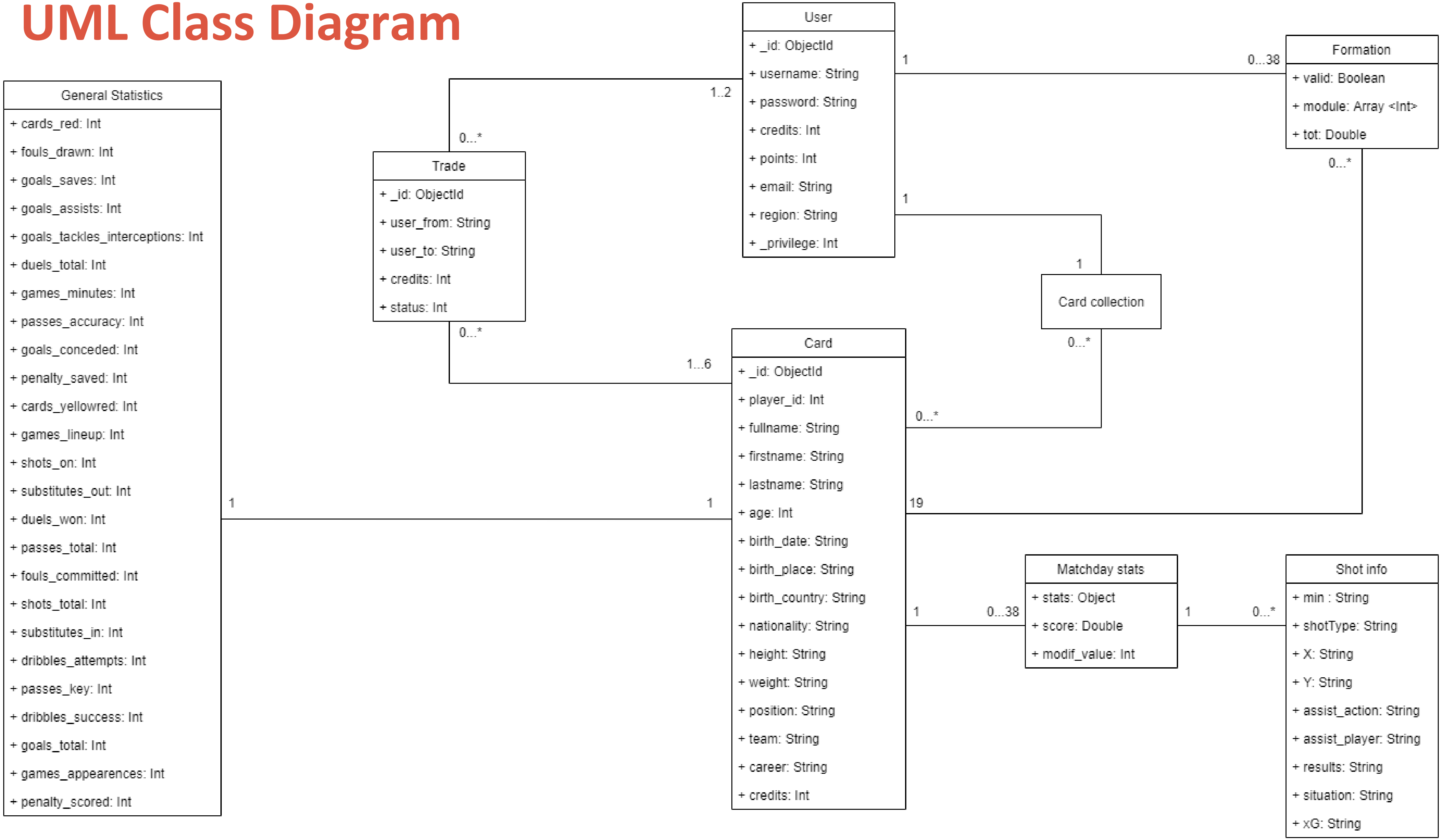
UNIVERSITÀ DI PISA

FantaManager

Large-Scale and Multi-Structured Databases

Emmanuel Piazza, Edoardo Focacci, Matteo Razzai

UML Class Diagram



General Statistics
- + cards_red: Int
- + fouls_drawn: Int
- + goals_saves: Int
- + goals_assists: Int
- + goals_tackles_interceptions: Int
- + duels_total: Int
- + games_minutes: Int
- + passes_accuracy: Int
- + goals_conceded: Int
- + penalty_saved: Int
- + cards_yellowred: Int
- + games_lineup: Int
- + shots_on: Int
- + substitutes_out: Int
- + duels_won: Int
- + passes_total: Int
- + fouls_committed: Int
- + shots_total: Int
- + substitutes_in: Int
- + dribbles_attempts: Int
- + passes_key: Int
- + dribbles_success: Int
- + goals_total: Int
- + games_appearences: Int
- + penalty_scored: Int

Trade
- + _id: ObjectId
- + user_from: String
- + user_to: String
- + credits: Int
- + status: Int

User
- + _id: ObjectId
- + username: String
- + password: String
- + credits: Int
- + points: Int
- + email: String
- + region: String
- + _privilege: Int

Formation
- + valid: Boolean
- + module: Array <Int>
- + tot: Double

Card collection

Card
- + _id: ObjectId
- + player_id: Int
- + fullname: String
- + firstname: String
- + lastname: String
- + age: Int
- + birth_date: String
- + birth_place: String
- + birth_country: String
- + nationality: String
- + height: String
- + weight: String
- + position: String
- + team: String
- + career: String
- + credits: Int

Matchday stats
- + stats: Object
- + score: Double
- + modif_value: Int

Shot info
- + min : String
- + shotType: String
- + X: String
- + Y: String
- + assist_action: String
- + assist_player: String
- + results: String
- + situation: String
- + xG: String

Dataset Description

Source: Wikipedia – Understats – Kickest - APIFootball

Description: Information about players retrieved by scraping. Users’ information and trade offers are randomly generated.

Volume: At the beginning, the database is filled up only with players and users' information. Every week the admin will add the rating of the week for every player.

Variety: Statistics and rating of players retrieved from different sites. Wikipedia page of players used for retrieve a brief description of him.

Velocity/Variability: Stats and rate of every player is retrieved at the end of every matchday. The statistics of the past weeks stays in the database and from them the value of the players is calculated

Non-Functional Requirements

- The application needs to be **consistent** regarding the application’s economy (trade and credits) to provide fair information among users.
- The transactions must be **monotonic**: every user must see the last version of the data and every trade request must be managed in the same order they are accepted.
- The password must be **protected** and stored **encrypted** for privacy issues.
- The system will be **available** 24 * 7.
- A **Graphical User Interface** is crucial for an involving game experience.
- The statistics regarding cards are not needed to be real-time, they are updated periodically, and they will be eventually consistent.

Non-Functional Requirements and CAP Theorem

According to *Non-Functional Requirements*, our system must guarantee a high consistency and stability.

Consistency and **Partitioning** are the edges chosen from the *CAP Triangle (Users and Trades* collection)

It is important that all users see the same version of the data, especially for the *Trades* entity. Only the primary server can accept writes, whereas the secondary in some cases can make the workload of the primary lighter by accepting some read (only from *Cards* entity).

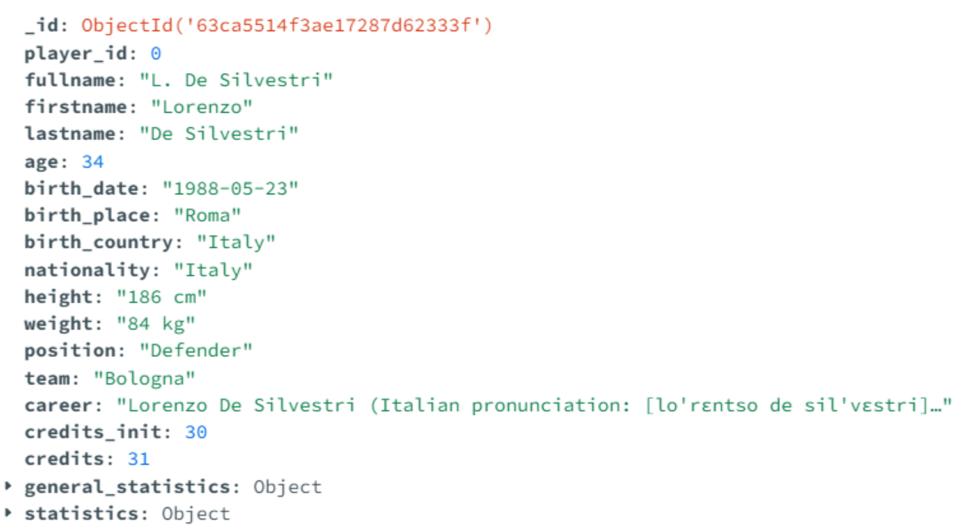
The partition tolerance is guarantee thanks to MongoDB replica set feature design. If one server is down, we can continue to offer out service by searching contents from replicas.

Database design MongoDB (1)

Stored entities:

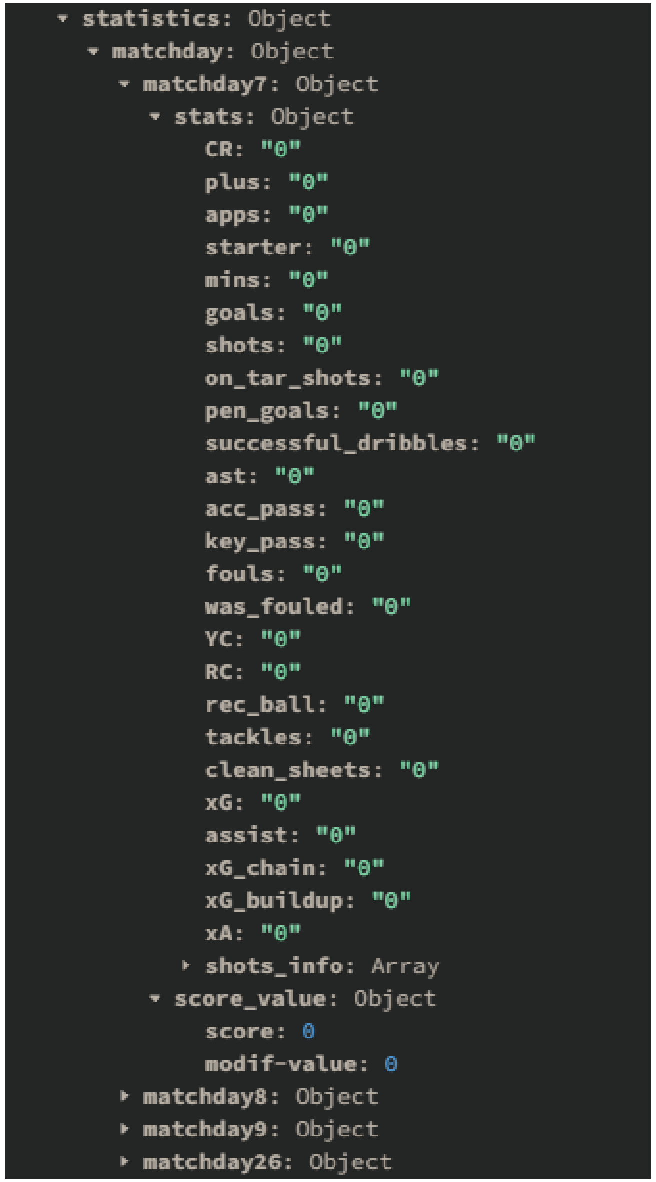
Users – Trades – Cards

Cards entity



```
  _id: ObjectId('63ca5514f3ae17287d62333f')
  player_id: 0
  fullname: "L. De Silvestri"
  firstname: "Lorenzo"
  lastname: "De Silvestri"
  age: 34
  birth_date: "1988-05-23"
  birth_place: "Roma"
  birth_country: "Italy"
  nationality: "Italy"
  height: "186 cm"
  weight: "84 kg"
  position: "Defender"
  team: "Bologna"
  career: "Lorenzo De Silvestri (Italian pronunciation: [lo'rɛntso de sil'vɛstri]…"
  credits_init: 30
  credits: 31
▸ general_statistics: Object
▸ statistics: Object
```

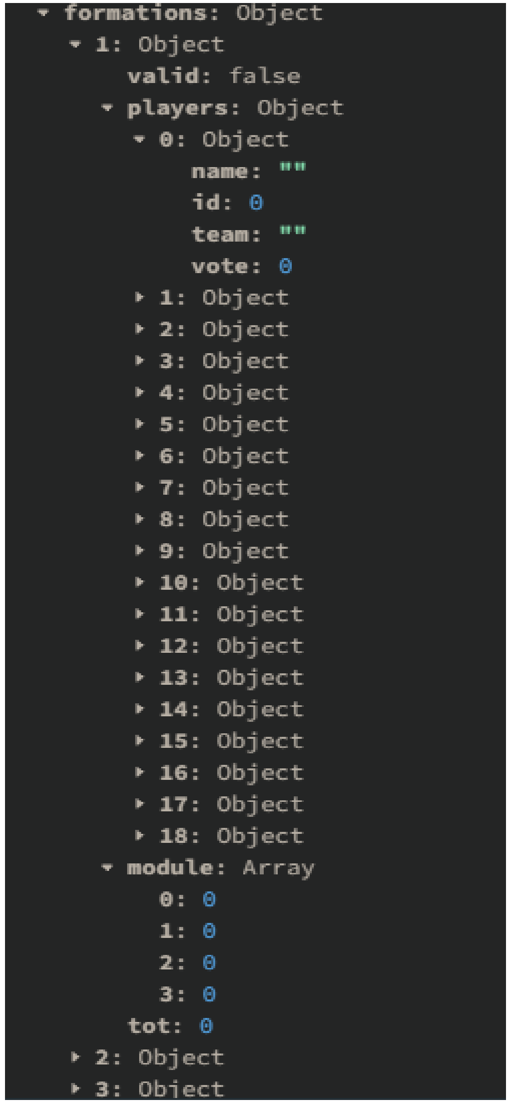
Statistics attribute from *Cards* entity



```
▾ statistics: Object
  ▾ matchday: Object
    ▾ matchday7: Object
      ▾ stats: Object
          CR: "0"
          plus: "0"
          apps: "0"
          starter: "0"
          mins: "0"
          goals: "0"
          shots: "0"
          on_tar_shots: "0"
          pen_goals: "0"
          successful_dribbles: "0"
          ast: "0"
          acc_pass: "0"
          key_pass: "0"
          fouls: "0"
          was_fouled: "0"
          YC: "0"
          RC: "0"
          rec_ball: "0"
          tackles: "0"
          clean_sheets: "0"
          xG: "0"
          assist: "0"
          xG_chain: "0"
          xG_buildup: "0"
          xA: "0"
        ▸ shots_info: Array
      ▾ score_value: Object
          score: 0
          modif-value: 0
    ▸ matchday8: Object
    ▸ matchday9: Object
    ▸ matchday26: Object
```

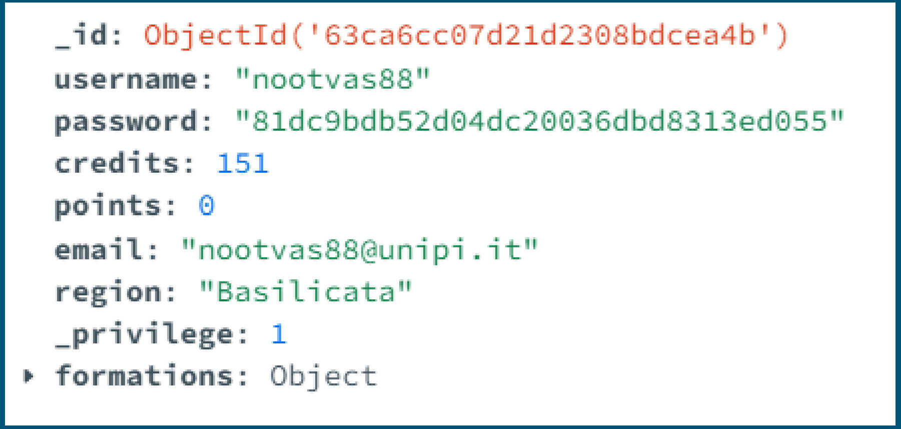
Database design MongoDB (2)

Formations attribute from *Users* entity



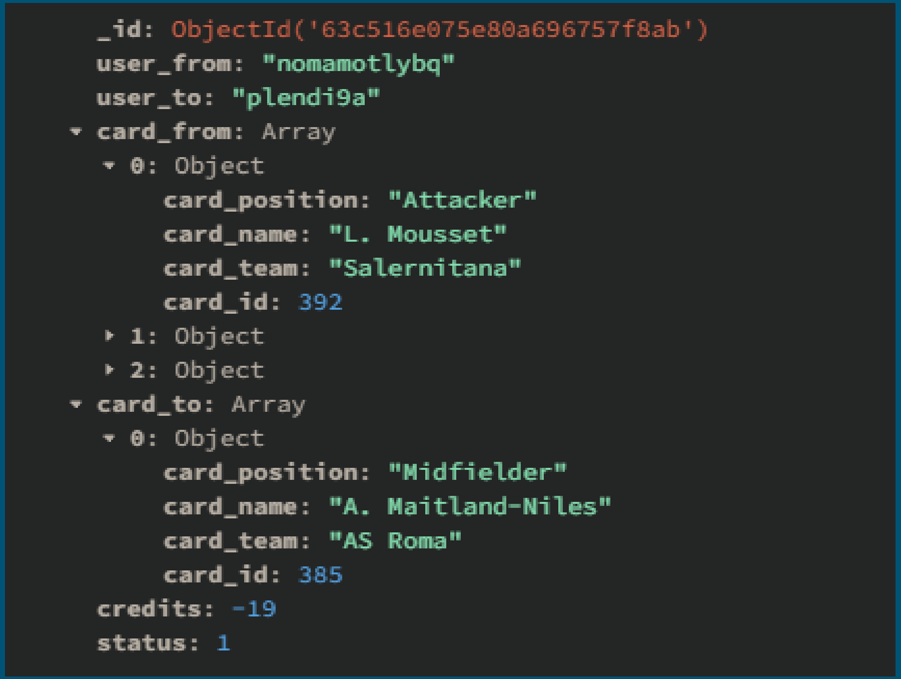
```
▾ formations: Object
  ▾ 1: Object
      valid: false
    ▾ players: Object
      ▾ 0: Object
          name: ""
          id: 0
          team: ""
          vote: 0
      ▸ 1: Object
      ▸ 2: Object
      ▸ 3: Object
      ▸ 4: Object
      ▸ 5: Object
      ▸ 6: Object
      ▸ 7: Object
      ▸ 8: Object
      ▸ 9: Object
      ▸ 10: Object
      ▸ 11: Object
      ▸ 12: Object
      ▸ 13: Object
      ▸ 14: Object
      ▸ 15: Object
      ▸ 16: Object
      ▸ 17: Object
      ▸ 18: Object
    ▾ module: Array
        0: 0
        1: 0
        2: 0
        3: 0
      tot: 0
  ▸ 2: Object
  ▸ 3: Object
```

Users entity



```
  _id: ObjectId('63ca6cc07d21d2308bdcea4b')
  username: "nootvas88"
  password: "81dc9bdb52d04dc20036dbd8313ed055"
  credits: 151
  points: 0
  email: "nootvas88@unipi.it"
  region: "Basilicata"
  _privilege: 1
▸ formations: Object
```

Trades entity



```
  _id: ObjectId('63c516e075e80a696757f8ab')
  user_from: "nomamotlybq"
  user_to: "plendi9a"
▾ card_from: Array
  ▾ 0: Object
      card_position: "Attacker"
      card_name: "L. Mousset"
      card_team: "Salernitana"
      card_id: 392
  ▸ 1: Object
  ▸ 2: Object
▾ card_to: Array
  ▾ 0: Object
      card_position: "Midfielder"
      card_name: "A. Maitland-Niles"
      card_team: "AS Roma"
      card_id: 385
  credits: -19
  status: 1
```

Database design Redis (1)

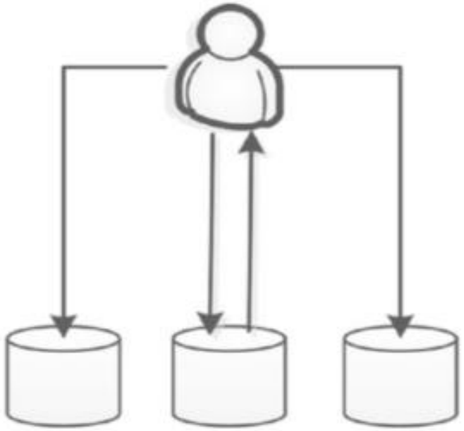
- **user_id:*xxxx*:card_id:*xxx*:name** – contains the name of the card owned by the user.
- **user_id:*xxxx*:card_id:*xxx*:position** – contains the position of the card owned by the user.
- **user_id:*xxxx*:card_id:*xxx*:team** – contains the name of the team of the card owned by the user.
- **user_id:*xxxx*:card_id:*xxx*:quantity** – contains the number of copies of the same card owned by the user.

Admin only:

- **admin:next_matchday –** is read by every user after the login and indicates the number of the matchday that is currently playing. Only the admin can update this value.
- **admin:updated_matchdays –** is an array that contains a list of the updated matchday, only the admin can read and write the value of the key. It is used to know which matchday has been updated and which not. The importance of this key is for situation in which the calculation of a matchday, for some reason, has been postponed (for example: a lot of matches has been postponed for some reason, like we saw during the pandemic)

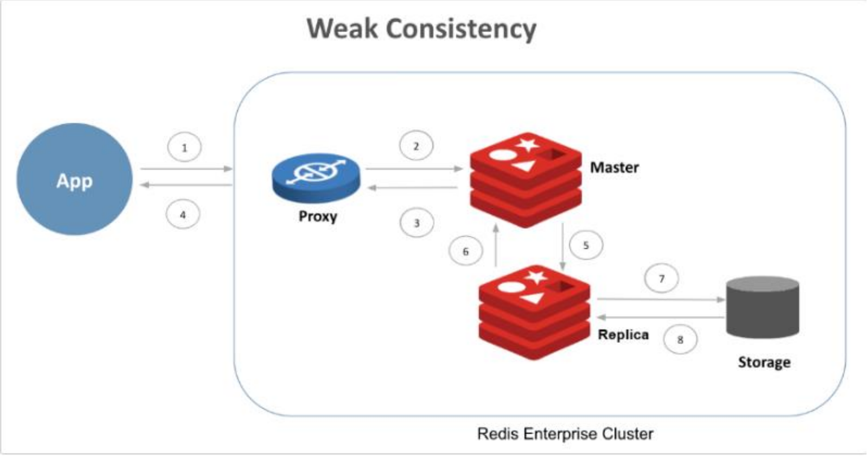
Database design Redis (2)

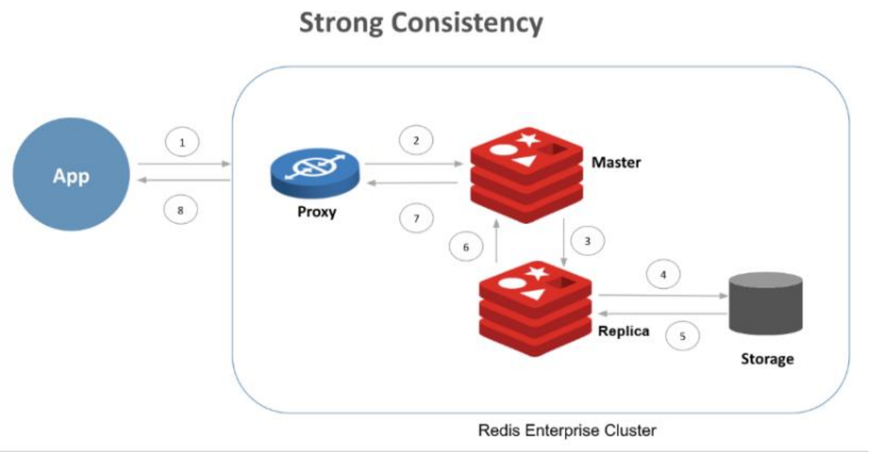
We decided to ensure a consistent read of the values.



N=3 W=3 R=1
Slow writes, fast reads, consistent
There will be 3 copies of the data.
A write request only returns when all 3 have written to disk.
A read request only needs to read one version.

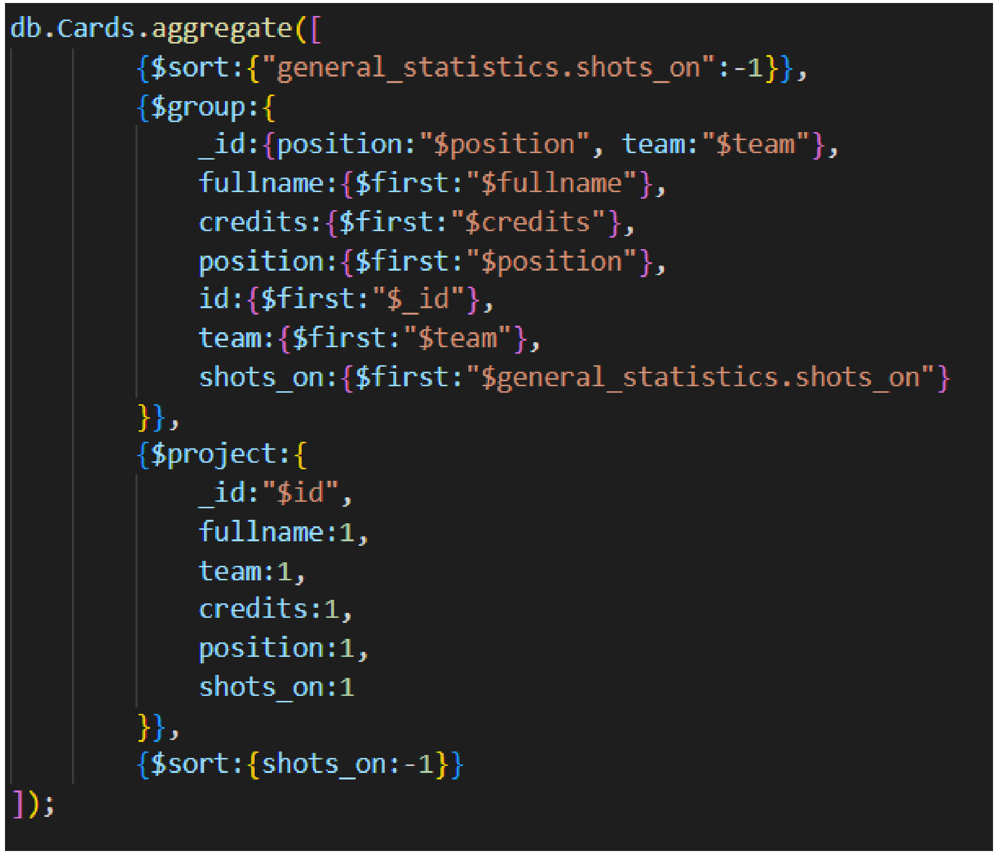
In order to do that, the *WAIT* option of Redis replication was used.
It confirms that the write is performed on at least *n* replicas.





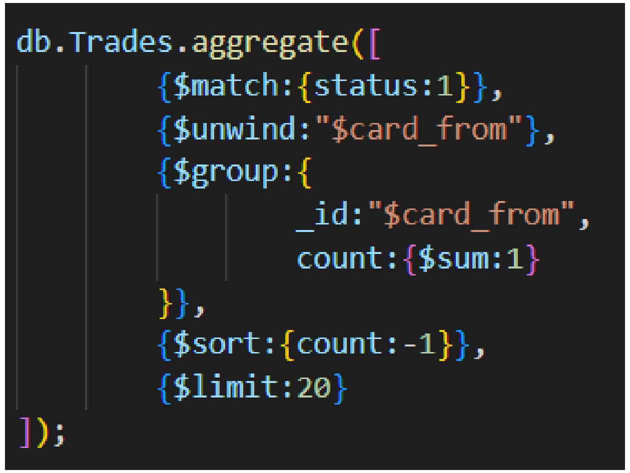
Main Queries (1)

Best Cards by skill (general statistics field)



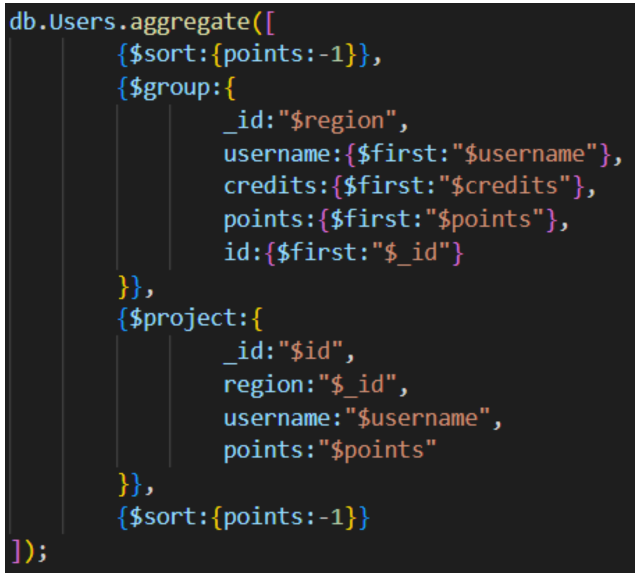
```
db.Cards.aggregate([
        {$sort:{"general_statistics.shots_on":-1}},
        {$group:{
            _id:{position:"$position", team:"$team"},
            fullname:{$first:"$fullname"},
            credits:{$first:"$credits"},
            position:{$first:"$position"},
            id:{$first:"$_id"},
            team:{$first:"$team"},
            shots_on:{$first:"$general_statistics.shots_on"}
        }},
        {$project:{
            _id:"$id",
            fullname:1,
            team:1,
            credits:1,
            position:1,
            shots_on:1
        }},
        {$sort:{shots_on:-1}}
]);
```

Most offered/wanted footballer's card in Trades



```
db.Trades.aggregate([
        {$match:{status:1}},
        {$unwind:"$card_from"},
        {$group:{
                _id:"$card_from",
                count:{$sum:1}
        }},
        {$sort:{count:-1}},
        {$limit:20}
]);
```

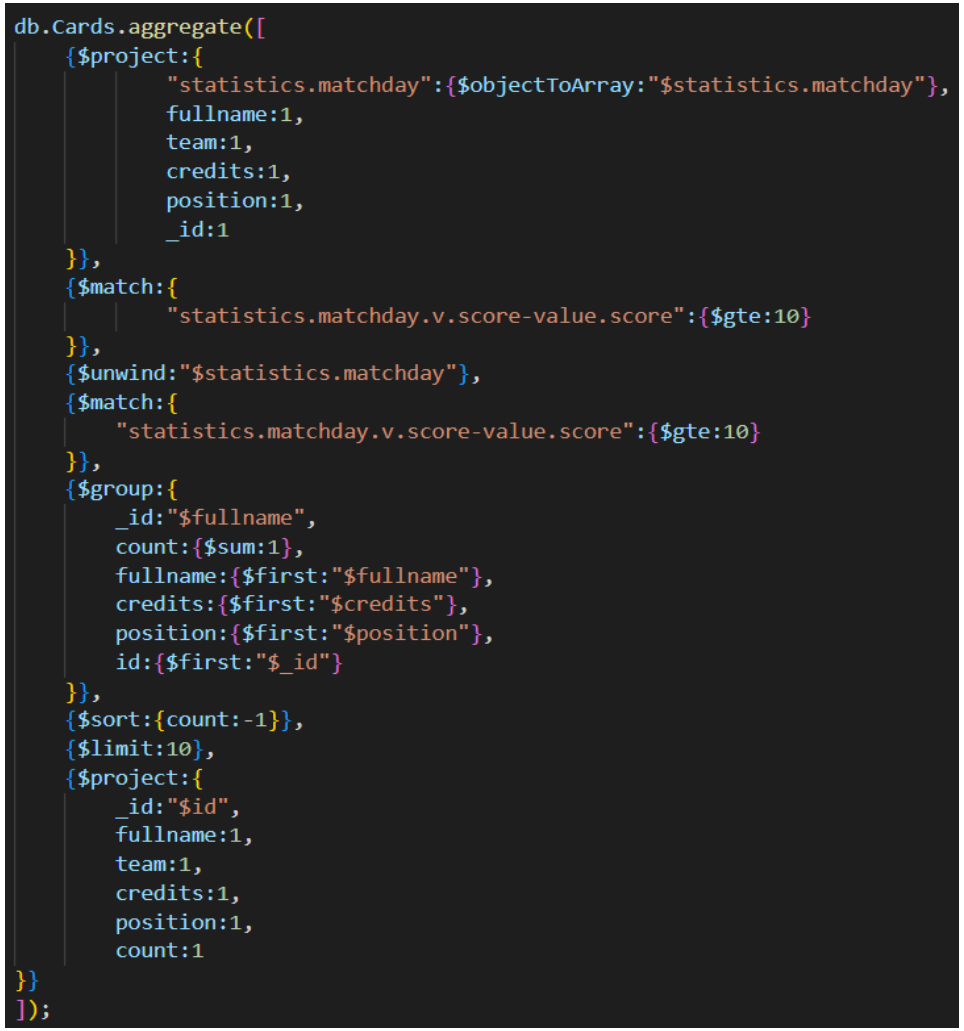
Best User for each region



```
db.Users.aggregate([
        {$sort:{points:-1}},
        {$group:{
                _id:"$region",
                username:{$first:"$username"},
                credits:{$first:"$credits"},
                points:{$first:"$points"},
                id:{$first:"$_id"}
        }},
        {$project:{
                _id:"$id",
                region:"$_id",
                username:"$username",
                points:"$points"
        }},
        {$sort:{points:-1}}
]);
```

Main Queries (2)

Best Cards for matchday’s score



```
db.Cards.aggregate([
    {$project:{
            "statistics.matchday":{$objectToArray:"$statistics.matchday"},
            fullname:1,
            team:1,
            credits:1,
            position:1,
            _id:1
    }},
    {$match:{
            "statistics.matchday.v.score-value.score":{$gte:10}
    }},
    {$unwind:"$statistics.matchday"},
    {$match:{
        "statistics.matchday.v.score-value.score":{$gte:10}
    }},
    {$group:{
        _id:"$fullname",
        count:{$sum:1},
        fullname:{$first:"$fullname"},
        credits:{$first:"$credits"},
        position:{$first:"$position"},
        id:{$first:"$_id"}
    }},
    {$sort:{count:-1}},
    {$limit:10},
    {$project:{
        _id:"$id",
        fullname:1,
        team:1,
        credits:1,
        position:1,
        count:1
}}
]);
```

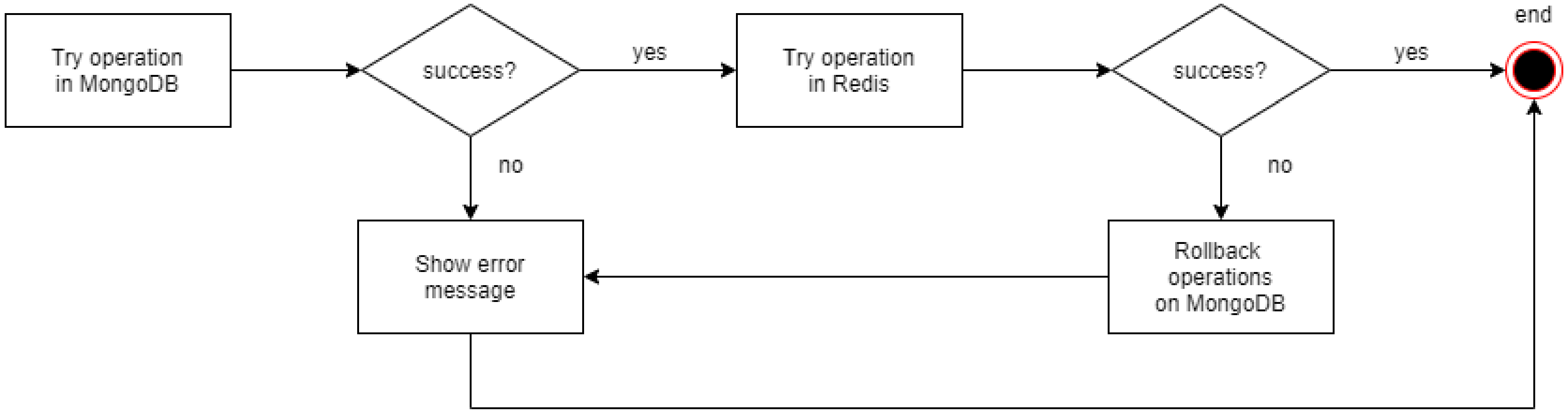
Operations on Redis

- Retrieve user’s cards collection
- Delete a card from user’s cards collection
- Add a card from user’s cards collection
- Change quantity of a card in user’s cards collection
- Delete all the cards owned by a specific User

Consistency

We must consider the problem of consistency between the two databases in the following cases:

- **Create, accept, delete a Trade →** Mongo operations are made first; if successful, Redis operations are done next, if fails, all documents are restored to their previous state.
- **Delete a user →** Redis operation are made first; if successful, Mongo operations are done next, if fails, all values are restored. In this case, is simplier reinsert the user's card collection into Redis instead of restore the deleted trades.



Sharding, a possible implementation

- For the *Users* collection we can consider the region field as a shard key and a partitioning method based on list. In this way user's information are segmented by country on the cluster and each server, locally deployed, would serve the incoming requests for the users' region. For regions that produce a critical amount of requests we should consider a higher number of servers and add some more replicas.
- For the *Trades* collection we can consider the *_id* as a shard key and a partitioning method based on Hashing. Trades information are not easily divisible by other means, but a Hashing algorithm on the *_id* field makes it possible.
- For the *Cards* collection we thought that a sharding method is not that useful, as its size is limited and known. However, considering for example the role as a shard key, a partitioning method based on list is possible.

Software and Hardware architecture

Programming Language:

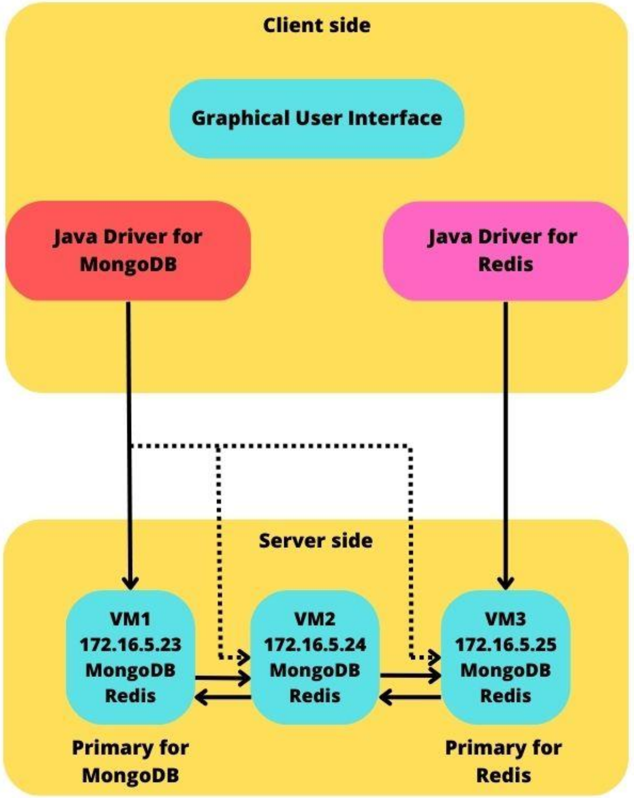
- JavaFX
- Java
- Python

DBMSs:

- MongoDB
- Redis

Frameworks:

- Maven



Final considerations

The current architecture made us think for possible implementations by making small adjustment that doesn’t change the whole system:

- Possibility for a card to change team
- Possibility to add/delete cards from the application
- Possibility to create private *Leagues*

We decided to add some indexes to improve the performance of the queries.

MongoDB collection	Attribute
Users	username
Users	region
Trades	user_from
Trades	Status
Cards	fullname

THANKS FOR YOUR ATTENTION

GitHub repo: https://github.com/Fochi1999/FantaManager

CREDITS: This presentation template was created by **Slidesgo**, including icons by **Flaticon**, and infographics & images by Freepik.

Please keep this slide for attribution.