TAH – GUI Software - Application

Jaiwanth Karthi M K – ED24B051

General Questionnaire

1. India would necessarily need hyperloop in the future, primarily to transport its growing population efficiently, thus reducing congestion and pollution from combustion. Overcrowding in public transport may also have safety risks and such a solution for rapidity would minimise it by reducing waiting times for people. The developing Indian economy and cheapness of existing transportation may pose challenges to fund hyperloop's infrastructure and safety standards and attract public investment, although such a feat can connect rural and urban areas to foster growth for the whole nation. The concept is thus feasible if supported by investors with a futuristic vision and the government.
2. I have always wanted to be part of a bigger mission with who share my personality and likings to software. I believe Team Avishkar would be such a family to me, while developing a revolutionary vision that could change the Indian landscape for the better. I believe that my coding experiences (mostly Python and C++) will help me bring new insights to the table, as do my organisational skills. I hope to hone my networking and my development experience in this association.
3. If I work in the GUI and communications side of the Electrical vertical I would like the development of GUI and integration it with the pod, and then dip a hand in other parts of electrical to explore my interests and see which one align to me. Apart from the general exploration aspect (designing PCBs sound very exciting and interesting to me), from the GUI aspect I would specifically like to work on developing AI integration with the errors and alerts to make it possible to automatically resolve these issues.
4. **Delft Hyperloop** have a *Linear Flux Switching Permanent Magnet Motor* (LFSPMM) where permanent magnets is placed perpendicular to the coils as a Halbach array to nearly have field on one side only. Instead of needing more power from continuously generating fields, the coil is merely there to 'steer' the electromagnetic field, increasing the efficiency of the motor. **Mu-**

Zero Hyperloop innovated a system that *switches between magnetic levitation and air bearings* (thin cushions of pressurized air) for different operational needs and safety. On the Electrical aspect of innovations, **ITU Hyperbee** came up with *regenerative braking* to recover the kinetic energy lost for later use, whereas **Vegapod Hyperloop** leveraged *artificial intelligence to monitor and optimize energy consumption* across all pod systems.

5. Till the beginning of March, I will be a Deputy Coordinator (DC) in CFI's AI and Programming Club. I can dedicate nearly 7-8 hours a week, although this depends on requirements.
6. I am an intermediate in TinkerCAD, and beginning the journey of AutoCAD and Fusion. A link to one of my older TinkerCAD project: https://www.tinkercad.com/things/iVQs6R1Plw3-4x4-drive-with-servo-buzzer-and-lights. Nevertheless, I have been coding in Python for about 5 years, and in C/C++ for a year, and have tinkered with some Arduino modules as a beginner.

Question 1

Link to Github Repository of the Dashboard:
https://github.com/CoderAlpha9/GUI_Software_Submission_2025_ED24B051/tree/main

References: Streamlit documentation, ChatGPT

Question 2

1. Researching MQTT
    - The **MQTT** (Message Queueing Telemetry Transport) protocol is a lightweight communications protocol for IoT/machine-to-machine data transmission. It uses a "publish-subscribe" model where clients (connected to a broker) can either publish information on a specific topic or subscribe to certain topics and receive it. It provides three Quality-of-Service (QoS) levels to set the level of integrity need by the system.

- The **Publisher** is a device which acts as the source of information in the protocol. It outsources (publishing) this with associated 'topics' (category of information) to the **Broker,** which is the device utilizing decoupling (separating publishers and subscribers) and acts as a reservoir for the information and associated topic. **Subscriber** devices can then connect to the broker for certain topics (subscription) to receive the related information.
- MQTT ensures lightweighted and efficient communication by primarily decoupling publishers and subscribers and using compact message headers in packets to reduce network overload. Persistent connections and "subscriptions" reduce repeated handshakes and irrelevant information being transmitted. reduces overhead. QoS levels also account for optimised resource usage.
- References: https://cedalo.com/blog/complete-mqtt-protocol-guide/, MQTT Essentials - All Core Concepts Explained, MQTT - Wikipedia

2. Short Notes
    - MQTT for real-time Hyperloop monitoring is a suitable match, and can be achieved by first recognising sensors in the pod as publishers, outsourcing under topics such as 'pod/pressure'. A broker is then involved, followed by the control stations being subscribers to these topics. Sensors will continuously collect and publish data in real-time to ensure the most recent and relevant data is delivered to the control stations by the broker, to analyse risks and safety and compute mitigation measures.

      Any interruptions in communication in the middle will trigger the broker to resend the messages to ensure no information is lost. Real-time alerts can be flagged with QoS level-2 to ensure reliable transmission and handling, whereas regular location/status updates can be dealt with QoS level-1. Other redundant features can be from QoS level-0 to ensure optimum network usage.

      The lightweight MQTT is also suitable for connecting to a pod travelling at very high speeds, which is bound to have unreliable network connections and thus retransmission and reduced network overheads appear as boons in this scenario to achieve seamless, reliable, and low-latency communication between the pod and

monitoring infrastructure. Segregation of topics and access with wildcards ('+' and '*') is further advantageous in view of data handling.

- MQTT is primarily used for IoT applications, being lightweight and consuming lesser power, and doesn't incorporate security features on its own (although it's flexible enough to externally adopt TLS – Transport Layer Security) while a variant of HTTP (HTTPS) does incorporate TLS and authorisation certificates to maintain security over the internet, without emphasis for being lightweight as it has far more communication and security features, along with a more complex data header. This mean MQTT is faster than HTTP, while the latter is more robust.

  Moreover, MQTT adopts a broker-based publish/subscribe model while HTTP has request/response model. This implies the former has decoupling whereas in the latter client and server are coupled, and that MQTT focuses more on regulating data flow over unstable networks with its QoS levels, whereas HTTP relies a bit more on stability of networks (although it has retry-mechanisms, it does not have complete control over this in the TCP – Transmission Control Protocol layer).

  Furthermore, the above features imply that while MQTT is suitable for real-time, low-latency application. Lastly, HTTP does not maintain a persistent connection, requiring a new connection for each request-response pair, which can again be inefficient in unstable networks, whereas MQTT does, aiding in Hyperloop connections where speed can induce network fluctuations.
- References: HTTP vs MQTT: Choose the best one for an IoT project | Cedalo, MQTT vs HTTP: Ultimate IoT Protocol Comparison Guide | EMQ, ChatGPT

3. Exploration
    - Eclipse Mosquitto is an open-source, lightweight MQTT broker designed to facilitate communication between MQTT clients in IoT and real-time data exchange scenarios. Focused on core MQTT functionalities, it lacks advanced/enterprise features.

- HiveMQ MQTT Essentials is a popular series of educational resources aimed at helping developers, businesses, and IoT professionals understand the MQTT protocol and its applications. It provides foundational knowledge about MQTT, covering its architecture, features, and best practices for implementation in IoT and real-time communication scenarios.
- References: Eclipse Mosquitto, What is Mosquitto MQTT?, HiveMQ – The Most Trusted MQTT platform to Transform Your Business, HiveMQ vs. Mosquitto: An MQTT Broker Comparison

4. MQTT setup with paho-mqtt in Python

----------publisher.py----------

```python
import paho.mqtt.client as mqtt
import time
import json
import random

#MQTT Broker details
BROKER = "test.mosquitto.org"  #Broker URL
PORT = 1883  #Default MQTT port
TOPIC = "hyperloop/pod_data"

#Other constants
PUBLISH_INTERVAL = 2  #seconds

battery = 100 #counter for battery level
#Simulated Pod data
def generate_pod_data():
    battery -= 1
    n = random.randint(0, 2) #randomly select a pod
    return {"pod_name": ["Avishkar-1", "Avishkar-2", "Avishkar-3"][n],
            "speed_kmph": [random.randint(100, 1200), 0, 0][n],
            "battery_level": battery,
            "status": ["Operational", "Maintenance", "Docked"][n]}

#Publisher code
client = mqtt.Client("Publisher", protocol=mqtt.MQTTv311)
client.connect(BROKER, PORT)

print("Publisher connected to broker")

#Publish data every <PUBLISH_INTERVAL> seconds
try:
    while True:
        pod_data = generate_pod_data()
        client.publish(TOPIC, json.dumps(pod_data))
        print(f"Published: {pod_data}")
        time.sleep(PUBLISH_INTERVAL)
except KeyboardInterrupt:
    print("Publisher stopped.")
    client.disconnect()

```

----------subscriber.py----------

```python
import paho.mqtt.client as mqtt

#MQTT Broker details
BROKER = "test.mosquitto.org"  #Broker URL
PORT = 1883  #Default MQTT port
TOPIC = "hyperloop/pod_data"

#Function to handle received messages
def on_message(client, userdata, msg):
    print(f"Received: {msg.payload.decode()}")

#Subscriber code
client = mqtt.Client("Subscriber", protocol=mqtt.MQTTv311)
client.connect(BROKER, PORT)

client.subscribe(TOPIC)
client.on_message = on_message

print("Subscriber connected to broker and waiting for messages...")
try:
    client.loop_forever()  #Keep listening for messages
except KeyboardInterrupt:
    print("Subscriber stopped.")
    client.disconnect()

```

- To run the above files, first run “python publisher.py” in one terminal and “python subscriber.py” in another terminal, with Mosquitto as broker service.

References: Paho MQTT Python client: a tutorial with examples | Cedalo blog, paho-mqtt · PyPI, ChatGPT