Developer’s Documentation

*Developer’s Documentation of the program Mock Football Bets for a class*

*Semestral Project (NPRG045) at*[*MFF UK*](http://www.mff.cuni.cz/to.en)*.*

Author: Tomáš Pop

Date of issue: 24/09/2021

# Introduction

In the developer’s documentation you can read about algoritms, decisions made and implementation details used in Mock Football Bets.

# Used tools

- For the development, the author used Visual Studio Code

- The main reason of choice for particular parts was primarily to extend knowledge about particular technologies

## Backend

- Backend of the application is written in Python – version 3.8.5+ (used 3.9.7)

- In general, it takes care of the connection to the database, sending important data to frontend (e. g. credentials, bets information)

- Machine learning part – training and testing 7 supervised classification models – used in auto mode

## Frontend

- Javascript – frontend is written in Javascript extensively using library called React

## Database

- MySQL – relational database system, run via XAMPP

# Libraries involved, what for they are used in the project

## Python

* From standard library
  + smtplib, ssl – email delivery handling
  + shutil, os, sys – files/directory manipulation
  + datetime – date and time parsing, manipulation
  + json – extensive work done using json – e. g. update of upcoming fixtures
* Flask
  + Web framework which takes care of communication with frontend
  + Runs on localhost:5000
  + Routing made by @app.route(“/pageyoudesire”), methods = […]
* mysql\_connector
  + Communication with database, SQL querying
* bcrypt
  + Password hashing – in order not to store exact passwords in the database
* soccerapi
  + Upcoming matches retrieval API
  + Note: Make sure you do not make way too many API calls (it is a free service and very frequent API calls can temporarily block access to API)
  + Reasonable amount of API calls per minute with useful data while still remaining free
* sklearn
  + Machine learning – models training, testing, accuracy score check
* pandas
  + Historic data are retrieved in csv – reading csv files
* bs4 (BeatifulSoup)
  + To download csv files with historic data
  + To receive information about recently finished matches – to target concrete html elements - those with score and names of the teams
* Selenium
  + Prepares the document for BeautifulSoup (it was necessary to load the page as a whole with JS, for the recently played matches BeautifiulSoup html parser did not work properly with static content only)
* Apscheduler
  + To schedule jobs which are required once upon a time
    - Upcoming matches (with its data – especially odds refresh) – once upon 20 seconds – make an API call
    - Refresh historic data and models daily – to remain relevant (football-data.co.uk where the csv files are downloaded from update recently played matches within 2-3 days) while time passes by
    - Auto mode predictions daily
    - Update balance after the results of the previous day are known -daily
* Joblib
  + To save and load machine learning models – to persist a model for use without retraining

## React

* React Router – to enable simple routing between particular pages for the client

# Data structures

## Classes

* **Python**
  + **Fixtures** – manipulation of data concerning upcoming matches
    - Consists of gathering the data from Unibet API (provided by the library soccerapi), filtering
    - Involves member variable **fixtures\_information** – a json file (taken from fixtures.json in flask\_api directory) which is transformed during the filtering contained in the filter\_useful\_data class method (after the upcoming matches are processed, json file (fixtures.json) is then updated for a next iteration)
  + **Database** – to provide connection via mysql\_connector with MySQL database
    - Consists of member variables **connection** (estabilish and commit) and **cursor** (query execution)
  + **DataPreparation**
    - Class concerning machine learning – prepares data for ML models and prepares models to be used in ML based modes
    - Consists of member variable **data** – information about match – from which the model should evaluate prediction of match result
    - Member variable **target** – expected result

## Components

* **React JS**
  + **Forms**
    - All have in common that they need to handle submission (handleSubmit) where body of the request changes according to the purpose of the form
    - All fetch their api url to obtain data from backend
    - **LoginForm** needs to check for conformity of credentials and redirects to Bets page if the credentials are correct – username and balance from now on can be accessed via localstorage
    - **BetsForm/AutoForm** – several input values and submit
      * After submission input values are reset
      * AutoForm, in addition, needs to take care of checkboxes – made by an array of bools accordingly to the
  + **Pages**
    - **Register** and **Login** make use of the associate forms above
    - **Bets** – manages correct display of particular columns which need to change accordingly to a concrete mode the user is in
    - **Profile** – involves credentials, history tracking, auto mode termination

## Particular structures within the implementation

For the particular parts of the project done in Python, there is an extensive usage of dictionaries which are used as a communication medium with frontend (JSON as a body of a POST request used to send data to backend).

Throughout the project while working with database, it is rather necessary to work with tuples (to avoid SQL injection it is not suggested to leave it hardcoded in the SQL queries itself).

Most of the logic made on the frontend is solved by React Hooks – useState – everytime when it was demanded to remember a state made within a function – e.g. API fetch, handling changes in the input values etc.

# Machine learning models used

* Across the project there have been used 7 machine learning models which can be well suited for supervised classification problems
* For the assist mode, multilevel perceptron has been chosen due to the fact that during the accuracy tests showed the best results
* The other 6 models – Logistic Regression, Support Vector Machine, Random Forest, Decision Tree, Gradient Boosting and Stochastic Gradient Descent have their parameters slightly tuned using scikit-learn’s [GridsearchCV](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html) - there is definitely still a space for an improvement of these models

## Strategy for auto mode

* Since the machine is allowed to place bets only once a day, the strategy consists of splitting the current balance into a number of candidates the machine wants to bet on (maximum 5) and splits it into (#of\_candidates – 1)/#of\_candidates, therefore the machine after it accomplishes the bets, it is left with one/#of\_candidates money to spend on upcoming matches.
* It is possible that the machine does not choose any of the matches provided
  + It is not allowed to bet on a same match twice
  + It needs to satisfy constraints set by the user (only a particular league, maximum risk to be taken, sufficient number of models which agreed on a particular result etc.)
  + It does not have enough balance to place any bets (anymore) – when it reaches less than a tenth of the initial balance, it is not allowed to place any bets

# Decisions made

## API for bets retrieval

* Soccerapi was chosen primarily for the reason that there are not many free services offering convenient data within a reasonable amount of API calls upon a time

## Results

* Initial intention was to utilize csv files with historic data, nevertheless, football-data.co.uk updates results approximately 2 days after the match is played which was found to be inconvenient
* Results are therefore crawled from the website goal.com