Data server specification for the HEV

First implementation of dataserver for the raspberry pi with broadcast and query capabilities.

TCP server implemented using the asyncio python library to serve data on two unix sockets and listen for requests on a third. Requests and broadcasts are dealt with on separate threads.

Socket addresses:
127.0.0.1:54320 - WebUI broadcast socket
127.0.0.1:54321 - Joint query socket
127.0.0.1:54322 - NativeUI broadcast socket

- Clients connect once to their broadcast socket and keep the connection open while polling constantly for the next packet.
- Clients connect to the joint query socket, send one query, await the reply, then disconnect to free up the socket.

Broadcast sockets
A broadcast is sent out to all connections every n seconds where n is configurable

Alarms are included in the broadcast packet but can also come through as a dedicated alarm packet if they come through alone, depending on the LLI implementation (TBC)

The packet is a json frame with format:
{
    "type": str,
    "sensors": List[float],
    "alarms": List[str],
    "settings": List[float]
}

- “type” refers to a string with maximum length 9 char
- “sensors” refers to a list of length 6, with each element containing a 32b float
- "alarms" refers to a list of strings of length varying between 1 and 6 inclusive, with each element containing a string of max length 21 char (easier to just give an int as below?)
  - 0: "none",
  - 1: "manual",
  - 2: "gas supply",
  - 3: "apnea",
  - 4: "expired minute volume",
  - 5: "upper pressure limit",
  - 6: "power failure"
Example broadcast packet:
{
    "type": "broadcast",
    "sensors": [862.359688536043, 376.8359070273678, 545.0542884669316, 989.5016042203939, 449.15263765793776, 152.43765749693776],
    "alarms": ['manual']
    "settings": [... ]
}

Example alarm packet:
{
    "Type": "alarm",
    "Alarms": ['apnea']
}

Request socket
The HEVServer class has a method to set a different mode of operation, this is done through the shared socket to avoid multiple threads writing to the same variables.

TODO return acknowledge to BOTH via broadcast packet, so they are both aware of the actual mode set in case of both trying to set at the same time.

Uplink packets
Query packet format:
{
    "type": str,
    "mode": str
    "settings": List[float]
}

Example query to change the mode to CPAP:
{
    "type": "setmode",
    "mode": "CPAP",
}

Example query to change the threshold settings:
{
    "type": "setthresholds",
    "settings": [...]}
Example query to set the mode and thresholds in one packet:

```
{
    "type": "setup",
    "mode": "CPAP",
    "settings": [...]
}
```

**Downlink packets**

Reply format:

```
{
    "type": str,
    "mode": str,
    "settings": List[Float]
}
```

Example acknowledge packet for change of mode:

```
{
    "type": "ackmode",
    "mode": "CPAP",
}
```

Example acknowledge packet for change of settings:

```
{
    "type": "ackset",
    "settings": [...]
}
```
Example hevclient.py Usage

import hevclient

# create instance of hevclient. Starts connection and polls in the background
hevclient = HEVClient()

# Play with sensor values and alarms, set mode
for i in range(10):
    print(f"Sensor values: {hevclient.get_values()}")
    print(f"Alarms: {hevclient.get_alarms()}")
    time.sleep(1)
    print(hevclient.set_mode("CPAP"))
    print(hevclient.set_thresholds([12.3, 45.6, 78.9]))