**PWS Project**

REST API For Remote File Storage and Encryption

Project by:

Arnav Sachdev

179301043

CSE – 7A

**Problem Statement:** To implement a RESTful web service prototype and show the interaction between the client and the server.

**Description:**

This project consists of a Python server written using the minimialist Flask framework which allows the user to statelessly transfer and retrieve files over HTTP/S.

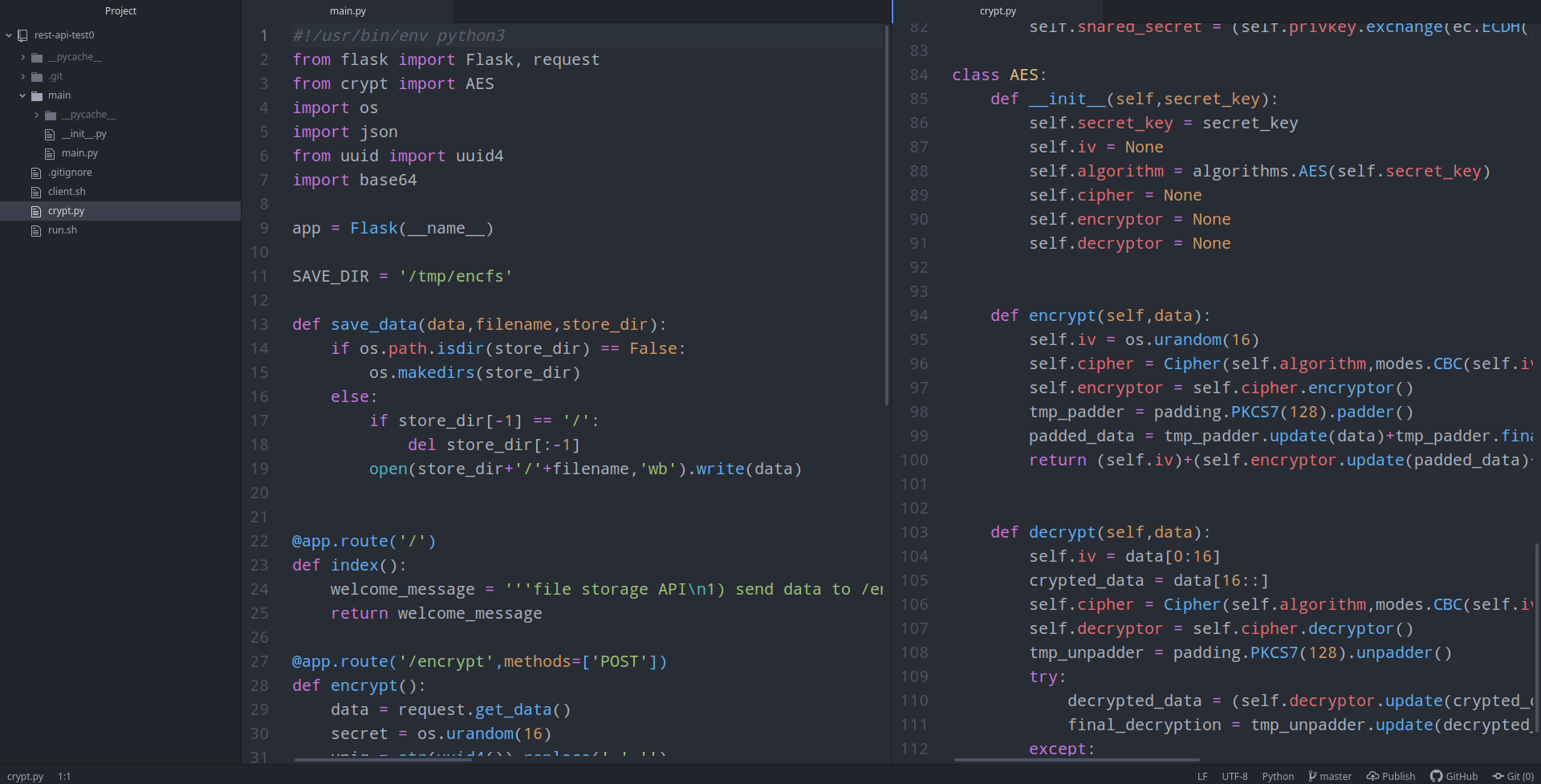
*The properties of the web service are:-*

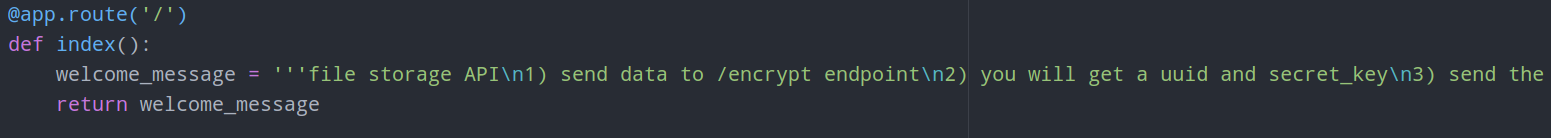
* Stateless transfer of files
* API endpoints which can be extended in functionality by other web services or client applications
* Files stored in the remote server are protected by AES-128 encryption
* The client and server are competely independent and new client software can easily be written by following the API rules.

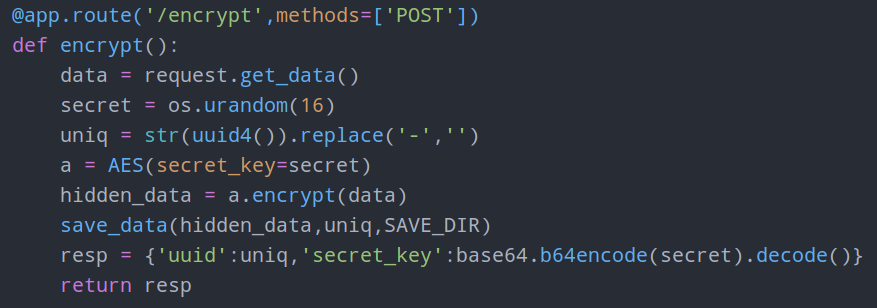
**Code:-**

The code is divided into 2 modules and one client application:

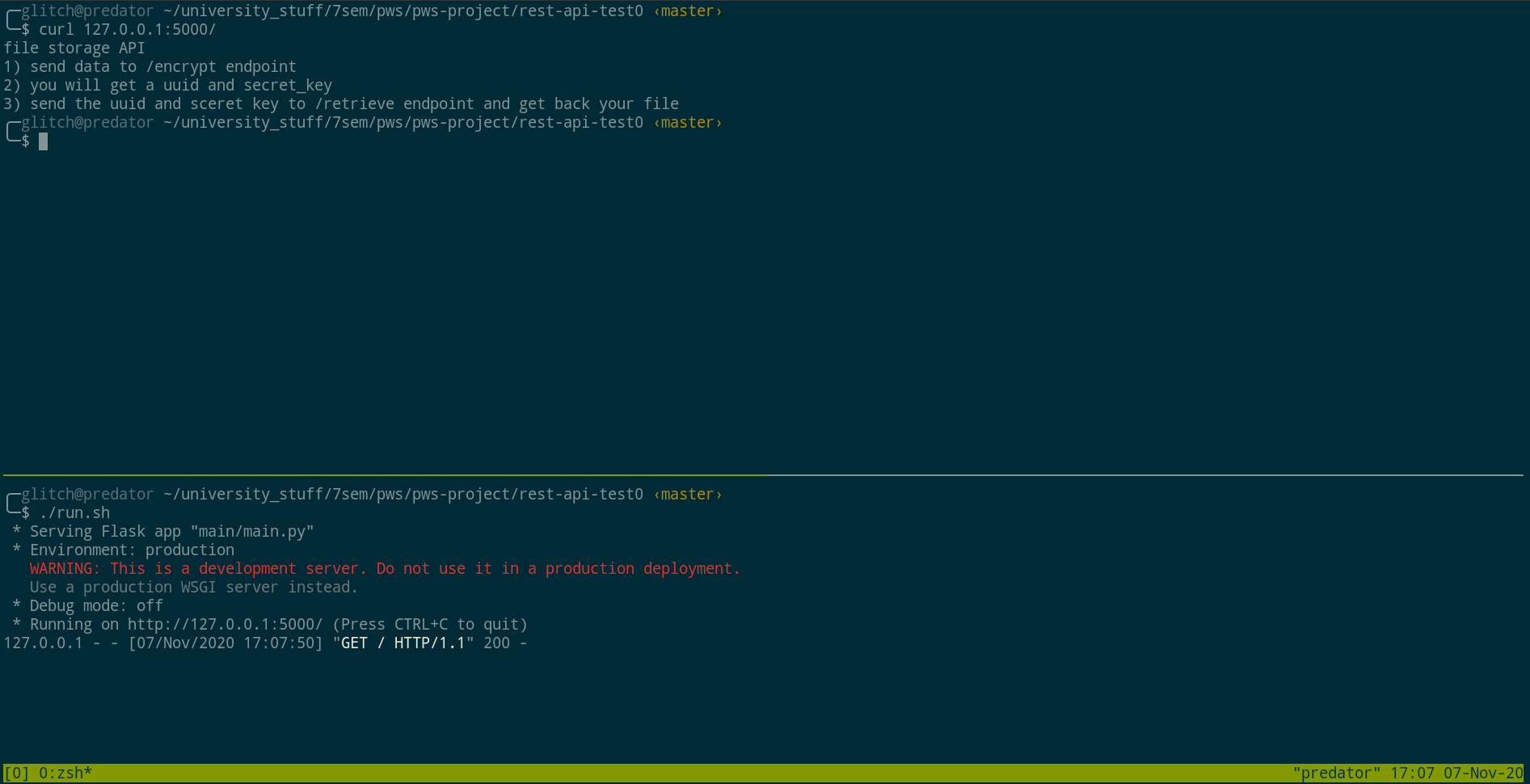
* main: *implements the web server and api*
* crypto: *implements the cryptographic functionality used by main*
* client.sh : *client to interact with the api written in shell script (using curl as the backend)*

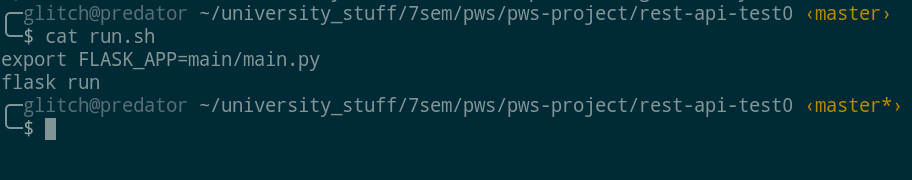
***Fig 1.1: The project directory structure and code***

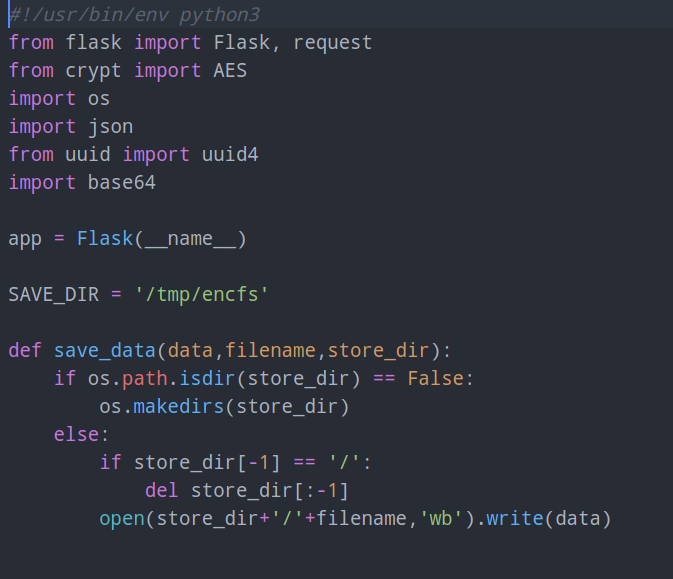
***Fig 1.2: / index endpoint***

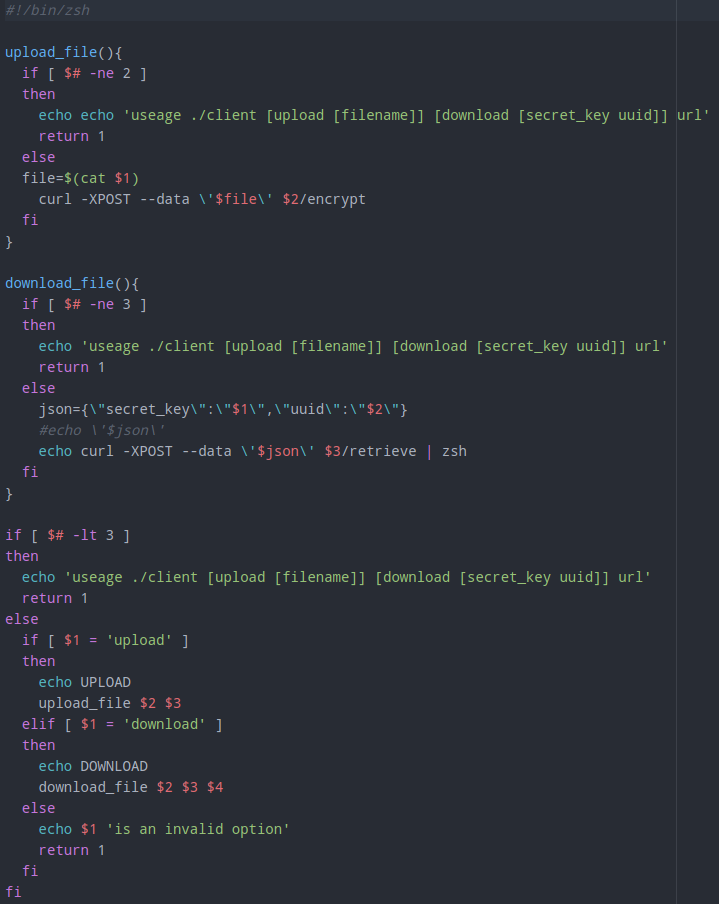
***Fig 1.3: /encrypt endpoint***

***Fig 1.4: /retrieve endpoint***

***Fig 1.5: Runtime screenshot of the webserver index***

***Fig 1.6: run.sh to run the application***

***Fig 1.7: Imports and helper functions***

***Fig 1.8: Client application***

**API Documentation:-**

The API contains **three** endpoints:

**a) /**

**Method-Type:** GET

**Response Content-Type:** text/plain

**Description:** The / or index endpoint returns basic information on the web service (as shown in the screenshot in **figure 1.5**)

**b) /encrypt**

**Method-Type:** POST

**Content-Type:** text/plain

**Response Content -Type:** application/json

**Description:** The endpoint accepts arbitrary raw data send via a HTTP POST request to *encrypt. The server reads the data in the POST request, generates a random 128 bit secret key and encrypts the POSTed data. Each file is identified by a UUID4 (universal unique identifier) of 32 bits and stored on the server’s filesystem (configureable). The returned data is of type* ***application/json***and contains two keys:

* uuid
* secret\_key

**c) /retrieve**

**Method-Type:** POST

**Content-Type:** application/json

**Response Content-Type:** text/plain

**Description:** The endpoint accepts json data containing the following two keys:

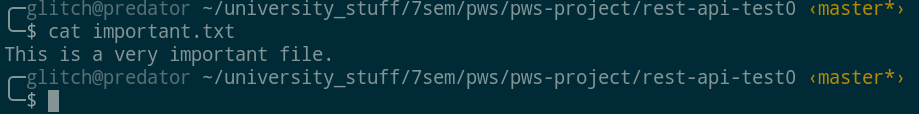
* uuid
* secret\_key

The server parses the json and if no errors are found and both keys are present, searches for any file on the server matching the requested UUID. If a file is found, then the secret\_key is used for it’s decryption and resulting data is returned.

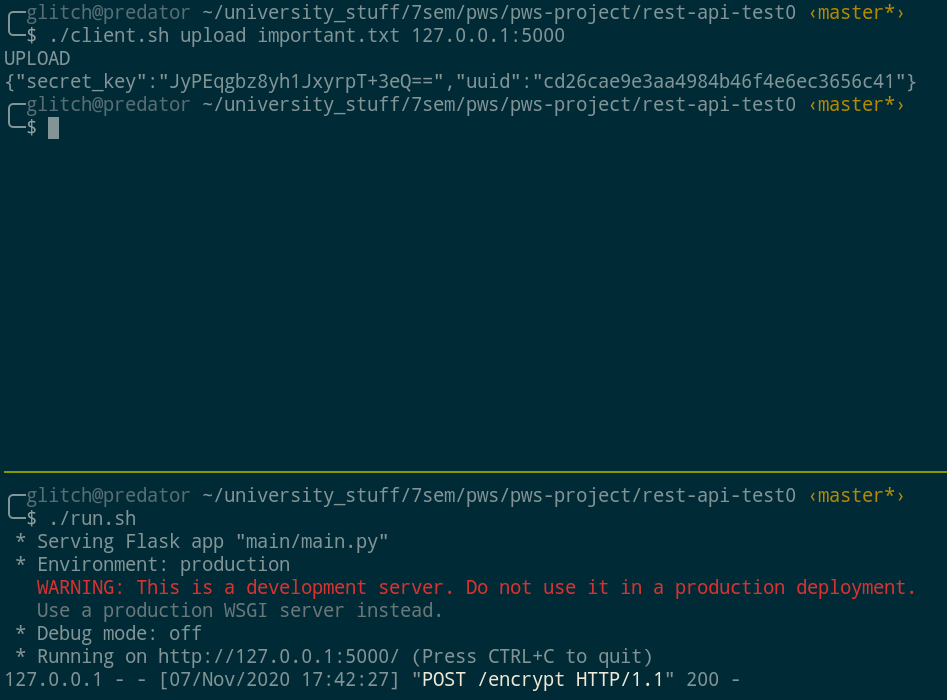
**Demonstration of Functionality:-**

**1) Uploading a file for encryption:-**

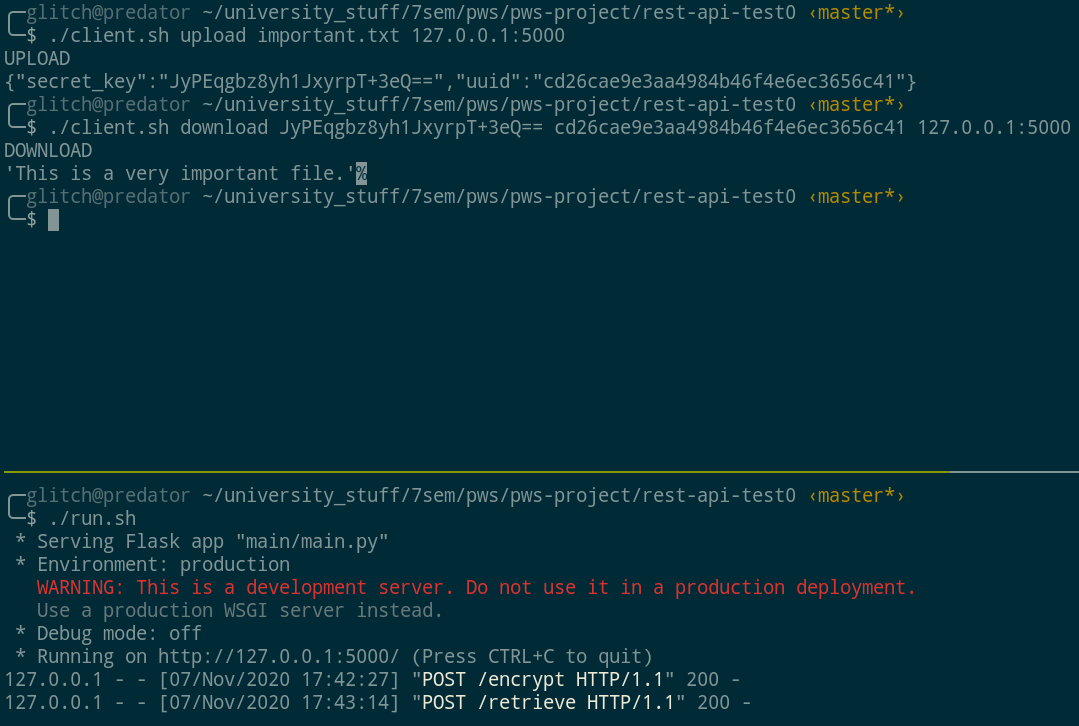
**a)** A file is ready for upload.



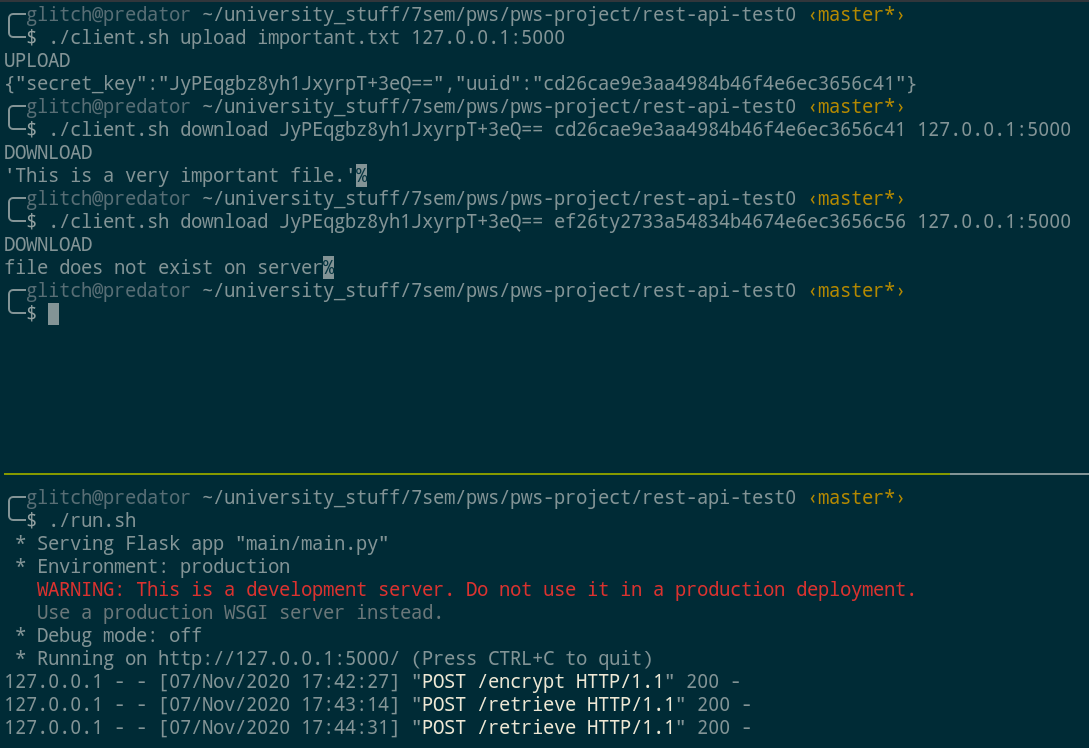
**b)** The file is uploaded and we get back a json object containing the secret key used for it’s encryption and a uuid to identify our file.



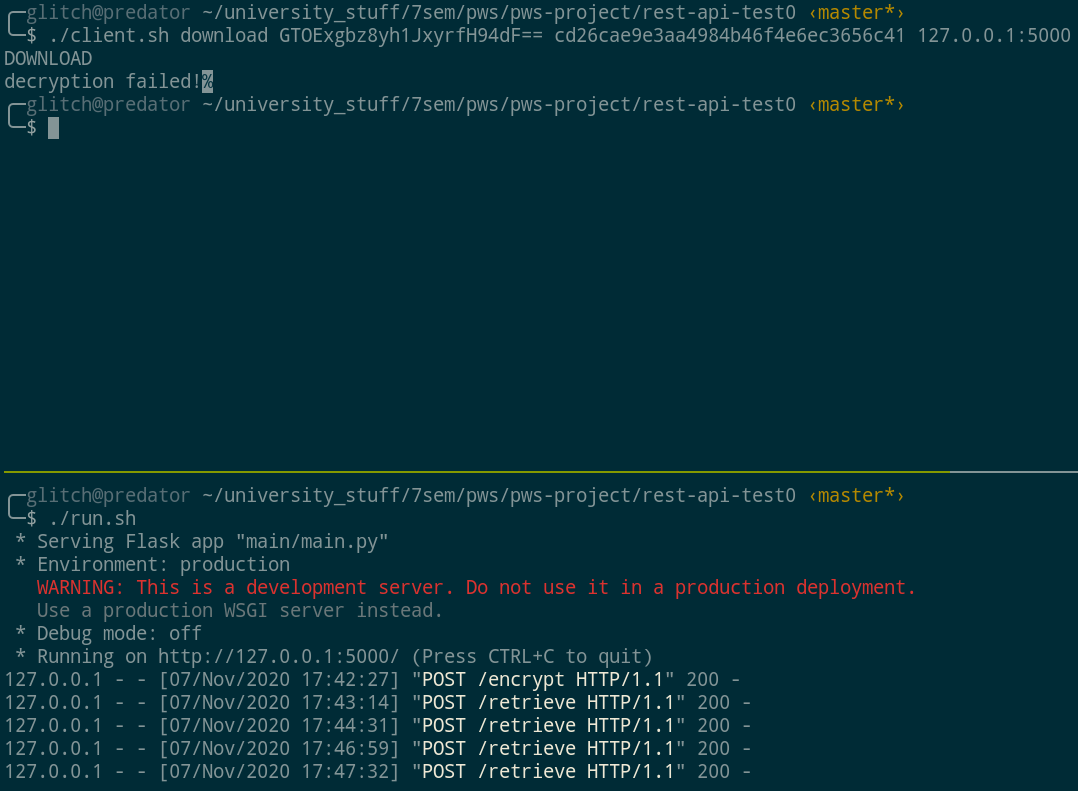
**c)** We use the give the client application the secret\_key, uuid and url for the web server and get our file back!



**d)** If for example we send an incorrect uuid, the api tells us that the file does not exist

****

**e)** If we send an incorrect decryption key, then the api tells us that the decryption failed



**Conclusion:**

To conclude, I have demonstrated that the API is stateless since the user’s session does not need to be retained. Simply a valid uuid and decryption key is necessary to retrieve a file.

It is also clear that the client and the server are completely indepedant, and all that is necessary for interacting with the API is HTTP. Infact, the server ignores HTTP headers as each endpoint has its own isolated function, thus we can interact with the API with various other tools as well, including netcat and Postman. For the purposes of this project I have written the client in shell script and utilized curl for making the HTTP requests.

**Dependancies/Requirements:**

* **Python3**
* **Flask framework**
* **python cryptography library (used in crypt.py)**
* **zsh**

**Ideally, should be run in a Linux environment. Use ./run.sh to start the server and ./client.sh to interact with the API.**