POLITECNICO
MILANO 1863

CodeKataBattle project by Russo Mario
and Picone Paolo

Implementation and Test Deliverable

Deliverable:	ITD
Title:	Implementation and Test Deliverable
Authors:	Picone Paolo and Russo Mario
Version:	1.0
Date:	22-January-2024
Download page:	https://github.com/piconepaolo/PiconeRusso
Copyright:	Copyright © 2023, Picone Paolo and Russo Mario – All rights reserved

Contents

Table of Contents **3**

List of Figures **4**

List of Tables **4**

1 Introduction **6**
1.1 Scope 6

2 Requirements and funtions implemented **7**

3 Adopted Development Frameworks and Technologies **8**
3.1 Programming Languages and Frameworks 8
3.2 Main libraries 8

4 Structure of the Source Code **9**

5 Performed Tests **10**

6 Installation Instructions **11**

7 Effort Spent **12**

References **13**

List of Figures

List of Tables

The team that implemented and tested the prototype of the CKB platform is composed by two members:

- Russo Mario <mario1.russo@mail.polimi.it>
- Picone Paolo <paolo.picone@mail.polimi.it>

The source code of the project is available at the following GitHub repository
In order to run the project, it is necessary to install the following software:

- Docker Desktop since it includes both Docker Engine and Docker Compose.
- MongoDB Compass [Optional]: a GUI for MongoDB in order to visualize the database. It is not needed to run the project, but it is useful to visualize the database.

1 Introduction

The document provides some useful informations about the implementation and the testing performed during the development of the prototype of the CKB platform.

1.1 Scope

The document shows the result of the implementation and testing of the prototype of the CKB platform following the RASD and the DD documents. It motivates the choices of programming languages, frameworks and technologies used to develop such prototype. It describes the structure of the source code as well as the setup of various environments in order to run the project. Regarding the testing, it shows the results of the tests performed on the prototype.

2 Requirements and funtions implemented

Since the developed code is referred to the prototype of the platform, the functions and the requirements implemented are the ones that are strictly necessary to show the feasibility of the project. The document does not cover the entire set of requirements and functions described in the RASD and DD documents.

The following list shows the functions actually implemented:

- **User authentication**: the system allows the user to register and login. It is compliant with *OAuth2* protocol.
- **Creation of a Tournament/Battle/Team**: since the adopted architcture is MVC, these entities are implemented as models with their respective controllers and views.
- **Automatic Evaluation**: this function has been implemented as an endpoint that the github action set up by the user can call. The calculation of the score is mocked returing a random number between 0 and 100.
- **Authorization Manager**: the handling of the permissions is granular since every path operation that require some kind of permissions has a middleware that checks the user's permissions.

The following list shows the functions that are not implemented:

- **User Interfaces**: the user interfaces are not implemented. The user interface does not provide any additional functionality to the system since the system can be interacted with using the API.
- **Manual Evaluation**
- **Educator Invitation**
- **Notification of the user**

3 Adopted Development Frameworks and Technologies

In the current section are described the programming languages, frameworks and technologies used to develop the prototype of the CKB platform.

3.1 Programming Languages and Frameworks

The programming language used to develop the prototype of the CKB platform is Python and the api are built with FastAPI. The choice of Python is due to the familiarity of the team with the language. The Frameworks FastAPI was chosen since it brings several advanteges to the development of the api, such as the automatic generation of the OpenAPI documentation, its performace and its ease of use and learning. Some disadvantages of FastAPI are the lack of some features that are present in other frameworks, such as the lack of a built-in ORM and the lack of a built-in authentication system. It also offers a lot of flexibility, which can be both an advantage and a disadvantage depending on the context it's used in.

The database used is MongoDB, a NoSQL database. The motivation behind the choice of MongoDB is because, like mentioned in the DD document, a document-based database is a good fit for the project and MongoDB is one the possible choices for this type of DB. MongoDB brings some advanteges such as a robust library (**PyMongo**) in Python to interact with, since the Python data stractures can be easily converted in document to insert and retrieve from the database. One of the main disadvantages of MongoDB is the difficulty to perform complex queries and the lack of transactions, which can be a problem in some contexts. Another technology used is Docker, which is used to containerize the application and the database. The choice of Docker was to make the development among the team members more consistent and to make the deployment of the application easier. Docker also brings some advantages such as the possibility to run the application in different environments without the need to install the dependencies and the possibility to run the application in a cloud environment. The main disadvantage of Docker is the overhead that it brings to the application, since the application is running in a container, which has to be configured and managed properly.

3.2 Main libraries

One of the main libraries used in this project is Pydantic. Pydantic is used to validate the data that is sent to the api and to convert the data to the correct type. It offers a lot of methods to validate data received by the api. The choice of Pydantic was due to the fact the has a good documentation and it's very powerful when used by FastAPI. An example could be the generation of the openapi.json file, which is done automatically by FastAPI using Pydantic.
Another important mention is PyTest to perform tests on the api. PyTest is a very powerful library to perform tests in Python and it's very easy to use and to learn. It offers a lot of features to perform tests such as fixture and mocks. It is also very easy to use and run the tests.

4 Structure of the Source Code

The folder of the source code includes two folders and installation scripts. The folders are:

- **app**: contains the source code of the application.
  - **api**: contains the source code of the endpoints and the dependencies used by the apis.
  - **core**: contains the application configuration file as well as security settings.
  - **crud**: contains the source code of the crud operations for the main entities of the application.
  - **EvaluationManager**: contains the source code of the evaluation manager.
  - **schemas**: contains the source code of the Pydantic schemas used both by apis and crud operations.
  - **github**: it's a module that provides methods to interact with the GitHub API.
  - **utils**: it's a general purpose module that provides some useful methods to the application like singleton to implement the singleton pattern and mongo utilities to embed the Pydantic models in nested MongoDB documents.
- **tests**: contains the source code of the tests. The concern of the tests is easily understandable by the name of the files which contains the entities that are tested.

5 Performed Tests

The perfomed tests does not completely respected the testing strategy included in the DD. The included tests are system tests that calls the API endpoints and checks the response. The tests, as described above, are implemented using the *PyTest* framework. The main test cases implemented are the following:

- **Creation and login of a user**: the test checks if a user by sending valid credentials is returned a JWT token.
- **Creation of a Tournament**: the test checks if a user has the right permission to create a tournament (is an educator) and if the tournament is created with the provided details.
- **Creation of a Battle**: the test checks if a user has the right permission to create a battle (is an educator) and if the battle is created with the provided details.
- **Creation of a Team**: the test checks if a user has the right permission to create a team (is a student) and if the team is created with the provided details.
- **Add a student to a team**: the test checks if a user has the right permission to add a student to a team: is a member of an already created team and all the limits about the number of students in a team are respected.
- **Creation of a repository**: the test checks the correct creation of a GitHub repository for a battle.
- **Invite collaborato to a repository**: the test checks if the call to the GitHub API to invite a collaborator to a repository is successful.
- **Download repository**: the tests checks if the call to the GitHub API to download a repository is successful.

6 Installation Instructions

Since the applicaton is dockerized the installation process is very simple.
In order to not expose sensible data the app has to read an environment file in which there are the sensible and configuration data. The file is called *app.env* and it should be placed in the root of the project. The file can be downloaded from the following Google Drive link
Once downloaded the *app.env* file, the installation process is the following:

- Clone the repository from the following GitHub Repository
- Place the *app.env* file in the root of the source code *Code*
- Build the docker images by running the command *docker-compose build*
- Run the docker images by running the command *docker-compose up*
- The application is now running and it is possible to access it at the following address: *http://localhost:8080*

7 Effort Spent

The effort spent to implement the prototype of the CKB platform is estimated to be 30 hours per person including the redaction of the ITD document. The time spent include also the time spent to learn the technologies used in the project, such as FastAPI, Pydantic, MongoDB and Docker.

References

- Document written using LaTeX and Visual Studio Code
- FastAPI documentation: FastAPI
- PyMongo documentation: PyMongo
- Pydantic documentation: Pydantic
- PyTest documentation: PyTest
- Docker documentation: Docker
- MongoDB documentation: MongoDB
- GitHub API documentation: GitHub API

