Usage Manual of Book Explorer

Contents

1 Generalities

In this section are found different procedures and basic concepts that are shared throughout the different profiles and stages.

1.1 Initial Installation

To start developing a feature, test a functionality or module, or deploy the service, it is required to clone the repository, for this purpose it is used:

```
git clone https://github.com/alexisuaguilaru/BookExplorer.git
```

Each of the modules that are written in Python, have a `requirements.txt` file that contains the libraries required for the operation of the module or, alternatively, to install to develop or implement a change in a feature of the respective module.

1.2 Docker Compose

The microservice is divided into three docker-services, where each one contains the different functionalities and code according to what it performs within the microservice. The docker-compose (`Compose.Backend,yml`) contain the following microservices:

- MongoDB database [BooksDatabase]
- Data extraction from the API [DataExtraction]
- API of the main microservice [SystemRecommender]

1.3 Configuration File *.env*

The `.evn_example` file serves as an example of the environment variables that must be changed to adjust the behavior of the three `docker-compose.yml` files. The configurable environment variables are:

- *UID*: ID of the non root user that executes the different services in the `docker-compose`
- *GID*: ID of the group the non root user belongs to
- *MONGO_INITDB_ROOT_USERNAME*: Username of the database administrator in MongoDB.
- *MONGO_INITDB_ROOT_PASSWORD*: Password for the MongoDB administrator.

- *MONGO_WRITE_USERNAME*: Username of non root user who can only write data to the database.
- *MONGO_WRITE_PASSWORD*: Password for user with write role.
- *MONGO_READ_USERNAME*: Username of non root user who only reads data in the database.
- *MONGO_READ_PASSWORD*: Password for the user with read role.
- *DB_NAME*: Name of the database to be created in MongoDB.
- *AMOUNT_BOOKS*: Number of books that are extracted from the API. It does not represent the final number found in the associated collection in MongoDB.
- *USER_AGENT*: Identification credentials for the OpenLibrary API (project name, email). It is not a token or key for the API.
- *FLASK_SECRET_KEY*: Secret key used by Flask for encryption and session management.
- *DEBUG_MODE*: Mode in which Flask applications run, whether in debug mode or production mode.

1.4 Execution File *Makefile*

The `Makefile` file contains commands that execute the different services contained in the `Compose.Backend.yml` file depending on whether it is the first time it is executed or not. It contains the following variables to configure:

- *LENGTH*: Controls the length of the key, preferably greater than 12 characters.
- *ENV*: Environment under which the service is being executed, depending on whether it is for development, testing or deployment.

Since it deals with both passwords and secret keys, the following command is available to generate them using alphanumeric characters:

```
make secret_key LENGTH=integer
```

For showing the different commands contained in `Makefile`, use the following command:

```
make help
```

2 Response to a Request

Regardless of the environment from which the request is being generated, whether locally (http://localhost:8013) or in a network of Docker containers (http://RecommenderSystem:8013), two types of requests can be generated:

- Request to obtain book recommendations or suggestions based on another book (its ISBN code):

  ```
  GET API_URI/recommendations?isbn=ISBN_CODE
  ```

  Where *ISBN_CODE* can be empty (`''`) to obtain random recommendations or be a valid ISBN code or one that exists in the database.

- Request to obtain the information of a book based on its ISBN code:

  ```
  GET API_URI/information_book?isbn=ISBN_CODE
  ```

  Where *ISBN_CODE* must be a valid ISBN code or exist in the database.

3 Base Configuration

For the implementation of a new feature (development), unit or integration tests (testing) or deployment of the microservice (deploy), it is necessary to first configure the environment variables in the `.evn_stage` file, for this purpose it is suggested to use the following command:

```
cp .env_example .env_production
```

In which, it is necessary to modify and change each one of the values of the environment variables following the suggestions and references in the sections 1.3 and 1.4.

Before executing the `Compose.Backend.yml` file, it is necessary to create an additional network that allows the communication between the API (running in Flask) with the different Docker containers and services; for this, both containers must belong to the same network, which is created using the following command:

```
docker network create books_explorer_internal
```

Once the network is created, it can be executed the Docker-Compose by means of the commands in `Makefile` and start response requests to the API. This is done with the following command:

```
make ENV=production deploy
```

In case any component has been updated, changed or stopped working, the following is used:

```
make ENV=production deploy_restart
```

4 Development

Following the guidelines established for the development and implementation of new features in the development team, that is, the nomenclature for naming branches and commits in the Git protocol, the following is done for the proper development and workflow. In addition to the above, it is necessary to do what is mentioned in 3.

It is suggested the creation of a virtual environment in Python using venv, in each of the modules developed in Python have their own set of libraries to install; therefore, depending on the feature to implement, the following commands are executed:

```
python3 -m venv path_virutal_environment
source path_virutal_environment\bin\activate
cd Module
pip install -r requirements_Module.txt
pip install pytest==8.3.5
```

To test that these new features work properly, it is suggested to raise the API locally, to do this first add the network `default` to the `RecommenderSystem` service found in `Compose.Backend.yml` to generate requests to the API locally.

For testing the integration of the new changes with the code base, it is suggested that running the tests using the following command in the root of the project:

```
pytest
```

5 Testing

First it is suggested to raise the microservice following what is mentioned in 3, in addition to add the network `default` to the `RecommenderSystem` service found in `Compose.Backend.yml` to generate requests to the API locally.

Once the API has been up, the tests of both the API itself and the different tests of the service that consumes the API that has been up have to be executed. For the first ones the following commands are executed in the root of the project:

```
pip install pytest==8.3.5
pytest
```

Additionally, it is suggested to add more tests in the folder `Test/` following the guidelines of Pytest, this to cover particular cases and applications. The tests available are only to validate the extraction, transformation and loading of the data from the books to the database.

6 Deployment

In addition to doing what is mentioned in 3 to build and up the API properly, the microservice/API network, `books_explorer_internal` network has to be added to the different Docker containers or services that will generate requests to the API created by means of the request structures mentioned in 2.