Assessment Brief

Module Details

Module Title	Internet of Things
Total number of assessments for this module	2
Year	2024-2025
Assessment Type	Worksheets
Weighting	75% of the module marks.

Dates

Submission Point

Date Issued to Students	Submission Date
worksheet 1: 11/Feb/2025	03/Apr/2025
worksheet 2: 04/Mar/2025	03/Apr/2025
Submission Channel	**Submission Time**
Electronic via GitHub Classroom	14:00
Feedback and Marks By	

Contents

Section 1: Overview

The assignment consists of 2 worksheets, which are themselves broken up into several activities. The activities collectively contribute to the development and functionality of an IoT system.

See **Assessment Specification** for more details.

Module learning outcomes assessed:

1. [MO1] Explain the principles and operation of the general Internet and the Internet of Things
2. [MO2] Demonstrate with examples, constraints and opportunities of wireless and mobile networks for Internet of Things
3. [MO3] Critically evaluate security issues within the domain of Internet of Things
4. [MO4] Compare the various network protocols used in IoT
5. [MO5] Describe the key wireless technologies used in IoT systems, such as WiFi, LoRaWAN, and Bluetooth LE
6. [MO6] Apply object-oriented approaches in C++, to the design of embedded systems with application to Internet of Things
7. [MO7] Develop and use test plans

Submission Rules

This assessed work must be submitted as follows:

- Commit and push the solution to YOUR OWN INDIVIDUAL GITHUB REPOSITORY THAT IS CREATED ONCE YOU ACCEPT THE GITHUB ASSIGNMENT. NO OTHER GITHUB REPOSITORIES WILL BE ACCEPTED.
- Make sure your GitHub username is linked to your student username and ID.
- Source code should be organised within the `src` folder.
- For all programming activities, the code should be documented thoroughly.
- Every file that you use in your solution should be copied in the repository. No links to partial solutions copied or uploaded to external repos or sites.

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                    [To reuse from graded worksheet 1]
├── build
├── data
│   └── sim-eBike-1.csv         [Generated by generateEBikeFile]
│   └── sim-eBike-2.csv         [Generated by generateEBikeFile]
│   └── sim-eBike-3.csv         [Generated by generateEBikeFile]
│   └── sim-eBike-4.csv         [Generated by generateEBikeFile]
├── src
│   ├── ebikeClient.cpp         [To reuse from graded worksheet 1]
│   ├── ebikeGateway.cpp        [*To update based on worksheet week 5*]
│   ├── MessageHandler.h        [*To create based on worksheet week 5*]
│   ├── GPSSensor.h             [To reuse from graded worksheet 1]
│   ├── SocketServer.h          [*To create based on worksheet week 5*]
│   ├── hal
│   │   ├── CSVHALManager.h     [To reuse from graded worksheet 1]
│   │   ├── IActuator.h         [To reuse from graded worksheet 1]
│   │   ├── IDevice.h           [To reuse from graded worksheet 1]
│   │   └── ISensor.h           [To reuse from graded worksheet 1]
│   ├── html
│   │   ├── map.html            [Given in graded worksheet 2]
│   ├── sim
│   │   ├── in.h                [Given in worksheet week 5]
│   │   ├── socket.h            [Given in worksheet week 5]
│   ├── util
│   │   ├── generateEBikeFile.cpp  [Given in graded worksheet 1]
│   └── web
│       └── EbikeHandler.h      [Given in graded worksheet 2]
│       └── WebServer.h         [Given in graded worksheet 2]
└── 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).

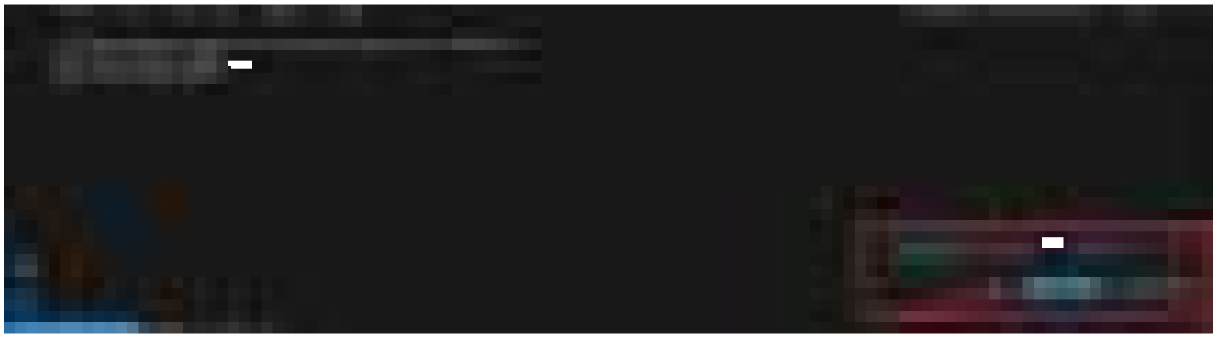
To run your server you need to open a Terminal and type:

```
./ebikeGateway
```

The following logs appear:

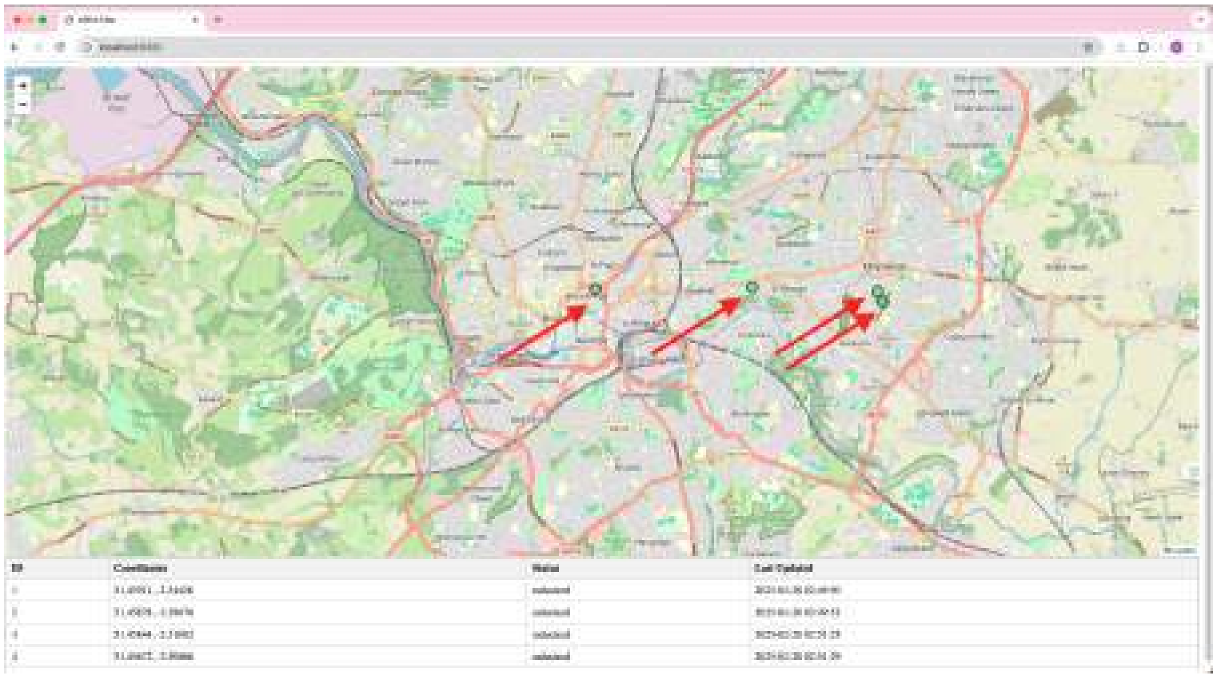
```
Server started on http://localhost:<<port_number>>
Press Ctrl+C to stop the server...
Socket Server waiting for messages...
```

..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":



Click there and the expected result is presented in the web browser, which is set to the URL: `http:\\localhost:<<port_number>>`.

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]
2	[illegible]	unlocked	[illegible]
3	[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	An eBike join the fleet controlled by the IoT System.	eBikeClient	eBikeGateway	ebike_id: 123 timestamp: "2025-02-27T12:00:00Z"
JACK	Acknowledgement that the 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	An eBike sends data to the 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	A human administrator triggers an 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	eBikeClient confirms to eBikeServer 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

Directive	Purpose	From	To	Example Payload
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

Activity 3: Process Documentation (up to 20% of worksheet 2 marks)

In Markdown file called `README.md`, document your development process with up to a maximum of 400 words (+/- 10%). You should cover the following aspects:

- What process (steps) you followed to develop your solution
- The main challenges you found during development and approach you followed to tackle them.
- The main learnings you achieve with this worksheet.

Your process documentation should be aligned to what you have done in Activity 1 and Activity 2 (if attempted).

Section 3: Deliverables

Activity	**Detail**	**Date**
1	C++ code, Makefile and CSV files of the project as per requirements.	3/Apr/2025
2	C++ code, Makefile and CSV files of the project as per requirements.	3/Apr/2025
3	*README.md* documenting your development process as requested.	3/Apr/2025

Section 4: Assessment Criteria

Worksheet 2

Assessment criteria	Percentage mark	0-39	40-49	50-59	60-69	70-100
Overall Descriptor eBike Location (40%)		Failed Code does not compile or run.	3rd Code compiles and runs with some warnings or not achieving output as expected.	2:2 Code runs correctly with minor issues. Comments/Documentation is minimal.	2:1 Code runs correctly with minor omissions. Comments/Documentation is acceptable but incomplete.	First Code runs flawlessly, well-structured, and efficient. Code is thoroughly commented/documented.
Maintenance Management (40%)		Code does not compile or run.	Code compiles and runs with some warnings or not achieving output as expected.	Code runs correctly with minor issues. Comments/Documentation is minimal.	Code runs correctly with minor omissions. Comments/Documentation is acceptable but incomplete.	Code runs flawlessly, well-structured, and efficient. Code is thoroughly commented/documented.
Process Documentation (20%)		No documentation or very poor.	Basic documentation, lacks detail.	Overall process described covering main aspects but with minimal details.	Reasonable process documentation, covers important aspects with some detail.	Thorough documentation, well-organised, and detailed with a solid and appropriate use of technical language.

Section 5: Feedback Mechanisms

- Informal feedback provided during development of the solution in the practical sessions.
- Formal feedback to be provided with marks after submission date.

Section 6: Appendices

Completing your assessment

Where should I start? Read carefully the specifications and identify key elements of the problem that should be considered in the solution. After that you can start outlining what are the elements to consider in your code. Use the code provided in the worksheets of the practical sessions as a reference point.

What do I need to do to pass? The minimum to pass is 40 marks. See the marking criteria at the end of this document.

How do I avoid an Assessment Offence on the m odule? This is an individual a ssignment. Even though you can reuse part of the code and material provided in the practical sessions, this assignment requires you to create and entire new files, code and document and solve the worksheet t asks. Naturally your approach and specific solution should be different from your cl assmates. Although you can discuss and collaborate with others during the practical sessions. The best way to stay safe is not sharing any part of your solution with others, this way the solution that you submit will be unique.

Assessment Offences Policy requires that you submit work that is entirely your own and reflects your own learning. It is important to:

- Ensure that you reference all sources used, using the Harvard system. Use the guidance available on Study Skills referencing pages.
- Avoid copying and pasting any work into this assessment, including your own previous assessments, work from other students or internet sources
- Develop your own style, arguments and wording. Avoid copying and changing individual words but keeping essentially the same sentences and/or structures from other sources
- Never give your work to others who may copy it
- If you are doing an individual assessment, develop your own work and preparation. Do not allow anyone to make amendments to your work (including proof-readers, who may highlight issues but not edit the work).

Use of Generative AI Tools

- You are allowed to use GenAI tools such as copilot, chatGPT and others if you want, although not necessary, nor recommended. Using these tools brings additional benefits but also risks, you MUST consider the following:
  - DO NOT feed these tools with any of the module materials (slides, assessments briefs,code provided, etc.) Doing so constitutes a breach to copyright and you could be subject to UWE penalties.
  - Make a thoughtful use of these tools, it is not enough to copy and paste outputs of these tools in your solution. Just copying and pasting **will increase the risk of somebody else getting the same code or text you got. ANY CONTENTS OF YOUR SOLUTION THAT IS FOUND SIMILAR TO SOMEONE ELSE WILL TRIGGER THE ACADEMIC OFFENCE PROCEDURE WITH A LIKELY PENALTY**.
  - Any use of GenAI tools needs to be properly acknowledged in the `README.md` file, you need to briefly indicate what tool(s) have you used and how.