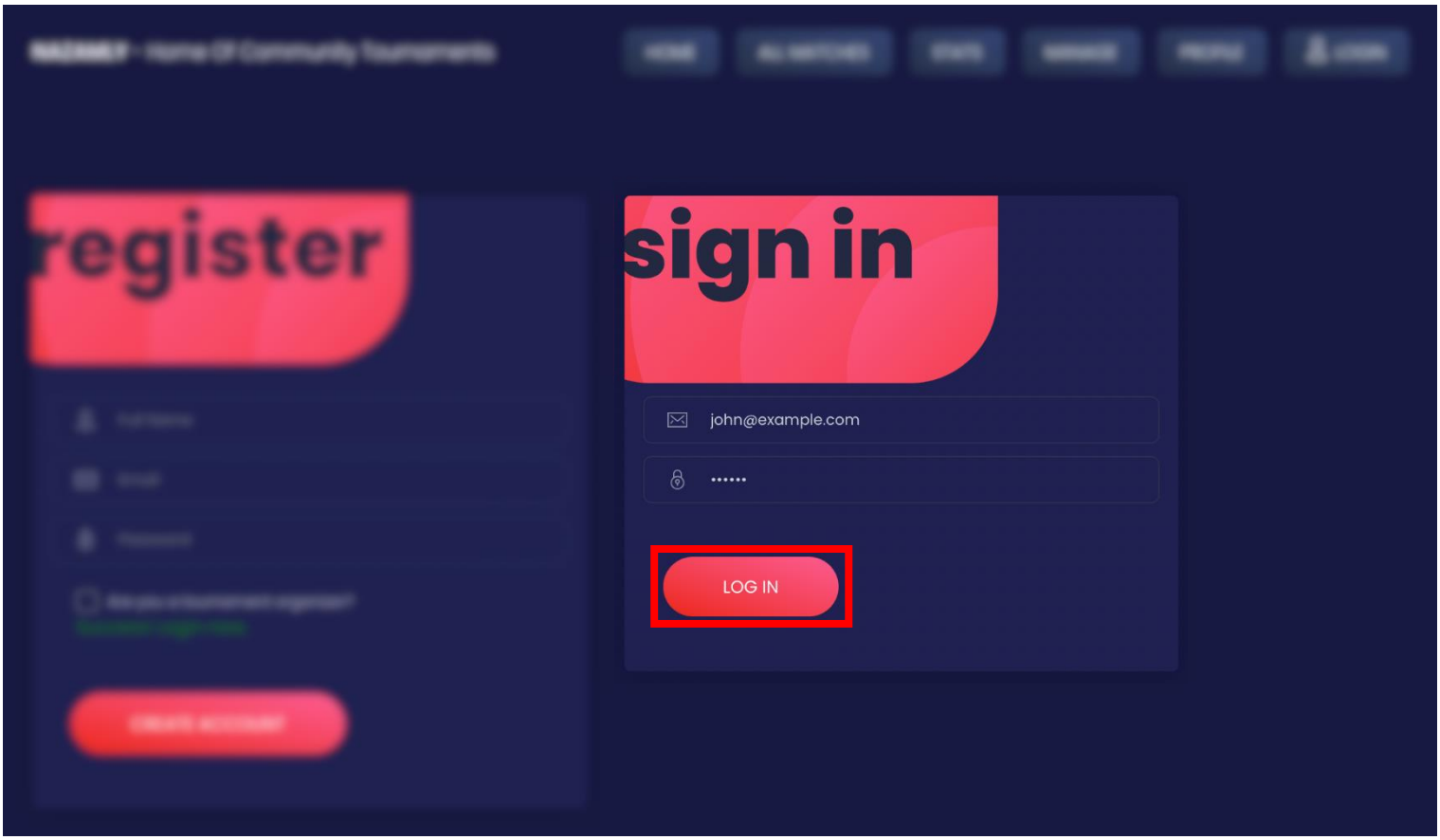


Figure 59: Log in

3- You will be redirected if your credentials were correct.

18.2. Browsing Nazamly tabs

Nazamly webapp has a lot of features to help you enjoy your sports following experience to the max. These features are divided between 6 tabs on the top of your screen.

18.2.1. All matches

This tab includes all the matches available for viewing through Nazamly, organized in match cards that include the tournament name, match date, opponents' names, and the match final score if it was played. Today's matches are the one available by default to view, while you can filter view the match cards by 3 different filters, available in the top of your screen.

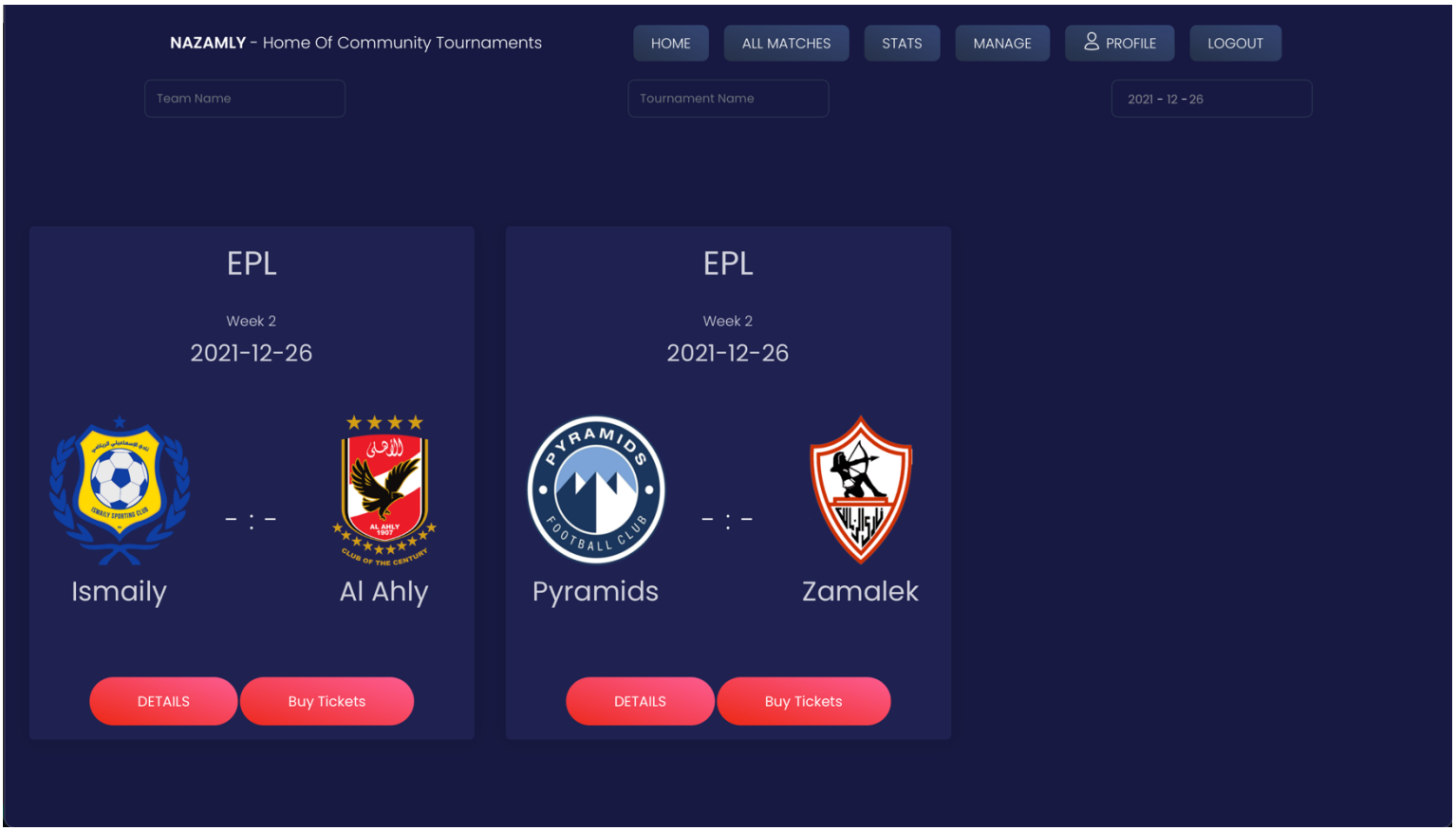


Figure 60: Default All Matches Tab

1- Filter by Tournament

A search bar that allows you to view matches of one of any of the available tournaments through Nazamly.

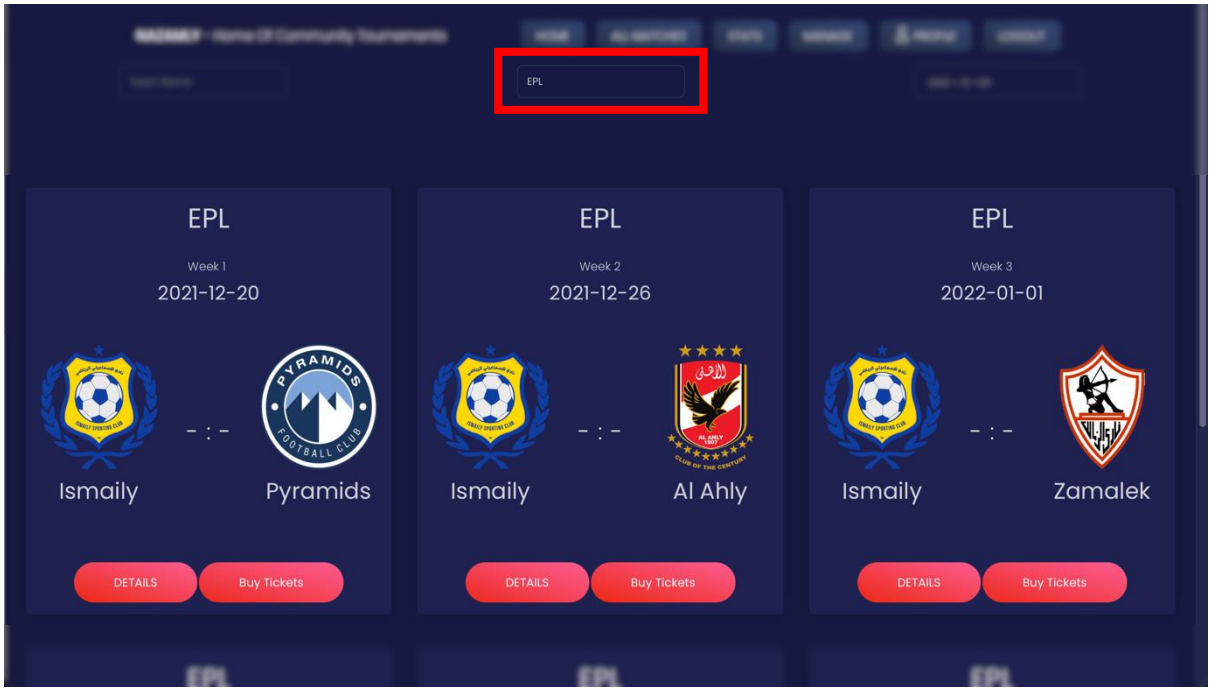


Figure 61: Filter by Tournament

2- Filter by Team

A search bar that allows you to view matches of one of any of the available teams.

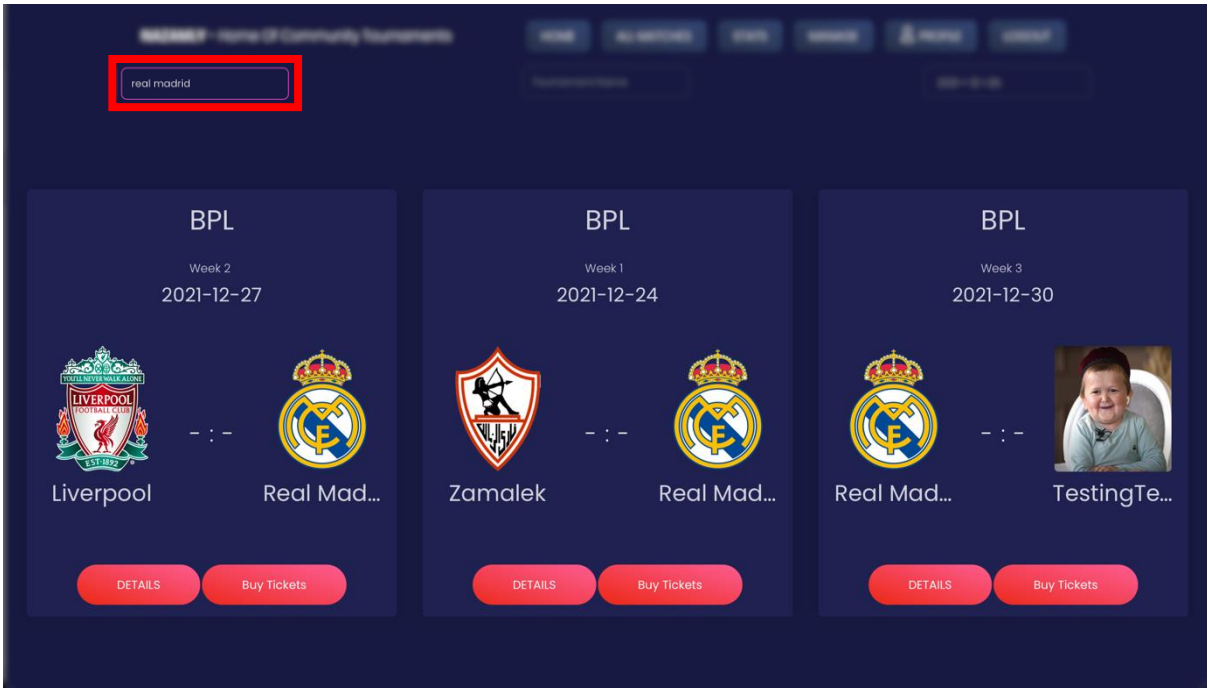


Figure 62: Filter by Team

3- Filter by Date

Choose whatever day that you want to view matches that were played or will be played on from an easy-to-use date-picker.

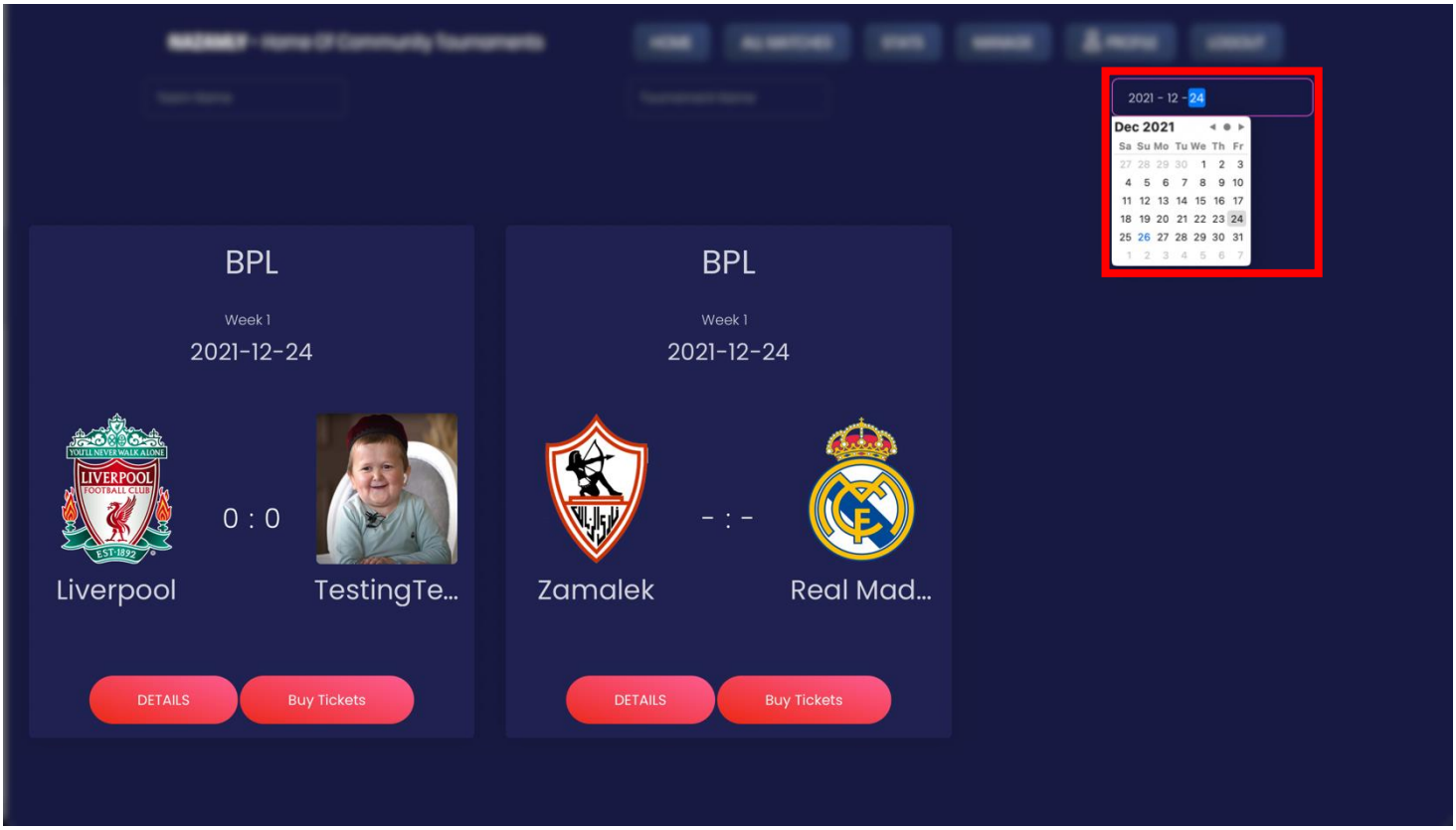


Figure 63: Filter by Date

18.2.2. ***Buying Tickets***

This feature is only available if you are logged into your free-to-create Nazamly account.

On both “All Matches” and “Home” tabs, you can find a “Buy Ticket” button on any of the match cards that will enable you to buy this specific match tickets paying in cash or using your card.

1- Press the “Buy ticket” button on any match card.

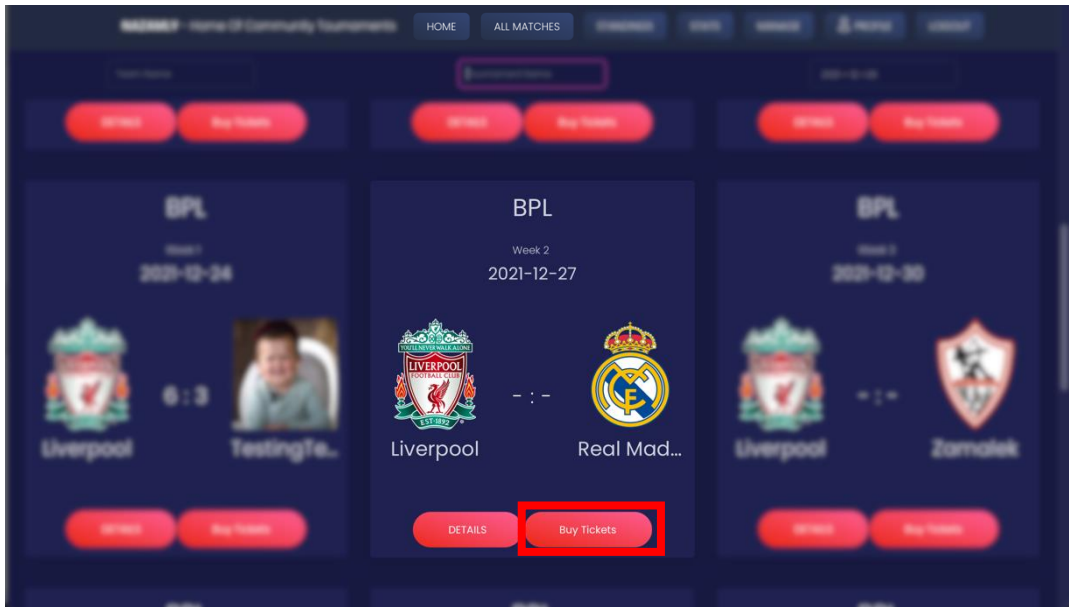


Figure 64: Buy Tickets Button

2- Choose the team, number of tickets and choose payment method. Confirm & review your purchase to pay and download your tickets as a PDF e-ticket.

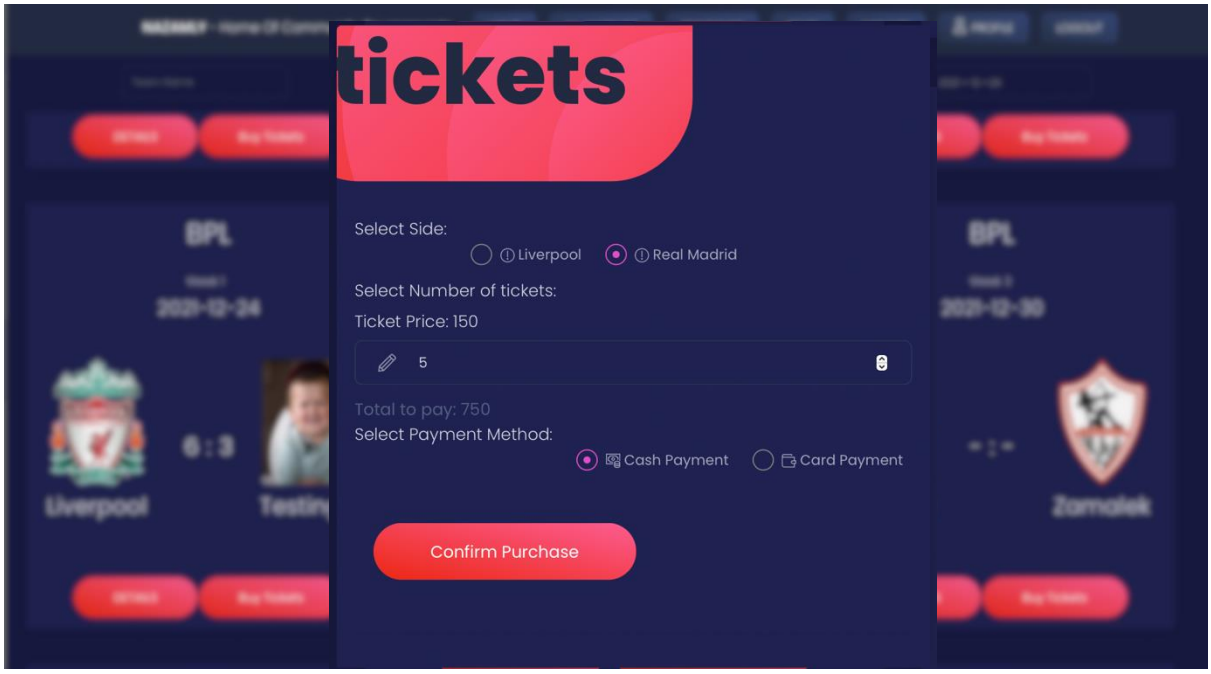


Figure 65: Tickets Cash Purchase

3- If you chose card payment you will have to fill in your card number, CVV, and card expiry date in order to pay.

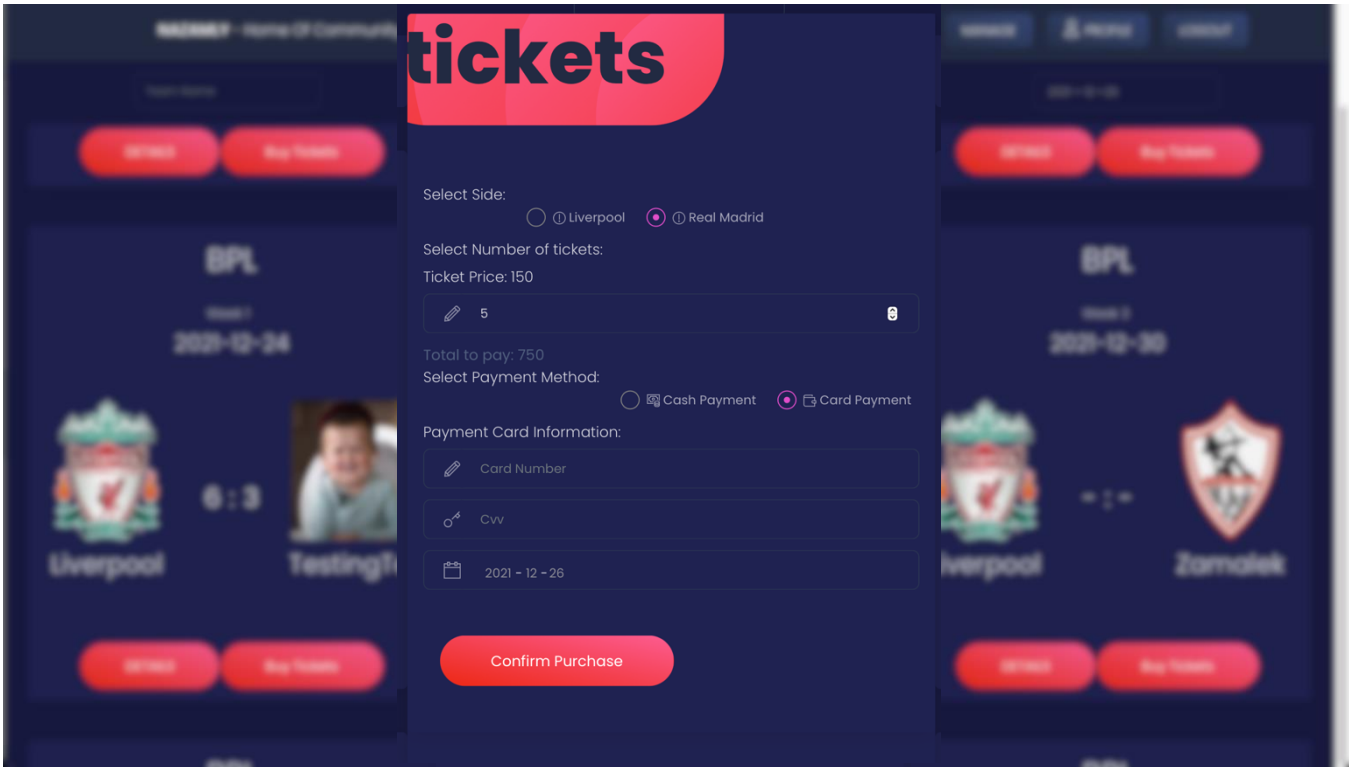


Figure 66: Tickets Card Purchase

4- A PDF formatted file will be downloaded containing your e-ticket as follows.

Figure 67: E-ticket example

18.2.3. ***Standings***

This tab includes the standings of all the tournaments available for viewing through Nazamly organized in table view. You can choose a tournament to view its standings and you can sort the tournament's teams through several sort options.

The table view is initially empty until you choose a tournament.

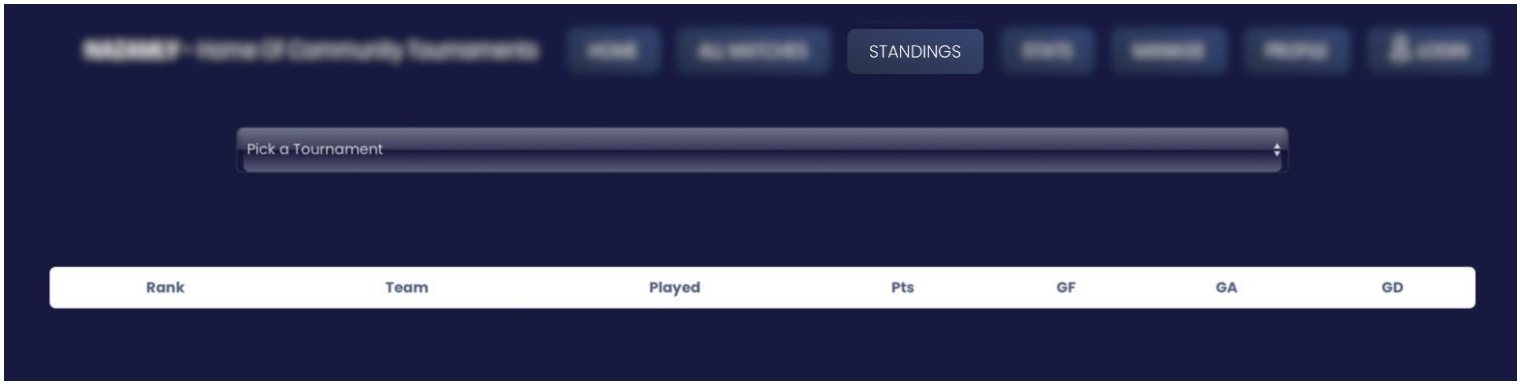


Figure 68: Standings Initial View

1- Choose a Tournament

A drop-down menu allows you to choose from the tournaments' name list to view its standings in table view. Each row displays a team record including all its details.

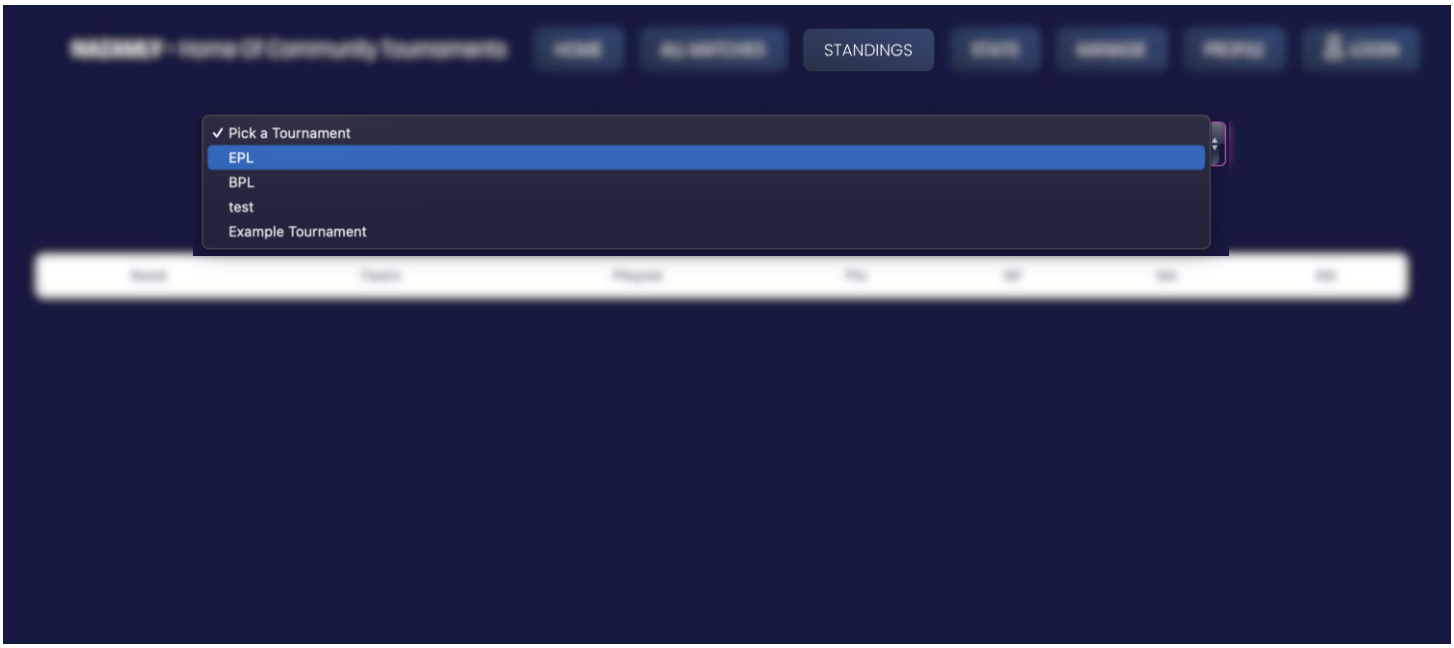


Figure 69: Choose Tournament Drop-down

2- Sort by Attribute

You can press on any of the table columns heads to sort the teams according to that attribute, ascending or descending. An example of descending sorting teams according to their league rank would be as in figure.

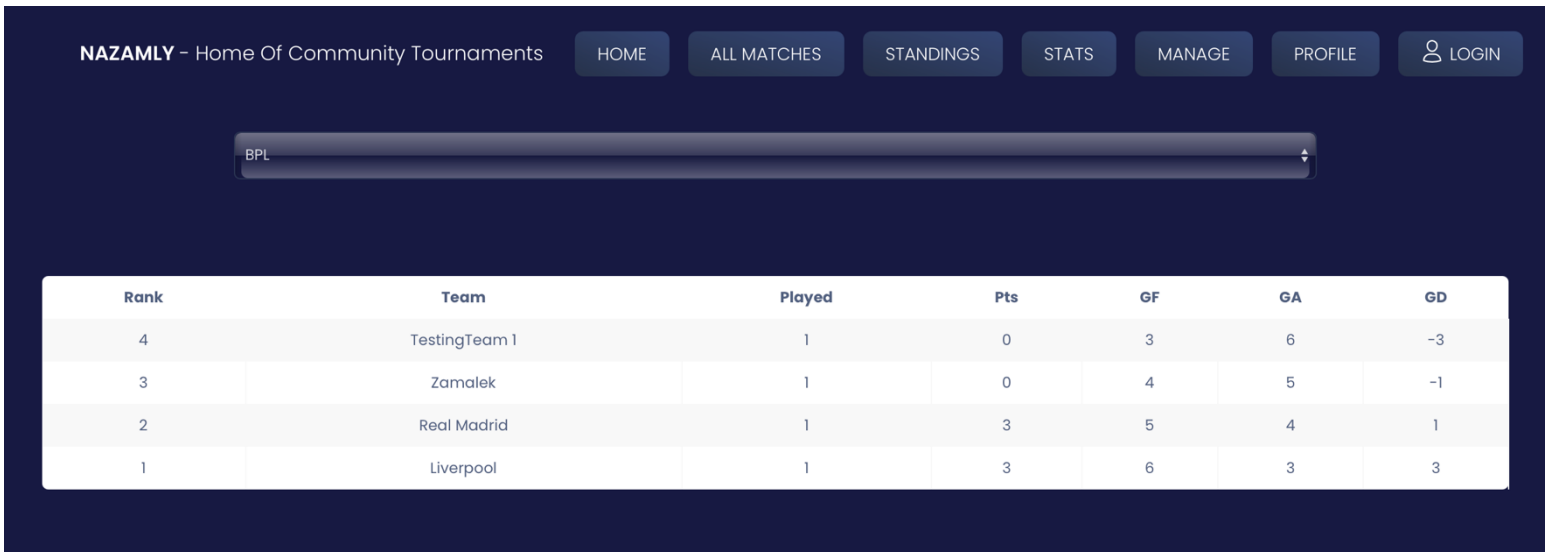
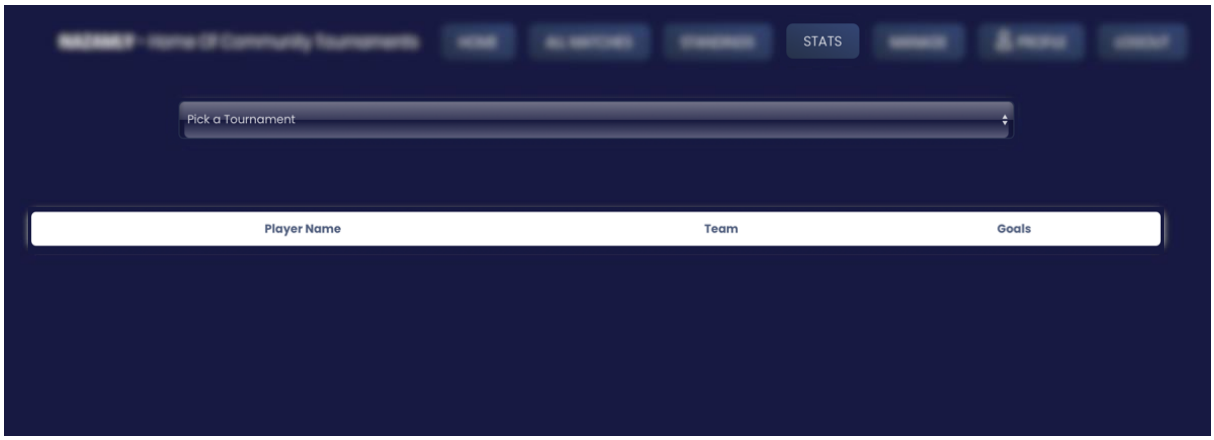


Figure 70: Sorting Standings Table

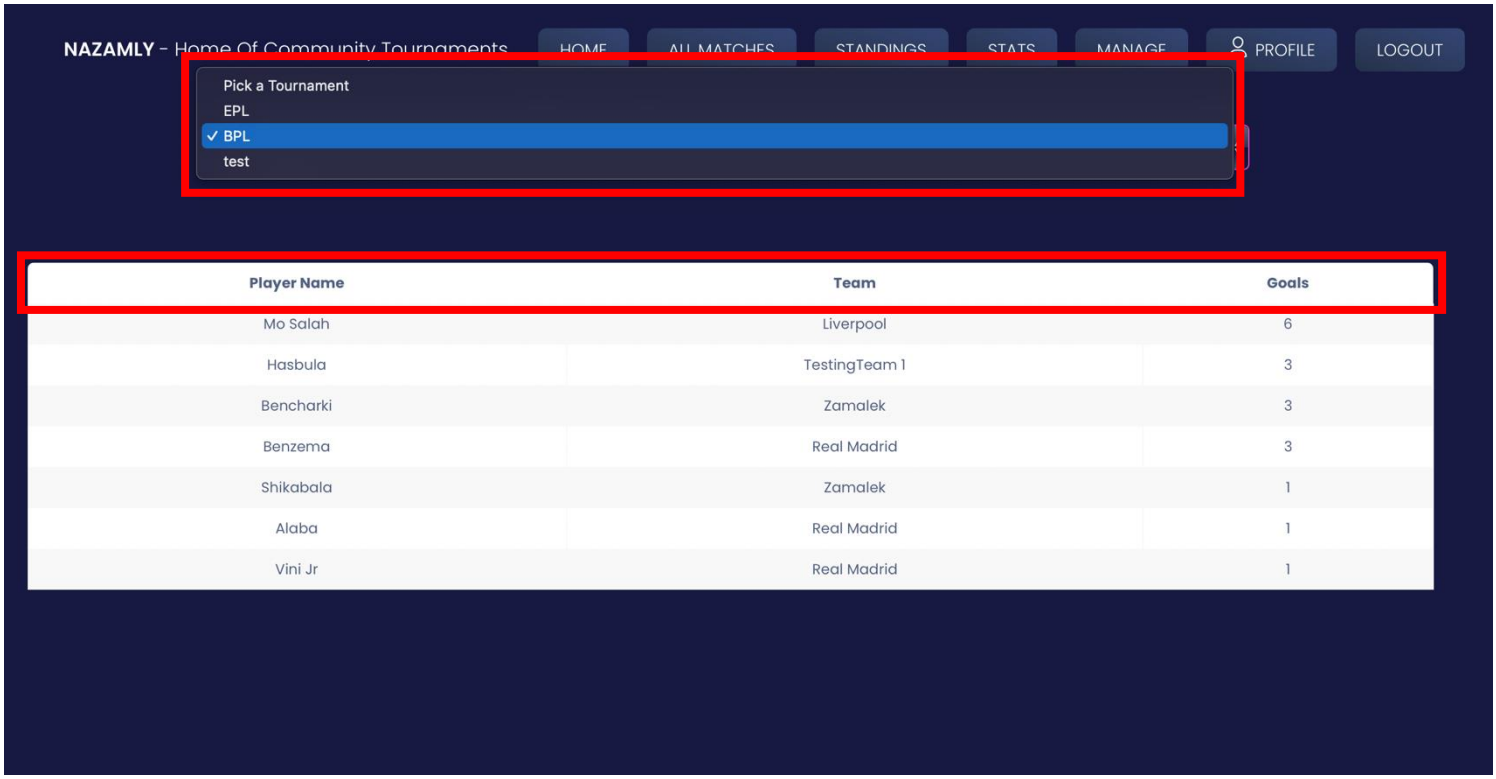
18.2.4. ***Statistics***

This tab features the player statistics of all tournaments available through Nazamly. The page is initially empty with a dropdown menu to choose a tournament to view its scorers.



1- Choose a tournament from the dropdown menu.

2- Sort ascendingly and discerningly by pressing on the table headers.



18.2.5. ***Profile Page***

This tab helps you customize your Nazamly experience as much as possible where you can add, remove favorite teams, view your account details, and change your account password.

1- To change your password, press “Change Password” button.

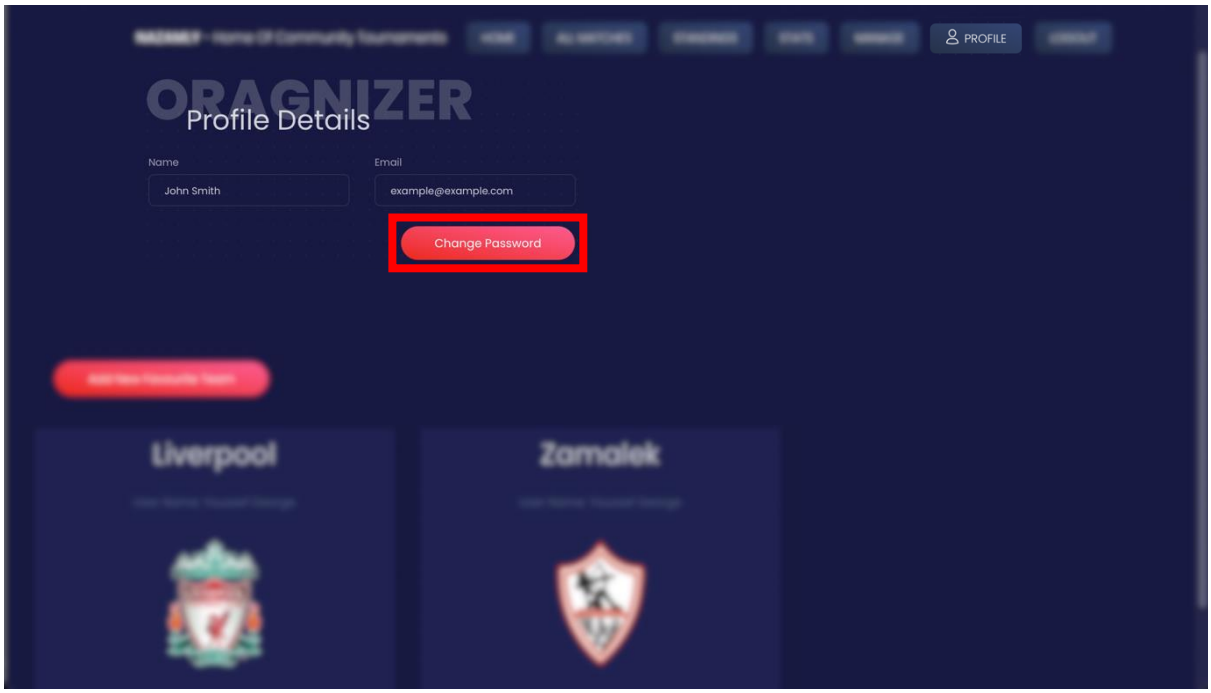


Figure 71: Profile Page

2- Enter your current password in addition to your new password, and submit.

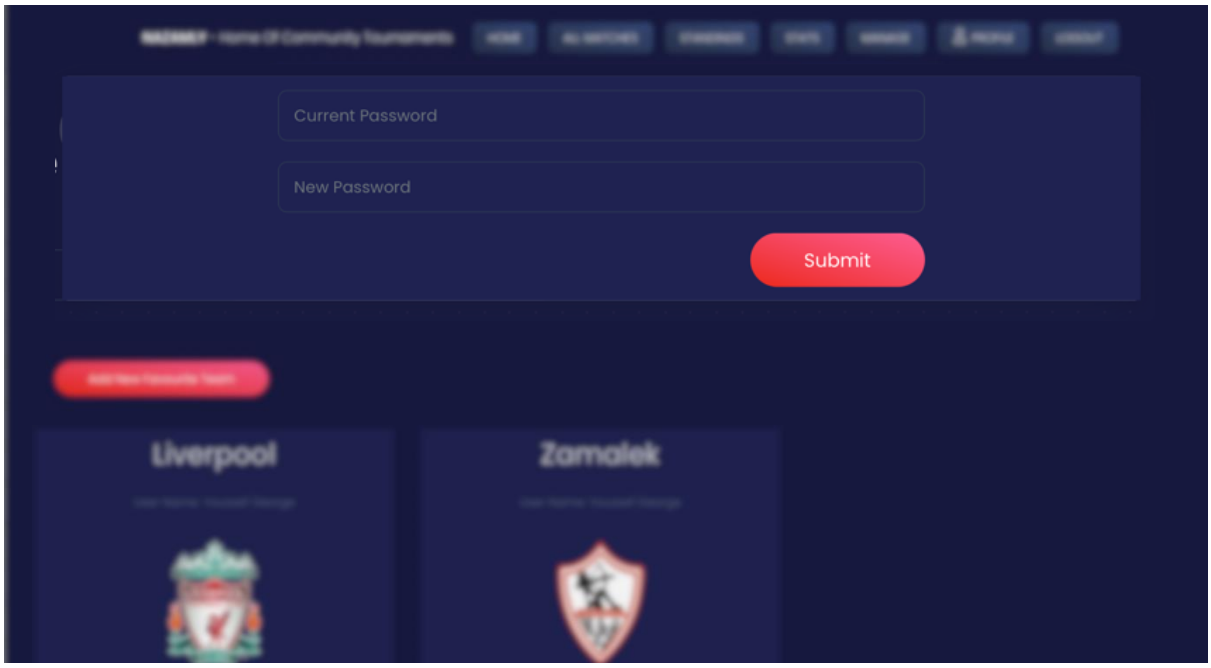


Figure 72: Changing your Password

1- To add to your favorite teams, press the “Add New Favourite Team” button.

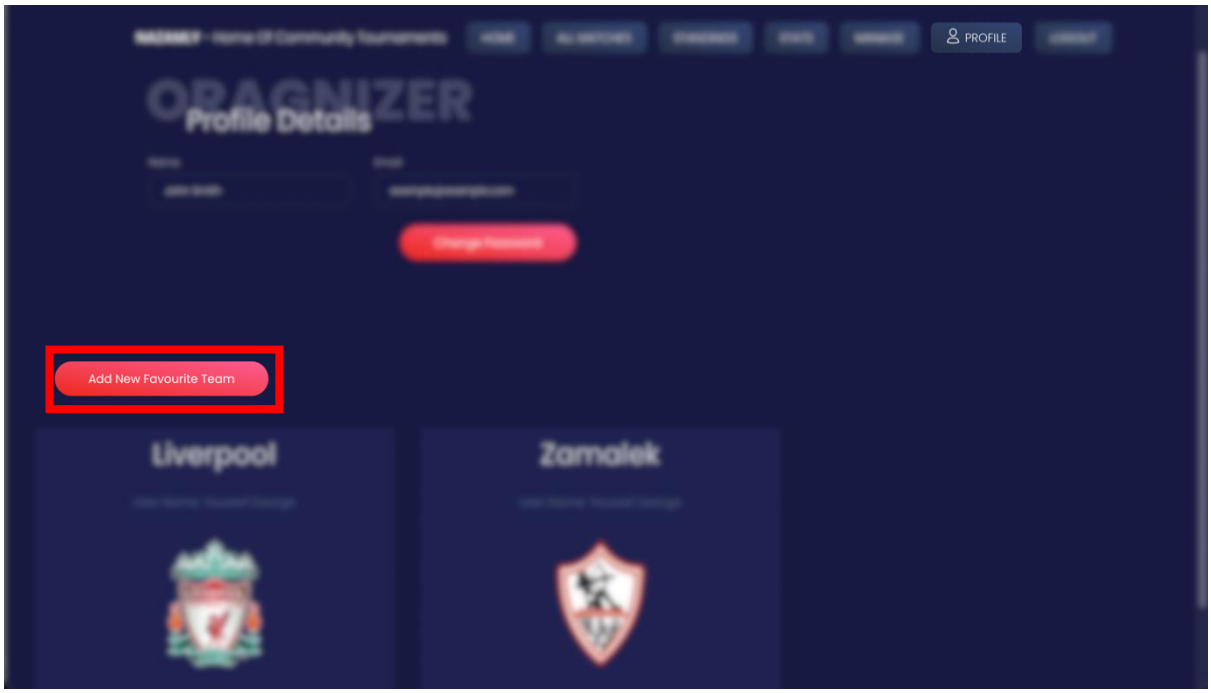


Figure 73: Add Favorite Team

2- Press the “Add” button on any number of teams to add to your favourite teams list.

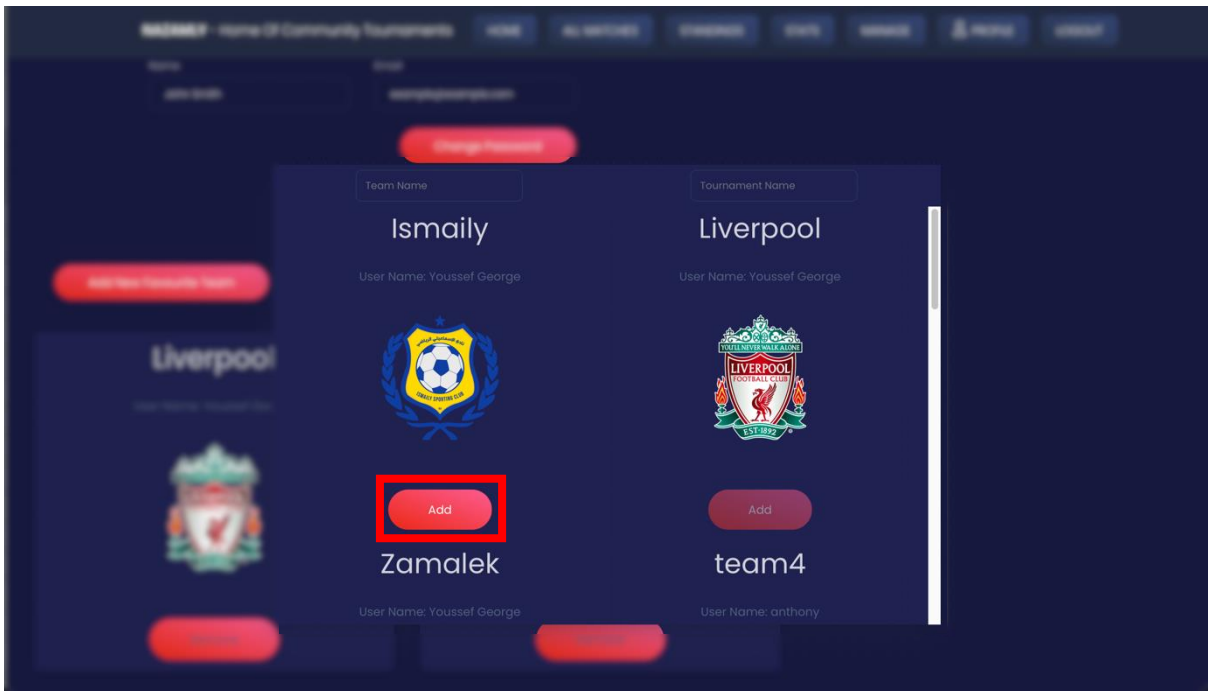


Figure 74: Add Favourite Teams Modal

18.3. Organizers-Only Features

As an organizer, you will have administrative and organizational features in the “Manage” tab where you can add teams, create a new tournament, update its matches’ final score, provide goal scorers’ names, and get a sales report of your tournaments’ tickets.

18.3.1. Adding new team

1- Press “Add Team” button on the top left of your page.

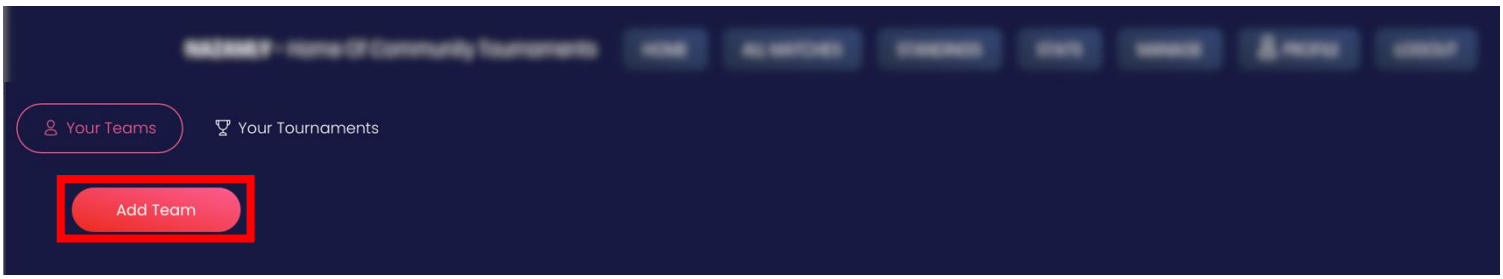


Figure 75: Add New Team

2- Enter the new team’s name and upload the team crest/ logo (optional).

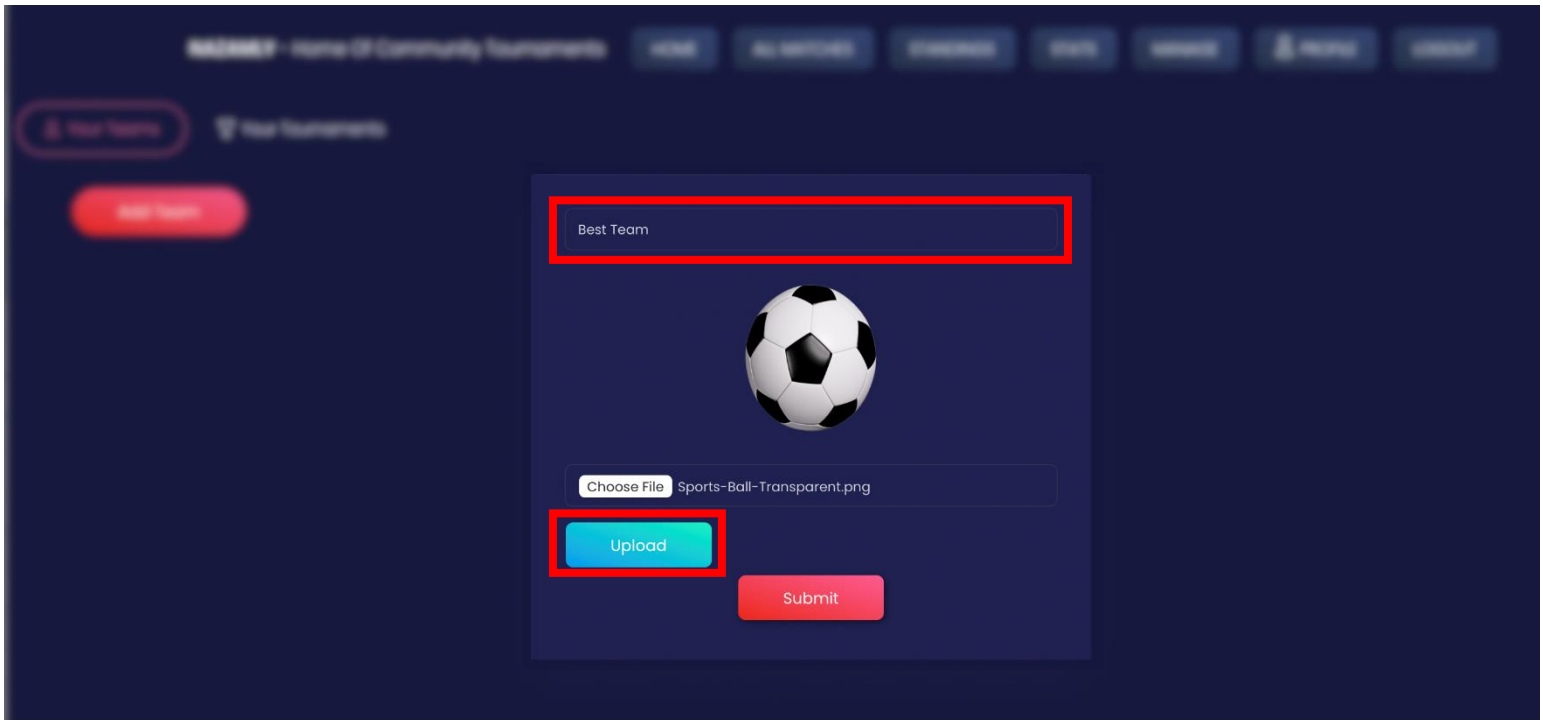


Figure 76: New Team Information

3- Once submitted, new team is added to Nazamly and you can view and edit your teams information from “Your Teams” sub-tab as shown.

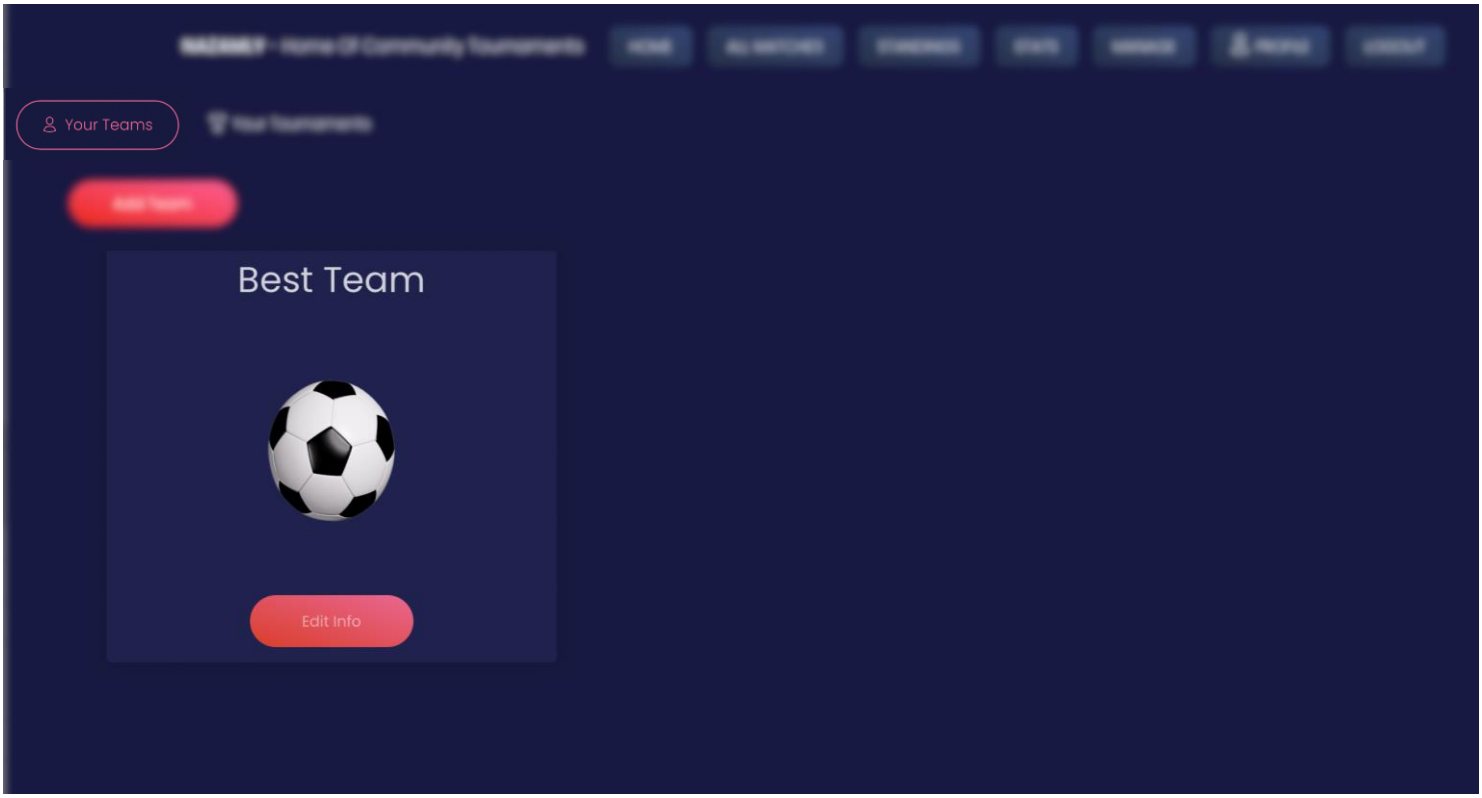


Figure 77: View your teams

18.3.2. ***Create New Custom Tournament***

1- Press “New Tournament” button under “Your Tournaments” sub-tab.

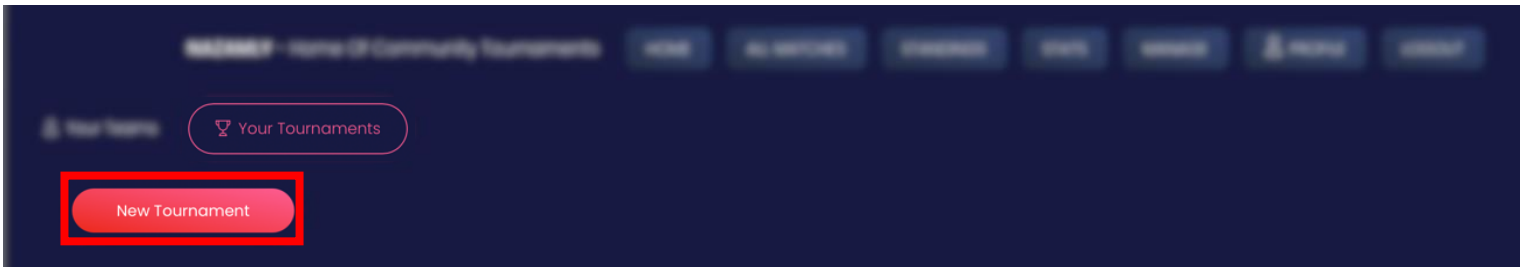


Figure 78: New Tournament Button

2- Fill in tournament's name, teams, start date, end date, number of tickets available per match, and ticket price.

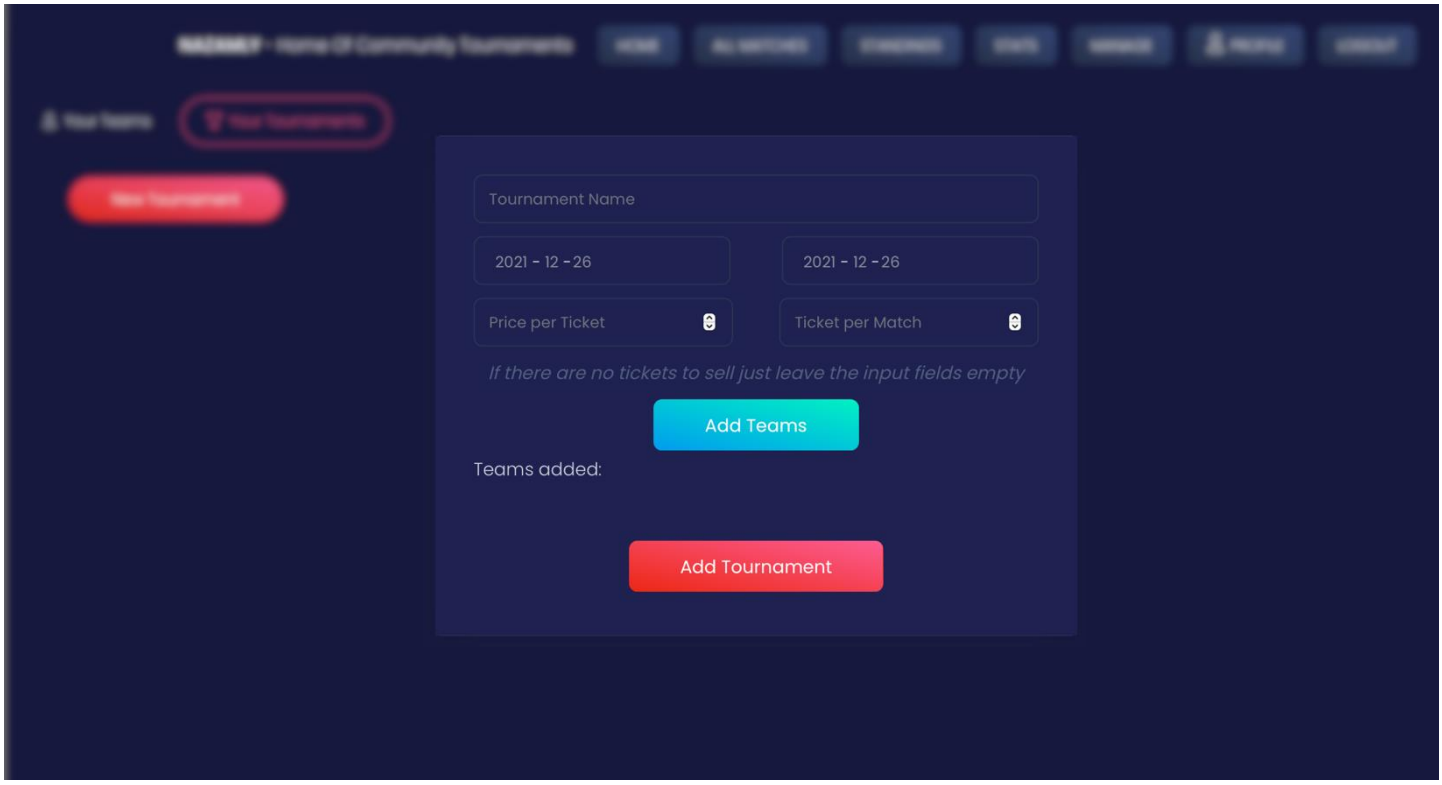


Figure 79: Create New Tournament

3- League matches are auto-generated and added to the matches list according to your start and end date.

*18.3.3. **Update Match Result***

After creating your custom tournament with the auto-generated matches schedule, you shall be able to update the match results for your tournament's fans to view.

1- On the "Manage" tab, under the "Your Tournaments" sub-tab, you will be able to see a horizontal list of your tournaments where you can press any of them to view your tournament matches before updating its score.

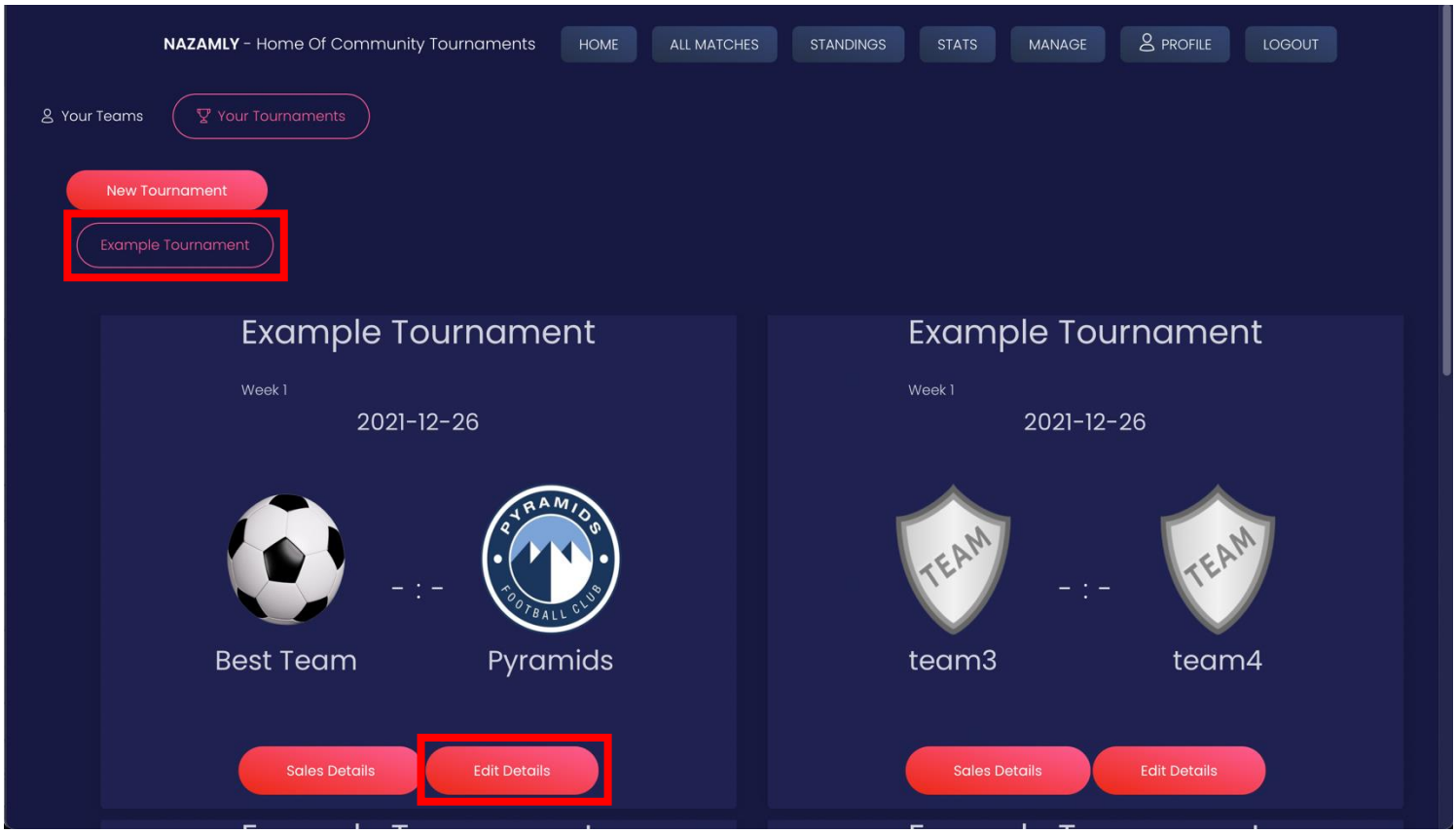


Figure 80: Your Tournaments

2- Pressing “Edit Details” button allows you to add the number of goals for each team in addition to optionally adding the goal scorer name of each goal.

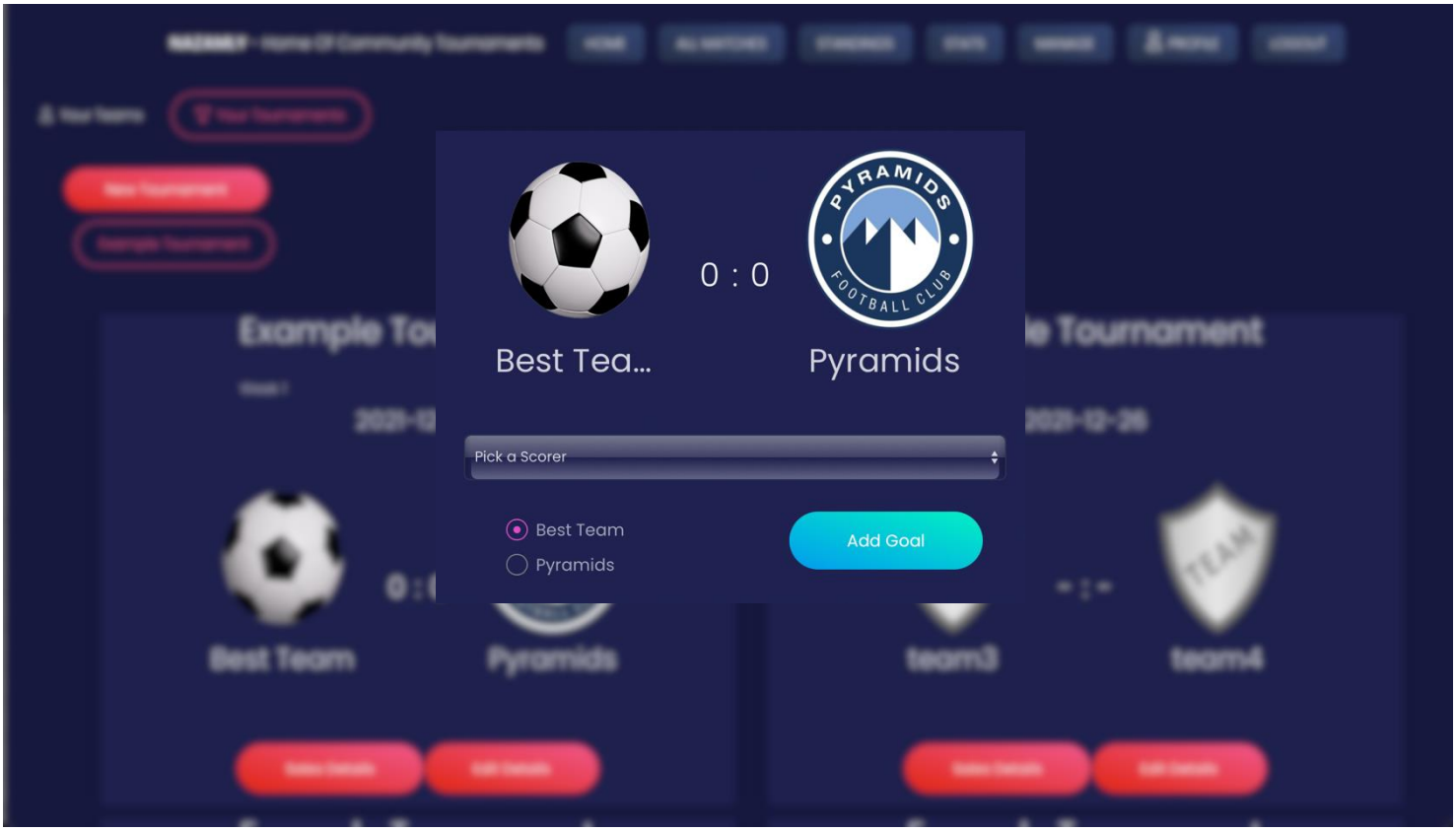


Figure 81: Update Match Result

19. Time Plan

The project will be developed in a waterfall process model, a document driven model, where each phase will be completed to move onto the next phase.

There are 5 phases that will take place. These include an analysis phase, a design phase, a coding and module testing phase, and integration and system testing phase and finally, deployment, delivery, and maintenance phase.

The project will be developed over a 10-week period divided among the phases starting on the 23rd of October 2021 to the 31st of December of the same year.

Details of the time plan is furtherly described in the Gantt chart provided in the figure below.

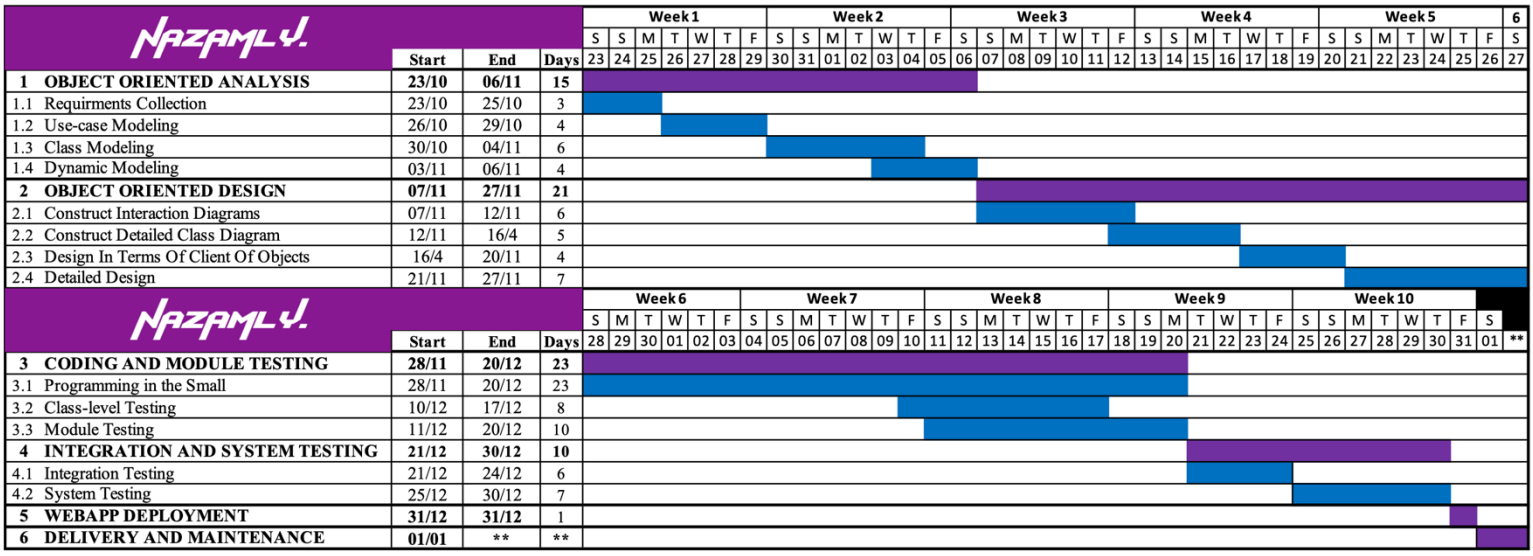


Figure 82: Nazamly Gantt Chart