See Deliverables expected for each activity.

Section 2: Assignment Specification

IoT System for e-Bike Fleet Monitoring e-Bikes have been rolled over in multiple cities around the world. For this assignment, assume you work for **Eco-Wheels**, a UK-based e-bike hire company operating across multiple cities, including Bristol city. **Eco-wheels** E-bikes are equipped with an on-board computer with Wi-Fi connectivity, a GPS device, and an accelerometer.

Your task is to contribute to the development of the IoT system to track e-bike locations and monitor their status. This system will enhance the visibility of the e-bike fleet, improve availability, and increase maintenance efficiency. You will focus on developing specific components of this IoT system that are detailed in each worksheet activity. You will use a provided Hardware Abstraction Layer (HAL) to simulate physical interfaces. Every HAL can represent a different on-board computer and bike.

Worksheet 2: Activity 1: Displaying location of eBikes in a map (up to 40% of worksheet 2 marks) In this activity you need enable your simulated GPS device program (eBikeClient) developed in the graded worksheet 1, to send its location, coming from the GPS readings, to a Gateway. The eBikeGateway will act as a server that will collect the readings and will present these in map with the help of Web server that is provided. For communication between eBikeClient and eBikeGateway you must use the UDP sockets over the emulated network as practised in the week 5 worksheet. Before sending the data from the client you need to format it in a way that will be easy to interpret at the other end. For that reason, you will use **JSON format**. At the server you can easily **parse** the data back to a JSON object using POCO library, see **a simple example here**. Before you can present the GPS location, you need to create a GeoJSON object and store it in a data structure, details below in each file.

There are two parts in this activity, the first part is pre-requisite for the second part:

1) (Up to 20% marks) Enabling eBike location transmission via UDP sockets for a single eBike with socket server running continuously.
2) (Up to 40% of marks) Enabling eBike location transmission via UDP sockets for multiple eBikes, with server running continuously, acknowledgment sent by server and logged by eBike client.

Before building your project, the structure should look like this:

```
├─── LICENSE
├───Makefile                    [Toreusefromgradedworksheet1]
├─── build
├─── data
│    └───sim-eBike-1.csv        [GeneratedbygenerateEBikeFile
│    └───sim-eBike-             ]
│   2.csv        └───sim-       [GeneratedbygenerateEBikeFile
│   eBike-3.csv                 ]
├─── src└───sim-eBike-          [GeneratedbygenerateEBikeFile
│   4.csv───ebikeClient.cpp     ][Toreusefromgradedworksheet1]
│    ├───ebikeGateway.c         [GeneratedbygenerateEBikeFile[*Toupdatebasedonworksheetweek5*]
│   pp                          ][*Tocreatebasedonworksheetweek5*]
│    ├───MessageHandle          [Toreusefromgradedworksheet1]
│   r.h  ├───GPSSensor.h        [*Tocreatebasedonworksheetweek5*]
│  ├├───hal SocketServer.h
│  │  ├───CSVHALManager.h       [Toreusefromgradedworksheet1]
│    │    ├───IActuator.h       [Toreusefromgradedworksheet1]
│    │    ├───IDevice.h         [Toreusefromgradedworksheet1]
│    │    └───ISensor.h         [Toreusefromgradedworksheet1]
│  ├───html
│    │    ├───map.html          [Giveningradedworksheet2]
│  ├───sim
│    │    ├───in.h              [Giveninworksheetweek5]
│    │    ├───socket.h          [Giveninworksheetweek5]
│  ├───util
│  │  ├───generateEBikeFile.cpp [Giveningradedworksheet1]
│  └───web
│         └───EbikeHandler.h    [Giveningradedworksheet2]
│         └───WebServer.h       [Giveningradedworksheet2]
└─── tests
```

Files in the src/hal,src/util and src/sim folders together with src/ebikeClient.cpp, src/GPSSensor.h and src/MessageHandler.h have been provided in previous worksheets. Files in src/web, src/html, a skeleton of src/eBikeGateway.cpp and a basic Makefile are provided in this worksheet repository. The src/util/generateEBikeFile.cpp utility program provided is prepared to generate multiple sim-eBike data files, operation is similar to that in worksheet 1. For this activity you need to **generate at least 4 files, each one with 10 records**.

In this activity you need to develop or update the following files:

- Makefile : Configuration file to build your entire project, it should enable generation of executable files for ebikeGateway, ebikeClient and generateEBikeFile.
- src/ebikeGateway.cpp : Main server-side application that creates and start a web server and a socket server. A shared eBikes list is used to push data from the socket server to the web server. Your web server must run in your own allocated port as per the table published here.. Both servers should run concurrently and continuously.
- src/SocketServer.h : Socket server that creates and binds the address of the socket server, it calls the MessageHandler.h once a message is received.
- src/MessageHandler.h : Class that includes the handleMessage method to parse the received data into a JSON object, converts it to a **GeoJSON object**. The GeoJSON is of type "Feature", geometry type: "Point", the ebike id, and timestamp, should be included as properties. Finally, push the GeoJSON object to the ebike list.
- src/ebikeClient.cpp: This file needs to be adjusted to set an ID for the eBike, pack the ID with the timestamp and sensor data and use the client-side socket to transmit it to the socket server.
- src/GPSSensor.h: This file needs to be adjusted so it formats the GPS data as a JSON string ready for transmission.

Ensure your code is fully documented (commented).

To run your server you need to open a Terminal and type:

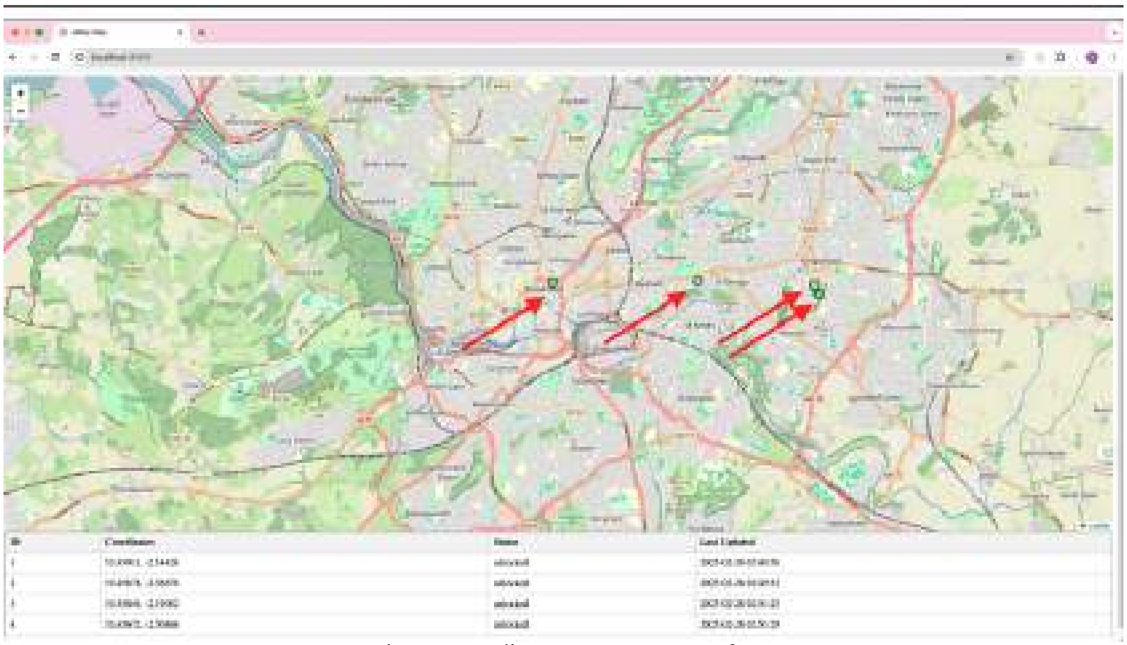
```
./ebikeGateway
```

The following logs appear:

```
Server started on http://localhost:<<port_number>>
Press Ctrl+C to stop the server...
Socket Server waiting for messages...
```

..and an emergent dialog box will appear on the right bottom, indicating that your port has been forwarded and you can open your server on the browser by clicking on "open in browser":

eBikes are presented in the map with a green circle. For the first activity status in the table is empty and the default color of the eBike markers in the map is green.



ID	Coordinates	Status	Last Updated
1	[illegible]	unlocked	[illegible]
[illegible]	[illegible]	unlocked	[illegible]
[illegible]	[illegible]	unlocked	[illegible]
4	[illegible]	unlocked	[illegible]

Figure 1: eBike IoT System Interface

Initially the map is presented empty, to enable eBikes to appear, you need to run the eBikeClient in a separate Terminal, as many times as eBikes you are simulating. The command to run an eBikeClient is now as follows:

```
./ebikeClient 192.168.1.10 1 data/sim-eBike-2.csv 1
```

where the first argument is emulated client IP and the second is the eBike ID. The remaining arguments are as specified in graded worksheet 1. When executing the client it is expected that some minimal logs will appear:

```
[CSVHALManager] Device attached to port 0.
[2025-02-12 11:26:34] GPS: 51.459079, -2.544360
Received response: OK
From: 192.168.1.1:8080
[2025-02-12 11:26:44] GPS: 51.459147, -2.544450

...
```

Activity 2: Maintenance Management of eBikes (up to 40% of worksheet 2 marks) Note: Please attempt this activity only once activity 1 is completed. In this activity, you need to enable management of eBikes when these require maintenance. The basic premise is that when an eBike requires maintenance it needs to be hard-locked, so it is not available for hiring until the maintenance is done. The maintenance can be triggered manually via a separate management client.

In this activity you are expected to develop your solution using what you have learned so far, with limited guidance provided.

This activity has two parts, the first part is pre-requisite of the second one:

1) (Up to 20% marks) The first part requires you to implement the following:

- The BASIC protocol defined **here** for enabling pre-registry of ebikes and data update based on protocol. eBikes and eBikeGateway operate as defined in the protocol.
- Displaying of eBike status: hard-locked (red color, status: “locked”) or no (green color, status: “unlocked”) in the web interface - map and table. This status can be implemented with a boolean flag in the eBikeClient and transmitted to the ebikeGateway.

2) (Up to 40% marks) The second part requires that you implement:

- The ENHANCED protocol defined **here** for enabling command-based operation of the ebikes. eBikes and eBikeGateway operate as defined in the protocol.
- A management client, for a human administrator, who can trigger maintenance-related commands manually and using the defined protocol, this client uses UDP sockets to send the commands to the eBikeGateway. Commands enable the administrator to change the status of the eBike, which should be reflected on the eBike and web interface.

Ensure your code is fully documented (commented).

Protocol Definition - BASIC

Directive	Purpose	From	To	Example Payload
JOIN	AneBikejointhefleetcontrolledby the IoT System.	eBikeClient	eBikeGateway	ebike_id: 123 timestamp: “2025-02-27T12:00:00Z”
JACK	Acknowledgementthatthe eBikeClient has been registered in the system. It sends back the configuration parameters for the ebike, particularly the interval for sending sensor reading. For this part, that interval can be hardcoded in the gateway.	eBikeGateway	eBikeClient	status: success data_interval: 5
DATA	AneBikesendsdatatothe eBikeGateway.	eBikeClient	eBikeGateway	ebike_id: 123 timestamp: “2025-02-27T12:00:00Z” gps: latitude: 51.5074 longitude: 0.1278 accelerometer: x: 5.0 y: -3.0 z: 0.8 lock_status: 0

Protocol Definition - ENHANCED

Directive	Purpose	From	To	Example Payload
COMMAND	Ahumanadministratortriggersan action to an eBike or a group of eBikes via the eBikeGateway. Possible actions include: lock (the eBike requires or is on maintenance) and unlock (eBike ready to use).	Management Client eBikeGateway	eBikeGateway eBikeClient	timestamp: “2025-02-27T12:00:00Z” action: “lock” ebike_ids: [123,456,789]
COMMACK	eBikeClientconfirmstoeBikeServer the action has been triggered correctly. eBikeGateway confirms to Management Client that action was triggered correctly.	eBikeClient eBikeGateway	eBikeGateway Management Client	ebike_id: 123 status: success
SETUP	Management Client configures parameters of interest, in this case, only the interval to send data by eBikes.	Management Client	eBikeGateway	timestamp: “2025-02-27T12:00:00Z” data_interval: 5