Final project (Unit 11) versus design proposal (Unit 6)

Design document	**Achieved**	**How**
Monolith and API	Yes	• Shared database accessible via the Monolith web application and API. • Flask, Flask-RESTful, PyJWT
MVC pattern/framework	Yes	• Flask, Flask-SQLAlchemy and front-end-languages (View developed time shortened by using Bootstrap). • Folder structure.
Normalised database to 3NF	Yes	• Designed through an Entity Relationship Diagram
Good readability	Partially	*
Time-based One-Time Passwords	Yes	• PyOPT and Google authenticator
Weak password checks	Yes	• ReGex
Email verification	Yes	• Flask-mail and email-validator
Encryption of personal data in database	Partially	**
Five failed password attempts allowed	Yes	• Through the controller
Up to date tested libraries	Partially	***
Rate limit of 10 requests per hour on the API	No	****
Role based access control	Yes	• Everything designed around this
CRUD functionality	Yes	• flask-wtf • Downloading and uploading of images. • Ability to download a CVS of cases.
Command Line Interface (CLI)	Yes	• CLI implemented outside the MVC

* Having separate folders for Model, View and Controller makes it simple to find the source of functionality and errors. However, we could have made better use of commenting. Additional commenting was added because of feedback from Pylint, however, this was done shortly before hand-in and was not always consistent. In hindsight we would have added detailed comments as we went along.

** Passwords were hashed in the database using werkzeug security. However, everything else was stored in the database in plaintext. Our design document was vague about what exactly needed encrypting, however, if our system was to have been implemented for the Dutch National Police then all personal details should have been encrypted.

*** We used well known libraries and used Safety to check for known vulnerabilities. We created our system with the most recent packages available. However, once functionality was achieved, we used Python *freeze* to find out which versions were being used, and then specified these in our requirements.txt file. We decided against automatically using the most recent libraries going forward, for two reasons:

1. Although there are certain advantages of using the most up to date packages (such as bug fixes), security vulnerabilities are often not found straight away, thus negating some of these advantages.
2. If a package has been updated we would want to conduct white-box and black-box testing to make sure it did not cause any unforeseen issues.

**** We ran out of time to implement this. It was decided that, with time running short, the fact that the JWT tokens expired after two hours would at least offer some limit on API requests. If the system was really being implemented by the Dutch National Police a rate limit would have been necessary.