UNIVERSITÀ DI PISA

University of Pisa

Laurea Magistrale (MSc) in Artificial Intelligence and Data Engineering

Project

Large-Scale and Multi-Structured Databases

FantaBook

Marco Simoni

2022

Index

1 Introduction

FantaBook is an application that was created to create a network for those who play FantaCalcio and to keep track of players' stats. Users can add all of their squads and browse through the squads of other owners. The application provides broad information about each player; it maintains and allows users to search for them using particular filters to narrow the scope of their search. Users can then use a "like" system to convey their feelings about players and eventually add them to their favorites. They can also follow other users to get recommendations for players and squads they might enjoy.

2 Functional Requirements

Standard Users

- Standard Users can search for Players:
  1. By Name
  2. By Position
  3. By Team
- Standard Users can manage Squads:
  1. Standard Users can create a new squad.
  2. Standard Users can add and remove players to their squads.
  3. Standard Users can update the name to their squads.
  4. Standard Users can delete their squads.
  5. Standard Users can follow/unfollow other User's squads.
- Standard Users can search other Users by username and follow/unfollow them.
- Standard Users can browse suggested squads.
- Standard Users can browse suggested players.
- Standard Users can browse best players.
- Standard Users can like Players.
- Standard Users can add players to their "favorite".
- Standard Users can log out.

Admins:

- Admins can do all the operation of a Standard Users.
- Admins can delete a player.
- Admins can promote a Standard User to Admin.
- Admins can access to the statistics of the platform.

3 Non-Functional Requirements

The system needs to have fast response times, in order to make the application enjoyable for users.
The content of the application should be highly available in any moment.
The system needs to be tolerant to data lost and to single point of failure.
The application needs to be user friendly, providing a GUI.

4 Specifications

Actors and a Diagram of a Use Case
The program is intended to be utilized by two actors, according to the software functional requirements:

Standard Users: they have access to all of the application's features, including creating a personalized profile, following other users, following other users' squads, visualizing personalized content based on their Followers on their home page, and searching for players by name, position, or team. Furthermore, they can build new squads to have specific players available at all times based on their mood.

Administrators: they have full access to the application's features, including the ability to delete or promote standard users to admin status, monitor all application stats, such as the total number of users, players, and teams, and do complicated analytics to provide rankings.

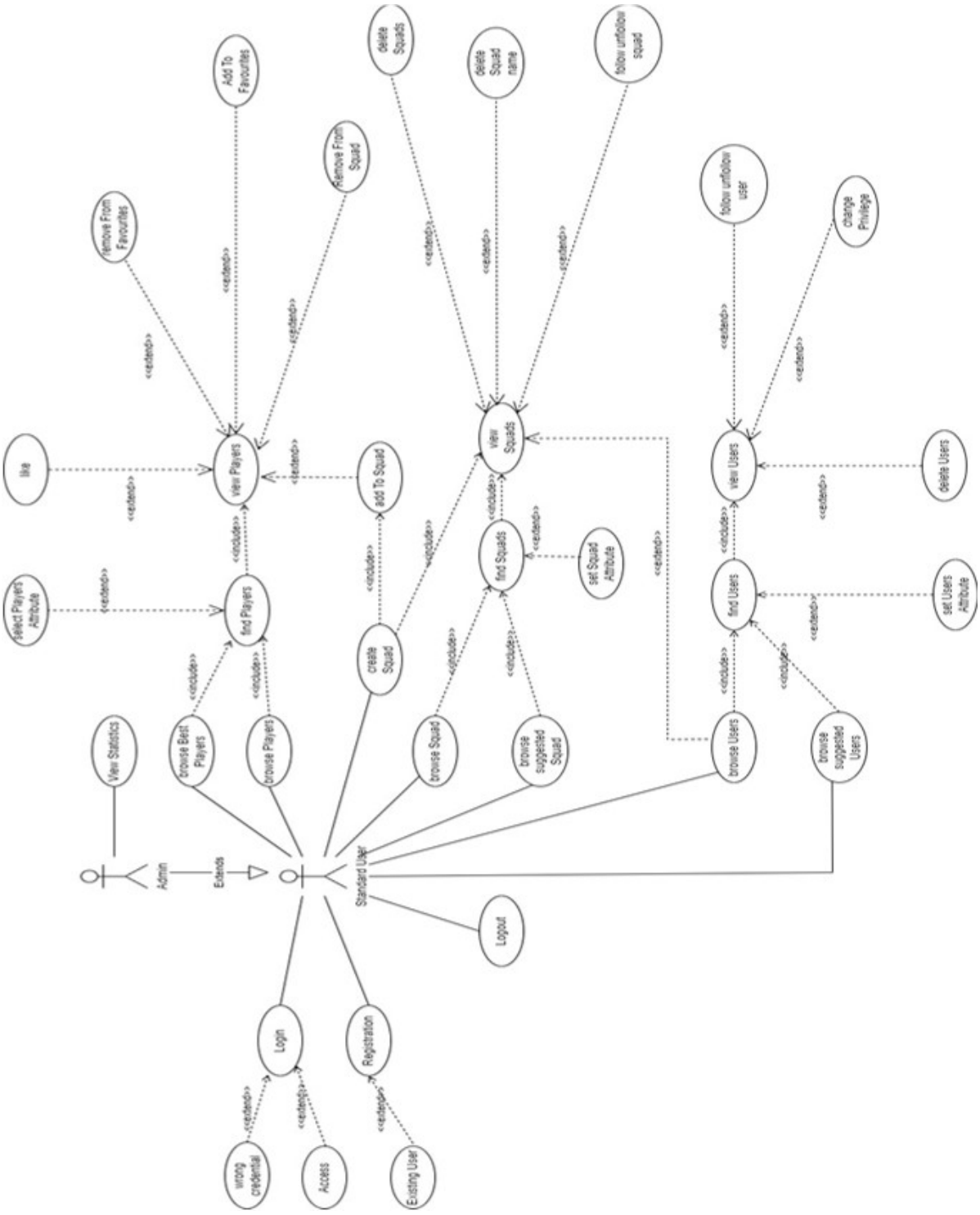


Figure 1: Diagram

5 Architectural Design Software Architecture

The application has been divided into two-tiers according to the Client-Server paradigm, exploiting Java as the core programming language.
Client Side
The client side is divided into:

1. **A Front-End module**, in charge of:
   - providing a GUI based on JavaFX for users to interact with the application.
   - communicating with the underlying middleware to retrieve information obtained by processing data stored on server side.
2. The **Middleware module** communicates with a server side. It includes the logic needed to connect to a MongoDB cluster and a Neo4j database on the server. It also provides the necessary logic to process data retrieved from the database. The server side is composed of three virtual machines that are respectively hosting a MongoDB cluster and a single Neo4j database.

Server Side Server composed of 3 virtual machines which hosts a MongoDB cluster and a single instance of the Neo4j database.

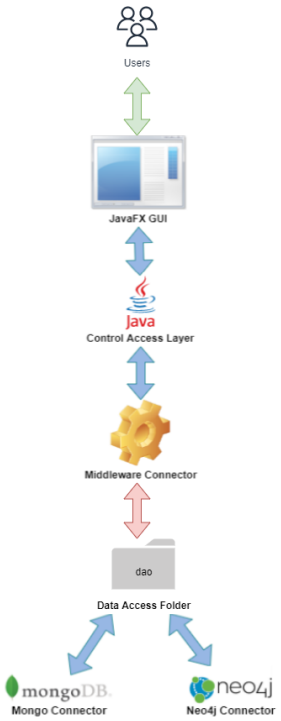


Figure 2: Server Side

6 Dataset Organization and Database Population Player information

The application dataset is manly composed by players' information, that are the core of my application. They have been collected through multiple steps and multiple smyces to respect the variety constraint. Firstly, i have obtained few information using services offered by RAPIDAPI which returns a raw json document for each player requested.
Most of the information have been retrieved performing scraping on "Transfermarket.com". The only attribute that has been added manually corresponds to the number of likes.
Since the application is also a kind of social network, it is important for it to contain a satisfying number of users and squads created by users. For this reason, i populate my databases using a script that randomly creates a high number of users, squads, "follow" and "like" relations among them.
The total amount of memory size is beyond 53MB.

7 MongoDB Design and Implementation

Information about players, users and squads are stored in MongoDB in two different collections.

7.1 User Collection

The document of a user contains all account information as an array of embedded documents describing the squads created by the user; due to the one-to-many relationship between those two entities and to improve the performance of some analytics, it was decided to embed the squad document inside the user collection. Information about players uploaded by the user is stored in each squad; this information is a subset of the total stored for each player, which can be accessed in the associated player record. This redundancy was used to increase read performance by allowing us to provide a quick preview of all players contained inside specified teams without having to immediately retrieve them.

Figure 3: User Collection

7.2 Player Collection


```
_id: "626e96e7a9017e2d071def48"
name: "Cristiano Ronaldo"
position: "Centre-Forward"
goals: 12
assists: "3"
age: " Feb 5, 1985 "
rating: 6
nationality: "Portugal"
likeCount: 850
team: "Manchester United"
league: "Premier League"
RedCard: 1
YellowCard: 3
Appearances: "25"
minutes: "2.009"
Value: "£31.50m"
ShirtNumber: "7"
imageUrl: "https://img.a.transfermarkt.technology/portrait/big/8198-1631656078.jp…"
```


Figure 4: Player Collection

8 CRUD Operations

8.1 Create

Users

```
private void createUserDocument(User user) throws MongoException {
    Document userDoc = user.toBsonDocument();
    //System.out.println(userDoc);

    MongoCollection<Document> userColl = MongoDriver.getInstance().getCollection(Collections.USERS);
    //System.out.println(userColl);
    userColl.insertOne(userDoc);
}
```

Figure 5: Create User Document

Players

```
private void createPlayerDocument(Player player) throws MongoException {

    MongoCollection<Document> playerCollection = MongoDriver.getInstance().getCollection(Collections.PLAYERS);

    Document playerDocument = player.toBsonDocument();

    playerCollection.insertOne(playerDocument);

}
```

Figure 6: Create Player Document

Squads

```
private void createSquadDocument(Squad squad) throws ActionNotCompletedException {
    UserDAOImpl userDAO = new UserDAOImpl();
    userDAO.addSquadToUserDocument(new User(squad.getOwner()), squad);
    System.out.println("\n\nSquad Added To User\n\n");
    addRandomPlayers(squad, numPlayers: 22);
}
```

Figure 7: Create Squad Document

8.2 Read

Get User

```
@Override
public User getUserByUsername(String username)  throws ActionNotCompletedException{
    User user = null;

    try (MongoCursor<Document> cursor =
                 MongoDriver.getInstance().getCollection(Collections.USERS).find(eq( fieldName: "_id", username)).iterator()) {
        if (cursor.hasNext()) {
            user = new User(cursor.next());
            System.out.println(user);
        }
    } catch (MongoException mEx) {
        logger.warn(mEx.getMessage());
        throw new ActionNotCompletedException(mEx);
    }
    return user;
```

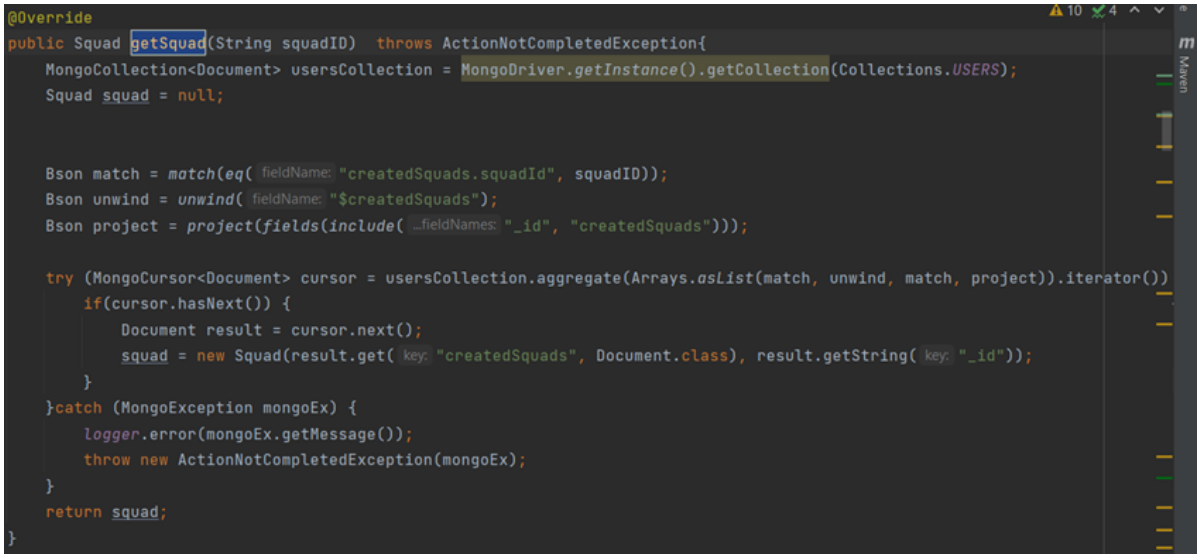
Figure 8: get user by username

Get Player

```
@Override
public Player getPlayerById(String playerID) {

    MongoCollection<Document> playerCollection = MongoDriver.getInstance().getCollection(Collections.PLAYERS);
    Player playerToReturn = null;

    try (MongoCursor<Document> cursor = playerCollection.find(eq( fieldName: "name", playerID)).iterator()) {
        if (cursor.hasNext()) {
            playerToReturn = new Player(cursor.next());
        }
    } catch (MongoException mongoEx) {
        logger.error(mongoEx.getMessage());
    }
    return playerToReturn;
}
```

Figure 9: get player by Id

Get Specific Squad



```
@Override
public Squad getSquad(String squadID)  throws ActionNotCompletedException{
    MongoCollection<Document> usersCollection = MongoDriver.getInstance().getCollection(Collections.USERS);
    Squad squad = null;



    Bson match = match(eq( fieldName: "createdSquads.squadId", squadID));
    Bson unwind = unwind( fieldName: "$createdSquads");
    Bson project = project(fields(include( ...fieldNames: "_id", "createdSquads")));

    try (MongoCursor<Document> cursor = usersCollection.aggregate(Arrays.asList(match, unwind, match, project)).iterator()) {
        if(cursor.hasNext()) {
            Document result = cursor.next();
            squad = new Squad(result.get( key: "createdSquads", Document.class), result.getString( key: "_id"));
        }
    }catch (MongoException mongoEx) {
        logger.error(mongoEx.getMessage());
        throw new ActionNotCompletedException(mongoEx);
    }
    return squad;
}
```

Figure 10: get squad by Id

Get All Squads

```java
@Override
public List<Squad> getAllSquad(User user) throws ActionNotCompletedException, IllegalArgumentException {
    if(user == null) throw new IllegalArgumentException();

    MongoCollection<Document> usersCollection = MongoDriver.getInstance().getCollection(Collections.USERS);
    List<Squad> squads = new ArrayList<>();
    System.out.println(user.getUsername());
    Bson match = match(eq( fieldName: "_id", user.getUsername()));
    Bson unwind = unwind( fieldName: "$createdSquads");

    try (MongoCursor<Document> cursor = usersCollection.aggregate(Arrays.asList(match, unwind)).iterator())
        while(cursor.hasNext()) {
            Document result = cursor.next();
            System.out.println("DOCUMENT RESULT");
            System.out.println(result);
            System.out.println(result.get( key: "createdSquads", Document.class));
            squads.add(new Squad(result.get( key: "createdSquads", Document.class), user.getUsername()));
            System.out.println("SQUADS");
            System.out.println(squads);
        }
```

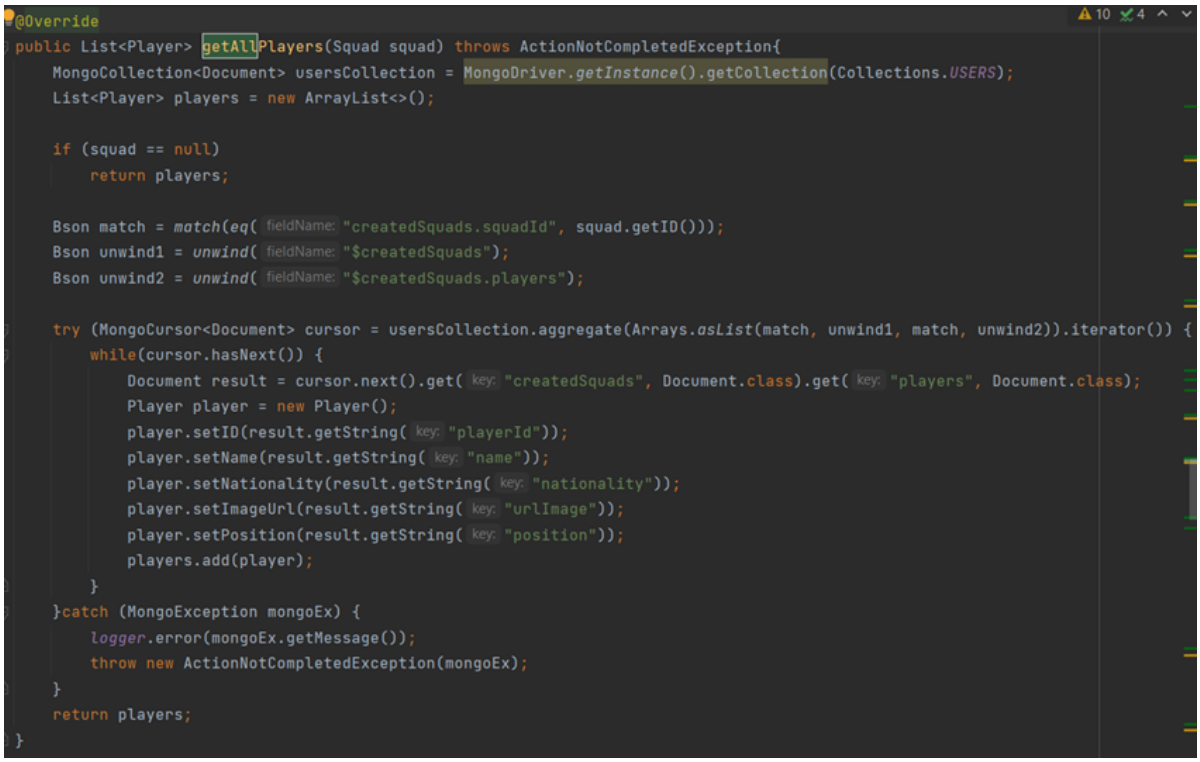
Figure 11: get all squads

Get User By Partial-Username

```java
@VisibleForTesting
List<User> getUserByPartialUsername(String partialUsername, int limitResult) throws ActionNotCompletedException, IllegalArgumentException {
    if(limitResult <= 0) throw new IllegalArgumentException();

    MongoCollection<Document> userCollection = MongoDriver.getInstance().getCollection(Collections.USERS);
    List<User> usersToReturn = new ArrayList<>();

    Bson match = match(regex( fieldName: "_id",  pattern: "(?i)^" + partialUsername + ".*"));

    try (MongoCursor<Document> cursor = userCollection.aggregate(Arrays.asList(match, limit(limitResult))).iterator()) {
        while(cursor.hasNext()) {
            usersToReturn.add(new User(cursor.next()));
        }
    } catch (MongoException mongoEx) {
        logger.error(mongoEx.getMessage());
        throw new ActionNotCompletedException(mongoEx);
    }
    return usersToReturn;
}
```

Figure 12: get user by partial username

Get All Player



```java
@Override
public List<Player> getAllPlayers(Squad squad) throws ActionNotCompletedException{
    MongoCollection<Document> usersCollection = MongoDriver.getInstance().getCollection(Collections.USERS);
    List<Player> players = new ArrayList<>();

    if (squad == null)
        return players;

    Bson match = match(eq( fieldName: "createdSquads.squadId", squad.getID()));
    Bson unwind1 = unwind( fieldName: "$createdSquads");
    Bson unwind2 = unwind( fieldName: "$createdSquads.players");

    try (MongoCursor<Document> cursor = usersCollection.aggregate(Arrays.asList(match, unwind1, match, unwind2)).iterator()) {
        while(cursor.hasNext()) {
            Document result = cursor.next().get( key: "createdSquads", Document.class).get( key: "players", Document.class);
            Player player = new Player();
            player.setID(result.getString( key: "playerId"));
            player.setName(result.getString( key: "name"));
            player.setNationality(result.getString( key: "nationality"));
            player.setImageUrl(result.getString( key: "urlImage"));
            player.setPosition(result.getString( key: "position"));
            players.add(player);
        }
    }catch (MongoException mongoEx) {
        logger.error(mongoEx.getMessage());
        throw new ActionNotCompletedException(mongoEx);
    }
    return players;
}
```

Figure 13: get all player

Filter Player

```java
@VisibleForTesting
public List<Player> filterPlayer(String partialInput, int maxNumber, String attributeField) throws ActionNotCompletedException {

    if (attributeField == null || maxNumber <= 0)
        throw new IllegalArgumentException();

    MongoCollection<Document> playerCollection = MongoDriver.getInstance().getCollection(Collections.PLAYERS);
    List<Player> playersToReturn = new ArrayList<>();

    String capitalPartialInput = partialInput.substring(0, 1).toUpperCase() + partialInput.substring(1);

    Bson match = match(regex(attributeField, pattern: "^" + capitalPartialInput + ".*"));
    Bson sortLike = sort(descending( ...fieldNames: "likeCount"));
    try (MongoCursor<Document> cursor = playerCollection.aggregate(Arrays.asList(match, sortLike, limit(maxNumber))).iterator()) {
        while (cursor.hasNext()) {
            playersToReturn.add(new Player(cursor.next()));
        }
    } catch (MongoException mongoEx) {
        logger.error(mongoEx.getMessage());
        throw new ActionNotCompletedException(mongoEx);
    }
    return playersToReturn;
}
```

Figure 14: filter players

Check User Password

```java
@Override
public boolean checkUserPassword(String username, String password) {
    try (MongoCursor<Document> cursor = MongoDriver.getInstance().getCollection(Collections.USERS)
            .find(eq( fieldName: "_id", username)).iterator()) {
        if (cursor.hasNext())
            if(password.equals(cursor.next().get("password").toString()))
                return true;
    } catch (MongoException mEx) {
        return false;
    }
    return false;
}
```

Figure 15: check user password

8.3 Update

Add Player

```java
@Override
public void addPlayer(Squad squad, Player player)  throws ActionNotCompletedException{
    try {
        MongoCollection<Document> usersCollection = MongoDriver.getInstance().getCollection(Collections.USERS);

        Document playerDocument = new Document("playerId", player.getID())
                .append("name", player.getName())
                .append("position", player.getPosition())
                .append("nation", player.getNationality())
                .append("team", player.getTeam())
                .append("age", player.getAge());
        if (player.getImageUrl() != null)
            playerDocument.append("urlImage", player.getImageUrl());

        Bson find = eq( fieldName: "createdSquads.squadId", squad.getID());
        Bson query = push( fieldName: "createdSquads.$.players", playerDocument);
        usersCollection.updateOne(find, query);
        logger.info("Added player " + player.getID() + " to squad " + squad.getID());
    } catch (MongoException mongoEx) {
        logger.error(mongoEx.getMessage());
        throw new ActionNotCompletedException(mongoEx);
    }

}
```

Figure 16: add players

When a user likes a player, the player's document must be changed in order to ensure redundancy. In this case, the function to utilize is increment like Count (there is an equivalent function when a like is removed).

Like Count

```java
@Override
public void incrementLikeCount(Player player) throws ActionNotCompletedException {

    if (player == null || player.getID() == null)
        throw new IllegalArgumentException();

    MongoCollection<Document> playerCollection = MongoDriver.getInstance().getCollection(Collections.PLAYERS);
    try {
        playerCollection.updateOne(eq( fieldName: "_id", player.getID()), inc( fieldName: "likeCount",  number: 1));
        player.setLikeCount(player.getLikeCount() + 1);
    } catch (MongoException mongoEx) {
        logger.error(mongoEx.getMessage());
        throw new ActionNotCompletedException(mongoEx);
    }

}
```

Figure 17: increment like count

User Privilege Level Modification

An admin can update the privilege level of a user.

```java
@Override
public void updateUserPrivilegeLevel(User user, PrivilegeLevel newPrivLevel) throws ActionNotCompletedException, IllegalArgumentException {
    if(user == null || newPrivLevel == null)
        throw new IllegalArgumentException();

    user.setPrivilegeLevel(newPrivLevel);
    try {
        updateUserDocument(user);
        logger.info("Updated privilege level of user <" +user.getUsername()+ "> : new level <" +user.getPrivilegeLevel()+ ">");
    } catch (MongoException mEx) {
        logger.warn(mEx.getMessage());
        throw new ActionNotCompletedException(mEx);
    }
}
```

Figure 18: update user privilege level

Update User Document

```
private void updateUserDocument(User user) throws MongoException {
    MongoCollection<Document> userColl = MongoDriver.getInstance().getCollection(Collections.USERS);
    userColl.updateOne(
            eq( fieldName: "_id", user.getUsername()),
            combine(
                    set("firstName", user.getFirstName()),
                    set("lastName", user.getLastName()),
                    set("age", user.getAge()),
                    set("privilegeLevel", user.getPrivilegeLevel().toString())
            ));
}
```

Figure 19: update user document

8.4 Delete

Delete User Document

```
@Override
public void deleteUserDocument(User user) throws MongoException {
    MongoCollection<Document> userColl = MongoDriver.getInstance().getCollection(Collections.USERS);
    userColl.deleteOne(eq( fieldName: "_id", user.getUsername()));
}
```

Figure 20: delete user document

Delete Squad Document

```
@Override
public void deleteSquadDocument(Squad squad) throws MongoException{
    Bson find = eq( fieldName: "createdSquads.squadId",squad.getID());
    MongoCollection<Document> usersCollection = MongoDriver.getInstance().getCollection(Collections.USERS);
    usersCollection.updateOne(find, pull( fieldName: "createdSquads", eq( fieldName: "squadId", squad.getID())));
}
```

Figure 21: delete squad document

9 Queries Analysis and MongoDB Connection String

Write operations are performed less frequently and are generally basic (just one document is created or altered at a time), whereas Read operations are

performed more frequently and include a larger number of documents.
Furthermore, the system must respond quickly and be available at all times, so it is tuned as follows:

- **Write with concern: 1**. This enables the system to run as quickly as possible during write operations, needing the use of an Eventual Consistency paradigm.
- **Read Preferences**: Nearest for further information. To get the fastest response, read operations are conducted on the node with the lowest network latency.

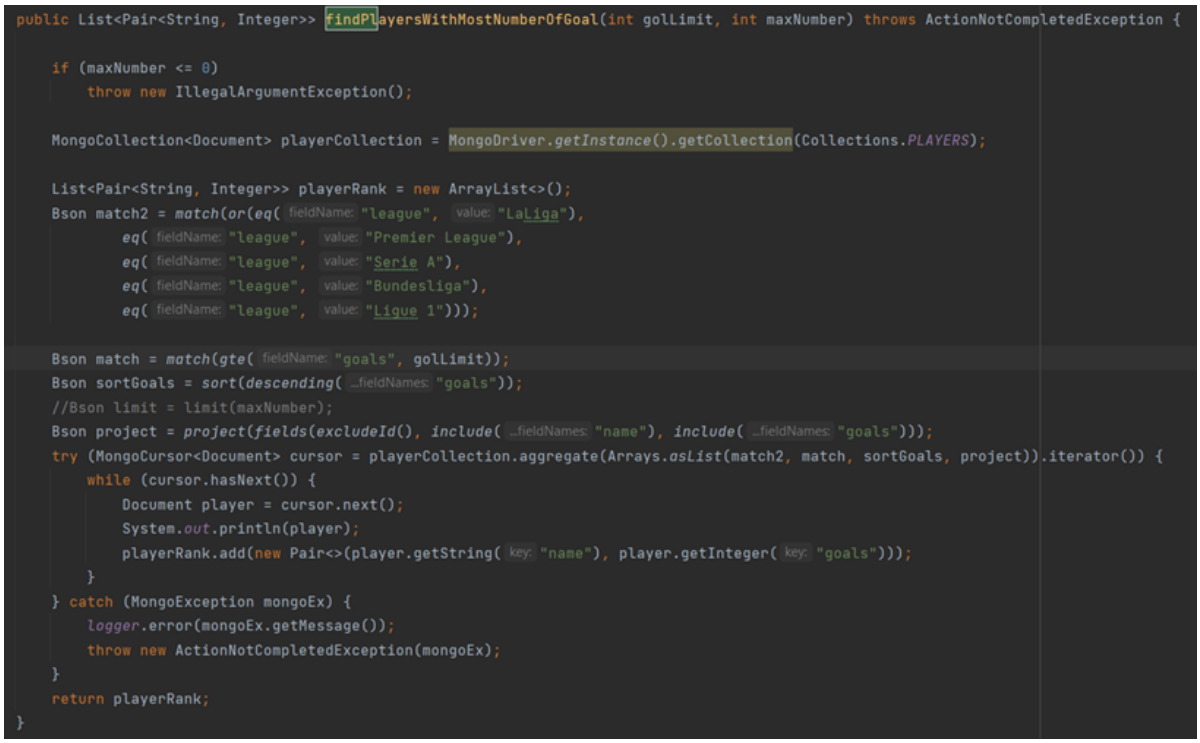
The MongoDB Java Driver uses the connection string:

mongodb:/172.16.4.223:27020,172.16.4.224:27020,172.16.4.225:27020/w=1&readPreference=nearest

10 Analytics and Statistics

In the following the fmy pipelines aggregations used to extract interesting information from data are described.

Players With the Most Number Of Goals



```
public List<Pair<String, Integer>> findPlayersWithMostNumberOfGoal(int golLimit, int maxNumber) throws ActionNotCompletedException {

    if (maxNumber <= 0)
        throw new IllegalArgumentException();

    MongoCollection<Document> playerCollection = MongoDriver.getInstance().getCollection(Collections.PLAYERS);

    List<Pair<String, Integer>> playerRank = new ArrayList<>();
    Bson match2 = match(or(eq( fieldName: "league", value: "LaLiga"),
            eq( fieldName: "league", value: "Premier League"),
            eq( fieldName: "league", value: "Serie A"),
            eq( fieldName: "league", value: "Bundesliga"),
            eq( fieldName: "league", value: "Ligue 1")));

    Bson match = match(gte( fieldName: "goals", golLimit));
    Bson sortGoals = sort(descending( ...fieldNames: "goals"));
    //Bson limit = limit(maxNumber);
    Bson project = project(fields(excludeId(), include( ...fieldNames: "name"), include( ...fieldNames: "goals")));
    try (MongoCursor<Document> cursor = playerCollection.aggregate(Arrays.asList(match2, match, sortGoals, project)).iterator()) {
        while (cursor.hasNext()) {
            Document player = cursor.next();
            System.out.println(player);
            playerRank.add(new Pair<>(player.getString( key: "name"), player.getInteger( key: "goals")));
        }
    } catch (MongoException mongoEx) {
        logger.error(mongoEx.getMessage());
        throw new ActionNotCompletedException(mongoEx);
    }
    return playerRank;
}
```

Figure 22: find players with the most number of goals

```
MongoCollection<Document> playerCollection = MongoDriver.getInstance().
    getCollection(Collections.PLAYERS);

        List<Pair<String, Integer>> playerRank = new ArrayList<>();
        Bson match2 = match(or(eq("league", "LaLiga"),
                eq("league", "Premier League"),
                eq("league", "Serie A"),
                eq("league", "Bundesliga"),
                eq("league", "Ligue 1")));

        Bson match = match(gte("goals", golLimit));
        Bson sortGoals = sort(descending("goals"));
```

Favourite Teams

```
public List<Pair<String, Integer>> getFavouriteTeam(int lim) throws ActionNotCompletedException, IllegalArgumentException{
    //if(team <= 0) throw new IllegalArgumentException();

    MongoCollection<Document> usersCollection = MongoDriver.getInstance().getCollection(Collections.USERS);
    List<Pair<String, Integer>> result = new ArrayList<>();
    Bson unwind1 = unwind( fieldName: "$createdSquads");
    Bson unwind2 = unwind( fieldName: "$createdSquads.players");
    Bson group = Document.parse("{$group: {" +
            "_id: \"$createdSquads.players.team\"," +
            "totalPlayers: { $sum: 1}" +
            "}}");
    Bson sort = sort(descending( ...fieldNames: "totalPlayers"));
    Bson limit = limit(lim);
    try (MongoCursor<Document> cursor = usersCollection.aggregate(Arrays.asList(unwind1, unwind2, group, sort, limit)).iterator()) {
        while(cursor.hasNext()) {
            Document team = cursor.next();
            System.out.println(team);
            result.add(new Pair<>(team.getString( key: "_id"), team.getInteger( key: "totalPlayers")));
        }
    } catch (MongoException mEx) {
        logger.error(mEx.getMessage());
        throw new ActionNotCompletedException(mEx);
    }
    return result;
}
```

Figure 23: get favourite teams

Favourite Player Per Position

```
public List<Pair<String, Pair<Player, Integer>>> findTopRatedPlayerPerPosition() throws ActionNotCompletedException {
    MongoCollection<Document> playerCollection = MongoDriver.getInstance().getCollection(Collections.PLAYERS);
    List<Pair<String, Pair<Player, Integer>>> topPlayers = new ArrayList<>();
    Bson project = project(fields(excludeId(), include( ...fieldNames: "imageUrl"), include( ...fieldNames: "position"), include( ...fieldNames: "name"), include( ...fieldNames: "likeCount")))
    Bson group1 = Document.parse("{$group:{_id : {name: \"$name\", position: \"$position\", imageUrl:\"$imageUrl\", likeCount: \"$likeCount\"}}}");
    Bson sortRate = sort(ascending( ...fieldNames: "rating"));
    Bson group2 = Document.parse("{$group:{" +
            "_id: \"$_id.position\"," +
            "topPlayerName: {$last: \"$_id.name\"}," +
            "topPlayerImage: {$last: \"$_id.imageUrl\"}," +
            "likeCount:{$last: \"$_id.likeCount\"}" +
            "}}");
    Bson sortId = sort(ascending( ...fieldNames: "_id"));
    Bson project2 = project(fields(excludeId(), computed( fieldName: "position",  expression: "$_id"), include( ...fieldNames: "topPlayerImage"), include( ...fieldNames: "topPlayerName"),


    try (MongoCursor<Document> cursor = playerCollection.aggregate(Arrays.asList(
            project,
            group1,
            sortRate,
            group2,
            sortId,
            project2
    )).iterator()){
        while(cursor.hasNext()) {
            Document record = cursor.next();
            System.out.println(record);
```

Figure 24: get favourite player per position

11 Index Analysis

League Index

Query : db.Players.find({league:{$regex:/^P/i}}).explain("executionStats")

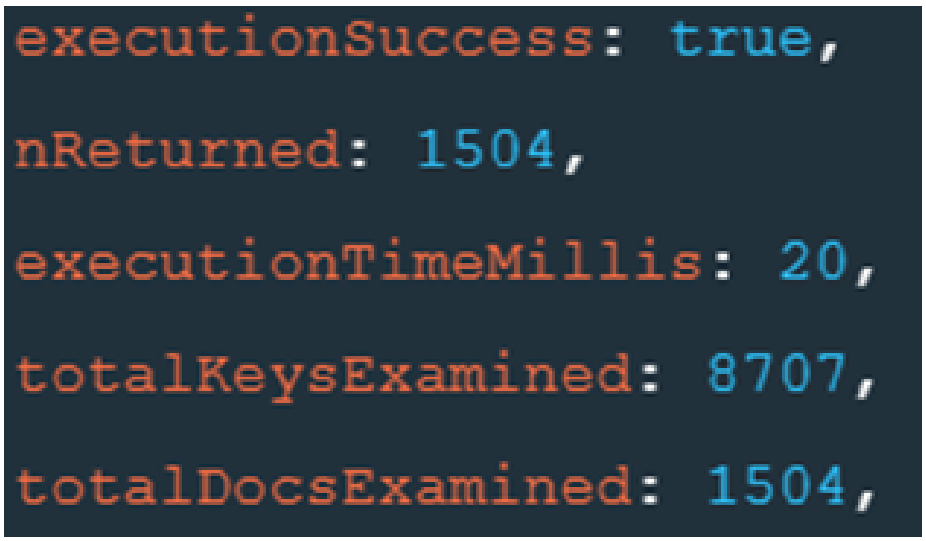


Figure 25: with Index

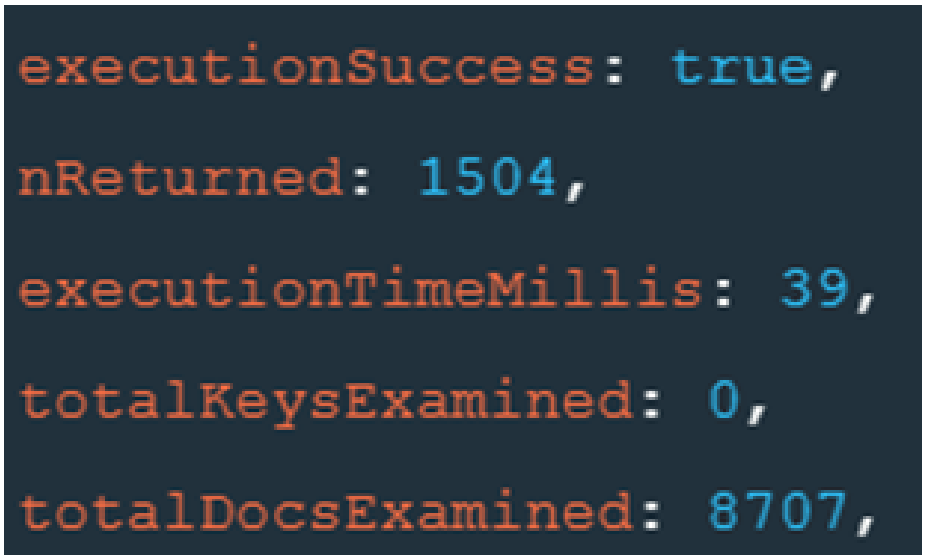


Figure 26: without Index

Shirt-Number Index

```
Query : db.Players.find({ShirtNumber: 10 }).explain("executionStats")
```

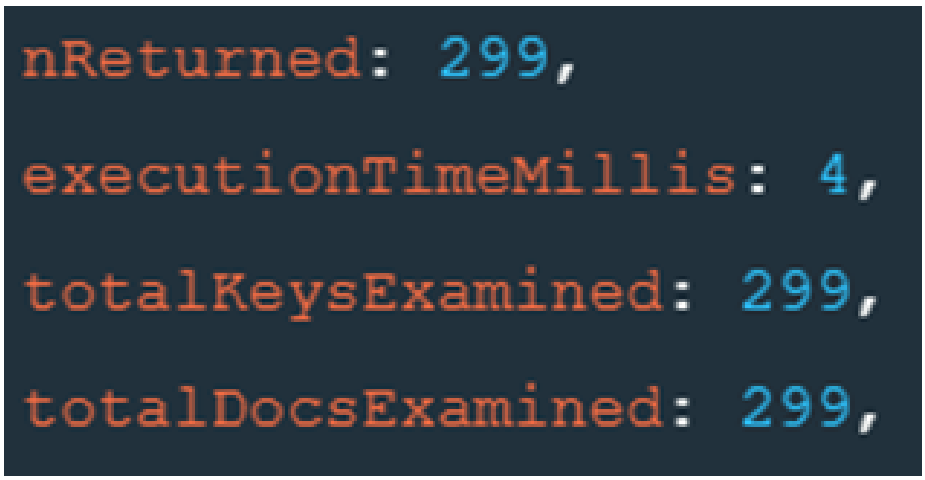


Figure 27: with Index

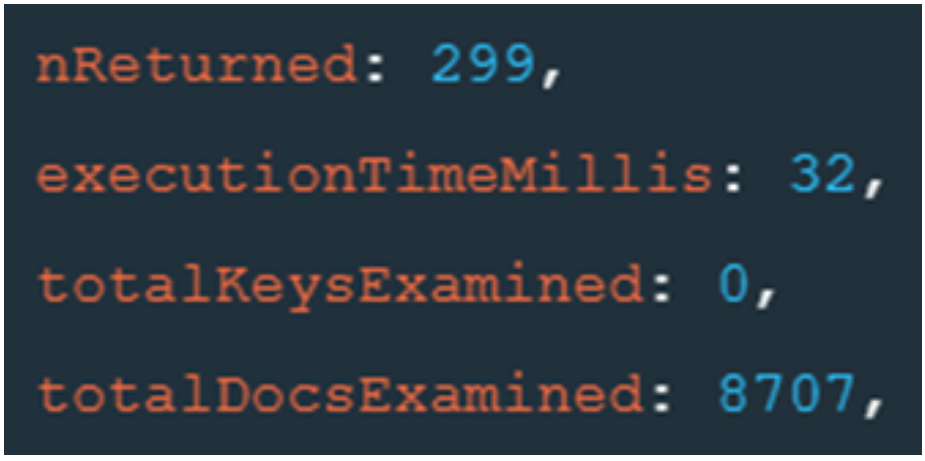


Figure 28: without Index

12 NEO4J Design And Implementation

Each main entity in my program has its own node, including (:Player), (:User), and (:Squad).
They just have the characteristics that are required to show a preview of the entity.

12.1 Relations

1) USER – FOLLOW USER –> USER

When a user adds another to his list of Followed users, this relationship is created. It's useful to get a list of recommended squads and users.

2)USER – FOLLOW SQUAD–> SQUAD

When a user saves another user's squad to his squad list, this relationship is created. It's useful to be able to retrieve suggested squads and have them available on the user's profile at all times. It's crucial to remember that it's just a link to the squad, so if the creator changes or removes it, it will have an impact.

3) USER – LIKES –> PLAYER

When a user likes a player, this relationship is created. It's important for retrieving popular players and the second tier of suggested users.

12.2 Queries Implementation

12.2.1 CRUD Operations

Create User Node

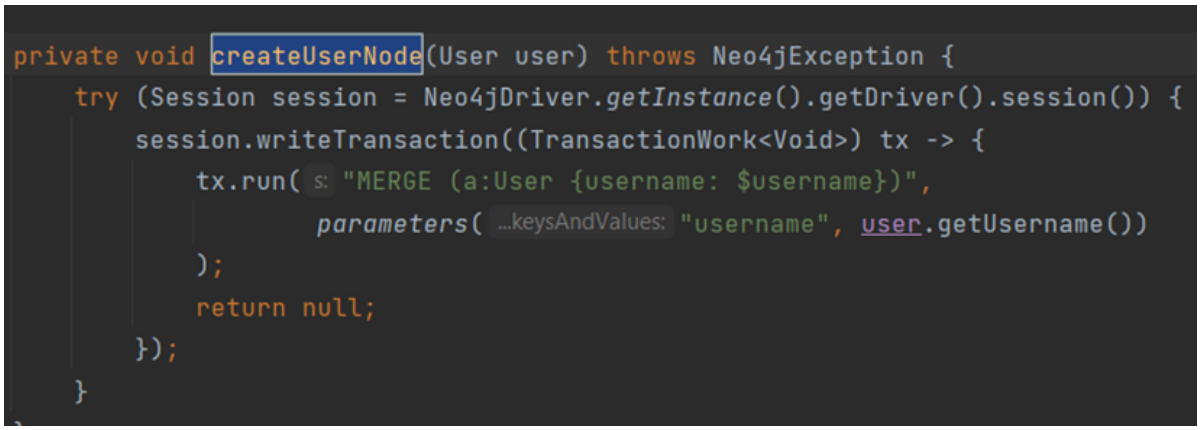


Figure 29: create user node

"MERGE (a:User {username: $username})"

Player Node

```
private void createPlayerNode(Player player) throws Neo4jException {

    try (Session session = Neo4jDriver.getInstance().getDriver().session()) {
        session.writeTransaction((TransactionWork<Void>) tx -> {

            tx.run( s: "CREATE (p:Player {playerId: $playerId, name: $name, team: $team," +
                            " imageUrl: $imageUrl, position: $position})",
                    parameters( ...keysAndValues: "playerId", player.getID(), "name", player.getName(),
                            "team", player.getTeam(), "imageUrl", player.getImageUrl(), "position", player.getPosition()));
            return null;
        });
    }
```

Figure 30: create player node

"CREATE (p:Player {playerId: $playerId, name: $name, team:$team," imageUrl: $imageUrl, position: $position})"

Squad Node

```
private void createSquadNode(Squad squad) {
    try ( Session session = Neo4jDriver.getInstance().getDriver().session() )
    {
        session.writeTransaction((TransactionWork<Void>) tx -> {
            tx.run( s: "CREATE (p:Squad {squadId: $squadId, name: $name, urlImage: $urlImage})",
                    parameters( ...keysAndValues: "squadId", squad.getID(), "name", squad.getName(),
                            "urlImage", squad.getUrlImage()) );
            return null;
        });
    }
}
```

Figure 31: create squad node

"CREATE (p:Squad {squadId: $squadId, name: $name, urlImage: $urlImage})"

Follow User

```java
@Override
public void followUser(User userFollowing, User userFollowed) throws ActionNotCompletedException, IllegalArgumentException {
    if(userFollowing == null || userFollowed == null) throw new IllegalArgumentException();

    try (Session session = Neo4jDriver.getInstance().getDriver().session()) {
        session.writeTransaction((TransactionWork<Void>) tx -> {
            tx.run( s: "MATCH (following:User { username: $following }) "
                    + "MATCH (followed:User { username: $followed }) "
                    + "WHERE following <> followed "
                    + "MERGE (following)-[:FOLLOWS_USER]->(followed)",
                    parameters( ...keysAndValues: "following", userFollowing.getUsername(), "followed", userFollowed.getUsername())
            );
            return null;
        });
        logger.info("User <" + userFollowing.getUsername() + "> follows user <" + userFollowed.getUsername() + ">");
    } catch (Neo4jException n4jEx) {
        logger.error(n4jEx.getMessage());
        throw new ActionNotCompletedException(n4jEx);
    }
}
```

Figure 32: follow user

”MATCH (following:User { username: $following })
MATCH (Followed:User { username: $Followed })
WHERE following <> Followed MERGE (following)-[:FOLLOWS_USER]->(Followed)”

Like Player

```
@Override
public void likePlayer(User user, Player player) throws ActionNotCompletedException, IllegalArgumentException {
    if(user == null || player == null) throw new IllegalArgumentException();

    try (Session session = Neo4jDriver.getInstance().getDriver().session()) {
        session.writeTransaction((TransactionWork<Void>) tx -> {
            tx.run( s: "MATCH (u:User { username: $username }) "
                        + "MATCH (s:Player { playerId: $playerId }) "
                        + "MERGE (u)-[:LIKES]->(s)",
                    parameters( ...keysAndValues: "username", user.getUsername(), "playerId", player.getID())
            );
            return null;
        });
        logger.info("User <" + user.getUsername() + "> likes player <" + player.getID() + ">");

        // Handle the redundancy $likeCount
        PlayerDAOImpl playerDAO = new PlayerDAOImpl();
        playerDAO.incrementLikeCount(player);
    } catch (Neo4jException n4jEx) {
        logger.error(n4jEx.getMessage());
        throw new ActionNotCompletedException(n4jEx);
    }
}
```

Figure 33: like player

”MATCH (u:User { username: $username }) MATCH (s:Player { playerId: $playerId })

Follow Squad

```
@Override
public void followSquad(User user, Squad squad) throws ActionNotCompletedException, IllegalArgumentException {
    if(user == null || squad == null) throw new IllegalArgumentException();

    try (Session session = Neo4jDriver.getInstance().getDriver().session()) {
        session.writeTransaction((TransactionWork<Void>) tx -> {
            tx.run( s: "MATCH (following:User { username: $following }) "
                    + "MATCH (followed:Squad { squadId: $followed }) "
                    + "MERGE (following)-[:FOLLOWS_squad]->(followed)",
                    parameters( ...keysAndValues: "following", user.getUsername(), "followed", squad.getID())
            );
            return null;
        });
        logger.info("User <" +user.getUsername()+ "> follows squad <" + squad.getID()+ ">");
    } catch (Neo4jException n4jEx) {
        logger.error(n4jEx.getMessage());
        throw new ActionNotCompletedException(n4jEx);
    }
}
```

Figure 34: follow squad

”MATCH (following:User { username: $following }) MATCH (Followed:Squad { squadId

12.3 Read

All Squads

```
public int getTotalSquads() {
    try (Session session = Neo4jDriver.getInstance().getDriver().session())
    {
        return session.readTransaction((TransactionWork<Integer>) tx -> {

            Result result = tx.run( s: "MATCH (:Squad) RETURN COUNT(*) AS NUM")
            if(result.hasNext())
                return result.next().get("NUM").asInt();
            else
                return -1;
        });

    }catch (Neo4jException neo4){
        neo4.printStackTrace();
        return -1;
    }
}
```

Figure 35: get all squads

Total Players

```java
@Override
public int getTotalPlayers() {
    try (Session session = Neo4jDriver.getInstance().getDriver().session()) {
        return session.readTransaction(tx -> {

            Result result = tx.run( s: "MATCH (:Player) RETURN COUNT(*) AS NUM");
            if (result.hasNext())
                return result.next().get("NUM").asInt();
            else
                return -1;
        });

    } catch (Neo4jException neo4) {
        neo4.printStackTrace();
        return -1;
    }
}
```

Figure 36: get total players

Total Users

```
@Override
public int getTotalUsers() {
    try (Session session = Neo4jDriver.getInstance().getDriver().session())
    {
        return session.readTransaction((TransactionWork<Integer>) tx -> {

            Result result = tx.run( s: "MATCH (:User) RETURN COUNT(*) AS NUM");
            if(result.hasNext())
                return result.next().get("NUM").asInt();
            else
                return -1;
        });

    }catch (Neo4jException n4jEx){
        logger.error(n4jEx.getMessage());
        return -1;
    }
}
```

Figure 37: get total users

Followed Squad

```
@Override
public List<Squad> getFollowedSquad(User user) throws ActionNotCompletedException, IllegalArgumentException {
    if(user == null) throw new IllegalArgumentException();

    try( Session session = Neo4jDriver.getInstance().getDriver().session()) {
        return session.readTransaction(tx -> {
            Result result = tx.run(
                    s: "MATCH (:User {username: $username})-[:FOLLOWS_squad]->(squad:Squad) RETURN squad ",
                    parameters( ...keysAndValues: "username", user.getUsername())
            );
            List<Squad> tmpList = new ArrayList<>();
            while ((result.hasNext())) {
                Squad squad = new Squad(result.next().get("squad"));
                squad.setOwner(user.getUsername());
                tmpList.add(squad);
            }
            return tmpList;
        });
    } catch (Neo4jException n4jEx) {
        logger.error(n4jEx.getMessage());
        throw new ActionNotCompletedException(n4jEx);
    }
```

Figure 38: get followed squad

Followed Users

```java
@Override
public List<User> getFollowedUsers(User user) throws ActionNotCompletedException, IllegalArgumentException {
    if(user == null) throw new IllegalArgumentException();

    try( Session session = Neo4jDriver.getInstance().getDriver().session()) {
        return session.readTransaction((TransactionWork<List<User>>) tx -> {
            Result result = tx.run(
                    s: "MATCH (:User {username: $username})-[:FOLLOWS_USER]->(followedUser:User) RETURN follo
                    parameters( ...keysAndValues: "username", user.getUsername())
            );
            List<User> tmpList = new ArrayList<>();
            while ((result.hasNext())) {
                User followedUser = new User(result.next().get("followedUser"));
                tmpList.add(followedUser);
            }
            return tmpList;
        });
    } catch (Neo4jException n4jEx) {
        logger.error(n4jEx.getMessage());
        throw new ActionNotCompletedException(n4jEx);
    }
}
```

Figure 39: get Followed users

Graph

Figure 40: graph representation

User - Like - Player

```java
public boolean userLikesPlayer(User user, Player player) throws IllegalArgumentException {
    if(user == null || player == null) throw new IllegalArgumentException();

    try( Session session = Neo4jDriver.getInstance().getDriver().session()) {
        return session.readTransaction((TransactionWork<Boolean>) tx -> {
            Result result = tx.run(
                    s: "MATCH (:User { username: $username })-[l:LIKES]->(:Player { playerId: $playerId }) "
                            + "RETURN count(*) AS Times",
                    parameters( ...keysAndValues: "username", user.getUsername(), "playerId", player.getID())
            );
            if ((result.hasNext())) {
                int times = result.next().get("Times").asInt();
                return times > 0;
            }
            return false;
        });
    } catch (Exception e) {
        logger.warn(e.getMessage());
        return false;
    }
}
```

Figure 41: user likes player

Followers

```java
@Override
public List<User> getFollowers(User user) throws ActionNotCompletedException, IllegalArgumentException {
    if(user == null) throw new IllegalArgumentException();

    try( Session session = Neo4jDriver.getInstance().getDriver().session()) {
        return session.readTransaction(tx -> {
            Result result = tx.run(
                    s: "MATCH (followingUser:User)-[:FOLLOWS_USER]->(:User {username: $username}) RETURN followingUser ",
                    parameters( ...keysAndValues: "username", user.getUsername())
            );
            List<User> tmpList = new ArrayList<>();
            while ((result.hasNext())) {
                User followingUser = new User(result.next().get("followingUser"));
                tmpList.add(followingUser);
            }
            return tmpList;
        });
    } catch (Neo4jException n4jEx) {
        logger.error(n4jEx.getMessage());
        throw new ActionNotCompletedException(n4jEx);
    }
}
```

Figure 42: get Followers

Following Squad

```java
@Override
public boolean isFollowingSquad(User user, Squad squad) throws IllegalArgumentException {
    if(user == null || squad == null) throw new IllegalArgumentException();

    try( Session session = Neo4jDriver.getInstance().getDriver().session()) {
        return session.readTransaction(tx -> {
            Result result = tx.run(
                    s: "MATCH (:User { username: $username })-[f:FOLLOWS_squad]->(:squad { squadId: $squadId }) "
                            + "RETURN count(*) AS Times",
                    parameters( ...keysAndValues: "username", user.getUsername(), "squadId", squad.getID())
            );
            if ((result.hasNext())) {
                int times = result.next().get("Times").asInt();
                return times > 0;
            }
            return false;
        });
    } catch (Exception e) {
        logger.warn(e.getMessage());
        return false;
    }
}
```

Figure 43: is Following squad

Followed By

```
@Override
public boolean isFollowedBy(User followed, User following) throws IllegalArgumentException {
    if(followed == null || following == null) throw new IllegalArgumentException();

    try( Session session = Neo4jDriver.getInstance().getDriver().session()) {
        return session.readTransaction(tx -> {
            Result result = tx.run(
                    s: "MATCH (:User { username: $following })-[f:FOLLOWS_USER]->(:User { username: $followed }) "
                            + "RETURN count(*) AS Times",
                    parameters( ...keysAndValues: "following", following.getUsername(), "followed", followed.getUsername())
            );
            if ((result.hasNext())) {
                int times = result.next().get("Times").asInt();
                return times > 0;
            }
            return false;
        });
    } catch (Exception e) {
        logger.warn(e.getMessage());
        return false;
    }
}
```

Figure 44: Followed by

Check User likes Player

```java
@Override
public boolean userLikesPlayer(User user, Player player) throws IllegalArgumentException {
    if(user == null || player == null) throw new IllegalArgumentException();

    try( Session session = Neo4jDriver.getInstance().getDriver().session()) {
        return session.readTransaction((TransactionWork<Boolean>) tx -> {
            Result result = tx.run(
                    s: "MATCH (:User { username: $username })-[l:LIKES]->(:Player { playerId: $playerId }) "
                            + "RETURN count(*) AS Times",
                    parameters( ...keysAndValues: "username", user.getUsername(), "playerId", player.getID())
            );
            if ((result.hasNext())) {
                int times = result.next().get("Times").asInt();
                return times > 0;
            }
            return false;
        });
    } catch (Exception e) {
        logger.warn(e.getMessage());
        return false;
    }
}
```

Figure 45: check if a user likes a player

12.4 Update

Change Squad Node

```java
private void updateSquadNameNode(Squad squad, String newName) {
    try ( Session session = Neo4jDriver.getInstance().getDriver().session() )
    {
        session.writeTransaction((TransactionWork<Void>) tx -> {
            tx.run( s: "MATCH (p:Squad { squadId: $squadId }) SET p.name = $newName",
                    parameters( ...keysAndValues: "squadId", squad.getID(), "newName", newName) );
            return null;
        });
    }
}
```

Figure 46: update squad node

12.5 Delete

Delete Like

```java
@Override
public void deleteLike(User user, Player player) throws ActionNotCompletedException, IllegalArgumentException {
    if(user == null || player == null) throw new IllegalArgumentException();

    try (Session session = Neo4jDriver.getInstance().getDriver().session()) {
        session.writeTransaction((TransactionWork<Void>) tx -> {
            tx.run( s: "MATCH (:User { username: $username })-[l:LIKES]->(:Player { playerId: $playerId }) "
                        + "DELETE l",
                    parameters( ...keysAndValues: "username", user.getUsername(), "playerId", player.getID())
            );
            return null;
        });
        logger.info("Deleted user <" + user.getUsername() + "> likes player <" + player.getID() + ">");

        // Handle the redundancy $likeCount
        PlayerDAOImpl playerDAO = new PlayerDAOImpl();
        playerDAO.decrementLikeCount(player);
    } catch (Neo4jException n4jEx) {
        logger.error(n4jEx.getMessage());
        throw new ActionNotCompletedException(n4jEx);
    }
}
```

Figure 47: delete like

Delete User Node

```java
public void deleteUserNode(User user) throws Neo4jException {
    try (Session session = Neo4jDriver.getInstance().getDriver().session())
    {
        session.writeTransaction((TransactionWork<Void>) tx -> {
            tx.run(
                    s: "MATCH (a:User {username: $username})"
                            + "DETACH DELETE a",
                    parameters( ...keysAndValues: "username", user.getUsername()));
            return null;
        });
    }
}
```

Figure 48: delete user node

Unfollow User

```
public void unfollowUser(User userFollowing, User userFollowed) throws ActionNotCompletedException, IllegalArgumentException {
    if(userFollowing == null || userFollowed == null) throw new IllegalArgumentException();

    try (Session session = Neo4jDriver.getInstance().getDriver().session()) {
        session.writeTransaction((TransactionWork<Void>) tx -> {
            tx.run( s: "MATCH (:User { username: $username1 })-[f:FOLLOWS_USER]->(:User { username: $username2 }) "
                        + "DELETE f",
                    parameters( ...keysAndValues: "username1", userFollowing.getUsername(), "username2", userFollowed.getUsername())
            );
            return null;
        });
        logger.info("Deleted user <" +userFollowing.getUsername()+ "> follows user <" +userFollowed.getUsername()+ ">");
    } catch (Neo4jException n4jEx) {
        logger.error(n4jEx.getMessage());
        throw new ActionNotCompletedException(n4jEx);
    }
}
```

Figure 49: unfollow user

Unfollow Squad

```
public void unfollowSquad(User user, Squad squad) throws ActionNotCompletedException, IllegalArgumentException {
    if(user == null || squad == null) throw new IllegalArgumentException();

    try (Session session = Neo4jDriver.getInstance().getDriver().session()) {
        session.writeTransaction((TransactionWork<Void>) tx -> {
            tx.run( s: "MATCH (:User { username: $username })-[f:FOLLOWS_squad]->(:squad { squadId: $squadId }) "
                        + "DELETE f",
                    parameters( ...keysAndValues: "username", user.getUsername(), "squadId", squad.getID())
            );
            return null;
        });
        logger.info("Deleted user <" +user.getUsername()+ "> follows squad <" + squad.getID()+ ">");
    } catch (Neo4jException n4jEx) {
        logger.error(n4jEx.getMessage());
        throw new ActionNotCompletedException(n4jEx);
    }
}
```

Figure 50: unfollow squad

12.6 On-Graph Queries

Suggested User - First Layer Who are the users that are Followed by user that "User A" follows?

```java
public List<User> getSuggestedUsers(User user, int limit) throws ActionNotCompletedException, IllegalArgumentException {
    if(limit <= 0 || user == null) throw new IllegalArgumentException();

    List<User> list;
    try( Session session = Neo4jDriver.getInstance().getDriver().session()) {
        list = session.readTransaction((TransactionWork<List<User>>) tx -> {
            Result result = tx.run(
                    s: "MATCH (me:User {username: $me})-[:FOLLOWS_USER]->(followed:User)"
                            + "-[:FOLLOWS_USER]->(suggested:User) WHERE NOT (me)-[:FOLLOWS_USER]->(suggested) "
                            + "AND me <> suggested RETURN suggested, count(*) AS Strength "
                            + "ORDER BY Strength DESC LIMIT $limit",
                    parameters( ...keysAndValues: "me", user.getUsername(), "limit", limit)
            );
            ArrayList<User> firstLayerUsers = new ArrayList<>();
            while ((result.hasNext())){
                Record r = result.next();
                firstLayerUsers.add(new User(r.get("suggested").get("username").asString()));
            }
            return firstLayerUsers;
        });
```

Figure 51: first layer

Suggested User - Second Layer Who are the users that have the highest number of likes in common with "User A"?

```
    final int firstSuggestionSize = list.size();
    if(firstSuggestionSize < limit) {
        List<User> secondLayerSuggestion = session.readTransaction((TransactionWork<List<User>>) tx -> {
            Result result = tx.run(
                    s: "MATCH (me:User {username: $username})-[:LIKES]->()<-[:LIKES]-(suggested:User) "
                            + "WHERE NOT (me)-[:FOLLOWS_USER]->(suggested) "
                            + "AND NOT (me)-[:FOLLOWS_USER]->()-[:FOLLOWS_USER]->(suggested) AND me <> suggested "
                            + "RETURN suggested, count(*) AS Strength ORDER BY Strength DESC LIMIT $limit",
                    parameters( ...keysAndValues: "username", user.getUsername(), "limit", limit - firstSuggestionSize)
            );
            ArrayList<User> secondLayerUsers = new ArrayList<>();
            while ((result.hasNext())){
                Record r = result.next();
                secondLayerUsers.add(new User(r.get("suggested").get("username").asString()));
            }
            return secondLayerUsers;
        });
        list.addAll(secondLayerSuggestion);
    }
} catch (Neo4jException n4jEx) {
    logger.error(n4jEx.getMessage());
    throw new ActionNotCompletedException(n4jEx);
}
return list;
```

Figure 52: second layer

Suggested Squad - First Layer Which are the Squads that are Followed by user that "User A" follows?

```
public List<Squad> getSuggestedSquads(User user, int limit) throws ActionNotCompletedException{
    if (user == null || limit <= 0)
        return new ArrayList<>();
    List<Squad> firstList;
    List<Squad> secondList = new ArrayList<>();
    try( Session session = Neo4jDriver.getInstance().getDriver().session()) {
        firstList = session.readTransaction((TransactionWork<List<Squad>>) tx -> {
            Result result = tx.run(
                    s: "MATCH (me:User {username: $me})-[:FOLLOWS_USER]->(followed:User)"
                            + "-[:FOLLOWS_squad]->(suggested:Squad) WHERE NOT (me)-[:FOLLOWS_squad]->(suggested) "
                            + "RETURN suggested, count(*) AS Strength "
                            + "ORDER BY Strength DESC LIMIT $limit",
                    parameters( ...keysAndValues: "me", user.getUsername(), "limit", limit)
            );
            ArrayList<Squad> squads = new ArrayList<>();
            while ((result.hasNext())){
                Record r = result.next();
                squads.add(new Squad(r.get("suggested")));
            }
            return squads;
        });
```

Figure 53: first layer

Suggested Squad - Second Layer Which are the Squads that are Followed by User that are Followed by User that "User A" follows?

```
        if (firstList.size() < limit) {
            secondList = session.readTransaction((TransactionWork<List<Squad>>) tx2 -> {
                Result result = tx2.run(
                        s: "MATCH (me:User {username: $me})-[:FOLLOWS_USER]->(followed:User)\n" +
                                "-[:FOLLOWS_USER]->(suggestedUser:User)-[:FOLLOWS_squad]->(suggestedsquad) \n" +
                                "WHERE NOT (me)-[:FOLLOWS_USER]->(suggestedUser)  \n" +
                                "AND me <> suggestedUser \n" +
                                "AND NOT (me)-[:FOLLOWS_squad]->(suggestedsquad)\n" +
                                "AND NOT (followed)-[:FOLLOWS_squad]->(suggestedsquad)\n" +
                                "RETURN suggestedsquad, count(*) AS Strength \n" +
                                "ORDER BY Strength DESC LIMIT $limit",
                        parameters( ...keysAndValues: "me", user.getUsername(), "limit", limit - firstSuggestionsSize)
                );
                ArrayList<Squad> squads = new ArrayList<>();
                while ((result.hasNext())) {
                    Record r = result.next();
                    squads.add(new Squad(r.get("suggestedSquad")));
                }
                return squads;
            });
        }
    } catch (Neo4jException n4jEx) {
        logger.error(n4jEx.getMessage());
        throw new ActionNotCompletedException(n4jEx);
    }
    firstList.addAll(secondList);
    return firstList;
}
```

Figure 54: second layer

Best Players Which are the players that received the highest number of likes?

```java
public List<Player> getBestPlayers(int limit) throws ActionNotCompletedException {

    List<Player> bestPlayers = new ArrayList<>();

    try (Session session = Neo4jDriver.getInstance().getDriver().session()) {
        session.writeTransaction(tx -> {

            String query = "MATCH (s:Player)<-[l:LIKES]-(u:User) WITH s, COUNT(*) as num ORDER BY num DESC RETURN s.playerId as playerId," +
                    " s.name as name, s.position as position,s.team as team, s.imageUrl as imageUrl LIMIT $limit";

            Result result = tx.run(query, parameters( ...keysAndValues: "limit", limit));
            while (result.hasNext()) {
                bestPlayers.add(new Player(result.next()));
            }
            return bestPlayers;
        });
    } catch (Neo4jException neoEx) {
        logger.error(neoEx.getMessage());
        throw new ActionNotCompletedException(neoEx);
    }
    return bestPlayers;
}
```

Figure 55: get best players

13 CAP Triangle and Non-Functional Requirements

The system must have high availability, fast reaction times, and be tolerant of data loss and single point of failure, according to the Non-Functional Requirements.
To achieve these results, I focus the application on the CAP triangle's A (Availability) P (Partition Tolerance) edge, necessitating the use of the Eventual Consistency paradigm on my dataset. This decision results in the application's content being highly available, even if a physical network mistake occurs, at the cost of returning to the user data that is not necessarily accurate.

14 Cross-Database Consistency Management

Add/Remove a Player/User/Squad and Add/Remove a Like to a Player are two procedures that involve cross-database consistency management:
If an error happens during the first write operation, i just log the error in the errors.txt file; but, if an error occurs during the second write operation, i also try to remove the previous write operation in addition to logging the error.
When using delete entity, i remove the document from MongoDb first, but if an error happens in Neo4j, i do not re-add the document to Mongo, instead logging the error.
In the case of a player, i first delete the edge in Neo4j, but if an error happens during the second write operation in MongoDb, i do not add the edge in Neo4j again, instead logging the error.

Administrators are in charge of manually checking and enforcing consistency when issues have arisen in order to restore consistency.

15 GUI

Welcome Page

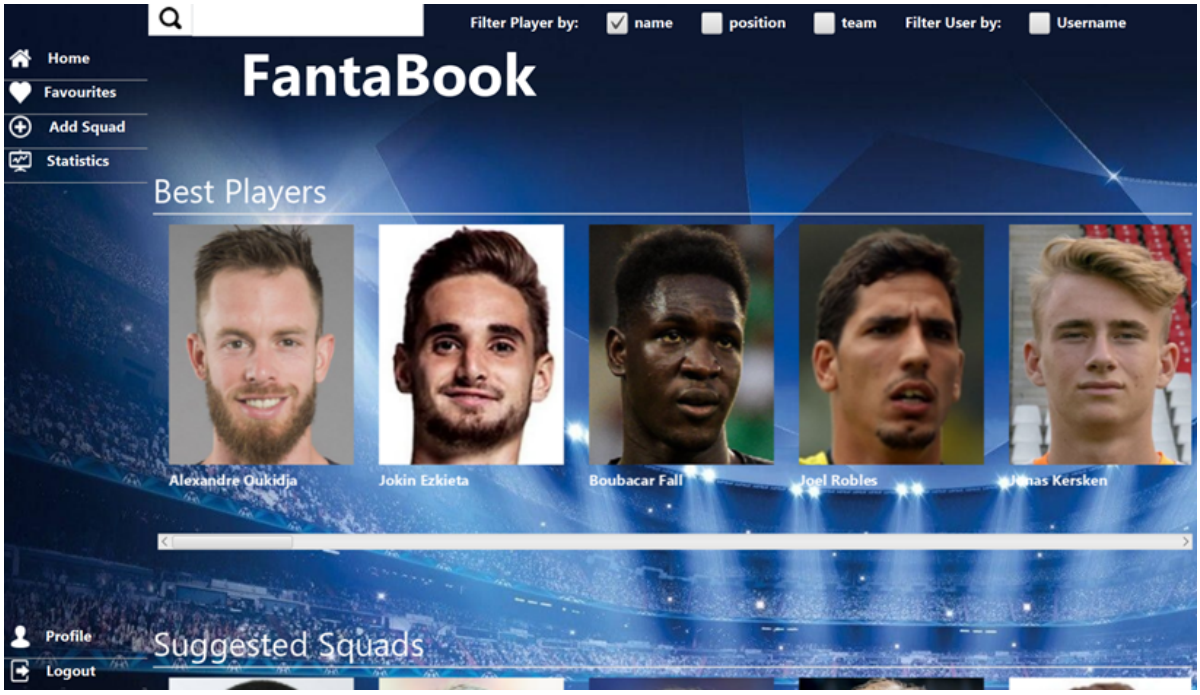


Figure 56: welcome page

Home Page

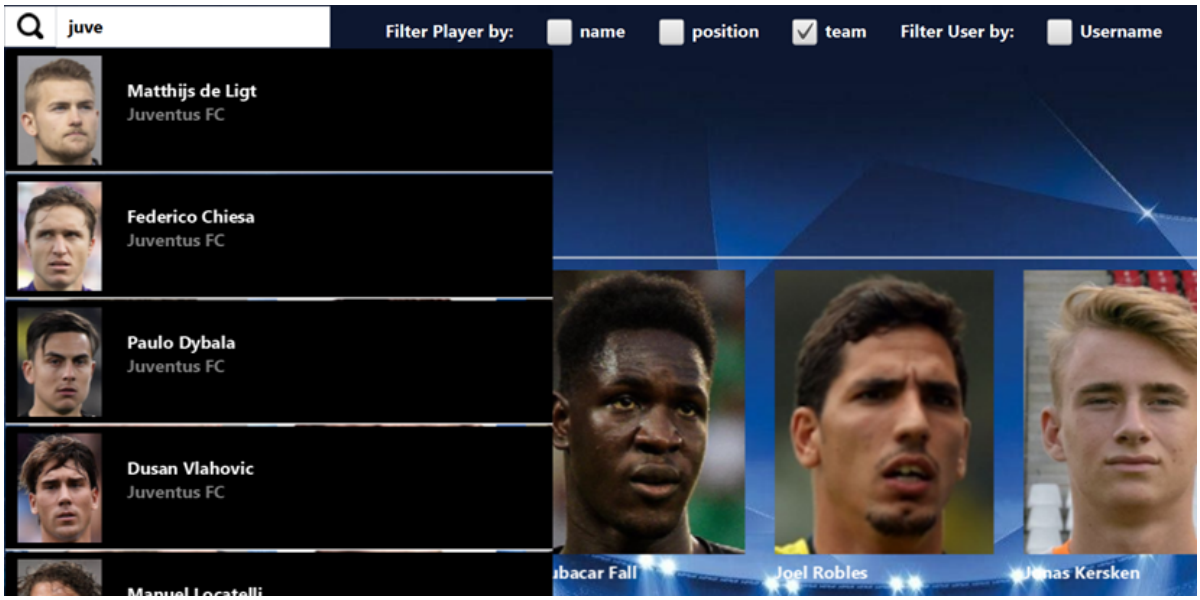


Figure 57: home page

Search Bar

Figure 58: search bar

Player Page

Figure 59: player page

User Page

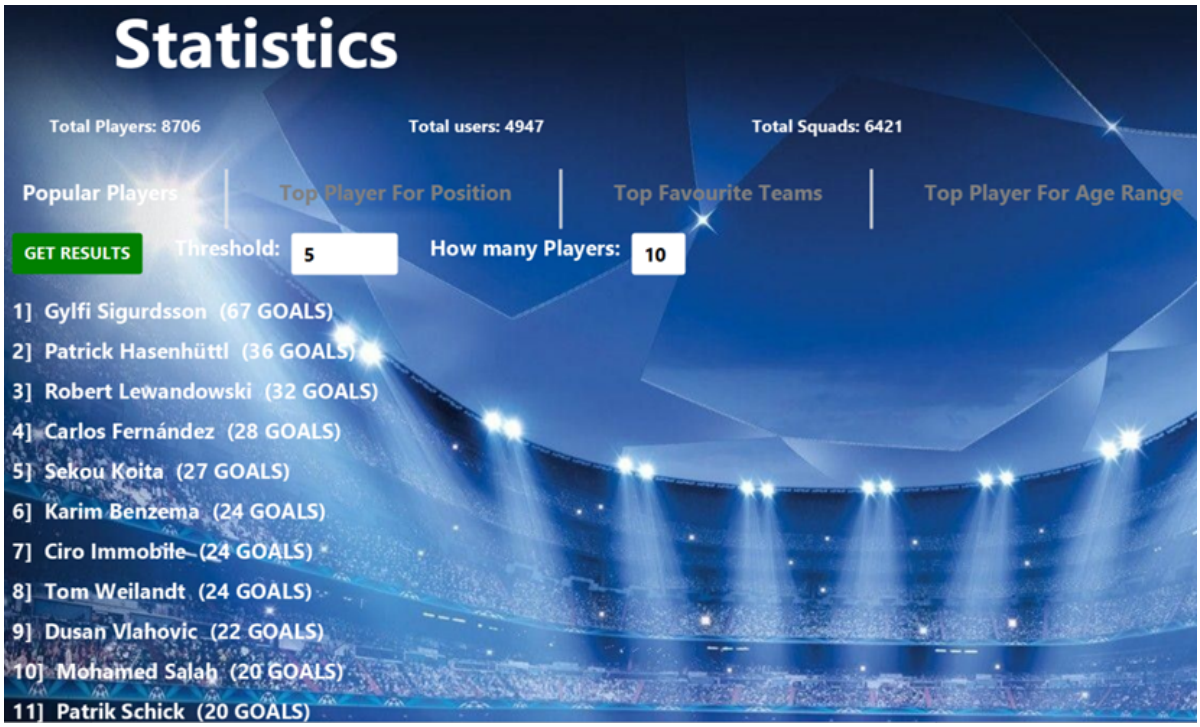


Figure 60: user page