****

*Large Scale and Multi-structured Databases*

*PokeMongo: Project Documentation*

Edoardo Fazzari, Mirco Ramo, Olgerti Xhanej

Summary

[1. Introduction 3](#_Toc56958848)

[2. Interface Mock-up 4](#_Toc56958849)

[3. Requirements 7](#_Toc56958851)

[3.1 Functional requirements and use cases 7](#_Toc56958852)

[3.2 Non-functional requirements 8](#_Toc56958853)

[3.3 Uml Relation Diagram 8](#_Toc56958854)

[4.Queries and Database Structure 9](#_Toc56958855)

[4.1 Main DB queries 9](#_Toc56958856)

[4.2 Json collections 9](#_Toc56958857)

[5 Further Ideas 10](#_Toc56958858)

# Introduction

PokeMongo is a gaming application in which users compete each other to build up the best Team choosing from the set of Pokemon available in the environment. Every user can make just one single Team.

Every Team is composed by up to 6 distinct Pokemons, and is assigned to a numerical value based on features and properties of the chosen Pokemons, for ranking purposes.

Users can also navigate through the ranking in order to visualize the best teams (according to the values cited before), most used/caught Pokemons.

The user can also search a specific Pokemon using the Pokedex tool, in which he/she can browse Pokemons according to specific search filters(eg. Pokemon name, Type, Points…).

Moreover, as a “real” Pokemon Trainer, user is invited to “Catch ‘em ‘all”, i.e. to catch Pokemon in order to create/update his own team. Thus, it is provided to the user a prefix number of daily Pokeball to be used to try to catch them.

At each Pokemon is associated a probability to catch it, the higher the Pokemon’s value, the lower the probability.

Under discussion are the following ideas:

-Creating a “social” structure in which users can follow each other in order to share his/her own team

-Creating a chat system to pair with the social structure

-Reduce catchable Pokemons to a daily subset of the entire Pokemon Database

# Interface Mock-up

LOGIN

## 

HOMEPAGE

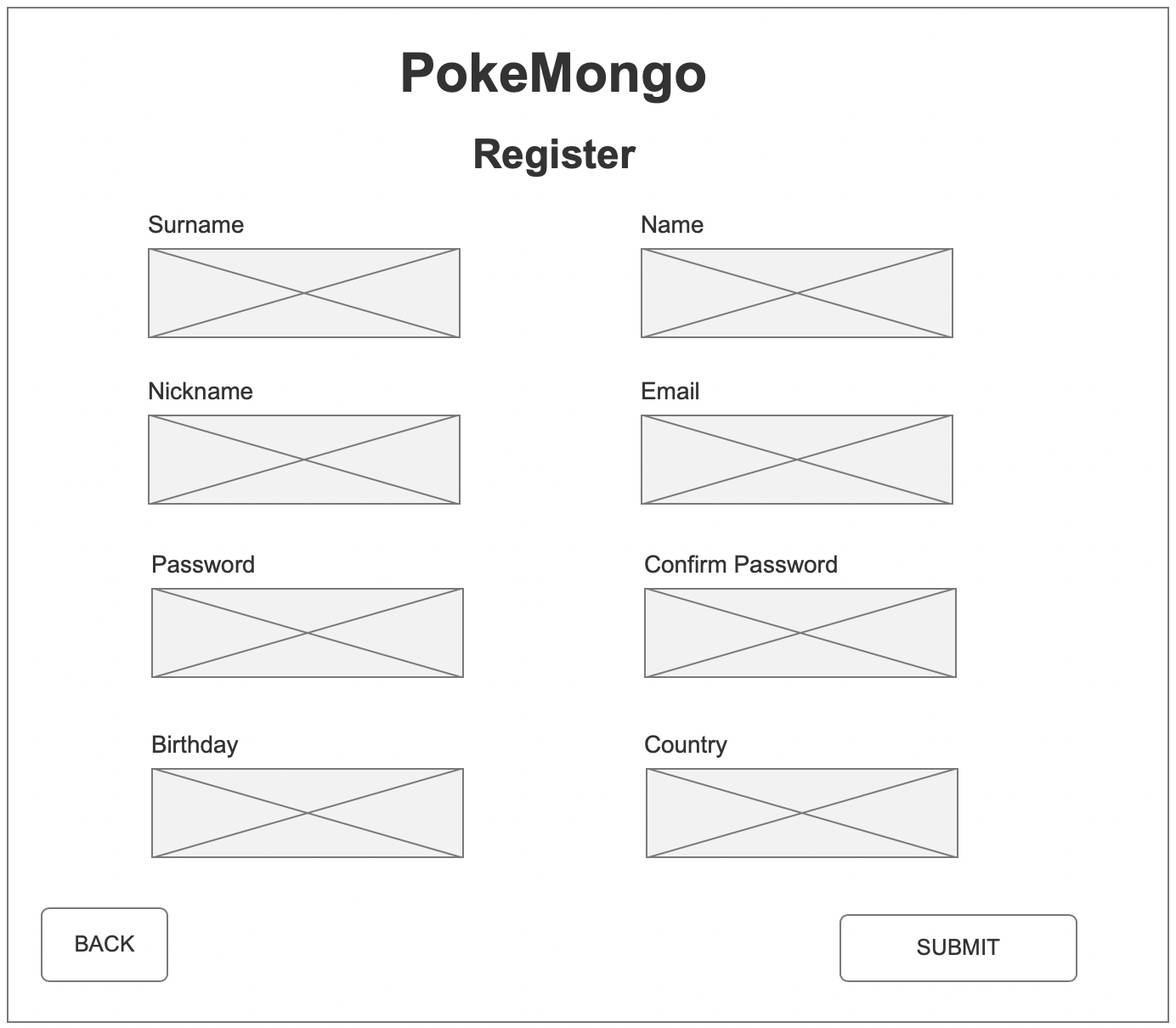
Black: everybody

Blue: just the normal user

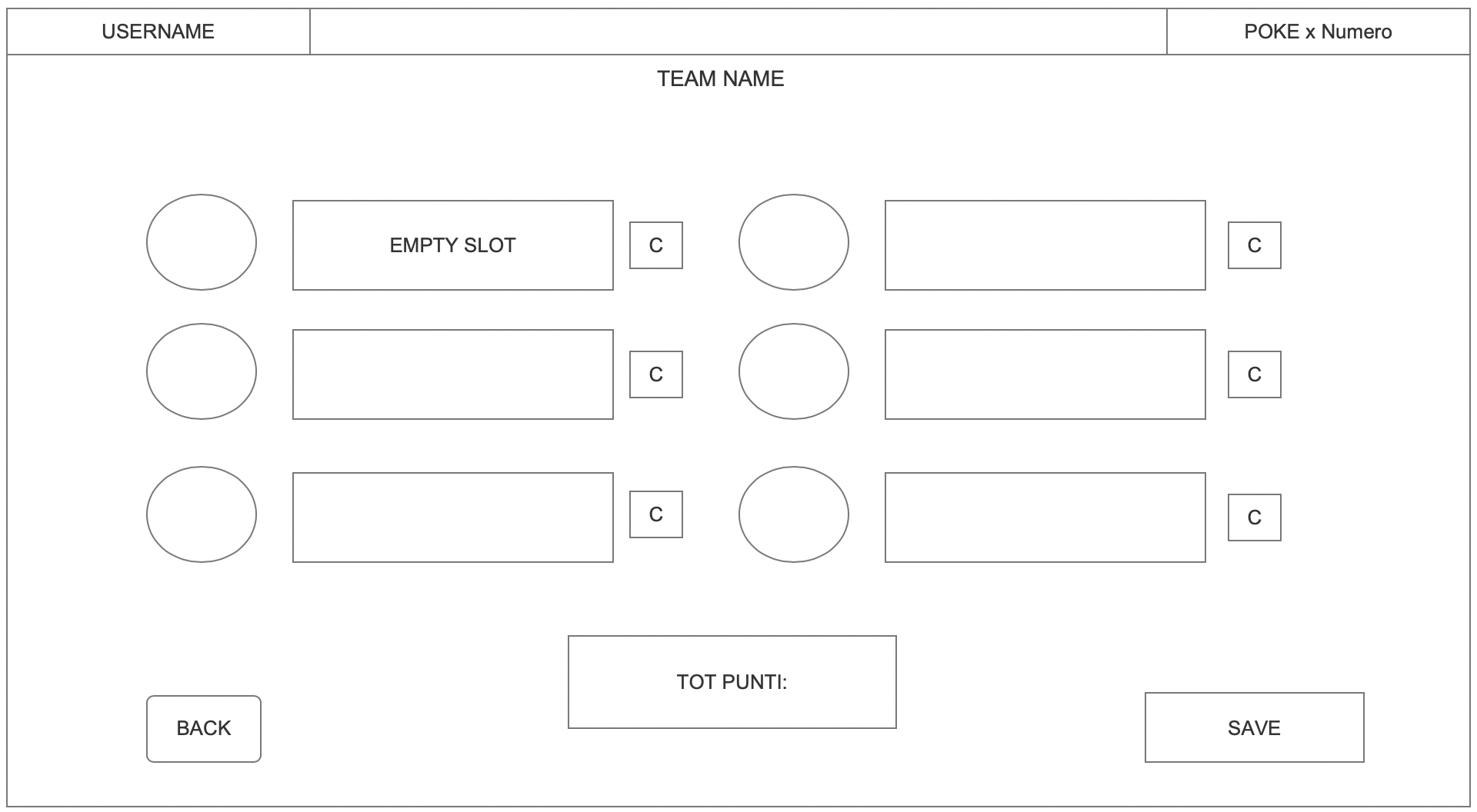
Red: just the admin



SIGNUP



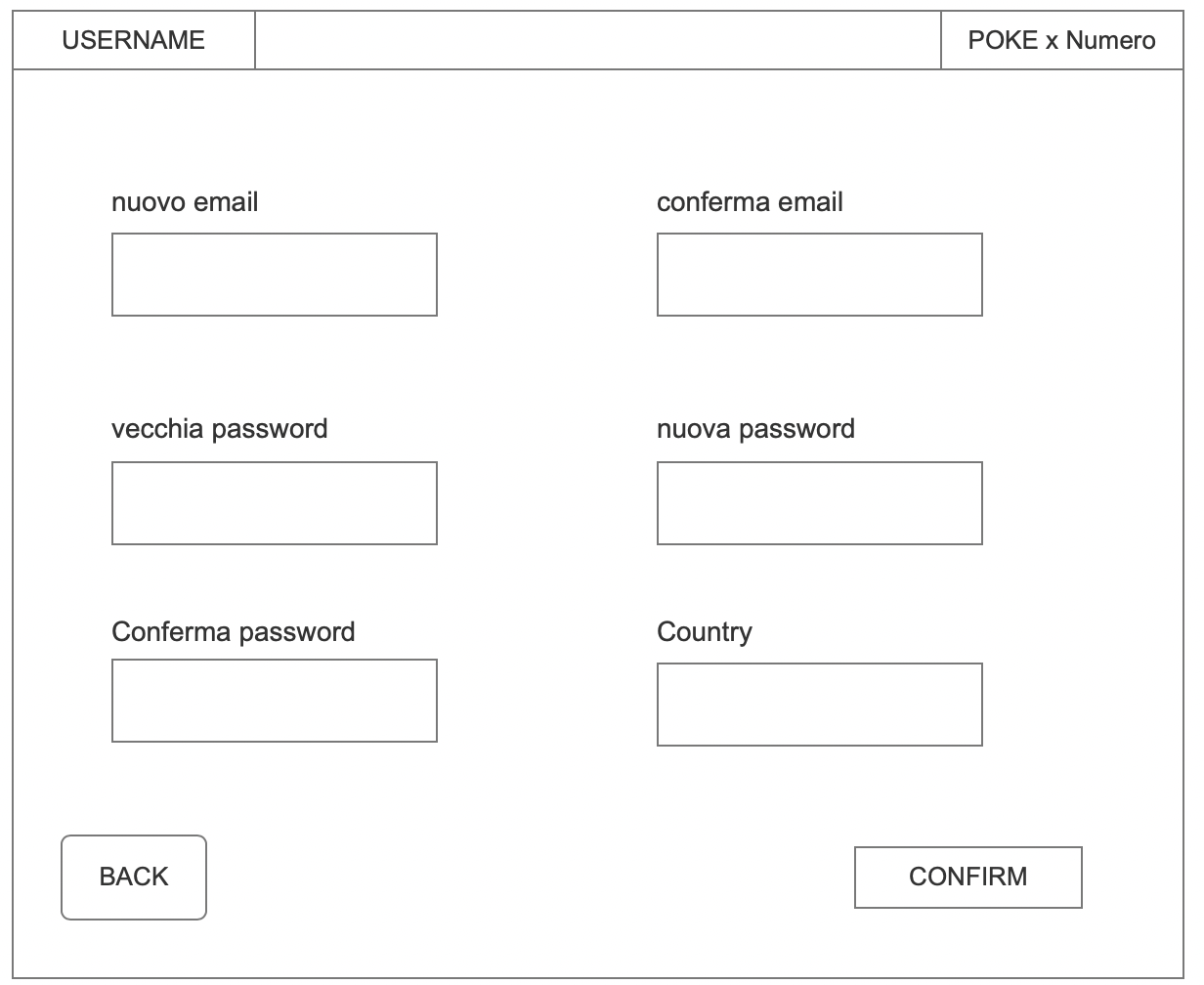
TEAM



POKEBALL NUMBER

POINTS

SETTINGS



POKEBALL NUMBER

Confirm E-mail address

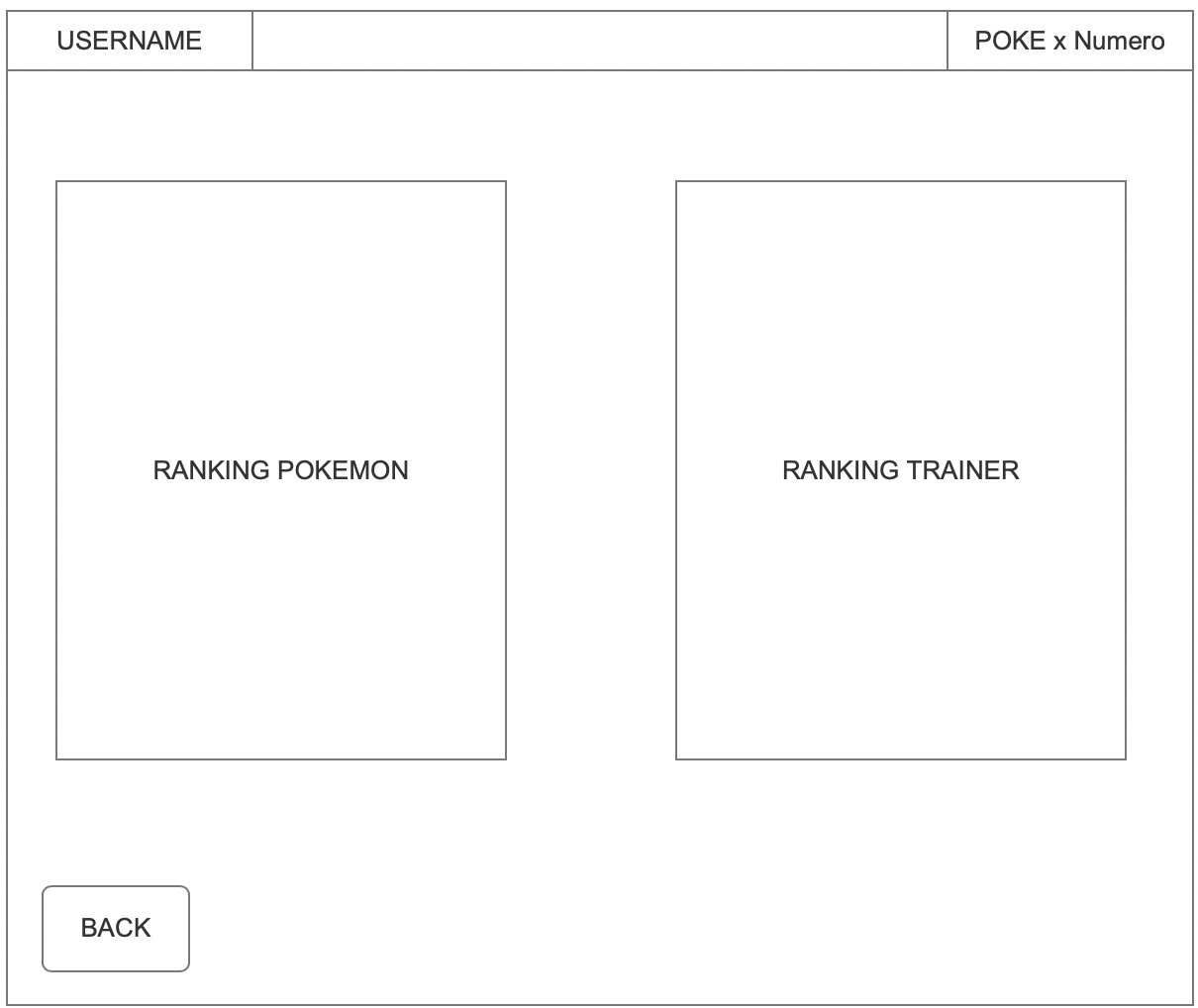
E-mail address

New Password

Old Password

Confirm Password

RANKING



*Note:* the remaining windows (Pokédex, catch’em all, add/remove pokemon) will be done after projecting the database, in order to make the most appropriate solution.

# Requirements

## 3.1 Functional requirements and use cases

* 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 generation

Search by Pokemon characteristics (i.e, height, weight,..)

* + - * + Consult ranking:

Most popular pokemon

Best team

* + - * + Team handling:

Remove Pokemon from the team

View team

Save modified team

View the value of the team

* + - * + Catching:

Try to catch a Pokemon to add to his team

* + - * + Settings:

Change email

Change password

Change country

* + - * + Logout:

Exit from the account

Return to the sign in window

* An admin can

Add pokemon to the Pokédex

Remove pokemon from the Pokédex

* The system should
  + Daily update Pokeball number of each user

## Non-functional requirements

//To define

## 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 attributes from Pokedex using several filters
* Analytics: most popular Pokemons, best ranked teams
* Create an user
* Modify user settings
* Update team (add/remove Pokemons)
* Remove/Add 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 Further Ideas

Social functionalities: a user can follow his/her friends. In this way he can visualize his/her team, receive suggestions on friends/Pokemons according to the people he/she follows, compete for a special friends’ ranking.

If so, a Graph Database will be used to connect each user with his/her friends and with the Pokemon of his/her team, in order to generate fast queries of recommendation, and to store easily different kinds of relationships among users(follows), user-pokemon(owns) and pokemon-pokemon(is the Evolution of)