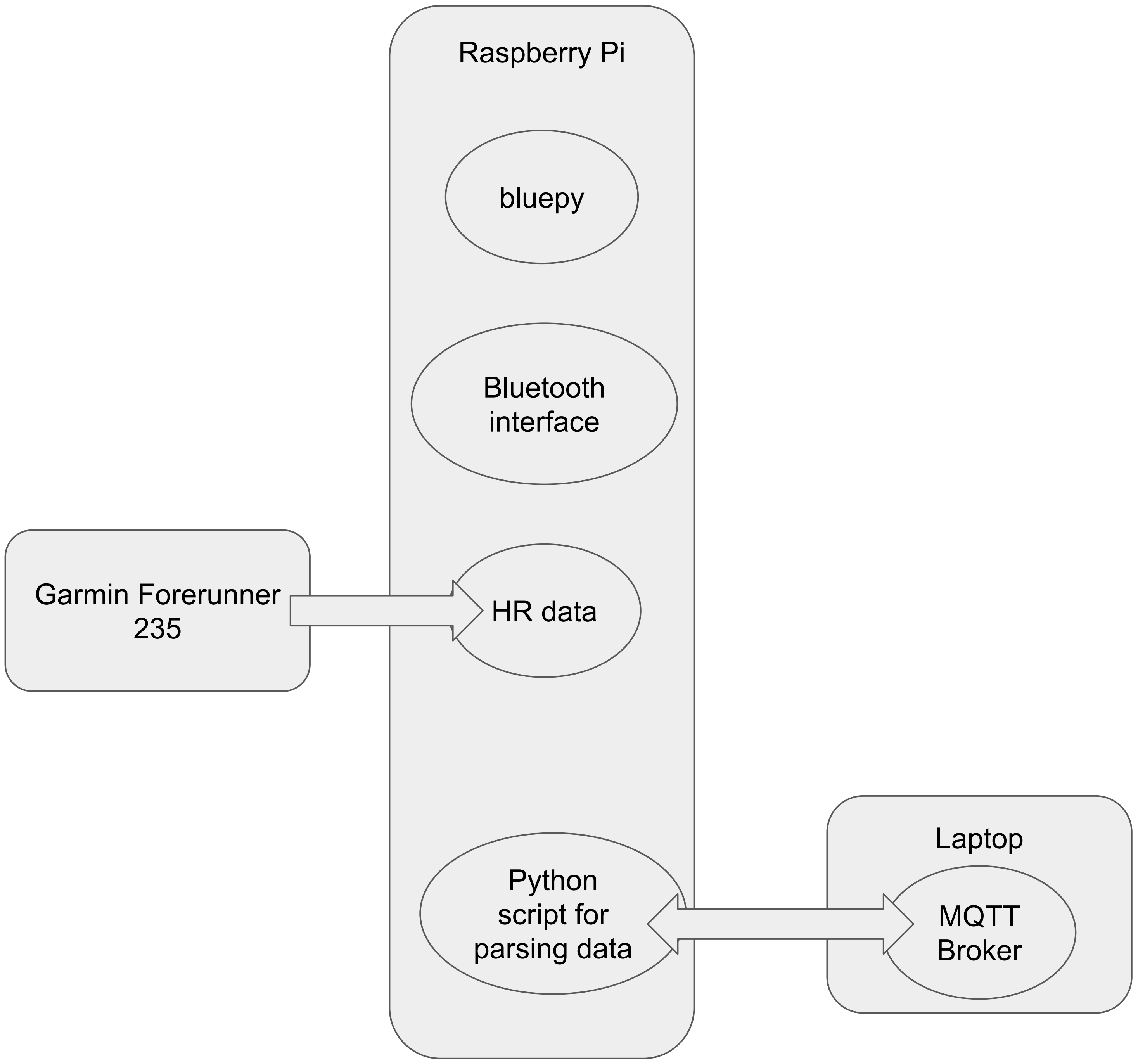
Team members: Dylan Gatua

A short description of your project and its functionality:
This project involves connecting a python script run on a Raspberry Pi to a Garmin Forerunner 235 watch using Bluetooth Low Energy (BLE) and receives heart rate data. It then extracts features and detects events such as exercise based on the received data. The code uses the bluepy library in Python to interface with the Bluetooth stack on the Raspberry Pi. The extracted features include average heart rate, maximum heart rate, and standard deviation of heart rate over a window of time. The code publishes this extracted data to a MQTT broker, which can be accessed by other devices or applications. The code also prints the extracted features to the console.



Briefly describes components, protocols used, key processing techniques, etc.
along with important implementation/design choices.:

1. Libraries and modules: The code imports the following libraries and modules:
   - numpy: a library for scientific computing with Python. It is used to compute the average, maximum, and standard deviation of the heart rate data.
   - bluepy.btle: a Python interface for Bluetooth Low Energy (BLE) devices. It is used to connect to the Garmin Forerunner 235 watch and receive heart rate data.
   - paho.mqtt.client: a Python client library for the MQTT protocol. It is used to connect to an MQTT broker and publish heart rate data.
2. MQTT client: The code sets up an MQTT client using the paho.mqtt.client module. The client connects to an MQTT broker at IP address "10.25.255.255" and port 1883 with a keepalive time of 60 seconds. It subscribes to the topic "heart_rate" and starts a loop to listen for incoming messages.
3. UUIDs for heart rate measurement: The code defines two UUIDs for the heart rate service and heart rate measurement characteristic of the Garmin Forerunner 235 watch.
4. Connect to the watch and discover services and characteristics: The code uses the bluepy.btle module to connect to the Garmin Forerunner 235 watch and discover the heart rate service and heart rate measurement characteristic.
5. Enable notifications for heart rate measurements: The code writes a value to the Client Characteristic Configuration Descriptor (CCCD) of the heart rate measurement characteristic to enable notifications for heart rate measurements.
6. Feature extraction and event detection: The code defines variables for feature extraction and event detection, including a heart rate buffer, a window size of 10 seconds, and an event threshold of 120 bpm. It continuously receives and parses heart rate data, computes features (average, maximum, and standard deviation), and detects events based on the maximum heart rate exceeding the event threshold.
7. Publish heart rate data to MQTT broker: The code formats the heart rate data as a JSON message and publishes it to the MQTT broker under the topic "heart_rate".
8. Error handling: The code uses a try-except-finally block to handle errors that may occur during execution. If an error occurs, it prints an error message to the console and disconnects from the watch and MQTT broker.
9. Implementation/design choices:
   - The code uses the bluepy.btle module to connect to the watch and receive heart rate data. This module provides a low-level interface to the BLE protocol, which may be more efficient and flexible than higher-level modules like pygatt or bleak.
   - The code uses the paho.mqtt.client module to publish heart rate data to an MQTT broker. This module is a popular and well-documented implementation of the MQTT protocol for Python.
   - The code defines a window size of 10 seconds for computing heart rate features. This window size may be too small or too large depending on the application and the expected variability in heart rate data.
   - The code uses a single topic ("heart_rate") to publish heart rate data to the MQTT broker. In a more complex system, multiple topics may be used to organize different types of data or to route data to different destinations.

Reflection on limitations and their causes as well as lessons learned:
One of the limitations of the code is that it assumes the device it is connected to always functions correctly and provides accurate data. In reality, devices may have hardware or software failures that could cause the data to be inaccurate, incomplete, or inconsistent. Additionally, the code does not have a built-in error handling mechanism to handle such situations, which could result in unexpected behavior or even program crashes.
Another limitation of the code is that it uses blocking calls for Bluetooth Low Energy (BLE) notifications. This means that the program will pause and wait for notifications to arrive, which could delay the execution of other tasks or cause the program to become unresponsive if there is a long delay in receiving the notification. Using non-blocking calls or a separate thread to handle notifications would help mitigate this issue.
Finally, the code uses MQTT for publishing heart rate data, but it does not implement any security measures, such as encryption or authentication. This could expose the data to potential attackers who could intercept, modify, or steal the data. Implementing security measures, such as Transport Layer Security (TLS) encryption and user authentication, would help protect the data from unauthorized access.