****

*Large Scale and Multi-structured Databases*

*PokeMongo: Project Documentation*

Edoardo Fazzari, Mirco Ramo, Olgerti Xhanej

Summary

[1. Introduction 3](#_Toc58866628)

[2. Analysis stage 4](#_Toc58866629)

[2.1 Functional requirements and use cases 4](#_Toc58866630)

[2.1.1 Use Cases List 4](#_Toc58866631)

[2.1.2 UML Use Case Diagram 7](#_Toc58866632)

[2.2 Non-functional requirements 8](#_Toc58866633)

[2.3 Sources, velocity properties and volume of data 8](#_Toc58866634)

[3.4Uml Relation Diagram 9](#_Toc58866635)

[4.Queries and Database Structure 10](#_Toc58866636)

[4.1 Main DB queries 10](#_Toc58866637)

[4.2 Json collections 10](#_Toc58866638)

[5 Graph db 11](#_Toc58866639)

# Introduction

PokeMongo is a gaming application in which users compete each other to build up the best Team choosing between the set of Pokemon available in the environment; they can also follow other users in order to make new friends basing on common friends or common interests. Moreover users can express sentiments on Pokémon, choosing their favorite ones and posting/commenting on them.

Every trainer (normal user) can build up his own team. Every Team is composed by up to 6 distinct Pokémon and is assigned to a numerical value (points) based on features and properties of the chosen Pokémon, for ranking purposes.

Users can also navigate through the ranking in order to visualize the best teams (according to the values cited before) and the most used/caught Pokémon, both among their friends, grouped by country and among worldwide players.

User can also browse for a specific Pokémon using the Pokédex tool, in which he/she can lookup for Pokémon according to search filters like Pokémon name, Type or Points.

Moreover, as a “real” Pokémon Trainer, the user is invited to *Catch ‘em‘ all*, i.e. to try to get a new Pokémon in order to create/update his/her own team. Thus, it is provided to the user a prefix number of daily Pokéball to be used to try to capture them. At each Pokémon is associated a probability to catch it, the higher the Pokémon’s value, the lower the probability.

Furthermore, the user can exploit the social network structure of the application to make new friends and discover new Pokémon. Indeed, he/she can search for new friends by username or choosing them among the provided *recommended friends* list.

The user can choose his/her favorite Pokémon, obtaining in this way a shortcut to catch it faster, and can *post* or *comment* to *posts* in order to express his/her opinion on that Pokémon.

In addition, to extend the dynamic behavior of the application, the *catch rate* (i.e. the probability to get a Pokémon using a Pokéball) changes in time depending on the number of users who have that Pokémon: the more it is popular, the harder will be to catch it. Since the rankings’ points are computed based on the *catch rate*, the winning strategy could be on predicting which Pokémon will become popular in the near future and try to get it early! Every user has access to the visualization of the temporal drift of the *catch rate*.

The safeguard and the improvement of the application is in charge of admin users. They are able to ban mischievous users, delete inappropriate posts, add/remove Pokémon to the collection, consult geo-temporal usage statistics which are useful to make new business plans.

# Analysis stage

## 2.1 Functional requirements and use cases

### 2.1.1 Use Cases List

* An unregistered user can
  + - * + Register
* A registered user
  + - * + Sign in
        + Consult Pokédex:

Search by name

Search by type(s)

Search by Pokédex ID

Search by catch rate

Search by points

Search by Pokémon characteristics like height or weight

* + - * + Consult rankings:

Most popular Pokémon among all users

Most popular Pokémon among friends

Most popular Pokémon in each country

Best world team

Best friends’ team

Best team by country

* + - * + Find users:

See recommended users based on common friends

See recommended users based on common Pokémon interests

Find users by username

Follow them

Unfollow them

* + - * + Interact with Pokémon network:

Insert a Pokémon in his/her own favorite Pokémon list

Remove a Pokémon from the favorite ones

Create a post on a Pokémon to share opinions

Comment to posts

Add answers to comments

The post owner can also remove the post at his/her will

* + - * + Team handling:

View team

Remove Pokémon from the team

Change name to the team

Save modified team

View the value of the team

* + - * + Catching:

Browse a Pokémon you want to catch searching it by name

Select a Pokémon you want to catch from the list of favorites

Try to catch the Pokemon to add to your team

* + - * + Settings:

Change email

Change password

Change country

* + - * + Logout:

Exit from the account

Return to the sign in window

* + - * + At each time can:

See the remaining daily Pokéballs

Start/Mute music

By clicking on a Pokémon name, visualize all the information about it

* An admin can

Add Pokémon to the Pokédex

Remove Pokémon from the Pokédex

See number of registered users in time

See number of logins per day

See number of logins per day in every country

Remove a user from the system

Remove posts from the network

* The system should

Daily update Pokéball number of each user

Periodically update Pokémon’ catch rates based on the number of users that own that Pokémon

Update team points if the user has 6 Pokémon of different types

Periodically compute usage statistics to be consulted by the administrators

### 2.1.2 UML Use Case Diagram

(\*) Only for the user who created the post

Browse-find-view comments and browse-find-view answers had not been reported

## 2.2 Non-functional requirements

* The application should guarantee a high availability
* It should be easy to use, especially for children and youngsters, and enjoyable
* It should have a read-your-own-writes consistency on each user’s own team, so he/she can always be sure that Pokémon have been correctly caught/freed up
* The application should always provide to each user the most recent version of the rankings in order to permit him/her to immediately verify his/her progresses
* The statistics regarding usage and catch rate evolution are not needed to be real-time, they can be updated periodically and be eventually consistent
* Posts, comments and answers must follow a causal-consistency
* Response time is an important issue: redundancies and larger memory consumptions are preferred over high latencies
* Passwords are crypted for security reasons
* A graphical interface and the usage of multimedia are crucial for an involving game experience

## 2.3 Sources, velocity properties and volume of data

Data stored in the application backend has been downloaded and imported from the following sources:

[*https://pokeapi.co*](https://pokeapi.co/), [*https://bulbapedia.bulbagarden.net/wiki*](https://bulbapedia.bulbagarden.net/wiki/Main_Page) 🡪 Pokémon data

[*https://gist.github.com/kalinchernev/486393efcca01623b18d*](https://gist.github.com/kalinchernev/486393efcca01623b18d) 🡪 Countries data

[*https://github.com/smashew/NameDatabases/blob/master/NamesDatabases/surnames/all.txt*](https://github.com/smashew/NameDatabases/blob/master/NamesDatabases/surnames/all.txt) 🡪 Data for the generation of realistic users

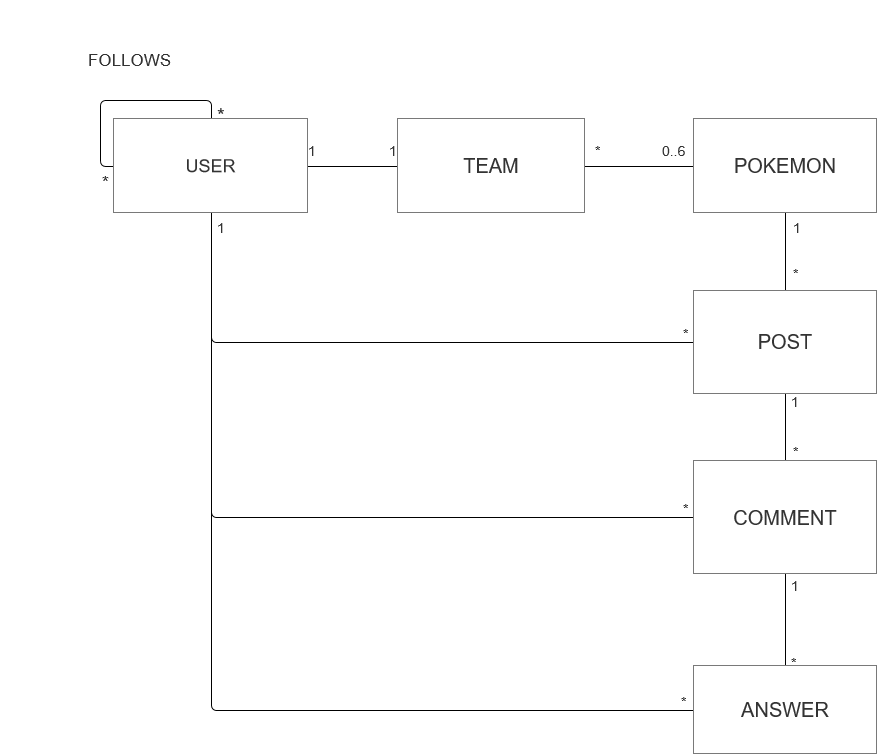
All the imported data has been modified, updated and preprocessed in order to satisfy the application needs.

Users added have the only purpose of showing the application functionalities, privacy issues they are not real people; anyway they have been created using realistic criteria.

Velocity is guaranteed by the dynamic *catch rate* mechanism: the popularity of a Pokémon influences both its *catch rate* and the amount of points that it will provide. As a consequence, users are continuously stimulated by catching new Pokémon, in order to try to raise their amount of points: in this way old teams’ data becomes quickly out-of-date.

Volume of data, considering 250K users, almost 1K Pokémon and about 500K posts is no lower than 100Mb

2.4 Uml Relation Diagram



A user can build up only 1 team: of course, each team has just one owner.

A team is composed of a maximum of 6 Pokemons, every Pokemon can be caught by anyone, so can belong to many teams.

# 4.Queries and Database Structure

## 4.1 Main DB queries

* Retrieve user information
* Retrieve team information based on user
* Retrieve Pokemon catch rates
* Retrieve Pokemon using several filters
* Retrieve recommended users
* Retrieve a user by username
* Retrieve list of a user’s friends
* Analytics: most popular Pokemons[by country], best ranked teams [among friends/by country], evolution on time of catchRates
* Analytics: evolution on time of # of users/logins per day (admin related)
* Create a user
* Remove a user (admin related)
* Modify user settings
* Update team (add/remove Pokemons)
* Remove/Add Pokemons from the database (admin related)

## 4.2 Json collections

Since they are very different Entities, User and Pokemon need their own collection.

In particular, for an admin user, some field are not needed: they have been eliminated.

As seen in the paragraph 3.3, the Team is strictly related with the user it was created by, so the most natural solution is to embed Team’s data into the collection *user*, so that to retrieve it faster.

Furthermore, a Team is an Array of Pokemon: that means that it is possible to replicate each Pokemon’s attribute into it. Anyway this solution is not scalable: the system is made of several Pokemon shared among many users, and each Pokemon is characterized by an high amount of attributes: the embedding of these documents would cause an exponential grow of storage occupation.

For this reason, it has been chosen to make the field Team an Array of Pokemon IDs.

Immagine che contiene monitor, schermo, computer, elettronico

Descrizione generata automaticamenteImmagine che contiene monitor, schermo, elettronico, computer

Descrizione generata automaticamenteThe final result is summarized in the two following collections, default value have the only purpose to show the type of fields.

# 5 Graph db

1. Query su graph e traduzione delle query

Retrieve of Pokemons composing a team ==🡺 Retrieve Pokemon Nodes connected to a User node

Retrieve recommended users 🡺 Retrieve User nodes at distance 2 from a input User Node

Retrieve most popular pokemon 🡺 Count # of edges associated to each Pokemon Node

1. Struttura del db (nodi-archi)

Pokemon

{pokId}

User

User

{username}

has

follows