UNIVERSITÀ DI PISA

Computer Engineering, Artificial Intelligence and Data Engineering

Large-Scale and Multi-Structured Database

PokèMongo

Project Documentation

TEAM MEMBERS:
Edoardo Fazzari
Mirco Ramo
Olgerti Xhanej

Academic Year: 2020/2021

Contents

1 — Introduction

PokeMongo is a gaming application in which users compete each other to build up the best Team choosing between the set of Pokémon available.

1.1 Description

Every **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.

A **User** 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 or commenting on them.

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 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 *Point*s.

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 answer 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 or comments*, *add/remove Pokémon* to the collection, *consult geo-temporal usage statistics* which are useful to make new business plans.

2 — Analysis

2.1 Functional Requirements and Use Cases

2.1.1 Use Cases List

- An *unregistered user* can
    - Register
- A *registered user* can
    - Login
    - Consult Pokèdex
        - Search by Name
        - Search by Type(s)
        - Search by Pokédex ID
        - Search by Catch Rate
        - Search by Points
        - Search by Pokemon characteristics like Height or Weight
    - Consult ranking:
        - Most popular Pokèmon among all Users
        - Most popular Pokèmon in each Country
        - Best World Teams
        - Best Teams among Friends
        - Best Teams by Country
    - Find Users:
        - See recommended users based on common friends
        - See recommended users based on common Pokémon interests
        - Find users by username
        - Follow/Unfollow them
    - Interact with Pokèmon network:
        - Insert/Remove a Pokémon in his/her own favorite Pokémon list
        - Create a post on a Pokémon to share opinions
        - Add answers to posts

    - Follow/Unfollow them
    - The post owner can also remove the post at his/her will
  - Team handling:
    - Remove Pokemon from the team
    - View team
    - Change name of 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 a 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 Pokèballs
    - Mute/Unmute Music
    - By clicking on a Pokémon name, visualize all the information about it

- An *admin* can
  - Sign In
  - Add Pokèmon to the Pokédex
  - Remove Pokèmon from the Pokédex
  - See the number of registered Users in time
  - See the numbers of login per day
  - See the numbers of login per day in every Country
  - Remove a User from the system
  - Remove Posts/Answers from the system

  - Consult Rankings
  - Logout

- The *system* should
  - Daily update Pokeball number of each user
  - Periodically update Pokemon catch rates based on the number of users that own that pokemon
  - 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 Cases Diagram

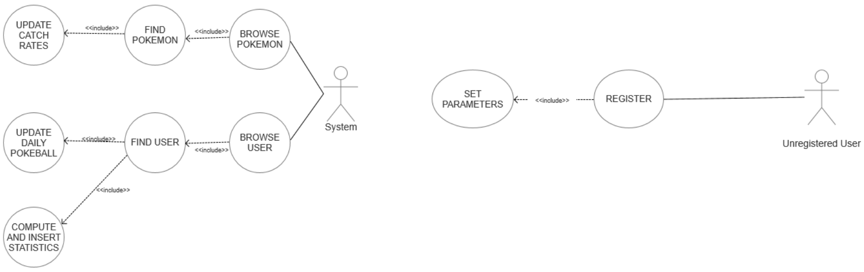

Figure 1: Use Case Diagram 1

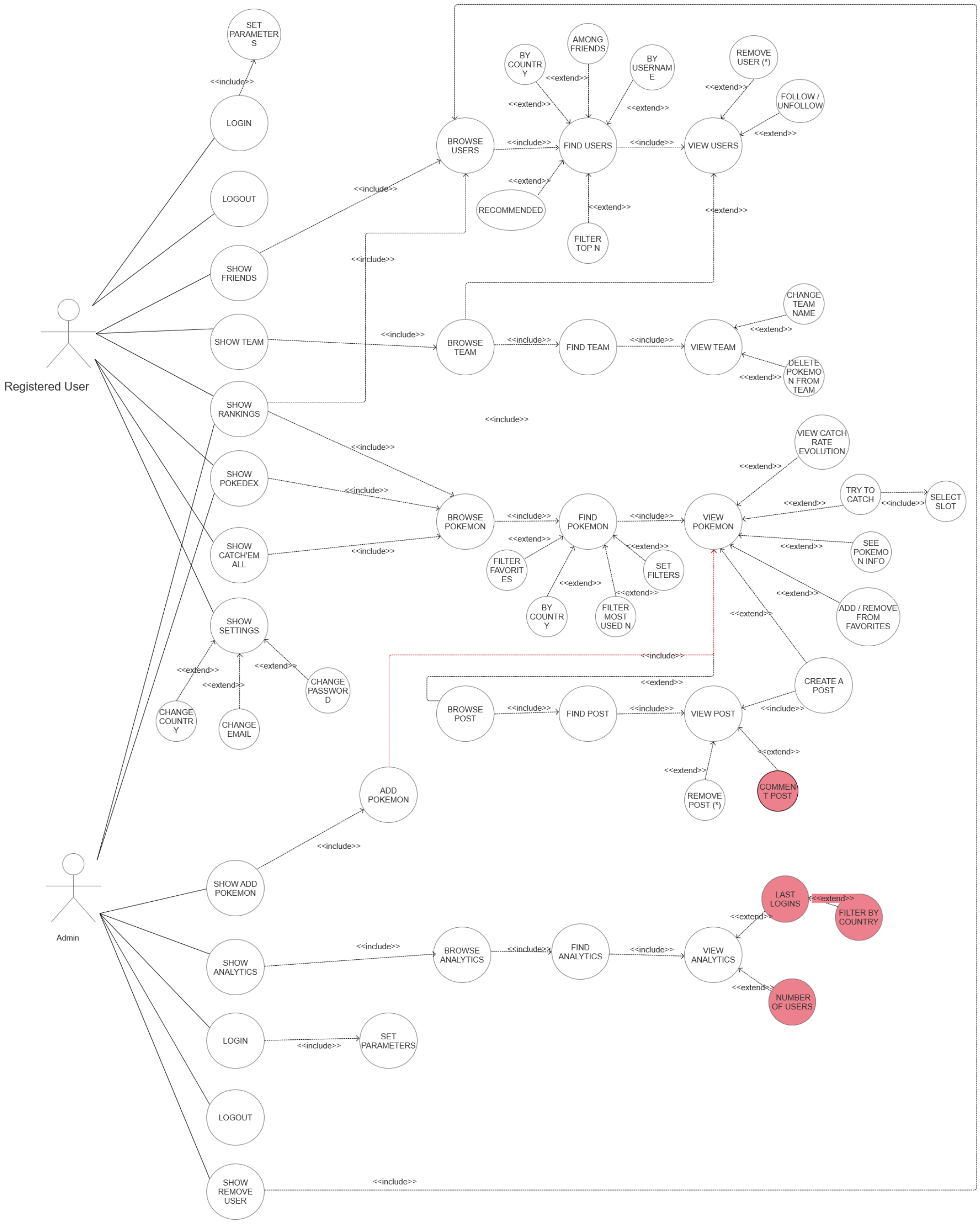

Figure 2: Use Case Diagram 2
(*) only for the User who created the Post and Admins, in Red Browse-find-view comments and browse-find-view answers had not been reported)

2.2 Non-Functional Requirements

- The application should guarantee a high availability. 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:

1. **Pokèmon Data** → `https://pokeapi.co`, `https://bulbapedia.bulbagarden.net/wiki`
2. **Countries data** → `https://gist.github.com/kalinchernev/486393efcca01623b18d`
3. **Data for the generation of realistic users** → `https://github.com/smashew/NameDatabases/blob/master/NamesDatabases/surnames/all.txt`

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, **for 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 Entities Diagram

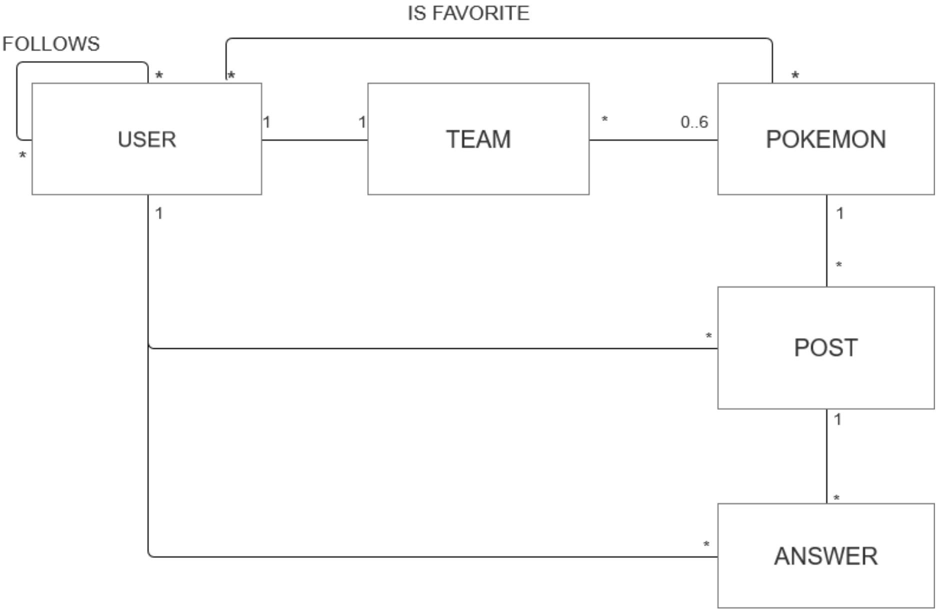

Figure 3: UML Entity Diagram

1. A **User** can build up only one **Team**: of course, each **Team** has just one owner.

2. A **Team** is composed of a maximum of six **Pokémon**, every **Pokémon** can be caught by anyone, so can belong to many **Teams**.

3. A **User** can follow many **Users**, in the meanwhile he/she can have many followers.
4. A **User** can have many favorites **Pokémon**. A **Pokémon** can be favorite of many **Users**.
5. A **Post** is created just by one **User** on one **Pokémon**. A **User** can create many posts and a **Pokémon** can have many **Posts** talking about it.
6. An **Answer** is written by one **User** and it refers to one **Post**. **Users** can submit many Answers and there can be many **Answers** behind a **Post**.

2.5 Main application queries

- Insert a **User** into the system at registration time
- Create a new **Pokémon** (admin only)
- Insert a **Pokémon** into a **Team**
- Create a new **Post**
- Create a new **Answer**
- Create a follow relationship
- Add a **Pokémon** to the favorites
- Retrieve **User** information at login time
- Retrieve a **User** by username when looking for a new friend
- Retrieve **Team** information based on user
- Retrieve **Pokémon** information using several filters
- Retrieve recommended **Users**
- Retrieve list of a **User**'s friends
- Retrieve a **Pokémon** by name when trying to catch it
- Retrieve all the **Posts** relative to a Pokémon
- Retrieve all the **Answers** to a **Post**

- Retrieve **User**'s favorite **Pokémon**
- Modify **User** settings (email, password, country)
- Update **Team**'s name
- Update **Team**'s points
- Update **Pokémon**'s catch rates Analytics: find % of **Users** that own that **Pokémon**
- Remove a **User** (admin only)
- Remove a **Pokémon** (admin only)
- Remove a **Post** (only admin and post's owner)
- Remove a follow relationship
- Remove a **Pokémon** from the favorite ones
- Analytics: ranking of most popular **Pokémon** in world/each country
- Analytics: ranking of best **Teams** in the world/each country/among friends
- Analytics: evolution on time of a **Pokémon** catch rate
- Analytics: evolution on time of number of logins per day/total **Users**/logins per day by country (admin only)

3 — Project

3.1 Adopted Databases

3.2 Document Database

3.2.1 Queries Handled

3.2.2 Entities handled

3.2.3 Collections structure

3.2.4 Indexes

3.3 Graph Database

3.3.1 Queries Handled

3.3.2 Entities handled

3.3.3 Graph structure

3.3.4 Indexes

3.4 Redundancies and consistency management

3.5 Database Properties

3.5.1 Availability

3.5.2 Replicas

3.5.3 Eventual Consistency

3.5.4 Sharding

3.5.5 Pros and Drawbacks

3.6 Client, Server and Daemon Thread

3.7 Technologies and Frameworks

4 — Implementation

4.1 Package structure and information hiding

4.1.1 Packaging strategy and information hiding

4.1.2 UML package diagram

4.2 APIs and SPIs

4.3 Main tools

4.3.1 GSON

4.3.2 Caching mechanism and multimedia management

4.3.3 Password Encryptor

4.3.4 Logger

4.4 Analytics queries

4.4.1 User Rankings

4.4.2 Pokémon Rankings

4.4.3 Usage Statistics

4.4.4 Dynamic Catch Rate

4.5 Business logic

4.5.1 Points computing

4.5.2 Dynamic Catch Rate Computing

5 — Test

5.1 Privacy and Security

5.2 Unit Test

5.3 Robustness

5.4 Performance