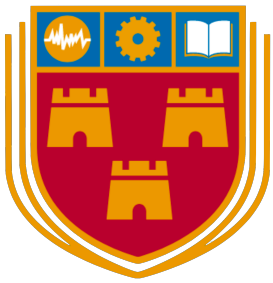
Secure Communication Platform
Technical Manual



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

List of Figures

List of files

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.

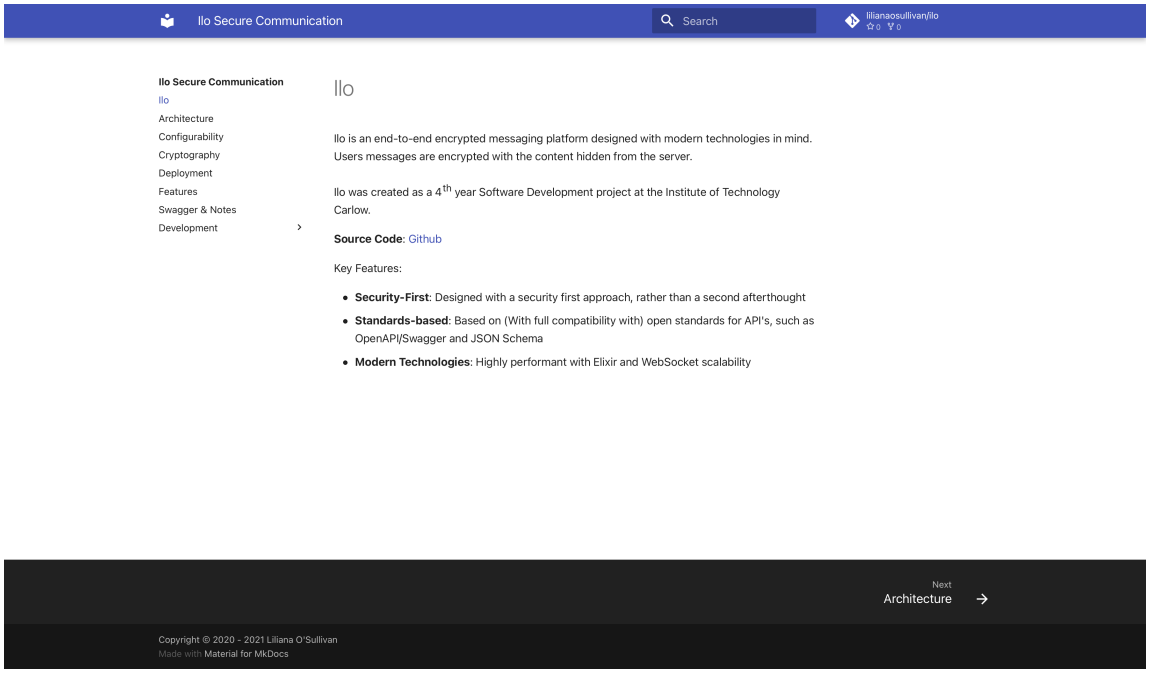


Figure 1.1: Created Developer Documentation

Additionally the content can be previewed as a PDF below.

Ilo Secure Communication

Security as a universal tool

Liliana O'Sullivan

Table of contents

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 4^{th} year Software Development project at the Institute of Technology Carlow.

Source Code: Github

Key Features:

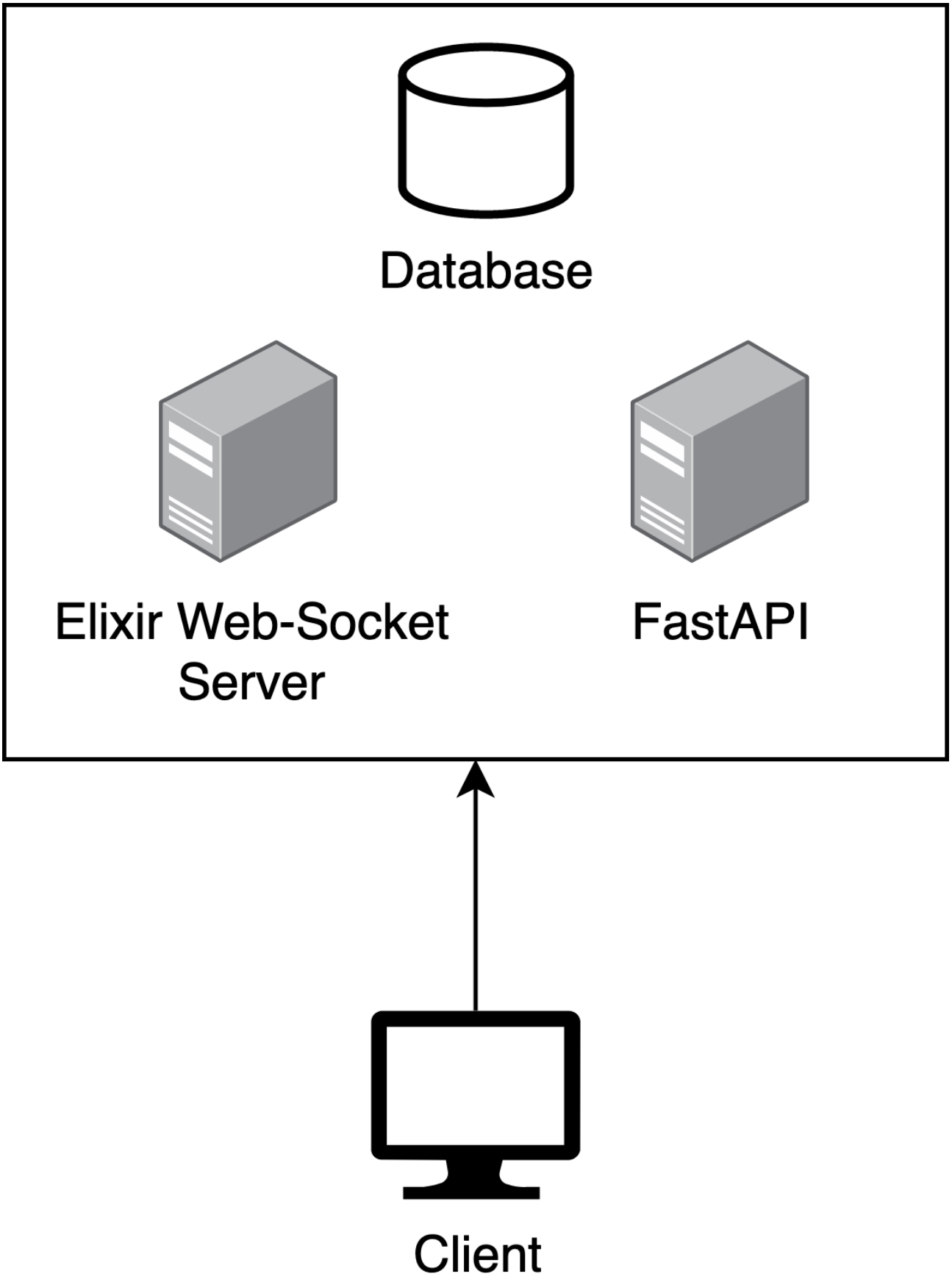
- **Security-First**: Designed with a security first approach, rather than a second afterthought
- **Standards-based**: Based on (With full compatibility with) open standards for API's, such as OpenAPI/Swagger and JSON Schema
- **Modern Technologies**: Highly performant with Elixir and WebSocket scalability

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.

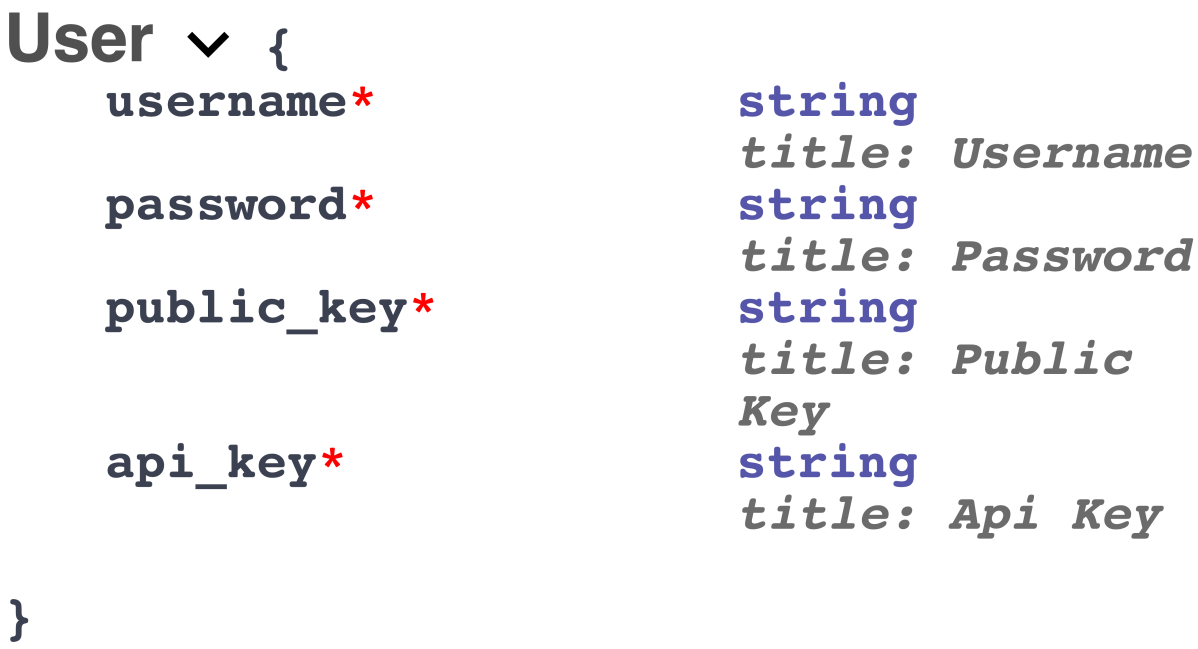
This will showcase the API made available through FastAPI. Additionally navigating to the `/openapi.json`, for example, `1.236.66.46:6921/openapi.json` will provide a JSON respecting the OpenAPI format of the public-facing functions. An excerpt is shown below


```
{
  "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



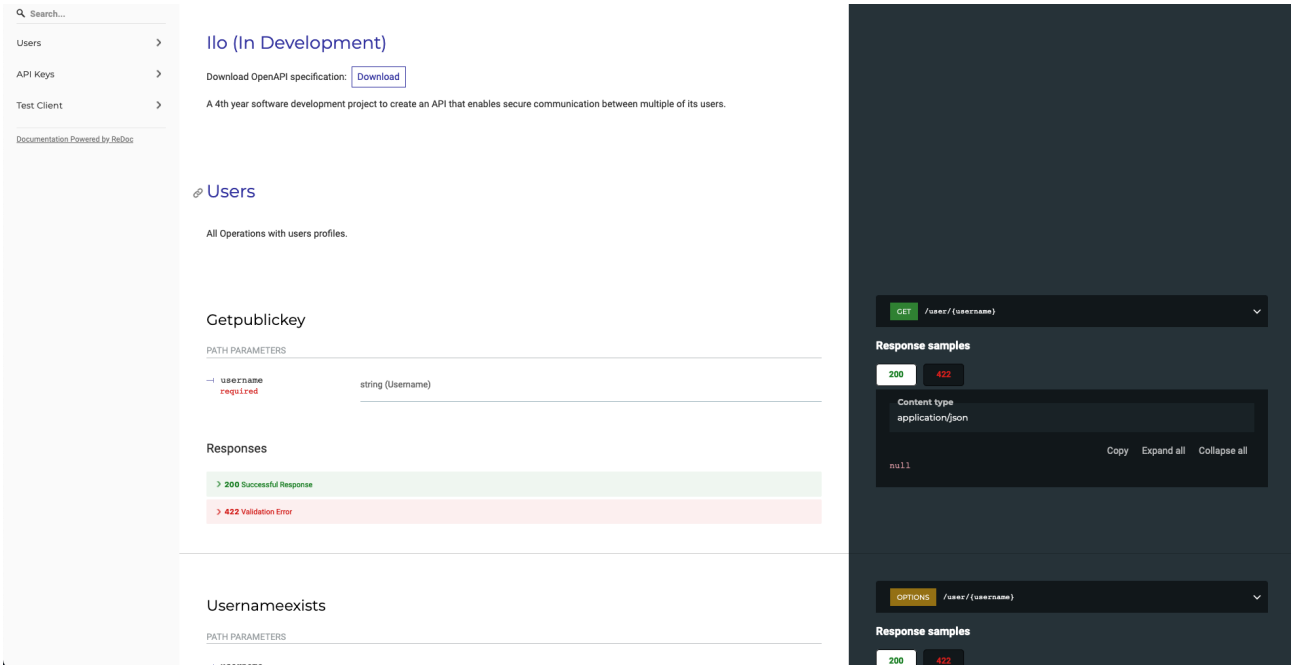
This schema in its' raw format from the OpenAPI JSON is shown below

```
"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`.



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 $_{u}$[?]$_{i}c_{o}d_{e}$ characters in sending messages and usernames. This allows none-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": "XzKSzgiX2qoPySbe5T4TSK2018V...",
  "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 website. Once downloaded and installed, the required Python libraries can be installed from PyPi using pip requirements

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.

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

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

```
fastapi
PyYAML
argon2-cffi
uvicorn
cassandra-driver
```

File 2.1: requirements.txt

2.1.2 FastAPI

app.py

```
import logging
import logging.config
import os
import sys
from typing import Dict

import yaml
from cassandra.cqlengine import connection
from cassandra.cqlengine.management import sync_table
from fastapi import FastAPI

import Helper
from CassandraModels import *
from Config import Config


def startup():

    import resource

    ## Fixes the issue "Too many open files error 24"
    ## https://stackoverflow.com/questions/2569620/socket-accept-error-24-to-many-open-files
    resource.setrlimit(resource.RLIMIT_NOFILE, (65536, 65536))

    if not os.path.exists("logging_config.yaml"):
        with open("logging_config.yaml", "w") as f:
            f.write(Config.LOGGING_CONFIG_DEFAULT)
            print("Created logging_config.yaml. Using default settings...")
            logging.config.dictConfig(yaml.safe_load(Config.LOGGING_CONFIG_DEFAULT))
    else:
        try:
            with open("logging_config.yaml", "r") as y:
                logging.config.dictConfig(yaml.safe_load(y.read()))
        except:
            print("Error Parsing logging_config. Using default Ilo configuration")
            logging.config.dictConfig(yaml.safe_load(Config.LOGGING_CONFIG_DEFAULT))

    _generalLogger = logging.getLogger("general")
    _generalLogger.info("Starting ilo...Parsing general config")
    if not os.path.exists("general_config.yaml"):
        with open("general_config.yaml", "w") as f:
            f.write(Config.GENERAL_CONFIG_DEFAULT)
            print("Created general_config.yaml. Using default 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
    browser",
93      },
94  ]
95  app = FastAPI(
96      title="Ilo",
97      description="A 4th year software development project to create
    an API that enables secure communication between multiple of its
     users.",
98      version="In Development",
99      openapi_tags=tags_metadata,
100 )
101
102 from routers.ApiKey import ApiKeyRouter
103 from routers.Potion import potionRouter
104 from routers.TestClient import TestClient
105 from routers.User import UserRouter
106
107 app.include_router(UserRouter)
108 app.include_router(ApiKeyRouter)
109 app.include_router(TestClient)
110 app.include_router(potionRouter)
111
112
113 if __name__ == "__main__":
114     import uvicorn
115
116     uvicorn.run(app, host="0.0.0.0", port=8000)
```

File 2.2: app.py

bash.sh

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

File 2.3: bash.sh

CassandraModels.py

```
import uuid
import time

from cassandra.cqlengine import columns
from cassandra.cqlengine.management import sync_table
from cassandra.cqlengine.models import Model


class users(Model):
    user_id    = columns.UUID(primary_key=True, default=uuid.uuid4)
    username   = columns.Text(required=True,index=True)
    password   = columns.Text(required=True)
    public_key = columns.Text(required=True)
    logged_in  = columns.Text(required=False,index=True)
    login_time = columns.Double(default=time.time)
    api_key    = columns.Text(required=False)

class api_keys(Model):
    key_id         = columns.UUID(primary_key=True, default=uuid.
    uuid4)
    creation_epoch = columns.Double(required=True)
```

File 2.4: CassandraModels

Config.py

```
from typing import Final


class Config:
    Cassandra_address: str = "127.0.0.1"
    Cassandra_keyspace: str = "ilo"
    Potion_IP: str = "0.0.0.0:4000"

    LOGGING_CONFIG_DEFAULT: Final = """version: 1
formatters:
    simple:
        format: "[%(name)s | %(levelname)s]:%(lineno)s - %(message)s
    "
    main:
        format: "[%(asctime)s]{%(name)s | %(levelname)s}:%(lineno)s
    - %(message)s"
handlers:
    console:
        class: logging.StreamHandler
        level: DEBUG
        formatter: main
        stream: ext://sys.stdout
    users:
        class: logging.handlers.RotatingFileHandler
        formatter: main
        filename: user.log
    general:
        class: logging.handlers.RotatingFileHandler
        formatter: main
        filename: general.log
    api:
        class: logging.handlers.RotatingFileHandler
        formatter: main
        filename: api.log
loggers:
    user:
        level: DEBUG
        handlers: [users]
    general:
        level: DEBUG
        handlers: [general]
    api:
        level: DEBUG
        handlers: [api]"""

    GENERAL_CONFIG_DEFAULT: Final = """Cassandra_address:
    "127.0.0.1"
Cassandra_keyspace: "ilo"
Potion_IP: "0.0.0.0:4000"
"""
```

File 2.5: Config.py

helper.py

```python
import logging
import re

from CassandraModels import api_keys

logger: logging.Logger = None


def validate_APIKey(
    key: str,
) -> bool:
    """
    Checks if an API Key is valid. Returns False if it's empty

    Parameters:
            key (str): The key to be validated

    Returns:
            bool: True if the key is valid, False if its invalid or empty.
    """
    if not (key := key.strip()):
        return False
    query = api_keys.objects(key_id=key)
    return False if query.count() == 0 else True


def validate_password(password: str) -> bool:
    """
    Validates that a password is meeting the minimum criteria, with confirmation.
    It must at minimum
        - Be of length 8
        - Contain minimum 1 number
        - Contain minimum 1 lowercase letter
        - Contain minimum 1 uppercase letter
        - 1 non-alphanumeric character

    Args:
        password (str): The password to validate

    Returns:
        bool: True if valid, False if invalid
    """
    if len(password) < 8:
        return False
    if re.search("[0-9]+", password) == None:  # Number
        return False
    if re.search("[a-z]+", password) == None:  # Lowercase
        return False
    if re.search("[A-Z]+", password) == None:  # Uppercase
        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

```yaml
version: 1
formatters:
    simple:
        format: "[%(name)s | %(levelname)s]:%(lineno)s - %(message)s
    "
    main:
        format: "[%(asctime)s]{%(name)s | %(levelname)s}:%(lineno)s
    - %(message)s"
handlers:
    console:
        class: logging.StreamHandler
        level: DEBUG
        formatter: main
        stream: ext://sys.stdout
    users:
        class: logging.handlers.RotatingFileHandler
        formatter: main
        filename: user.log
    general:
        class: logging.handlers.RotatingFileHandler
        formatter: main
        filename: general.log
    api:
        class: logging.handlers.RotatingFileHandler
        formatter: main
        filename: api.log
loggers:
    user:
        level: DEBUG
        handlers: [users]
    general:
        level: DEBUG
        handlers: [general]
    api:
        level: DEBUG
        handlers: [api]
```

File 2.7: logging config.yaml

models.py

```python
from pydantic import BaseModel


class User(BaseModel):
    username: str
    password: str
    public_key: str
    api_key: str


class Detail(BaseModel):
    detail: str = ""
```

File 2.8: models.py

2.1.3 FastAPI - Routers

ApiKey.py

```
import logging
import time
from http import HTTPStatus
from cassandra.cqlengine.query import DoesNotExist


from fastapi.exceptions import HTTPException
from fastapi.routing import APIRouter
from models import Detail
from CassandraModels import *
from cassandra.cqlengine import ValidationError

# from cassandra.cqlengine.models import DoesNotExist

ApiKeyRouter: APIRouter = APIRouter(tags=["API Keys"])
_apiLogger: logging.Logger = logging.getLogger("api")

# Create API Key
@ApiKeyRouter.post(
    path="/key/",
    status_code=HTTPStatus.CREATED,
    summary="Create a API Key",
    responses={
        HTTPStatus.CREATED.value: {
            "description": "201 response is sent on a successful creation of a API Key.",
            "model": Detail,
            "content": {
                "application/json": {
                    "example": {"detail": "b24aj62cb-1625-4ab5-212b-aah08cxc9a"}
                },
            },
        },
    },
)
def createKey():
    key = api_keys.create(creation_epoch=time.time())
    return {"detail": key.key_id}


@ApiKeyRouter.delete(
    path="/key/{key}",
    status_code=HTTPStatus.OK,
    summary="Delete an API Key",
    responses={
        HTTPStatus.BAD_REQUEST.value: {
            "description": "Occurs on a invalid key submitted",
            "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 formated 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

```
from CassandraModels import users
from Config import Config
from fastapi import Form, HTTPException, Request
from fastapi.routing import APIRouter
from http import HTTPStatus

potionRouter = APIRouter()


@potionRouter.post("/potion", include_in_schema=False)
def loggedIn(request: Request, address: str = Form("address")):
    """Returns True or False depending on a users logged in state

    Args:
        username (str): Username to be checked

    Returns:
        bool: True if logged in, False if logged out.
    """
    if not request.client.host == Config.Potion_IP:
        raise HTTPException(status_code=HTTPStatus.UNAUTHORIZED)
    query = users.objects(logged_in=request.client.host)
    return True if query.count() > 0 else False
```

File 2.10: Potion.py

TestClient.py

```python
from fastapi.responses import HTMLResponse
from fastapi.routing import APIRouter

TestClient: APIRouter = APIRouter(
    default_response_class=HTMLResponse, tags=["Test Client"]
)


@TestClient.get(path="/")
def home():
    with open("client/login.html") as f:
        return f.read()


@TestClient.get(path="/message")
def message():
    with open("client/sendMessage.html") as f:
        return f.read()

@TestClient.get(path="/cryptodemo")
def demo():
    with open("client/SubtleCryptoDemo.html") as f:
        return f.read()

@TestClient.get(path="/genkey")
def genkeys():
    with open("client/GenKeys.html") as f:
        return f.read()


@TestClient.get(
    path="/client/static/socket_code.js",
    include_in_schema=False,
)
def jsFile():
    with open("client/static/socket_code.js") as f:
        return f.read()
```

File 2.11: TestClient.py

User.py

```
import logging
from http import HTTPStatus

import Helper
from argon2 import PasswordHasher
from argon2.exceptions import VerificationError, VerifyMismatchError
from CassandraModels import users
from fastapi import APIRouter, HTTPException, Request
from models import Detail, User

UserRouter: APIRouter = APIRouter(tags=["Users"])

_hasher = PasswordHasher()
_userLogger: logging.Logger = logging.getLogger("user")


@UserRouter.options(
    path="/user/{username}",
    status_code=HTTPStatus.OK,
    summary="Checks if a username exists.",
    responses={
        HTTPStatus.OK.value: {
            "description": "Returned as a success if the username exists or not.",
            "model": Detail,
            "content": {
                "application/json": {"example": {"detail": "true"}},
            },
        },
    },
)
def usernameExists(username: str):
    query = users.objects(username=username)
    if query.count() > 0:
        return {"detail": "true"}
    return {"detail": "false"}


@UserRouter.get(
    path="/user/{username}",
    status_code=HTTPStatus.OK,
    summary="Get the users public key.",
    responses={
        HTTPStatus.CONFLICT.value: {
            "description": "Occurs if a username does not exists, or they are not logged in.",
            "model": Detail,
            "content": {
                "application/json": {
                    "example": {"detail": "The user Alex13 does not exist"}
                },
```

```
            },
        },
        HTTPStatus.OK.value: {
            "description": "This is returned with the public key.",
            "model": Detail,
            "content": {
                "application/json": {
                    "example": {"detail": "
    rbdm4YHtR4d8ykwZ8DxL0SHDUQJ"}
                },
            },
        },
    },
)
def getPublicKey(username: str):
    query = users.objects(username=username)
    if query.count() == 0:
        raise HTTPException(
            status_code=HTTPStatus.CONFLICT,
            detail=f"The user {username} does not exist",
        )
    return {"detail": query[0].public_key}


# Create User
@UserRouter.post(
    path="/user",
    status_code=HTTPStatus.CREATED,
    summary="Creates a user.",
    responses={
        HTTPStatus.CONFLICT.value: {
            "description": "Occurs if a user already exists.",
            "model": Detail,
            "content": {
                "application/json": {
                    "example": {
                        "detail": "User already exists. A logged in
    user can delete their account by a delete request to /user"
                    }
                },
            },
        },
        HTTPStatus.BAD_GATEWAY.value: {
            "description": "Occurs if the password does not meet
    minimum requirements.",
            "model": Detail,
            "content": {
                "application/json": {
                    "example": {
                        "detail": "This password does not meet the
    minimum requirements."
                    }
                },
            },
```

```
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

```
import base64
import json
import threading
import os
from tkinter import *

import websocket
from Crypto.Cipher import PKCS1_OAEP
from Crypto.PublicKey import RSA

import helpers
import gui_layouts

private_key, public_key = None, None
with open("otherPrK.pem") as f:
    private_key = RSA.import_key(f.read())
with open("otherpK.pem") as f:
    public_key = RSA.import_key(f.read())


encrypt_other = PKCS1_OAEP.new(public_key).encrypt
decrypt_other = PKCS1_OAEP.new(private_key).decrypt


def jsonify(m: str):
    m_encrypted = encrypt_other(m.encode("utf-8"))
    sending = base64.b64encode(m_encrypted).decode()
    return json.dumps({"data": {"message": sending}})


class GUI:
    def message(self,ws, m):
        for decrypt in [self.decrypt_me, decrypt_other]:
            try:
                received = base64.b64decode(m)
                m = decrypt(received).decode("utf-8")
                break
            except Exception:
                pass
        else:
            m = self.msg
        self.textCons.config(state=NORMAL)
        self.textCons.insert(END, m + "\n\n")
        self.textCons.config(state=DISABLED)
        self.textCons.see(END)
    def __init__(self):
        self.Window = Tk()
        self.Window.withdraw()
        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.
    get_personal_private_key(name)
60          self.encrypt_me = PKCS1_OAEP.new(public_key).encrypt
61          self.decrypt_me = PKCS1_OAEP.new(private_key).decrypt
62
63          self.login.destroy()
64          gui_layouts.chatroom(self, name)
65          self.run_thread = threading.Thread(target=self.run)
66          self.run_thread.start()
67
68      def run(self):
69          self.ws.run_forever()
70
71      def send_button(self, msg):
72          self.textCons.config(state=DISABLED)
73          self.entryMsg.delete(0, END)
74          self.msg = msg
75          send_thread = threading.Thread(
76              target=lambda x: self.ws.send(jsonify(x)), args=(msg,)
77          )
78          send_thread.start()
79
80
81  if not os.path.exists("users"):
82      os.mkdir("users")
83  g = GUI()
```

File 2.13: gui.py

gui_layouts.py

```
from tkinter import *
from tkinter import messagebox  # ? Caused by __all__ ?
import helpers
import tkmacosx


def login(self):
    self.login = Toplevel(bg="#1e1e1e")
    self.login.title("Ilo Login")
    self.login.resizable(width=False, height=False)
    self.login.configure(width=505, height=365)
    self.title_message = Label(
        self.login,
        text="Please login to continue",
        justify=CENTER,
        font="Helvetica 14 bold",
        bg="#1e1e1e",
        fg="#ffffff",
    )
    self.title_message.place(x=160, y=30, width=213, height=30)
    self.label_name = Label(
        self.login,
        text="Username",
        font="Helvetica 13",
        bg="#1e1e1e",
        fg="#ffffff",
    )
    self.label_name.place(x=220, y=90, width=70, height=25)
    self.entry_name = Entry(
        self.login,
        font="Helvetica 14",
        bg="#1e1e1e",
        fg="#ffffff",
        highlightcolor="#505050",
        highlightbackground="#505050",
        highlightthickness="0",
    )
    self.entry_name.place(x=150, y=120, width=210, height=37)
    self.entry_name.focus()
    self.entry_password = Entry(
        self.login,
        show="*",
        font="Helvetica 14",
        bg="#1e1e1e",
        fg="#ffffff",
        highlightcolor="#505050",
        highlightbackground="#505050",
        highlightthickness="0",
    )
    self.entry_password.place(x=150, y=230, width=210, height=37)
    self.password_label = Label(
        self.login,
```

```
        text="Password",
        font="Helvetica 13",
        bg="#1e1e1e",
        fg="#ffffff",
    )
    self.password_label.place(x=220, y=200, width=70, height=25)
    self.login_button = tkmacosx.Button(
        self.login,
        text="Login",
        command=lambda: self.goAhead(self.entry_name.get()),
        borderless=1,
        bg="#1e1e1e",
        fg="#ffffff",
    )
    self.login_button.place(x=210, y=300, width=78, height=34)
    self.register_button = tkmacosx.Button(
        self.login,
        text="Register",
        borderless=1,
        bg="#1e1e1e",
        fg="#ffffff",
        command=lambda: _register_command(self),
    )
    self.register_button.place(x=20, y=300, width=74, height=34)


def chatroom(self, name):
    self.name = name
    self.Window.deiconify()
    self.Window.title("Ilo")
    self.Window.resizable(width=False, height=False)
    self.Window.configure(width=470, height=550, bg="#1e1e1e")
    self.labelHead = Label(
        self.Window,
        bg="#1e1e1e",
        fg="#ffffff",
        text=self.name,
        font="Helvetica 13 bold",
        pady=12,
    )
    self.labelHead.place(relwidth=1)
    self.textCons = Text(
        self.Window,
        width=20,
        height=2,
        bg="#1e1e1e",
        fg="#ffffff",
        font="Helvetica 14",
        padx=5,
        pady=5,
        borderwidth=0,
    )
    self.textCons.place(relheight=0.745, relwidth=1, rely=0.08)
    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,
    relx=0.011)
116     self.entryMsg.focus()
117     self.buttonMsg = tkmacosx.Button(
118         self.labelBottom,
119         text="Send",
120         font="Helvetica 12 bold",
121         width=20,
122         bg="#1e1e1e",
123         fg="#ffffff",
124         borderless=1,
125         command=lambda: self.send_button(self.entryMsg.get()),
126     )
127     self.newChat = tkmacosx.Button(
128         self.Window,
129         text="New Chatroom",
130         font="Helvetica 11 bold",
131         width=120,
132         bg="#1e1e1e",
133         fg="#ffffff",
134         borderless=1,
135         command=lambda: _popup(self),
136     )
137     self.newChat.place(x=10, y=10)
138     self.Window.bind("<Return>", lambda x: self.send_button(self.
    entryMsg.get()))
139     self.buttonMsg.place(relx=0.77, rely=0.008, relheight=0.06,
    relwidth=0.22)
140     self.textCons.config(cursor="arrow")
141     scrollbar = Scrollbar(self.textCons, bg="#1e1e1e")
142     scrollbar.place(relheight=1, relx=0.974)
143     scrollbar.config(command=self.textCons.yview)
144     self.textCons.config(state=DISABLED)
145
146
147 def _popup(self):
148     self.popup = Toplevel(self.Window)
149     self.l = Label(self.popup,text="Chatroom name")
150     self.l.pack()
151     self.e = Entry(self.popup)
152     self.e.pack()
153     self.b=tkmacosx.Button(self.popup,text="Connect",command=lambda:
    _popup_cleanup(self))
154     self.b.pack()
155     self.Window.wait_window(self.popup)
156
```

```
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

```
import base64
import os
import threading
from tkinter.constants import END
from typing import Tuple

import requests as r
import websocket
from Crypto.Cipher import PKCS1_OAEP
from Crypto.PublicKey import RSA

ilo_user: str = "https://0.0.0.0:8000/user"
api_key: str = "399d79ac-8725-11eb-83c2-acde48001122"


def get_personal_private_key(username: str) -> Tuple[RSA.RsaKey, RSA.RsaKey]:
    """Retrieves a private and public from disk for a specified username.

    Args:
        username (str): The username to get the private key of

    Returns:
        Tuple[RSA.RsaKey, RSA.RsaKey]: [Private Key, Public Key]
    """
    private_key = None
    if not os.path.exists(f"users/{username}"):
        os.mkdir(f"users/{username}")
        private_key = RSA.generate(4096)
        with open(f"users{os.sep}{username}{os.sep}private_key.pem", "w") as f:
            f.write(private_key.export_key().decode("utf-8"))
    elif os.path.isfile(f"users{os.sep}{username}{os.sep}private_key.pem"):
        with open(f"users{os.sep}{username}{os.sep}private_key.pem", "r") as f:
            private_key = RSA.import_key(f.read())
    else:
        private_key = RSA.generate(4096)
        with open(f"users{os.sep}{username}{os.sep}private_key.pem", "w") as f:
            f.write(private_key.export_key().decode("utf-8"))
    return private_key, private_key.public_key()


def register(username: str, password: str) -> bool:
    """
    Registers with Ilo with the provided username and password.
    The cryptographic keys will be created automatically

    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",
    "w") as f:
70              f.write(private_key.export_key().decode("utf-8"))
71          return True
72      return False
73
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

```
defmodule Ilopotion do
  use Application

  def start(_type, _args) do
    HTTPoison.start
    children = [
      Plug.Cowboy.child_spec(
        scheme: :http,
        plug: Ilopotion.Router,
        options: [
          dispatch: dispatch(),
          port: 4000
        ]
      ),
      Registry.child_spec(
        keys: :duplicate,
        name: Registry.Ilopotion
      )
    ]

    opts = [strategy: :one_for_one, name: Ilopotion.Application]
    Supervisor.start_link(children, opts)
  end

  defp dispatch do
    [
      {:_,
        [
          {"/ws/[...]", Ilopotion.SocketHandler, []},
          {:_, Plug.Cowboy.Handler, {Ilopotion.Router, []}}
        ]
      }
    ]
  end
end
```

File 2.16: ilopotion.ex

router.ex

```
require IEx

defmodule Ilopotion.SocketHandler do
    @behaviour :cowboy_websocket

    def init(request, _state) do
      # IEx.pry
      state = %{registry_key: request.path}
            r = HTTPoison.post!('http://0.0.0.0:8000/potion', {:multipart, [
              {"address", elem(request.peer,0) |> Tuple.to_list |> Enum.join(".")}
            ]})
            try do
                if r.body == "false" do
                    Process.exit(self(),:kill)
                end
                catch
                    :throw,_ -> {IEx.pry,raise "Potion Failed to reply"}
            end
      {:cowboy_websocket, request, state}
    end

    def websocket_init(state) do
        IEx.pry
        Registry.Ilopotion
        |> Registry.register(state.registry_key, {})

      {:ok, state}
    end

    def websocket_handle({:text, json}, state) do
      payload = Jason.decode!(json)
      message = payload["data"]["message"]

      Registry.Ilopotion
      |> Registry.dispatch( state.registry_key, fn(entries) ->
          for {pid, _} <- entries do
            if pid != self() do
              Process.send(pid, message, [])
            end
          end
        end
      )

      {:reply, {:text, message}, state}
    end

    def websocket_info(info, state) do
      {:reply, {:text, info}, state}
    end
```

```
50    end
```

File 2.17: router.ex

socket_handler.ex

File 2.18: socket_handler.ex

application.js

```
(() => {
    class socket_handler {
        setupSocket() {
            this.socket = new WebSocket("ws://localhost:4000/ws/chat
    ")

            this.socket.addEventListener("message", (event) => {
                const pTag = document.createElement("p")
                pTag.innerHTML = event.data

                document.getElementById("main").append(pTag)
            })

            this.socket.addEventListener("close", () => {
                this.setupSocket()
            })
        }

        submit(event) {
            event.preventDefault()
            const input = document.getElementById("message")
            const message = input.value
            input.value = ""

            this.socket.send(
                JSON.stringify({
                    data: { message: message },
                })
            )
        }
    }

    const ws = new socket_handler()
    ws.setupSocket()

    document.getElementById("button")
        .addEventListener("click", (event) => ws.submit(event))
})()
```

File 2.19: application.js

application.html.eex

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8"/>
    <script type="application/javascript" defer src="js/application.
    js"></script>
    <title>
      Ilo Potion
    </title>
  </head>
  <body>
    <main id="main"></main>
    <br>
    <form>
      <input id="message" type="text">
      <br>
      <br>
      <input type="submit" value="send" id="button">
    </form>
  </body>
</html>
```

File 2.20: application.html.eex

2.3 HTML/CSS/JavaScript

GenKeys.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Ilo Potion</title>
</head>
<body>
    <script>

        window.crypto.subtle.generateKey({
            name: "RSA-OAEP",
            modulusLength: 4096,
            publicExponent: new Uint8Array([1, 0, 1]),
            hash: "SHA-256",
        },
        true,
        ["encrypt", "decrypt"]
        ).then((keyPair) => {
            const public_key = document.getElementById("public_key")
    ;
            window.crypto.subtle.exportKey("jwk",keyPair.publicKey).
    then((key) =>{

                public_key.innerHTML = key;
            });

            const private_key = document.getElementById("private_key
    ");
            var key = window.crypto.subtle.exportKey("raw",keyPair.
    privateKey);
            debugger;
            private_key.innerHTML = key;
        });

    </script>
    <br>
    <input id="message" type="text">
    <br>
    <br>
    <h3>Public Key</h3>
    <p id="public_key"></p>
    <br><br>
    <h3>Private Key</h3>
    <p id="private_key"></p>
</body>
</html>
```

File 2.21: GenKeys.html

Login.html

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8" />
    <script type="application/javascript" defer src="client/static/variables.js"></script>
    <title>
        Ilo Login
    </title>
</head>
<script>
    function sendJSON() {
        let name = document.querySelector('#username').value;
        let password = document.querySelector('#password').value;
        let request = new XMLHttpRequest();
        let url = "user";
        request.open("POST", url, true);

        request.setRequestHeader("Content-Type", "application/json");

        request.onreadystatechange = function () {
            if (request.status === 200) {
                window.location.replace('/')
            }

            // Converting JSON data to string
            var data = JSON.stringify({
                "username": name,
                "password": password,
                "apiKey": api_key,
            });

            // Sending data with the request
            request.send(data);
        }
    }
</script>

<body>
    <h2>Ilo Login</h2>
    <form>
        <input id="username" type="text">
        <br>
        <input id="password" type='password'>
        <br>
        <input type="submit" onclick="sendJSON()" value="send" id="button">
    </form>
</body>

```

```
50 </html>
```

File 2.22: Login.html

sendMessage.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Ilo Potion</title>
</head>
<body>
    <!-- <script>
        var ciphertext;

        function get_message() {
            var message = document.getElementById("message").value;
            return new TextEncoder().encode(message);
        }

        async function encrypt_message(key)
        {
            var encoded = get_message();
            ciphertext = await window.crypto.subtle.encrypt({
                    name: "RSA-OAEP"
                },
                key,
                encoded
            );
            var cipher = new TextDecoder().decode(ciphertext);
            console.log(cipher);
            return cipher;
        }

        async function decrypt_message(key)
        {
            var decrypted = await window.crypto.subtle.decrypt({
                    name: "RSA-OAEP"
                },
                key,
                ciphertext
            );

            var dec = new TextDecoder().decode(decrypted);
            console.log(dec);
            return dec;
        }

        window.crypto.subtle.generateKey({
            name: "RSA-OAEP",
            modulusLength: 4096,
            publicExponent: new Uint8Array([1, 0, 1]),
            hash: "SHA-256",
        },
        true,
        ["encrypt", "decrypt"]
        ).then((keyPair) => {
```

```
53              const submit_button = document.getElementById("e_button"
    );
54              submit_button.addEventListener("click", () => {
55                  encrypt_message(keyPair.publicKey);
56              });
57
58              const decrypt_Button = document.getElementById("d_button
    ");
59              decrypt_Button.addEventListener("click", () => {
60                  decrypt_message(keyPair.privateKey);
61              });
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

```
(() => {
    const private_key = "743677397
    A2443264629484O4D635166546A576E5A72347537782141254 42A47" //in
    Hex
    class myWebsocketHandler {
        setupSocket() {
            this.socket = new WebSocket("ws://localhost:4000/ws/chat
    ")

            this.socket.addEventListener("message", (event) => {
                const pTag = document.createElement("p")
                pTag.innerHTML = event.data

                document.getElementById("main").append(pTag)
            })

            this.socket.addEventListener("close", () => {
                this.setupSocket()
            })
        }

        submit(event) {
            event.preventDefault()
            const input = document.getElementById("message")
            const message = input.value
            input.value = ""

            this.socket.send(
                JSON.stringify({
                    data: { message: message },
                })
            )
        }
    }

    const ws = new myWebsocketHandler()
    ws.setupSocket()

    document.getElementById("button")
        .addEventListener("click", (event) => ws.submit(event))
})()
```

File 2.24: socket_code.js

2.4 Testing

conftest.py

```
from enum import auto
from unittest.mock import patch

import app
import pytest
import yaml
from cassandra.cqlengine import connection
from cassandra.cqlengine.management import sync_table
from CassandraModels import *
from fastapi.testclient import TestClient

from tests import mocks

config = yaml.safe_load(open("general_config.yaml", "r").read())


@pytest.fixture
def client():
    return TestClient(app.app)


@pytest.fixture(scope="session", autouse=True)
def cassandra():
    connection.setup(
        [config["Cassandra_address"]], config["Cassandra_keyspace"],
    protocol_version=3
    )
    sync_table(users, [config["Cassandra_keyspace"]])
    sync_table(api_keys, [config["Cassandra_keyspace"]])


@pytest.fixture()
def apikey():
    key = api_keys.create()
    invalid_key = str(uuid.uuid4())
    yield dict(
        valid=str(key.key_id),
        invalid=invalid_key,
        invalid_correct_format=invalid_key,
        invalid_incorrect_format="the-weft-and-weave-of-fate-guides"
    ,
    )
    try:
        api_keys.delete(key)
    except:
        pass


```

```
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

```
from unittest.mock import patch, mock_open


def MOCKED_open(filename, read_mode=None):
    content = ""
    if filename == "general_config.yaml":
        # Address is null
        # Missing MongoDB_user_collection &
   MongoDB_apiKey_collection
        content = """
MongoDB_address: ""
MongoDB_database: ilo_mock
Potion_IP: "   "
"""
    elif filename == "x 2.txt":
        content = "Mocking!\n"
    else:
        raise FileNotFoundError(filename)
    file_object = mock_open(read_data=content).return_value
    file_object.__iter__.return_value = content.splitlines(True)
    return file_object
```

File 2.26: mocks.py

test_apikeys.py

```
from uuid import UUID

from cassandra.cqlengine.query import DoesNotExist
from CassandraModels import *
from pytest import fail, mark


@mark.apikeys
def test_create_key_format(client):
    """Ensures a UUID is returned"""
    response = client.post("/key/")
    json = response.json()
    key = json["detail"]
    try:
        UUID(key)
    except ValueError:
        fail("A none-UUID was returned")
    try:
        key = api_keys.get(key_id=key)
    except DoesNotExist:
        fail("Failed to write to database")


@mark.apikeys
def test_create_key_written_to_db(client):
    """Ensures the key is written to the database"""
    response = client.post("/key/")
    json = response.json()
    key = json["detail"]
    try:
        key = api_keys.get(key_id=key)
    except DoesNotExist:
        fail("Failed to write to database")


@mark.apikeys
def test_create_key_general(client):
    """Ensures the returns are of the correct format"""
    response = client.post("/key/")
    assert response.status_code == 201
    json = response.json()
    assert "detail" in json

    # db_cleanup.send(None)
    # db_cleanup.send({"api_keys":[key]})


@mark.apikeys
def test_create_validate_UUID_version(client):
    """Validates a UUID4 is used"""
    response = client.post("/key/")
    try:
```

```
        assert (
            UUID(response.json()["detail"]).version == 4
        ), "4 was not the UUID version being used."
    except ValueError:
        fail("A none-UUID was returned")


@mark.apikeys
def test_delete_removed_from_database(client, apikey):
    """Ensures a key is deleted from a database"""
    response = client.delete(f"/key/{apikey['valid']}")
    assert response.status_code == 200
    json = response.json()
    assert "detail" in json
    assert f"Successfully deleted {str(apikey['valid'])}" in json["detail"]
    try:
        key = api_keys.get(key_id=apikey["valid"])
    except DoesNotExist:
        return
    fail("API Key was not removed from the database")


@mark.apikeys
def test_delete_invalid_key_incorrect_format(client):
    """Sends a none-valid UUID formatted string"""
    key = "Mittens"
    response = client.delete(f"/key/{key}")
    json = response.json()
    assert response.status_code == 400
    assert "detail" in json
    assert json["detail"] == f"The key {key} is not a valid key"


@mark.apikeys
def test_delete_invalid_key_correct_format(client, apikey):
    """Sends a valid UUID that does not exist as a key"""
    response = client.delete(f"/key/{apikey['invalid_correct_format']}")
    json = response.json()
    assert response.status_code == 404
    assert "detail" in json
    assert (
        json["detail"] == f"The key {apikey['invalid_correct_format']} does not exist"
    )
```

File 2.27: test_apikeys.py

test_helper.py

```
import Helper
import pytest


@pytest.mark.helper_validate_keys
def test_valid_key(apikey):
    """Ensure the helper returns True for a valid key"""
    assert Helper.validate_APIKey(apikey["valid"]) == True


@pytest.mark.helper_validate_keys
def test_invalid_key(apikey):
    """Ensure the helper returns false for an invalid key"""
    assert Helper.validate_APIKey(apikey["invalid"]) == False


@pytest.mark.helper_validate_keys
def test_empty_key():
    """Ensure the helper returns false for an empty key"""
    assert Helper.validate_APIKey("") == False


@pytest.mark.helper_validate_keys
def test_invalid_format_key(apikey):
    """Ensure the helper returns false for an incorrectly formatted key"""
    assert Helper.validate_APIKey(apikey["invalid_incorrect_format"]) == False


@pytest.mark.helper_validate_password
def test_correct_password():
    """Ensure the helper returns True for a valid password"""
    assert Helper.validate_password("Feathers_F@lling_0n_Fresh_Sn0w") == True


@pytest.mark.helper_validate_password
def test_correct_no_minimum_length():
    """Ensure the helper enforces length"""
    assert Helper.validate_password("M0on!") == False


@pytest.mark.helper_validate_password
def test_correct_no_numbers():
    """Ensure the helper enforced numbers"""
    assert Helper.validate_password("Mooooooooooooon!") == False


@pytest.mark.helper_validate_password
def test_correct_no_lowercase():
    """Ensure the helper enforces lowercase"""
```

```
50      assert Helper.validate_password("FEATHERS_F@LLING_0N_FRESH_SN0W"
        ) == False
51
52
53  @pytest.mark.helper_validate_password
54  def test_correct_no_uppercase():
55      """Ensure the helper encforces uppercase"""
56      assert Helper.validate_password("feathers_f@lling_0n_fresh_sn0w"
        ) == False
57
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"
        ) == False
```

File 2.28: test_helper.py

test_potion.py

```python
from routers.Potion import loggedIn
import Helper
import pytest
from Config import Config
from CassandraModels import users


@pytest.mark.helper_validate_keys
def test_blacklist_enforced(client, apikey):
    """Ensure potion IP blacklisting is working as intended"""
    response = client.post("/potion", data={"address": "127.0.0.1"})
    assert response.status_code == 401


@pytest.mark.helper_validate_keys
def test_no_logged_in_users(client):
    """Ensure FastAPI returns false for no logged in users"""
    from cassandra.cluster import Cluster

    c = Cluster()
    s = c.connect("ilo")
    s.execute("TRUNCATE users")
    Config.Potion_IP = "testclient"
    response = client.post("/potion", data={"address": "127.0.0.1"})
    assert response.text == "false"


@pytest.mark.helper_validate_keys
def test_logged_in_users(client):
    """Ensure the helper returns True for a valid key"""
    user = users.create(
        username="mittens",
        password="password",
        public_key="pk",
        logged_in="127.0.0.1",
        api_key="123-123-123",
    )
    Config.Potion_IP = "testclient"
    response = client.post("/potion", data={"address": "127.0.0.1"})
    assert response.text == "true"
    users.delete(user)
```

File 2.29: test_potion.py

test_startup.py

```
from unittest.mock import patch

import pytest
from Config import Config

from tests import mocks


@pytest.mark.startup
def test_successful_startup(client):
    """
    Ensures documentation is available
    """
    response = client.get("/docs")
    assert response.status_code == 200
    assert "Ilo" in response.text


# @pytest.mark.startup
# def test_config_no_empty(client_w_mock):
#     config_vars = {key:value for key, value in Config.__dict__.items() if not key.startswith('__') and not callable(key)}
#     assert all(i for i in config_vars.values())

# def test_config_uses_defaults(client_w_mock):
#     with open('general_config.yaml', 'r') as f:
#         s = f.read()
#         print(s)
#     config_vars = {key:value for key, value in Config.__dict__.items() if not key.startswith('__') and not callable(key)}
#     assert config_vars["MongoDB_address"] == "mongodb://127.0.0.1:27017/"
#     assert config_vars["MongoDB_database"] == "ilo_mock"
#     assert config_vars["MongoDB_user_collection"] == "users"
#     assert config_vars["MongoDB_apiKey_collection"] == "api_keys"
#     assert config_vars["Potion_IP"] == "0.0.0.0:4000"
```

File 2.30: test_startup.py

test_user.py

```python
import pytest
import json

public_key: str = "Ck1JSUpLUUlCQUFLQOFn (SHORTENED FOR BREVITY)"


@pytest.mark.user
def test_create_user_no_unicode(client, apikey):
    """Create user without unicode"""
    data = {
        "username": "moonbeam",
        "password": "FeathersFallingOnFr3shSn0w!",
        "public_key": "public_key",
        "api_key": apikey["valid"],
    }
    response = client.post("/user", json=data)
    assert response.status_code == 201
    response_json = response.json()
    assert "detail" in response_json
    assert "Successfully created moonbeam" == response_json["detail"]


@pytest.mark.user
def test_create_user_with_unicode_username(client):
    """Create user with unicode"""
    data = {
        "username": "moonbëam",
        "password": "FeathersFallingOnFr3shSn0w!",
        "public_key": public_key,
        "api_key": "abd5c02d-ccaf-435e-ac50-b0b459b4e328",
    }
    response = client.post("/user", json=data)
    assert response.status_code == 201
    response_json = response.json()
    assert "detail" in response_json
    assert "Successfully created moonbëam" == response_json["detail"]


@pytest.mark.user
def test_create_user_with_unicode_username_password(client):
    """Create user with unicode"""
    data = {
        "username": "FastAPI",
        "password": "FeathersFallingOnFr3shSn0w!",
        "public_key": public_key,
        "api_key": "abd5c02d-ccaf-435e-ac50-b0b459b4e328",
    }
    response = client.post("/user", json=data)
    assert response.status_code == 201
    response_json = response.json()
```

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

File 2.31: test_user.py