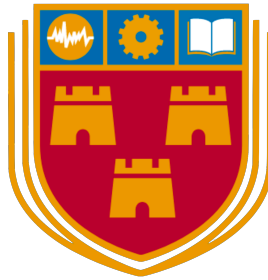


Secure Communication Platform Technical Manual



Liliana O'Sullivan

C00227188

Submitted in partial satisfaction of the requirements for the
degree of

BSc of Science (Hons) Software Development

Faculty of Science

Institute of Technology, Carlow

27th April 2021

Abstract

This technical manual showcases the code created within the project, and additional documentation developer documentation.

Table of Contents

1	Documentation	1
2	Codebase	20
2.1	Python	20
2.1.1	requirements.txt	20
2.1.2	FastAPI	21
	app.py	21
	bash.sh	24
	CassandraModels.py	25
	Config.py	26
	helper.py	27
	logging_config.yaml	29
	models.py	30
2.1.3	FastAPI - Routers	31
	ApiKey.py	31
	Potion.py	33
	TestClient.py	34
	User.py	35
2.1.4	Reference client	41
	gui.py	41
	gui_layouts.py	43
	helpers.py	47
2.2	Elixir	49
	ilopotion.ex	49
	router.ex	50
	socket_handler.ex	52
	application.js	53
	application.html.eex	54

2.3	HTML/CSS/JavaScript	55
	GenKeys.html	55
	Login.html	56
	sendMessage.html	58
	socket_code.js	60
2.4	Testing	61
	conftest.py	61
	mocks.py	63
	test_apikeys.py	64
	test_helper.py	66
	test_potion.py	68
	test_startup.py	69
	test_user.py	70

List of Figures

1.1	Created Developer Documentation	1
-----	---	---

List of files

2.1	requirements.txt	20
2.2	app.py	21
2.3	bash.sh	24
2.4	CassandraModels	25
2.5	Config.py	26
2.6	helper	27
2.7	logging config.yaml	29
2.8	models.py	30
2.9	ApiKey.py	31
2.10	Potion.py	33
2.11	TestClient.py	34
2.12	User.py	35
2.13	gui.py	41
2.14	gui_layouts.py	43
2.15	helpers.py	47
2.16	ilopotion.ex	49
2.17	router.ex	50
2.18	socket_handler.ex	52
2.19	application.js	53
2.20	application.html.eex	54
2.21	GenKeys.html	55
2.22	Login.html	56
2.23	sendMessage.html	58
2.24	socket_code.js	60

2.25	conftest.py	61
2.26	mocks.py	63
2.27	test_apikeys.py	64
2.28	test_helper.py	66
2.29	test_potion.py	68
2.30	test_startup.py	69
2.31	test_user.py	70

Chapter 1

Documentation

Developer aimed documentation is created and resides within the docs folder. It is additionally available online at lilianaosullivan.github.io/ilo/. The documentation in the HTML state can be seen in figure 1.1.

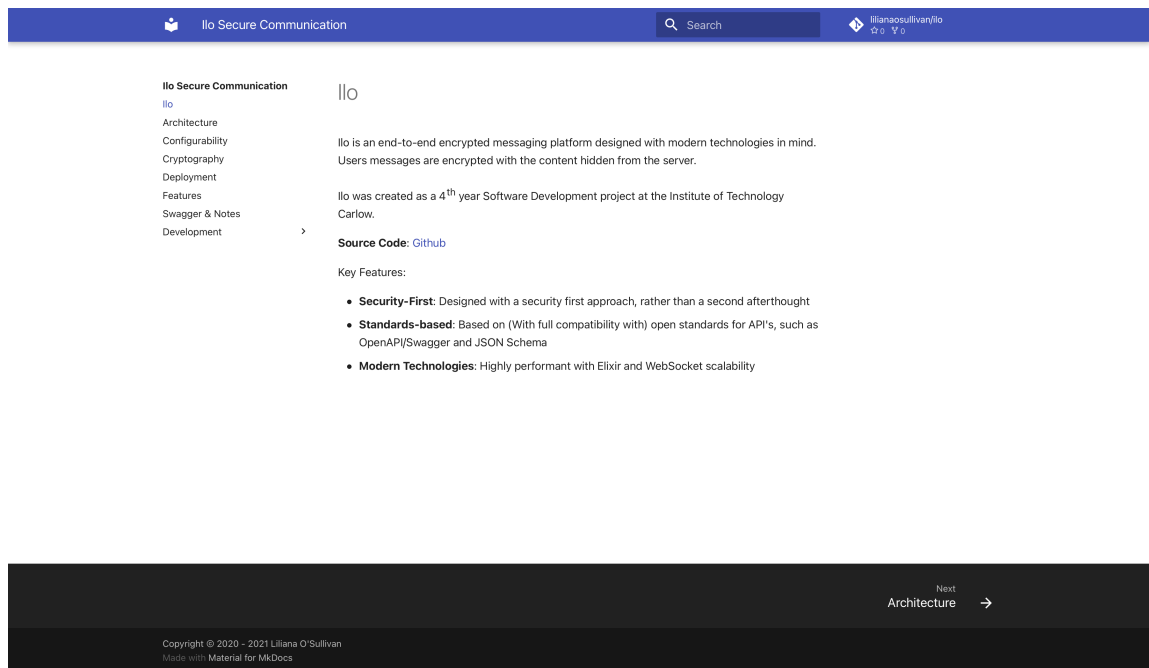


Figure 1.1: Created Developer Documentation

Additionally the content can be previewed as a PDF below.



Ilo Secure Communication

Security as a universal tool

Liliana O'Sullivan

Copyright © 2020 - 2021 Liliana O'Sullivan

Table of contents

1. Ilo	3
2. Architecture	4
2.1 Python 	6
2.2 Elixir 	6
2.3 Project layout	6
2.4 Additional Information	6
3. Configurability	7
3.1 Logging	7
3.2 General	7
4. Cryptography	8
5. Deployment	9
6. Features	10
6.1 Security First	10
6.2 Standards-based	10
6.3 Modern Technologies	10
7. Swagger & Notes	11
7.1 JSON Schema	11
7.2 Noteworthy information	12
7.3 ReDoc documentation	12
8. Development	14
8.1 Development - Intro	14
8.2 API Usage	15
8.3 Ilo Development	17

1. Ilo

Ilo is an end-to-end encrypted messaging platform designed with modern technologies in mind. Users messages are encrypted with the content hidden from the server.

Ilo was created as a 4th year Software Development project at the Institute of Technology Carlow.

Source Code: [Github](#)

Key Features:

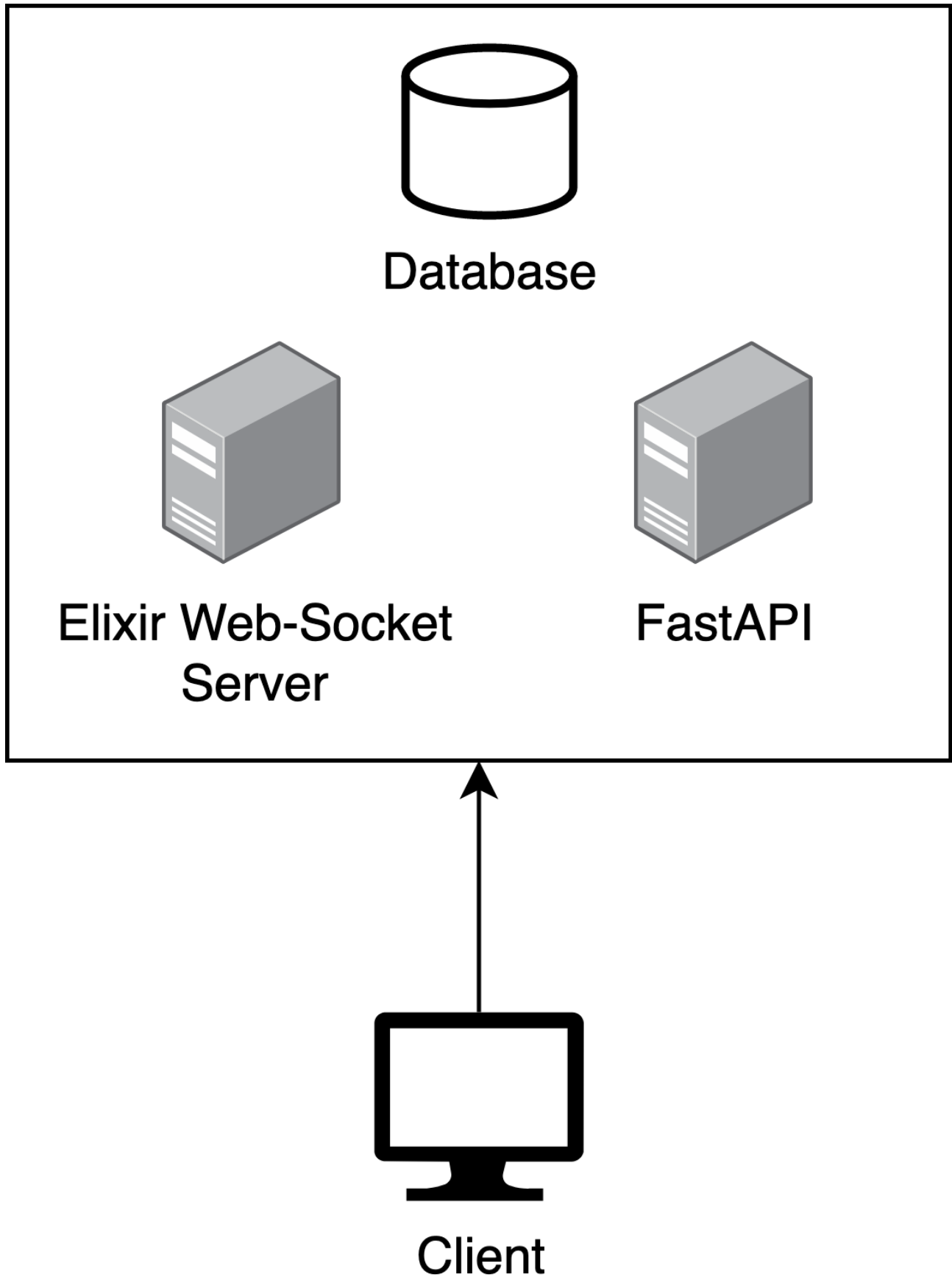
- **Security-First:** Designed with a security first approach, rather than a second afterthought
- **Standards-based:** Based on (With full compatibility with) open standards for API's, such as OpenAPI/Swagger and JSON Schema
- **Modern Technologies:** Highly performant with Elixir and WebSocket scalability

2. Architecture

Ilo is powered predominantly by two of the below technologies

- **Python** - FastAPI
- **Elixir** - Cowboy

The above technologies are combined in an encapsulated manner, much as showcased below.



2.1 Python



Python handles all the 'business' end of the platform. Such as the creation of users or the validation of API keys.

The Python backend is created using [FastAPI](#). FastAPI is a high-performance web framework built on Starlette and Pydantic.

2.2 Elixir



Elixir is referenced as *Potion* within the system. Potion is used as a WebServer for the sending of messages. Elixir is a functional programming language that specialises in concurrency and fault-tolerant systems. The WebSocket server is created using Cowboy.

A room is used to refer to a url identifier after the `/ws/...`. For example a room for the connection `62.3.11.253/ws/PythonProgramming` would be *PythonProgramming*. Each room has an isolated WebSocket connection, and users connected to a different room do not receive the messages of another room.

2.3 Project layout

The general folder structure can be seen below.

```

├── LICENSE
├── docs                    # All documentation is stored here
│   ├── docs
│   │   ├── css            # CSS for documentation
│   │   ├── img           # Images used in documentation
│   │   └── js            # JavaScript used in documentation
│   └── mkdocs.yml        # Configuration file for documentation
├── elixir                 # Potion system
├── python                 # FastAPI
│   ├── Config.py         # Empty class for storing general_config
│   ├── Helper.py         # Helper functions used in this system
│   ├── app.py            # 'main' FastAPI startup file
│   ├── client            # Folder containing legacy reference client
│   ├── models.py         # Contains models used by FastAPI
│   ├── py_client         # Tkinter reference client
│   ├── requirements.txt  # Used by pip to install required packages
│   ├── routers           # Contains separated routers used by FastAPI
│   └── run.sh            # Bash script to run with localhost SSL Certs

```

The above list is not exhaustive and has arbitrary files removed, such as log files for brevity and clarity purposes

2.4 Additonal Information

2.4.1 Password Hashing

Passwords are hashed using Argon2id by FastAPI and are stored in their hashed state.

3. Configurability

The platform includes two separate configuration files based on YAML

- Logging
- General

3.1 Logging

The logging configuration file contains customisation on how FastAPI will log. This file is created to be in a [Python Logging Configuration](#) format. For example, the filename of a log can be changed here, or the logging format can be customised here. A sample of a file can be seen below

```
formatters:
  simple:
    format: "[% (name)s | %(levelname)s]:%(lineno)s - %(message)s"
handlers:
  console:
    class: logging.StreamHandler
    formatter: main
    stream: ext://sys.stdout
loggers:
  user:
    level: DEBUG
    handlers: [console]
```

3.2 General

The general configuration controls variables upon deployment that are desirable to be changed without code changes. For example Cassandra's address. This file looks as follows

```
Cassandra_address: "127.0.0.1"
Cassandra_keyspace: "ilo"
Potion_IP: "0.0.0.0:4000"
```



Note

The Cassandra keyspace must be created, Ilo will not make it. A keyspace can be created with the following command within the CQL shell

```
CREATE keyspace ilo WITH replication={'class':'SimpleStrategy','replication_factor':3};
```

The tables used by Ilo will be created automatically.

4. Cryptography

As originally intended, Ilo enables the sending of plaintext and Cryptographic messages. Assuming a valid user exists, is sending the request, Potion will forward the message without verifying keys.

Potion will only enable requests from users that are logged in. A user can log in by sending a put request to `/user` to FastAPI. More information can be found at [Swagger](#). Due to the general-purpose design used within development, Ilo can support any Cryptographic algorithm. The expectation for clients is to use RSA with OAEP (Optimal Asymmetric Encryption Padding).

5. Deployment

Ilo requires FastAPI and Potion running concurrency to function correctly.

FastAPI can be deployed with multiple different WSGI/ASGI web servers. For an ASGI environment, [Uvicorn](#) is suggested.

For a WSGI environment, [Waitress](#) is recommended. If you are unsure of which to use, ASGI ([Uvicorn](#)) is recommended. Additional deployment information can be found [here](#).

FastAPI requires an instance of Cassandra. The Cassandra address and additional information can be configured within the [general config](#)

6. Features

Ilo provides the following features

6.1 Security First

Ilo was designed as a security-first platform, using modern algorithms (Such as Argon2id for hashing) and putting user privacy first. The platform was created as an inspiration for an end-to-end encrypted platform as users increasingly rely on platforms to be able to communicate.

6.2 Standards-based

The entire system is designed around OpenAPI and JSON Schema to conform to common practice in use today. The use of OpenAPI often can allow client code generation in many languages.

6.3 Modern Technologies

All technologies behind the project are based on maximising the capabilities of each technology. Development of the Python codebase utilises type-hinting, new features such as the walrus operator, docstring and more. Elixir's concurrency is utilised to send messages without Python overhead.

7. Swagger & Notes

FastAPI will generate Swagger documentation. This can be accessed with the IP Address, and accessing `/docs`, for example, `1.236.66.46:6921/docs`. An example of this can be seen below.

Ilo In Development OAS3
/openapi.json

A 4th year software development project to create an API that enables secure communication between multiple of its users.

Users All Operations with users profiles. ∨

- GET** `/user/{username}` Getpublickey
- OPTIONS** `/user/{username}` Usernameexists
- PUT** `/user` Login a user.
- POST** `/user` Creates a user.
- DELETE** `/user` Deletes a user.
- HEAD** `/user` Deletes a user.

API Keys Manage API Keys. Enables the generation and deletion of keys. ∨

- POST** `/key/` Create a API Key
- DELETE** `/key/{key}` Delete an API Key

This will showcase the API made available through FastAPI. Additionally navigating to the `/openapi.json`, for example, `1.236.66.46:6921/openapi.json` will provide a JSON respecting the OpenAPI format of the public-facing functions. An excerpt is shown below

```
{
  "openapi": "3.0.2",
  "info": {
    "title": "Ilo",
    "description": "A 4th year software development project .....",
    "version": "In Development"
  },
  "paths": {
    "/user/{username}": {
      "get": {
        "tags": [
          "Users"
        ],
        "summary": "Getpublickey",
        "operationId": "getPublicKey_user_username_get",
        "parameters": [
          {
            "required": true,
            "schema": {
              "title": "Username",
              "type": "string"
            },
            "name": "username",
            "in": "path"
          }
        ]
      }
    }
  }
}
```

7.1 JSON Schema

In addition, a [JSON Schema](#) is provided for each OpenAPI function. This is annotating how a JSON should be formatted to be accepted as a valid request, this can include the use of optional parameters with the request. This information is provided by the Swagger documentation and is provided as part of the OpenAPI JSON. The server will reject the request if the schema is not abided to.

An example of what a Schema may look like is shown below

```

User ∨ {
  username*           string
                       title: Username
  password*         string
                       title: Password
  public_key*       string
                       title: Public
                       Key
  api_key*          string
                       title: Api Key
}

```

This schema in its' raw format from the OpenAPI JSON is shown below

```

"User":
{
  "title": "User",
  "required": [
    "username",
    "password",
    "public_key",
    "api_key"
  ],
  "type": "object",
  "properties":
  {
    "username": {
      "title": "Username",
      "type": "string"
    },
    "password": {
      "title": "Password",
      "type": "string"
    },
    "public_key": {
      "title": "Public Key",
      "type": "string"
    },
    "api_key": {
      "title": "Api Key",
      "type": "string"
    }
  }
}

```

7.2 Noteworthy information

7.3 ReDoc documentation

Documentation in the [ReDoc](#) format can be accessed by navigating to `/redoc`.

The image shows two parts of the Ilo API documentation. On the left is a light-themed page for the 'Users' endpoint. It includes a sidebar with 'Users', 'API Keys', and 'Test Client'. The main content area is titled 'Ilo (In Development)' and provides a 'Download' link for the OpenAPI specification. Below this, it lists 'Users' as the endpoint, followed by 'Getpublickey' with a path parameter 'username' of type 'string'. The 'Responses' section shows a '200 Successful Response' and a '422 Validation Error'. The right part of the image shows a dark-themed API client interface. It displays a 'GET /user/{username}' endpoint with 'Response samples' for status codes 200 and 422. The 200 response is shown as 'null' with a content type of 'application/json'. Below this, there are 'OPTIONS /user/{username}' and another 'Response samples' section for 200 and 422.

7.3.1 User Creation

A user must have a password. Ilo's minimum password requirements are

- Length of 8 characters
- Contain minimum 1 number
- Contain minimum 1 lowercase letter
- Contain minimum 1 uppercase letter
- 1 non-alphanumeric character

7.3.2 Unicode support

The platform has support for `u[?]c,d` characters in sending messages and usernames. This allows non-latin characters to be used, or support for sending emotes such as 🍌🍌🍌🍌🍌

8. Development

8.1 Development - Intro

This section will outline development operations on the Ilo platform. This can be in the form of using Ilo as a backend, or developing the core platform further.

8.2 API Usage

This section documents the usage of Ilo as a backend; It is highly recommended to have glanced at the [architecture](#) section, this section assumes some technical insights of Ilo.

A client must at minimum support

- **WebSockets:** This is a requirement for the sending of messages between Potion and the client.
- **JSON Parsing:** The information exchanged within the WebSocket connection is in a JSON format, and as such is essential to the understanding of the information exchanged.

To access public-facing methods, all information is exchanged using an appropriate HTTP method, and a [JSON schema](#). These are documented within the [Swagger](#) section.

8.2.1 Message Sending

To send a message, the following pre-conditions must exist

- An API Key must be obtained [1](#)
- A user must be created [2](#)
- A user must be logged-in [3](#)

Obtaining an API Key

An API Key can be obtained by sending a *POST* request to `/key`. The API will reply with a JSON as follows

```
{
  "detail": "824a47ae-d0b9-5350-bffb-cc2ee48424a3"
}
```

User Creation

A user can be created by sending a *POST* request to `/user`, with a JSON attached contained the required information, a sample has been provided below. Information on password requirements can be seen [here](#)

```
{
  "username": "cookielover57🍪",
  "password": "MySuperSecurePassword57!",
  "public_key": "XzKSzqiX2qoPySbe5T4TSK2018V...",
  "api_key": "824a47ae-d0b9-5350-bffb-cc2ee48424a3"
}
```

Log a user in

To log a user in, we send a *PUT* request to `/user`. The JSON we send along with this request will look as follows

```
{
  "username": "cookielover57🍪",
  "password": "MySuperSecurePassword57!",
  "api_key": "824a47ae-d0b9-5350-bffb-cc2ee48424a3"
}
```

8.2.2 Reference Client

A reference client implemented in Python's Tkinter GUI toolkit has been created to provide a sample implementation of a client. This can be found within the Python folder, under `py_client` or the project path of `ilo/python/py_client`.

This client is aimed to showcase the use of the Ilo API. It provided the following functionality

- Registering a user
- Connecting to a Potion room
- Changing rooms

8.3 Ilo Development

Ilo requires Python and Elixir to function, with both FastAPI and Potion running concurrency to function correctly.

Python can be downloaded from the official [Python.org](https://www.python.org/) website. Once downloaded and installed, the required Python libraries can be installed from PyPi using `pip requirements`



bash

```
ilo/python $ pip install -r requirements.txt
```

Within the Python folder, there is a bash to run FastAPI with `localhost` certificates. A localhost SSL certificate needs to be created within the python directory. To do this enter the following command at the terminal. Additional information on localhost certificate generation can be found [here](#)

```
openssl req -x509 -out localhost.crt -keyout localhost.key \
-newkey rsa:2048 -nodes -sha256 \
-subj '/CN=localhost' -extensions EXT -config <( \
printf "[dn]\nCN=localhost\n[req]\ndistinguished_name = dn\n[EXT]\nsubjectAltName=DNS:localhost\nkeyUsage=digitalSignature\nextendedKeyUsage=serverAuth")
```

The FastAPI server will additionally need a Cassandra backend to function. Specifically, it requires the creation of a keyspace as specified in the `general_config.yaml/Cassandra_keyspace`

The Cassandra address and keyspace can be specified in the `general_config.yaml` file within the python folder. All tables will be created automatically by FastAPI.

A bash script is created to start the FastAPI server with the localhost SSL certificate. It will start the server on port 7999.



bash

```
ilo/python $ bash run.sh
```

Potion dependencies can be installed using `mix deps.get` from the potion folder



bash

```
ilo/elixir $ mix deps.get
```

Potion can be launched in interactive mode with `iex`



```
bash
```

```
ilo/elixir $ iex -S mix
```

8.3.1 Potion -> FastAPI communication

Potion does communicate to FastAPI. This occurs once upon user connection to validate a user is logged in to enable communication. This is a HTTP call made to FastAPI. This can be found in the `python/routers/Potion.py`. It has Potion's IP Address whitelisted from the `python/general_config.yaml`

Chapter 2

Codebase

The final codebase can be found among the below sections.

2.1 Python

Multiple requirements files exist within the project, they are plaintext files that contain the required packages to be used by the Python Pip tool. An example of the FastAPI Python server can be seen below.

2.1.1 requirements.txt

```
1 fastapi
2 PyYAML
3 argon2-cffi
4 uvicorn
5 cassandra-driver
```

File 2.1: requirements.txt

2.1.2 FastAPI

app.py

```

1 import logging
2 import logging.config
3 import os
4 import sys
5 from typing import Dict
6
7 import yaml
8 from cassandra.cqlengine import connection
9 from cassandra.cqlengine.management import sync_table
10 from fastapi import FastAPI
11
12 import Helper
13 from CassandraModels import *
14 from Config import Config
15
16
17 def startup():
18
19     import resource
20
21     ## Fixes the issue "Too many open files error 24"
22     ## https://stackoverflow.com/questions/2569620/socket-accept-
23     error-24-to-many-open-files
24     resource.setrlimit(resource.RLIMIT_NOFILE, (65536, 65536))
25
26     if not os.path.exists("logging_config.yaml"):
27         with open("logging_config.yaml", "w") as f:
28             f.write(Config.LOGGING_CONFIG_DEFAULT)
29             print("Created logging_config.yaml. Using default
30 settings...")
31         logging.config.dictConfig(yaml.safe_load(Config.
32 LOGGING_CONFIG_DEFAULT))
33     else:
34         try:
35             with open("logging_config.yaml", "r") as y:
36                 logging.config.dictConfig(yaml.safe_load(y.read()))
37         except:
38             print("Error Parsing logging_config. Using default Ilo
39 configuration")
40             logging.config.dictConfig(yaml.safe_load(Config.
41 LOGGING_CONFIG_DEFAULT))
42
43     _generalLogger = logging.getLogger("general")
44     _generalLogger.info("Starting ilo...Parsing general config")
45     if not os.path.exists("general_config.yaml"):
46         with open("general_config.yaml", "w") as f:
47             f.write(Config.GENERAL_CONFIG_DEFAULT)
48             print("Created general_config.yaml. Using default
49 settings...")

```

```

44         _generalLogger.info(
45             "Created general_config.yaml. Using default settings
... "
46         )
47     else:
48         config: Dict = {}
49         try:
50             with open("general_config.yaml", "r") as y:
51                 config = yaml.safe_load(y.read())
52         except:
53             print("Error parsing general_config. Using default Ilo
configuration")
54             config = yaml.safe_load(Config.GENERAL_CONFIG_DEFAULT)
55             for k, v in config.items():
56                 k, v = k.strip(), v.strip()
57                 if len(k) != 0 and len(v) != 0:
58                     setattr(Config, k, v)
59             del config
60         _generalLogger.info("Creating Helper")
61         Helper.logger = _generalLogger
62         _generalLogger.info("Connecting to Cassandra")
63         try:
64             connection.setup(
65                 [Config.Cassandra_address], Config.Cassandra_keyspace,
protocol_version=3
66             )
67             sync_table(users, [Config.Cassandra_keyspace])
68             sync_table(api_keys, [Config.Cassandra_keyspace])
69         except Exception as e:
70             _generalLogger.critical(
71                 "Failed to connect to Cassandra with the following
exception"
72             )
73             _generalLogger.exception(e)
74             print(e)
75             sys.exit(1)
76         _generalLogger.info("Starting FastAPI")
77
78
79 startup()
80
81 tags_metadata = [
82     {
83         "name": "Users",
84         "description": "All Operations with users profiles.",
85     },
86     {
87         "name": "API Keys",
88         "description": "Manage API Keys. Enables the generation and
deletion of keys.",
89     },
90     {
91         "name": "Test Client",

```

```
92         "description": "A test client showcasing API usage from a
93         browser",
94     ],
95     app = FastAPI(
96         title="Ilo",
97         description="A 4th year software development project to create
98         an API that enables secure communication between multiple of its
99         users.",
100        version="In Development",
101        openapi_tags=tags_metadata,
102    )
103
104 from routers.ApiKey import ApiKeyRouter
105 from routers.Potion import potionRouter
106 from routers.TestClient import TestClient
107 from routers.User import UserRouter
108
109 app.include_router(UserRouter)
110 app.include_router(ApiKeyRouter)
111 app.include_router(TestClient)
112 app.include_router(potionRouter)
113
114 if __name__ == "__main__":
115     import uvicorn
116     uvicorn.run(app, host="0.0.0.0", port=8000)
```

File 2.2: app.py

bash.sh

```
1 #!/bin/bash
2 uvicorn app:app --ssl-keyfile localhost.key --ssl-certfile localhost
  .cert --port 7999
```

File 2.3: bash.sh

CassandraModels.py

```
1 import uuid
2 import time
3
4 from cassandra.cqlengine import columns
5 from cassandra.cqlengine.management import sync_table
6 from cassandra.cqlengine.models import Model
7
8
9 class users(Model):
10     user_id      = columns.UUID(primary_key=True, default=uuid.uuid4)
11     username     = columns.Text(required=True, index=True)
12     password     = columns.Text(required=True)
13     public_key   = columns.Text(required=True)
14     logged_in    = columns.Text(required=False, index=True)
15     login_time   = columns.Double(default=time.time)
16     api_key      = columns.Text(required=False)
17
18 class api_keys(Model):
19     key_id        = columns.UUID(primary_key=True, default=uuid.
20     creation_epoch = columns.Double(required=True)
```

File 2.4: CassandraModels

Config.py

```

1 from typing import Final
2
3
4 class Config:
5     Cassandra_address: str = "127.0.0.1"
6     Cassandra_keyspace: str = "ilo"
7     Potion_IP: str = "0.0.0.0:4000"
8
9     LOGGING_CONFIG_DEFAULT: Final = """version: 1
10 formatters:
11     simple:
12         format: "[% (name)s | %(levelname)s]:%(lineno)s - %(message)s"
13     "
14     main:
15         format: "[% (asctime)s]{%(name)s | %(levelname)s}:%(lineno)s
16         - %(message)s"
17 handlers:
18     console:
19         class: logging.StreamHandler
20         level: DEBUG
21         formatter: main
22         stream: ext://sys.stdout
23     users:
24         class: logging.handlers.RotatingFileHandler
25         formatter: main
26         filename: user.log
27     general:
28         class: logging.handlers.RotatingFileHandler
29         formatter: main
30         filename: general.log
31     api:
32         class: logging.handlers.RotatingFileHandler
33         formatter: main
34         filename: api.log
35 loggers:
36     user:
37         level: DEBUG
38         handlers: [users]
39     general:
40         level: DEBUG
41         handlers: [general]
42     api:
43         level: DEBUG
44         handlers: [api]"""
45
46     GENERAL_CONFIG_DEFAULT: Final = """Cassandra_address:
47     "127.0.0.1"
48     Cassandra_keyspace: "ilo"
49     Potion_IP: "0.0.0.0:4000"
50     """

```

File 2.5: Config.py

helper.py

```

1 import logging
2 import re
3
4 from CassandraModels import api_keys
5
6 logger: logging.Logger = None
7
8
9 def validate_APIKey(
10     key: str,
11 ) -> bool:
12     """
13     Checks if an API Key is valid. Returns False if it's empty
14
15     Parameters:
16         key (str): The key to be validated
17
18     Returns:
19         bool: True if the key is valid, False if its invalid or
20     empty.
21     """
22     if not (key := key.strip()):
23         return False
24     query = api_keys.objects(key_id=key)
25     return False if query.count() == 0 else True
26
27 def validate_password(password: str) -> bool:
28     """
29     Validates that a password is meeting the minimum criteria, with
30     confirmation.
31     It must at minimum
32     - Be of length 8
33     - Contain minimum 1 number
34     - Contain minimum 1 lowercase letter
35     - Contain minimum 1 uppercase letter
36     - 1 non-alphanumeric character
37
38     Args:
39         password (str): The password to validate
40
41     Returns:
42         bool: True if valid, False if invalid
43     """
44     if len(password) < 8:
45         return False
46     if re.search("[0-9]+", password) == None: # Number
47         return False
48     if re.search("[a-z]+", password) == None: # Lowercase
49         return False
50     if re.search("[A-Z]+", password) == None: # Uppercase
51         return False

```

```
51     if re.search("[^\w\d\s]+", password) == None: # Special
character
52         return False
53     return True
```

File 2.6: helper

logging_config.yaml

```
1 version: 1
2 formatters:
3     simple:
4         format: "[% (name)s | %(levelname)s]:%(lineno)s - %(message)s"
5     "
6     main:
7         format: "[% (asctime)s]{%(name)s | %(levelname)s}:%(lineno)s
8         - %(message)s"
9 handlers:
10    console:
11        class: logging.StreamHandler
12        level: DEBUG
13        formatter: main
14        stream: ext://sys.stdout
15    users:
16        class: logging.handlers.RotatingFileHandler
17        formatter: main
18        filename: user.log
19    general:
20        class: logging.handlers.RotatingFileHandler
21        formatter: main
22        filename: general.log
23    api:
24        class: logging.handlers.RotatingFileHandler
25        formatter: main
26        filename: api.log
27 loggers:
28    user:
29        level: DEBUG
30        handlers: [users]
31    general:
32        level: DEBUG
33        handlers: [general]
34    api:
35        level: DEBUG
36        handlers: [api]
```

File 2.7: logging config.yaml

models.py

```
1 from pydantic import BaseModel
2
3
4 class User(BaseModel):
5     username: str
6     password: str
7     public_key: str
8     api_key: str
9
10
11 class Detail(BaseModel):
12     detail: str = ""
```

File 2.8: models.py

2.1.3 FastAPI - Routers

ApiKey.py

```

1 import logging
2 import time
3 from http import HTTPStatus
4 from cassandra.cqlengine.query import DoesNotExist
5
6
7 from fastapi.exceptions import HTTPException
8 from fastapi.routing import APIRouter
9 from models import Detail
10 from CassandraModels import *
11 from cassandra.cqlengine import ValidationError
12
13 # from cassandra.cqlengine.models import DoesNotExist
14
15 ApiKeyRouter: APIRouter = APIRouter(tags=["API Keys"])
16 _apiLogger: logging.Logger = logging.getLogger("api")
17
18 # Create API Key
19 @ApiKeyRouter.post(
20     path="/key/",
21     status_code=HTTPStatus.CREATED,
22     summary="Create a API Key",
23     responses={
24         HTTPStatus.CREATED.value: {
25             "description": "201 response is sent on a successful
creation of a API Key.",
26             "model": Detail,
27             "content": {
28                 "application/json": {
29                     "example": {"detail": "b24aj62cb-1625-4ab5-212b-
aah08cxc9a"}
30             },
31         },
32     },
33 )
34
35 def createKey():
36     key = api_keys.create(creation_epoch=time.time())
37     return {"detail": key.key_id}
38
39
40 @ApiKeyRouter.delete(
41     path="/key/{key}",
42     status_code=HTTPStatus.OK,
43     summary="Delete an API Key",
44     responses={
45         HTTPStatus.BAD_REQUEST.value: {
46             "description": "Occurs on a invalid key submitted",
47             "model": Detail,

```

```

48         "content": {
49             "application/json": {
50                 "example": {"detail": "The API Key 123-456-789
is invalid"}
51             },
52         },
53     },
54     HTTPStatus.NOT_FOUND.value: {
55         "description": "Occurs validly formatted key that does
not exist is submitted",
56         "model": Detail,
57         "content": {
58             "application/json": {
59                 "example": {"detail": "The API Key 123-456-789
does not exist"}
60             },
61         },
62     },
63     HTTPStatus.OK.value: {
64         "description": "Occurs on a successful deletion",
65         "model": Detail,
66         "content": {
67             "application/json": {
68                 "example": {
69                     "detail": "Successfully deleted 123-456-789
does not exist"
70                 }
71             },
72         },
73     },
74 },
75 )
76 def deleteKey(key: str):
77     try:
78         key = api_keys.get(key_id=key)
79         api_keys.delete(key)
80     except ValidationError as ex:
81         raise HTTPException(
82             status_code=HTTPStatus.BAD_REQUEST.value,
83             detail=f"The APIKey {key} is not a valid key",
84         )
85     except DoesNotExist as e:
86         raise HTTPException(
87             status_code=HTTPStatus.NOT_FOUND.value,
88             detail=f"The APIKey {key} is does not exist",
89         )
90
91     return {"detail": f"Successfully deleted {key}"}

```

File 2.9: ApiKey.py

Potion.py

```
1 from CassandraModels import users
2 from Config import Config
3 from fastapi import Form, HTTPException, Request
4 from fastapi.routing import APIRouter
5 from http import HTTPStatus
6
7 potionRouter = APIRouter()
8
9
10 @potionRouter.post("/potion", include_in_schema=False)
11 def loggedIn(request: Request, address: str = Form("address")):
12     """Returns True or False depending on a users logged in state
13
14     Args:
15         username (str): Username to be checked
16
17     Returns:
18         bool: True if logged in, False if logged out.
19     """
20     if not request.client.host == Config.Potion_IP:
21         raise HTTPException(status_code=HTTPStatus.UNAUTHORIZED)
22     query = users.objects(logged_in=request.client.host)
23     return True if query.count() > 0 else False
```

File 2.10: Potion.py

TestClient.py

```
1 from fastapi.responses import HTMLResponse
2 from fastapi.routing import APIRouter
3
4 TestClient: APIRouter = APIRouter(
5     default_response_class=HTMLResponse, tags=["Test Client"]
6 )
7
8
9 @TestClient.get(path="/")
10 def home():
11     with open("client/login.html") as f:
12         return f.read()
13
14
15 @TestClient.get(path="/message")
16 def message():
17     with open("client/sendMessage.html") as f:
18         return f.read()
19
20 @TestClient.get(path="/cryptodemo")
21 def demo():
22     with open("client/SubtleCryptoDemo.html") as f:
23         return f.read()
24
25 @TestClient.get(path="/genkey")
26 def genkeys():
27     with open("client/GenKeys.html") as f:
28         return f.read()
29
30
31 @TestClient.get(
32     path="/client/static/socket_code.js",
33     include_in_schema=False,
34 )
35 def jsFile():
36     with open("client/static/socket_code.js") as f:
37         return f.read()
```

File 2.11: TestClient.py

User.py

```

1 import logging
2 from http import HTTPStatus
3
4 import Helper
5 from argon2 import PasswordHasher
6 from argon2.exceptions import VerificationError, VerifyMismatchError
7 from CassandraModels import users
8 from fastapi import APIRouter, HTTPException, Request
9 from models import Detail, User
10
11 UserRouter: APIRouter = APIRouter(tags=["Users"])
12
13 _hasher = PasswordHasher()
14 _userLogger: logging.Logger = logging.getLogger("user")
15
16
17 @UserRouter.options(
18     path="/user/{username}",
19     status_code=HTTPStatus.OK,
20     summary="Checks if a username exists.",
21     responses={
22         HTTPStatus.OK.value: {
23             "description": "Returned as a success if the username
24 exists or not.",
25             "model": Detail,
26             "content": {
27                 "application/json": {"example": {"detail": "true"}}},
28             },
29     },
30 )
31 def usernameExists(username: str):
32     query = users.objects(username=username)
33     if query.count() > 0:
34         return {"detail": "true"}
35     return {"detail": "false"}
36
37
38 @UserRouter.get(
39     path="/user/{username}",
40     status_code=HTTPStatus.OK,
41     summary="Get the users public key.",
42     responses={
43         HTTPStatus.CONFLICT.value: {
44             "description": "Occurs if a username does not exists, or
45 they are not logged in.",
46             "model": Detail,
47             "content": {
48                 "application/json": {
49                     "example": {"detail": "The user Alex13 does not
50 exist"}
51             },
52     },

```

```

50     },
51   },
52   HTTPStatus.OK.value: {
53     "description": "This is returned with the public key.",
54     "model": Detail,
55     "content": {
56       "application/json": {
57         "example": {"detail": "
rbdm4YHtr4d8ykwZ8DxLOSHDUQJ"}
58       },
59     },
60   },
61 },
62 )
63 def getPublicKey(username: str):
64   query = users.objects(username=username)
65   if query.count() == 0:
66     raise HTTPException(
67       status_code=HTTPStatus.CONFLICT,
68       detail=f"The user {username} does not exist",
69     )
70   return {"detail": query[0].public_key}
71
72
73 # Create User
74 @UserRouter.post(
75   path="/user",
76   status_code=HTTPStatus.CREATED,
77   summary="Creates a user.",
78   responses={
79     HTTPStatus.CONFLICT.value: {
80       "description": "Occurs if a user already exists.",
81       "model": Detail,
82       "content": {
83         "application/json": {
84           "example": {
85             "detail": "User already exists. A logged in
user can delete their account by a delete request to /user"
86           }
87         },
88       },
89     },
90     HTTPStatus.BAD_GATEWAY.value: {
91       "description": "Occurs if the password does not meet
minimum requirements.",
92       "model": Detail,
93       "content": {
94         "application/json": {
95           "example": {
96             "detail": "This password does not meet the
minimum requirements."
97           }
98         },
99     },

```

```

100     },
101     HTTPStatus.CREATED.value: {
102         "description": "Returned upon successful creation",
103         "model": Detail,
104         "content": {
105             "application/json": {
106                 "example": {"detail": "Successfully created
Alex13"}
107         },
108     },
109 },
110 },
111 )
112 def createUser(user: User):
113     _userLogger.info(f"Processing Create :{user.username}")
114     if not Helper.validate_APIKey(key=user.api_key):
115         raise HTTPException(
116             status_code=HTTPStatus.BAD_REQUEST,
117             detail="Invalid API Key",
118         )
119     if usernameExists(user.username)["Exists"]:
120         _userLogger.info(f"Username {user.username} already exists")
121         raise HTTPException(
122             status_code=HTTPStatus.CONFLICT,
123             detail="User already exists. A logged in user can delete
their account by a delete request to /user",
124         )
125     if not Helper.validate_password(user.password):
126         raise HTTPException(
127             status_code=HTTPStatus.BAD_REQUEST,
128             detail="This password does not meet the minimum
requirements.",
129         )
130     user.password = _hasher.hash(user.password)
131     users.create(
132         username=user.username,
133         password=user.password,
134         public_key=user.public_key,
135         api_key=user.api_key,
136     )
137     return {"detail": f"Successfully created {user.username}"}
138
139
140 # Delete User
141 @UserRouter.delete(
142     path="/user",
143     status_code=HTTPStatus.OK,
144     summary="Deletes a user.",
145     responses={
146         HTTPStatus.CONFLICT.value: {
147             "description": "Occurs if a username does not exists, or
they are not logged in.",
148             "model": Detail,
149             "content": {

```

```

150         "application/json": {
151             "example": {"detail": "The user Alex13 does not
exist"}
152         },
153     },
154 },
155     HTTPStatus.OK.value: {
156         "description": "This is returned on a successful
deletion.",
157         "model": Detail,
158         "content": {
159             "application/json": {
160                 "example": {"detail": "Successfully deleted
Alex13"}
161             },
162         },
163     },
164 },
165 )
166 def deleteUser(user: User):
167     _userLogger.info(f"Processing Delete:{user.username}")
168     query = users.objects(username=user.username)
169     if query.count() == 0:
170         _userLogger.info(
171             f"Username {user.username} does not exists to delete.
Raising Exception."
172         )
173         raise HTTPException(
174             status_code=HTTPStatus.CONFLICT,
175             detail=f"The user {user.username} does not exist or is
not logged in",
176         )
177     _userLogger.info(f"User Exists. Checking if logged in of {user.
username}")
178     db_user = query[0]
179     if not db_user.logged_in:
180         raise HTTPException(
181             status_code=HTTPStatus.CONFLICT,
182             detail=f"The user {user.username} does not exist or is
not logged in.",
183         )
184     try:
185         _hasher.verify(db_user.password, user.password)
186     except (VerificationError, VerifyMismatchError):
187         raise HTTPException(
188             status_code=HTTPStatus.CONFLICT,
189             detail=f"The user {user.username} does not exist or is
not logged in.",
190         )
191     users.delete(db_user)
192     _userLogger.info(f"Successfully deleted {user.username}")
193     return {"detail": f"Successfully deleted {user.username}"}
194
195

```

```

196 # Login User
197 @UserRouter.put(
198     path="/user",
199     status_code=HTTPStatus.OK,
200     summary="Login a user.",
201     responses={
202         HTTPStatus.BAD_REQUEST.value: {
203             "description": "Occurs on a invalid API Key or if
invalid information is sent.",
204             "model": Detail,
205             "content": {
206                 "application/json": {
207                     "example": {"detail": "Invalid Password or
username"}
208             },
209         },
210     },
211     HTTPStatus.OK.value: {
212         "description": "Returned on a successful login.",
213         "model": Detail,
214         "content": {
215             "application/json": {
216                 "example": {"detail": "Successful Login for
Alex13"}
217             },
218         },
219     },
220 },
221 )
222 def loginUser(request: Request, user: User):
223     _userLogger.info(f"Logging in {user.username} with key {user.
api_key}")
224     if not Helper.validate_APIKey(user.api_key):
225         raise HTTPException(
226             status_code=HTTPStatus.BAD_REQUEST, detail="Invalid API
Key"
227         )
228     query = users.objects(username=user.username)
229
230     if query.count() == 0:
231         raise HTTPException(
232             status_code=HTTPStatus.BAD_REQUEST,
233             detail="Invalid password or username",
234         )
235     db_user = query[0]
236     try:
237         _hasher.verify(db_user.password, user.password)
238     except (VerificationError, VerifyMismatchError):
239         raise HTTPException(
240             status_code=HTTPStatus.BAD_REQUEST,
241             detail="Invalid Password or username",
242         )
243     db_user.logged_in = request.client.host
244     db_user.save()

```

```
245     return {"detail": f"Successful Login for {user.username}"}
```

File 2.12: User.py

2.1.4 Reference client

gui.py

```

1 import base64
2 import json
3 import threading
4 import os
5 from tkinter import *
6
7 import websocket
8 from Crypto.Cipher import PKCS1_OAEP
9 from Crypto.PublicKey import RSA
10
11 import helpers
12 import gui_layouts
13
14 private_key, public_key = None, None
15 with open("otherPrK.pem") as f:
16     private_key = RSA.import_key(f.read())
17 with open("otherpK.pem") as f:
18     public_key = RSA.import_key(f.read())
19
20
21 encrypt_other = PKCS1_OAEP.new(public_key).encrypt
22 decrypt_other = PKCS1_OAEP.new(private_key).decrypt
23
24
25 def jsonify(m: str):
26     m_encrypted = encrypt_other(m.encode("utf-8"))
27     sending = base64.b64encode(m_encrypted).decode()
28     return json.dumps({"data": {"message": sending}})
29
30
31 class GUI:
32     def message(self, ws, m):
33         for decrypt in [self.decrypt_me, decrypt_other]:
34             try:
35                 received = base64.b64decode(m)
36                 m = decrypt(received).decode("utf-8")
37                 break
38             except Exception:
39                 pass
40         else:
41             m = self.msg
42             self.textCons.config(state=NORMAL)
43             self.textCons.insert(END, m + "\n\n")
44             self.textCons.config(state=DISABLED)
45             self.textCons.see(END)
46     def __init__(self):
47         self.Window = Tk()
48         self.Window.withdraw()
49         gui_layouts.login(self)

```



```

50
51     self.ws = websocket.WebSocketApp(
52         "ws://localhost:4000/ws/testroom",
53         on_message=self.message,
54     )
55     self.Window.mainloop()
56
57     def goAhead(self, name):
58         # ? This might not be needed
59         self.private_key, self.public_key = helpers.
60 get_personal_private_key(name)
61         self.encrypt_me = PKCS1_OAEP.new(public_key).encrypt
62         self.decrypt_me = PKCS1_OAEP.new(private_key).decrypt
63
64         self.login.destroy()
65         gui_layouts.chatroom(self, name)
66         self.run_thread = threading.Thread(target=self.run)
67         self.run_thread.start()
68
69     def run(self):
70         self.ws.run_forever()
71
72     def send_button(self, msg):
73         self.textCons.config(state=DISABLED)
74         self.entryMsg.delete(0, END)
75         self.msg = msg
76         send_thread = threading.Thread(
77             target=lambda x: self.ws.send(jsonify(x)), args=(msg,)
78         )
79         send_thread.start()
80
81 if not os.path.exists("users"):
82     os.mkdir("users")
83 g = GUI()

```

File 2.13: gui.py

gui_layouts.py

```

1 from tkinter import *
2 from tkinter import messagebox # ? Caused by __all__ ?
3 import helpers
4 import tkmacosx
5
6
7 def login(self):
8     self.login = Toplevel(bg="#1e1e1e")
9     self.login.title("Ilo Login")
10    self.login.resizable(width=False, height=False)
11    self.login.configure(width=505, height=365)
12    self.title_message = Label(
13        self.login,
14        text="Please login to continue",
15        justify=CENTER,
16        font="Helvetica 14 bold",
17        bg="#1e1e1e",
18        fg="#ffffff",
19    )
20    self.title_message.place(x=160, y=30, width=213, height=30)
21    self.label_name = Label(
22        self.login,
23        text="Username",
24        font="Helvetica 13",
25        bg="#1e1e1e",
26        fg="#ffffff",
27    )
28    self.label_name.place(x=220, y=90, width=70, height=25)
29    self.entry_name = Entry(
30        self.login,
31        font="Helvetica 14",
32        bg="#1e1e1e",
33        fg="#ffffff",
34        highlightcolor="#505050",
35        highlightbackground="#505050",
36        highlightthickness="0",
37    )
38    self.entry_name.place(x=150, y=120, width=210, height=37)
39    self.entry_name.focus()
40    self.entry_password = Entry(
41        self.login,
42        show="*",
43        font="Helvetica 14",
44        bg="#1e1e1e",
45        fg="#ffffff",
46        highlightcolor="#505050",
47        highlightbackground="#505050",
48        highlightthickness="0",
49    )
50    self.entry_password.place(x=150, y=230, width=210, height=37)
51    self.password_label = Label(
52        self.login,

```

```

53         text="Password",
54         font="Helvetica 13",
55         bg="#1e1e1e",
56         fg="#ffffff",
57     )
58     self.password_label.place(x=220, y=200, width=70, height=25)
59     self.login_button = tkmacosx.Button(
60         self.login,
61         text="Login",
62         command=lambda: self.goAhead(self.entry_name.get()),
63         borderless=1,
64         bg="#1e1e1e",
65         fg="#ffffff",
66     )
67     self.login_button.place(x=210, y=300, width=78, height=34)
68     self.register_button = tkmacosx.Button(
69         self.login,
70         text="Register",
71         borderless=1,
72         bg="#1e1e1e",
73         fg="#ffffff",
74         command=lambda: _register_command(self),
75     )
76     self.register_button.place(x=20, y=300, width=74, height=34)
77
78
79     def chatroom(self, name):
80         self.name = name
81         self.Window.deiconify()
82         self.Window.title("Ilo")
83         self.Window.resizable(width=False, height=False)
84         self.Window.configure(width=470, height=550, bg="#1e1e1e")
85         self.labelHead = Label(
86             self.Window,
87             bg="#1e1e1e",
88             fg="#ffffff",
89             text=self.name,
90             font="Helvetica 13 bold",
91             pady=12,
92         )
93         self.labelHead.place(relwidth=1)
94         self.textCons = Text(
95             self.Window,
96             width=20,
97             height=2,
98             bg="#1e1e1e",
99             fg="#ffffff",
100            font="Helvetica 14",
101            padx=5,
102            pady=5,
103            borderwidth=0,
104        )
105        self.textCons.place(relheight=0.745, relwidth=1, rely=0.08)
106        self.labelBottom = Label(self.Window, bg="#1e1e1e", height=80)

```

```

107     self.labelBottom.place(relwidth=1, rely=0.825)
108     self.entryMsg = Entry(
109         self.labelBottom,
110         bg="#1e1e1e",
111         fg="#ffffff",
112         font="Helvetica 13",
113         borderwidth=0,
114     )
115     self.entryMsg.place(relwidth=0.74, relheight=0.06, rely=0.008,
116 relx=0.011)
117     self.entryMsg.focus()
118     self.buttonMsg = tkmacosx.Button(
119         self.labelBottom,
120         text="Send",
121         font="Helvetica 12 bold",
122         width=20,
123         bg="#1e1e1e",
124         fg="#ffffff",
125         borderless=1,
126         command=lambda: self.send_button(self.entryMsg.get()),
127     )
128     self.newChat = tkmacosx.Button(
129         self.Window,
130         text="New Chatroom",
131         font="Helvetica 11 bold",
132         width=120,
133         bg="#1e1e1e",
134         fg="#ffffff",
135         borderless=1,
136         command=lambda: _popup(self),
137     )
138     self.newChat.place(x=10, y=10)
139     self.Window.bind("<Return>", lambda x: self.send_button(self.
140 entryMsg.get()))
141     self.buttonMsg.place(relx=0.77, rely=0.008, relheight=0.06,
142 relwidth=0.22)
143     self.textCons.config(cursor="arrow")
144     scrollbar = Scrollbar(self.textCons, bg="#1e1e1e")
145     scrollbar.place(relheight=1, relx=0.974)
146     scrollbar.config(command=self.textCons.yview)
147     self.textCons.config(state=DISABLED)
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

```
157 def _popup_cleanup(self):
158     name = self.e.get()
159     helpers.connect_new_chatroom(self, name)
160     self.popup.destroy()
161     del self.popup
162
163 def _register_command(self):
164     username: str = self.entry_name.get()
165     password: str = self.entry_password.get()
166     if helpers.register(username, password):
167         self.login.destroy()
168         chatroom(self, username)
169     else:
170         messagebox.showwarning(
171             "Ilo", "This username and password could not be
registered"
172         )
```

File 2.14: gui_layouts.py

helpers.py

```

1 import base64
2 import os
3 import threading
4 from tkinter.constants import END
5 from typing import Tuple
6
7 import requests as r
8 import websocket
9 from Crypto.Cipher import PKCS1_OAEP
10 from Crypto.PublicKey import RSA
11
12 ilo_user: str = "https://0.0.0.0:8000/user"
13 api_key: str = "399d79ac-8725-11eb-83c2-acde48001122"
14
15
16 def get_personal_private_key(username: str) -> Tuple[RSA.RsaKey, RSA
    .RsaKey]:
17     """Retrieves a private and public from disk for a specified
    username.
18
19     Args:
20         username (str): The username to get the private key of
21
22     Returns:
23         Tuple[RSA.RsaKey, RSA.RsaKey]: [Private Key, Public Key]
24     """
25     private_key = None
26     if not os.path.exists(f"users/{username}"):
27         os.mkdir(f"users/{username}")
28         private_key = RSA.generate(4096)
29         with open(f"users{os.sep}{username}{os.sep}private_key.pem",
30                 "w") as f:
31             f.write(private_key.export_key().decode("utf-8"))
32     elif os.path.isfile(f"users{os.sep}{username}{os.sep}private_key
    .pem"):
33         with open(f"users{os.sep}{username}{os.sep}private_key.pem",
34                 "r") as f:
35             private_key = RSA.import_key(f.read())
36     else:
37         private_key = RSA.generate(4096)
38         with open(f"users{os.sep}{username}{os.sep}private_key.pem",
39                 "w") as f:
40             f.write(private_key.export_key().decode("utf-8"))
41     return private_key, private_key.public_key()
42
43
44 def register(username: str, password: str) -> bool:
45     """
46     Registers with Ilo with the provided username and password.
47     The cryptographic keys will be created automatically
48
49     Args:

```

```

47     username (str): Username to register with
48     password (str): Password of the username
49
50 Returns:
51     bool: True or False indication the success of a registration
52     """
53     private_key = RSA.generate(4096)
54     public_key = private_key.public_key()
55     public_key_str: str = base64.b64encode(public_key.export_key("
DER")).decode()
56     response = r.post(
57         ilo_user,
58         json = {
59             "username": username,
60             "password": password,
61             "public_key": public_key_str,
62             "api_key": api_key,
63         },
64         verify=False,
65     )
66     if response.status_code == 201:
67         if not os.path.exists(f"users/{username}"):
68             os.mkdir(f"users/{username}")
69         with open(f"users{os.sep}{username}{os.sep}private_key.pem",
70 "w") as f:
71             f.write(private_key.export_key().decode("utf-8"))
72         return True
73     return False
74
75 def connect_new_chatroom(self, name: str):
76     self.ws.close()
77     self.textCons.delete(1.0,END)
78     self.ws = websocket.WebSocketApp(
79         f"ws://localhost:4000/ws/{name}", on_message=self.message
80     )
81     self.run_thread = threading.Thread(target=self.run)
82     self.run_thread.start()

```

File 2.15: helpers.py

2.2 Elixir

ilopotion.ex

```
1 defmodule Ilopotion do
2   use Application
3
4   def start(_type, _args) do
5     HTTPoison.start
6     children = [
7       Plug.Cowboy.child_spec(
8         scheme: :http,
9         plug: Ilopotion.Router,
10        options: [
11          dispatch: dispatch(),
12          port: 4000
13        ]
14      ),
15      Registry.child_spec(
16        keys: :duplicate,
17        name: Registry.Ilopotion
18      )
19    ]
20
21    opts = [strategy: :one_for_one, name: Ilopotion.Application]
22    Supervisor.start_link(children, opts)
23  end
24
25  defp dispatch do
26    [
27      {:_ ,
28       [
29         {"/ws/[...]", Ilopotion.SocketHandler, []},
30         {:_ , Plug.Cowboy.Handler, {Ilopotion.Router, []}}
31       ]
32     }
33  ]
34  end
35 end
```

File 2.16: ilopotion.ex

router.ex

```

1 require IEx
2
3 defmodule Ilopotion.SocketHandler do
4   @behaviour :cowboy_websocket
5
6   def init(request, _state) do
7     # IEx.pry
8     state = %{registry_key: request.path}
9     r = HTTPoison.post!("http://0.0.0.0:8000/potion", {:
multipart, [
10      {"address", elem(request.peer,0) |> Tuple.to_list |>
Enum.join(".")}]})
11     try do
12       if r.body == "false" do
13         Process.exit(self(), :kill)
14       end
15       catch
16         :throw, _ -> {IEx.pry, raise "Potion Failed to
reply"}
17     end
18     {:cowboy_websocket, request, state}
19   end
20
21   def websocket_init(state) do
22     IEx.pry
23     Registry.Ilopotion
24     |> Registry.register(state.registry_key, {})
25
26     {:ok, state}
27   end
28
29   def websocket_handle({:text, json}, state) do
30     payload = Jason.decode!(json)
31     message = payload["data"]["message"]
32
33     Registry.Ilopotion
34     |> Registry.dispatch(state.registry_key, fn(entries) ->
35       for {pid, _} <- entries do
36         if pid != self() do
37           Process.send(pid, message, [])
38         end
39       end
40     end
41   end
42 )
43
44   {:reply, {:text, message}, state}
45 end
46
47 def websocket_info(info, state) do
48   {:reply, {:text, info}, state}
49 end

```

50 `end`

File 2.17: router.ex

`socket_handler.ex`

File 2.18: `socket_handler.ex`

application.js

```

1 (( ) => {
2   class socket_handler {
3     setupSocket() {
4       this.socket = new WebSocket("ws://localhost:4000/ws/chat
5     ")
6
7     this.socket.addEventListener("message", (event) => {
8       const pTag = document.createElement("p")
9       pTag.innerHTML = event.data
10
11      document.getElementById("main").append(pTag)
12    })
13
14    this.socket.addEventListener("close", () => {
15      this.setupSocket()
16    })
17  }
18
19  submit(event) {
20    event.preventDefault()
21    const input = document.getElementById("message")
22    const message = input.value
23    input.value = ""
24
25    this.socket.send(
26      JSON.stringify({
27        data: { message: message },
28      })
29    )
30  }
31
32  const ws = new socket_handler()
33  ws.setupSocket()
34
35  document.getElementById("button")
36    .addEventListener("click", (event) => ws.submit(event))
37 })()

```

File 2.19: application.js

application.html.eex

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8"/>
5     <script type="application/javascript" defer src="js/application.
6       js"></script>
7     <title>
8       Ilo Potion
9     </title>
10  </head>
11  <body>
12    <main id="main"></main>
13    <br>
14    <form>
15      <input id="message" type="text">
16      <br>
17      <input type="submit" value="send" id="button">
18    </form>
19  </body>
20 </html>
```

File 2.20: application.html.eex

2.3 HTML/CSS/JavaScript

GenKeys.html

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="utf-8" />
5      <title>Ilo Potion</title>
6  </head>
7  <body>
8      <script>
9
10         window.crypto.subtle.generateKey({
11             name: "RSA-OAEP",
12             modulusLength: 4096,
13             publicExponent: new Uint8Array([1, 0, 1]),
14             hash: "SHA-256",
15         },
16         true,
17         ["encrypt", "decrypt"])
18         .then((keyPair) => {
19             const public_key = document.getElementById("public_key")
20         ;
21             window.crypto.subtle.exportKey("jwk",keyPair.publicKey).
22             then((key) =>{
23
24                 public_key.innerHTML = key;
25             });
26
27             const private_key = document.getElementById("private_key
28             ");
29             var key = window.crypto.subtle.exportKey("raw",keyPair.
30             privateKey);
31             debugger;
32             private_key.innerHTML = key;
33         });
34
35     </script>
36     <br>
37     <input id="message" type="text">
38     <br>
39     <br>
40     <h3>Public Key</h3>
41     <p id="public_key"></p>
42     <br><br>
43     <h3>Private Key</h3>
44     <p id="private_key"></p>
45 </body>
46 </html>

```

File 2.21: GenKeys.html

Login.html

```

1 <!DOCTYPE html>
2 <html>
3
4 <head>
5     <meta charset="utf-8" />
6     <script type="application/javascript" defer src="client/static/
    variables.js"></script>
7     <title>
8         Ilo Login
9     </title>
10 </head>
11 <script>
12     function sendJSON() {
13         let name = document.querySelector('#username').value;
14         let password = document.querySelector('#password').value;
15         let request = new XMLHttpRequest();
16         let url = "user";
17         request.open("POST", url, true);
18
19         request.setRequestHeader("Content-Type", "application/json")
20     ;
21
22     request.onreadystatechange = function () {
23         if (request.status === 200) {
24             window.location.replace('/')
25         }
26
27         // Converting JSON data to string
28         var data = JSON.stringify({
29             "username": name,
30             "password": password,
31             "apiKey": api_key,
32         });
33
34         // Sending data with the request
35         request.send(data);
36     }
37 </script>
38
39 <body>
40     <h2>Ilo Login</h2>
41     <form>
42         <input id="username" type="text">
43         <br>
44         <input id="password" type='password'>
45         <br>
46         <input type="submit" onclick="sendJSON()" value="send" id="
    button">
47     </form>
48 </body>
49

```

```
50 </html>
```

File 2.22: Login.html

sendMessage.html

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="utf-8" />
5   <title>Ilo Potion</title>
6 </head>
7 <body>
8   <!-- <script>
9     var ciphertext;
10
11     function get_message() {
12       var message = document.getElementById("message").value;
13       return new TextEncoder().encode(message);
14     }
15
16     async function encrypt_message(key)
17     {
18       var encoded = get_message();
19       ciphertext = await window.crypto.subtle.encrypt({
20         name: "RSA-OAEP"
21       },
22         key,
23         encoded
24       );
25       var cipher = new TextDecoder().decode(ciphertext);
26       console.log(cipher);
27       return cipher;
28     }
29
30     async function decrypt_message(key)
31     {
32       var decrypted = await window.crypto.subtle.decrypt({
33         name: "RSA-OAEP"
34       },
35         key,
36         ciphertext
37       );
38
39       var dec = new TextDecoder().decode(decrypted);
40       console.log(dec);
41       return dec;
42     }
43
44     window.crypto.subtle.generateKey({
45       name: "RSA-OAEP",
46       modulusLength: 4096,
47       publicExponent: new Uint8Array([1, 0, 1]),
48       hash: "SHA-256",
49     },
50     true,
51     ["encrypt", "decrypt"]
52   ).then((keyPair) => {

```

```
53     const submit_button = document.getElementById("e_button"  
54   );  
55     submit_button.addEventListener("click", () => {  
56       encrypt_message(keyPair.publicKey);  
57     });  
58     const decrypt_Button = document.getElementById("d_button"  
59   );  
60     decrypt_Button.addEventListener("click", () => {  
61       decrypt_message(keyPair.privateKey);  
62     });  
63  
64   </script> -->  
65   <br>  
66     <input id="message" type="text">  
67     <br>  
68     <br>  
69     <input type="submit" value="encrypt" id="e_button">  
70     <input type="submit" value="decrypt" id="d_button">  
71 </body>  
72 </html>
```

File 2.23: sendMessage.html

socket_code.js

```

1 (() => {
2   const private_key = "743677397
  A244326462948404D635166546A576E5A7234753778214125442A47" //in
  Hex
3   class myWebsocketHandler {
4     setupSocket() {
5       this.socket = new WebSocket("ws://localhost:4000/ws/chat
  ")
6
7       this.socket.addEventListener("message", (event) => {
8         const pTag = document.createElement("p")
9         pTag.innerHTML = event.data
10
11        document.getElementById("main").append(pTag)
12      })
13
14      this.socket.addEventListener("close", () => {
15        this.setupSocket()
16      })
17    }
18
19    submit(event) {
20      event.preventDefault()
21      const input = document.getElementById("message")
22      const message = input.value
23      input.value = ""
24
25      this.socket.send(
26        JSON.stringify({
27          data: { message: message },
28        })
29      )
30    }
31  }
32
33  const ws = new myWebsocketHandler()
34  ws.setupSocket()
35
36  document.getElementById("button")
37    .addEventListener("click", (event) => ws.submit(event))
38 })()

```

File 2.24: socket_code.js

2.4 Testing

conftest.py

```

1 from enum import auto
2 from unittest.mock import patch
3
4 import app
5 import pytest
6 import yaml
7 from cassandra.cqlengine import connection
8 from cassandra.cqlengine.management import sync_table
9 from CassandraModels import *
10 from fastapi.testclient import TestClient
11
12 from tests import mocks
13
14 config = yaml.safe_load(open("general_config.yaml", "r").read())
15
16
17 @pytest.fixture
18 def client():
19     return TestClient(app.app)
20
21
22 @pytest.fixture(scope="session", autouse=True)
23 def cassandra():
24     connection.setup(
25         [config["Cassandra_address"]], config["Cassandra_keyspace"],
26         protocol_version=3
27     )
28     sync_table(users, [config["Cassandra_keyspace"]])
29     sync_table(api_keys, [config["Cassandra_keyspace"]])
30
31 @pytest.fixture()
32 def apikey():
33     key = api_keys.create()
34     invalid_key = str(uuid.uuid4())
35     yield dict(
36         valid=str(key.key_id),
37         invalid=invalid_key,
38         invalid_correct_format=invalid_key,
39         invalid_incorrect_format="the-weft-and-weave-of-fate-guides"
40     )
41     try:
42         api_keys.delete(key)
43     except:
44         pass
45
46

```

```
47 # @pytest.fixture(scope="function")
48 # def db_cleanup():
49 #     to_delete = yield
50 #     for k, v in to_delete.items():
51 #         if k == "api_keys":
52 #             for key in v:
53 #                 api_keys.delete(key_id=key)
54
55
56 @pytest.fixture
57 def client_w_mock():
58     p = patch("app.open", new=mocks.MOCKED_open)
59     p.start()
60     import app
61
62     yield TestClient(app.app)
63     p.stop()
```

File 2.25: conftest.py

mocks.py

```
1 from unittest.mock import patch, mock_open
2
3
4 def MOCKED_open(filename, read_mode=None):
5     content = ""
6     if filename == "general_config.yaml":
7         # Address is null
8         # Missing MongoDB_user_collection &
9         MongoDB_apiKey_collection
10        content = ""
11 MongoDB_address: ""
12 MongoDB_database: ilo_mock
13 Potion_IP: " "
14 ""
15
16 elif filename == "x 2.txt":
17     content = "Mocking!\n"
18
19 else:
20     raise FileNotFoundError(filename)
21 file_object = mock_open(read_data=content).return_value
22 file_object.__iter__.return_value = content.splitlines(True)
23 return file_object
```

File 2.26: mocks.py

test_apikeys.py

```

1 from uuid import UUID
2
3 from cassandra.cqlengine.query import DoesNotExist
4 from CassandraModels import *
5 from pytest import fail, mark
6
7
8 @mark.apikeys
9 def test_create_key_format(client):
10     """Ensures a UUID is returned"""
11     response = client.post("/key/")
12     json = response.json()
13     key = json["detail"]
14     try:
15         UUID(key)
16     except ValueError:
17         fail("A none-UUID was returned")
18     try:
19         key = api_keys.get(key_id=key)
20     except DoesNotExist:
21         fail("Failed to write to database")
22
23
24 @mark.apikeys
25 def test_create_key_written_to_db(client):
26     """Ensures the key is written to the database"""
27     response = client.post("/key/")
28     json = response.json()
29     key = json["detail"]
30     try:
31         key = api_keys.get(key_id=key)
32     except DoesNotExist:
33         fail("Failed to write to database")
34
35
36 @mark.apikeys
37 def test_create_key_general(client):
38     """Ensures the returns are of the correct format"""
39     response = client.post("/key/")
40     assert response.status_code == 201
41     json = response.json()
42     assert "detail" in json
43
44     # db_cleanup.send(None)
45     # db_cleanup.send({"api_keys": [key]})
46
47
48 @mark.apikeys
49 def test_create_validate_UUID_version(client):
50     """Validates a UUID4 is used"""
51     response = client.post("/key/")
52     try:

```

```

53         assert (
54             UUID(response.json()["detail"]).version == 4
55         ), "4 was not the UUID version being used."
56     except ValueError:
57         fail("A none-UUID was returned")
58
59
60 @mark.apikeys
61 def test_delete_removed_from_database(client, apikey):
62     """Ensures a key is deleted from a database"""
63     response = client.delete(f"/key/{apikey['valid']}")
64     assert response.status_code == 200
65     json = response.json()
66     assert "detail" in json
67     assert f"Successfully deleted {str(apikey['valid'])}" in json["
68     detail"]
69     try:
70         key = api_keys.get(key_id=apikey["valid"])
71     except DoesNotExist:
72         return
73     fail("API Key was not removed from the database")
74
75 @mark.apikeys
76 def test_delete_invalid_key_incorrect_format(client):
77     """Sends a none-valid UUID formatted string"""
78     key = "Mittens"
79     response = client.delete(f"/key/{key}")
80     json = response.json()
81     assert response.status_code == 400
82     assert "detail" in json
83     assert json["detail"] == f"The key {key} is not a valid key"
84
85
86 @mark.apikeys
87 def test_delete_invalid_key_correct_format(client, apikey):
88     """Sends a valid UUID that does not exist as a key"""
89     response = client.delete(f"/key/{apikey['invalid_correct_format
90     ']}")
91     json = response.json()
92     assert response.status_code == 404
93     assert "detail" in json
94     assert (
95         json["detail"] == f"The key {apikey['invalid_correct_format
96         ']} does not exist"
97     )

```

File 2.27: test_apikeys.py

test_helper.py

```

1 import Helper
2 import pytest
3
4
5 @pytest.mark.helper_validate_keys
6 def test_valid_key(apikey):
7     """Ensure the helper returns True for a valid key"""
8     assert Helper.validate_APIKey(apikey["valid"]) == True
9
10
11 @pytest.mark.helper_validate_keys
12 def test_invalid_key(apikey):
13     """Ensure the helper returns false for an invalid key"""
14     assert Helper.validate_APIKey(apikey["invalid"]) == False
15
16
17 @pytest.mark.helper_validate_keys
18 def test_empty_key():
19     """Ensure the helper returns false for an empty key"""
20     assert Helper.validate_APIKey("") == False
21
22
23 @pytest.mark.helper_validate_keys
24 def test_invalid_format_key(apikey):
25     """Ensure the helper returns false for an incorrectly formatted
26     key"""
27     assert Helper.validate_APIKey(apikey["invalid_incorrect_format"
28 ]) == False
29
30
31 @pytest.mark.helper_validate_password
32 def test_correct_password():
33     """Ensure the helper returns True for a valid password"""
34     assert Helper.validate_password("Feathers_F@lling_On_Fresh_SnOw"
35 ) == True
36
37
38 @pytest.mark.helper_validate_password
39 def test_correct_no_minimum_length():
40     """Ensure the helper enforces length"""
41     assert Helper.validate_password("MOon!") == False
42
43
44 @pytest.mark.helper_validate_password
45 def test_correct_no_numbers():
46     """Ensure the helper enforced numbers"""
47     assert Helper.validate_password("Mooooooooooooooooon!") == False
48
49
50 @pytest.mark.helper_validate_password
51 def test_correct_no_lowercase():
52     """Ensure the helper enforces lowercase"""

```

```
50     assert Helper.validate_password("FEATHERS_F@LLING_ON_FRESH_SNOW"  
51     ) == False  
52  
53 @pytest.mark.helper_validate_password  
54 def test_correct_no_uppercase():  
55     """Ensure the helper enforces uppercase"""  
56     assert Helper.validate_password("feathers_f@lling_0n_fresh_sn0w"  
57     ) == False  
58  
59 @pytest.mark.helper_validate_password  
60 def test_correct_special_character():  
61     """Ensure the helper enforces the use of a special character"""  
62     assert Helper.validate_password("Feathers_Falling_0n_Fresh_Sn0w"  
63     ) == False
```

File 2.28: test_helper.py

test_potion.py

```

1 from routers.Potion import loggedIn
2 import Helper
3 import pytest
4 from Config import Config
5 from CassandraModels import users
6
7
8 @pytest.mark.helper_validate_keys
9 def test_blacklist_enforced(client, apikey):
10     """Ensure potion IP blacklisting is working as intended"""
11     response = client.post("/potion", data={"address": "127.0.0.1"})
12     assert response.status_code == 401
13
14
15 @pytest.mark.helper_validate_keys
16 def test_no_logged_in_users(client):
17     """Ensure FastAPI returns false for no logged in users"""
18     from cassandra.cluster import Cluster
19
20     c = Cluster()
21     s = c.connect("ilo")
22     s.execute("TRUNCATE users")
23     Config.Potion_IP = "testclient"
24     response = client.post("/potion", data={"address": "127.0.0.1"})
25     assert response.text == "false"
26
27
28 @pytest.mark.helper_validate_keys
29 def test_logged_in_users(client):
30     """Ensure the helper returns True for a valid key"""
31     user = users.create(
32         username="mittens",
33         password="password",
34         public_key="pk",
35         logged_in="127.0.0.1",
36         api_key="123-123-123",
37     )
38     Config.Potion_IP = "testclient"
39     response = client.post("/potion", data={"address": "127.0.0.1"})
40     assert response.text == "true"
41     users.delete(user)

```

File 2.29: test_potion.py

test_startup.py

```

1 from unittest.mock import patch
2
3 import pytest
4 from Config import Config
5
6 from tests import mocks
7
8
9 @pytest.mark.startup
10 def test_successful_startup(client):
11     """
12     Ensures documentation is available
13     """
14     response = client.get("/docs")
15     assert response.status_code == 200
16     assert "Ilo" in response.text
17
18
19 # @pytest.mark.startup
20 # def test_config_no_empty(client_w_mock):
21 #     config_vars = {key:value for key, value in Config.__dict__.
22 #         items() if not key.startswith('__') and not callable(key)}
23 #     assert all(i for i in config_vars.values())
24
25 # def test_config_uses_defaults(client_w_mock):
26 #     with open('general_config.yaml', 'r') as f:
27 #         s = f.read()
28 #         print(s)
29 #     config_vars = {key:value for key, value in Config.__dict__.
30 #         items() if not key.startswith('__') and not callable(key)}
31 #     assert config_vars["MongoDB_address"] == "mongodb
32 #         ://127.0.0.1:27017/"
33 #     assert config_vars["MongoDB_database"] == "ilo_mock"
34 #     assert config_vars["MongoDB_user_collection"] == "users"
35 #     assert config_vars["MongoDB_apiKey_collection"] == "api_keys"
36 #     assert config_vars["Potion_IP"] == "0.0.0.0:4000"

```

File 2.30: test_startup.py

test_user.py

```

1 import pytest
2 import json
3
4 public_key: str = "Ck1JSUpLUU1CQUFLQOFn (SHORTENED FOR BREVITY)"
5
6
7 @pytest.mark.user
8 def test_create_user_no_unicode(client, apikey):
9     """Create user without unicode"""
10    data = {
11        "username": "moonbeam",
12        "password": "FeathersFallingOnFr3shSn0w!",
13        "public_key": "public_key",
14        "api_key": apikey["valid"],
15    }
16    response = client.post("/user", json=data)
17    assert response.status_code == 201
18    response_json = response.json()
19    assert "detail" in response_json
20    assert "Successfully created moonbeam" == response_json["detail"]
21
22
23 @pytest.mark.user
24 def test_create_user_with_unicode_username(client):
25     """Create user with unicode"""
26    data = {
27        "username": "moonbëam",
28        "password": "FeathersFallingOnFr3shSn0w!",
29        "public_key": public_key,
30        "api_key": "abd5c02d-ccaf-435e-ac50-b0b459b4e328",
31    }
32    response = client.post("/user", json=data)
33    assert response.status_code == 201
34    response_json = response.json()
35    assert "detail" in response_json
36    assert "Successfully created moonbëam" == response_json["detail"]
37
38
39 @pytest.mark.user
40 def test_create_user_with_unicode_username_password(client):
41     """Create user with unicode"""
42    data = {
43        "username": "FastAPI",
44        "password": "FeathersFallingOnFr3shSn0w!",
45        "public_key": public_key,
46        "api_key": "abd5c02d-ccaf-435e-ac50-b0b459b4e328",
47    }
48    response = client.post("/user", json=data)
49    assert response.status_code == 201
50    response_json = response.json()

```

```
51     assert "detail" in response_json
52     assert "Successfully created FastAPI" == response_json["detail"]
```

File 2.31: test_user.py