Distributed Systems Project

Christophe Verdonck and José Oramas

2020-2021

1 Introduction

This document discusses the project for the practical part of the Distributed Systems course. The goal of this project is to understand the principles of micro/web services and successfully apply them in practice. In this document we provide you with a detailed list of features and requirements. The assignment needs to be completed individually.

2 Project Description

In this project you will build an application for a small football league: the 'Liefhebbers Voetbal Liga' (http://l-v-l.be), which is currently active in the province of Antwerp. The application has to be built using a microservice architecture which exposes its functionality through a RESTful API. Your application will keep track of teams, matches and league divisions over multiple seasons and provide some interesting statistics. The app allows teams to upload their match scores after each matchday and keep their club info up-to-date. The database can be managed through an admin interface which can be used to reschedule matches, assign referees to matches, override incorrect match scores, etc. To showcase the application, you will build a simple website that consumes the created web service.

3 Requirements

In this section we list the different features and components that need to be implemented in the assignment.

3.1 Functionality

Your website should be able to show the following information, how you structure this on your page(s) is up to you:

- The league table for a given division
- All fixtures/specific matchweek for a given division. It should be possible to filter on a specific team.
- List for each division the team with the best attack (most goals), best defense (least goals conceded) and most clean sheets.

- A specific fixture. This should include information like: date and time of kickoff, referee assigned. If the match is yet to be played, this should also include head-to-head statistics (i.e., how many times have both teams played each other, how many times has team A won vs team B), the historical scores of the last 3 times both teams have faced each other and the current form of both teams (i.e., results of previous five matches, for example WWDLW). For the upcoming matchweek (i.e, at most 7 days in the future), also include the weather forecast. This can be done by using an external weather API (https://openweathermap.org)
- A team page. This should list all information you have on a team including the results of the previous 3 games and the upcoming fixtures.
- Each team will be able to login which leads them to a portal where they can enter match scores for all of their home games. For the sake of this project it is allowed to enter or edit scores for any home game. You do not have to implement a time check to see if the game was played recently. Each team can also update their club information from within this portal. Login credentials are allowed to be hard coded. No registration is necessary. Alternatively, a user that is linked to a particular team can be added by the administrator (see next point).
- An admin interface that gives CRUD capabilities for (almost) all entities in the database in user-friendly manner. Apart from adding/updating/deleting referees, it should be possible to assign referees to matches for a given match week. Make sure no referee can be double booked. There should be two types of admin, a super administrator can add new/update/delete users for the admin interface. Again you may hard code users to seed the application.

3.2 Database structure

It is up to you to design a proper database structure to store all the application data. To seed your database, some data will be provided on blackboard. Minimally, your database has to store the following:

- Matches: date, kick-off time, home team, away team, goals home team, goals away team, status. When set, the status can be one of the following three: Postponed, Canceled, Forfait. (Think about adding or linking additional information about the season/division/match week/referee)
- Clubs and teams. It is important to note that some teams might have multiple teams (A and B or an additional veteran team). The following information should be stored for clubs: stam number, name, address, zip code, city, website. Additionally each team stores an optional suffix and their outfit colors.
- Referee: first name,last name, address, zip code, city, phone number, e-mail, date of birth
- User: username, password, e-mail. A user can be linked to a club. Only users linked to a club can log in to enter scores, whereas only users that are admins can access the admin interface.

This list is only an indication. Some additional fields and/or tables might be necessary to capture the application data.

3.3 Microservices

The application must be based on a microservice architecture, meaning that the application should be broken down into different smaller parts that interact and communicate with each other. This architecture needs to be provided in your manual (see later).

4 Tools

You will use a number of tools, technologies, and languages to successfully solve this project. While some of these are mandatory, others can be used based on your own preferences.

Below you will find a list of tools and some information.

- Docker (https://www.docker.com/): the most popular open source container technology available today. Docker allows you to configure a microservice per container.
- Python 3 and Flask: The application has to be written in Python 3 using the Flask microframework (https://flask.palletsprojects.com/en/1.1.x/), which is very straightforward to setup.
- Postgress and Flask-SQLAlchemy (https://flask-sqlalchemy.palletsprojects.com/en/2.x/): the Postgress database is used to store all the application's data. (Flask-)SQLAlchemy will be used to access and edit that data.
- Weather service: To get the weather forecast for a game, you can use any online weather service. One possibility is Open Weather Map (https://openweathermap.org/, for which the daily forecast for the next 7 days is available for free (with a max of 1000 API calls per day). To get the coordinates of a particular address one possibility is to use the Google Geocoding API (https://developers.google.com/maps/documentation/geocoding/overview).
- Web technologies and languages: for the construction of your website you are free to choose which technologies, libraries, or languages you want to use. The typical ones you might want to use are HTML (for the structure of the website), CSS (for the layout of the website), and Javascript (for adding dynamic functionalities to the website). As mentioned, it can be interesting to look at the Bootstrap library (https://getbootstrap.com/) to accelerate your website design.

For all of the listed tools, technologies, and libraries, there is an extensive amount of documentation online waiting for you to be consulted. How to combine all of these tools in a nicely working microservice application is described in detail in the TestDriven.Io tutorial (provided on Blackboard). The Part 1 tutorial provides you with an elaborate tutorial on how to start. It also explains other useful tools such as Jinja Templates, Flask Blueprints, etc. to ease the implementation of your application.

5 Minimal requirements

There are some minimum requirements that your solution should conform to, in order to pass for this part of the course. If these requirements are not fulfilled, it is not possible to pass:

- A manual needs to be present in your submission or it is not possible to evaluate your solution. This manual needs to describe the architecture of your application (by text and by diagram), how we can run and test your solution and an extensive explanation of the design choices you made and which tools were used. The installation of tools not mentioned in the tutorial should be explained in your manual. Also make sure to give an overview of your API, which endpoints are available and what data they return (in json).
- A single "install.sh" or "install.py" file should be present that installs all the different components. Libraries can be installed through a "requirements.txt" file. Similar to the manual, if these two files are not present (or do not work appropriately), we will not be able to evaluate your solution.
- It needs to be able to run your solution (e.g., start up your Docker containers, open the web interface, ...). Therefore, make sure that your solution is complete upon submission and no configuration files and/or libraries are missing. All programming should be done using Python 3.
- Make sure to only use web services and web communication (e.g., HTTP calls) within this assignment and no other communication methods. The communication with the database may be the exception to this as it uses built-in Flask functionality.

Remark: if your solution complies with these minimum requirements, this does not mean that you will certainly pass this assignment.

6 Deadline and submission

The deadline for this project is **Sunday 10th of January 2021 at 23:59**. Your solution has to be submitted through blackboard. Good luck!