# FIS

This repository contains the implementation of the Dutch Forensics system for the Secure Software Development course by

University of Essex.

Instructions for Execution

One could install the required dependencies with “pip install -r */path/to/requirements.txt*”. However, it is suggested to activate the existing virtual environment which already has all the required dependencies.

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. In both instances activate the virtual environment which contains all the required dependencies: 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

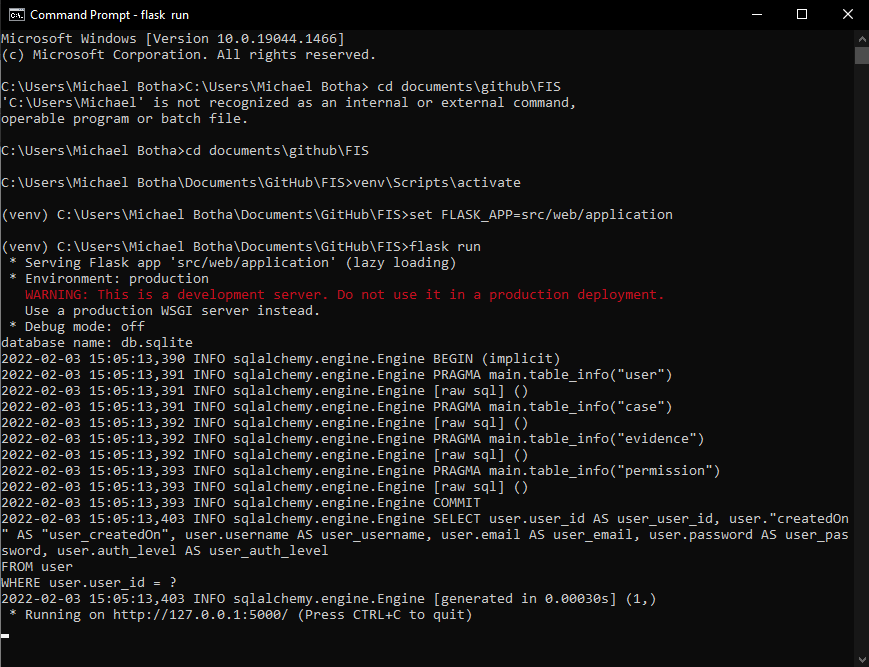
In figure 1 and 2 one can see the process for the two separate instances.

Once both Flask servers have been activated, one can open their 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.

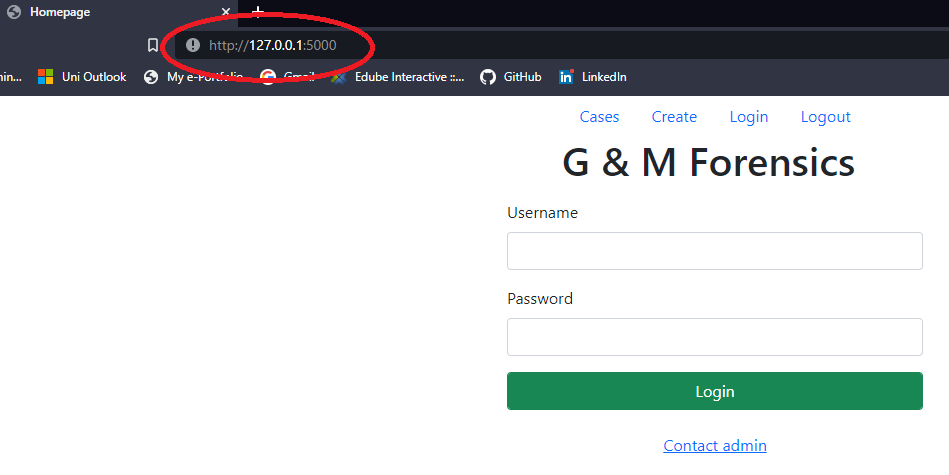
Text

Description automatically generated

**Figure 1**



**Figure 2**



**Figure 3**

# Differences between implementation and proposal

The list that follows represents the differences or the incomplete implementations realized versus what we designed in

the beginning.

\* RBAC authorization model hasn't been implemented. Instead, in PermissionRepository has been implemented a simpler

mechanism.

\* JsonWebToken are not used. Instead, the system verifies the credential and store current username in the session.

\* Search hasn't been implemented.

\* Pagination is incomplete feature.

# Project structure

###Source code

In the folder `src` there are all the project sources. The system is composed by 4 modules:

\* web, defines the web layer with all the templates and endpoint.

\* core, contains business logic, service classes and authorization.

\* data, defines the models through ORM definition (SqlAlchemy) and repository classes to implement CRUD operations on

the entities using the sqlalchemy ORM module.

\* auth, microservice for authentication.

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 more appropriate database engine as Postgres or MySQL, and the

use of SQLAlchemy ORM perfectly support to change database engine without any code modification.

### Test

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