Forensic Investigation System (FIS)

Purpose

This repository contains a mock implementation of the Dutch Forensics System to fulfil requirements of the (Master of Science: Computer Science) Secure Software Development module offered by University of Essex.

Source Code Layout

All the system’s modules reside in the “venv” and “src” folders, with the “venv” folder containing all imported modules and the “src” folder those developed specifically for the project. The “src” directory is composed of four folders:

1. “web” - defines the web layer with all the HTML templates and Flask webserver endpoints.
2. “core” - contains business logic, service classes, and authorization facilities.
3. “data” - defines the relational entities through an Object-Relational Mapper (ORM) definition implemented using the SqlAlchemy package. Additionally, repository classes to implement CRUD operations on relevant relations using the sqlalchemy ORM module.
4. “auth” – houses the microservice module supporting authentication.

Execution Instructions

One could install the required dependencies with “pip install -r */path/to/requirements.txt*”. Furthermore, it is suggested to do so within a standard Python virtual environment, to keep project dependencies isolated. However, one may just run the virtual environment supplied. Execution instructions follow:

1. Open two separate instances of Command Prompt
2. Set the FIS folder as your current directory in both instances e.g. in Windows: C:\Users\Michael Botha> cd documents\github\FIS
3. Should you use the supplied virtual environment containing all the required dependencies, activate it in both instances: venv\Scripts\activate
4. Set the relevant flask variable in each instance:
   1. In Instance 1 use the command: set FLASK\_APP=src/auth/Authenticate
   2. In Instance 2 use the command: set FLASK\_APP=src/web/application
5. Run the Flask server in each command prompt:
   1. In Instance 1 use the command: flask run -h localhost -p 5005
   2. In Instance 2 use the command: flask run

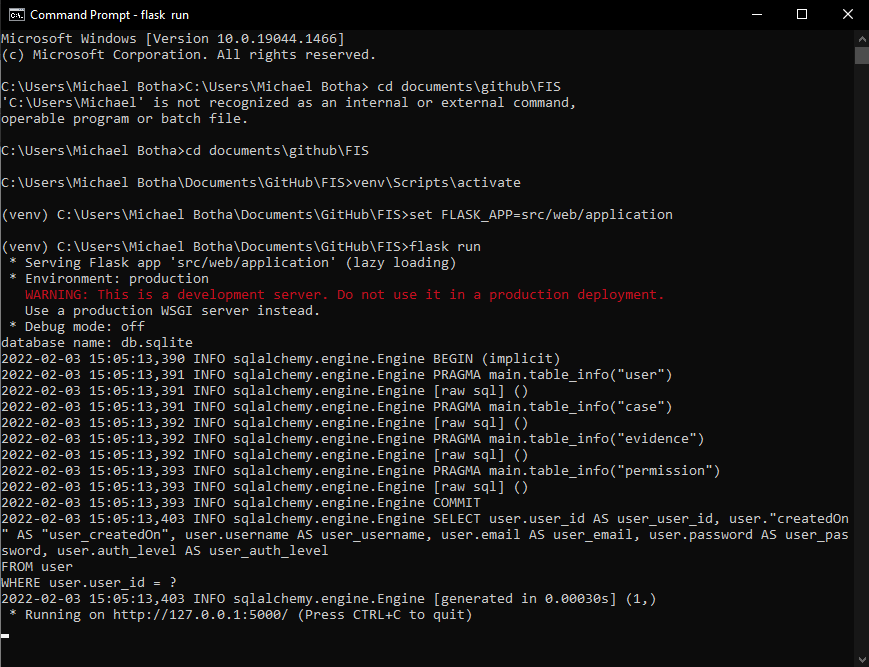
In figure 1 and 2 one can see the process for the two separate instances.

Once both Flask servers are running, one can open a browser and enter the URL of 127.0.0.1:5000 which is the IP address of the local loopback port, and the TCP port of the Flask web application. This will fetch the homepage/login page of the web application as seen in figure 3. Once there the application can be interacted with.

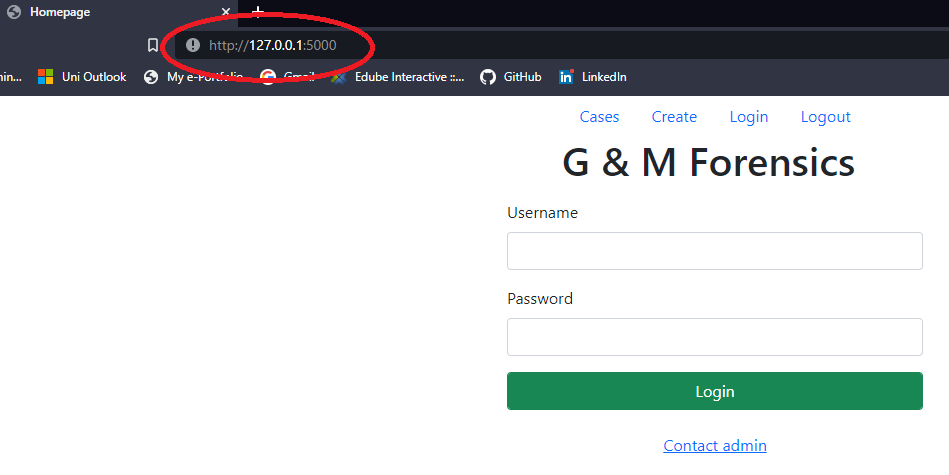
Text

Description automatically generated

**Figure 1**



**Figure 2**



**Figure 3**

Defaults

The system has been designed to have user accounts in the form of username, password, and authorisation level entered into the database via a system administrator. Furthermore, this will only be performed after a change-control process has been completed. Therefore, to be able to interact with the system for test purposes six users with varying authorisation levels have been created as per Table 1:

**Table 1**

|  |  |  |
| --- | --- | --- |
| **Username** | **Password** | **Auth. Level** |
|  |  |  |
| user1 | 1234 | 1 |
| user2 | 1234 | 1 |
| user3 | 1234 | 2 |
| user4 | 1234 | 2 |
| user5 | 1234 | 3 |
| user6 | 1234 | 3 |

In addition to the above, there is a log file which by default has been set to be created and updated in the root directory of “FIS”. This contains various system information, explained in further detail in the “Evidence” directory in the root.

Differences Between Implementation and Design

There were various aspects from the original design which were changed or not used, in line with an agile philosophy, and the limited resources with regards to time, manpower, as well as expertise. The list that follows represents the differences or the incomplete implementations realized versus what we designed in the beginning.

1. RBAC authorization model hasn't been implemented. Instead, in PermissionRepository has been implemented a simpler mechanism.
2. JsonWebToken are not used. Instead, the system verifies the credential and store current username in the session.
3. Search hasn't been implemented.
4. Pagination is incomplete feature.

As we declared in our design document, the base system composed by the data and core modules, is reusable and object-oriented. In fact future development could for example implement REST API based on these modules. Or complete the implementation of a RBAC authorization model, or other improvements, this is mainly achieved because of good programming techniques have been applied like SOLID principles.

Database used in this implementation is sqlite for the simplicity of setup and usage. Obviously, it cannot keep up with the demand of a live system, this can be resolved using a more appropriate database engine as Postgres or MySQL, and the use of SQLAlchemy ORM perfectly support to migrate to another database engine without any code modification.

## Test

In the folder `test` there are tests of the main classes using pytest framework.