Peter Hamilton

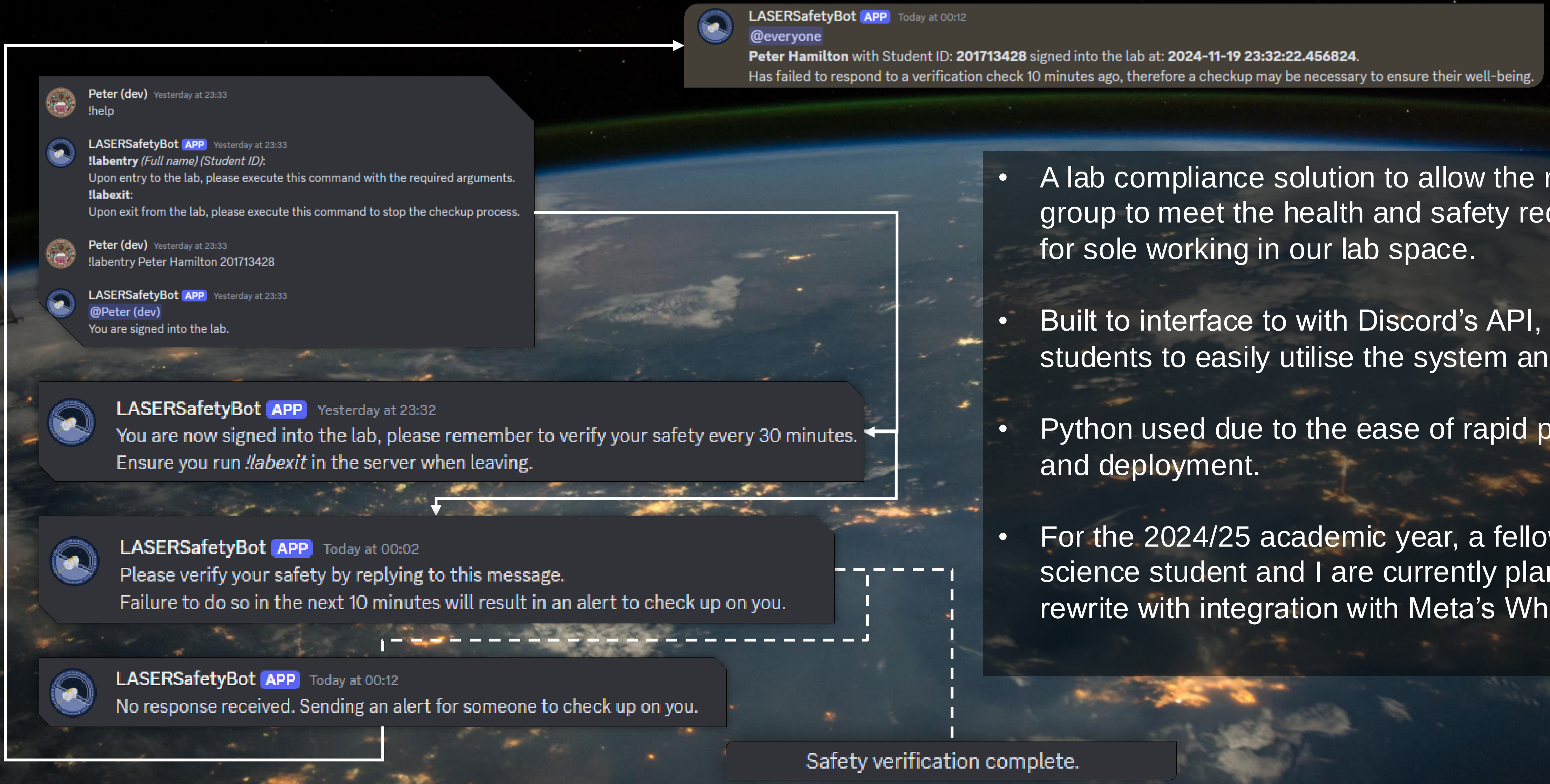
LASER Safety Compliance Service

About me:

- *Second year **computer scientist** at The University of Liverpool.*
- *Currently dedicate significant time outside my studies to, maintaining and developing software and solutions for LASER (Liverpool Association for space engineering and research).*
- *Hobbies include: Climbing, maintaining a home-lab and flight simulation.*

Contents:

Project overview:



Peter (dev) Yesterday at 23:33
!help

LASERSafetyBot APP Yesterday at 23:33
!labentry *(Full name) (Student ID)*:
Upon entry to the lab, please execute this command with the required arguments.
!labexit:
Upon exit from the lab, please execute this command to stop the checkup process.

Peter (dev) Yesterday at 23:33
!labentry Peter Hamilton 201713428

LASERSafetyBot APP Yesterday at 23:33
@Peter (dev)
You are signed into the lab.

LASERSafetyBot APP Yesterday at 23:32
You are now signed into the lab, please remember to verify your safety every 30 minutes.
Ensure you run *!labexit* in the server when leaving.

LASERSafetyBot APP Today at 00:02
Please verify your safety by replying to this message.
Failure to do so in the next 10 minutes will result in an alert to check up on you.

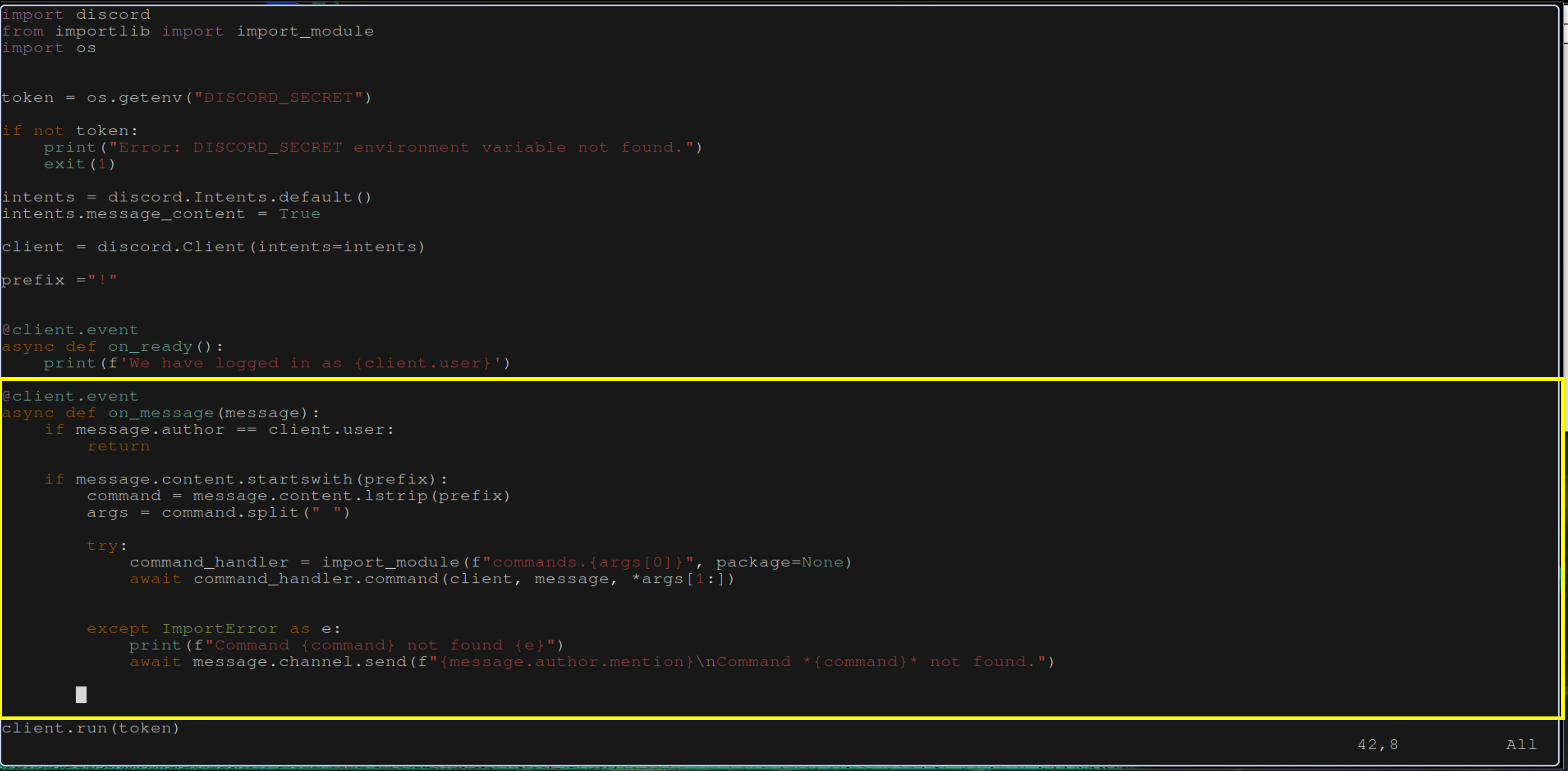
LASERSafetyBot APP Today at 00:12
No response received. Sending an alert for someone to check up on you.

Safety verification complete.

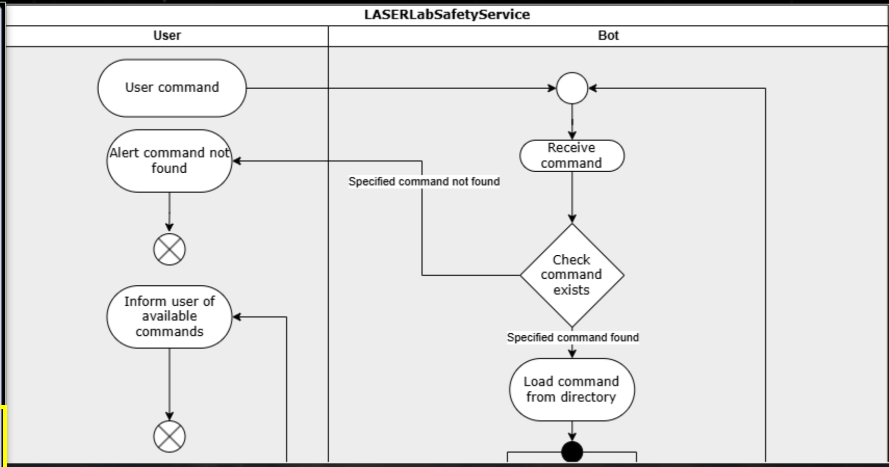
LASERSafetyBot APP Today at 00:12
@everyone
Peter Hamilton with Student ID: **201713428** signed into the lab at: **2024-11-19 23:32:22.456824**.
Has failed to respond to a verification check 10 minutes ago, therefore a checkup may be necessary to ensure their well-being.

- A lab compliance solution to allow the research group to meet the health and safety requirements for sole working in our lab space.
- Built to interface to with Discord’s API, allowing students to easily utilise the system and login.
- Python used due to the ease of rapid prototyping and deployment.
- For the 2024/25 academic year, a fellow computer science student and I are currently planning on a rewrite with integration with Meta’s WhatsApp API.

Main:



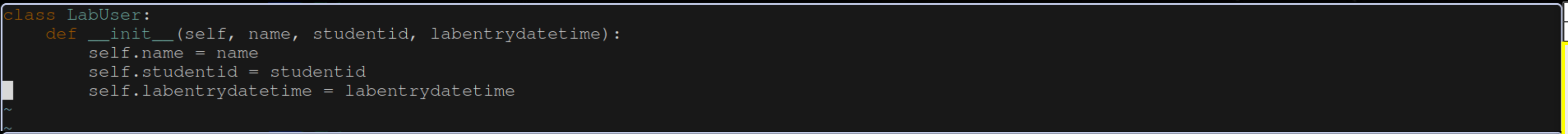
```
import discord
from importlib import import_module
import os


token = os.getenv("DISCORD_SECRET")

if not token:
    print("Error: DISCORD_SECRET environment variable not found.")
    exit(1)

intents = discord.Intents.default()
intents.message_content = True

client = discord.Client(intents=intents)

prefix ="!"


@client.event
async def on_ready():
    print(f'We have logged in as {client.user}')

@client.event
async def on_message(message):
    if message.author == client.user:
        return

    if message.content.startswith(prefix):
        command = message.content.lstrip(prefix)
        args = command.split(" ")

        try:
            command_handler = import_module(f"commands.{args[0]}", package=None)
            await command_handler.command(client, message, *args[1:])


        except ImportError as e:
            print(f"Command {command} not found {e}")
            await message.channel.send(f"{message.author.mention}\nCommand *{command}* not found.")

client.run(token)
                                                                          42,8          All
```



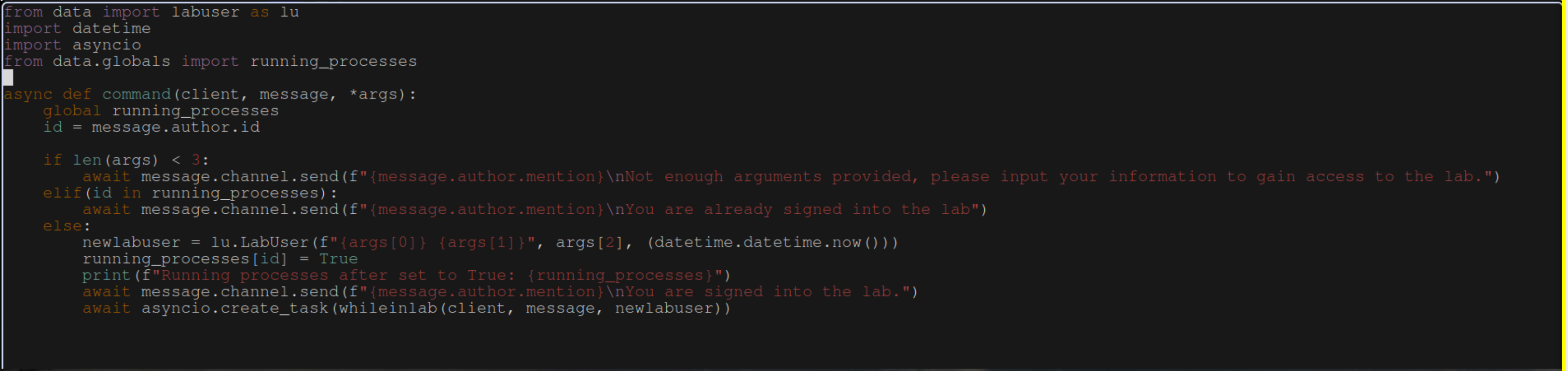
- ***on_message*** triggers whenever a message is sent in a message channel the system has access to.
- System checks that the message is not from itself.

- Checks for messages with command prefix, splits message into command and arguments and dynamically loads a matching python module with the same name.
- Handles unknown commands, sending an error message back to the user.
- This modularity allows for ease of integration for future modules or commands if needed.

Lab entry process:



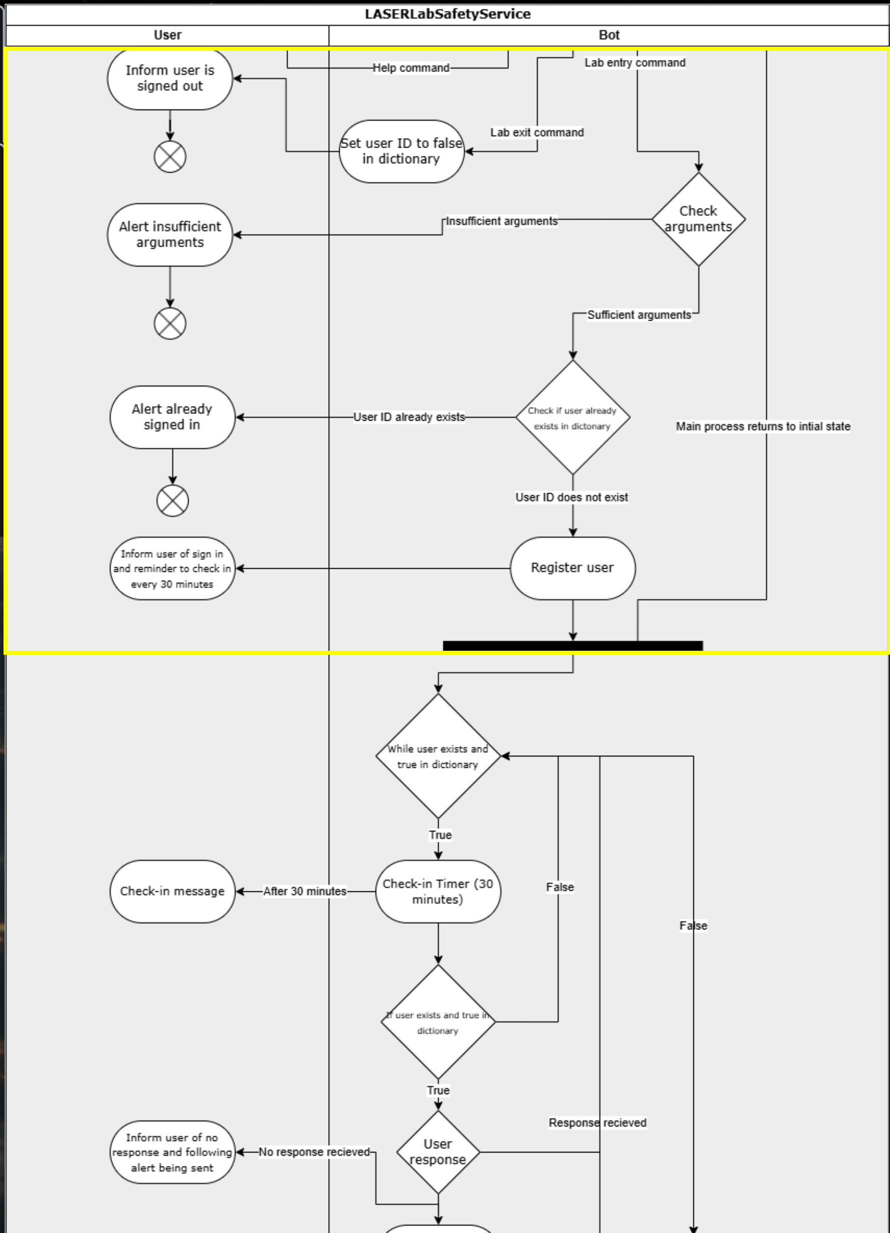
```
class LabUser:
    def __init__(self, name, studentid, labentrydatetime):
        self.name = name
        self.studentid = studentid
        self.labentrydatetime = labentrydatetime
```



```
from data import labuser as lu
import datetime
import asyncio
from data.globals import running_processes

async def command(client, message, *args):
    global running_processes
    id = message.author.id

    if len(args) < 3:
        await message.channel.send(f"{message.author.mention}\nNot enough arguments provided, please input your information to gain access to the lab.")
    elif(id in running_processes):
        await message.channel.send(f"{message.author.mention}\nYou are already signed into the lab")
    else:
        newlabuser = lu.LabUser(f"{args[0]} {args[1]}", args[2], (datetime.datetime.now()))
        running_processes[id] = True
        print(f"Running processes after set to True: {running_processes}")
        await message.channel.send(f"{message.author.mention}\nYou are signed into the lab.")
        await asyncio.create_task(whileinlab(client, message, newlabuser))
```

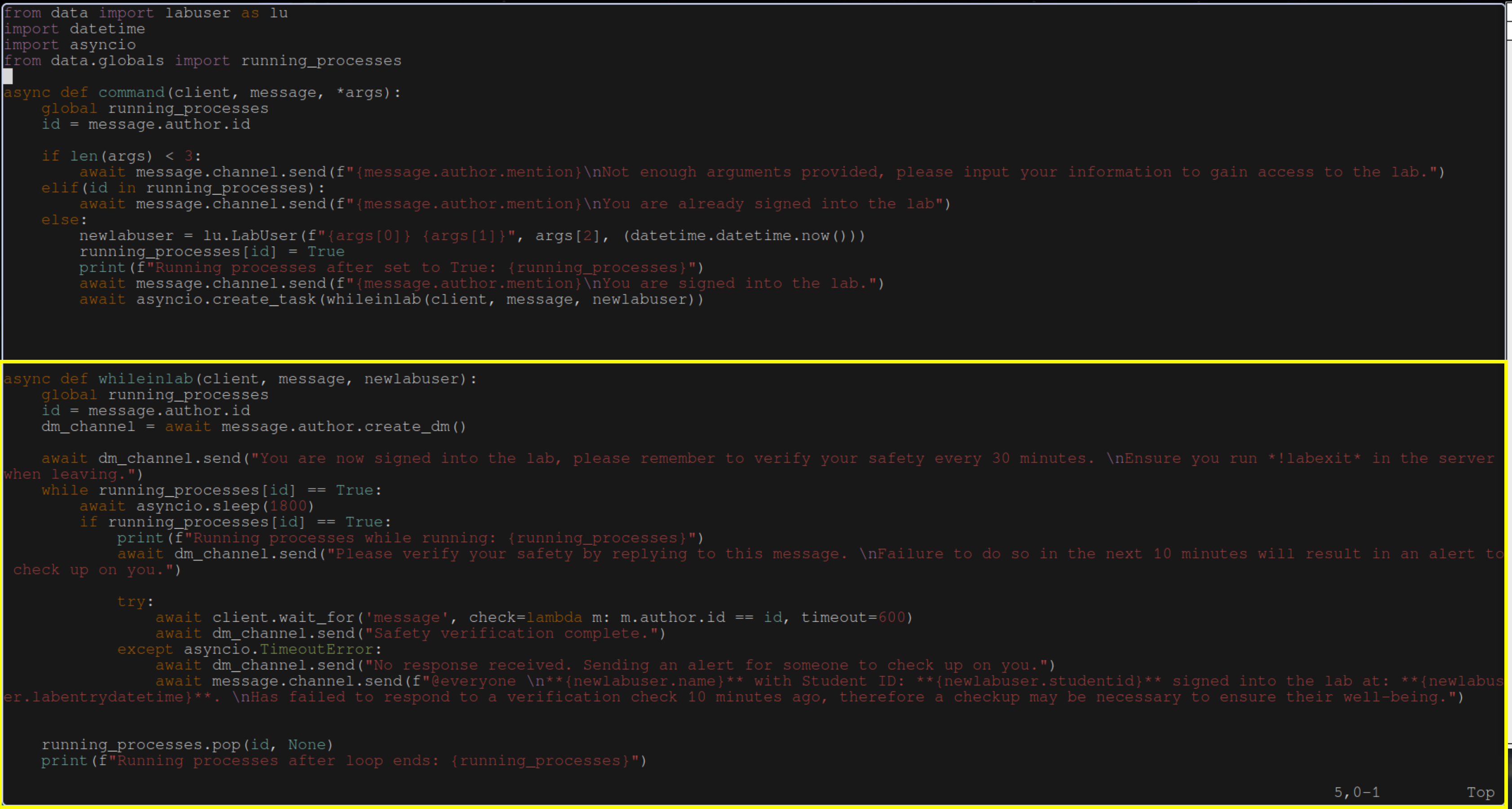
Handles the ***!labentry*** command, allowing users to sign into the system. Before handing off checks for user safety to a concurrent task.

- Ensures valid arguments have been provided and if the user is already signed in.
- A new ***labuser*** object is created, containing user details and time logged in.
- Adds user id to global ***running_processes*** dictionary to track who is signed in.
- Utilises ***asyncio.create_task()*** spawn a concurrent task per user, which enables the system to remain responsive to commands.



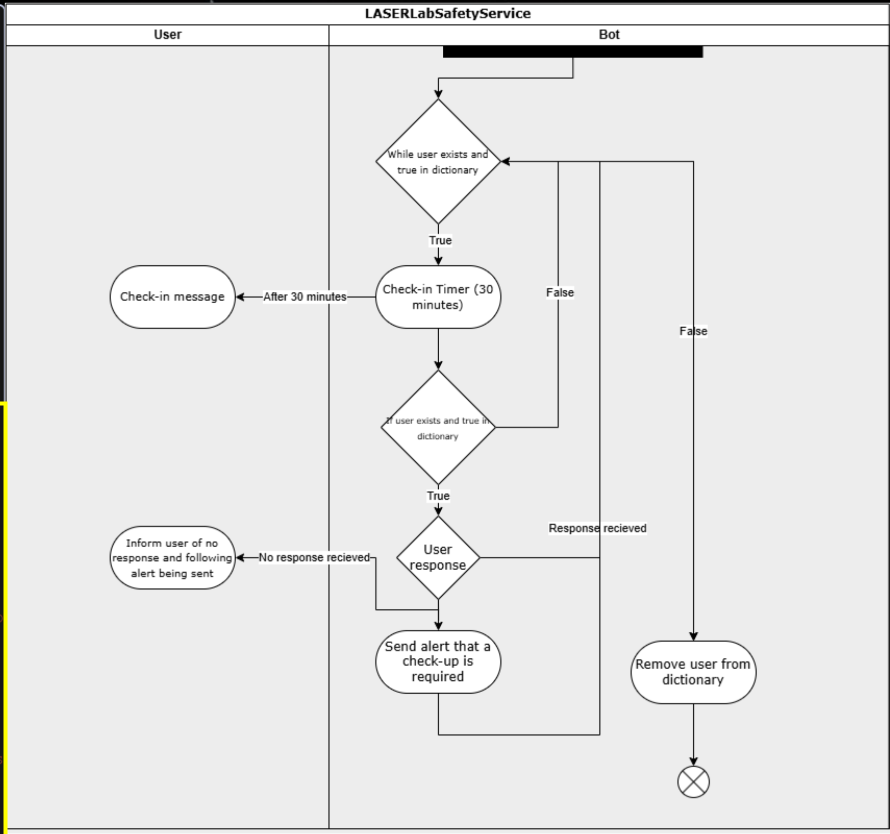
LASERLabSafetyService
User
Bot
Help command
Lab entry command
Inform user is signed out
Set user ID to false in dictionary
Lab exit command
Check arguments
Insufficient arguments
Alert insufficient arguments
Sufficient arguments
Check if user already exists in dictonary
User ID already exists
Alert already signed in
Main process returns to intial state
User ID does not exist
Register user
Inform user of sign in and reminder to check in every 30 minutes
While user exists and true in dictionary
True
Check-in Timer (30 minutes)
After 30 minutes
Check-in message
False
False
If user exists and true in dictionary
True
User response
Response recieved
No response recieved
Inform user of no response and following alert being sent

Lab entry process:



```python
from data import labuser as lu
import datetime
import asyncio
from data.globals import running_processes

async def command(client, message, *args):
    global running_processes
    id = message.author.id

    if len(args) < 3:
        await message.channel.send(f"{message.author.mention}\nNot enough arguments provided, please input your information to gain access to the lab.")
    elif(id in running_processes):
        await message.channel.send(f"{message.author.mention}\nYou are already signed into the lab")
    else:
        newlabuser = lu.LabUser(f"{args[0]} {args[1]}", args[2], (datetime.datetime.now()))
        running_processes[id] = True
        print(f"Running processes after set to True: {running_processes}")
        await message.channel.send(f"{message.author.mention}\nYou are signed into the lab.")
        await asyncio.create_task(whileinlab(client, message, newlabuser))
```

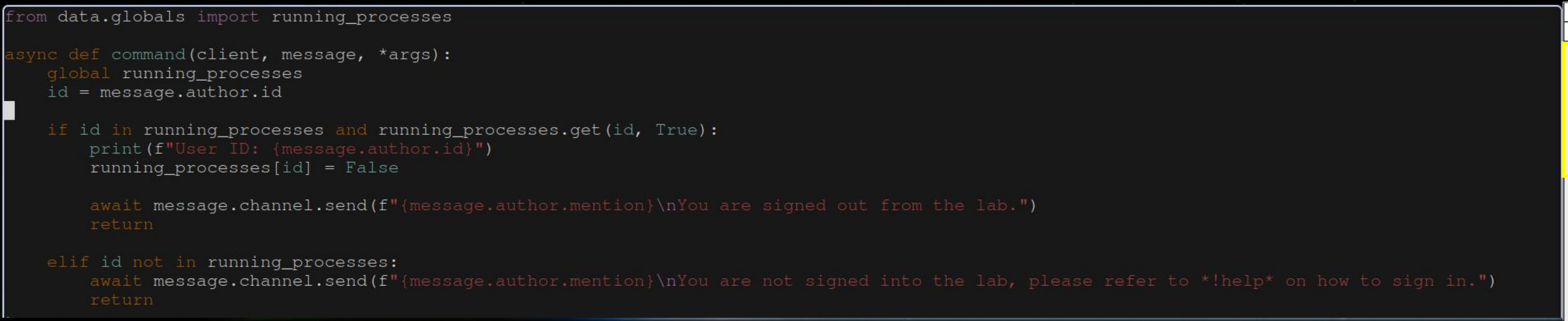
```python
async def whileinlab(client, message, newlabuser):
    global running_processes
    id = message.author.id
    dm_channel = await message.author.create_dm()

    await dm_channel.send("You are now signed into the lab, please remember to verify your safety every 30 minutes. \nEnsure you run *!labexit* in the server
when leaving.")
    while running_processes[id] == True:
        await asyncio.sleep(1800)
        if running_processes[id] == True:
            print(f"Running processes while running: {running_processes}")
            await dm_channel.send("Please verify your safety by replying to this message. \nFailure to do so in the next 10 minutes will result in an alert to
 check up on you.")

            try:
                await client.wait_for('message', check=lambda m: m.author.id == id, timeout=600)
                await dm_channel.send("Safety verification complete.")
            except asyncio.TimeoutError:
                await dm_channel.send("No response received. Sending an alert for someone to check up on you.")
                await message.channel.send(f"@everyone \n**{newlabuser.name}** with Student ID: **{newlabuser.studentid}** signed into the lab at: **{newlabus
er.labentrydatetime}**. \nHas failed to respond to a verification check 10 minutes ago, therefore a checkup may be necessary to ensure their well-being.")


    running_processes.pop(id, None)
    print(f"Running processes after loop ends: {running_processes}")
                                                                                                                                          5,0-1         Top
```


LASERLabSafetyService
User
Bot
While user exists and true in dictionary
True
Check-in Timer (30 minutes)
After 30 minutes
Check-in message
False
False
If user exists and true in dictionary
True
User response
Response recieved
No response recieved
Inform user of no response and following alert being sent
Send alert that a check-up is required
Remove user from dictionary

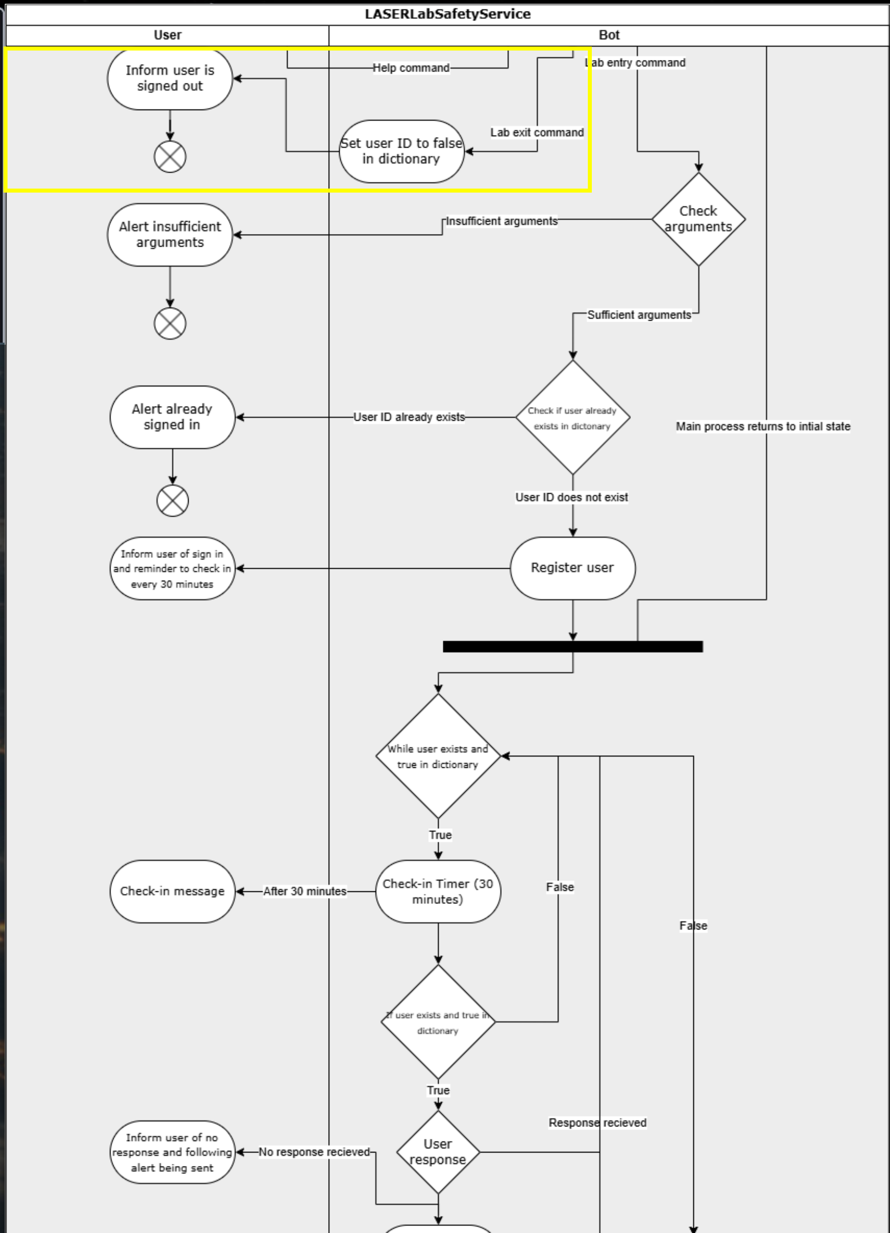
Concurrent task per user, that enables real-time lab monitoring per user. Non-blocking process keeps the main event loop free to handle other commands.

- Sends the user a direct message every 30 minutes with a 10-minute grace-period, requiring a response to ensure the user is still safe. If no response is received, then other members and university staff are alerted.
- Messages can be sent via direct message or if the user is active at the time in LASER message channels.

Lab exit process:



```
from data.globals import running_processes

async def command(client, message, *args):
    global running_processes
    id = message.author.id

    if id in running_processes and running_processes.get(id, True):
        print(f"User ID: {message.author.id}")
        running_processes[id] = False

        await message.channel.send(f"{message.author.mention}\nYou are signed out from the lab.")
        return

    elif id not in running_processes:
        await message.channel.send(f"{message.author.mention}\nYou are not signed into the lab, please refer to *!help* on how to sign in.")
        return
```



LASERLabSafetyService
User
Bot
Inform user is signed out
Help command
Lab entry command
Set user ID to false in dictionary
Lab exit command
Alert insufficient arguments
Insufficient arguments
Check arguments
Sufficient arguments
Alert already signed in
User ID already exists
Check if user already exists in dictonary
Main process returns to intial state
User ID does not exist
Inform user of sign in and reminder to check in every 30 minutes
Register user
While user exists and true in dictionary
True
Check-in message
After 30 minutes
Check-in Timer (30 minutes)
False
False
If user exists and true in dictionary
True
Response recieved
Inform user of no response and following alert being sent
No response recieved
User response

Handles the ***!labexit*** command, allowing users to sign out of the system and gracefully ends the associated non-blocking task.

- Verifies if the user is currently logged in, ***id in running_processes***.
- Sets associated id Boolean in dictionary to false, ending their monitoring and logging them out.
- Non-blocking task handles the false Boolean flag and terminates, regardless of current state of the concurrent task.
- Ensures clean termination and prevents unnecessary background processes.

Deployment:

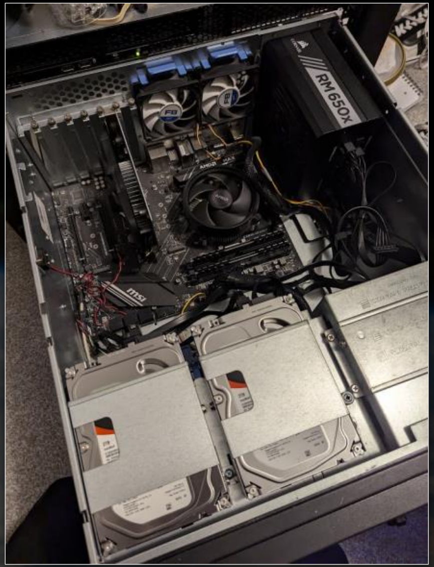
The LASER Safety Compliance Service is deployed on my home-lab temporarily due to sorting out issues with having on-site hardware connected to the University network.

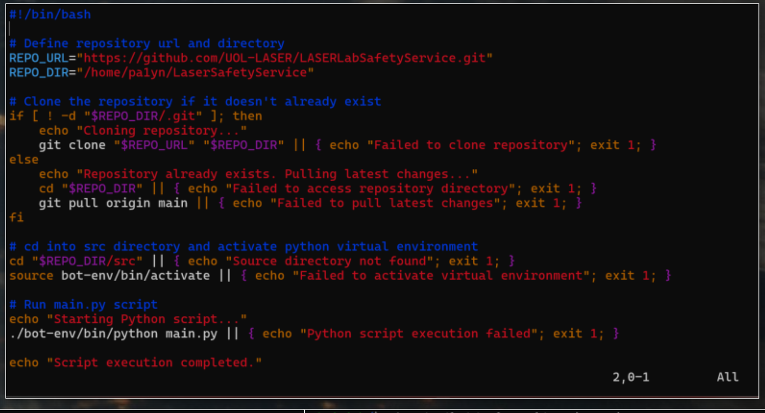
- The server running Debian Linux is accessible from anywhere, anytime over ssh. Utilising ***Systemd*** to ensure minimal downtime.

- ***Systemd*** service file references both the associated environment file and shell script.

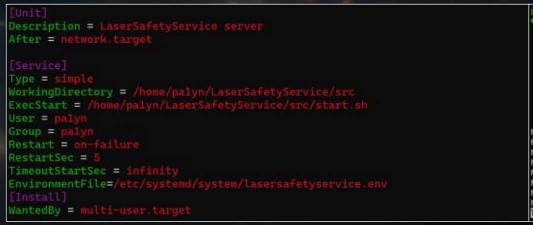
- Environment file contains the API secret to keep it secure and prevent it from being possibly uploaded via git.

- The shell script ***start.sh*** runs pre-execution of the service to ensure the local version is updated with the git repository, before launching a python virtual environment and executing ***main.py***.

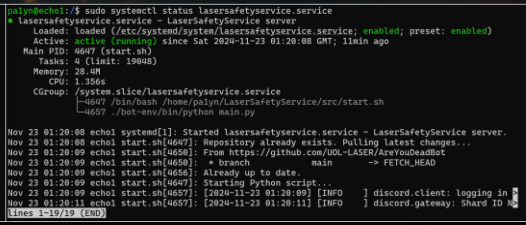




```
#!/bin/bash

# Define repository url and directory
REPO_URL="https://github.com/UOL-LASER/LASERLabSafetyService.git"
REPO_DIR="/home/palyn/LaserSafetyService"

# Clone the repository if it doesn't already exist
if [ ! -d "$REPO_DIR/.git" ]; then
    echo "Cloning repository..."
    git clone "$REPO_URL" "$REPO_DIR" || { echo "Failed to clone repository"; exit 1; }
else
    echo "Repository already exists. Pulling latest changes..."
    cd "$REPO_DIR" || { echo "Failed to access repository directory"; exit 1; }
    git pull origin main || { echo "Failed to pull latest changes"; exit 1; }
fi

# cd into src directory and activate python virtual environment
cd "$REPO_DIR/src" || { echo "Source directory not found"; exit 1; }
source bot-env/bin/activate || { echo "Failed to activate virtual environment"; exit 1; }

# Run main.py script
echo "Starting Python script..."
./bot-env/bin/python main.py || { echo "Python script execution failed"; exit 1; }

echo "Script execution completed."
                                                                      2,0-1         All
```



```
[Unit]
Description = LaserSafetyService server
After = network.target

[Service]
Type = simple
WorkingDirectory = /home/palyn/LaserSafetyService/src
ExecStart = /home/palyn/LaserSafetyService/src/start.sh
User = palyn
Group = palyn
Restart = on-failure
RestartSec = 5
TimeoutStartSec = infinity
EnvironmentFile=/etc/systemd/system/lasersafetyservice.env
[Install]
WantedBy = multi-user.target
```



```
palyn@echo1:/$ sudo systemctl status lasersafetyservice.service
● lasersafetyservice.service - LaserSafetyService server
     Loaded: loaded (/etc/systemd/system/lasersafetyservice.service; enabled; preset: enabled)
     Active: active (running) since Sat 2024-11-23 01:20:08 GMT; 11min ago
   Main PID: 4647 (start.sh)
      Tasks: 4 (limit: 19048)
     Memory: 28.4M
        CPU: 1.356s
     CGroup: /system.slice/lasersafetyservice.service
             ├─4647 /bin/bash /home/palyn/LaserSafetyService/src/start.sh
             └─4657 ./bot-env/bin/python main.py

Nov 23 01:20:08 echo1 systemd[1]: Started lasersafetyservice.service - LaserSafetyService server.
Nov 23 01:20:08 echo1 start.sh[4647]: Repository already exists. Pulling latest changes...
Nov 23 01:20:09 echo1 start.sh[4650]: From https://github.com/UOL-LASER/AreYouDeadBot
Nov 23 01:20:09 echo1 start.sh[4650]:  * branch            main       -> FETCH_HEAD
Nov 23 01:20:09 echo1 start.sh[4656]: Already up to date.
Nov 23 01:20:09 echo1 start.sh[4647]: Starting Python script...
Nov 23 01:20:09 echo1 start.sh[4657]: [2024-11-23 01:20:09] [INFO    ] discord.client: logging in >
Nov 23 01:20:11 echo1 start.sh[4657]: [2024-11-23 01:20:11] [INFO    ] discord.gateway: Shard ID N>
lines 1-19/19 (END)
```

Synopsis:

- This project has further developed both my technical and soft skills such as ***Communication*** and ***teamwork***. From utilising ***UML*** diagrams and other annotation to convey the project with non-technically aware members of committee during the design and implementation phases to gather appropriate feedback.

- Designing the system with a focus on modularity has worked out for my benefit during both maintenance and planning a rewrite for this academic year, due to changing requirements from the department in relation to health and safety.

- Rapid turn-around time for this project was a good demonstration of my time management and problem-solving skills.

- The project was successful at meeting the requirements set out by the client and was developed in such a way, that ensured robustness and reliability.

- Some aspects of the project were neglected in favour of a rapid solution being developed, with documentation being somewhat sparse and some QOL/ease of use features were left out. Such as integration with new systems built into Discord such as slash commands and hints.
    - For future projects and especially with the two projects I have running in parallel development, I am working towards ensuring full and easy to understand documentation. Especially to aid the handover of my projects in a few years once I finish my degree.

- If hypothetically, I was able to go back and change anything with or relating to the project, what would I do differently?
    - During the testing stage of the rapid SDLC, I would have prioritised usage of more unit and integration tests rather than manual testing. Although most issues were caught before production, some users encountered minor edge-case bugs that could have been addressed in pre-production.
    - In future projects, my aim will be to ensure more thorough testing and evaluation before production, such as usage of ***pytest*** or other libraries.

LASER Safety Compliance Service:

Thank you for listening!

I'd be happy to answer questions.