SRH HOCHSCHULE HEIDELBERG

ADVANCE INFORMATION SYSTEM

Football Information System

Authors
Akshay Tadapathri
Binita Nayak
Gaston Vejar
Gautam Rattihalli
Nithin Bhardwaj

Supervisors
Prof.Dr Ajinkya Prabhune
Prof.Dr Barbara Sprick

May 9, 2019

Contents

List of Figures

1 Introduction

1.1 Introduction to the project

Football is played in hundreds of countries in all continents .It has a wide fan-base and considerable amount of revenue is generated worldwide and there is a constant demand from the fans of this sport for information pertaining to football. This market for football information demands a system that can consolidate and analyse a vast amount of data and provide the users with what is relevant for them individually, allowing them to focus on their teams or players of choice.
The purpose of this project is to design and develop data models across different types of databases in order to support a football information system; allowing multiple levels of data granularity, from match events happening live to consolidated data for clubs, matches, players and managers. However, just gathering data is not the sole purpose of this system, it will also allow users who are following the same clubs or players to be able to connect with each other via user created meetings providing a whole new social aspect, which the data model should also support. This data model also supports another social aspect that allows the users to comment on the ongoing matches.
Gathering data throughout different seasons will also provide the opportunity to act on historical data and identify trends in the performance of clubs and players alike; however this also presents the biggest challenge since all data across databases should be carefully curated so it can be successfully integrated while maintaining consistency. Data should flow flawlessly from the database that is gathering live match events such as *(kick off, half time, full time, passes, shots, shots on target, goals, fouls, yellow cards, red cards, and substitutions)* to the database consolidating these events and gathering all historical information for clubs, players and matches, and finally to the database that has all the relationships between these entities.
In this document we will present the databases selected to achieve the goal of the project, and the data models that support it, along with the reasoning behind each choice and its implementation.

1.2 Organisation

In order to attain the end result of the project successfully, we followed a specific organisational structure and the agile methodology of project management. Being a team of five members, we divided our tasks evenly and

coordinated accordingly.
We aimed at achieving the results in sprints and divided the project into five sprints (details in **appendix A**).Daily stand-up meetings, sprint review and sprint retrospective were held to ensure seamless progress in the project. The team members came up with new ideas, discussed the tasks done and put forth the difficulties they faced while achieving each task. At the beginning of each sprint, the planning meeting was organised and various project management tools were used to track our progress and to provide a common platform for all the members to interact and present their ideas.
The tools and techniques used for team collaboration are as follows -

- Brain Storming session - This session was conducted initially to garner the ideas for formulating the user-stories.
- Trello Board - It was used to assemble user-stories and track the sprints.
- Slack - This platform was used for convenient communication among the team members.

The tasks were divided among the team members and carried out respectively. An overview of tasks performed by the members is given through the **RCI matrix** below.

	Akshay Kumar Tadapathri	Binita Nayak	Gaston Vejar	Gautam Rattihalli	Nithin Bhardwaj
User Stories and Sub user stories formulation	R	R	R	R	R
Selection of Databases	R	R	R	R	R
Data Modelling	R	R	R	R	R
Mongo DB queries for CRUD operations/ structure of collections	I	R	I	I	C
Redis queries for CRUD operations	I	I	R	I	I
Neo4j queries for CRUD operation	I	C	I	I	R
Scripts	C	I	R	C	R
Weekly Status Report maintenance	C	R	C	C	C
Final Report Writing	R	R	R	R	R

R – Responsible, C – Consulted, I – Informed

Figure 1: RCI Matrix

2 Project Use Case

Amongst many sports across the globe, football is considered to have a wide fan-base. After many a discussions on the use case, our team gauged the dominant details that a fan or follower of the sport might expect from an efficient football application. In order to have a better realisation of the use case in general, we first identified the various components **(figure 2)** of the sport to be considered and then came up with five user stories from the end-user's perspective. Each user-story has sub user-stories which specify the scope of the user-story in detail.The user-stories helped us understand the requirements of the information system better.
The figure below provides a general idea of the use-case.

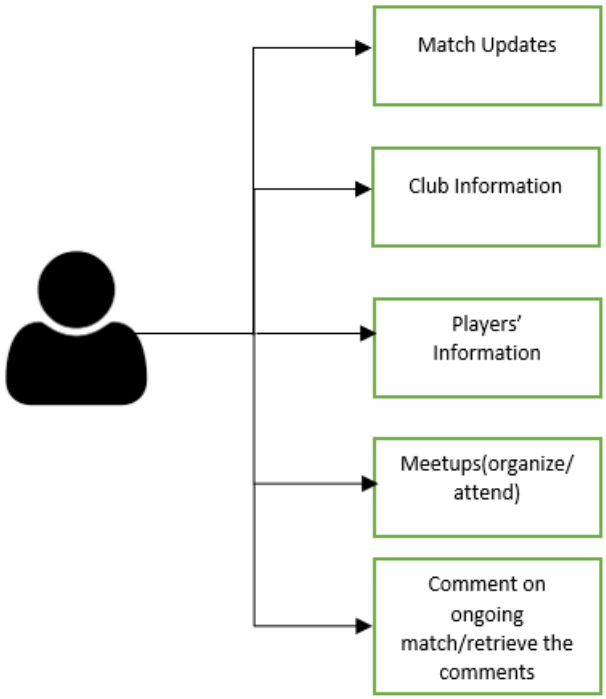

Figure 2: Visual Realisation of the Use Case

The user-stories and the sub user-stories that define the scope of the project are explained further.

2.1 User Story - 1

The first user-story is related to the expectations, of an end user of the football application, with regard to the match specific information. In this user-story we tried to think from both "live match" and "a match in past" view and tried to break down into sub user-stories.
"As an end user, I would like to stay updated with the match details so that I can schedule my timetable for a match of my interest and keep a track of it."

The following sub-user stories were considered for this user-story -

- As an end user, I want to know the event details of a live match .The details that I want to know are as follows- **kick-off, pass, throw in,**

shot, goal kick, foul, goal, w/assist, off side, yellow card, red card, half time, second half, substitution, corner, game over

- As an end user, I want to know the general information that include the date,time and venue of the match of my interest.
- As an end user, I want to know the starting line-up of both home and away teams for the match of my interest.
- As an end user, I would like to know about the event details of both home and away teams of the completed matches such as - **goals, assists, fouls, yellow cards, red cards, corner, total shots, shots on target, total passes, passes completed**.
- As an end user, I would like to know the matches scheduled between two particular dates, so that I can schedule my timetable accordingly.

2.2 User Story - 2

The second user-story covers the component of use-case that deals with the football clubs' information. We defined the scope of this user-story by laying six sub user-stories.
"As an end user I want to see the details of a club of my choice in order to stay updated about the events and general information."

The following sub-user stories are to be considered for this user story -

- As an end user, I would want to know how the league table **(played,lost,won,draw,points)** is going on for club of my choice for a specific league.
- As an end user, I would like to know the standings of clubs in a specific league on basis of points for matches played.
- As an end user, I want to acquire general details of the club of my choice such as - **founded , city , country , stadium , telephone , email , address , store, manager**.
- As an end user, I would like to know the names of players playing for my club of my choice.
- As an end user, I want to know the number of matches the team of my choice is playing as home or away team in a league of my choice.
- As an end user, I want to know the count of players in different playing positions for a specific club.

2.3 User Story - 3

The third user-story is about the players' information that a follower of the sport would want to retrieve from the information system.
"As an end user, I would like to know the details of the player of my choice so that I can have an idea about his performance in the current season."

The following sub-user stories are to be considered for this user story -

- As an end user, I want to know the event details such as- **minutes played, goals, assists, fouls, penalties conceded, penalties taken,yellow cards, red cards, total shots, shots on target,total passes,passes completed** for a player of my choice on -
  - match level - details of the events, mentioned above, for a particular match
  - league level - details of the event, mentioned above, for a particular league
  - career event details - details of events, mentioned above, in all the matches played in his career.
- As an end user, I would like to know the standings/rankings of the players on basis of total goals scored on-
  - club level
  - league level
- As an end user, I would like to know the general information such as - **name,club,age,national team,playing position** of the player of my choice.
- As an end user, I would like to know the transfer details such as - **from, to, fee, date** of the player of my choice.

2.4 User Story - 4

The fourth user-story considers the social aspect of the information system that allows the fans to meet and greet each other or attend player training camps being organised by the football clubs.
"As an end user, I want to organize or attend meet-ups scheduled in area of my choice so that I can watch a match with other fans or meet my favourite team or players."

The following sub-user stories are to be considered for this user-story -

- As an end user, I want to know the **name,venue,date,description** of the meet-ups associated with club of my choice .
- As an end user, I want to know the total number of meet-ups for club of my choice.
- As an end user, I want to know the count of users attending a particular meet-up that I am interested in.
- As an end user, I want to know the contact information of the organizer of the meet-up that I am attending.
- As an end user, I want to know the type of meet-ups available

2.5 User Story - 5

The last user-story is also a part of the social aspect of the application that allows the users to comment on a live match.
"As an end user, I would like to comment/cheer for my favourite league team in match of my choice in order to have an interactive experience."

The following sub-user story is to be considered for this user-story -

- As an end user, I want to go through the comments given for a particular match of my choice.

3 Fundamentals

3.1 Technology Used

For this project we are using three different NoSQL databases with very specific purposes:

- MongoDB: for storing all detailed and historical data
- Neo4j: for storing all relations between entities
- Redis: for all data concerning live matches

3.1.1 MongoDB

MongoDB is one of the most used NoSQL databases nowadays. Unlike relational databases where the data is stored in tables, MongoDB stores the information in JavaScript Object Notation (JSON) documents, which is a standard for many applications. This allows MongoDB and these applications to be integrated much more easily.
Using JSON documents is also very useful when the data structure of the information we want to store is not always the same. For example, if I'm storing user comments in a collection, some comments will differ in their data structure; some will be replies to other comments so, they will have a field to store the data that references the main comment they are replying to, but the main comment won't have this field at all. This allows your database to grow without executing scripts that would create default field values every time you need to add a new field to your documents. As mentioned before, is completely normal that all documents in the same collection don't have the same fields. Therefore, MongoDB is referred as a Schema-Less database.
Another important point to consider when using a database for an application is its scalability. Multiple requests from users will require that your database solution scales with the demand, and MongoDB is a distributed database that can achieve this not only in the vertical meaning (incrementing CPU and RAM), but also in a horizontal way creating more nodes. A common scenario for applications today is having users from all around the globe, so not having your application regionally distributed can be a major pain point in performance. MongoDB solves this allowing you to have distributed clusters, reducing the latency that exist between the database cluster and the service that executes the query, improving the querying speed considerably and providing high availability.
From version 4.0 MongoDB also allows ACID (Atomicity, Consistency, Isolation, Durability) transactions between multiple documents, providing validity to transactions even if power failures or errors occur.
In MongoDB we can find multiple operators that allow us to create powerful queries (field, range of fields, and regular expressions), which can return a specific field in a document, or a JavaScript function defined by the user. We also can use aggregators, which provide a mechanism that allows us to perform operations between multiple collections.

3.1.2 Neo4J

Neo4j is a graph database management system, which is another type of NoSQL database. A graph is composed by 2 elements: nodes (or vertices) and edges. A node represents an entity (like a person, institution, thing, or category) and the edges represent how the nodes are related to each other. This structure allows us to model every scenario defined by entities and relations between them, providing flexibility, fast searches and indexing.
Flexibility is provided since the data doesn't have to maintain a rigid structure; every node can be of a different type, and additional attributes can be added easily. Searches are based on the edges of the nodes, and since graph databases are indexed naturally by their edges, querying many degrees of relations between nodes is exponentially faster than doing multiple joins in traditional RDSMS databases.
Neo4j uses labeled graphs, meaning that each node can have one or more strings assigned that define the node. Edges have relationship types, that are tokens used to defined how the two nodes are related. Both, nodes and edges, can have properties that are used to store information about the entity or relation.

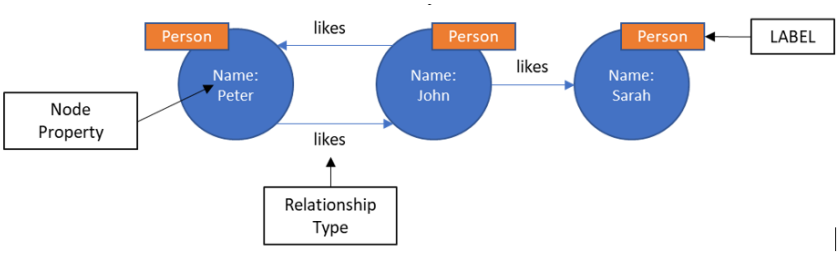

Figure 3: Neo4J Basics

Neo4j is also an ACID database, meaning that a transaction is done completely or not at all, there is data and referential integrity and each transaction exist independently of each other. It also provides high availability, balancing the querying load.
The query language for Neo4j is called Cypher, which was created inspired by SQL, but based on patterns. It is both a query language and a data manipulation language

3.1.3 Redis

Redis is an acronym of *REmote DIctionary Server* and is a key-value structure in-memory data store. All data in Redis is in the core memory of the server, this differs from most databases that store their data on hard drives or SSD. By removing the necessity to access hard drives, Redis allows for faster and simpler access to the data with algorithms that use fewer instructions of the CPU. Typical operations are executed in less than a millisecond. Redis allows users to store keys that correspond with diverse data types. The fundamental data type is the String, that can be composed of text or binary data. Redis also allows for Lists of strings in the order they were added; Sets that are unordered collection of strings with the ability to intersect, union, and diff other set types; Sorted Sets ordered by a value; Hashes that store a list of fields and values; Bitmaps that provide operations at bit level; and HyperLogLogs that count unique elements of a data set.
Redis provides a number of tools that facilitate development and operations, like Pub/Sub to publish messages in channels that are delivered to subscribers, useful particularly for chat and messaging systems; TTL keys that are automatically deleted which prevents the database to be filled with unnecessary data; atomic counters to guarantee that running conditions don't create incoherent results; and Lua, a lightweight but powerful scripting language.
Redis provides high availability with replication of data to multiple servers by employing a primary-replica architecture that allows for asynchronous replication. This means that requests can be split among the servers, providing improved performance and faster recovery when the primary server is down. Redis also allows for persistence copying the data to disk. Applying the primary-replica architecture in a clustered topology allows Redis to be scalable on demand.

3.2 Data Model

Because of the reasons mentioned previously in **3.1.1**, MongoDB is particularly good at handling high amounts of unstructured data at high speeds, scalable and reliable; therefore, using it as the main database to store all the historic detailed information for our project was an easy choice.
One aspect in which MongoDB is not very good though is providing explicit relations between entities, therefore we decided to store all relations between entities in Neo4j. Redis is built mainly for speed, since it is an in-memory database, and is highly scalable, so we decided that all infor-

mation that needs to be shown fast and will have a large number of users accessing the same data concurrently, as is the case with live matches with their large number of events and comments being inserted constantly during the event, should be in Redis. Also, since it is single-threaded and transaction atomic, it is the best choice to propagate the information between databases, since it will ensure that all tasks are done only once and there is no 'locked' data. This can be done at an application level using Redis as a queue where the transactions would be lpush (pushed from the left) and rpull (pulled from the right) to be executed as FIFO.

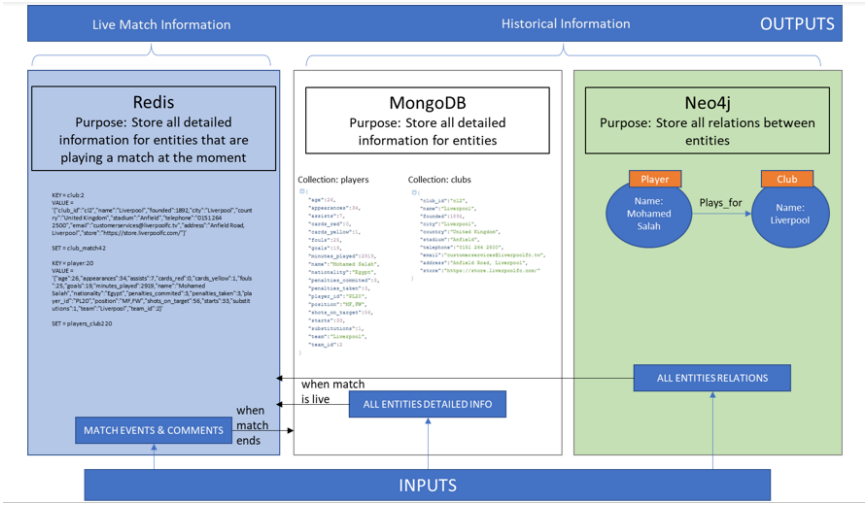

Figure 4: Data Model Overview

3.3 CRUD Operations

In this section we will take a look at the basic CRUD **(Create, Read, Update, Delete)** operations in the three databases used in our project. This section gives just the overview of the operations, the details can be understood in section **[4.4]**.

3.3.1 MongoDB - CRUD

- *use <database_name>*
  This command creates a new database or switches to an already existing database with the same name.

- *show collections*
  This command displays the names of the collections in the database.
- *db.createCollection("<collection_name>")*
  This command creates a collection by the given name in the selected database.
- *db.<collection_name>.insert()*
  This command inserts one document to the collection.If the collection does not currently exist then insert operations will create the collection.
- *db.<collection_name>.insertMany()*
  This command inserts multiple documents at a time to the collection. It returns -
  - A boolean acknowledged as valid if the operation ran with write concern or false if write concern was disabled.
  - An array of _id for each successfully inserted documents.This _id is generated by MongoDB automatically.

  For example, after succesfully inserting two documents, the acknowledgement is as follows -

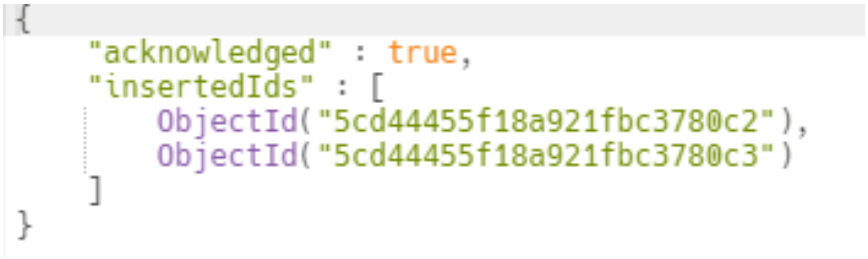

Figure 5: MongoDB insertMany result

- *db.<collection_name>.find().pretty()*
  This command returns all the documents in the collection in an ordered manner.
- *db.<collection_name>.find().project(_id:0)*
  This command returns the documents without their generated _id . We can similarly project other fields in the documents too. Appending 1 will ensure that the field is displayed.

- *db.<collection_name>.updateOne()*
  This command updates a single document and *db.<collection_name>.updateMany()* updates many documents.

- *db.<collection_name>.deleteOne()*
  This command deletes a single document and *db.<collection_name>.deleteMany()* deletes many documents.

3.3.2 Neo4J - CRUD

There are a few words in Cypher reserved for specific actions in parts of a query. We need to be able to create, read, update, or delete data in Neo4j, and some keywords help us accomplish that functionality.Below are the keywords used to accomplish the functionalities in Neo4J in our project-

- CREATE - Adding data in Cypher works very similarly to any other data access languages insert statement.CREATE in Cypher is much similar to INSERT keyword in SQL. We have used CREATE to insert nodes, relationships, and patterns into Neo4j.

- MATCH - The MATCH keyword in Cypher is what searches for an existing node, relationship, label, property, or pattern in the database. MATCH works pretty much like SELECT in SQL.

- RETURN - The RETURN keyword in Cypher specifies what values or results you might want to return from a Cypher query. One can tell Cypher to return nodes, relationships, node and relationship properties, or patterns in their query results. RETURN is not required when doing write procedures, but is needed for reads.

- UPDATE - Modifying node or relationship and its properties, can be done by matching the pattern and using the SET keyword to add, remove, or update properties.

- DELETE - Cypher uses the DELETE keyword for deleting nodes and relationships. It is very similar to deleting data in other languages like SQL, with one exception.

3.3.3 Redis - CRUD

- SET
  - Usage: SET *key "value"*.

  - Description: Sets *key* to hold the string *value*.

- SADD
  - Usage: SADD set *"value"*.
  - Description: Add the string *value* to the specified set, if the set doesn't exist it's created.

- GET
  - Usage: GET *key*.
  - Description: Retrieves the value of the specified *key*.

- MGET
  - Usage: MGET *key1 key2 key-n*.
  - Description: Retrieves the values of all specified *keys*.

- SMEMBERS
  - Usage: SMEMBERS *set*.
  - Description: Returns all members of the set value stored

4 Project Data Model

For building an information system, appropriate data is a prerequisite and modelling the data i.e defining the type and format of the data is crucial. As our information system uses data from the three databases - MongoDB, Redis and Neo4J, data modelling in all the three databases and linking on some common parameter was important.Attaining *scalability* and *removing redundancy* were the major factors that shaped our data modelling. In this section we will take a look at the data models adopted by us across the three databases, the entities and relationships between each of them.

4.1 MongoDB - Data Model

In our project, MongoDB holds the general information of all entities and the aggregate of some data from Redis **[4.4]**. The details of the data modelling i.e the collections, the parameters of the collections, data types are mentioned in the following section.
MongoDB holds the information of each entity in a more detailed manner

in several collections. The collections are created in the database named FIS. The components that are to majorly provide information to the end users, are created as collections and the entities are stored as documents. Further, the documents have various parameters. The brief description of each collection can be found below and the examples *(with details of the parameters in each document within the collection)* from our data model are mentioned in **[appendix B]**.

- clubs -
  The *clubs* collection stores the general information of the club (club_id, name, founded, city, country, stadium, telephone, email, address, store)and an array of documents named 'league_table' which contains statistics of the club in a particular league. The league_table contains fields such as - *league_id, played, won, draw, lost, points*,using which the points table is obtained for all the teams of a particular league.

- managers -
  The *managers* collection contains the general information of the managers() and there personal achievements. No relationship is established between any collection/ documents in MongoDB as all relationship will be obtained in Neo4J.

- matches -
  The *matches* collection contains various general information about the team such as Date, Time, Venue, etc., of the match. The collection also includes a comment array that holds all the comments of users with respect to the match.
  The result of the match is also stored in the collection, the result is dynamically found out using the query rather than updating it manually. Once the aggregated event data and comments are obtained from Redis post match, the details of the home and away teams are updated, a query to find out the result of the match and also to update points, games played, won and lost of the teams are executed.

- meetups -
  The *meetups* collection has all the details of the meetup such as Venue, Type, date and time along with a breif description of the same.

- players -
  The *players* document contains all the general information of the player. The key field to be considered here is the "event-details", it contains

complete details of all the events with respect to a match along with event details transfers array has all the details of transfers of a player.

- users -
  The *users* collection is a simple document with all the general information of the users.

4.2 Neo4J - Data Model

In Neo4J, all the possible relationships between the entities are stored. There are several nodes that represent the entities involved in the graph such as -

- Users
- Match
- League
- Manager
- Player
- Country
- Meetup
- Club

The edges define the relationships such as *Manages, Home, Belongs, Organize, Attends, Associated, Follows, Plays_for*. The relationships bind two entities with each other. In our model, each node has 2 properties, namely *"name"* and *"id"*. The detailed data model from Neo4J with each entity and relationships is mentioned in **[appendix C]**

4.3 Redis - Data Model

Redis is an in-memory high-performance database and hence is used in our project to get the events of a match at real time i.e event details of a live match . Once the data is obtained it is further processed and aggregated before saving it in the document database i.e MongoDB . Following entities are present in our Redis data model -

- Users:

  - KEY = user:X (where X is the number of the user id). VALUE = string with the user details in JSON format.
  - SET = users_clY X (where Y is the number of the club id and X is the number of the user id).

- Match:
  - KEY = match:X (where X is the number of the match id). VALUE = string with the match details in JSON format.
  - SET = matches_Y X (where Y is the identifier for the league and X is the number of the match id).

- Teams:
  - KEY = team:X (where X is the number of the team id). VALUE = string with the team details in JSON format.
  - SET = teams_matY X (where Y is the number of the match id and X is the number of the team id).

- Managers:
  - KEY = manager:X (where X is the number of the manager id). VALUE = string with the manager details in JSON format.
  - SET = managers_clY X (where Y is the number of the club id and X is the number of the manager id).

- Players:
  - KEY = player:X (where X is the number of the player id). VALUE = string with the player details in JSON format.
  - SET = players_clY X (where Y is the number of the club id and X is the number of the player id).
  - SET = starting_players_clY X (where Y is the number of the club id and X is the number of the player id).
  - SET = bench_players_clY X (where Y is the number of the club id and X is the number of the player id).

- Events:
  - KEY = event:X (where X is the number of the event id). VALUE = string with the event details in JSON format.

  - SET = events_matY X (where Y is the number of the match id and X is the number of the event id).
  - SET = events_plZ X (where Z is the number of the player id involved in the event and X is the number of the event id).

- Comments:
  - KEY = comment:X (where X is the number of the comment id). VALUE = string with the comment details in JSON format.
  - SET = comments_matY X (where Y is the number of the match id and X is the number of the comment id).

4.4 Link between the three data models

ID here plays a major role as the same IDs are maintained across the system and the relationship is established, for example in the above graph the node Tim (user) has properties name: "Tim", id: "us9", the same id i.e. us9 is used in the document collection of MongoDB and also in Redis DB.

5 Implementation

5.1 Implementation of Data Model

5.2 Implementation of user stories and queries

5.3 Queries

6 Conclusion

As we said in the introduction, football is one of the most popular sport in the world. This means that there are more fans of football clubs than there are of any other sport, so the potential users for a football information system is incredible high if this system can integrate at least the most popular leagues around the globe.
One issue that we quickly encountered is that these leagues not always follow the same structure, some are only leagues, and some have cups with knock-off stages. Players also differ from each other, not only in position, but the number of clubs they played for, and the matches they have played in. This make us realize that we would have a great amount of unstructured data, so a JSON document database such as MongoDB would be perfect to

accomplish the task of storing that data. MongoDB provides the flexibility in data structure necessary for the system to be scalable and add different fields for each entity as its needed when adding new data. Also using JSON documents in the database facilitates the integration with one-page applications, which are perfect for these type of systems since most users will prefer to access it through their mobile devices.
The problem with using a document-based database arose when we identified that the entities involved in this information system are so closely related to each other. Players, clubs, managers, matches, even users have multiple relations between them, and MongoDB is not the ideal place to see all these interconnections. That is why we chose to store all relations in Neo4J, since being a graph database it indexes by relationship between nodes, so it is not incredible fast to retrieve those relations, but also very pragmatic in how they are shown.
Most user stories were easily solved by combining the relations stored in Neo4J and applying to that relation the detailed information stored in MongoDB. But there was the use case for live matches that we identified that could be greatly improved in performance by the addition of a third database. These live matches when seeing them from a data access perspective differ from the others in that they will have many users retrieving data constantly for the same match, as users tend to follow their teams more closely when a match is being played. For this reason, we decided that using Redis for this particular use case would be the best option. Redis, being an in-memory data store, would provide a lightning fast response to queries while being scalable for increment in users. This also means that Redis will be constantly hit by the application to insert the game event data, since the data model is prepared to receive a very granular sets of events. For these reasons we also decided to delete the data for the match from Redis once the game is over and the relevant information is sent to MongoDB to be stored and aggregated, so data for non-live games (past or future) would be retrieved from MongoDB and Neo4J.
Unfortunately, one thing that the data model cannot solve and will need to be handle at the application level is to assure that all data is consistent across all databases, and that the data transfers are executed correctly. Redis will demand the game information (including match, club, players, users, etc.) from MongoDB and Neo4J, in the same way MongoDB will require the aggregated data of the live match from Redis once it's over. It is important to notice that a great way to facilitate this is by also using Redis as a job queue for these updates, since it is single-threaded and transaction atomic, but always triggered at the application level.

A Protocols of the meetings

Along with daily stand-up meetings of 15 minutes, sprint planning of 30 minutes at the beginning of each sprint, sprint review of 1 hour at the end of every sprint, weekly report maintenance and sprint retrospective of 10 minutes everyday, following are the details of each sprint -

- Sprint 1 (2nd April 2019 - 9th April 2019)
  - Understanding the use case from perspective of all the team members and arriving at a conclusion of the statement to be realized.
  - Stakeholder analysis and market analysis were done to come up with some unique aspect of the application.
  - Brain storming session was held in which each member came up with 4 user stories each .
  - Laying out the plan for upcoming sprints and deciding meeting times.
  - Focussing on the initial twenty user stories and with continuous discussions, coming down to seven user stories.
- Sprint 2 (9th April 2019 - 16th April 2019) - Draft version of the data model was discussed
  - Discussing the initial draft version of user stories with our supervisor and modifying them.
  - Finalising the user-stories and deciding the databases to be used.
  - Discussing the data model and deciding which database is to be used for the specific requirements.
  - In this sprint we arrived at the decision that all the live matches' information will be stored in Redis, the historical data in MongoDB and all the relationships between the entities will be stored in Neo4J.
  - Then we proceeded with the detailed modelling of each database which included deciding the collection structure in MongoDB, deciding the names of the collections in MongoDB, nodes,labels and relationships in Neo4J and key,value pair in Redis.
- Sprint 3 (16th April 2019 - 23rd April 2019) -
  - Inserting dummy data into the databases.

- Creating the queries as per the user-stories and modifying the data model continuously for better performance while running the queries.
- Analysing the loopholes in the data model as we encountered some issues repeatedly.
- Planning the next sprint and the target for alpha version.

- Sprint 4 (23rd April 2019 - 30th April 2019) -
  - For the alpha version, we finally brought down the seven user stories to five.
  - Our team was ready with the queries satisfying the user-stories but the granularities were still to be taken care of.
  - We analysed the user-stories again and formed the sub user-stories again.
  - The queries were modified and finalised.

- Sprint 5 (30th April 2019 - 5th May 2019) -
  - Combining the results of the queries from all three databases were done.
  - Modification of the queries to display better results was done.

B MongoDB Data Model

■ Clubs Collection


```
{
        "_id" : ObjectId("5ccf02bb334b6d70f0e5a076"),
        "club_id" : "cl1",
        "name" : "Chelsea",
        "founded" : "1905",
        "city" : "London",
        "country" : "United Kingdom",
        "stadium" : "Staford Bridge",
        "telephone" : "0207 958 2190",
        "email" : "enquiries@chelseafc.com",
        "address" : "Fulham Road,London",
        "store" : "https://www.chelseamegastore.com/stores/chelsea/de",
```

```
        "league_table" :
        [
                {
                        "league_id" : "l1",
                        "played" : 2,
                        "won" : 1,
                        "lost" : 1,
                        "draw" : 0,
                        "points" : 10
                },
                {
                        "league_id" : "l2",
                        "played" : 4,
                        "won" : 1,
                        "lost" : 3,
                        "draw" : 0,
                        "points" : 10
                }
        ]
}
c
```

■ Managers Collection

```
{
        "_id" : ObjectId("5ccea31c334b6d70f0e5a018"),
        "name" : "Ole Gunnar Solskjaer",
        "id" : "m3",
        "status" : "active",
        "joined_club" : "19 December 2018",
        "age" : "46 years 62 days",
        "dob" : "26/02/1973",
        "premier_league_seasons" : 2,
        "premier_league_debut_match" :
        "West Ham United (h),
        11 January 2014,Lost 0 - 2",
        "awards" :
        {
                "title" : "Manager of the Month",
                "awarded_in" :
```

```
                [
                        "January 2019"
                ]
        }
}
c
```

- Matches Collection

```
{
        "_id" : ObjectId("5ccef23a334b6d70f0e5a06e"),
        "match_id" : "mat3",
        "name" : "Liverpool vs Manchester_United",
        "timestamp" : ISODate("2018-12-16T18:00:00.000+01:00"),
        "venue" : "Anfield",
        "comments" :
        [
                {
                        "commentID" : "c7",
                        "user" : "us5",
                        "team" : "cl3",
                        "commentType" : "main",
                        "content" : "Go Red Devils!",
                        "timeStamp" : "2018-12-16T18:08:29.234Z"
                }
        ],
        "home" :
        {
                "id" : "cl2",
                "goals" : 1,
                "assists" : 1,
                "fouls" : 0,
                "cards_yellow" : 2,
                "cards_red" : 0,
                "corner" : 16,
                "total_shots" : 24,
                "shots_on_target" : 15,
                "total_passes" : 24,
                "passes_completed" : 18
        },
```

```
        "away" :
        {
                "id" : "cl3",
                "goals" : 0,
                "assists" : 0,
                "fouls" : 5,
                "cards_yellow" : 6,
                "cards_red" : 0,
                "corner" : 10,
                "total_shots" : 25,
                "shots_on_target" : 21,
                "total_passes" : 26,
                "passes_completed" : 16
        },
        "result" : "cl2"
}
c
```

- Meetups Collection

```
{
        "_id" : ObjectId("5ccea92b334b6d70f0e5a01e"),
        "meetup_id" : "mt3",
        "description":
        "This is a meet up for the Manchester United club fans.
        We will be meeting at this pub,which is the most popular
        place at Heidelberg. We gonna have an exciting evening
        watching the match while drinking the beer together.
        Let's cheer for the team GGMU.",
        "venue" : "The Dubliner",
        "type" : "Match Screening",
        "date" : ISODate("2018-05-13T20:00:00.000+02:00"),
        "city" : "Heidelberg"
}
c
```

- Players Collection

```
{
        "_id" : ObjectId("5cceb427334b6d70f0e5a030"),
        "player_id" : "PL18",
```

```
        "name" : "Sadio Mané",
        "club" : "Liverpool",
        "age" : 26,
        "playing_position" :
        [
                "Mid-Fielder",
                "Attacker"
        ],
        "event_details" :
        [
                {
                        "match_id" : "mat1",
                        "minutes_played" : 90,
                        "goals" : 2,
                        "assists" : 1,
                        "fouls" : 0,
                        "penalties_conceded" : 1,
                        "penalties_taken" : 1,
                        "cards_yellow" : 1,
                        "cards_red" : 1,
                        "total_shots" : 2,
                        "shots_on_target" : 1,
                        "total_passes" : 1,
                        "passes_completed" : 1
                }
        ],
        "transfers" :
        [
            {
                        "from" : "Basel",
                        "to" : "Chelsea",
                        "fee" : "13M EUR",
                        "date" : "26 Jan 2014"
                }
        ]
}
c
```

- Users Collection

```
{
        "_id" : ObjectId("5ccea007999dcdf7a7a2fc6b"),
        "userID" : "us1",
        "name" : "binita",
        "phone" : 1765645297,
        "email" : "nayakbinita@gmail.com"
}
c
```

C Neo4J Data Model

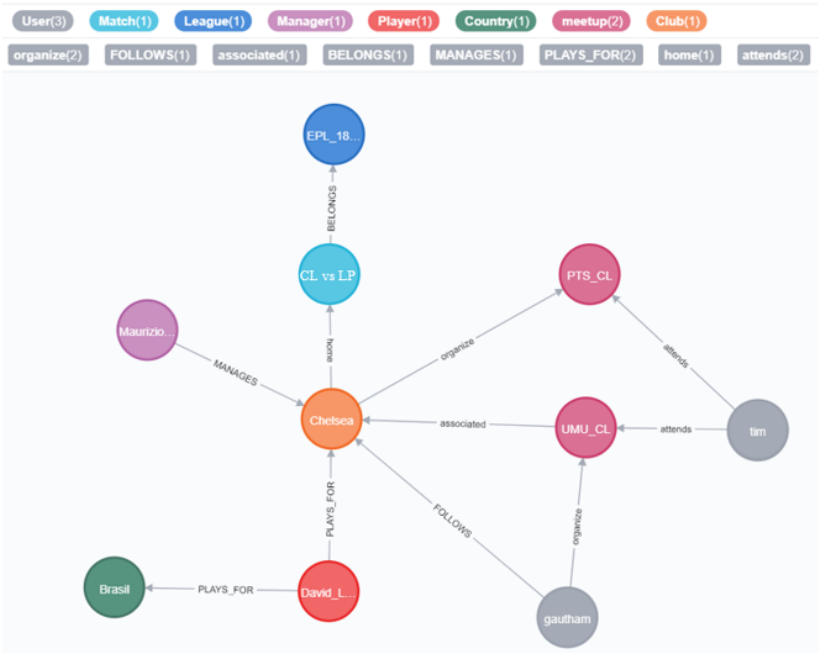

Figure 6: Neo4J Data Model

D Dummy Data Insertion- MongoDB

```
Dummy Data insert query for 'users' collection -

db.createCollection("users")
db.users.insertMany(
```

```
[
{"userID":"us1",
 "name": "binita",
 "phone":1765645297,
 "email": "nayakbinita@gmail.com"},
{"userID":"us2",
 "name": "nithin",
 "phone":1765645297,
 "email": "nithinbs18@gmail.com"}
 ])
==================================================
Dummy Data insert query for 'managers' collection -

db.createCollection("managers")
db.managers.insertMany
([{
"name":"Maurizio Sarri",
"id":"m1",
"status":"active",
"joined_club":"14 July 2018",
"age":"60 years 95 days",
"dob":"10/01/1959",
"premier_league_seasons":1,
"premier_league_debut_match":"Huddersfield Town,11 August 2018,Won 3 - 0",
"awards":
    {
    "title": "NA",
        "awarded_in": ["NA"]
    }
},
{
"name":"Jurgen Klopp",
"id":"m2",
"status":"active",
"joined_club":"8 October 2015",
"age":"51 years 303 days",
"dob":"16/06/1967",
"premier_league_seasons":4,
"premier_league_debut_match":"Tottenham Hotspur (a),17 October 2015,Drawn 0 - 0",
"awards":
```

```
    {
    "title": "Manager of the Month",
        "awarded_in": ["January 2012","August 2013","March 2014"]
    }
}])
==================================================
Dummy Data insert query for 'matches' collection -

db.createCollection("matches")
db.matches.insert
(
{
        "match_id" : "mat6",
        "name" : "Chelsea vs Manchester_United",
        "timestamp" : ISODate("2018-10-20T18:00:00.000Z"),
        "venue" : "Stamford Bridge",
        "result":"",
        home:
   { id: 'cl1',
    goals: 1,
    assists: 0,
    fouls: 0,
    cards_yellow: 2,
    cards_red: 1,
    corner: 0,
    total_shots: 0,
    shots_on_target: 0,
    total_passes: 2,
    passes_completed: 1 },
        away:
   { id: 'cl3',
    goals: 1,
    assists: 0,
    fouls: 0,
    cards_yellow: 2,
    cards_red: 1,
    corner: 0,
    total_shots: 0,
    shots_on_target: 0,
    total_passes: 2,
```

```
    passes_completed: 1 }
        "comments" :
        [
        {"commentID":"c1",
        "user":"us5",
        "team":"c13",
        "commentType":"main",
        "content":"Go Red Devils!",
        "timeStamp":"2019-04-30T13:08:29.234Z"},
    {"commentID":"c2",
        "user":"us4",
        "team":"c13",
        "commentType":"reply",
        "replyTo":"c001",
        "content":"Man U FTW!",
        "timeStamp":"2019-04-30T13:10:29.234Z"}
        ]
})


==================================================
Dummy Data insert query for 'clubs' collection -

db.createCollection("clubs")
db.clubs.insertMany([
{
"club_id":"cl1",
"name":"Chelsea",
"founded":"1905",
"city":"London",
"country":"United Kingdom",
"stadium":"Staford Bridge",
"telephone":"0207 958 2190",
"email":"enquiries@chelseafc.com",
"address":"Fulham Road,London",
"store":"https://www.chelseamegastore.com/stores/chelsea/de",
"league_table":
[
{league_id:"l1",played:4,won:2,lost:0,draw:2,points:32},
{league_id:"l2",played:5,won:6,lost:1,draw:1,points:39}
```

```
]
},

{"club_id":"cl2",
"name":"Liverpool",
"founded":"1892",
"city":"Liverpool",
"country":"United Kingdom",
"stadium":"Anfield",
"telephone":"0151 264 2500",
"email":"customerservices@liverpoolfc.tv",
"address":"Anfield Road, Liverpool",
"store":"https://store.liverpoolfc.com/",
"league_table":[]}
])
==================================================
Dummy Data insert query for 'players' collection -

db.createCollection("players")
db.players.insert
(
{
"player_id" : "PL20",
"name" : "Mohamed Salah",
"club" : "Liverpool",
"age" : 26,
"playing_position" : ["Mid-Fielder","Attacker"],
"event_details":
[
{  "match_id":"mat1",
        "minutes_played" : 90,
        "goals" : 1,
        "assists" : 1,
        "fouls" : 1,
        "penalties_conceded" : 1,
        "penalties_taken" : 1,
        "cards_yellow" : 1,
        "cards_red" : 0,
        "total_shots":2,
        "shots_on_target" : 1,
```

```
        "total_passes": 1,
        "passes_completed":1},
        {  "match_id":"mat3",
        "minutes_played" : 75,
        "goals" : 2,
        "assists" : 4,
        "fouls" : 1,
        "penalties_conceded" : 3,
        "penalties_taken" : 1,
        "cards_yellow" : 0,
        "cards_red" : 0,
        "total_shots":2,
        "shots_on_target" : 1,
        "total_passes": 1,
        "passes_completed":1}
],
"transfers":
[
{from:"Basel",to:"Chelsea",fee:"13M EUR",date:"26 Jan 2014"},
{from:"Chelsea",to:"Fiorentina",fee:"Loan",date:"2nd Feb 2015"},
{from:"Chelsea",to:"Roma",fee:"Loan",date:"6th Aug 2015"},
{from:"Chelsea",to:"Roma",fee:"15M EUR",date:"1st Aug 2016"},
{from:"Roma",to:"Liverpool",fee:"42M EUR",date:"1st July 2017"}
]
}
)
==================================================
Dummy Data insert query for 'meetups' collection -

db.createCollection("meetups")
db.meetups.insert
({
"meetup_id" : "mt1",
"description" : "This is a meet up for the Liverpool club fans.Let us celebrate!!",
"type": "Fans Meet Up",
"venue" : "Murphy's Law",
"date" : ISODate("2018-10-10T18:00:00.000Z"),
"city" : "Mannheim"
})
c@
```

E Dummy Data Insertion- Neo4J

F Dummy Data Insertion- Redis

```
Dummy Data insert query for 'players'-

SET player:55 '{"age":30,"appearances":14,"assists":1,"cards_red":0,"cards_yellow":1,
"fouls":15,"goals":0,"minutes_played":746,"name":"Marouane Fellaini",
"nationality":"Belgium","penalties_commited":0,"penalties_taken":0,
"player_id":"PL55","position":"MF","shots_on_target":3,"starts":6,
"substitutions":8,"team":"Manchester United","team_id":3}'
SADD players_cl3 55
SADD starting_players_cl3 55
SET player:58 '{"age":29,"appearances":17,"assists":3,"cards_red":0,
"cards_yellow":3,"fouls":14,"goals":1,"minutes_played":791,
"name":"Alexis Sánchez","nationality":"Chile","penalties_commited":0,
"penalties_taken":0,"player_id":"PL58","position":"MF,FW",
"shots_on_target":9,"starts":8,"substitutions":9,
"team":"Manchester United","team_id":3}'
SADD players_cl3 58
SADD starting_players_cl3 58
==================================================
Dummy Data insert query for 'match' -

SET match:3 '{"id":"mat3","name":"Liverpool_vs_Manchester_United",
"home_team":"cl2","away_team":"cl3"}'
SADD matches_epl 3
SET match:4 '{"id":"mat4","name":"Manchester_United_vs_Liverpool",
"home_team":"cl3","away_team":"cl2"}'
SADD matches_epl 4
==================================================
Dummy Data insert query for 'users' -

SET user:1 '{"userID":"us1","name": "binita","phone":1765645297,
"email": "nayakbinita@gmail.com","favourite_team": "liverpool"}'
SADD users_cl2 1
SET user:5 '{"userID":"us5","name": "gaston","phone":1765645297,
"email": "gaston@gmail.com","favourite_team": "manchesterU"}'
SADD users_cl3 5
```

```
==================================================
Dummy Data insert query for 'comments' -

SET comment:10 '{"commentID":"c10","user":"us5","team":"cl3",
"commentType":"main",
"content":"Go Red Devils!","timeStamp":"2019-04-30T13:08:29.234Z"}'
SADD comments_mat4 10
SET comment:11 '{"commentID":"c11","user":"us4","team":"cl3",
"commentType":"reply","replyTo":"c10",
"content":"Man U FTW!","timeStamp":"2019-04-30T13:10:29.234Z"}'
SADD comments_mat4 11
==================================================
Dummy Data insert query for 'events' -

SET event:1 '{"eventID":"e1","match":"mat4","eventType":"KickOff",
"gameTime":"0 FH","timeStamp":"2019-04-30T12:54:29.234Z"}'
SADD events_mat4 1
SET event:2 '{"eventID":"e2","match":"mat4","eventType":"pass",
"team":"cl3","passType":"forward",
"passResult":"successful","fromPlayer":"PL55","toPlayer":"PL58",
"gameTime":"3:15 FH","timeStamp":"2019-04-30T12:57:29.234Z"}'
SADD events_mat4 2
SADD events_pl55 2
==================================================
Dummy Data insert query for 'managers' -

SET manager:2 '{"name":"Jurgen Klopp","id":"m2","status":"active",
"joined_club":"8 October 2015","age":"51 years 303 days",
"dob":"16/06/1967","premier_league_seasons":4,
"premier_league_debut_match":"Tottenham Hotspur (a),
17 October 2015,Drawn 0 - 0"}'
SADD managers_cl2 2
==================================================
Dummy Data insert query for 'clubs' -

SET team:2 '{"club_id":"cl2","name":"Liverpool","founded":1892,
"city":"Liverpool","country":"United Kingdom",
"stadium":"Anfield","telephone":"0151 264 2500",
"email":"customerservices@liverpoolfc.tv","address":"Anfield Road,
Liverpool","store":"https://store.liverpoolfc.com/"}'
```

```
SADD teams_mat4 2
c@FancyVerbLinee2
```

G Output for queries