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*” (Microsoft, 2022). Furthermore, it is suggested to do so within a standard Python virtual environment, to keep project dependencies isolated (Sarmiento, 2019). However, one may just store and 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

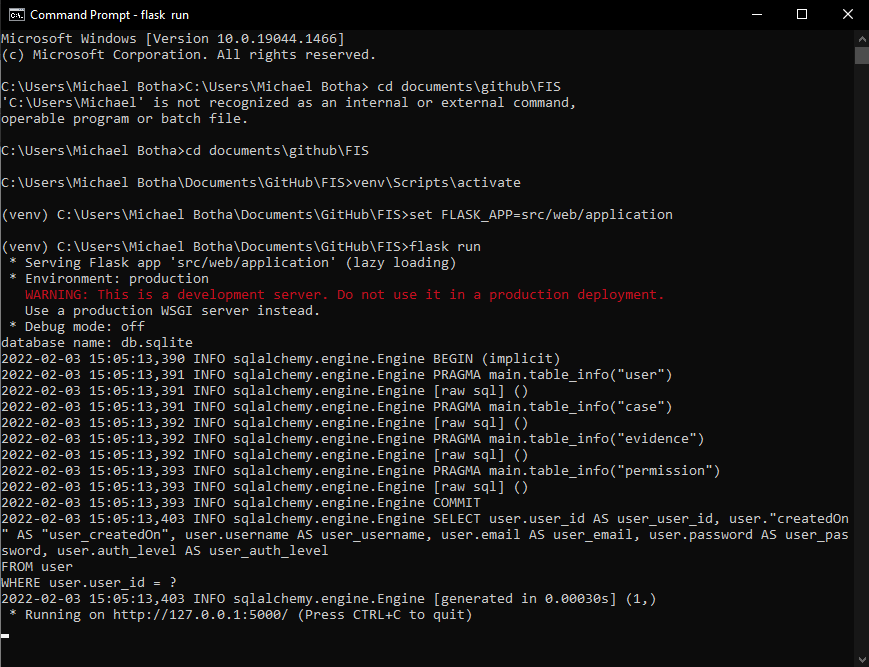
In figure 1 and 2 one can see the process for the two separate instances.

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.

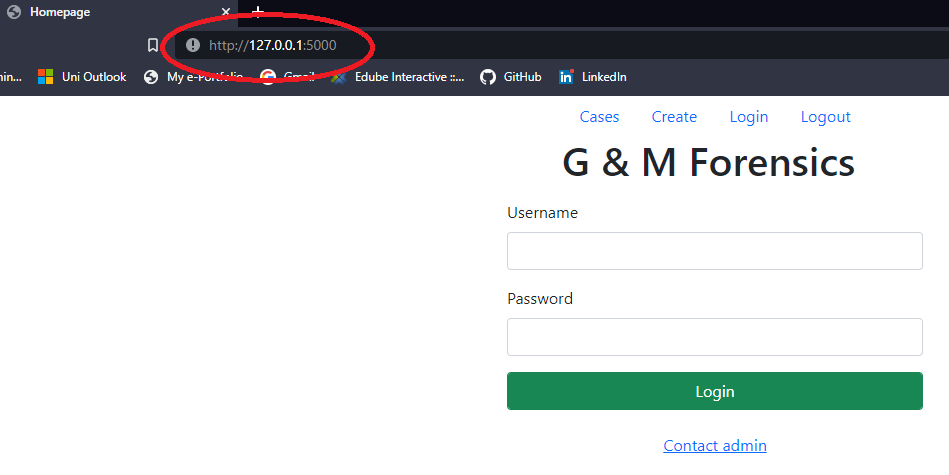
Text

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 notes the major differences:

1. HTTPs was not used as the layer 7 communication protocol between the browser and server as well as the main web application and the microservice. This due to the certificate required from a trusted Certificate Authority (Grinberg, 2017). However, HTTPs can be run between the main web application and client using self-signed certificates, where a connection needs to be explicitly allowed by the user in their browser (Grinberg, 2017). Unfortunately, we never resolved how to prevent the microservice from creating an error when receiving a self-signed certificate. A 3DES algorithm was employed for encryption of the password when connecting to the microservice from the main web application.
2. The encryption of logs was not performed so as to simplify testing and viewing code response. However, this could easily be implemented for production purposes.
3. The RBAC authorization model was not implemented. Instead, the “PermissionRepository” was created to implement a simpler mechanism.
4. A JsonWebToken was not used to perform authentication. Instead, the system verifies a users’ credentials, and thereafter the relevant username and authorisation level in a Flask session. The session passes a cryptographically signed session cookie to the user’s browser, where it is passed back to the server each time a request is made for authentication and authorisation purposes (AskPython, N.D).
5. Search functionality was not implemented, with regards to finding cases.
6. Pagination is an incomplete feature.

References

AskPython. (N.D) Flask Sessions – Setting User Sessions in Flask. Available from: <https://www.askpython.com/python-modules/flask/flask-sessions> [Accessed 22 January 2022].

Grinberg, M. (2017) Running Your Flask Application Over HTTPs. Available from: <https://blog.miguelgrinberg.com/post/running-your-flask-application-over-https> [Accessed 7 February 2022].

Microsoft. (2022) Manage Required Python Packages with requirements.txt. Available from: <https://docs.microsoft.com/en-us/visualstudio/python/managing-required-packages-with-requirements-txt?view=vs-2022> [Accessed 28 January 2022].

Sarmiento, M. (2019) A Guide to Python’s Virtual Environments. Available from: <https://towardsdatascience.com/virtual-environments-104c62d48c54> [Accessed 7 February 2022].