UNIVERSITY OF PISA

Large Scale and Multi-Structured Databases

Year 2019/20

OrgaBET

An aggregator website for sport bets, powered by MongoDB

GERARDO ALVARO

MARCO BONGIOVANNI

RICCARDO POLINI

GIULIO SILVESTRI

INDEX

1. INTRODUCTION

OrgaBet, acronym of Organize your Bets, is a web-application that acts as an aggregator for sports betting odds, suggesting to the user which bookmakers offer the best odds for the desired events.

When a visitor lands on the homepage, he is prompted to log-in (sign up first if he is not registered) in order to use the functionalities of the application.

Once logged in, the user is presented with a homepage containing a default match list of the day. The application will have a side panel in which the sport and competition may be selected by the user in order to display all the playable matches for the current date, with the respective average odds. If the user wishes to bet on a certain result, he/she will select the desired odd and the event will be added to "My Coupon".

"My Coupon" is a recap of all selected events and will be displayed in a side panel. The list will show the total odds for each available bookmaker (sorted by convenience) so that the user may choose the one to bet on. The coupon may be saved by the user: if this happens, the list is saved in an archive accessible through the personal profile.

Some sports statistics are also available in a specific panel, accessible to users. Here one can browse various analytics about Teams and Players of different sports and divisions.

2. REQUIREMENTS ANALYSIS

2.1. Application Actors

The actors of the application are the *User* and the *Admin.*

The first one is the main user of the application.

The second, has the same functionalities as the regular user but also has additional responsibilities available only to an administrator of the website.

2.2. Functional and Non-Functional Requirements

The *functional* requirements of this application, divided with respect to the two actors, are as follows:

- The application is available only to registered users.
- The application acts as an aggregator for sports betting odds.
- For each sport, the application presents all the events of the day for the available competitions in the available nations.
- For each event, the application displays the average odds of the available bookmakers.
- For each sport and division, the application presents statistics and analytics relative to the previous years.

- A User selects a result to bet on and this is added to "My Coupon".
- A User may remove a selected result from "My Coupon".
- A User may clear "My Coupon".
- For each element added to "My Coupon", the application shows the most convenient bookmakers for betting on the selected events, if any.
- A User may save the "My Coupon" in the personal profile.
- A User can browse the saved coupons in his profile
- A User may delete his/her profile.
- A User may edit his/her personal profile.
- A User may remove a previously saved coupon from the personal archive.

- An Admin can view the profiles of registered users, including their coupon archive.
- An Admin can ban users who violate the Terms & Agreements.
- An Admin can add matches to the collection.

The *non-functional* requirements of the application are:

- The application's interface must be user-friendly.
- The application must have a low response time.
- The application will store information in a non-relational Database (MongoDB).
- The application must guarantee data availability.
- The application must ensure prevention against server crashes thanks to replicas.
- The application, thanks to sharding, allows the possibility to store more data and handle more load without requiring more powerful machines.
- The application must be easily scalable.
- The application must be reliable: no system crashes, exceptions are handled etc.
- Admins of the application periodically monitor the behaviour of Users in order to guarantee that they comply to the Terms & Agreements.

3. UML DIAGRAMS

3.1. Use-Case Diagram

See the linked PDF file for a high-resolution version of the diagram.

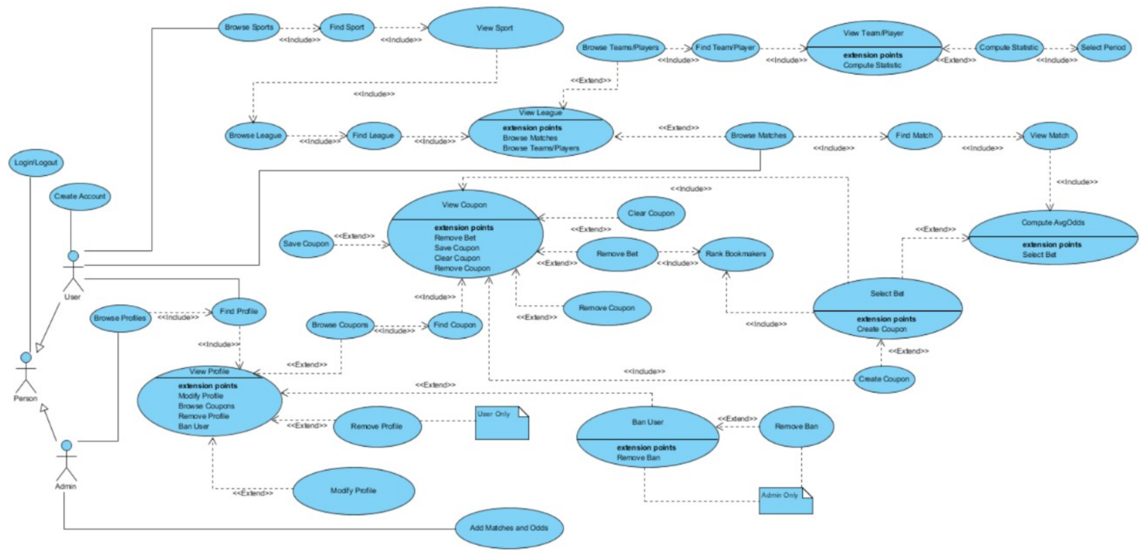


Figure 1: Use-Case Diagram

In the above figure is reported the Use-Case diagram in which we can see: the actors of the application, *Admin* and *User*; their respective action lists and some notes to specify when an action is available only to the admin or only to the user (if it is not already obvious from the diagram).

3.2. Class Analysis Diagram

In *Figure 2* are reported the main entities of the application and the relationships among them. *User* and *Admin* are generalized into *Person*.

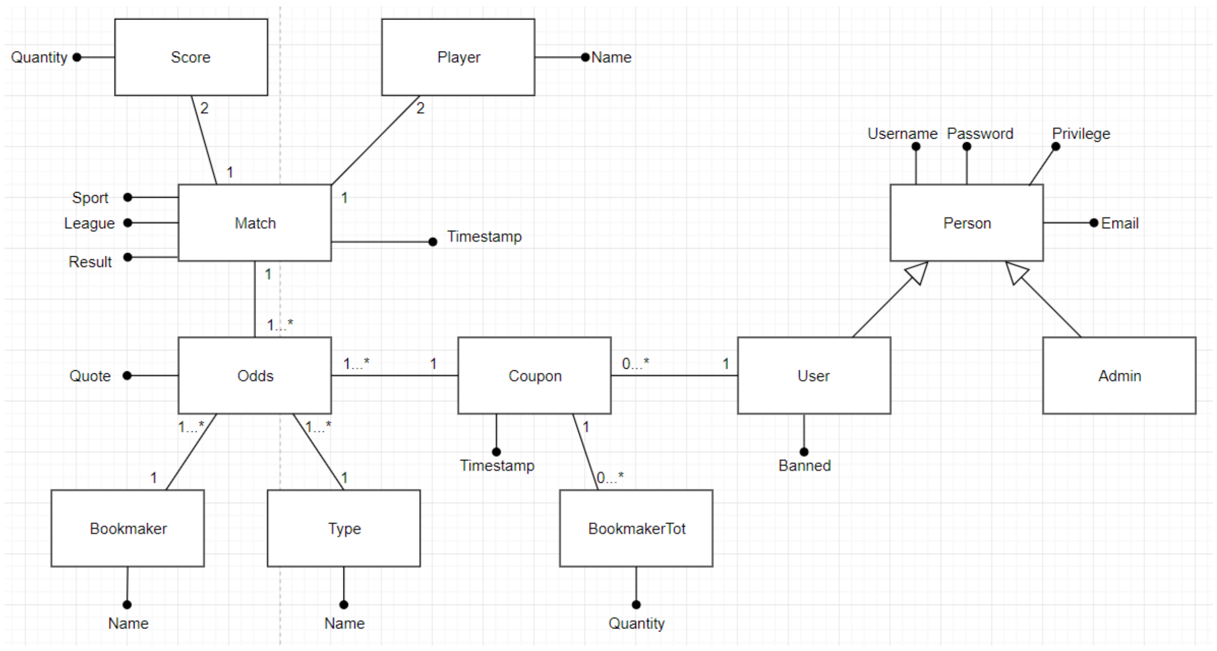


Figure 2: Class Diagram

Each user can create one or more coupons, each of which will be composed of one or more odds and will generate a list of *BookmakerTot* which represent the total multiplier for each available bookmaker.

Odds are characterized by a bookmaker, a type of bet and a unique quote. All matches are made of two players with the relative score and other fields such as the sport, the league, the timestamp and the outcome of the game.

4. DATABASE ORGANIZATION

The dataset used for OrgaBet is a real dataset characterized by a large volume and has the *variety* feature, indeed it is:

- Multi-source: data was collected from multiple betting sites.
- Multi-format: depending on the sport.

After the data collection phase, all the documents were formatted according to our data model (more on this in the next paragraph) and uploaded in a single collection called *match*.

4.1. Data Model

```
                                <User>
{
      username:
      password:
      email:
      name:
      surname:
      roles:[]
      banned:
      coupons: [{
            id
            date:
            bookmakerTot: [{
                        bookmaker:
                        quoteTot:
                        }]
                  bets: [{
                        matchId:
                         homeTeam:
                        awayTeam:
                        result:
                        avgOdd:
                        quotes: [{
                              bookmaker:
                              odd:
                              }]
                         }]
               }]
}
```

The roles are provided by a collection used by Spring Security to authenticate users and restrict access to some functionalities (admin only ones).

<Match>

```
{
      sport:
      division:
      date:
      year:
      time:
      homeTeam:
      awayTeam:
      fullTimeHomeScore:
      fullTimeAwayScore:
      fullTimeResult:
      odds: [{
            type:
            quotes: [{
                    bookmaker:
                    odd:
                    }]
      }]

      //other sport-dependant data in our dataset
}
```

5. SOFTWARE ARCHITECTURE

OrgaBet is a web application programmed with the use of the S*pring Framework,* an application framework for the Java platform.

Spring uses a Model-View-Controller (MVC) paradigm where the *view* layer is handled by *Thymeleaf,* a template engine fully integrated with *Spring*. *Thymeleaf* allows the programmer to extend the functionalities of HTML5 tags by adding new types of tags and options fully accessible by the associated java-written controller class.

The back-end is composed of a *MongoDB* document database which is used by *Spring Data,* a native MongoDB driver for the *Spring Framework*.

Another extension called *Spring Security* is used to handle the user registration and authentication process.

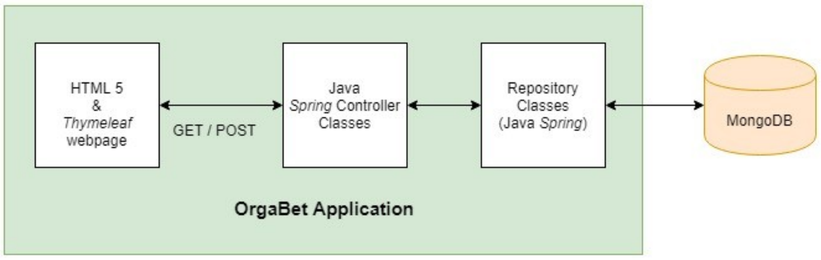

HTML 5 & Thymeleaf webpage
GET / POST
Java Spring Controller Classes
Repository Classes (Java Spring)
MongoDB
OrgaBet Application

Figure 3: Software architecture of OrgaBet

5.1. Repository Structure

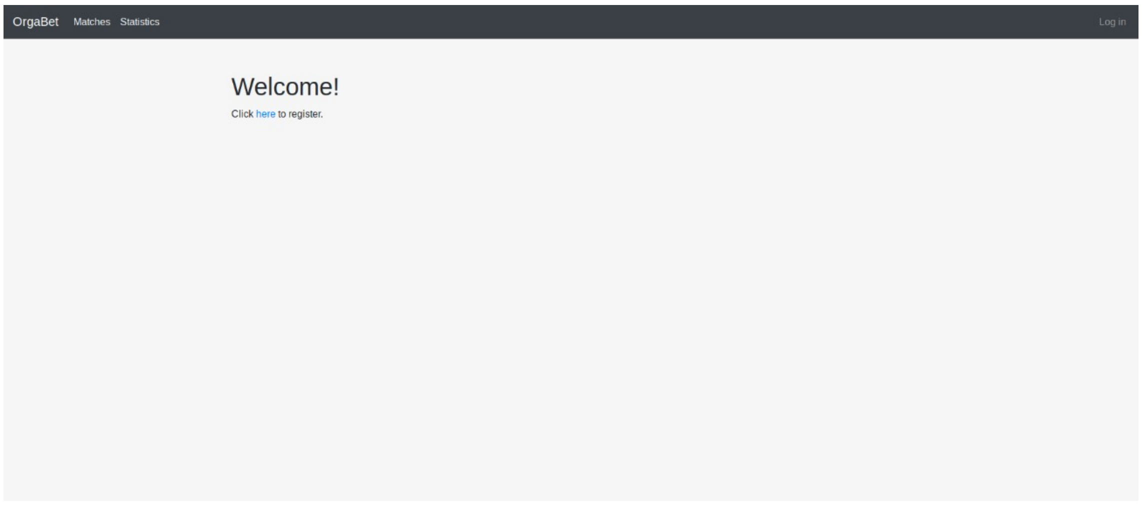
OrgaBet is a *Maven* project. The project repository is organized as follows:

- **`./`** : contains the *makefile* and the *POM* file used to generate the maven dependencies and build the project.
- **`src/main/java/com/example/Orgabet:`** contains all the source files of the application.
    - o **`/controller`**: contains the controller classes.
    - o **`/repository:`** contains the *MongoDB* repository classes, used to store/retrieve data from the database.
    - o **`/dto:`** contains the Data Transfer Object classes.
    - o **`/models:`** contains the Java classes that reflect the entities of the application.
    - o **`/config:`** contains configuration classes used to implement functionalities of the *Spring Security* extension framework like password encoding and customized Authentication handler.
    - o **`/services:`** contains functions used to build authorities on registration and authenticate users.
- **`src/main/resources`**: contains the subdirectories *static* and *templates* which hold all the HTML, CSS and image files used in the application.

6. INSTRUCTION MANUAL

How to use OrgaBet

As soon as a user enters the site, he will visit OrgaBet home page from which he can choose whether to log in (if he is already registered) by clicking in the upper right corner or to register on the application.



Welcome!

Click here to register.

Figure 4: Orgabet home page

Below are the login and registration forms respectively, to sign in only the username and the password are required; to sign up, instead, the user must also provide his first name, his last name and his email address.

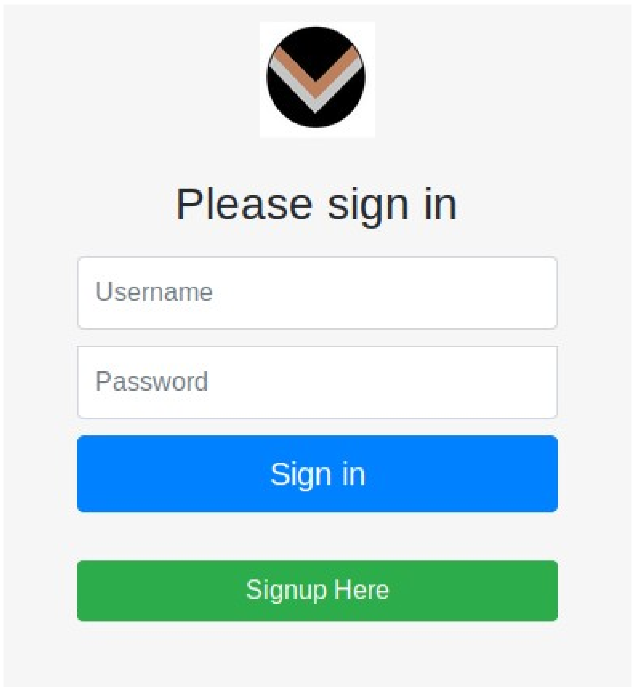


Figure 5: Orgabet login form

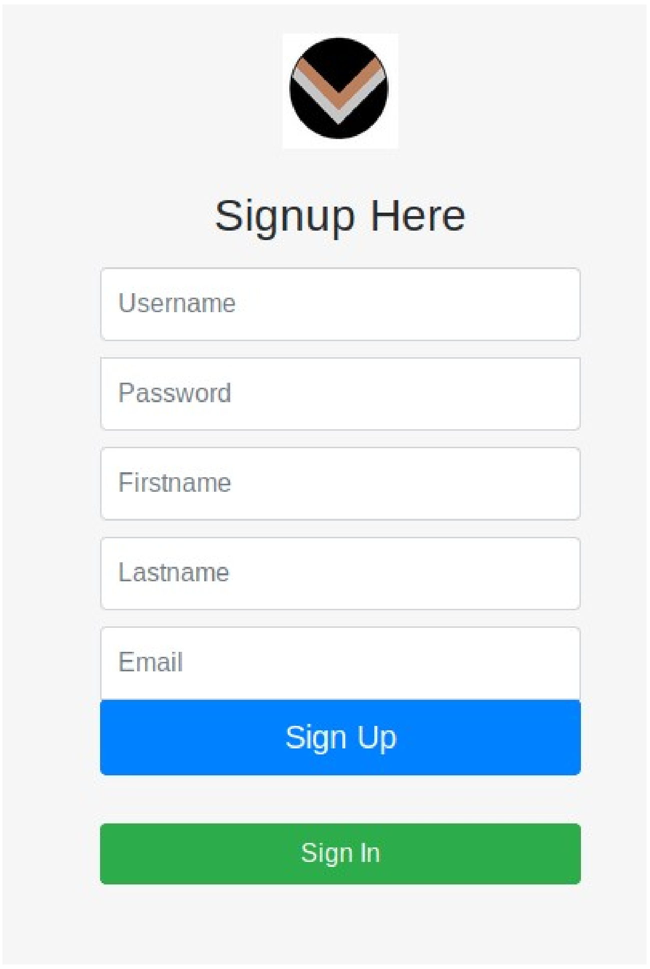
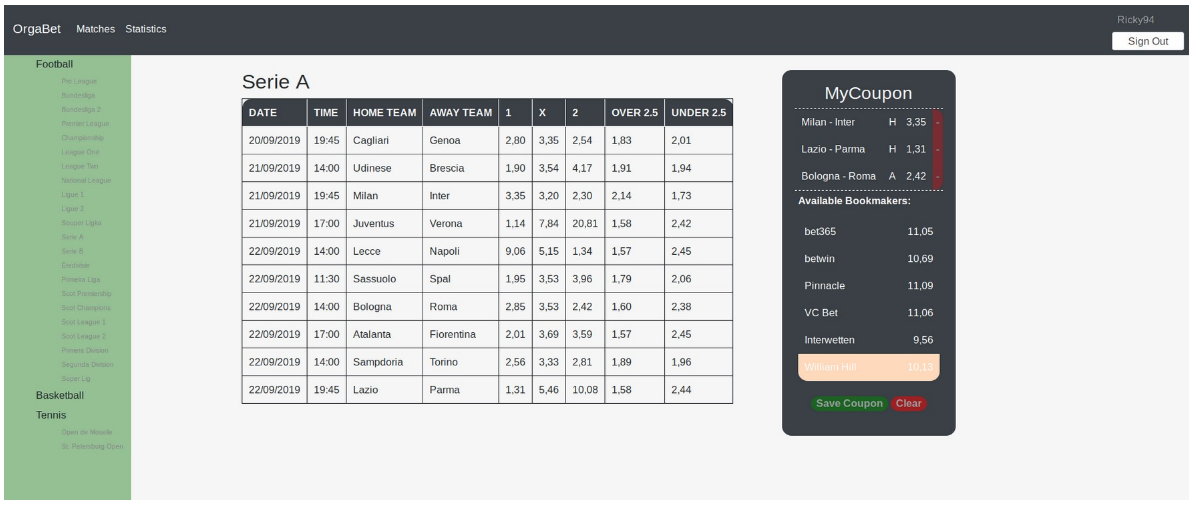


Figure 6: Orgabet registration form

After logging in the user will be redirected to the following page:



Serie A

DATE	TIME	HOME TEAM	AWAY TEAM	1	X	2	OVER 2.5	UNDER 2.5
20/09/2019	19:45	Cagliari	Genoa	2,80	3,35	2,54	1,83	2,01
21/09/2019	14:00	Udinese	Brescia	1,90	3,54	4,17	1,91	1,94
21/09/2019	19:45	Milan	Inter	3,35	3,20	2,30	2,14	1,73
21/09/2019	17:00	Juventus	Verona	1,14	7,84	20,81	1,58	2,42
22/09/2019	14:00	Lecce	Napoli	9,06	5,15	1,34	1,57	2,45
22/09/2019	11:30	Sassuolo	Spal	1,95	3,53	3,96	1,79	2,06
22/09/2019	14:00	Bologna	Roma	2,85	3,53	2,42	1,60	2,38
22/09/2019	17:00	Atalanta	Fiorentina	2,01	3,69	3,59	1,57	2,45
22/09/2019	14:00	Sampdoria	Torino	2,56	3,33	2,81	1,89	1,96
22/09/2019	19:45	Lazio	Parma	1,31	5,46	10,08	1,58	2,44

MyCoupon

Milan - Inter	H	3,35	-
Lazio - Parma	H	1,31	-
Bologna - Roma	A	2,42	-

Available Bookmakers:

bet365	11,05
betwin	10,69
Pinnacle	11,09
VC Bet	11,06
Interwetten	9,56
William Hill	10,13

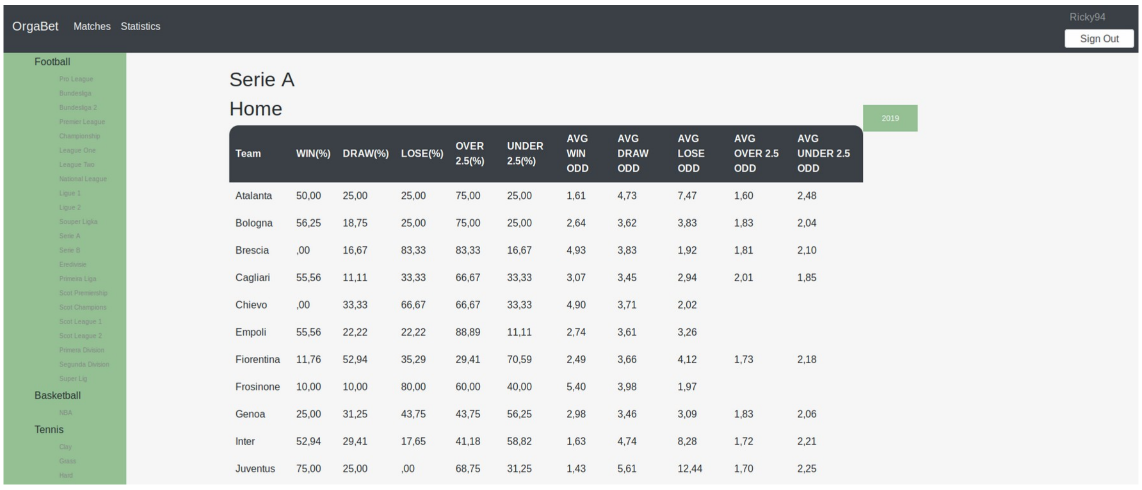
Figure 7: Orgabet matches page

This is the main web page of the application where the consumer can browse the next matches of the sport and the division he is interested in (default homepage is the Italian *Serie A*), with their respective average odds. If the user wishes to bet on a certain result he will click on the corresponding average odd and the event will be added to "MyCoupon", if the customer wants to remove one of the selected results from the coupon he can simply click on the red *minus* sign next to the event.

From the side panel on the left you can select the desired sport and division, if there is at least one match in the next few days, otherwise the championship is not shown.

"My Coupon" is a recap of all selected events. The list will show the total coupon multiplier for each available bookmaker so that the user may choose the most convenient one to bet on. The coupon may be saved by the user for future use or cleared of all events.

From the top bar the user can move to the statistics page shown below:



Serie A

Home

Team	WIN(%)	DRAW(%)	LOSE(%)	OVER 2.5(%)	UNDER 2.5(%)	AVG WIN ODD	AVG DRAW ODD	AVG LOSE ODD	AVG OVER 2.5 ODD	AVG UNDER 2.5 ODD
Atalanta	50,00	25,00	25,00	75,00	25,00	1,61	4,73	7,47	1,60	2,48
Bologna	56,25	18,75	25,00	75,00	25,00	2,64	3,62	3,83	1,83	2,04
Brescia	,00	16,67	83,33	83,33	16,67	4,93	3,83	1,92	1,81	2,10
Cagliari	55,56	11,11	33,33	66,67	33,33	3,07	3,45	2,94	2,01	1,85
Chievo	,00	33,33	66,67	66,67	33,33	4,90	3,71	2,02		
Empoli	55,56	22,22	22,22	88,89	11,11	2,74	3,61	3,26		
Fiorentina	11,76	52,94	35,29	29,41	70,59	2,49	3,66	4,12	1,73	2,18
Frosinone	10,00	10,00	80,00	60,00	40,00	5,40	3,98	1,97		
Genoa	25,00	31,25	43,75	43,75	56,25	2,98	3,46	3,09	1,83	2,06
Inter	52,94	29,41	17,65	41,18	58,82	1,63	4,74	8,28	1,72	2,21
Juventus	75,00	25,00	,00	68,75	31,25	1,43	5,61	12,44	1,70	2,25

Figure 7: Orgabet statistics page

To offer the best service to its users, OrgaBet provides statistics for each year (starting from 2017). The year of interest can be selected from the drop-down menu on the right of the statistics table.

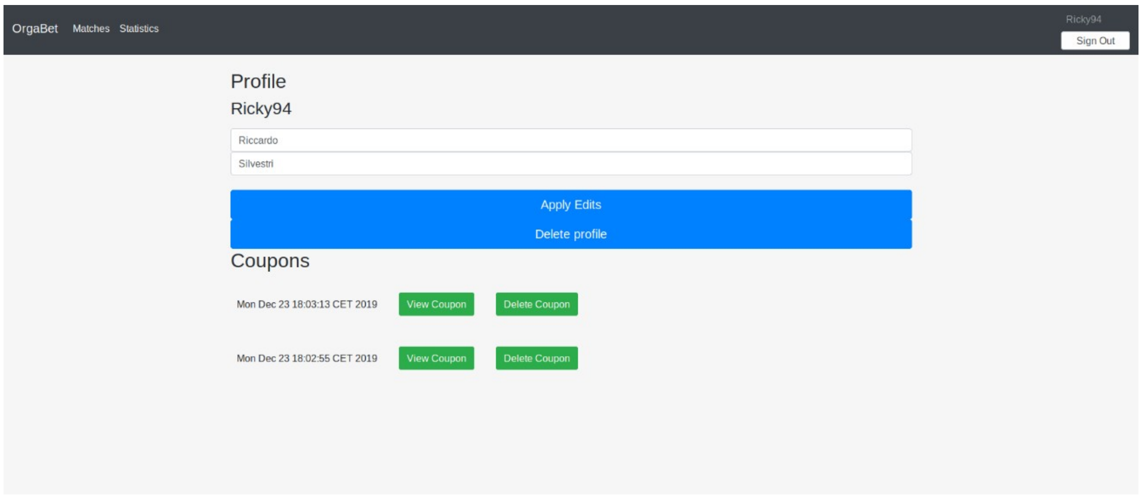
Statistics include:

- **WIN (%):** percentage of won matches, out of total played.
- **DRAW (%):** percentage of draw matches, out of total played (football only).
- **LOST (%):** percentage of lost matches, out of total played.
- **OVER 2.5 (%):** more than two goals in the game percentage (football only).
- **UNDER 2.5 (%):** less than three goals in the game percentage (football only).

And the relative average odds both in home and away games.

While Football and Basketball analytics are divided by championships, Tennis statistics are divided by the type of court surface.

Finally, by clicking on his username at the top right of the web page, the user can visit his personal page or disconnect from the application from the "Sign Out" button just below the username.



Profile

Ricky94

Riccardo

Silvestri

Coupons

Mon Dec 23 18:03:13 CET 2019

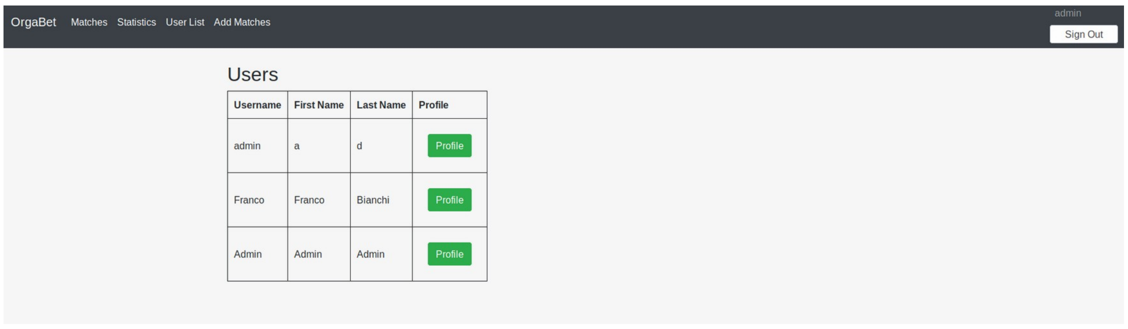
Mon Dec 23 18:02:55 CET 2019

Figure 7: OrgaBet profile page

Inside his personal page the user has the option to edit personal information tied to his account such as his first name or his last name, to view or delete saved coupons and to delete his account.

How to use OrgaBet – Admin side

The site administrator, as mentioned above, has the same functionalities as the regular user but also has additional responsibilities.

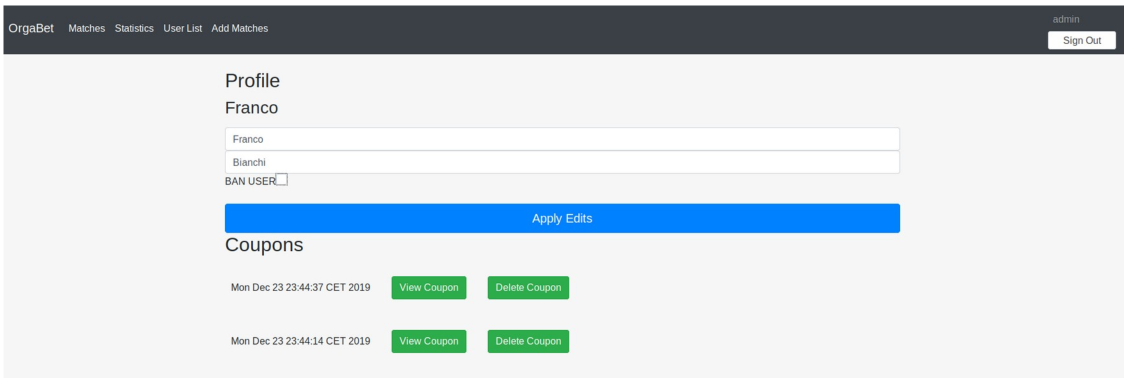


Users

Username	First Name	Last Name	Profile
admin	a	d	Profile
Franco	Franco	Bianchi	Profile
Admin	Admin	Admin	Profile

Figure 8: OrgaBet profile page

By clicking on "User List" on the top bar, the administrator can view the list of users and enter each of their profile pages with the corresponding green button.



Profile

Franco

Franco

Bianchi

BAN USER

Coupons

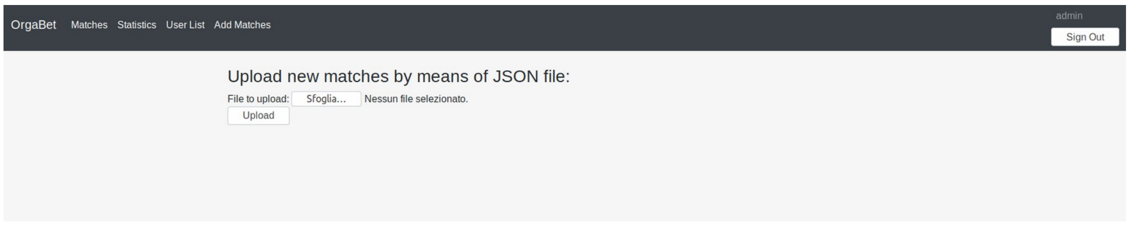
Mon Dec 23 23:44:37 CET 2019

Mon Dec 23 23:44:14 CET 2019

Figure 9: Example profile page

For example, in Figure 9, is shown Franco Bianchi's profile page, and the administrator can, if he deems it necessary, ban the user from OrgaBet. Whenever he wants, "admin" can remove the previously assigned ban.

The second responsibility of the administrator is to upload new matches and odds, to do this he has to select "Add Matches" item in the top bar and will be redirected to the following web page.



Upload new matches by means of JSON file:

File to upload: Sfoglia... Nessun file selezionato.

Figure 10: Example profile page

Here the administrator can browse the desired JSON file and then upload it using "Upload" button.

7. AGGREGATIONS AND INDEXES

7.1. Aggregations

In *OrgaBet* there are two pages which show to the user the main information he can exploit to create his coupon: the first one contains the matches on which he can bet with the relative odds, divided by sport and division; the second one contains the statistics computed for the last three years about all the team (or tennis player), divided by sport and division. Both times this information is computed using MongoDB Aggregation pipelines.

```
                              <Compute Odds Function>
@Override
public List<AvgDTO> computeAverageOdds(String id) {
      MatchOperation filterMatch = Aggregation.match(new Criteria("id").is(id));
      UnwindOperation unw = Aggregation.unwind("odds");
      UnwindOperation unw2 = Aggregation.unwind("odds.quotes");

      GroupOperation grp = Aggregation.group("odds.type").avg("odds.quotes.odd").as("avg");

      Aggregation aggr = Aggregation.newAggregation(filterMatch, unw, unw2, grp);


      List<AvgDTO>res=mongoTemplate.aggregate(aggr,Match.class,AvgDTO.class).getMappedResults();

      return res;
}
```

In the Matches page there is a function called to compute the average odds for every bet type and every match. The following code describes the aggregation:

Code 1: Aggregation to compute average Odds

The function `computeAverageOdds` filters the match collection using the match id in order to find a single match `Aggregation.match(new Criteria("id").is(id))`. Then it uses the *unwind* operations to deconstruct the array field "odds" and "odds.quotes" from the input document in order to produce an output a document for each element. Each output document has the value of the array replaced by a single element `Aggregation.unwind("odds")`.

The group operation

```
Aggregation.group("odds.type").avg("odds.quotes.odd").as("avg")
```

groups the output documents from the previous aggregation, according to the bet type (`"odds.type"`), and computes the average odd for each bet type.

Finally, the aggregation pipeline is created using an Aggregation object:

```
Aggregation aggr = Aggregation.newAggregation(filterMatch, unw, unw2, grp)
```

`It` is then applied on the match collection to get the desired result:

```
List<AvgDTO>res = MongoTemplate.aggregate(aggr, Match.class,
                  AvgDTO.class).getMappedResults()
```

and the function returns it.

In the Statistics page the application shows the team statistics, regarding the winning percentage and the average odds about the different bet type in home or away matches.

```
                                  <Home Statistic Function>
@Override
public StatsDTO computeTeamHome(String division, String team, Double totHome, String sport, String
date) {
      …
      MatchOperation filterDiv = Aggregation.match(new Criteria("division").is(division));
      MatchOperation filterTeam = Aggregation.match(new Criteria("homeTeam").is(team));
      MatchOperation filterHomeWin = Aggregation.match(new Criteria("fullTimeResult").is("H"));
      MatchOperation filterDate = Aggregation.match(new Criteria("date").regex(date+"$"));

      GroupOperation grpHW =  Aggregation.group("homeTeam").count().as("count");
      Aggregation aggr;
      if(sport.equals("Football"))
       aggr = Aggregation.newAggregation(filterDiv, filterTeam, filterDate, filterHomeWin, grpHW);
      else aggr = Aggregation.newAggregation(filterTeam, filterDate, filterHomeWin, grpHW);

      List<countDTO>res=mongoTemplate.aggregate(aggr,Match.class,countDTO.class).getMappedResults();

      try {
            homeWin = ((res.get(0).getCount())/totHome) * 100;
      } catch(Exception e){}

      …
      ProjectionOperation prjHO=Aggregation.project("id").and("fullTimeHomeScore").plus("fullTimeAwayScore").as("totScore");

      MatchOperation filterHomeOver = Aggregation.match(new Criteria("totScore").gt(2));
      GroupOperation grpHO = Aggregation.group().count().as("count");

      Aggregation aggr4;
      if(sport.equals("Football"))
      aggr4=Aggregation.newAggregation(filterDiv,filterTeam,filterDate,prjHO,filterHomeOver,grpHO);


List<countDTO>res4=mongoTemplate.aggregate(aggr4,Match.class,countDTO.class).getMappedResults();

      try {
            homeOver = ((res4.get(0).getCount())/totHome) * 100;
      } catch(Exception e){}

      …

      stats = new StatsDTO(team, homeWin, homeDraw, homeLost, homeOver, homeUnder, res5);

      return stats;
}
```

The function `computeTeamHome` receives in input a team name and the number of home matches regarding the team in a certain year (received in input) and returns a data transfer object (a `StatsDTO` object), previously created to store all the statistics we are interested in.

A `StatsDTO` object contains: the team name, the winning, drawing and losing percentage, the percentage of over/under matches played and the average odds for each bet type.

Code 2: Aggregation to compute Home Statistics

In the previous code we report an extract of the `computeTeamHome`, in particular the aggregation to computes the winning percentage and the one to compute the percentage of over matches. In each of these aggregation we want to filter the match collection basing on division and year (`filterDiv, filterDate`).

In the first aggregation of this function, the function filters the collection on the homeTeam:

```
Aggregation.match(new Criteria("homeTeam").is(team))
```

and then, exploiting a group operation, it calculates the number of match won by the team we are interested in:

```
Aggregation.group("homeTeam").count().as("count")
```

The aggregation pipeline is created and applied on the match collection and, finally, the function computes the winning percentage:

```
homeWin = ((res.get(0).getCount())/totHome) * 100;
```

In the other aggregation reported, the function exploits a *ProjectionOperation* to calculate the total points scored in a match:

```
Aggregation.project("id").and("fullTimeHomeScore").plus("fullTimeAwayScore").as("totScore"));
```

So it can filter the resulting collection by selecting only the matches in which the total score is more than 2.5

```
Aggregation.match(new Criteria("totScore").gt(2))
```

and then counts the number of these matches

```
Aggregation.group().count().as("count")
```

Finally, like in the previous aggregation, the function computes the percentage of over matches:

```
homeOver = ((res4.get(0).getCount())/totHome) * 100;
```

7.2. Indexes

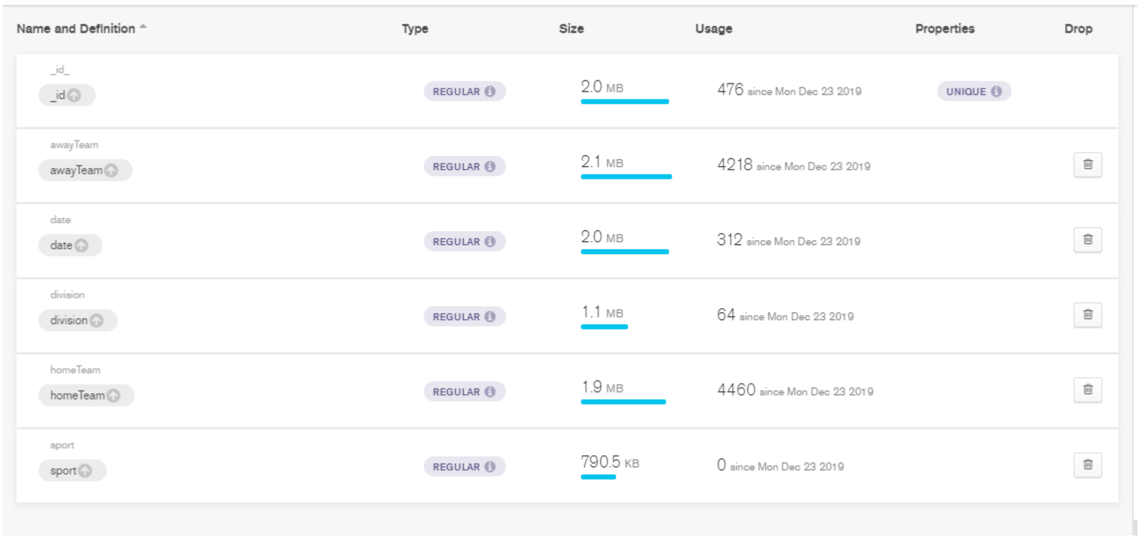
In order to support the efficient execution of queries in MongoDB, *OrgaBet* makes use of indexes.

Without indexes, certain MongoDB queries should scan every document in a collection to select those documents that match the query statement. With an appropriate choice of indices, instead, the number of documents that must be inspected is limited. This is particularly relevant in our case since OrgaBet's database has well over 200 000 documents

Indexes we chose for the *match* collection are the following:

- sport
- division
- date
- homeTeam
- awayTeam
- year

We verified the correctness of our choices by testing the use of indexes through MongoDB application.



Name and Definition	Type	Size	Usage	Properties	Drop
_id_ _id	REGULAR	2.0 MB	476 since Mon Dec 23 2019	UNIQUE	
awayTeam awayTeam	REGULAR	2.1 MB	4218 since Mon Dec 23 2019		🗑
date date	REGULAR	2.0 MB	312 since Mon Dec 23 2019		🗑
division division	REGULAR	1.1 MB	64 since Mon Dec 23 2019		🗑
homeTeam homeTeam	REGULAR	1.9 MB	4460 since Mon Dec 23 2019		🗑
sport sport	REGULAR	790.5 KB	0 since Mon Dec 23 2019		🗑

Figure 11: usage of indexes in OrgaBet

8. SHARDING AND REPLICAS

In order to simulate a realistic scenario, our application was deployed on virtual machines kindly provided by the *University of Pisa*. We were provided 6 machines that have been organized in our deployment of the database as you can see in figure 12.

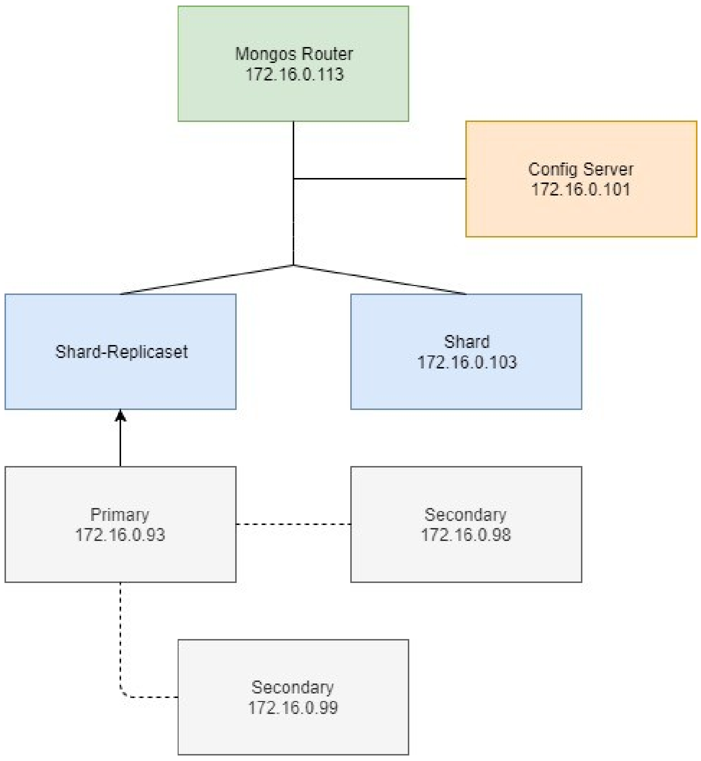


Figure 12: deployment organization diagram

Config Servers:
MongoDB requires the deployment of a replica-set (or three mirrored config servers in previous versions) when you want to setup Sharding. These machines store the metadata for a single sharded cluster including the list of chunks on every shard and the ranges defining chunks. Each sharded cluster must have its own config servers.
As we only had 6 machines available, we provided a single-machine config servers replica-set. If the config servers are unavailable, you can still read and write from shards, but chunk migration or split won't occur until the config replica-set has a primary online.

Mongos Routers:
For a sharded cluster these instances provide the interface between the client and the cluster. From an application perspective this is like any other mongoDB instance. The mongos instances cache the metadata from the config servers and use it to route read and write operation to the correct shards.

9. CONCLUSIONS

The proposed application is only provided with the main features requested for this task however, it would be possible to implement other functionalities that a realistic context would otherwise require.