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

Chapter 1

Introduction

PokèMongo 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.

1.1 Description

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 (e.g. Pokemon name, Type, Points...).

Moreover, as a "real" Pokemon Trainer, the 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

1.2 Code Snippets

Other things: let's show some code snippets!

```
    import requests
    import json


    #exampleW
    new_json = []
    description = ""

    for i in range(500, 894):
        response = requests.get(f"https://pokeapi.co/api/v2/pokemon/{i}/")
        work_string_json = response.json()
        response = requests.get(f"https://pokeapi.co/api/v2/pokemon-species/{i}/")
        work_string_json2 = response.json()

        for desc in work_string_json2['flavor_text_entries']:
            if(desc['language']['name'] == "en"):
                description = desc['flavor_text']
                break

        curr_json = {
            "id": work_string_json['id'],
            "name": work_string_json['name'],
            "weight": work_string_json['weight'],
            "height": work_string_json['height'],
            "capture_rate": work_string_json2['capture_rate'],
            "biology": description,
            "types": [],
            "portrait": work_string_json['sprites']['other']['official-artwork']['front_default'],
            "sprite": work_string_json['sprites']['front_default']
        }

        print(i)
        for i in work_string_json['types']:
            curr_json["types"].append(i['type']['name'])

        new_json.append(curr_json)

    with open('pokemon2.json', 'a', encoding='utf-8') as f:
        json.dump(new_json, f, ensure_ascii=False, indent=4)
```

Listing 1.1: Python example

```java
    package it.unipi.dii.lsmsd.pokeMongo.utils;

import java.time.LocalDate;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import javafx.scene.control.*;

public class FormValidatorPokeMongo {

    /**
     * In this section are present the event handler for the 'setOnKeyReleased' event in the form.
     */
    public static void handleName(TextField nameTF, Label invalidNameLabel){
        if(FormValidatorPokeMongo.isPersonNoun(nameTF.getText()))
            invalidNameLabel.setVisible(false);
        else
            invalidNameLabel.setVisible(true);
    }

    /**
     * Check if the string contains only letters, spaces, dots and apostrophes.
     */
    public static boolean isPersonNoun(String possibleNoun){
        Pattern pattern = Pattern.compile("^[a-zA-Z '.]*$");
        Matcher matcher = pattern.matcher(possibleNoun);
        return matcher.find();
    }

    public static void handleEmail(TextField emailTF, Label invalidEmailLabel){
        if(FormValidatorPokeMongo.isValidEmail(emailTF.getText()))
            invalidEmailLabel.setVisible(false);
        else
            invalidEmailLabel.setVisible(true);
    }

    /**
     * Check if the email follows the format example@domain.tld
     */
    public static boolean isValidEmail(String possibleEmail){
        Pattern pattern = Pattern.compile("^[\\w-\\.]+@([\\w-]+\\.)+[\\w-]{2,4}$");
        Matcher matcher = pattern.matcher(possibleEmail);
```

```
42          return matcher.find();
43      }
44
45      public static void handlePassword(TextField passwordTF,
    Label invalidPasswordLabel){
46          if(FormValidatorPokeMongo.isValidPassword(passwordTF.
    getText()))
47              invalidPasswordLabel.setVisible(false);
48          else
49              invalidPasswordLabel.setVisible(true);
50      }
51
52      /**
53       * Checks if the password contains minimum eight
    characters, at least one letter and one number.
54       */
55      public static boolean isValidPassword(String
    possiblePassword){
56          Pattern pattern = Pattern.compile("^(?=.*[A-Za-z])
    (?=.*\\d)[A-Za-z\\d]{8,}$");
57          Matcher matcher = pattern.matcher(possiblePassword);
58          return matcher.find();
59      }
60
61      public static void handleConfirmField(TextField fieldTF,
    TextField confirmFieldTF, Label invalidConfirmFieldLabel){
62          String password = fieldTF.getText(), confirmPassword
    = confirmFieldTF.getText();
63
64          if(password.equals(confirmPassword))
65              invalidConfirmFieldLabel.setVisible(false);
66          else
67              invalidConfirmFieldLabel.setVisible(true);
68      }
69
70      /**
71       * Checks if the birthday date selected is valid: future
    dates cannot be picked
72       */
73      public static void handleBirthday(DatePicker birthdayDP,
    Label invalidBirthdayLabel){
74          LocalDate localDate = birthdayDP.getValue();
75          LocalDate today = LocalDate.now();
76          System.out.println(today);
77
78          if(localDate.isAfter(today)){
79              invalidBirthdayLabel.setVisible(true);
80          } else {
81              invalidBirthdayLabel.setVisible(false);
```

```
            }
        }
}
```

Listing 1.2: Java example

Chapter 2

Interface Mockup

2.1 Something

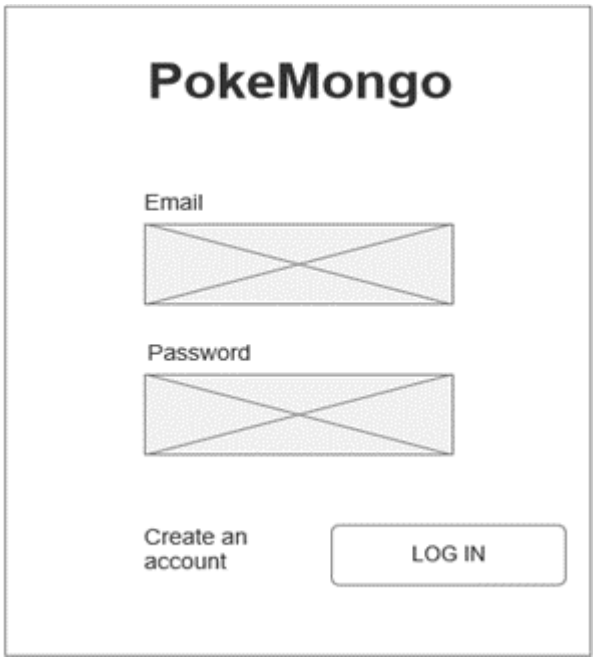

Figure 2.1: Login Mockup

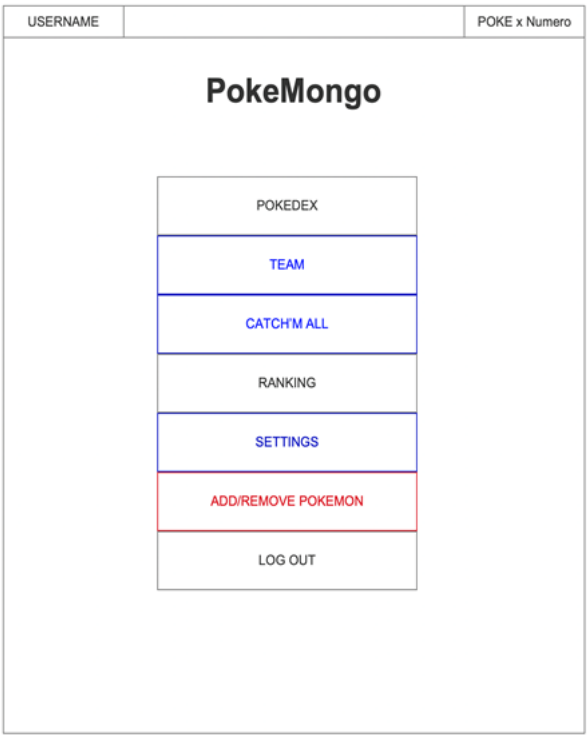

Figure 2.2: Homepage Mockup

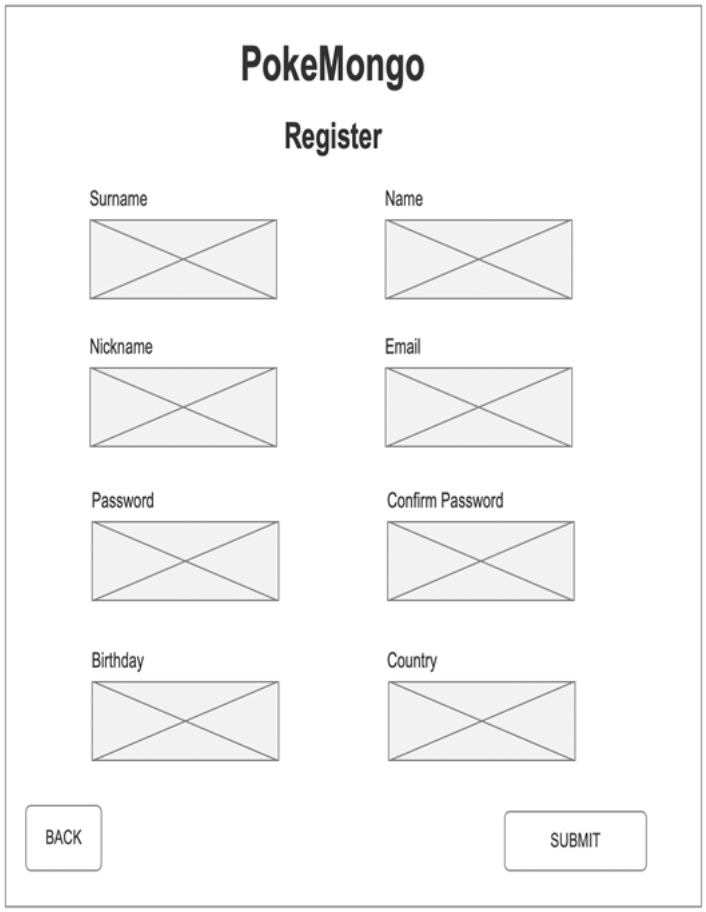

Figure 2.3: Signup Mockup

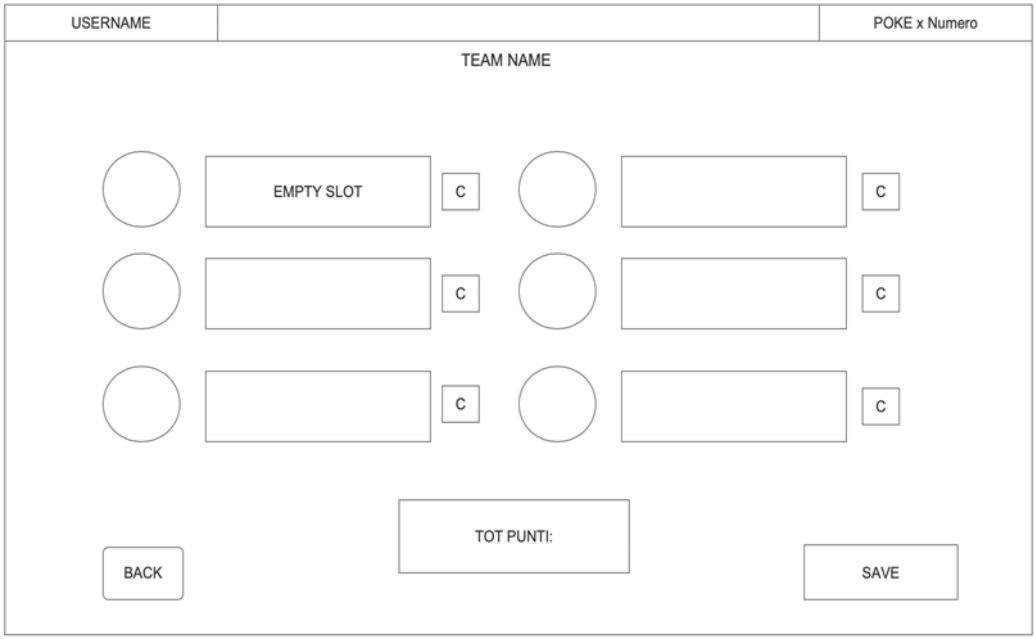

Figure 2.4: Team Mockup

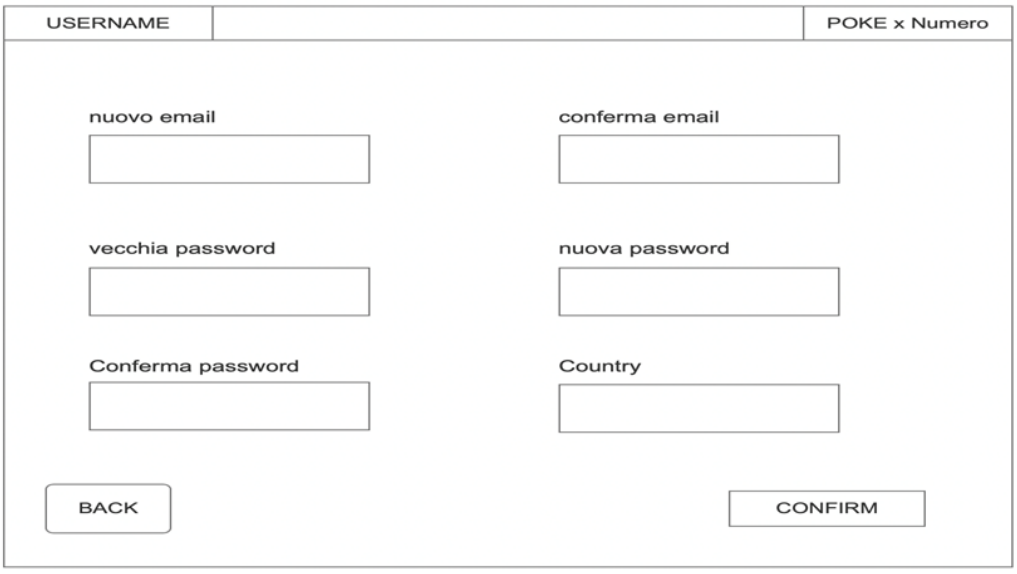

Figure 2.5: Settings Mockup

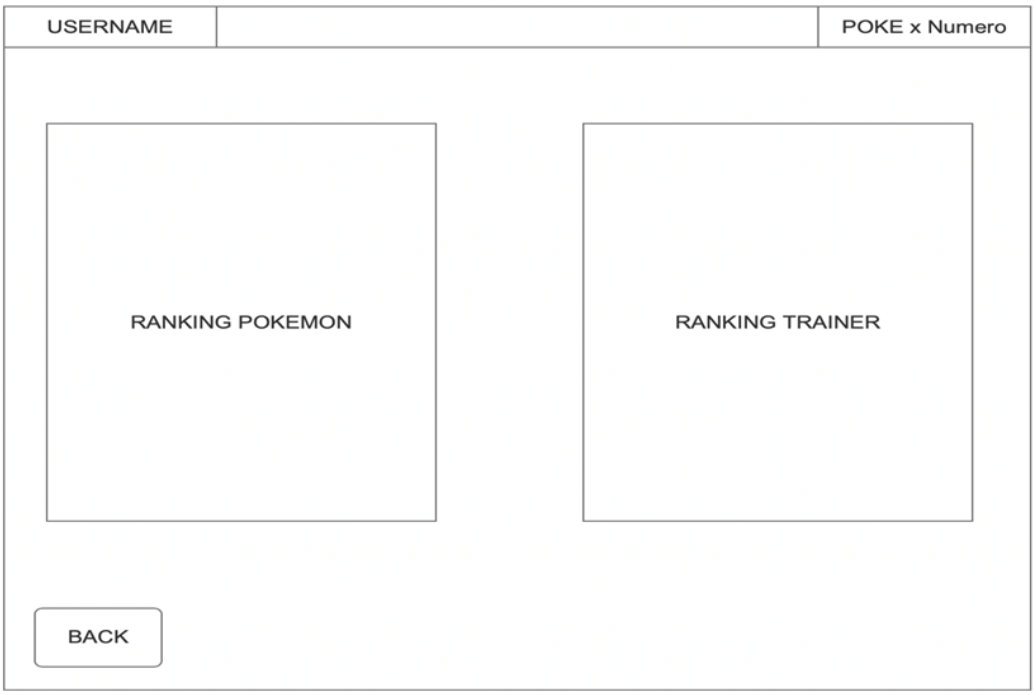

Figure 2.6: Ranking Mockup

Chapter 3

Requirements

3.1 Something

- 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 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 windo

- An *admin* can
  - Add pokemon to the Pokédex
  - Remove pokemon from the Pokédex
- The *system* should
  - Daily update Pokeball number of each user

3.2 Non-functional Requirements

//To define

3.3 UML relation diagram

Figure 3.1: Login Mockup

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.